University of Cyprus

DEPARTMENT OF COMPUTER SCIENCE

Conditional Generative Denoising Autoencoder

Savvas Karatsiolis

A Dissertation Submitted to the University of Cyprus in Partial Fulfillment of the Requirements for the Degree of Doctor of Philosophy

December 2019

VALIDATION PAGE

Doctoral Candidate: Savvas Karatsiolis

Doctoral Thesis Title: Conditional Generative Denoising Autoencoder

The present Doctoral Dissertation was submitted in partial fulfillment of the requirements for the Degree of Doctor of Philosophy at the ***Department of Computer Science*** *and was approved on the 25th of October 2019 by the members of the* ***Examination Committee***.

Examination Committee:

Research Supervisor: _____
Schizas N. Christos, Professor, Department of Computer Science, University of Cyprus

Committee Member: _____
Pattichis Constantinos, Professor, Department of Computer Science, University of Cyprus

Committee Member: _____
Christodoulou Christos, Professor, Department of Computer Science, University of Cyprus

Committee Member: _____
Stafylopatis Andreas, Professor, School of Electrical and Computer Engineering, National Technical University of Athens

Committee Member: _____
Papadopoulos Harris, Assistant Professor, Department of Computer Science and Engineering, Frederick University of Cyprus

DECLARATION OF DOCTORAL CANDIDATE

The present doctoral dissertation was submitted in partial fulfillment of the requirements for the degree of Doctor of Philosophy of the University of Cyprus. It is a product of original work of my own, unless otherwise mentioned through references, notes, or any other statements.

Savvas Karatsiolis

ACKNOWLEDGEMENTS

First and foremost, I would like to thank my supervisor Dr. Schizas N. Christos for his support, advice, enthusiasm and for giving me the freedom to explore my topics of interest. He has been helpful and supportive throughout my PhD studies and his guidance has been particularly valuable.

I would also like to thank the dissertation committee members, Professors Chris Christodoulou, Constantinos S. Pattichis, Andreas Stafylopatis and Harris Papadopoulos for offering their advice and providing constructive suggestions for the improvement of this thesis.

I am deeply indebted and thankful to my beloved family, especially to my wife Theognosia for her encouragement and love.

Conditional Generative Denoising Autoencoder

Σάββας Καρατσιόλης

Πανεπιστήμιο Κύπρου, 2019

Η διατριβή μελετά ανοικτά προβλήματα στο χώρο της Μηχανικής Μάθησης και προτείνει μοντέλα επίλυσης ή μετριασμού τους. Ένα από αυτά τα προβλήματα είναι η σύνθεση εικόνων που υπόκειται σε συγκεκριμένες συνθήκες. Προτείνεται ένα πρωτότυπο μοντέλο με σαφές μαθηματικό υπόβαθρο που στηρίζει την λειτουργία του στον κλασσικό κωδικοποιητή αφαίρεσης θορύβου (denoising autoencoder). Το μοντέλο παράγει εικόνες βασισμένο στις προτιμήσεις του χρήστη ως προς το αν αυτός επιθυμεί να αποκομίσει εικόνες που ανήκουν σε κάποια ή κάποιες επιλεγμένες κατηγορίες και ταυτόχρονα αποκλείουν κάποιες άλλες επιλεγμένες κατηγορίες του πεδίου εκπαίδευσης. Για παράδειγμα, αν το μοντέλο εκπαιδευτεί στην παραγωγή εικόνων προσώπων που κατέχουν συγκεκριμένα χαρακτηριστικά όπως χρώμα μαλλιών και ματιών, σχήμα προσώπου, φύλο, παρουσία μουστακιού ή γενειάδας, παρουσία γυαλιών ή καπέλου, κατάταξη ηλικίας κλπ., το μοντέλο μπορεί να συνθέσει εικόνες προσώπων που κατέχουν κάποια συγκεκριμένα από αυτά τα χαρακτηριστικά και ταυτόχρονα να μην κατέχουν κάποια άλλα. Η αξία του προτεινόμενου μοντέλου έγκειται στην ικανότητα του να παράγει εικόνες καλής ποιότητας σύμφωνα με τις προκαθορισμένες απαιτήσεις χαρακτηριστικών αλλά και στο γεγονός ότι ο αριθμός των διαθέσιμων μοντέλων μάθησης μηχανών που διεκπεραιώνουν την ίδια λειτουργία είναι ελάχιστος σε αριθμό και η ποιότητα των εικόνων που παράγουν είναι γενικά μέτρια.

Η δεύτερη μελέτη ασχολείται με την μετάφραση εικόνων από ένα πεδίο σε άλλο. Προτείνεται ένα αρθρωτό μοντέλο που δέχεται μια εικόνα από ένα πηγαίο πεδίο και τη μεταφράζει σε μια αντίστοιχη εικόνα που ανήκει στο στοχευμένο πεδίο. Η εικόνα εισόδου διατηρεί σημαντικό μέρος της σημειολογίας της αλλά αναπαρίσταται με διαφορετικό τρόπο ο οποίος είναι συμβατός με το πεδίο εξόδου. Το μοντέλο αποτελείται από αυτόνομα δίκτυα που εκπαιδεύονται ξεχωριστά και ακολούθως ενσωματώνονται σε μια κοινή αρχιτεκτονική.

Η τρίτη μελέτη ασχολείται με τη βελτίωση γενίκευσης της διαδικασίας ταξινόμησης επιτρέποντας την επιλογή διαφορετικής επεξεργασίας για ένα μοτίβο, ανάλογα με την περιοχή εισόδου στην οποία ανήκει. Το προτεινόμενο μοντέλο υιοθετεί την παραδοχή ότι διαφορετικά

μοτίβα προερχόμενα από ένα πεδίο πληροφορίας δύναται να κωδικοποιούν διαφορετικά χαρακτηριστικά ανάλογα με την περιοχή της πληροφορίας από την οποία προέρχονται. Συνεπώς, η ομαδοποίηση των παραδειγμάτων μάθησης αναλόγως του βαθμού πολυπλοκότητας που απαιτείται για την ταξινόμησή τους, αποδεικνύεται χρήσιμη πηγή πληροφόρησης για τη φύση της επεξεργασίας που πρέπει να εφαρμόσει ο ταξινομητής.

Η τέταρτη μελέτη ασχολείται με την μετρίαση του προβλήματος μη ισορροπημένων συνόλων δεδομένων. Το συγκεκριμένο πρόβλημα είναι συχνό σε προβλήματα Μηχανικής Μάθησης που πραγματεύονται ιατρικά δεδομένα και ιδιαίτερα όταν τα δεδομένα αφορούν σπάνιες ασθένειες. Η μη ισορροπημένη πληροφορία έχει ισχυρά αρνητική επίδραση στην αποτελεσματικότητα των αλγόριθμων ταξινόμησης. Προτείνεται μία νέα προσέγγιση η οποία έχει ως στόχο την αναγνώριση των περιοχών της πληροφορίας που αντιστοιχεί στα τοπικά ελάχιστα και μέγιστα της συνάρτησης του μοντέλου. Ακολούθως, αυτές οι αντιστοιχίες χρησιμοποιούνται για την εκπαίδευση των παραμέτρων του μοντέλου. Με αυτό τον τρόπο, επιτυγχάνεται ο μετριασμός της επίδρασης του φαινομένου της μη ισορροπημένης πληροφορίας.

Η τελευταία μελέτη ασχολείται με τον υπολογισμό αποτελεσματικών χαρακτηριστικών αναπαράστασης για προβλήματα ταξινόμησης και βασίζεται στη μεγιστοποίηση της απόστασης μεταξύ των χαρακτηριστικών που αφορούν δεδομένα διαφορετικών κατηγοριών. Η χαρτογράφηση των χαρακτηριστικών γίνεται με συγκεκριμένα κριτήρια τα οποία συνθέτουν μια συνάρτηση βελτιστοποίησης της οποία η επίλυση προσφέρει κατανοητές μαθηματικές εκφράσεις, που όταν εφαρμοστούν στα δεδομένα του προβλήματος, παράγουν αναπαραστάσεις των βασικών δομών της πληροφορίας.

Conditional Generative Denoising Autoencoder

Savvas Karatsiolis

University of Cyprus, 2019

This thesis investigates several open research problems in Machine Learning and proposes methodologies that deal with them or mitigate their effect. One such problem is image generation according to specific characteristics. The first study proposes an original model and its mathematical foundation whose functionality relies on the conventional denoising autoencoder. The model generates images according to the user's preferences defining both the desired and the undesired image characteristics that are evident in the training dataset. For example, assuming the model is trained for generating images of faces that may or may not possess some specific characteristics like eye and hair color, face shape, gender, presence of a moustache or a beard, wearing a hat or glasses, being old or young etc., then it is able to generate images of faces possessing or not such characteristics. The value of the proposed model rises from its ability to generate good quality images according to predefined conditions (data labels), while the available Machine Learning models performing the specific task are very few in number and generally perform moderately.

The second study deals with image translation from one domain to another domain. It proposes a modular model that takes an image from the source domain and translates it to a corresponding image belonging to the target domain. The original image maintains a significant amount of its semantics but is represented in a way compatible to the output domain. The model is composed of individual, separately trained networks that are then embedded into a unified architecture.

The third study deals with improving the generalization ability of the classification process by allowing the selection of a different model function, depending on the region of the input space to which a pattern belongs. The suggested model makes the assumption that different patterns of the same domain may contain different features depending on the region of information they represent. Consequently, clustering the problem patterns according to the complexity required for their classification proves useful for choosing the most appropriate classifier for a specific pattern.

The fourth study deals with the problem of imbalanced datasets. The specific problem is often seen in medical-related Machine Learning problems and more specifically in problems involving rare medical conditions. Imbalanced information significantly diminishes the effectiveness of classification algorithms. A new approach is proposed that aims in recognizing the input information areas that correspond to local minimums or maximums of the model function. These associations are then used to adapt the model parameters. In this way, the effect of data misbalancing is mitigated.

Finally, the last study in this thesis deals with the calculation of efficient feature mappings for classification problems and relies on maximizing the distance between the feature mappings of patterns belonging to different problem classes. The feature mapping is performed according to specific criteria reflecting some desired qualities. These criteria compose an objective function which is optimized for providing self-explanatory mathematical functions that produce representations regarding the basic structures of the input information.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

LIST OF ALGORITHMS

LIST OF ACRONYMS

ADAM	ADAptive Moment
ADAP	Adaptive
AE	AutoEncoder
AI	Artificial Intelligence
ARTMAP-IC	Adaptive resonance theory mapping with instance counting
BFGS	Broyden–Fletcher–Goldfarb–Shanno
BN	Batch Normalization
CAE	Contractive AutoEncoder
CelebA	Celebrity Attributes
CGAN	Conditional Generative Adversarial Network
CG-DAE	Conditional Generative Denoising AutoEncoder
CIFAR	Canadian Institute for Advanced Research
CNN	Convolutional Neural Network
COCO	Common Objects in Context
CoGAN	Coupled Generative Adversarial Network

CPU	Central Processing Unit
CycleGAN	Cycle-consistent Generative Adversarial Network
DAE	Denoising AutoEncoder
DBM	Deep Boltzmann Machine
DBN	Deep Belief Network
DCGAN	Deep Convolutional Generative Adversarial Network
DNA	Deoxyribonucleic Acid
EBGAN	Energy Based Generative Adversarial Network
EBM	Energy Based Model
GAN	Generative Adversarial Network
GPU	Graphical Processing Unit
GSN	Generative Stochastic Networks
GTSRB	German Traffic Sign Recognition Benchmark
ILSVRC	ImageNet Large Scale Visual Recognition Challenge
ISOMAP	ISOmetric MAPping
IT	Information Technology

KL	Kullback Leibler
k-NN	K-Nearest Neighbors
L-BFGS	Low memory Broyden–Fletcher–Goldfarb–Shanno
MCMC	Markov Chain Monte Carlo
MLP	Multi-Layer Perceptron
MNIST	Modified National Institute of Standards and Technology
MNIST_Noisy	Noise-corrupted MNIST dataset
MNIST_Rot	MNIST dataset corrupted with random rotations
MNIST_RotBck	MNIST dataset corrupted with random rotations and background noise
MSE	Mean Square Error
NN	Neural Network
PCA	Principal Components Analysis
PID	Pima Indian Diabetes
PVAE	Plain Variational AutoEncoder
RBF	Radial Basis Function
RBM	Restricted Boltzmann Machine

ReLU	Rectified Linear Unit
RMSProp	Root Mean Square Propagation
SVHN	Street View House Numbers
SVM	Support Vector Machine
TAID	Total Atomic Inter-lass Distance
t-SNE	t-Distributed Stochastic Neighbor Embedding
UCI	University of California Irvine
VAE	Variational AutoEncoder
VC	Vapnik Chervonenkis
VGG	Visual Geometry Group
WGAN	Wasserstein GAN
XOR	eXclusive Or
ZFNET	Zeiler and Fergus NET
β-hcg	Human chorionic gonadotropin

LIST OF SYMBOLS

L_c	Gradient of the output logit c in respect to the input
λ	Chapter 1: Scalar coefficient of the regularization penalty
$\phi_l(I)$	Output of layer l for a specific input I
$R(I')$	Regularization term acting on pattern I'
$f(k,l)$	Chapter 2: Value of a convolutional kernel for horizontal shift k and vertical shift l
$x(i-k,j-l)$	Chapter 2: Value of the input at some coordinates i,j shifted by operators k,l
$sign(.)$	Sign function
$f_c(x+z)$	Output of a model with input x and some input disturbance z
$p(y\|x)$	Probability of output y conditioned on input x
$p(x,y)$	Joint probability of x and y
$\Omega(h)$	Chapter 3: Penalty function of generalized autoencoders
$f(.)$	Chapter 3: Encoding function of the generalized autoencoder
$g(.)$	Chapter 3: Decoding function of the generalized autoencoder
$\|\|.\|\|_F$	Frobenious norm

$\hat{x}$	Pattern x corrupted with noise
$c(\hat{x}\|x)$	Noise corrupting process resulting in $\hat{x}$ given an input x
$x \sim p(x)$	Drawing a sample x from a distribution $p(x)$
$\mathcal{N}(\mu, \sigma^2)$	Gaussian Noise process with mean μ and variance σ^2
$\mathcal{N}(0, \sigma I)$	Gaussian Noise process with mean 0 and variance σI
I	Chapter 3: Identity matrix
$P_\theta(Z\|X)$	Chapter 3: Posterior distribution
$Q_\theta(Z\|X)$	Chapter 3: Approximate posterior
$D(x)$	Output of the discriminator structure of a Generative Adversarial Network
$G(z)$	Output of the generator structure of a Generative Adversarial Network
$r(.)$	Chapter 3: Output of an intermediate layer r of a Generative Adversarial Network's discriminator structure
$\mathbb{E}_{x \sim p_{data}(x)}(.)$	Expected value of a function of x sampled from distribution of $p_{data}(x)$
$clip(x, a, b)$	Clip value of x if smaller than a or larger than b
f_w	Output of the critic structure in a Wasserstein Generative Adversarial Network

$q^t(x)$	Distribution of x at time-step t
$T(x' \mid x)$	Transition distribution from state x to state x' in a Markov process
A	Transition distribution matrix of a Markov process
$A_{i,j}$	Transition probability from Markov state j to state i
$diag(\lambda)$	Diagonal matrix having the eigenvalues of A in its diagonal
$Q_\theta(Z \mid X)$	Chapter 4: Encoding function of a denoising autoencoder resulting in a latent distribution Z
$P_\varphi(X \mid Z)$	Chapter 4: Decoding function of a denoising autoencoder provided a latent distribution Z
$D_{KL}(Q \parallel P)$	Kullback-Leibler divergence between distributions Q and P
Z_1	Chapter 5: Latent vector provided by the encoder structure of the conditional generative denoising autoencoder
Z_2	Chapter 5: Latent vector provided by the classifier structure of the conditional generative denoising autoencoder
$Q_\theta(Z\|\tilde{x}_i)$	Encoding function acting on a noise corrupted pattern resulting in a latent distribution Z
φ	Chapter 5: Parameters of the encoder structure
θ	Chapter 5: Parameters of the decoder structure
$P(Y\|Z_1)$	Chapter 5: Output distribution of the classifier due to an input Z_1

$P_{\varphi}(X\|Z_1, Z_2)$	Chapter 5: Distribution of X conditioned on the joint distribution of Z_1, Z_2 provided by the encoder and the classifier structure of the conditional generative denoising autoencoder
$\{Z_1, Z_2\}$	Joint distribution of $Z_1,\ Z_2$
$MB_{labeled}$	Mini-batch comprising of labeled examples
$MB_{unlabeled}$	Mini-batch comprising of unlabeled examples
$\widetilde{MB}_{labeled}$	Noise corrupted mini-batch of labeled examples
$\widetilde{MB}_{unlabeled}$	Noise corrupted mini-batch of unlabeled examples
$\propto$	Proportional operator
$U(0,1)$	Uniform distribution in the range [0,1]
$M: Z \rightarrow \hat{X}$	Mapping function M to transform pattern Z to an $\hat{X}$
L	Chapter 6: Latent distribution of target-domain autoencoder
K	Chapter 6: Dimensionality of the latent distribution of the target-domain autoencoder
$\vec{r}_m$	Chapter 6: Binary vectors sampled from the latent distribution
$F = \{T, g, D\}$	Chapter 6: Model F results from merging sub-modules T, g, D
$\overline{l^k}$	Mean value of element k of vector l
$\|\ Cases^+\|$	Chapter 7: Number of training set patterns belonging to the positive class of an SVM

$\lvert Cases^{-} \rvert$	Chapter 7: Number of training set patterns belonging to the negative class of an SVM
P	Chapter 7: The population of the genetic algorithm
p_i	Chapter 7: An individual of the population of the genetic algorithm
$f(p_i)$	Chapter 7: Fitness function of individual p_i in the population
N	Chapter 7: The size of the genetic algorithm population
x_i	Chapter 7: Elements of individual p_i that hold the pointer-values
SVM^{+}	Chapter 7: Positive region of an SVM signifying the positive side of the discriminating boundary
SVM^{-}	Chapter 7: Negative region of an SVM signifying the negative side of the discriminating boundary
K^i	Chapter 8: Number of hidden neurons in layer i
K	Chapter 8: Number of hidden neurons
$h(z_k^i)$	The output of neuron k at layer i using an activation function h
n	Chapter 8: Number of input pattern dimensions
z_k	Chapter 8: Activation of neuron k in the hidden layer
y	Chapter 8: Scalar output of the Neural Network

w_{ij}	Weigh connecting neuron j of the previous layer with neuron i of the current layer
$E[f(X_+)]$	Chapter 9: Expected value of the feature mapping of the patterns belonging to the positive class of the problem
$E[f(X_-)]$	Chapter 9: Expected value of the feature mapping of the patterns belonging to the negative class of the problem
$\lfloor . \rfloor$	Floor function
K	Chapter 9: Number of problem classes
h	Chapter 9: Hypothesis function of a classifier model
X_+	Chapter 9: Random variable representing a sample drawn from the distribution of the problem's positive class
X_-	Chapter 9: Random variable representing a sample drawn from the distribution of the problem's negative class
Φ_i^j	Chapter 9: Feature mapping values of function i for pattern j
$f_i(x^j)$	Chapter 9: Feature mapping function i acting on pattern j
d	Chapter 9: Margin between the expected value of the feature mappings of the patterns of the positive class and the expected value of the feature mappings of the patterns of the negative class
$\lvert . \rvert$	Absolute value
M	Chapter 9: Number of positive-class patterns in the dataset

N	Chapter 9: Number of negative-class patterns in the dataset
a	Chapter 9: Minimum distance of the feature mapping of a pattern from the expected value of the feature mappings of the positive class patterns in the increasing direction of feature mapping values
β	Chapter 9: Minimum distance of the feature mapping of a pattern from the expected value of the feature mappings of the positive class patterns in the decreasing direction of feature mapping values
γ	Chapter 9: Minimum distance of the feature mapping of a pattern from the expected value of the feature mappings of the negative class patterns in the increasing direction of feature mapping values
δ	Chapter 9: Minimum distance of the feature mapping of a pattern from the expected value of the feature mappings of the negative class patterns in the decreasing direction of feature mapping values
X_{-a}	Chapter 9: A random variable representing the distribution of data feature mapped beyond distance a
$X_{-\beta}$	Chapter 9: A random variable representing the distribution of data feature mapped beyond distance β
$X_{+\gamma}$	Chapter 9: A random variable representing the distribution of data feature mapped beyond distance γ
$X_{+\delta}$	Chapter 9: A random variable representing the distribution of data feature mapped beyond distance δ
A	Chapter 9: Candidate feature space of a synthetic dataset

B	Chapter 9: Candidate feature space of a synthetic dataset
$\mathring{f}(.)$	Chapter 9: Vector of feature mappings of a pattern produced with different mapping functions
s	Chapter 9: Sparseness coefficient
l	Chapter 9: Number of attributes used in the calculation of a feature

Chapter 1

Introduction

1.1 A very brief history of Artificial Intelligence

Human intelligence has been manifested into building artificial intelligence of some sort from the ancient times. Especially in ancient Greece, people were fascinated by myths of capable inventors and crafters that created intelligent (in terms of the available technology) machines. The Greek God of blacksmiths, craftsmen and carpenters Hephaestus and the Roman god Vulcan were believed to inspire people for building various machines. More interestingly, people started processing the idea of building machines or structures and statues that had human abilities and characteristics. In ancient Cyprus, Pygmalion carved an iron statue of a woman called Galatea and fell in love with it. According to the myth, goddess Aphrodite gave life to the statue and Pygmalion married Galatea and had a daughter with her. They called her Paphos which is the name of a city of the island. Another famous human-like creation was built by Hephaestus and was named Talos. It was a man made of bronze and had a divine purpose: to protect Europe, the mother of king Minos who ruled the island of Crete. Europa was especially important to Zeus because he was once in love with her and a Cretan myth says that he abducted her by taking the form of a white bull to carry her away.

There is a strong link between the development of philosophy and rational thinking and the interest in creating human-like entities. It seems there's always been a strong spiritual need, almost as strong as the human eagerness to explore bigger questions regarding genesis and religion. In the ancient years, this urge most probably rose from a romantic attitude and a fictitious disposition. Maybe that is why during those years all myths about human-like machines involved an intervention from gods in order to gain their lively features. Humans were just discovering their broad intellectual capacity and were "explaining" physical phenomena through human-like god abilities, like the thunder-throwing Zeus. People thinking of human-like machines was a natural progression taking place. Without having a clue on how human brain works they could not disentangle the idea of a human solely building an intelligent machine from the idea of a human building a machine and gods giving it life.

An important milestone was established years later by Aristotle, who was inspired by his teacher Plato to describe syllogism based on deductive reasoning. This was the first time the human thought takes the form of a formal method and an algorithmic hypostasis. For almost two thousand years after Aristotle, many inventors, great thinkers, philosophers, mathematicians and artist came up with interesting (either scientifically or historically) ideas, theories and inventions relating machines with human characteristics and abilities. Of course, tremendous discoveries took place during these years in many scientific fields like mathematics and physics. But in terms of machine learning innovations there was more of a mental struggle for compiling appealing theories and concepts, exploring possibilities and engraving the foundations of the field. Al Jazari (1136-1206) wrote the "Book of knowledge of ingenious mechanical devices" which included a programmable orchestra consisting of mechanical humanoids. Ramon Llull (1232-1315) invented Ars Magna, a mechanical processor that was given a question from the user and returned an answer based on logic arguments. The answer is composed of several statements generated by the use of charts and visual aids found in an accompanying book. Llull was also a theologian so he mainly used his invention to disprove Muslim religion over Christianity through logic. Despite the theological bias of his device, it is considered as a remarkable piece of primitive machine learning technology. The great French philosopher Rene Descartes (1596-1650) proposed the historically important idea of "body-mind duality" in his famous book "Principles of Philosophy". Nowadays this idea seems eccentric and Descartes is mostly known about his famous quote "I think, therefore I am" rather than his idea of separated "substances". Nevertheless, his argument was a stepping stone and a motive for many great

thinkers to analyze what intelligence is. Rene Descartes also suggested that the "dictionary" of a universal language should consist of primitive elements [1]. This proposition led Gottfried Leibniz (1646-1716) to the conceptualization of the alphabet of human thought [2]. According to his theory, the systematic combination of these simple elements under some syntactical rules can generate complex structures required to represent highly complex entities like human language. Leibniz also developed the stepper reckoner, a mechanical calculator that performed multiplication and division which was a significant improvement over Pascal's calculator. In 1726, Jonathan Smift published a satire called Gulliver's travels which was describing an engine that was able to provide a speculation, an inductive hypothesis based on acquired knowledge. In pure satiric and ironic spirit, the writer claimed that even the most ignorant person can use the machine for writing books in philosophy, poetry and mathematics. In the same spirit, Wolfgang Von Kepmpelen was the conductor of one of the most famous hoaxes in human history. He claimed and eventually convinced a lot of people that he processed an automated machine for playing chess. Actually this involved a human player hidden into the machine moving the strings of the act. The Turk hoax [3] incredibly held for decades before being revealed and proved an already stressed point: people tend to maintain a romantic attitude and a fictitious disposition towards artificial intelligence. This holds true since the ancient years and has also been the case during the 20^{th} century during which artificial intelligence exploded and really started emerging as a promising scientific field. Inevitably, this positive attitude holds nowadays as well but it might be justifiable for the first time since machine learning models have reached and even surpassed human performance on specific tasks.

The twentieth century has been a very productive era for science in general. As far as artificial intelligence and machine learning are concerned, it was the era during which their parental sciences flourished: computer science has become one of the most popular research areas involving a broad spectrum of topics while electronics and computer hardware have rapidly evolved providing the stepping stones for complex numerical analysis. Furthermore, neuroscience presented astonishing achievements and revealed some of the magic taking place in the human brain. Neuroscience and cognitive sciences have always been an inspiration for machine learning researchers in the sense that approximate brain function models should provide approximate functionalities, which seems to be adequate for solving real-life problems provided the enormous capabilities of the human brain. The elasticity of human brain manifested by effective rewiring in order for the brain to overcome specific region failures [4] reinforced

the belief that there is a universal training algorithm implemented in the brain that controls simultaneous learning of different tasks. Furthermore, it is evident that simple elements like the individual neurons may not provide any meaningful or impressive functionality when operating alone but connecting them into a large network provides a model with substantial learning capacity. Fukushima [5] proposed an image processing architecture based on his studies on the mammalian visual system, the so called neocognitron. Almost twenty years later this biology inspired model was established with minor modifications by Le Cun [6] as the basis of convolutional neural networks. Fukushima also established another neuroscience paradigm through his work on the original version of the cognitron: the rectified linear unit [7]. Naturally, the activation function proposed by Fukushima was more complex since it encapsulated a biological analogy to the brain functionality. Glorot et al [8] proposed a simplified version of this activation function possessing very nice properties that make it very compelling when considering what activation function to use in a neural network. It is not surprising that it is considered the first option when activation functions are concerned. Jarrett et al [9] performed a comprehensive study supporting the benefits of rectified linear units.

While neuroscience provided inspiration to the machine learning community, its inability to explain the complex brain functionality and its internal mechanisms, the neuron modelling complexity and the lack of training algorithms that work efficiently with such complex neural models kept machine learning and neuroscience to a distance. The Machine learning community will most probably keep carefully observing every neuroscience achievement or development but receive this work as a guideline and maintain it in a pool of inspiration rather than a research beacon. Machine learning models closely reflecting biological models with increased fidelity tend to produce poor results when trained on meaningful tasks. On the contrary, simplified versions of biological models produce promising results. For example, the McCulloch-Pitts neuron [10] was an oversimplified early attempt to model the brain neurons. It was a linear discriminator and its weights were engineered to deliver the desired function. About ten years later (1952) Hodgkin and Huxley [11] proposed their conductance-based model of the neuron which could explain the propagation of internal signals in the squid axon through a set of equations. This achievement granted them the Nobel Prize in Physiology/Medicine in 1963. Despite this success, machine learning researchers kept working on simplified models of neurons but the fact that a functional neuron can be simplified to electrical signals and appropriate signal equations was an important confidence boost. Some years after the work

publications of Hodgkin and Huxley and McCulloch-Pitts there was another important breakthrough. Rosenblatt [12] proposed the perceptron which was accompanied with an adaptive weight learning algorithm based on given examples for the first time in machine learning history. The neuroscience Hebbian theory [4] of adaptive learning (1949) was an early attempt to explain the increase of synaptic strength between simultaneously activated neurons but is not considered as supervised learning.

While the perceptron truly raised the hopes for the development of models that could be efficiently trained on available patterns there was a significant criticism on their natural flaw of being linear models. Their inability to learn the XOR (Exclusive OR)function stressed out that linear models cannot be considered as serious candidates for tackling real problems. Soon neural networks experienced their first wave of skepticism which drew them out of popularity. In the early 1970's logic programming and computational linguistics became the main form of applied Artificial Intelligence. This was primary due to the development of Prolog, a programming language based on first-order-logic. Prolog became very popular and is still used for theorem proving, expert systems design, automated planning and natural language processing. Logic programming along with search algorithms, statistical reasoning, planning and symbolic reasoning were the primary approaches to Artificial Intelligence for many years. Connectionist models were never considered among the first paths to follow when applying Artificial Intelligence. This only changed during the last couple of decades when their popularity raised significantly and nowadays are the most popular AI approaches by far. However, during the 1970's neural networks lost their credibility due to their limitations. After several quiet years on neural networks research activity, a significant breakthrough revitalized the interest of the research community in neural networks: the back propagation algorithm. Rumelhald et al [13] (1986) presented an algorithm for training multi-layer neural networks and this in turn provided the means to learn efficient internal representations. During the following decade many researchers were building applications trying to solve real-world problems and there was (again) a general belief that everything is possible. The performance boost provided by back-propagation was reflecting the fact that multiple layers could now be trained which in turn allowed for the construction of complex internal representations. The input is mapped to simple features at the first layer which in turn are mapped to more complex features which in turn are mapped to higher-level concepts in the next layer. This hierarchy in feature mapping is called representation learning. The first couple of layers represent the low-level features that before

multi-layer neural networks needed to be hand-engineered and this required hard work and field expertise. While moving away from the input and towards the output, previous layers' outcome are combined to form higher-level abstract representations. The natural thing to do is to add more and more layers to enhance performance at the cost of computational complexity. Despite the expectations, deeper models were very difficult to train and often resulted in worse performance than shallower models. This frustrating fact and the disproval of the expectations accompanying deep models led to a second wave of criticism that once again hurt the credibility of neural networks. Perhaps the most significant problem during that period was the lack of powerful hardware to experiment with deep models. On the other hand Cortes and Vapnik [14] published their support vector machine algorithm which contributed in making kernel methods very popular. This also had a negative impact on the popularity of neural networks. After some years of very little theoretical advancement, neural networks finally got their third and perhaps mightier boost, entering the era of deep learning.

Figure 1.1: Worldwide interest trends about the subject of deep learning through recent years as registered by google trends. The numbers represent relative popularity in terms of the highest value registered.

The important factors for this were the presence of powerful hardware to execute the computations required to train models with a huge number of parameters and the availability of large-scale datasets due to the "digital world" era. Besides the supporting technology a scientific trigger was necessary. This triggering was provided by Hinton in 2006 with the introduction of greedy layer-wise pre-training [15] that helped overcome one of the major problems of training

deep architectures: vanishing gradients. Sometime later (2007), Bengio et al [16] provided the theoretical foundations and necessary proofs for the algorithm. From that point on, many researchers provided important theories, algorithms and novel architectures that popularized deep learning and supported the development of impressive applications that in some cases outperform human performance [17, 18, 19].

1.2 Current AI adoption

Companies nowadays are applying AI in functional areas in which they likely have existing analytic applications, building atop that base [20]. Recent years surveys show that half of the organizations in the USA use deep learning while a great portion of the organizations use other AI algorithms. Figures 1.2 and 1.3 [20] show the stage of maturity for AI adoption in general and per industry.

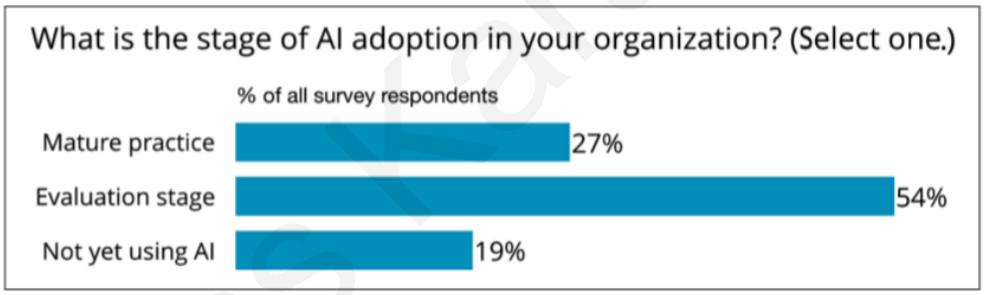


Figure 1.2: Stage of Maturity for AI adoption in USA Organizations [20].

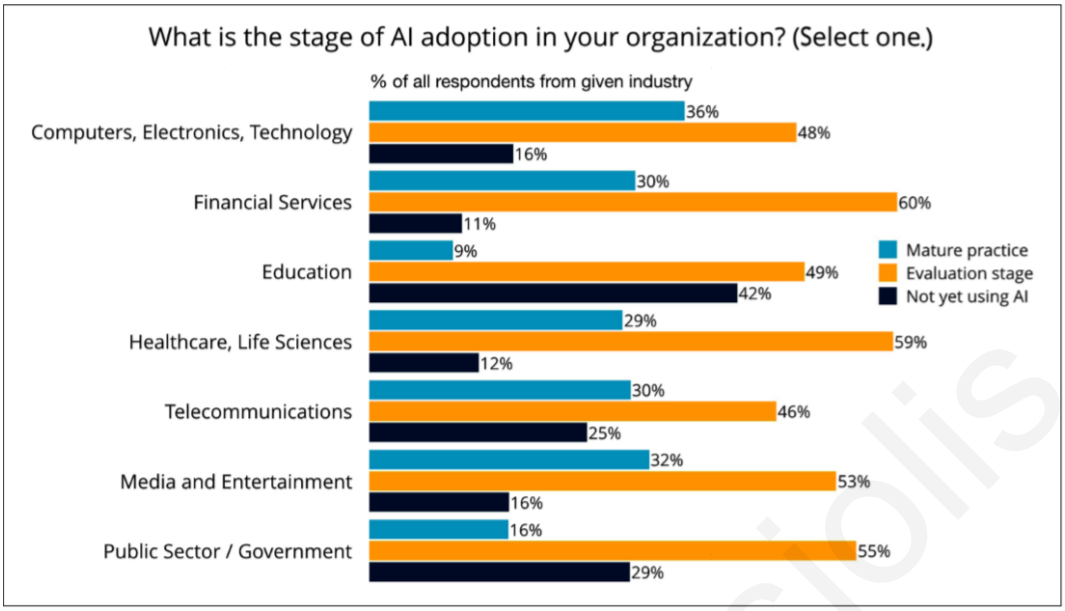


Figure 1.3: Stage of Maturity for AI adoption per industry [20].

At the same time, 61% of the companies plan to commit at least 5% of their IT (Information and Technology) budget in the next 12 months to AI projects [20]. These facts reveal that AI is not constrained to a research framework anymore. It is a fast developing sector and many of its fields are producing valuable results and are considered as mature technologies. Currently, Generative Adversarial Network (GAN) [21] is one of the dominant models pushing the specific research direction forward.

1.3 Current Artificial Intelligence Challenges

On the research field there are two strong trends: developing a deep understanding of Neural Networks by especially concentrating on their weaknesses and contributing to the unsupervised learning section of AI primarily focusing on generative models.

1.3.1 Building a deeper understanding of Neural Networks

Several techniques have been proposed for visualizing neural networks and especially convolutional networks. The most popular of these methods can be loosely divided into indirect methods, direct methods and through-an-adversary methods. Indirect methods mainly perform forward passes through the network and obtain a neuron's activations trying to understand how

they are influenced by different images. This strategy assists the detection of the patches or images that excite a neuron the most. A similar technique is applying occlusion using sliding windows over the input images [22]. For each sliding window the corresponding image patch is set to zero values and a heat-map is constructed based on the output class probabilities. Low intensity heat-map locations provide information on what parts of the images are mostly used for the classification. Another way to visualize what the network learns to detect, is by visualizing the weights of the network layers, especially the ones of the first layer. Any other layer's weights are very hard to interpret. Of course, convolutional networks always calculate low level feature-detectors at their first layer (like edge detectors and primitive image components' detectors) so this method is not that helpful. Finally, a popular visualization technique for deep networks is dimensionality reduction (usually to 2-D representations) of the features fed to the final soft-max classifier. The location of the images producing this 2-D map can provide some information about how the network constructs its feature space. The most popular algorithm used for dimensionality reduction visualization is t-SNE (t-Distributed Stochastic Neighbor Embedding) [23].

Direct methods apply backward passes to update the network's input towards reinforcing a specific neuron's activation or a specific class output [22, 23]. According to this method, an image is selected from the input distribution and all gradients except the ones contributed from the neuron of interest are neglected. A well performing flavor of this approach is guided propagation [22] which considers the ReLU (Rectified Linear Unit) activation function's zero-value region and does not account for the gradient contributions of zero-activated neurons. A very interesting alternative is to start with a random image and use the gradient of the desired output class logit L_c to apply a gradient ascent step to the input. It is essential to add regularization to the specific optimization objective function to avoid exaggerated unrealistic artifacts in the produced image. This regularization term may be the l_2 norm of the image I such as

$$\underset{I}{argmax}\ L_c(I) - \lambda \parallel I \parallel_2^2 \tag{1.1}$$

Instead of visualizing the input space (image) explicitly, one can visualize the data gradients of the gradient ascent process and form a good idea on how the input influences the model [25]. More specifically, the gradients of the image in respect to the specific class output are squashed by taking the maximum absolute gradient value for each pixel update along all dimensions of a

particular channel. In the same analogy, a random image I' could be transformed to a specific image I^* by minimizing the Mean Squared Error (MSE) of its features ϕ at a specific layer l and the features of a given image I at the same layer [26]. The optimization objective is

$$I^* = \underset{I'}{argmin} \parallel \phi_l(I) - \phi_l(I') \parallel_2^2 + R(I') \tag{1.2}$$

with $R(I')$ being a regularization term applied on the input image. The pursuit of solving a very frustrating problem of machine learning models in general may also promote a better understanding of the inner mechanisms of neural networks. The specific problem arises by slightly (but strategically) altering an input image to fool a network into making a false decision. For example, carefully crafted image patches can be inserted to an image which is otherwise correctly classified with high confidence and cause a misclassification also accompanied with a high confidence [27]. This frustrating observation is shown in Figure 1.4 [28] where an image patch showing a distorted version of a toaster fools the model into misclassifying a banana. Interestingly, placing a patch of an actual toaster and not a distorted version of it does not fool the model. The distorted version of the toaster image is carefully crafted for maximizing the detection of features that signify the presence of a toaster, overwhelming the decision process of the model.

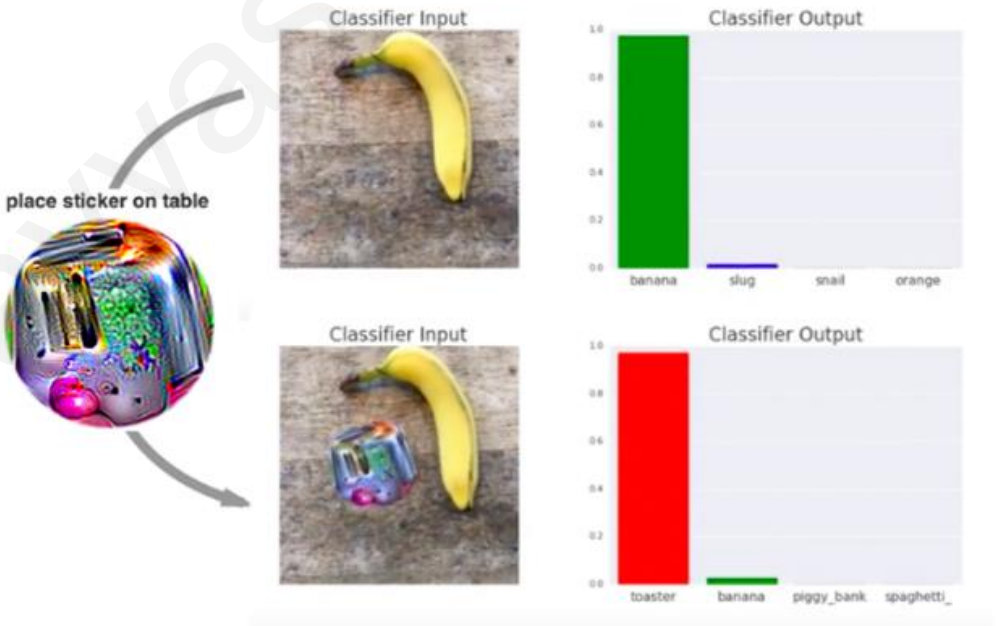


Figure 1.4: A carefully crafted patch of a toaster fools the model into not detecting the banana [28].

Crafting a surrealistic image of a toaster to maximize the appropriate feature detection mechanisms is one thing. Using it for preventing the model from detecting a well-defined, size-dominating object (like the banana in the example), inevitably raises some concerns. It seems that image features are learned in a way that neglects an object's spatial relations. The model seems to behave greedily during training and is only interested in expanding the classification-misclassification gap by constructing high level features that are monolithic and are over-describing versions of the input data. The problem features are not set to a "scale" in the sense that a feature may completely dominate the decision-making process by eliminating the influence that other features have on this decision. In Figure 1.4, the model seems to deal with the image using a rather clumsy approach. It considers both objects (the banana and the toaster patch) as a merged single object upon which it must make a classification decision based on the magnitude of detected features. Since the patch is crafted to maximize toaster features detection and since during training toasters and bananas were most probably very rarely or never presented together, the specific decision is not surprising at all.

1.3.2 The importance of generative models

Training without labels is the learning regime that is closest to how humans learn in practice. Our brains have a capacity of about 10^{14} (100 trillion) parameters (synapses) in a span of an average lifetime of 10^{9} seconds which means that for each second of input 100000 parameters must be updated [29]. This means that a tremendous amount of training data is required to support the update of such a large number of parameters. Perception and preconception input can supply this vast amount of training data. This implies that the brain is doing a significant amount of unsupervised learning. Joshua Bengio [30] recently expressed the opinion that brain seems to be learning continuously chasing a target, being slightly nudged towards it. According to this view, any perceptual input snapshot serves the equivalent of labeled data to be predicted and the prediction error serves as means to adjust the synapses of the brain. Yann Lecun [31] also expressed a similar opinion regarding the amount of unsupervised learning occurring in the human brain.

While the importance of unsupervised learning is understood by the research community, the accomplishments of generative models have been less astonishing than the ones achieved by supervised learning. Some of the reasons for this unbalanced situation are the following:

- Unsupervised learning is a process residing at the core of human learning and thus is more difficult to access, understand and express.

- It is easier to compress a concept into a single dimensional value and even better to a binary value like the labels we use in supervised learning. This blurs the interaction of the concepts that synthesize information which is a key requirement for unsupervised learning.

- Efficient learning without labels requires a near holistic knowledge of the data field. Constructing features out of data is relatively easy and it is a process that we already do very efficiently with supervised learning. Learning the interactions of these features to a degree of expressing the input data space according to these complex interactions is difficult. This is a task that seems to be missed by current Neural Networks and it is their main limitation.

- Mastering the task of learning the semantic meaning of data could provide the stepping stone towards overcoming many problems of current generative models. Unfortunately, this is a difficult task not mastered yet.

- Good generative models may require build-in knowledge and reasoning beyond learning the feature interactions and the data semantic meaning. It may be the case that the often-occurring assumption that unsupervised learning tends to serve supervised learning is totally wrong and the opposite is true.

- Biology of the brain and neuroscience discoveries inspired the progress of both fully-connected and convolutional neural networks. Maybe, still undiscovered biological brain processes will provide a fundamentally new paradigm for unsupervised learning in the near future.

1.4 Thesis Contribution

The current thesis proposes, among other models, an innovative Conditional Generative Denoising Autoencoder (CG-DAE). A conventional Denoising Autoencoder architecture is enriched with a classifier that blends its feature mappings with the input of the decoder structure.

Meanwhile, the classifier learns based on the internal feature mappings of the encoder. This interaction adds new capabilities to the Autoencoder such as significantly better performance in terms of sample quality and the addition of a conditional nature on the sampling process. Embedding classification information obtained through supervised learning to an unsupervised learning model enhances the model's learning capacity and allows categorical data generation.

The proposed architecture applies unsupervised learning in a way that allows its support from supervised learning. The use of supervised learning enriches the generative model by providing some kind of feature space knowledge manifested by its mappings. During data reconstruction, the decoder of the Autoencoder considers this knowledge for building a sample according to the categorical feature mappings. During the categorical inference step which guides the generative model towards the desired categories and away from the undesired categories, the encoder considers the classifier's knowledge expressed through its internal mappings. This functionality exposes the fact that supervised learning adds a special kind of input-space awareness to unsupervised learning that has an advantageous impact on its performance.

The thesis also proposes a modular domain-to-domain translation network that translates an image from one domain to another domain. More specifically, the model takes an image from a source domain and outputs a corresponding image belonging to a target domain. The contribution of this model is two-fold. First, it deals with the problem of image translation without relying on a Generative Adversarial Network like all state-of-the-art models in the specific field and thus avoids the problems of mode-collapsing and training instability. Second, it proposes the embedding of pre-trained models in a unified architecture that achieves state-of-the-art results. These results suggest that integrating self-contained modules that perform different tasks might be beneficial for performing a more complex task. In other words, strategically combining modules capable of performing simple sub-tasks can lead to models that build upon embedded knowledge and perform complex tasks.

Another proposal of this thesis is the Region based Support Vector Machine algorithm that applies a different classification model for the input patterns depending on their characteristics. In practice, the input data is identified either as straight-forward or difficult to handle and the appropriate model takes over the classification task. This results in training two classification models and each one handles different input patterns. The model applies different feature mapping strategies that recognize solely different data relations and modifies the way it

conceives data according to the nature of the input. This strategy is fundamentally different from building a conventional ensemble method because the classifiers are of totally different nature and deal with different regions of the input. The strategy is also different from bagging algorithms because it builds the regions according to the patterns' degree of separability and not the error rate of the classifier.

Another contribution of the thesis is an adaptive neural network targets model for dealing with extremely imbalanced datasets. This algorithm applies an alternative approach for performing the classification task that takes advantage of the structural form of the model function. More specifically, the algorithm tries to discover what inputs would produce a neural network output that resides on the local minimum or maximum points of the given model function, depending on the problem class. Positive class inputs target local maximums while negative class inputs target local minimums. After finding these points of zero gradient the model updates its parameters by using a regression problem having as input the original patterns and as output these points. By iteratively applying this strategy the model function becomes smoother and gets a more convex form.

Finally this thesis proposes a novel feature mapping method that uses maximization of the Atomic Interclass Distance and aims in calculating features that improve data separation. The algorithm constructs simple well-defined features that are self-explanatory in the sense that are composed of mathematical expressions. The model is also able to identify the most important data attributes and the resulting representations output values that have a large margin when patterns belong to different classes.

1.5 Thesis Outline

The thesis is organized so that Chapters 2-4 provide the fundamental theoretical foundations of all aspects of the proposed model: deep learning and convolutional networks (Chapter 2), Unsupervised Learning and Generative Models (Chapter 3), Drawing samples and evaluating Generative Models (Chapter 4). Chapter 5 presents, analyzes and studies the proposed Conditional Generative Denoising Autoencoder. Chapter 6 presents an image translation model that takes an image from one domain and outputs a corresponding image belonging to another domain called "Modular domain-to-domain translation network". Chapter 7 presents an algorithm that combines several strategically designed Support Vector Machines (SVMs) to

construct a model that has a higher generalization ability than using a single model, by choosing the appropriate classifier to apply for every pattern requiring classification. The specific work is called “Region based Support Vector Machine algorithm for medical diagnosis on Pima Indian Diabetes dataset”. Chapter 8 presents the algorithm “Adaptive Neural Network target values for dealing with highly imbalanced datasets” which is an approach for dealing with highly imbalanced datasets primarily found in biomedical problems. Chapter 9 presents a feature mapping methodology called “Feature Mapping Through Maximization of the Atomic Interclass Distances”. Finally chapter 10 includes a discussion regarding the Conditional Generative Denoising Autoencoder and the rest of the algorithms and further discusses some conclusions and future work.

Chapter 2

Convolutional Neural Networks

Convolutional neural networks were the primary reason that neural networks in general became so popular the recent years. The specific modelling architecture is also the reason that deep learning has flourished during the last decade. Despite the fact that convolutional neural networks have been around for many years before the deep learning revolution, some technology barriers had to be overtaken in order for their advantages to be highlighted. Powerful graphical processing units (GPUs) can perform trillions of floating point operations per second and access local memory much faster than CPUs (Central Processing Units) as shown in Figure 2.1 [32]. At the same time the available information for the problems solved with machine learning has exponentially increased due to the use of computers for nearly every task performed in our everyday life. The availability of data acquisition devices at the majority of the professional fields and non-expert environments also contributed to the popularity of deep learning. Convolutional neural networks have been largely inspired by discoveries in the neuroscience field originated to Hubel and Wiesel [33, 34], in their work on the mammalian vision system. They used a cat to study how its neurons respond to visual stimuli by projecting visual patterns on a screen. Having the cat looking at these patterns revealed that its neurons

responded with increased activation only during the projection of specific patterns. The deciding factor that increased or reduced neural activation was the orientation of the patterns.

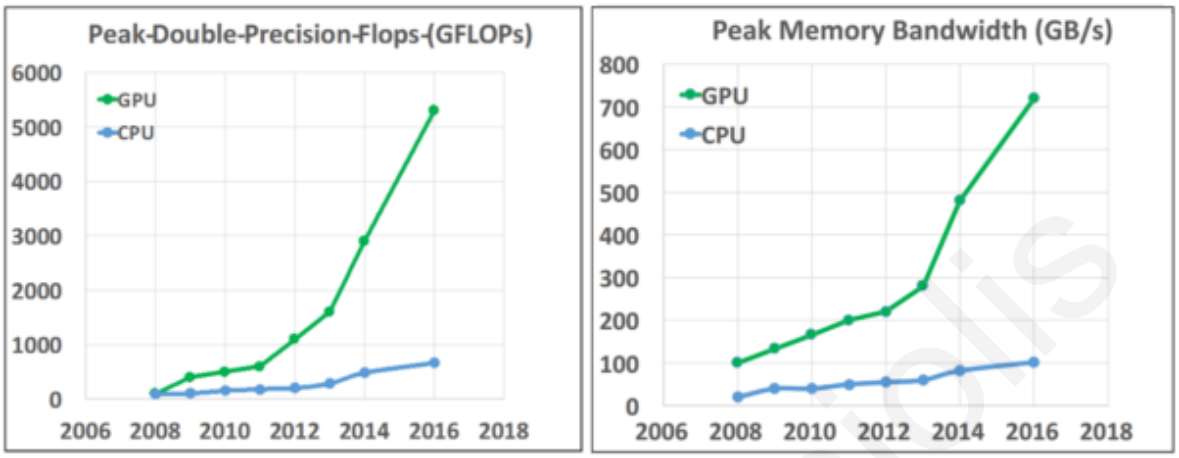


Figure 2.1: Performance comparison of GPUs and CPUs in terms of double-precision floating point operations per second and memory bandwidth. It is evident that GPUs are clearly more suitable than CPUs for deep learning applications since they require a huge number of model parameters' updates [32].

This gave birth to the idea that primary visual cortex performs edge detection in several orientations. Now we know that neurons in the visual cortex are lined up into a two-dimensional structure and cells have a local reception field. This essentially means that they are performing their functions considering only a small portion of the two-dimensional grid of the retina. These are the foundational principles of the convolutional neural networks: data must possess a two-dimensional structure of one or more channels of information and the neurons observe only an area of the grid-aligned data.

2.1 The convolution operation

As the name implies, convolutional neural networks use the convolution operation which is broadly used in signal processing. Convolution is a mathematical operation that uses a "catalytic" function on an input function in order to produce a third function which is mostly seen as the transformation of the input functions. This "catalytic" function is called the kernel.

In two-dimensional structures which is the data format of images, the convolution operation on a pattern x using a kernel f is formally defined as

$$C(i,j) = \sum_k \sum_l x(i-k, j-l)\, f(k,l) \tag{2.1}$$

where i, j are the pixel coordinates on the 2D plane and k, l are the shifting operators for applying the kernel on different locations on the input.

The convolution operation is commutative so if the index shifting is used in the kernel instead, the result is equivalent. Convolution operation can be visualized as shifting the flipped kernel and sliding it over the input performing the multiplications and additions at each of its positions one at a time. Figure 2.2 shows the operations graphically.

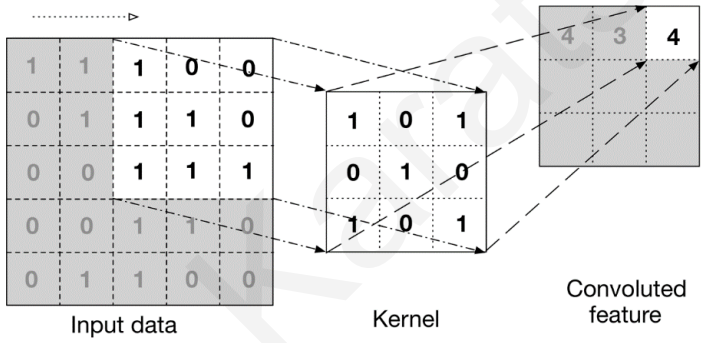


Figure 2.2: The convolution operation performed on a 2D input data grid. The kernel is shifted on the input image and overlapping elements are multiplied together. All products are accumulated to produce an element of the resulting feature. The shifting can be one location at a time or multiple locations depending on the stride property defined for the specific convolution operation.

Practically when implementing convolutional neural networks we do not apply a kernel flip since the kernel is a set of learnable parameters that are adjusted during the training process so the kernel learned can be considered as a flipped version of the non-flipped kernel. Mathematically the operation which is identical to the convolution without the kernel flip is called cross-correlation. This is why some times convolutional neural networks are considered as learning cross-correlation kernels instead of convolutional kernels.

2.2 Advantages of convolutional neural networks

Deep fully connected networks are very difficult to train and often provide no significant advantage in terms of performance. For a fully connected layer of m inputs and n outputs the model parameters are $m \times n$ and the computation complexity is $O(m \times n)$. Using l convolutional kernels of size k instead, limits the required parameters to $k \times l$ where k and l are usually several orders smaller than m. Analogically, the same reduced requirements stand for the execution time. This sparse connectivity comes from using sparse weights between the neural network layers and enables the addition of more and more layers since they can be handled by computer systems in terms of computations and memory-handling efficiency. Having sparse interactions does not necessarily mean that neurons do not indirectly "observe" the input. On the contrary, while neurons in the first model layers tend to have a small receptive field, neurons in the deeper layers have a complete or almost complete (depending on the kernel sizes used and the depth of the residing layer) view of the input and of the subsequent layers. This is one of the most important characteristics of convolutional neural networks because it is a manifestation of the principles of representational learning. Shallow layers learn low-level features and deeper layers create higher-level abstractions of the problem concepts. This feature hierarchy plays a critical role in why convolutional neural networks are so effective.

Another characteristic of convolutional neural networks that contributes to their effectiveness is parameter sharing. Parameter sharing and sparse connectivity are two sides of the same coin. As a consequence of the convolutional neural networks mechanics, the parameters of the model are reused many times for calculating different functions of the input or the previous layer. In some occasions parameter sharing is also referred to as having "tied weights". This property makes convolutional neural networks extremely efficient in terms of the number of model parameters when compared to fully connected models. For a convolutional neural network layer of l convolutional filters of size $a \times a \times b$ only $l \times a \times a \times b$ parameters need to be stored. Most of the time a is very small at the range of 3, 5 or 7 pixels and b is also relatively small since it is the number of the previous layer's channels. In comparison to the $m \times n$ weights required by a fully connected layer where m and n are the previous and next layer sizes respectively, the number of parameters of a convolutional layer is significantly smaller. Actually, the parameters of the last couple of layers of a convolutional neural network are usually fully connected layers and comprise the majority of the model's parameters in spite of

the fact that the model may have tens of layers. This is one of the reasons that researchers have proposed not using fully connected layers at all and replace them with convolutional layers with kernel sizes of 1×1. This architectural approach is called network in network and was initially proposed by Lin et al [35].

Besides efficiency, convolutional neural networks also possess the useful property of equivariance to translation. This means that a shifted input produces the same output that would produce without the shift. More formally $f(g(x)) = g(f(x))$ provided that g is a shifting function. This property is essential for machine learning algorithms that deal with two-dimensional maps like images because an image retains the exact semantic information when translated by any number of pixels in space. Image features that are comprised of small spatial spaces like edges and corners can be found in multiple regions of the image and can be considered as multiple translated versions of the same kind of information. While convolution is equivariant to translation, it is vulnerable to other kinds of transformations like scaling and rotation. This is why data augmentation using elastic distortions is almost always used for training models that provide state-of-the-art results.

Convolutional neural networks are used in deep learning architectures that provide state-of-the-art performance on tasks that seemed very difficult some years ago. The main benefit that rises from using deep convolutional neural networks in respect to fully connected models is that provided there is a huge amount of data to train them, they clearly surpass the performance of the latter.

2.3 The architecture of convolutional neural networks

Convolutional neural networks consist of two main components: the feature extractors and the classifier. The latter commonly consists of a couple of fully connected layers that perform the classification of the features detected by the previous layers. These features are extracted by the use of two types of layers: the convolutional layers and the pooling layers. The convolutional layers implement the actual feature detection by sliding their kernels on the input or the feature maps supplied by the previous layer. The pooling layers provide feature maps that hold spatial characteristics. By using a rectangular neighborhood window they provide a value based on the values in the feature map that belong to the selected region. This way, small translations in the

neighborhood do not have a great impact on the feature extraction process. Essentially, pooling layers provide a degree of invariance to translations of the input that are however relatively quite small in magnitude. For example, if a model needs to identify whether an image contains a face or not, it could examine the features and look for an eye on one location and another eye some space apart. The distance between the detected eye features is not a deterministic factor for rejecting the possibility of a face present in the image. Approximate eye distances that may differ because of image scaling, angle of view or objective facial characteristics still imply the presence of a face. The information by itself that two eyes are detected at the two sides of the concept "face" should be enough to support the belief that such a concept is present in the image. Furthermore, if this specific concept is detected, this supports the belief that a human being is observed. The most popular operation used by pooling layers is the maximum feature value of a region of the maps that are provided by the previous layer. This simple operation described in Figure 2.3 besides providing a degree of translation invariance to the model, it also holds no parameters and adds no extra memory requirements to the model.

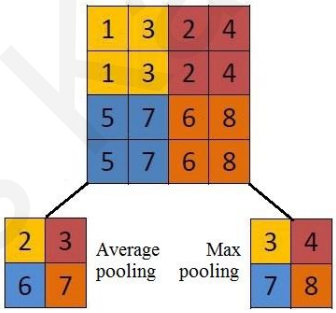


Figure 2.3: Example of average and max pooling operations on a feature map using a 2×2 kernel and a stride of 2.

Pooling layers are also very simple to apply. Another popular operation of the pooling layer is average pooling. As a matter of fact average pooling was introduced first in LeNet model [36]. Max pooling was used by Alex Krizhersky et al [37] in their famous model called AlexNet that became a milestone in convolutional neural networks research due to its success in ImageNet competition in 2012. The specific model was very successful in classifying natural images and contributed to the popularity of the convolutional architectures for dealing with such data. He et al [38] introduced pyramid pooling which is another slightly more sophisticated way to

perform the pooling operation. Pooling layers theoretically add an infinitely strong prior for learning translation invariant representations [39]. However, using pooling layers inevitably discards a huge amount of information and this fact cannot be overseen. Hinton et al [29] considers pooling operations to be unacceptable and this motivated his research for an alternative approach that provides the advantages of the pooling information without discarding valuable information [40, 41]. They proposed capsule networks that encapsulate object pose information in the learned representations. We will revisit capsules later. Despite the objections against them, pooling layers have been extensively used during the last years. There is another advantage of using pooling layers that led to this direction: they reduce the size of the feature maps provided by the previous layer. This reduction is accomplished due to the striding factor used in pooling layers. We define stride as the number of positions in a feature map of some layer or of pixels in an input image that a kernel or a pooling window shifts over before being applied again. These positions are not convolved with the kernel of the layer and this results in a great reduction in the size of the produced feature map. Generally, a convolutional or pooling layer of l kernels or pooling windows of size $n \times n \times d$, with zero-padding p applied on the input feature map of size $m \times m \times d$ and stride s, produces a new feature map of size $\lfloor \frac{m-n+p}{s} + 1 \rfloor$. Padding is the addition of extra zero-valued elements (pixels or features) to a feature map or image in order to control the size of the resulting feature map and to also perform convolutional operations on the far edges of the input. Padding is particularly useful when implementing very deep architectures because without it, the feature maps resulting from every convolutional layer keep being reduced in size and after a finite number of layers they become tiny and unusable in the sense of applying another convolutional layer.

2.4 The road to surpassing human performance

The first convolutional neural network as perceived nowadays (structurally and conceptually) was introduced by LeCun in the 1990's and is known by the name LeNet-5 [36]. Its architecture was greatly confined by the lack of sufficient hardware computation capabilities and memory capacity. It consisted of two convolutional layers, two pooling layers and two fully connected layers as shown in Figure 2.4.

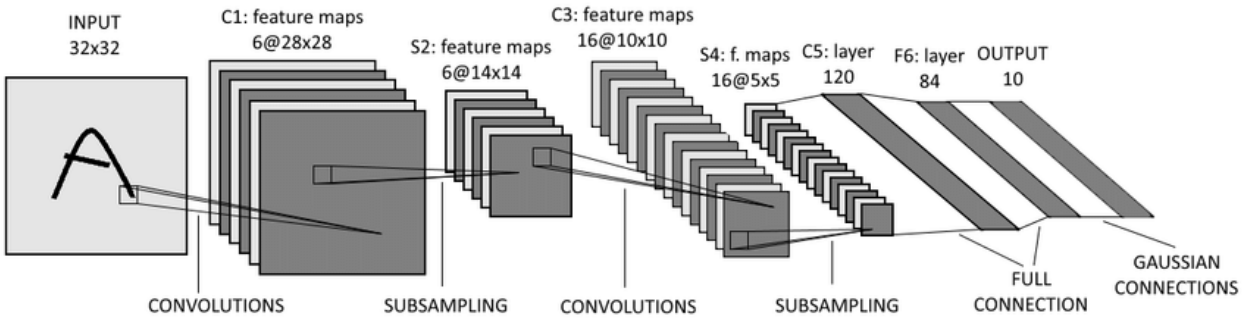


Figure 2.4: Architecture of LeNet-5. It is considered the first modern-form convolutional neural network containing convolutional layers and pooling layers denoted as subsampling [36].

LeNet-5 is a milestone in the history of convolutional neural networks but it proved to be ahead of its time and the next breakthrough came after computational hardware was able to support larger architectures that proved to be game-changers in the computer vision and machine learning fields. Several years after, in 2012, Alex Krizhevesky et al [37] proved that convolutional networks is the appropriate approach for computer vision problems. Their proposed model called AlexNet, won the ImageNet competition that year. AlexNet was very similar to LeNet-5 except the fact that it was much deeper and wider. Its architecture is shown in Figure 2.5.

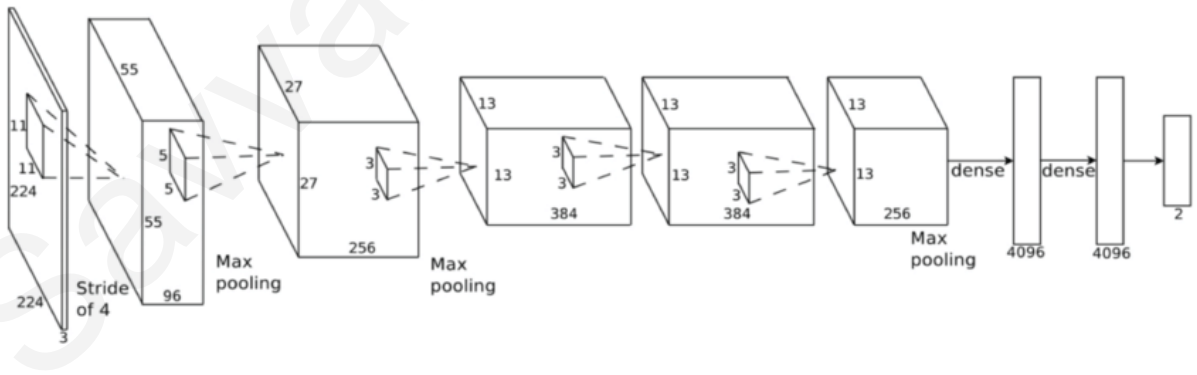
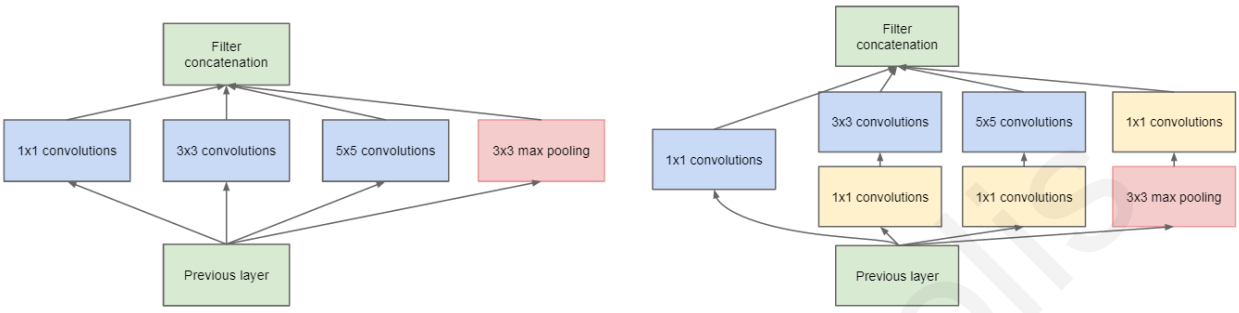


Figure 2.5: The AlexNet architecture which consists of the same structural features of LeNet-5 but is deeper and wider [37].

The concept of these two architecture is the same but Alex additionally introduced local response normalization which is not used so much anymore in convolutional neural networks. AlexNet also used dropout [42] at the fully connected layers to improve generalization. The performance of AlexNet in the ImageNet dataset was an inspiration and a strong motive for using convolutional neural networks during the next years. More specifically, AlexNet achieved 16.4% classification success rate which was a significant improvement over the techniques (neural networks or not) that had been used before that year.

Next year, Mathew Zeiler and Rob Fergue introduced ZFNet [22] which was more or less a fine-tuned version of the AlexNet and won the 2013 ImageNet challenge with a success rate of 11.2%. A major difference from AlexNet was that ZFNet was using 7×7 kernels instead of 11×11 at the first convolutional layer which not only reduced the model parameters but also increased performance. The success of ZFNet reinforced the belief that convolutional neural networks could take over the computer vision research and encouraged more efforts towards this direction. As expected, the following years were very important for the popularity of convolutional neural networks. In 2014, the Visual Geometry Group (VGG) was the runner up of the ImageNet challenge and proposed three versions of a network that still remain popular [43]. They basically proposed three models with increasing depth, namely VGG-11, VGG-16 and VGG-19. The first model consists of eight convolutional layers, the second of thirteen and the latter of sixteen convolutional layers. All models implement three fully connected layers as their last stages. While the computational cost increases with the increase of convolutional layers, the achieved accuracy also increases at the same time. VGG model also implement 3×3 kernels and a stride of two. The accuracy of VGG was 7.4% but had substantially more parameters than AlexNet (138M vs 60M). However, the winner of that year's challenge was GoggleNet [44] that incorporated a new architectural feature called the inception layer. Inception layer applies different-size kernels on the input and concatenates the results into a unified feature map. This strategy has some advantages like the implementation of various receptive fields that capture correlation patterns in the output map. Inception layers also contribute to the reduction of the model parameters to an extend that cannot be ignored since the GoogleNet only has 7M parameters. The proposed architectural feature comes in two flavors: the naïve and the one with a dimension reduction as shown in Figure 2.6. The accuracy of the specific model on the ImageNet challenge was 93.3%. While GoogleNet performs better and has less parameters than VGG, researchers frequently use VGG as a pre-trained model since

the former model adds complexity due to the inception layers and the use of several softmax layers at various depths that allow smoother gradient flow back to the initial layers of the model.



(a) Inception module, naïve version (b) Inception module with dimension reductions

Figure 2.6: GoogleNet's inception modules implemented (a) without dimensionality reduction and (b) with dimensionality reduction. Dimensionality reduction in (b) is an effect of using the 1×1 convolutions that shrink the depth of the previous layer's feature maps [44].

One of the latest convolutional neural network milestones is the residual network [45, 46] which won the contest in 2015. As the VGG models suggest, increasing the depth of the model usually comes with performance improvement provided that an appropriate training procedure is applied. Increasing the number of layers inevitably increases the probability that the vanishing gradients problem will appear. With the means to deal with such problems at hand, the deeper the convolutional networks the better the chance of increasing performance [47]. This principle was taken to the extreme by residual networks that implement ultra-deep convolutional networks. The number of layers in the proposed residual networks varies from 34, 50, 101 and 152 up to 1202. However, the most popular of these models is the one with 50 layers and is called ResNet50. It actually uses 49 convolutional layers and a fully connected layer at its output end. The way ResNets deal with the vanishing gradients problem is through the use of residual blocks shown in Figure 2.7.

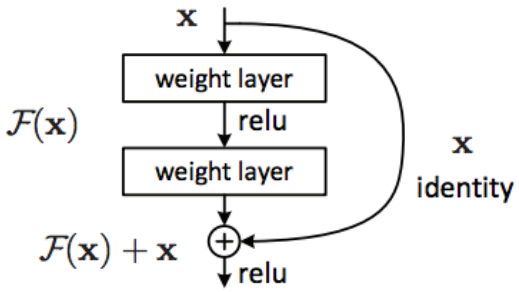


Figure 2.7: A residual block which is the fundamental building block of residual networks

According to the architecture of the residual block, there exists a skip connection from the input of the block to its pre-output level. This skip connection allows the gradients to flow better during training and provides some interesting properties to the model. A deep model can be interpreted as a shallow model plus some extra layers that are actually identity mapping layers. This means that their output just replicates the input. For example Figure 2.8 shows that the output of a shallower network denoted as x_l is provided to the next layers of the model but is also transferred to the output of these next layers, essentially skipping them altogether.

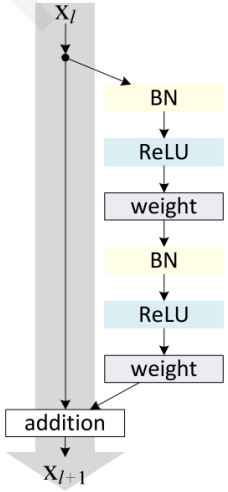


Figure 2.8: Residual networks can be rationalized as containing shallow networks and several extra layers that just implement the identity function. When necessary, residual networks use these extra layers to add extra computations to the model processing accordingly. The figure shows the full pre-activation application of Resnets with batch-normalization (BN) applied before the non-linearity (ReLU).

The two networks perform the same function and consequently are equivalent in terms of the computation they perform which is $y = F(x)$. In the worst case scenario the additional layers are not necessary to learn the function and are just implementing the identity function in order to pass over the output of the shallow network to the output of the model. On the other hand, there are situations that the extra layers contribute to the learning of the required task which could not be sufficiently approximated by the shallow network alone. The authors of the original residual network paper also hypothesize that it is easier to optimize the residual mapping function $F(x)$ than to optimize the original unreferenced mapping. The RsNet152 achieved 96.43% accuracy on the ImageNet dataset. It also won the COCO 2015 (Common Objects in Context) competition for detection and segmentation and ImageNet localization and detection competitions. It practically performed astonishingly on various tasks. Many variations of ResNets were proposed afterwards and some further studies revealed the reasons the specific architecture is so efficient. One of its variations introduced was InceptionV4 architecture [48] by Christian Szegedy. This variant implements the inception-residual unit. Recently an even better version of the inception-residual network was proposed by X. Zhang et al, the PolyNet [49]. On the search for the reasons that residual networks perform so well, there is a study by Andreas Veit et al [50] that proves experimentally that residual networks behave like ensembles of relatively shallower networks.

It is important to note that human performance on ImageNet is around 95%. Obviously, ResNet152 surpassed human baseline while GoogleNet was not that far away. Figure 2.9 [51] shows the success rates of some popular convolutional neural networks developed during the recent years on the ImageNet classification challenge. Some more recent model proposals add architectural modifications that improve network characteristics. For example Gao Huang et al [52] introduced Densely connected networks (DenseNet) implementing a technique that greatly reduces network parameters. More specifically, they condense the output feature maps of each layer to dense blocks and all layers have access to the features calculated by all previous layers. This enables feature reuse by concatenating the features of all previous layers into a single tensor. DenseNets achieve state-of-the-art accuracy on classification tasks with a reasonable number of model parameters.

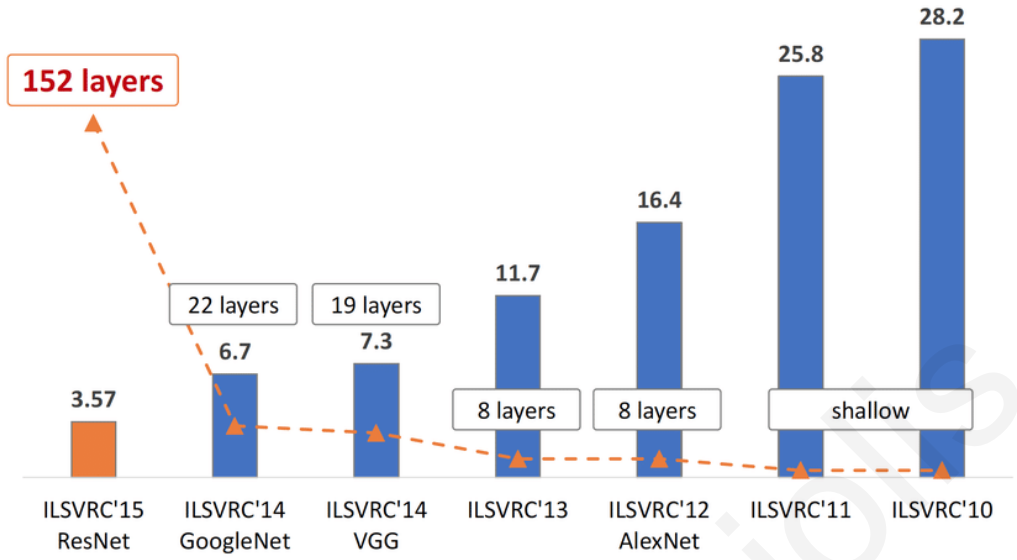


Figure 2.9: The evolution of winning entries on the ImageNet Large Scale Visual Recognition Challenge (ILSVRC) during the recent years. Depth of the winning models is also noted showing that improved performance has come with increased model depth [51].

Larsson et al [53] proposed FractalNets, a model architecture that empowers ultra-deep convolutional neural networks using a different approach than ResNets: they use many alternative feed forward paths and apply drop path as a means for training through efficient gradient flow and regularization.

The mentioned models are just a subset of the models proposed during the recent years and have been able to achieve remarkable performance. Convolutional neural networks and computational hardware are progressing along-side allowing for deeper and wider networks. Already fascinating commercial applications have been developed or are under development: self-driving cars [54, 55, 56, 57] , deep learning applications in health care [58, 59, 60, 61], Voice assistants like Siri, Microsoft Cortana, GoogleNow, Text translation from one language to the other, automatic image caption, data-driven predicting advertising etc. Recently Harvard University announced the implementation of deep learning in performing viscoelastic calculations for predicting earthquakes. Deep learning improved the specific calculation time by 50000% which makes the application of earthquake protecting measures feasible.

2.5 The limitations of Convolutional Neural Networks

Besides their impressive success in solving various tasks, convolutional neural networks are not the holy grail of artificial intelligence and machine learning. They are really good in learning the features of input data but they perform unexpectedly when the distribution of the data deviates from the distribution they were trained on. Convolutional neural networks are extremely vulnerable to input space transformations like scaling, rotation, flipping etc. But are also vulnerable to even tiny changes that are strategically chosen which proves that they face serious problems regarding their performance on inputs that do not obey the data distribution used to train them. The two basic forms of situations that really expose the convolutional networks' limitations are the adversarial patterns attacks and their lack of invariance to input transformations like scaling and rotation. Generally, machine learning models are vulnerable to various distortions that make the input depart even slightly from the training data distribution.

2.5.1 Adversarial patterns vulnerability

Szegedy et al [27] discovered a surprising and frustrating vulnerability experienced by numerous machine learning models: persistent misclassification of adversarial input patterns. Adversarial patterns are patterns produced by slightly varying training examples in a strategic manner such that the produced data seems extremely similar to the original data. These small variations are capable of causing extreme misclassifications on many different models independently of architecture or training algorithm used. This essentially means that adversarial patterns carefully crafted from a given dataset can fool, with high confidence, many models trained with the specific dataset. Figure 2.10 [27] shows some examples of adversarial patterns, the original data used to create them and the difference between the patterns. The adversarial patterns are in fact indistinguishable from their source but can cause misclassification with unexpectedly high confidence.

Figure 2.10: The original images on the left are correctly classified by a convolutional network as "school bus" and "dog". The adversarial images on the right column are strategically created from the original images by applying the difference shown in the middle column. While the crafted images seem indistinguishable from the original images to the human eye, they are surprisingly misclassified as "ostrich" with high confidence [27].

Szegedy et al also showed that highly "deceiving" adversarial examples can be produced using the L-BFGS (Low-Memory Broyden-Fletcher-Goldfarb-Sahnno) algorithm. The explanation for why these patterns laying in the lower probability areas of the data distribution are so good in fooling various models trained with the same data, is indeed very intriguing. This phenomenon becomes even more interesting by the fact that the 'fooled' models may be trained with different subsets of the dataset or trained with different algorithms or could possess entirely different architectures. It seems certain that adversarial examples' consistent misclassification reveals a flaw in machine learning algorithms that cannot be ignored. Szegedy et al also discovered that adversarial examples can effectively regularize a model but the application of such a strategy was not practical at the time because of the cost accompanying their calculation with the BFGS (Broyden-Fletcher-Goldfarb-Sahnno) algorithm. Nevertheless, this discovery gave rise to some misconceptions and some rather hastily formulated conclusions such as the theory of a problematic extremely-non-linear behavior of deep neural networks. As researchers studied the phenomenon extensively, it has been hypothesized that the opposite situation is more likely to occur: our models behave too linearly especially in extremely high dimensional spaces. Neural networks use activation functions that have a highly linear behavior like ReLUS and Maxout activations [62]. Even networks with non-linear activation functions, like the sigmoid activation, generalize well when these activations spend most of the training time in their linear

region. Goodfellow et al [63] introduced the fast gradient method for generating adversarial examples. They considered the activation change of a neuron having weights w to an adversarial pattern $\hat{x} = x + n$, with n being constrained by $\| n \| < \varepsilon$, where ε is small enough so that the pattern variation is stealthy to the data acquisition system sensitivity or to the computing system numerical precision. The new activation due to the adversarial pattern is $w^T \hat{x} = w^T x + w^T n$. The activation increase is expressed by the second term $w^T n$ and is maximized (according to the norm constraint) when $n = sign(w)$. According to this analysis, an effective method for creating an adversarial example is modifying a training pattern towards the direction of the gradient of the objective function $J(w, x, y)$ such as $n = \varepsilon \, sign\big(\nabla_x J(w, x, y)\big)$. Using the fast gradient method, adversarial examples can be produced much faster than the case of using the BFGS algorithm. An adversarial example produced by this method is shown in Figure 2.11.

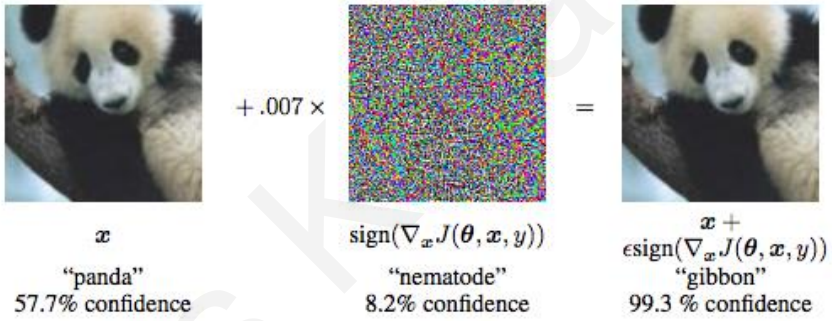


Figure 2.11: On the left, an image of a panda is correctly classified by a convolutional network with 57.7% confidence. Using the fast gradient method, the modification update in the middle image is calculated and used to produce the adversarial pattern on the right. The produced image is classified as "gibbon" with 99.3% confidence [63].

After Szegedy et al revealed the problem of adversarial attacks there was an increased interest in studying the specific problem and investigating for defenses against it. Papernot et al [64] conducted a study on the implications and limitations of deep learning in adversarial settings. It is evident that such a vulnerability is dangerous considering the fields of action of machine learning applications nowadays: autonomous self-driving cars [65] could be fooled into making devastating decisions. Malware classification systems [66, 67] could be evaded, medical applications could provide false diagnosis and security systems using biometrics or face

recognition could allow access to malevolent individuals. The list of unpleasant consequences is long and this could potentially result in restraining machine learning and artificial intelligence from handling tasks autonomously and without full-time human supervision. The study of the problem in the direction of hardening deep learning models against adversarial attacks promoted the understanding of the vulnerability and provided further analysis of the adversarial perturbations [68, 69]. The creation of adversaries was placed in a framework based on the initial studies of Szegedy and Goodfellow and is shown in Figure 2.12 [70].

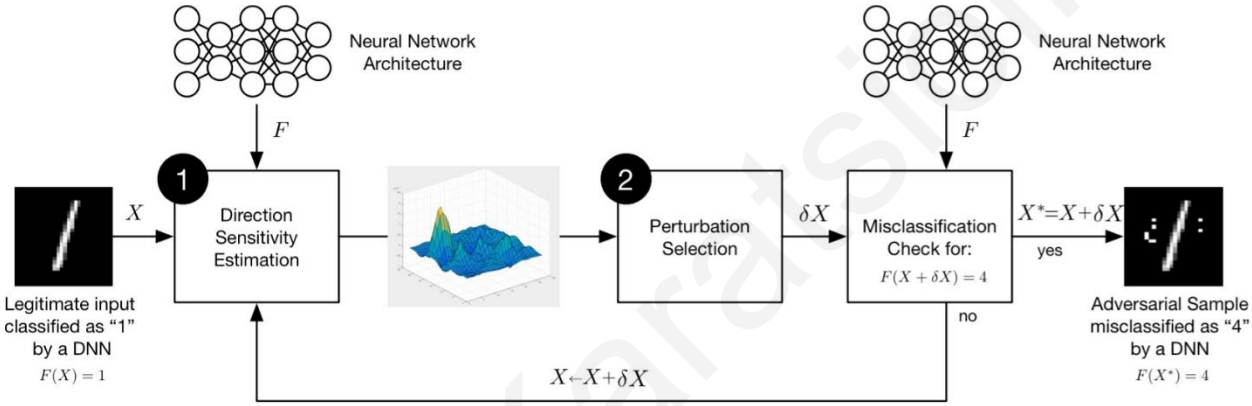


Figure 2.12: Adversarial crafting can be described in two distinct steps: (1) estimation of the direction of model sensitivity and (2) perturbation selection. The first step finds the direction that the input should change in order to make a deterministic step towards fooling the model and the second step applies this directional update. The original image is updated and then classified by the model to test the efficacy of the attack. Here, the newly created pattern is misclassified as "4" instead of "1" [70].

The goal of adversaries in the case of targeted attacks that aim a misclassification to a specific class c is to find the optimized solution for an adversarial update vector z^* on input x such as [71]

$$\underset{z^*}{maximize}\ f_c(x+z) \quad \text{subject to} \quad \|z\| \leq l\ . \tag{2.2}$$

This problem is practically broken to two parts: the objective is to find the correct value of z that maximizes the belief of the model that the adversarial pattern belongs to class c while the norm of the update vector is restrained to be less than a value l .

Another way to defend against adversarial attacks is called "defensive distillation" and was introduced by Papernot et al [70]. As the name suggests, it uses the concept of model distillation introduced by Hinton et al [72] which is a training procedure used to transfer knowledge from a complex high capacity model to a smaller manageable model. This way resource-constrained devices like smartphones or embedded processors that do not have powerful GPUs onboard can still run a small model which is trained on the knowledge of a large, accurate model. Distillation allows smaller networks to increase their accuracy, which in some cases is very close to the accuracy of the large model, while having much less parameters. The intuition of distillation was introduced by J. Ba and R.Caruana [73] while studying whether depth is important to neural networks. Hinton et al turned their intuition into a formal defined training procedure. The goal of distillation is to extract class probability vectors for the training examples using a high capacity model or an ensemble of networks. The extracted vectors are used to train a second smaller network. Effectively, the second network may be smaller but has access to more information than simple labels during training. The procedure enables the smaller network to take advantage of the knowledge provided by the larger model and achieve success rates that are near to the ones provided by the larger model. Hinton calls the knowledge gained from training the larger network "dark knowledge". This kind of knowledge contains additional information to the monolithically defined labels. In order to enrich the output of the larger model with useful information which reflects the patterns' probability distribution, the softmax function of the output layer is computed using a distillation temperature T during training. Given an input pattern x and N output logits such as Z_i , $i \in [0..N-1]$ the softmax function calculates the output probabilities as follows

$$f_i(x) = \frac{e^{z_i(x)/T}}{\sum_{k=0}^{N-1} e^{z_k(x)/T}} \tag{2.3}$$

Usually cross-entropy loss functions use $T = 1$. When higher values are used, the distribution of the output probabilities becomes closer to the uniform distribution. With $T \to \infty$ the softmax output is indeed a uniform distribution with $f_i(x) = \frac{1}{N}$. The lowest the temperature value the more discrete the probability distribution becomes and tends to assign a unit probability to one class and zero probabilities to all other classes. Usually the smaller network is trained using both kinds of information, the initial problem labels and the distilled output distribution. During training the smaller network uses the same high distillation temperature used by the larger

network. At test time the temperature is reduced to the normal unity value in order to allow for more discrete output probabilities.

Defensive distillation applies the distillation procedure on a model which is not smaller but has the same architecture with the one used to construct the soft labels. Papernot et al claim that training a network with this explicit relative information about classes, prevents the model from fitting too tightly to the data and contributes to a better generalization around training points. Table 2.1 shows their empirical results on adversarial patterns' success rates in terms of the distillation temperature used for obtaining the soft labels.

TABLE 2.1

Results for defensing neural networks against adversarial pattern attacks using distillation [70]. It is evident that while the distillation temperature increases, the success of the attacks is reduced.

Distillation Temperature	MNIST Adversarial Samples Success Rate (%)	CIFAR10 Adversarial Samples Success Rate (%)
1	91	92.78
2	82.23	87.67
5	24.67	67
10	6.78	47.56
20	1.34	18.23
30	1.44	13.23
40	0.45	9.34
50	1.45	6.23
100	0.45	5.11
No distillation	95.89	87.89

These results are interesting not just for hardening networks against adversarial attacks but also because they reveal that strong regularization may be the key to overcoming the limitations of neural networks. It also reveals that the way we train machine learning models using monolithic labels may not be the appropriate way since this kind of training targets omit a great deal of information.

Another recent and interesting work was conducted by Jiawen et al [71] and proves that deep neural networks are equally vulnerable to one-pixel-attacks. They practically used targeted attacks using differential evolution [74] to find adversarial patterns that are created by modifying only one pixel in the training patterns. They showed that more than 68% of the CIFAR10 (Canadian Institute for Advanced Research) test images and more than 41% of the ImageNet validation images can have one of their pixels modified and still be able to cause misclassifications. The implications of these results are significant and very concerning since they expose the vulnerabilities of convolutional neural networks in a great extent. These attacks were successful on various networks like VGG-16, Alexnet, Network in Network models and AllConv [75] models.

The adversarial patterns' attacks could be used to improve the performance of models by training such low probabilistic areas to be properly mapped in the learned feature space. This practically means that adversarial examples can be used to regularize a model during training. Goodfellow et al used a modified objective function to train a model on the MNIST (Modified National Institute of Standards and Technology) dataset. More specifically, each training pattern contributes to the training procedure in a two-fold way: with its information content intact and by introducing an extra piece of information which is a lower probability counterpart, namely its adversarial pattern. The contribution of the normal objective function minimization and the contribution of the adversarial version of the objective function minimization is regulated by the parameter a.

$$\hat{J}(w, x, y) = a\, J(w, x, y) + (1 - a)\, J(w, x + \varepsilon\, sign\big(\nabla_x J(w, x, y)\big), y) \qquad (2.4)$$

Goodfellow et al trained a well performing MNIST model without adversarial training that had an error rate of 89.4% on adversarial patterns. Using adversarial training the resulting model not only provided better performance on the test set (0.78 % error rate) but also reduced the adversarial examples misclassification to 17.9%. The performance of the model on the test set was slightly better than the performance reported by Srivastava et al (2014) for DBM (Deep Boltzmann Machine) fine-tuning with dropout [42]. Adversarial training is a very interesting approach because it provides a mechanism that regulates the model and facilitates the optimization of the objective at the same time. Through a series of experiments Goodfellow et al made a number of observations that reinforce the mystery around adversarial patterns:

- Models that are easy to optimize are also easy to perturb.
- Linear models lack the capacity to resist adversarial perturbation. Only models having at least one hidden layer should be trained with adversarial patterns. It is noted that the universal approximation theorem [76] applies on such models.
- Adversarial training regularizes the model to an extent that sometimes surpasses the effect of dropout [42]
- Ensemble models are not resistant to adversarial attacks
- Radial Basis Function networks are to some extend resistant to adversarial attacks

2.5.1.1 The intuition behind the success of adversarial pattern attacks

The adversarial patterns vulnerability reveals the need to better understand the inner mechanisms of neural network models. More specifically, the limitations of neural networks have to be thoroughly studied and recognized in order to overcome them with novel approaches and learning algorithms. The way features are combined to make a decision is efficient and adequate when data is drawn from the data distribution used for training the model. As soon as this stops being the case, models start making wrong and even strange decisions. The majority of the state-of-the-art performing models published are trained with heavy data augmentation and millions of data patterns to cover as much of the input domain data distribution as possible. It seems that parameterized models fit their parameters as better as possible in the span of the data distribution as it is experienced during training and totally neglect areas outside that span. The problem in real-life tasks is that small probability areas located at the boundaries of the domain distribution span still share some features with the well-defined data located at the high probability areas of the domain. Adversarial patterns are semantically very close to well-defined high probability patterns but according to the learned model are probabilistically distant. This makes it difficult for a model to discriminate between extremely similar patterns (semantically) that strategically differ in the way that the model interprets the high level features of the domain. One- pixel-attacks show that the problem is not in the low-level feature learning. The problem is in the way the model constructs the high-level features of the domain that provide the understanding of the domain through abstract concepts. These high-level features is the path to truly understanding the domain in an essential and holistic way.

2.5.2 Lack of input transformation invariance

It has been evident since the introduction of convolutional neural networks that a shift in the test image viewing angle may cause severe model misclassifications. In contrast to the adversarial perturbation cases, a change in the pose of the object in an image significantly changes the input content (in terms of pixel values) but this is generally easily detected by humans. Pose change, image flipping and viewing angle modifications alter the input to an extent that confuses the model's internal feature mapping mechanisms. Transferring a well performing model from the training environment to the production (implementation) environment may reveal that the model is useless if the implementation environment's data distribution is different from that of the training data. This data distribution shift may happen due to image acquisition system properties and characteristics or due to environmental conditions like different lighting conditions etc. Sometime this data distribution deviation may be small but its consequences may be devastating in terms of pursuing state-of-the-art performance especially for commercial tasks. This phenomenon is frustrating and interesting at the same time because systems that outperform human performance on specific tasks are indeed constrained to operate in the strict vicinity of the training data distribution in order to maintain their state-of-the-art performance. It seems rational to expect a better generalization ability and at least a primitive concept understanding from models that perform so well. For instance, convolutional neural networks are terrible in recognizing negative images [77]. Negative images are complementary images to the original ones contained in the training set. Such images can be constructed by inverting the light intensity of the original patterns. Dark original pixels are brighter and bright original pixels become darker. Assuming an original image's i^{th} pixel value in channel k such as $x_{i,k} \in [0,1]$ the corresponding pixel value in the negative image is $x'_{i,k} = 1 - x_{i,k}$. The specific transformation preserves the image structure, edges and semantics. Consequently, humans easily recognize negative images. On the contrary, convolutional neural networks greatly underperform when used to classify negative images corresponding to images that had been trained on. Figure 2.13 shows some examples of negative images and their misclassification while Table 2.2 shows the extent of the models' underperformance when used to classify negative images.

Figure 2.13: Examples of original images (top row) and their corresponding negative images (bottom row). Patterns are taken from the MNIST dataset, the GTSRB (German Traffic Sign Recognition Benchmark, both color and gray-scale versions) and the CIFAR10 datasets. After CNNs (Convolutional Neural Networks) are trained with the original patterns they are asked to classify the negative images with the misclassification results shown [77].

TABLE 2.2

Comparison of the accuracy results of various CNNs trained on the original images and tested on negative images [77].

Classifier	Dataset	Accuracy on regular images	Accuracy on negative images
LeNet−5	MNIST	99.21%	34.65%
LeNet−5	MNIST with data augmentation	99.25%	12.38%
MVGG−5	CIFAR-10	78.24%	38.55%
MVGG−6	CIFAR-10	80.86%	41.66%
MVGG−7	CIFAR-10	82.78%	45.51%
MVGG−8	CIFAR-10	83.17%	46.16%
MVGG−9	CIFAR-10	84.01%	47.88%
MVGG−8	GTSRB-color	98.54%	12.66%
MVGG−8	GTSRB-gray	98.12%	12.29%
Human	GTSRB-color	98.48%	97.31%
Human	GTSRB-gray	98.00%	96.41%

Hardt [78] claims that deep neural networks memorize the dataset which could explain this problem of convolutional neural networks. They actually trained deep models on random labels and proved that convolutional neural networks can even learn random assignments of no meaningful interpretation. Their claim triggered a reaction from several researchers that studied and rejected this claim. Arpit et al [79] showed that deep neural networks tend to learn simple patterns of the training set at first and then resolve to memorization. This fact proves that the optimization process is content aware. In other words, besides the fact that models have the increased capacity to memorize the data, they first learn patterns that are easier to learn most probably because this is the easy thing to do. Krueger et al [80] also used some empirical methods to prove that deep networks don't learn via memorization. They claimed that learning real datasets and learning random noise are fundamentally two different processes. Learning noise requires more model capacity and convergence time is longer. Real datasets tend to make the models calculate simpler features a fact which they verified by calculating the sharpness of the loss function in the vicinity of convergence. Lastly, regularization techniques have a different effect on learning real datasets than the cases of learning noise. For example dropout when applied at the fully connected layers of the models reduced the learning ability on the random noise case but definitely helps when using real data for training the models. This argument implies that memorization may be used by neural network models when no structure is evident in the training set and without any other means available towards learning the data, memorization is the only strategy left to apply. When training data is characterized with information structures and thus a rational feature space is feasible to construct, convolutional neural networks map the data structure to meaningful domain features.

Deep neural networks possess the capacity to approximate a huge set of functions and an extensive number of model families capable of explaining complex data. Stacking layers of artificial neurons is a way to construct a complex mathematical function for computing an output value (or values) based on the input data. A neural network is nothing more or nothing less than a mathematical expression whose variables take values from the input layer. Under this simplistic point of view it is not surprising that deep neural networks have the capacity to fit a random function. This is actually anticipated by the long-known fact that neural networks can easily over-fit a dataset. In essence, over-fitting a dataset and memorizing a random function are two sides of the same coin in the sense that the same theoretical principles and mechanisms drive these two outcomes. The important fact is that neural networks as we train and apply them

have some important flaws that must be overcome in order to model human concept understanding.

To mitigate the problem, deep neural networks researchers use an approach that contributes in regularizing the model and adding small transformation invariance: data augmentation. Almost all state-of-the-art models that were trained without the luxury of having an extremely large dataset available have been trained with heavy data augmentation. For each example in the training set, numerous additional patterns are created by slightly altering the original pattern through the application of transformations. For images these transformations are affine transformations. Performing pattern transformations is fundamentally different from adding noise to the pattern during training. Training with input noise enables manifold learning [81]: the model learns lower-dimensional structures that are composed of simple features. Because of their reduced dimensions, similar data is mapped in the vicinity of the manifold. Training with data augmentation besides adding much more artificial information to the optimization procedure, it also makes the optimizer aware of spatial transformations beyond the ones already present in the input. In other words, the model becomes, to some extent, invariant to transformations. Still, data augmentation cannot help convolutional neural networks to explore relationships regarding the data content. Fortunately, new architectures have been proposed towards that direction.

2.5.3 Capsule Networks

Hinton gave a speech in 2014 at MIT University [82] and talked about what he believes is wrong with convolutional neural networks. He specifically blamed back propagation for many drawbacks. He also claimed that current learning techniques and the max-pooling operation are the wrong way of adding translation invariance to the models. He also stressed the need for alternative training algorithms and his belief that structures representing the instantiation parameters of a specific type of entity (for example an object part) are a promising research direction for solving the convolutional neural networks lack of input invariance. He called such structures as “Capsules” and some years later proposed a training procedure that applied dynamic routing between such structures [40]. Capsule Networks intend to solve fundamental problems present in current convolutional neural networks architectures. For example, most convolutional neural networks could be fooled into predicting the presence of a human face if

the input image consists of facial objects (eyes, mouth, ears etc.) positioned randomly on the input canvas. For example Figure 2.14 shows a Picasso painting where the different facial objects are artistically placed in the image but the presence of each individual subpart suggests the existence of a human face.

Figure 2.14: Picasso's portrait of a woman (Olga). The surrealistic representation of the subject's different facial parts could fool a convolutional neural network into predicting the existence of a human face

The Picasso painting scenario is just an illustrative way to stress a major flaw of convolutional neural networks: they are really good in detecting features of object components but they combine them in a naïve way to make a decision. They behave like their only task is to detect the presence of characteristic patterns in the input data that are shared by training examples of the same class and that presence by itself is enough to make a prediction. This means that they totally neglect the spatial relationship between the different object parts. Hinton claims that capsules remove the naively performed combination of features and replace it with spatial relationship representations like orientation, scaling, perspective etc. Figure 2.15 shows the architecture of the Capsules network which is comprised of four parts: the convolutional layers, the primary capsules with the linear transformation matrix $W_{i,j}$, the digital capsules that reflect the representation of the input and the decoder structure.

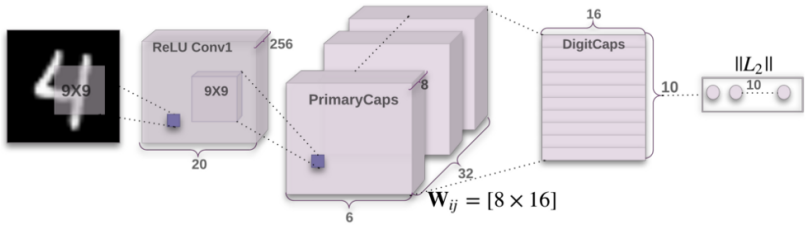


Figure 2.15: Capsules networks consist of the convolutional layers, the primary and digital capsules and the decoder component that reconstructs the input image from the digital capsules [40].

The effectiveness of capsules is mainly due to two important novel architectural differences. The first innovation introduced by Hinton et al is the vector processing activation functions performed by the capsules. Instead of scalar activation functions, they use neurons that take vectors as inputs and output a vector. They call this activation function, the squashing function and its properties are compared with the traditional scalar activation functions in Figure 2.16.

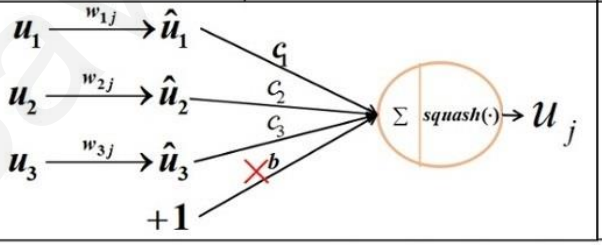
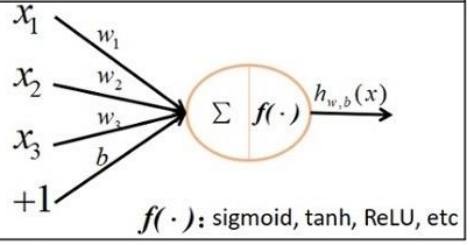
		capsule	traditional neuron
Input from low-level neuron/capsule		vector(u_i)	scalar(x_i)
Operation	Affine Transformation	$\hat{u}_{j\|i} = W_{ij} u_i$	–
	Weighting	$s_j = \sum_i c_{ij} \hat{u}_{j\|i}$	$a_j = \sum_{i=1}^{3} W_i x_i + b$
	Sum		
	Non-linearity activation fun	$v_j = \frac{\|s_j\|^2}{1+\|s_j\|^2} \frac{s_j}{\|s_j\|}$	$h_{w,b}(x) = f(a_j)$
output		vector(v_i)	scalar(h)
			

Figure 2.16: Comparison of a capsule neuron with neurons using traditional activation functions. The most important difference are the vectored inputs and output of the capsule that allows it to process more complex and information-rich input [40].

The second important factor that enables capsule networks to encode spatial relationships is the actual dynamic routing algorithm responsible of deciding the extend of the contribution that each primary capsule will make to the encoding vector maintained by the digital capsules at the last layer before the decoder of the model. This algorithm is iterative and updates the contribution magnitude of each capsule based on the agreement of the information it holds with the information of the other capsules. For example, if a capsule represents the nose in the Picasso painting in Figure 2.14 must comply in terms of spatial properties (orientation, size, distance) with the information represented by the other capsules. If a nose is detected in the image but is not located between two eyes, or above a mouth or a little lower and between two hears or at least a partial relation of the above holds (due to face orientation or another object obstructing the view), the model maintains some skepticism on the presence of a face. If other objects with good spatial relationships are present it might be the case that the model is reluctant in making a classification decision based on the presence of a face.

Chapter 3

Unsupervised Learning

Unsupervised learning is one of the major forms of Machine Learning, the other being supervised learning. Their major difference is the fact that unsupervised learning lacks any kind of information regarding the properties of the training data. In most of the cases, this information involves the category of the input patterns thus encapsulating the requirements of the desired classification task. Unsupervised learning usually experiences only the data and aims in discovering some useful properties of it. Ideally, a useful discovered property is the probability distribution itself, which in turn describes the function that generates the data. The distribution of the data $p(x)$ may be calculated explicitly (density estimation) or implicitly. In practice most unsupervised learning algorithms try to learn the joint probability $p(x, z)$ where z is a latent random variable. On the contrary supervised learning is about learning the probability of the data labels conditioned on the input data $p(y|x)$. Assuming that an unsupervised learning algorithm models the joint probability distribution $p(x, y)$ then

$$p(y|x) = \frac{p(x,y)}{p(x)} \tag{3.1}$$

Using the sum rule of probabilities gives

$$p(y|x) = \frac{p(x,y)}{\sum_{y'} p(x,y')} \tag{3.2}$$

The latter equation shows that unsupervised and supervised learning are interconnected concepts in the sense that each learning regime can be transferred (at least theoretically) to the other learning form. Of course, the calculation of the marginal probability $p(x, y)$ is not easy and in practice is intractable for most of the real life problems.

Besides density estimation, unsupervised learning may extract data information by means of denoising data, calculating a data manifold and clustering data into groups based on a distance metric which reflects patterns similarity. Other algorithms like t-SNE [23] focus on visualization of data by greatly reducing the dimensionality of data while preserving the spatial interrelations. Principal component analysis (PCA) [178] is an analytic approach for calculating the independent axes of maximum data variation. While PCA is a fast, informative and sometimes a quite effective pre-processing step, it has the disadvantage of being a linear process which means that it acts only on the linear relations of the data. Detecting non-linear dependencies is crucial for most practical problems. Another popular approach to unsupervised learning is the k-means algorithm proposed by MacQueen in 1967 for clustering data that is close to each other mostly in terms of Euclidian space. Of course there is a plethora of unsupervised learning algorithms based on neural network architectures proposed through the years.

3.1 Autoencoders

Auto-encoder is a special type of unsupervised learning and is comprised of a function that attempts to copy the input to its output. Of course this could be accomplished trivially by learning the identity function $I(x) = x$ which is of no use. To avoid this undesired outcome, auto-encoders are usually built or trained under some constraints in order to force the model into learning useful representations. In other words, auto-encoders are restrained from learning the input perfectly but are driven towards learning its underlying data concepts. Auto-encoders use two deterministic mappings from input data to the actual output. The first mapping transforms the input to a latent distribution $p(h|x)$ and is called the encoder of the model. The second mapping $p(x|h)$ transforms the latent representation to an output r which of course should be as close to the input as possible. Figure 3.1 shows the general structure of an auto-encoder.

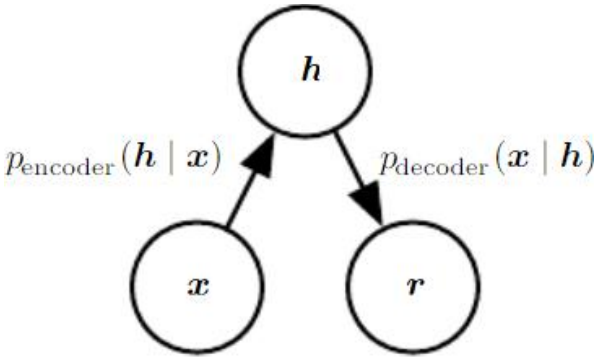


Figure 3.1: The general structure of an auto-encoder. The encoder maps the input x to a latent distribution h and the decoder outputs r in terms of the latent distribution as a reconstruction of the input.

To force the auto-encoder into learning some useful data properties instead of performing the identity mapping, two major techniques are generally used: under-completeness and regularization. Under-completeness is effectively an input dimensionality reduction approach since the latent code is designed to be smaller than the input data size. This by definition makes it impossible for the model to simply copy the input to the output and forces it into learning low-dimensional representations of the input patterns. Such a scenario greatly resembles principal component analysis and in fact an auto-encoder with linear layers (encoding and decoding) has a representation ability equivalent to PCA provided the mean square error (MSE) is used as the loss function. Of course, auto-encoders with non-linear layers are more powerful than PCA. Under-complete architectures allow auto-encoders to perform meaningfully in a natural way: given their reduced latent distribution dimensionality, the auto-encoders must detect the best, more expressive and informative features to encode in a limited-capacity feature pool in order to be able to reconstruct the original patterns. In practice, under-complete auto-encoders are mostly used for cases that require their main characteristic, that is, the dimensionality reduction of the data.

Most practical applications of auto-encoders use some means of regularization in order for the auto-encoder to learn meaningful representations. This approach avoids the requirement of using smaller than the input latent codes and thus does not reduce the representation ability of the model. Regularized auto-encoders use a loss function that incorporates additional objectives besides the one concerning the reconstruction of the input. These objectives require some extra

model characteristics which in turn increases the quality of the auto-encoder. Well-established autoencoder regularization objectives are representation sparsity [96], small magnitude representation derivatives, denoising the input and reconstruction of partially observed input (input missing a portion of its attributes). It should be noted that auto-encoders' regularization is fundamentally different from the concept of model regularization as broadly used in Machine Learning for applying a prior over the model parameters. For example L_2 norm applies a prior preference for small weights while L_1 norm is itself a prior preference for sparse weights. Auto-encoders regularization does not enforce a prior on the model's parameters but a prior for preferring specific model functions over others. The preferred model functions have some specific properties that are compelling for an auto-encoder to possess.

3.1.1 Sparse Auto-encoders

A sparse auto-encoder uses a sparsity penalty $\Omega(h)$ on the representation layer. This term penalizes the representations that consist of latent codes that have many (or all) their variables active. Having many of the representation elements being very small or zero, prevents identity mapping and enables the auto-encoder to feature map the input in a meaningful manner by encoding its most significant characteristics. In some sense, sparse auto-encoders are dynamic under-determined models because they push the model towards not using a significant part of its capacity but it does this in a more gentle and no-strict manner. The loss function of a sparse auto-encoder is $L\left(x, g\left(f(x)\right)\right) + \Omega(h)$ with $f(x) = h$ being the encoding function, h the latent representation and $g(.)$ the decoding function. One way to achieve representation sparsity is to use L_1 penalty on the representation values. Penalizing the absolute values of the latent distribution effectively applies a prior of having many near zero-valued (inactive) neurons. Consequently, many neurons of the internal representation layer are driven inactive, implicitly making the model under-complete. Another way to promote sparse representations is the use of Kullback-Leibler divergence penalty between the fraction of activated representation neurons and a small constant corresponding to a desirable fraction of active neurons.

3.1.2 Contractive auto-encoders

Another way of regularizing the representation code of an auto-encoder is to introduce a penalty on the norms of the partial derivatives of the encoder. Practically, this penalty is the squared Frobenious norm of the encoder's Jacobian matrix such as $\Omega(x) = \lambda \left\| \frac{\partial f(x)}{\partial x} \right\|_F^2$. By applying a contractive regularizer, the auto-encoder is driven into calculating encodings that do not change much due to small input changes. The name contractive comes from the property of such auto-encoders to map input regions to smaller output regions which produces a space contraction or a space wrapping. The effect of this space contraction phenomenon is learning the data manifold, which is a lower-dimensional structure that describes the data and the way it changes. Rifai et al [97] studied contractive auto-encoders and their internal mechanisms by examining the singular values of the encoder's Jacobian and their effect on the calculated encoding of the input. Most of these singular values have contractive magnitudes that are smaller than unity and only a few of them actually have a magnitude larger than unity. The singular values with the largest magnitudes seem to be approximations of the tangent planes of the manifold and consequently correspond to data-content variation. Rifai et al also used an image of a dog drawn from the CIFAR10 dataset to demonstrate how PCA and contractive auto-encoders calculate tangent vectors of the data manifold. Figure 3.2 shows a representative case of their experimental outcomes: contractive auto-encoders capture better approximations of the local target vectors that actually correspond to input image semantic variations like an object having different components and parts. While contractive auto-encoders provide some attractive properties and an intriguing theoretical foundation, their implementation could be tricky. For example, deep contractive auto-encoders have a troubling computational cost due to the calculation of the Jacobian of the latent representation. Even more troubling is the fact that many trivial functions satisfy the requirements of the objective function (reconstruction plus the regularization term).

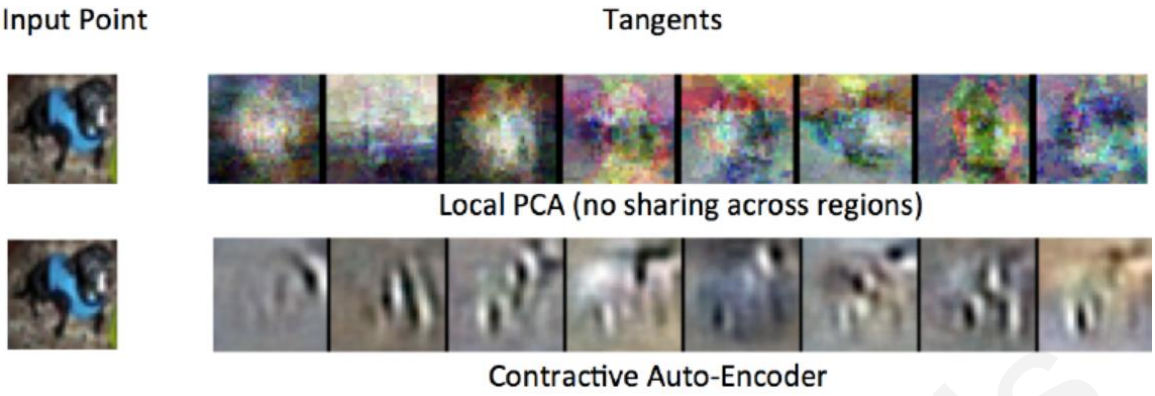


Figure 3.2: Illustrations of the tangent vectors of the manifold estimated by a PCA and a CAE (Contractive AutoEncoder) based on the input image of a dog taken from the CIFAR10 dataset. The tangents correspond to singular points in the Jacobian of the representation $\frac{\partial h}{\partial x}$. The PCA cannot capture good tangent vectors while the CAE captures tangents that correspond to changing various parts of the dog's body [97].

Naturally, these trivial functions are meaningless since they do not describe the data distribution nor they hold any info of any use. To prevent this, one could tie the weights of the decoder and the encoder by setting the decoder weights to be the transposed weight matrix of the encoder. Still, these disadvantages have established the contractive auto-encoder to be of more theoretical value than of practical use.

3.1.3 Denoising Autoencoders

Denoising Autoencoders (DAE) are perhaps the most popular of the auto-encoders because of their theoretical soundness and the ease of its implementation. They are also less expensive to optimize because they do not require an additional regularization term estimation like calculating Jacobians or implementing sparsity terms. On the contrary, they apply a simple and effective way for reconstructing the input pattern while learning its structure. Denoising autoencoders are trained using noise-corrupted data and learn to output the original pattern without the corruption. The corrupting process is a distribution of the noisy data $\hat{x}$ conditioned on the original data x and is denoted by $c(\hat{x}|x)$. The denoising autoencoder learns the reconstruction distribution $p(x|\hat{x})$ by the use of training pairs of the form $(x, \hat{x})$. Like other regularized auto-encoders, denoising autoencoders use a specific technique in order to maintain

high-capacity and at the same time avoid learning useless functions like the identity function. Denoising the input is not the primary task of the denoising autoencoders. This objective has an indirect effect on the learning process by enabling the model to learn good internal representations. This side effect of the denoising task in multilayer feed forward networks was strongly stressed by Vincent [98, 99]. Earlier, LeCun (1987) and Gallinari et al (1987) used neural networks primarily for denoising tasks. Denoising autoencoders use the following basic procedure:

1. Sample x from the training set such as $x \sim p(x)$
2. Corrupt the input x using $c(\hat{x}|x)$
3. Reconstruct the original input x by estimating $p(x|\hat{x})$. This is accomplished by learning two distributions: the encoding $f(\hat{x}) = p(h|\hat{x})$ where h is the internal representation and $p(x|h) = g(f(\hat{x}))$

Training a denoising autoencoder can be done by expressing the objective function in terms of maximizing the log-likelihood of the reconstruction $\log_2 p(x|h)$ or (equivalently) minimizing its negative value. While this objective function is sufficient for training the denoising autoencoder, it does not provide the insight understanding provided by an alternative approach: score matching. Generally, score matching [100] is considered as an alternative to maximum likelihood. Provided our need to estimate a probability distribution for the given training data without having an analytic approach, we can use an estimator as a distance indication of our model and the actual distribution. This estimator is calculated for every training pattern. Its summarized similarity to a score of the actual data, indicates the quality of the model in approximating the original data distribution. This estimator is the data gradient field $\nabla_x \log_2 p(x)$. By learning this field, the autoencoder implicitly learns the data distribution. Denoising autoencoders can specifically estimate the score of the data distribution given by the vector field $g\big(f(x)\big) - x$ provided they apply Gaussian noise corruption [101] such as $c(\hat{x} \mid x) = \mathcal{N}(\hat{x},\ \mu = x,\ \Sigma = \sigma^2 \mathrm{I})$, σ^2 being the noise variance and then use the squared error loss function $\|g(f(\hat{x})) - x)\|^2$. Alain and Bengio [101] provided an analysis of how this vector-field-learning arms the denoising autoencoders with data reconstruction capability as shown in Figure 3.3. The data manifold is represented by the curve-inducing circles and the 2-D area around that curve contains vector fields pointing to the nearest point on the manifold. The further each point is from the manifold, the larger the magnitude of the arrow is.

The magnitude is proportional to the difference of the reconstruction and the input and represents the value of the estimator $g(f(x)) - x$. The magnitude is also inversely proportional to the probability of the point being generated by the data distribution $p(x)$. This explains why the vector fields have a very small magnitude in the vicinity of the manifold and a (theoretically) zero value on it.

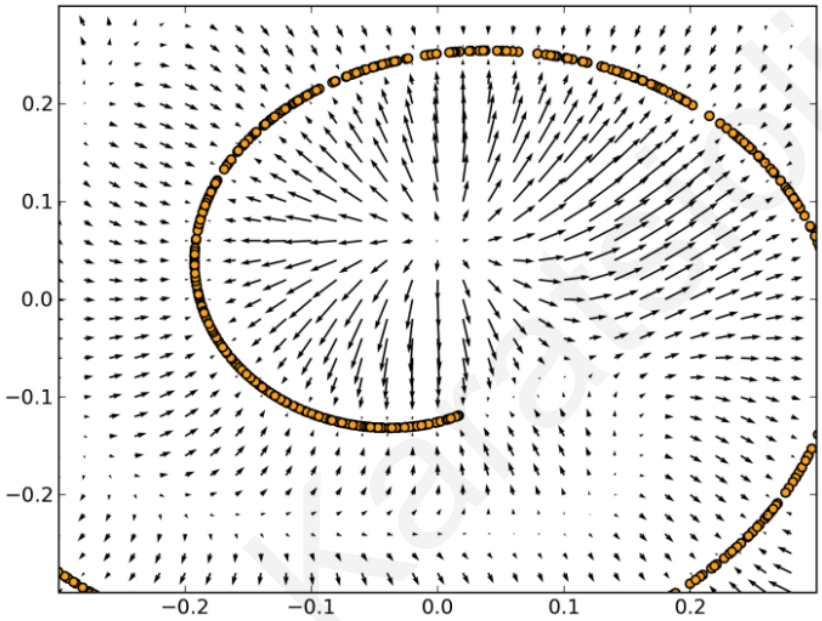


Figure 3.3: Denoising autoencoders learn the manifold of the data. Each mapping point is attracted to this manifold by a magnitude that is equal to the vector field provided by score matching and has a value of $g(f(x)) - x$. A point far from the manifold has a larger vector field since it results to a reconstruction that is not close to the original data. On the manifold surface, the vector fields have a zero value [101].

In this context, the corrupting process $c(\hat{x}|x)$ is relocating a data point x from its manifold to a point $\hat{x}$ in the vicinity of the manifold as shown in Figure 3.4. According to this figure the manifold is one-dimensional for visualization purposes and the relocation due to the corrupting process results to a new data point located in the circle shown. The denoising autoencoder learns how to move this data point back on the manifold because the vector field which the model is trained to minimize, is a vector pointing to the nearest point on the manifold. This vector field corresponds to the score $\nabla_x \log_2 p(x)$ up to a multiplicative factor which in turn corresponds to the average root mean square error of the auto-encoder during training.

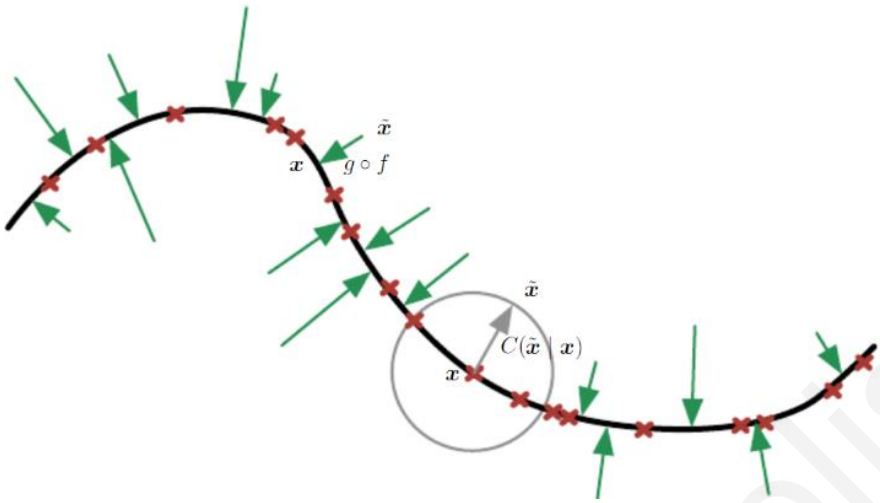


Figure 3.4: A denoising autoencoder learns to map a corrupted data point $\hat{x}$ back to the original data x . The black curve represents the data manifold and added noise moves the corrupted data away of it. The model returns this point on the manifold since the corresponding vector field is minimized during training. The corresponding vector field is also a vector pointing towards the nearest manifold point and is equal to the data reconstruction loss [101].

3.1.4 Applications of Autoencoders

During the first few years of this century (2000-2007), before the development of Variational Autoencoders (VAEs) and Generative Adversarial Networks (GANs), there was a general belief that Autoencoders were a very promising building structure for machine learning models. Their simplicity and effectiveness raised the expectation that in the near future they could provide the stepping stone for building effective complex models. The development of Restricted Boltzmann Machines (RBMs) by Hinton et al [102] and the popularity of deep Belief Networks [103] contributed to the general feeling that representation learning using architectures of embedded simple structures would turn to be very effective. Unfortunately, some years later, Autoencoders gave up their primacy to other unsupervised learning models that use different approaches for extracting information in the absence of data labels. However, Autoencoders provided some very important results in machine learning that formed the roadmap towards deep learning and provided the foundations of unsupervised learning as we experience it today. It is no surprise that research interest for Autoencoders in general is escalating through the years as shown in Figure 3.5 [104].The consistent interest in Autoencoders through the years proves that the model is a very useful tool in the machine learning tool case.

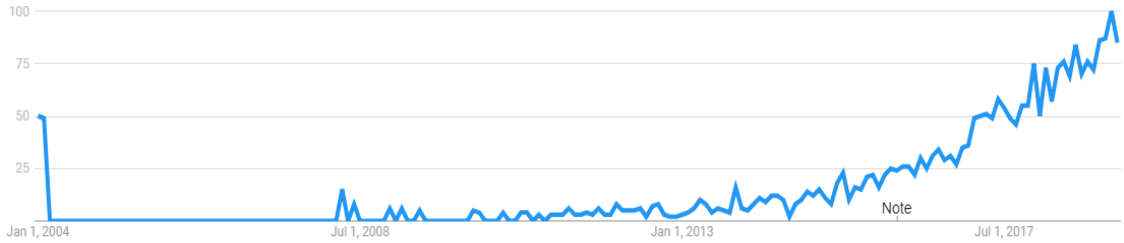


Figure 3.5: Interest over time for Autoencoders as provided by Google Trends from year 2004 until today. The y-axis shows the relative interest of the term "AutoEncoders" in the World Wide Web in terms of the highest point in the graph. This means that the interest in Autoencoders has a general trend upwards and peaked around last year. This fact is really interesting and suggests that there may be some hidden caliber in the specific modelling paradigm that has not exploited to its full extent yet.

Interestingly, Denoising AutoEncoders (DAEs) also maintain the interest of the research community through the years as shown in Figure 3.6 [104].

Figure 3.6: Interest over time for Denoising Autoencoders as provided by Google Trends from year 2004 until today.

The most profound application of Autoencoders is dimensionality reduction. High dimensional data can be encoded to a code described by much less dimensions. This process resembles the operation of Principal Component Analysis (PCA) [105] with the difference that Autoencoders are non-linear models while PCA discovers only linearly changing factors of the data. Hinton and Salakhutinov [106] used a stack of RBMs of decreasing size until a bottleneck layer of size 30 was reached. This layer was providing the code of the data in a compressed data representation form. Dimensionality reduction serves many tasks like classification, data visualization, clustering and information retrieval. Information retrieval can be applied

effectively using semantic hashing which is depicted after a dimensionality reduction and 'binarization' of the extracted code. A model using semantic hashing for retrieving the entries in a textual database that have the same code was developed by Hinton and Salakhutinov [107]. A similar model for retrieving images was published by Krizhevsky and Hinton [108]. Besides retrieving data that can be encoded to the same code, such systems can retrieve data that has similar codes by just flipping some bits of the code. The retrieved data is semantically similar to the data providing the initial code.

The capabilities of Autoencoders were revealed a few years before deep learning took over the Machine Learning society to a new era. During those years it was very difficult for deep models to perform well because of the lack of the theoretical principles that could help overcome basic problems arising from using many layers of neurons in a model. The most catastrophic problems were gradient vanishing (or exploding), the absence of powerful hardware (GPUs) and most importantly the lack of appropriate ways of weights' initialization which prevented efficient gradient flowing. Without efficient gradient flowing the training of deep models is at least very difficult [109, 110]. The solution was provided by using stacked single-layer Autoencoders, trained individually at each successive addition. Each layer learns the most important aspects (features) of the previous layer output and this allows the application of an increased number of layers which in turn forms a deep network. The whole process practically allows the application of a good initial network configuration that enables the successful training of the whole unified model. The approach was one of the first applications of deep networks in Machine Learning without requiring specialized architectures like convolutional or recurrent layers and is called Greedy Layer-wise unsupervised pre-training. Early deep learning methodologies and techniques developed partly due to the progress made by using Greedy Layer-wise unsupervised pre-training revealed that initializing the deep model with unsupervised learning is not always beneficial. Ma et al [111] found that unsupervised pre-training for chemical activity prediction resulted in slightly worse results while it was significantly beneficial for other tasks. The most helpful insights on how unsupervised pre-training affects the training of a model was provided by Erhan et al [112] who studied how the function spaces of many neural networks change during training time and whether this depends if the network went through pre-training or not. The study revealed that there are significant differences in the way neural networks construct their internal functions depending on the occurrence of an unsupervised pre-training. They proved that by constructing 2D visualizations of these functions using t-SNE (T-distributed

Stochastic Neighbor Embedding) [23] and ISOMAP (Isometric Mapping) [113] which are data dimensionality reduction algorithms basing their functionality on data similarity. ISOMAP is a simple method for estimating the intrinsic geometry of a data manifold based on a rough estimate of each data point's neighbors on the manifold. T-SNE models each high-dimensional object by a two- or three-dimensional point in such a way that similar objects are modeled by nearby points and dissimilar objects are modeled by distant points with high probability. The differences were more emphatic in the case of ISOMAP visualization. Figure 3.7 shows the visualization using the ISOMAP algorithm. It is evident that network functions start at different spaces depending on whether the network experienced a pre-training or not. In the networks without pre-training, the functions follow a more diverse trajectory while the network functions in the pre-trained case are much more consistent in their trajectory and their convergence space. The lighter color in the figure represents later training time (iteration) which means that functions of networks without pre-training end up to more diverged local minimums. Evidentially, unsupervised pre-training acts as an agent of regularization restricting the convergence function space to a region that conforms to the unsupervised learning criterion. However, the applied regularization is very different from the one enforced by L_2 norm regularization (weight decay) or L_1 norm regularization. The latter regularization methods benefit simple solutions by penalizing functions' complexity through restraining the weights' values while the former benefits network functions that conform to the unsupervised learning task.

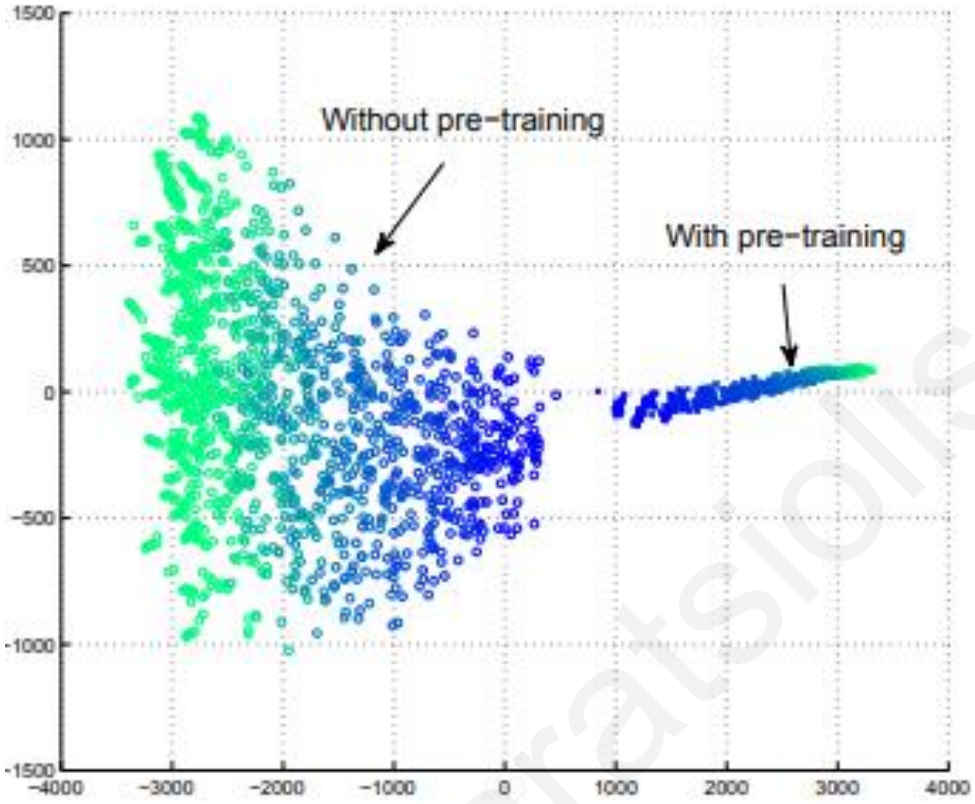


Figure 3.7: The function-learning trajectories in different networks pre-trained on the unsupervised learning task and networks without the pre-training. The marker changes to a lighter color depending on the training iteration which means that cyan colored markers constitute functions near local minimums. The network functions without pre-training follow more diverse trajectories with convergence points that are much different compared to the networks that do experience unsupervised pre-training [112].

3.2 Variational Autoencoders

Based on the variational inference framework, variational autoencoders [83] apply a prior $P(z)$ on the latent variable Z and maximize the variational lower bound of the model. Usually this prior is an isotropic Gaussian distribution $\mathcal{N}(0, \sigma I)$ with σ being the scalar variance and I the identity matrix. This approach solves the problem of the intractable posterior distribution $P_\theta(Z|X)$ by using an inference step that maps the input to an approximate posterior $Q_\theta(Z|X)$ which is a multivariate Gaussian and its parameters are learned from the data. Figure 3.8 shows the architecture of a variational autoencoder. The log data likelihood of the variational autoencoder is given by

$$logP(X) = E_{z\sim Q_\theta(Z|X)}[\log_2 P_\varphi(X|Z)] - D_{KL}(Q_\theta(Z|X) \parallel P(Z)) + D_{KL}(Q_\theta(Z|X) \parallel P_\theta(Z|X)) \quad (3.3)$$

The first term is estimated by the decoder network through sampling. The second term is the KL (Kullback-Leibler) divergence of the Gaussian modelling assumption and is minimized by learning the Gaussian parameters of the approximate posterior. The last term is always equal or greater to zero and thus the equation provides a lower likelihood bound. Jimwoog et al [84] propose the injection of noise to the input of the variational autoencoder accompanied with a modified training criterion. This approach combines the qualities of the original model and some additional ones provided from using a denoising process.

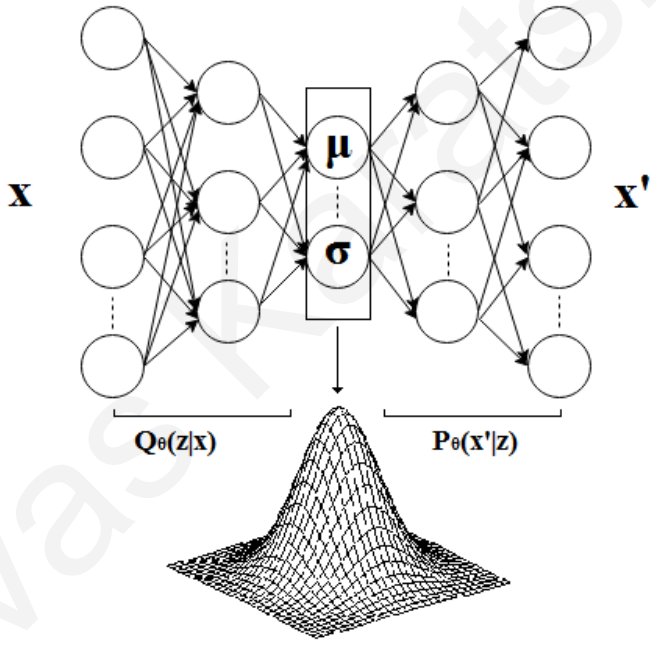


Figure 3.8: The architecture of a variational autoencoder. The encoder part of the model performs the inference step by mapping the input on a multivariate Gaussian distribution fully defined by its parameters μ, σ. The generative step implemented by the decoder of the model reconstructs a sample from the data distribution using the latent code [83].

Using an isotropic Gaussian as the latent code distribution allows sampling of the model to be performed by simply sampling a latent code such as $z\sim N(0,1)$ and feeding it to the decoder structure.

The variational autoencoder performs well and is theoretically elegant. These qualities establish it as a frequent choice for building generative models. However, it suffers from two major drawbacks: its samples tend to be blurry and it does not utilize the latent space effectively in the sense that it tends to use only a fraction of the available latent dimensions (Goodfellow 2015). The blurriness in the generated samples of a variational autoencoder may be caused by the fact that the model minimizes $D_{KL}(Q_\theta(Z|X) \parallel P(Z))$ in order to perform maximum likelihood. This issue is also applicable to generative models that use log-likelihood optimization. The use of a Gaussian distribution for $P_\varphi(X|Z)$ also contributes to the problem as shown in Figure 3.9. Several attempts have been made in order to improve its performance. Hou et al [85] replace the pixel-wise loss of a variational autoencoder with a feature consistency loss. More specifically, they forward pass the original and the generated image through a VGG-19 network [43]. Then they evaluate the similarity of the intermediate layers feature mappings produced by the two images. This feature-perceptual-loss is measured in squared Euclidean distance and comprises the new loss used for training their deep feature consistent VAE. This approached produces two models that are based on the loss applied on different intermediate VGG layers: VAE123 applies the loss on the first 3 ReLU layers and VAE345 applies the loss on ReLU layers 3-5.

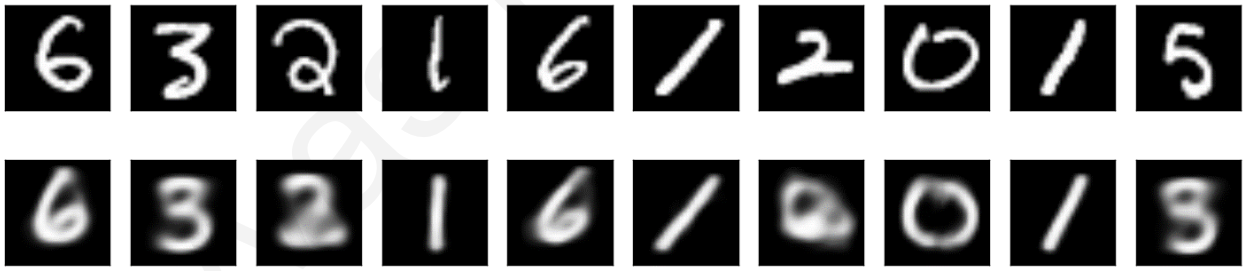


Figure 3.9: Reconstruction of MNIST digits through a variational autoencoder. It is evident that images tend to be a little blurry partly because the model applies log-likelihood optimization and partly because it relies on a Gaussian distribution to define the latent space of the problem. Using a smooth, continuous distribution inevitably enforces some ambiguity at the edges of the latent space clusters. For example numbers '5' and '3' on the last column are not clearly distinguishable because of the smooth transition from the latent space definition-area of number '5' to the latent space cluster-area of number '3'.

3.3 Generative Adversarial Networks (GANs)

Generative adversarial Networks (GANs) were introduced by Goodfellow in 2014 [21] and since then they constitute one of the most researched subjects in Machine Learning. According to the model, a generative model $G(z;\theta_z)$ and a discriminative model $D(x;\theta_d)$ are simultaneously trained with competing objectives: the generator gets some noise z and produces an image that should confuse the discriminator in believing that this is a real image in the sense that it is produced from the actual image probability distribution. The discriminator is also trained with real images interchangeably and after adequate training, the generator is able to produce images that are in conformance with the distribution of the actual input domain. Training a GAN is actually a two player minimax game with the following value function

$$\min_G \max_D V(D,G) = \mathbb{E}\, x\sim p_{data(x)} [\log_2 D(x)] + \mathbb{E}_z\sim p_{z(z)} [\log_2(1 - D\big(G(z)\big) \quad (3.4)$$

The discriminator D tries to maximize the probability of correctly recognizing true images over the images produced by the generator G while the generator tries to maximize the probability of fooling the discriminator in believing that its produced images are real. In turn, this results in training a generator on producing images according to the probability distribution of the domain of the problem. Figure 3.10, borrowed from Medium ml-everything website [86], shows the architecture of Generative Adversarial Networks.

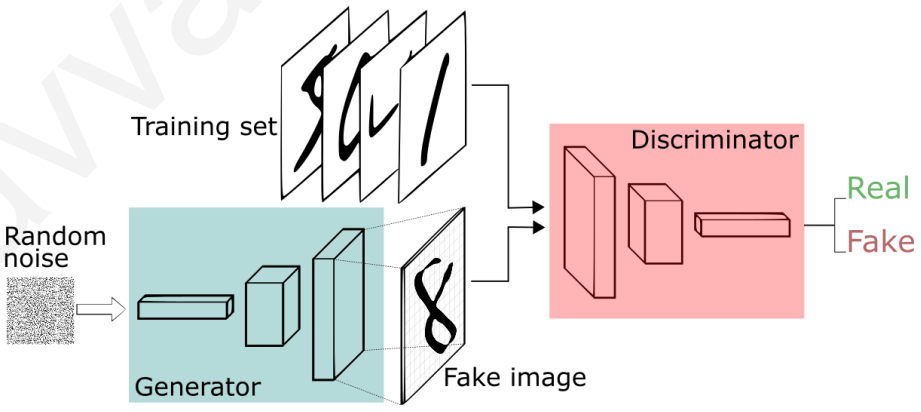


Figure 3.10: The architecture of Generative Adversarial Networks. Random noise is fed into the generator to produce an image that is discriminated by the classifier from real images sampled from the data distribution. The generator is trained to evade the detection of its images as fake by the classifier. The whole process constitutes a game theoretic approach for training a generator.

As soon as the Generative Adversarial Networks proved to be effective generative models, they went through a lot of investigation and modifications that improved their performance and their potential in dealing with unsupervised and semi-supervised problems. Mirza and Osindero [87] implemented the conditional flavor of the original generative adversarial network algorithm by feeding the generator and the discriminator networks with an extra piece of information that enabled conditioning the generated output on some auxiliary data such as a class label vector. Denton et al [88] used a Laplacian pyramid framework with conditional generative adversarial networks to construct a coarse-to-fine generative model producing CIFAR10 images that human evaluators mistaken for real images 40% of the time. Odena [89] modified the GAN discriminator in order not only to predict whether an image was real or was synthesized by the generator, but also to provide the class of the real images. In other words, the discriminator had *K+1* classes instead of just 2 (real or fake image) which is advantageous for training a data efficient classifier and generating good quality images when trained in a semi-supervised fashion. Conditional adversarial Networks were also used by Isola et al [90] to implement image to image translation and effectively synthesize photos from labels, reconstruct images from edge maps and colorize images. This work is notable because it illustrates that GANS can be trained to learn a mapping relating one image domain with another domain, without the need of hand-engineering these mappings. Cross-domain image generation has been studied by Taigman, Polyak and Wolf [91] by implementing a modified multiclass GAN that they called Domain Transfer Network to produce an image that is relevant to an input image.

Finally, Salimans et al [92] published a set of improved techniques for training GANs with their most interesting method being their semi-supervised learning using feature mapping. They considered a softmax classifier with *K+1* outputs with the first *K* outputs representing the problem classes while the remaining output represents the probability that an image came from the generator and not the actual data distribution. To train the network they used a lot of unlabeled information (unsupervised learning) and some labeled information (supervised learning). They also applied a different approach for training the generator in the sense that the generator's objective is no longer to maximize the real/fake output of the discriminator but to match the expected feature values of one of its intermediate layers such as

$$min \left\| \mathbb{E}_{x \sim p_{data}} r(x) - \mathbb{E}_{z \sim p_z(z)} r\big(G(z)\big) \right\|_2^2 \tag{3.4}$$

where $r(x)$ denotes the activations of such a layer. They called this technique feature mapping. The new objective function is the standard softmax loss function for the supervised part of the training and a modified version of the standard GAN game-function for the unsupervised part of the training during which samples can be drawn from the actual data distribution or produced by the generator. More specifically, for the unsupervised training the discriminator's objective is to minimize the probability of the fake (*K+1*) output for any data drawn from the real data distribution and maximize the specific probability output when the data examined is artificial (produced by the generator). Consequently, the two-fold objective function for the classifier becomes

$$L_{supervised} = -\mathbb{E}_{x,y \sim p_{data}(x,y)} \log_2 p(y \mid x), \; y \leq K \tag{3.5}$$

$$L_{unsupervised} = -\mathbb{E}_{x \sim p_{data}(x)} \log_2 (1 - p(y = K+1 \mid x)$$

$$-\mathbb{E}_{x \sim G} \log_2 (p(y = K+1 \mid x)) \tag{3.6}$$

During training, the generator is also performing feature mapping based on the features calculated by the discriminator. The reported results of this semi-supervised learning method are really good and surpass a great semi-supervised learning algorithm, the ladder network [93], when using just *100* labeled examples.

Generative Adversarial Networks do not require a Markov Chain sampling to draw a sample from the model. They perform the sampling process with a single forward pass through the generator. While this is a tremendous advantage as in the case of the variational autoencoders, they usually experience several issues during training. Their most important drawback is their tendency to present instability during training and frequent single-point collapsing. Single-point collapsing is the situation that the generator constantly outputs a specific pattern and is not able to escape such a situation during training. This is generally addressed as an instability in training and during the first years of the model's life the specific problem was mitigated by using a bag of tricks induced by experience in training such models. Some of these tricks are choosing the correct activation function for the output layer of the generator, objective function modifications to equivalent forms and the correct sizing of the discriminator's capacity. Although frequently unstable, GANs provided promising results when fine-tuned into proper functioning.

3.3.1 Wasserstein GAN

Single-point collapsing was properly addressed by Wasserstein GAN (WGAN) [94]. The authors of WGAN propose considering the Wasserstein divergence instead of the Jensen-Shannon divergence considered by the original formalization provided by Goodfellow. This approach minimizes the Wasserstein distance between $G(z)$ and p_{data} . The new loss functions become

$$L_D = -\mathbb{E}\, x{\sim}p_{data(x)}\,[D(x)] + \mathbb{E}_z{\sim}p_{z(z)}\left[D\big(G(z)\big)\right] \tag{3.7}$$

$$L_G = -\mathbb{E}_z{\sim}p_{z(z)}\left[D\big(G(z)\big)\right] \tag{3.8}$$

Applying the modified objective function loss departs the discriminator from the role of being a direct critic of telling the fake samples apart from the real ones. Instead, it is trained to learn a $\mathcal{K} - Lipschitz$ continuous function to help compute the Wasserstein distance between $G(z)$ and p_{data}. As the loss function decreases during training, the Wasserstein distance gets smaller and the generator's output comes closer to the real data distribution. The effectiveness of the proposed modification relies on maintaining the $\mathcal{K} - Lipschitz$ continuity of the discriminator's function which is not trivial. The authors propose the clamp of the discriminator's weights to a small window $[-0.01\ 0.01]$ after every weight update. This technique results in a more compact parameter space which helps the discriminator preserving the $\mathcal{K} - Lipschitz$ continuity. The complete training algorithm of the Wasserstein Generative Adversarial Network is shown in Algorithm 3.1 [94]. The authors in their experiments used $a = 0.00005,\ c = 0.01,\ m = 64,\ n_{critic} = 5$. They also proposed using the RMSProp (Root Mean square propagation) optimizer instead of the ADAM (Adaptive Moment) optimizer based on experimental results. The Wasserstein Generative Adversarial Network is indeed able to mitigate the problems of the original model but did not solve them in a definite manner. Weight clipping as a method for enforcing a $\mathcal{K} - Lipschitz$ constraint to a function is not an elegant one and many implications can arise from implementing it. If the clip uses small values it causes the vanishing gradients problem. On the other hand, large values may cause slow convergence and mode-collapsing which are the initial problems the modification is trying to solve. Some recent approaches try to replace weight clipping with other techniques for example gradient penalty [95]. The specific problem is a field of on-going research.

Algorithm 3.1 : Wasserstein Generative Adversarial Network [94]

Data: learning rate a, clipping parameter c, batch size m, critic iterations per generator iteration n_{critic}, initial critic parameters w_0, initial generator parameters θ_0.
Result: generator parameters θ

---

while θ has not converged **do**
  for $t = 0, ..., n_{\text{critic}}$ **do**
    Sample $\{x^{(i)}\}_{i=1}^{m} \sim \mathbb{P}_r$ a batch from the real data.
    Sample $\{z^{(i)}\}_{i=1}^{m} \sim p(z)$ a batch of prior samples.
    $g_w \leftarrow \nabla_w \left[\frac{1}{m}\sum_{i=1}^{m} f_w(x^{(i)}) - \frac{1}{m}\sum_{i=1}^{m} f_w(g_\theta(z^{(i)}))\right]$
    $w \leftarrow w + \alpha \cdot \text{RMSProp}(w, g_w)$
    $w \leftarrow \text{clip}(w, -c, c)$
  end for
  Sample $\{z^{(i)}\}_{i=1}^{m} \sim p(z)$ a batch of prior samples.
  $g_\theta \leftarrow -\nabla_\theta \frac{1}{m}\sum_{i=1}^{m} f_w(g_\theta(z^{(i)}))$
  $\theta \leftarrow \theta - \alpha \cdot \text{RMSProp}(\theta, g_\theta)$
end while

Chapter 4

Drawing Samples from Autoencoders and evaluating generative models

Autoencoders learn the data distribution by constructing an internal representation recalled during the encoding phase and reconstruct an output by expanding this data encoding during the decoding phase. In the case of using an explicit probability distribution for the internal representation like in Variational Autoencoders, the sampling process is more straight-forward: the mapping of the data to a distribution of predefined form and nature allows sampling of the model by using data points on this well-defined internal representation. Autoencoders that do not explicitly encode the input data to a specific and predefined distribution perform sampling through a more general approach that implements Markov Chain Monte Carlo (MCMC) sampling. For example, contractive autoencoders learn the tangent planes of the data manifold. Sampling a contractive autoencoder can be achieved by making random walks on the manifold surface using a sequential encoding-decoding procedure and injecting noise after each step. This allows for a transition from one point of the manifold to another. These data points on the manifold are naturally decoded to a sample of the data distribution. The whole process constitutes a Markov Chain. Denoising autoencoders can also be sampled using a MCMC approach.

4.1 Markov Chain Monte Carlo sampling

The Markov Chain Monte Carlo (MCMC) method is a statistical approach for sampling from a probability distribution when a direct or analytical approach is difficult or impossible. For example, there are situations that a data distribution is unknown and can only be sensed through experience. Starting from an arbitrary point, a chain of stochastic steps is developed looking for data points of higher probability. Pure Monte Carlo methods produce chains of statistically independent steps while Markov Chain Monte Carlo methods produce autocorrelated samples. Theoretically, the more steps induced in a MCMC sampling process, the closer the data points in the final steps of the chain are to the underlying distribution. In practice, it is quite difficult for denoising autoencoders to learn highly complex data distributions in such a detail to guarantee the proximity of a huge number of sampling steps to the data distribution. It is generally usual for autoencoders acting on complex data not to be able to maintain sampling quality for infinite chains. This happens because the MCMC cannot converge to a stationary distribution close enough to the target distribution. Another reason for the MCMC not converging to a stationary distribution is the inadequate representational power of the autoencoder.

Markov chains are essentially a random walk on a graph. The Autoencoders provide the next state of the walk by reconstructing an output based on their internal representation. Since the encoding and decoding phases are deterministic, the stationary distribution theorem [114] that governs the Markov Chain sampling is not satisfied without a crucial step: noise injection to make state-to-state transitions stochastic. The stationary distribution theorem relies on the fact that Markov chains are applied on a strongly connected graph where each state (graph vertex) has a path to all other states. Injecting noise to a previous state makes it possible to make a transition to any other state of the probability distribution. Noise injection provides a probability to the current model state for transitioning to any other possible model state. In this way, given a current state, any other state can be reached during sampling. The transition probabilities are near the expected value of a uniform distribution during the initial steps of the chain and become biased towards certain samples as the sampling process progresses. After a significant number of sampling steps, the transition probability should get significantly biased and manifest into the production of a stationary data point according to the underlying data distribution. The distribution that the Markov chain settles to and uses to draw its data points is called the equilibrium distribution and approximates the actual data distribution.

4.1.1 Markov Chain equilibrium distribution

Let's assume the application of infinitely many separate Markov chains sampling a distribution $q^t(x)$ at time-step t. To begin the process, we sample the initial values of x for each chain from some distribution $q^0(x)$. As the Markov chain is built, $q^t(x)$ is modified accordingly from the visited states. The probability that a single Markov chain state x transits to x' is

$$q^{t+1}(x') = \sum_x q^t(x)\, T(x' \mid x) \tag{4.1}$$

where $T(x' \mid x)$ is the transition distribution from state x to state x', meaning that each state may transit to any other state with a finite probability. This transition distribution can be represented by a matrix A such as

$$A_{i,j} = T(x' = i \mid x = j) \tag{4.2}$$

Considering all the Markov chains in parallel, we define the vector probability Q^t reflecting the Markov chains distributions. This means that on each step, Q^t changes according to

$$Q^t = A\, Q^{t-1} \tag{4.3}$$

Subsequent updates of the above form essentially correspond to subsequent matrix A multiplications of the model states. This in turn is equivalent to exponentiating the matrix such as

$$Q^t = A^t\, Q^0 \tag{4.4}$$

because each step operates on the previous value of Q. Since matrix A is a stochastic matrix because it represents a probability distribution (state-transition probabilities), the Perron-Frobenius theorem [115, 116] is applicable: the largest eigenvalue of the matrix is real and equal to 1 . A necessary precondition for the Perron-Frobenius theorem is that there exists a non-zero probability for the transition of each state of the model to all other states. Using this theorem and the fact that the transition matrix gets exponentiated we can easily observe that decomposing matrix A using eigen decomposition [117] results to

$$Q^t = (V\, diag(\lambda)\ V^{-1})^t\, Q^0 = V\ diag(\lambda)^t\ V^{-1}\, Q^0 \tag{4.5}$$

This result shows that all eigenvalues less than 1 decay at each step because there are exponentiated until their value is close to 0 which drives Q into converging to a stationary distribution that does not change for any subsequent step. The transition operation produces new states (data points) but the sampled distribution providing the data points stays the same. The equilibrium distribution is the same as the data distribution we try to model provided that the transition operation is appropriate. One way to ensure that the transition operation is effective, is to use Gibbs sampling [118, 119] for selecting the next state from a model that is trained to learn the data distribution. Each variable sampled from the model is conditioned on its neighbors. As long as the variables are statistically independent from each other given their neighbors, they can be sampled simultaneously leading to the so called block Gibbs sampling approach.

4.1.2 The problem of mode-mixing

Markov Chain Monte Carlo methods tend to mix poorly especially when the data distribution has separate modes and when there is increased correlation between the variables. Ideally, MCMC would make a random visit to the x surface, preferring areas of high probability. In practice, in high dimensional spaces and complex data distributions, MCMC methods experience poor mixing and samples tend to be very similar. This happens because the chain gets stack in a mode of the data distribution where the surrounding space represents variations of similar concepts. In terms of energy-based models (EBMs), the chain prefers transitioning into configurations of lower energy and gets trapped in specific areas becoming reluctant of choosing other configurations. While the chain may occasionally escape from such areas, it tends to do this for only a brief time. It eventually returns to such areas and thus produces samples of little variation.

MCMC methods using Gibbs sampling are especially vulnerable to such problems mainly because the updates are conditioned on other variables. This strongly affects their willingness for transitioning from higher probability areas to lower probability areas even if short beyond the lower probability area exists a high probability neighborhood. Figure 1 illustrates the problem of poor mixing in Markov Chains using Gibbs sampling.

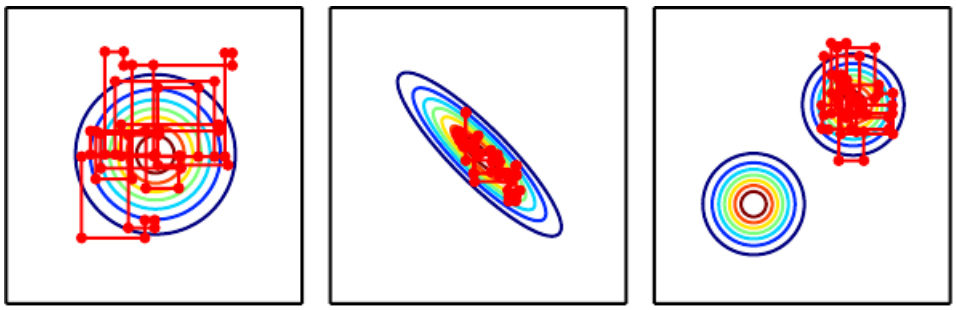


Figure 4.1: Chain mixing for three different distributions. The chains are initialized at the mode of the distributions. On the left, a multivariate Gaussian distribution with independent variables is sampled with a Markov chain that has a satisfactory mixing. In the middle, the multivariate Gaussian distribution with correlated variables makes it difficult for the Gibbs sampling method to escape from the space close and around the mode. This is due to poor chain mixing because of the variables' correlation. Generally, statistical dependence of the distribution variables make Gibbs sampling inefficient because the updates of the variables are conditioned on other variables. On the right, a mixture of Gaussians shows that Gibbs sampling never escapes the mode at which the chain is initialized.

In real problems modelled by autoencoders the complications of poor mixing are even greater because of the presence of many modes in the data distribution. The high dimensionality of the data also contributes to the problem. Generally, the presence of data modes raises a difficulty for generative models and restrains them from producing samples of increased quality and diversity. Besides autoencoders, other well-established models like GANs suffer from the similar problem of "mode-collapsing'' which may compromise their functionality completely.

4.2 MCMC for sampling Denoising Autoencoders

Generative Autoencoders can be sampled by a Markov Chain Monte Carlo (MCMC) process. Markov chain sampling is established as a legitimate methodology for drawing samples from denoising Autoencoders due to the work of Vincent et al [120, 121]. More specifically, Vincent et al provided a formal probabilistic interpretation for denoising Autoencoders reflecting their ability to approximate the input distribution. Vincent et al proved that the training criterion of a denoising Autoencoder is equivalent to matching the score of a specific energy-based model to that of a nonparametric Parzen density data estimator. Alain and Bengio [101] additionally proved the ability of a denoising Autoencoder to account for the derivative of the data log-

density in proportion to the difference between the reconstruction and the actual input. The denoising auto-encoder learns how to move the corrupted data back to the data manifold because it is trained to minimize the vector field $P_\varphi(X|Z) - X$ which is actually a vector pointing to the nearest point on the manifold. This vector field corresponds to the score $\nabla_x \log_2 p(x)$ up to a multiplicative factor which in turn corresponds to the average root mean square error of the Autoencoder during training. Bengio et al generalized this result for data corruptions other than the Gaussian and reconstruction losses other than the mean square error. This establishes the denoising autoencoders as generative models approximating the underlying data distribution. Their work also stressed that for capturing the true data distribution, the model should be able to build multi-modal representational distributions of the data probability $P(x)$. They next showed [122] that the stationary distribution of a MCMC method is an estimator of the actual data distribution learned by the autoencoder. Furthermore, Jimwoog Im et al [84] provide some necessary conditions, which if satisfied, any autoencoder defines an energy landscape that is representative of the data log-density. The conformance to these conditions adds to the autoencoder the characteristic of conservativeness. Jimwoog Im et al [84] also showed that denoising autoencoders enforce these conditions locally which means that a classifier working synergically with a denoising autoencoder can use its features to drive the autoencoder into configuration areas of high probability during the sampling process. Markov chain sampling for Autoencoders can be implemented in two flavors. The first approach [122] starts the chain by sampling the input space $P(x)$, then calculating the latent representation and finally generating an output according to it. The generated sample is applied to the input of the model and the process is repeated for several iterations.

$$x_0 \sim P(x)\,, \quad z_0 \sim Q_\theta(Z \mid X = x_0)\,, \quad x_1 \sim P_\varphi(X \mid Z = z_0) \tag{4.6}$$

This process is a Markov sampling process with a transition operator

$$T(Z_{t+1} \mid Z_t) = \int Q_\theta(Z_{t+1} \mid X)\, P_\varphi(X \mid Z_t)\, dX \quad , \quad t \geq 0 \tag{4.7}$$

Using a sample from the input distribution to initialize the MCMC chain tends to produce biased samples. A second approach by Kingma and Welling [83] and Makhzani et al [123] may be used when a prior distribution is enforced on the latent space Z. The initial latent representation can be directly sampled from the prior distribution $P(Z)$ and then passed to the decoder part of the generator producing a sample that is hopefully in proximity to $P(X)$. The MCMC chain is

formed by applying the output of the decoder to the input of the model and repeating the process multiple times such as

$$z_0 \sim P(Z)\,, \;\; x_0 \sim P_\varphi(X \mid Z = z_0) \tag{4.8}$$

Generally, a generative autoencoder can be sampled from the following process according to Creswell et al. [124]

$$z_0 \sim P(Z)\,, Z \in R^k$$

$$x_{t+1} \sim P_\varphi(X|Z_t), \quad z_{t+1} \sim Q_\theta(Z \mid X_{t+1)}) \tag{4.9}$$

with k being the dimensionality of the latent representation. This process converges to a stationary distribution $P'(Z)$ which is close to the prior distribution $P(Z)$.

For denoising generative autoencoders there is an extra step of adding noise to the input such as

$$z_0 \sim P(Z)\,, Z \in R^k,$$

$$x_{t+1} \sim P_\varphi(X|Z_t), \;\; \tilde{x}_{t+1} \sim C\left(\tilde{X}\middle|X_{t+1}\right), \;\; z_{t+1} \sim Q_\theta(Z \mid \tilde{X}_{t+1}) \tag{4.10}$$

where C is the noise-corrupting operation.

The generalized sampling process of a denoising autoencoder is shown in Figure 4.2 and is comprised of the following steps assuming a denoising autoencoder using Mean Square Error loss function with an encoder function $Q(z|x)$ and a decoder function $P(x|z)$:

1. Given a previous state x , inject noise to sample $\tilde{x} \sim C(\tilde{x}\,|x)$
2. Encode $\tilde{x}$ into $z = Q(z \mid \tilde{x})$
3. Decode z into a new state $\hat{x} \sim P(\hat{x} \mid z)$

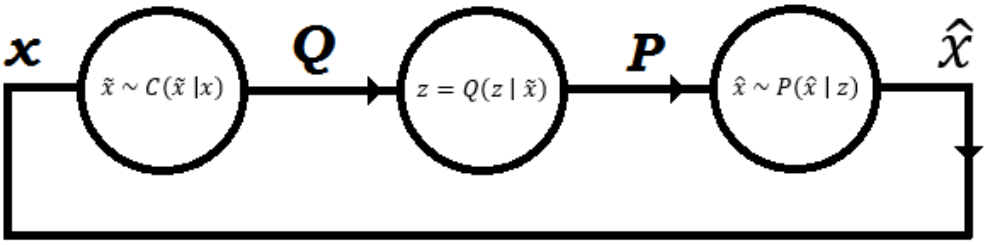


Figure 4.2: The Markov chain associated with sampling a denoising autoencoder trained using the Mean Square reconstruction Error (MSE). Each model state passes through the Markov chain following a corruption phase, an encoding phase and finally a decoding phase to produce the next state. Afterwards, the reconstruction is fed back to the input of the autoencoder as the previous model state, forming a continuous sampling chain.

4.2.1 The walk back algorithm

Bengio et al [122] proposed a modified training procedure for denoising autoencoders that accelerates learning and improves performance of the model when specifically used as a generative model. The one-step encoding/decoding pass during training may be replaced by a k-step iterative reconstruction of patterns located in the vicinity of an original training example. For every training pattern a list of reconstructed patterns is created and each of these patterns can be found on the trajectory of a random walk initiated with the training example. The process is similar to the contractive divergence algorithm used for training Restricted Boltzmann Machines [125]. The loss function can be applied either to every k-step of the walk or only to the last k-steps. In theory the walk back algorithm does not produce a different solution from the 1-step training in the sense that the same stationary distribution should be obtained. The difference is that during training, many spurious modes are more likely to be removed due to the random walk around a training example. Figure 4.3 [122] shows how the walk back algorithm may improve the samples drawn from a generative denoising autoencoder.

Figure 4.3: Samples drawn from a denoising autoencoder trained on MNIST digits. On the left, the samples of a model not implementing the walk back procedure are shown. On the right the samples drawn from a model implementing the walk back procedure suggest that there are less spurious samples [122].

The idea of the walk back algorithm is also used in Variational Autoencoders (VAE) by Anirudh Goyal et al [126] as a way to learn a transition operator. Anirudh Goyal et al claim that experiments performed using the Variational walk back procedure systematically produces better samples and the learned transition operator keeps producing samples of good quality for longer lengths of the sampling chain.

4.3 Drawing Conditional Samples from Generative models

Generative models use an auxiliary distribution from which a sample is reconstructed. Variational Autoencoders apply an inference step during training to map an input to such a distribution and a generative step to reconstruct the pattern. Generative Adversarial Networks do not apply an inference step and perform the generative step directly and output a data sample from a noise signal. Autoencoders need an encoding step to map the input to a latent distribution and a decoding step to reconstruct an output with both steps being part of a Markov Chain. The intermediate distribution in VAEs and the generalized Autoencoder (AE) is modelled directly from the input considering the data distribution and all its modes collectively. It cannot explicitly account for the different modes resulting from having different problem classes. In order for the data generative process to be aware of the problem classes, the encoder (inference) and the decoder (generative) steps must be conditioned on a variable representing the classes. For example, the unconditional VAE objective function must be changed from its original form

$$log_2 P(X) = E_{z\sim Q_\theta(Z|X)}\Big[log_2 P_\varphi(X|Z)] - D_{KL}(Q_\theta(Z|X) \parallel P(Z)\Big)$$

$$+ D_{KL}(Q_\theta(Z|X) \parallel P_\theta(Z|X)) \quad (4.11)$$

to its conditional form that depends on a variable c which represents the problem class such as

$$log_2 P(X|C) = E_{z\sim Q_\theta(Z|X)}[log_2 P_\varphi(X|Z,C)] - D_{KL}(Q_\theta(Z|X,C) \parallel P(Z|C))$$

$$+ D_{KL}(Q_\theta(Z|X,C) \parallel P_\theta(Z|X,C)) \quad (4.12)$$

This formulation conditions all distributions in the objective function on a variable which encodes the input class. In order to incorporate the new conditional probabilities the class label of each training example must also be provided to the model along with the data itself. The architecture of the conditional VAE is shown in Figure 4.4.

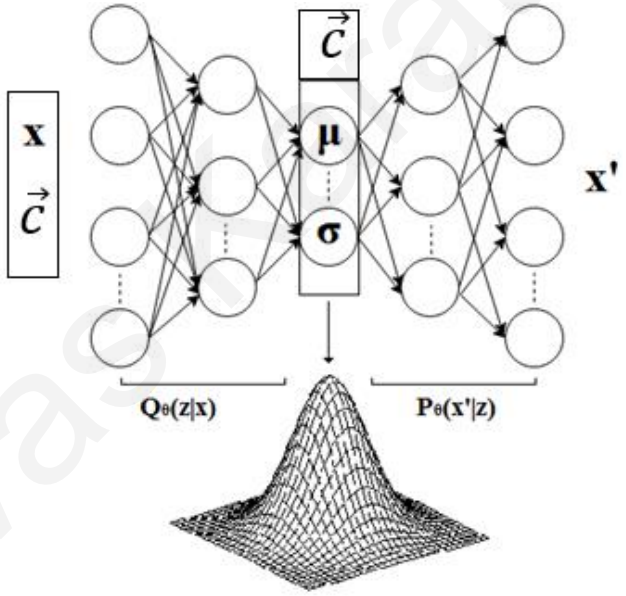


Figure 4.4: The conditional VAE architecture requires the merging of the input data with a vector $\vec{c}$ representing the label of the training example. The same vector must also be merged with the latent distribution.

The same principle can be used for GANs. Instead of only feeding the noise signal to the generator, a vector label is applied as well to the input of the generator and the input of the discriminator as shown in Figure 4.5 [87] .

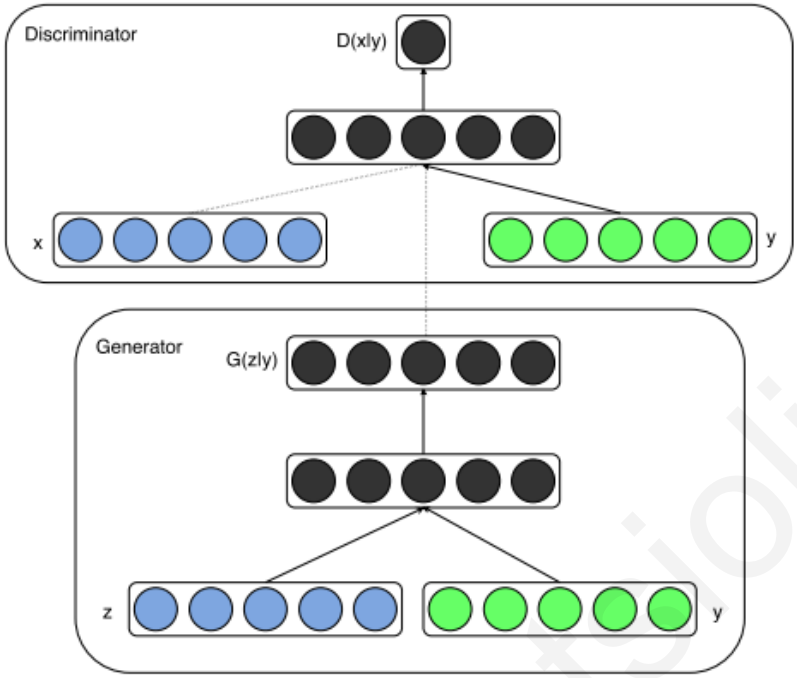


Figure 4.5: The CGAN [87] architecture for generating samples conditioned on the input label y.

4.3.1 Generative class-conditional denoising autoencoders

Merging the label of each training example to the actual data and to the constructed latent code also works for denoising Autoencoders. The data labels make the model learn a separate manifold for each problem class. This means that, within a common network, denoising Autoencoder models are learned for each problem class but they are not entirely distinct from each other because they share weights. By sharing parts of the network, all implicitly learned denoising autoencoders may use common features which may result in better performance and higher efficiency. Jan Rudy and Graham Taylor [127] propose a gated autoencoder architecture first introduced by Memisevics [128] that operates as a generative class-conditioned denoising autoencoder. Gated autoencoders learn a function of the labels $w(y)$ and use it to modulate the weights of the autoencoder. During training, noise is applied both to the input data and to the labels and the denoising criterion is applied like in the case of the plain denoising autoencoder. Figure 4.6 [127] shows their proposed architecture.

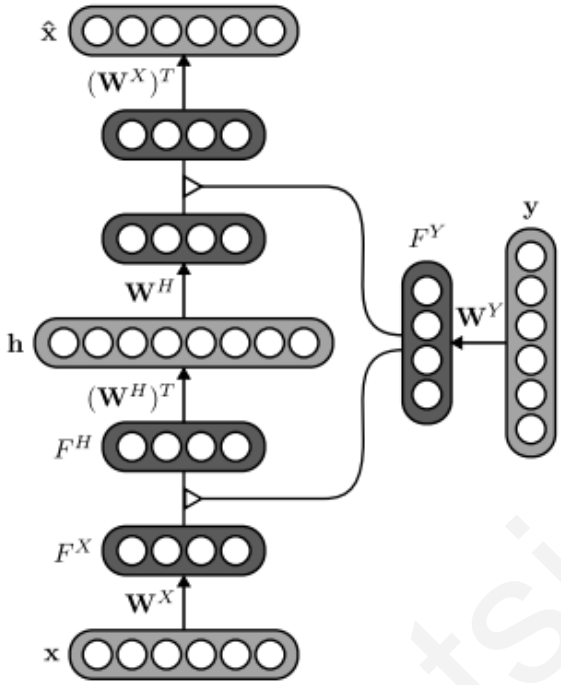


Figure 4.6: The conditional gated autoencoder model. The label is forwarded to layer F^y, creating factors that modulate the weights of the network. Scaling the weights modifies the calculated features depending on the data class and thus the network implicitly learns a model for every problem class [127].

Encoding the labels through a network layer produces F^y factors that are used to scale the features of the model. This scaling constitutes the gating effect of the factors since they control the magnitude of each feature depending on the problem class. While this approach promotes mode learning based on the labels, it also applies significant weight sharing between the learned data modes.

The class-conditional Gated Autoencoder does partly comply with the theoretical foundations of the classical autoencoder but also raises some theoretical concerns regarding the requirements of the MCMC sampling procedure. Markov Chain Monte Carlo methods are stochastic algorithms that perform random walks in the form of sampling chains. Fixing a part of the input and the latent distribution violates the requirement for the transition matrix being a stochastic matrix representing state-transition probabilities from each model state to every other state. By definition, generating a pattern belonging to class y_k through fixing a part of the model's input implies a zero probability of producing a pattern belonging to any problem class other than y_k. Additionally, in MCMC methods the output of the AEs are interpreted as model states that must

be applied to the input of the model in a series of steps that make up the Markov chain. In this sense no elements of the state should be fixed. Strictly speaking, the class-conditional Gated Autoencoder is theoretically adequate because it is equivalent to training k separate models for k distinct problem classes incorporating a degree of feature sharing.

4.4 Generative models Evaluation

Generative models, like every other Machine Learning model, are evaluated by researchers in order to demonstrate their effectiveness and the quality of the generated samples. Unfortunately, there are not any straight forward and conclusive procedures to test how well a generative model performs. Claiming that a newly proposed model is superior to any of the existing models is risky and such a claim is often doubtful. This situation arises from the fact that there are not decisive universally-applicable metrics that can definitely measure the quality of the samples.

4.4.1 Difficulties in comparing generative models

A descent but not definite metric for evaluating generative models is the estimate of the log-likelihood of the samples drawn from it: the higher the log-likelihood the better the model's samples reflect the actual data distribution. However, many times the log-likelihood of the model cannot be calculated directly and an approximating estimation is calculated instead. On the other hand, some models like the Variational Autoencoders (VAEs) provide a theoretical lower bound for the log-likelihood. The comparison between a lower bound and a rough estimation of the log-likelihood is generally inconclusive unless the lower bound is higher. Since the lower bound provided by VAEs is actually quite low compared to the performance of modern generative models, a trained network having an actual log-likelihood even near this lower bound is usually discarded before an evaluation is conducted. Even if the log-likelihood of a model is estimated implicitly or directly, it does not mean that it is directly comparable to the log-likelihood of other models. As Theis et al [129] point out, a model may be better at assigning high probability to most realistic points while another model may be better at rarely assigning high probability to unrealistic points. This behavior results from choosing to minimize $D_{KL}(p_{data} \parallel p_{model})$ or $D_{KL}(p_{model} \parallel p_{data})$. The Kullback-Leibler (KL) divergence measures the non-negative difference between two distributions P, Q such as

$$D_{KL}(P \parallel Q) = E_{x\sim P}\left[\log_2 \frac{P(x)}{Q(x)}\right] = E_{x\sim P}\left[\log_2 P(x) - \log_2 Q(x)\right]$$

D_{KL} divergence is not a distance measure because it is not symmetric. This reflects the fact that

$$D_{KL}(P \parallel Q) \neq D_{KL}(Q \parallel P)$$

Symmetry is a required mathematical property for every distance measure. The asymmetry of the D_{KL} divergence relies on the fact that it is the outcome of an expected value whose terms cannot be exchanged and still maintain the same result. Because of this asymmetry, choosing the order of the examined distributions becomes very important and plays a decisive role on how the model behaves. Figure 4.7 shows the different results obtained when a mixture of Gaussians is approximated by a single Gaussian for both cases: minimizing $D_{KL}(P \parallel Q)$ and minimizing $D_{KL}(Q \parallel P)$

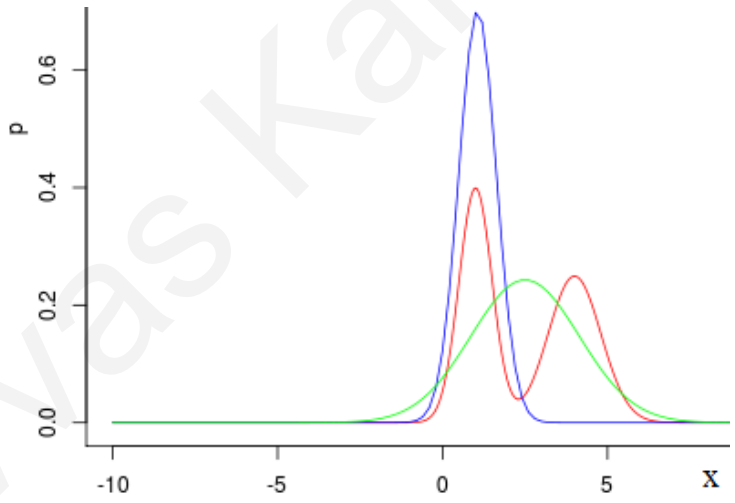


Figure 4.7: A multimodal Gaussian distribution P (shown in red color) is comprised of a Gaussian with $\mu = 1, \sigma = 0.8$ and a second Gaussian with $\mu = 4$, $\sigma = 0.5$. Using KL divergence, a single Gaussian fits the original distribution in two different ways depending on the minimization objective. Minimizing $D_{KL}(P \parallel Q)$ results in the green distribution while minimizing $D_{KL}(Q \parallel P)$ results in the blue curve. The blue distribution prefers to assign high probability to the most probable points of P while the green distribution prefers to put high probability on many points and resorts in blending the modes together.

Another issue when evaluating generative models is the different data types representing the input data. For example, two models may be trained on the same dataset having different data types. The popular MNIST dataset is usually represented with real values but sometimes its binary-valued version is used to train models as well. This results in having log-likelihoods that occur at different spaces. The log-likelihood for models trained on binary-valued datasets has a maximum value of zero. On the other hand, models trained on real-valued datasets may have large log-likelihoods for representing the model's density. It is obvious that there is no meaning in comparing two such models. Even in the case of two models being trained with the same binary dataset, the way that the binarization was performed is important. If the first dataset binarization uses thresholding (e.g. at value 0.5) and the other uses probabilistic binarization (sampling with a threshold value equal to the real value of each variable) then the comparison is still taking place at different spaces.

Log-likelihood is widely used as a metric of a model's performance but special care should be taken to avoid arbitrary conclusions. A model may apply a special treatment on specific areas of the data that are not the most important to the problem trying to solve. For example, a model could put energy and effort towards assigning low variance to the background of some image data and not at the foreground which is more important. This could provide a considerable boost in terms of a log-likelihood evaluation without serving the correct task.

4.4.2 Evaluation by visual inspection

A model performing really well is expected to generate realistic samples which are really hard to distinguish from actual samples drawn from the data distribution. Naturally, it is a usual practice to evaluate the performance of generative models with visual inspection of the samples they produce. Denton et al [88] suggest that this task should be performed by experimental subjects that are unaware of the source distribution. A common problem of this approach is the fact that models learning the data distribution (or part of the data distribution) by heart, output plain copies of the training examples and are rewarded with high evaluations. One way of overcoming this problem is by verifying that generated samples are not exact copies of any of the training data. The samples are compared to their nearest neighbor in the training set in terms of Euclidean distance to detect whether the model over fits the data. A variation of this problem is the case of a model learning the distribution of only a few classes of the dataset and ignores the rest of them.

Despite the problems of visual inspection, the specific approach is the one method that is almost always used at least by the designer of the model during its development. Its primary advantage is the consideration of those performance aspects that are the most important to the task of generating realistic samples.

4.4.3 Parzen-window density estimation

Many times there is not an analytical way to directly estimate the density of a distribution. In such situations a variety of density estimation methods can provide an approximate value for the log-likelihood of the underlying distribution. The most popular methods are histograms and kernel density estimation. Using histograms can be tricky and most of the times the estimations are far from the actual values because of the data bins used which must be tuned depending on the actual case. Kernel or Parzen-window density estimation is a non-parametric method for estimating continuous density function from the data distribution f comprising of data points $x_1, x_2, x_3 \dots x_n$. The main idea is to approximate f with a mixture of continuous distributions Φ usually called kernels. These kernels are centered at the data points x_i and have a scale (bandwidth) h such as

$$\hat{f}_h(x) = \frac{1}{n\,h} \sum_i \phi\left(\frac{x - x_i}{h}\right)$$

The most commonly used distribution as a kernel function in Parzen-window density estimation is the equivariant Gaussian distribution

$$p(x) = \frac{1}{(2\pi)^{\frac{d}{2}}\,|\Sigma|^{\frac{1}{2}}}\; e^{-\frac{1}{2}(x-x_i)^T\,\Sigma^{-1}\,(x-x_i)}$$

The method relies on the fact that the probability q that a vector falls in region $\mathcal{R}$ is given by

$$q = \int_{\mathcal{R}} p(x)\,dx$$

Assuming that $\mathcal{R}$ is small enough that $p(x)$ is almost constant within the specific region, then it holds that

$$q = \int_{\mathcal{R}} p(x)\, dx \cong p(x) \int_{\mathcal{R}} dx = p(x)\, V$$

where V is the volume of the region $\mathcal{R}$. Further assuming that a number k of the samples $x_1, x_2, x_3 \ldots x_n$ fall into the region induces

$$q = \frac{k}{n} \quad \rightarrow \quad p(x) = \frac{k/n}{V}$$

A metric for the k data points located in the volume V defined by the bandwidth (variance) parameter is estimated by the Gaussian distributions. This metric becomes a density estimator given some data points of the original distribution. Generally, the variance of the Gaussian distributions is empirically tuned but a rule-of-thumb estimator is provided by Silverman [130]. Figure 4.8 shows the application of the Parzen-window density estimation on some data points in one dimensional space.

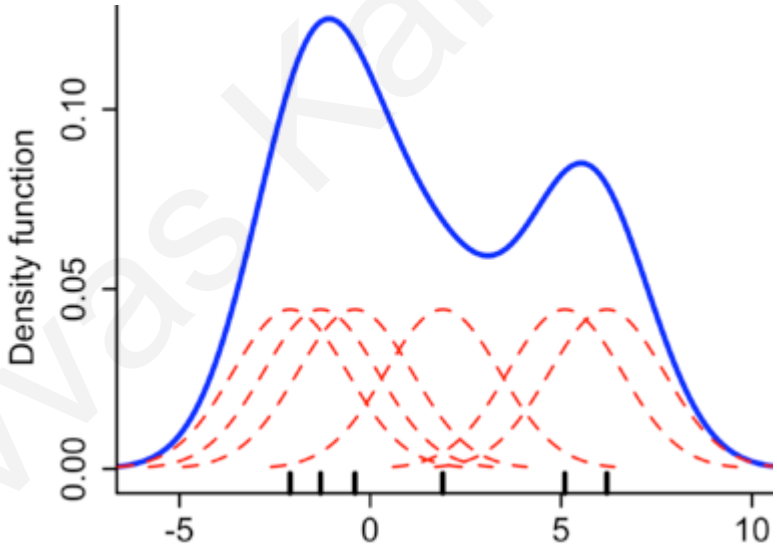


Figure 4.8: Density estimation (blue line) as provided by a mixture of Gaussians (red line) of the same variance. Each Gaussian is centered at a data point shown with short black lines on the x-axis.

The Parzen-window density estimation method for the evaluation of generative models with intractable likelihoods was proposed by Breleux et al [131]. The specific approach has been used in several cases during the recent years [132, 133, 134, 21].

Chapter 5

Conditional Denoising AutoEncoder (CG-DAE)

The proposed model is a generative denoising autoencoder that has an embedded data classifier in its architecture in order to take advantage of class-based discriminating features and produce better data samples. CG-DAE is a conditional generative model and is sampled with a Markov Chain Monte Carlo (MCMC) process according to a label that denotes the desired (or undesired) class or classes. In this sense any predefined class or characteristic may be positive or negative, meaning that it can be instructed to be present or absent from the generated sample. Allowing discriminative information in the form of feature detectors to be present in the latent representation of the autoencoder can be generally beneficial. This technique is an alternative approach to the Variational Autoencoders (VAEs) that enforce a prior on the latent distribution. Supervised learning may be generally able to serve unsupervised learning through an interaction between the two paradigms. However, the extreme majority of research done on the interaction of the two learning regimes has the goal of using un-supervised learning to improve supervised learning. This Thesis explores the two learning paradigms' interaction in the opposite direction.

5.1 Introduction

Supervised learning has been dominating the interest of the machine learning research society for decades mainly because it is the straight forward approach for building a reasonable understanding on how machines make decisions and generalize well on data concepts. Consequently, supervised learning seems to provide an efficient path for establishing the foundations of AI and is a stepping stone for pursuing the understanding of the various components of human intelligence. So far, great progress has been achieved in the field of supervised learning and many tasks that seemed difficult-to-deal-with in the past can be now performed by machines with close to or even better than "human performance" [179, 180 ,181]. On the other hand, unsupervised learning has also progressed during the last decade. While new ideas have produced several meritorious results [182, 83, 183, 184, 185], there is admittedly a gap between the achievements of supervised learning and those reported by unsupervised learning. While supervised learning research has still a lot way to cover, it might have already covered more way than we realize. Making further progress in the field could be much slower unless a cooperative paradigm among the different types of learning is established. The amount of information available to a learning machine during training depends on the type of learning adopted. Supervised learning provides significantly less information than self-supervised learning. The latter is a kind of unsupervised learning that does not require labels produced by humans but uses embedded metadata and naturally available data properties as supervisory signals instead. In this spirit, Yann LeCun (2019) provided the "cake analogy" to stress the wealth of information hidden in self-supervised learning (and unsupervised learning in general) in comparison to supervised learning. LeCun claims that self-supervised learning generally provides several orders more information than supervised learning. Current machine learning approaches are able of utilizing labeled data and thus are highly competent on performing supervised learning. Still, these approaches require a huge number of training examples in order to reach a high level of machine sophistication. In other words, they require a huge amount of labeled data so as to collect the amount of information required for achieving high performance. This fact advocates LeCun's claim. Current unsupervised learning algorithms are still unable of unleashing the information wealth hidden in the training process of the specific learning regime. At the same time, supervised learning may be coming closer to a point that it becomes increasingly difficult for it to take advantage of progressively increasing amounts of available information in the pursuit of training really intelligent models. More interestingly, human brain

seems to mostly operate under the unsupervised regime and more specifically it uses some kind of self-supervised learning signals manifested as bio-chemical substances (for example dopamine which plays a major role in the motivational component of reward-motivated behavior).

In this work we are focusing on the relationship of supervised and unsupervised learning. This interaction has been mostly investigated in a single direction: using unsupervised learning to improve supervised learning performance. Unsupervised pre-training by stacking restricted Boltzmann machines [186, 42, 187] or layer-wise training with autoencoders [98] provides good initial network parameter values which in turn produces better results after training with supervised learning. With a good initial network state, the regularization of the network is easier which allows the use of bigger models with less chance of over fitting the training dataset. Pre-training the model with unsupervised learning, allows supervised training to apply weight updates at the vicinity of appropriate network configurations. While these approaches seemed very promising at first, they turned to be unnecessary when deep learning set off some years later. However, some very interesting semi-supervised learning algorithms were proposed during the last decade that can really boost classification results by using a small number of labeled examples and a lot of unlabeled data [155, 188, 92]. The alternative direction of the supervised-unsupervised learning relationship has been investigated far less than its counterpart.

We introduce a scheme that combines both learning regimes in such a way that unsupervised learning performance is enhanced and enriched by using labeled examples during training. The usual practice and the dominating form relating the two machine learning paradigms is using unsupervised learning to improve supervised learning's performance.

5.2 Model Architecture

The proposed model is composed of a denoising autoencoder and a classifier trained in an interconnecting architecture as shown in Figure 5.1. In autoencoder structural terms, the encoder structure provides a latent representation Z_1 which is forwarded to the classifier mechanism while it is also fed into the decoding structure as seen in Figure 5.2.

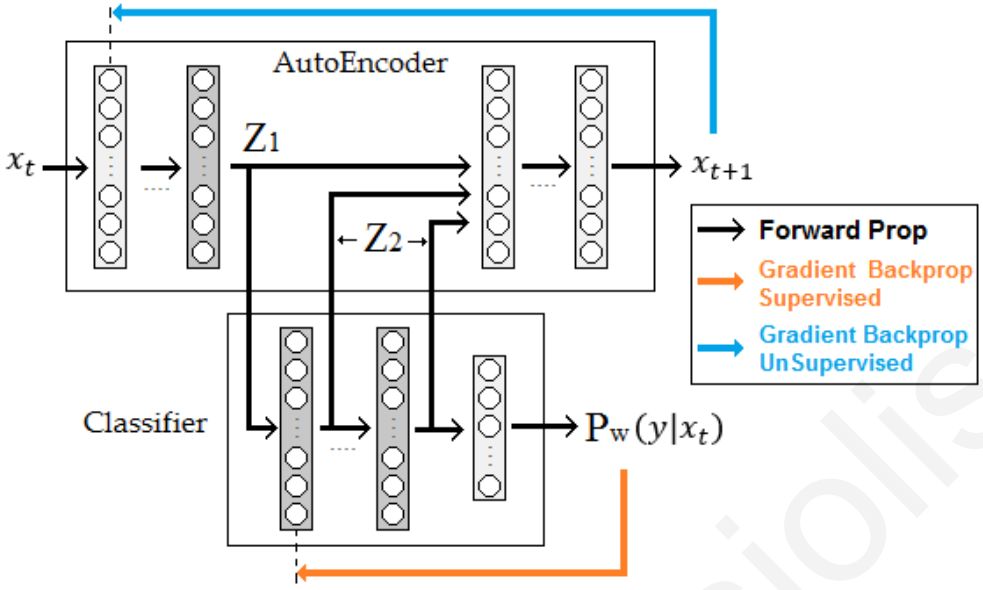


Figure 5.1: The architecture of the proposed model. The layers shown in darker color compose the latent representation of the network. The classifier is fed with the outcome of the encoder structure of the autoencoder while the hidden layers of the classifier contribute to the latent representation that excites the decoder structure. The backpropagation gradient flows are shown for both the supervised and unsupervised training examples. The figure is best seen in color.

The classifier feeds the decoder with an additional latent representation which is constructed by using only the encoder's output without the provisioning of direct input space information. This means that the input space X is not directly connected to the classifier. However, input labels are of course provided to the classifier in order to support its discriminating operation. The latent representation contributed by the classifier is composed of the outputs of its hidden layers. A unified latent representation Z is formed by merging the two latent variable sets to form a joint distribution $Z = \{Z_1, Z_2\}$. This joint distribution is fed to the decoder structure that calculates an output according to $P_\varphi(X|Z_1, Z_2)$.

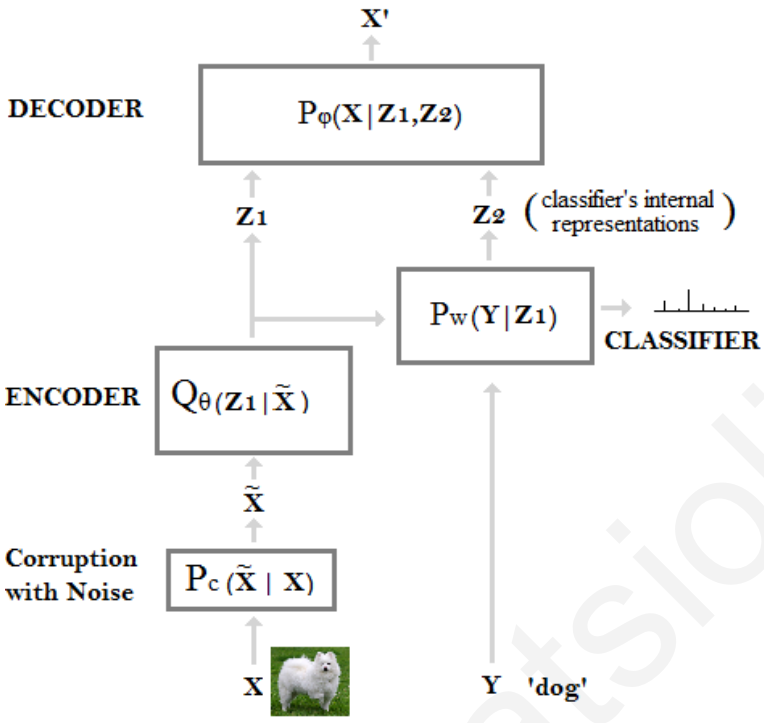


Figure 5.2: The structural components of the conditional generative denoising autoencoder: an input is corrupted with noise and encoded into a latent representation Z_1. This representation is fed into a classifier and a second latent representation component Z_2 is formed using the outputs of its hidden layers. Z_1 and Z_2 are provided to the decoder structure in order to reconstruct the input.

This essentially means that the unified latent space is composed of two informative distributions: the feature mappings of the classifier Z_2 that encapsulate discriminative information within the data distribution and the encoding Z_1 provided by the encoder structure. The objective function for the training procedure unifies both distinct mechanisms and drives the optimizer to find a solution that satisfies both aspects of the problem: to correctly classify the data and be able to generate reasonable data samples. Achieving state of the art classification performance on a given problem is not the primary target of the model. Nevertheless, implementations of the model have shown that good classification performance tends to increase the quality of the generated samples.

5.3 Training the CG-DAE

We assume that input data has values in the range [0,1] and thus input normalization is applied if necessary. By further assuming M unlabeled examples, N labeled examples, K problem classes and corruption of the normalized input with a Gaussian Noise process $\mathcal{N}(0\,,0.5)$, the objective functions used in turn to train the model are shown below.

$$\tilde{x}_i = x_i + \mathcal{N}(0\,,0.5)\,,\; Z_1 = Q_\theta(Z|\tilde{x}_i)$$

$$J_{supervised} = -\frac{1}{N}\sum_{i=1}^{N}\sum_{j=1}^{K} y_{ij}\log P_w(Y|Z_1) \tag{5.1}$$

$$J_{unsupervised} = \frac{1}{M}\sum_{i=1}^{M}\left(x_i - P_\varphi(X|Z_1,Z_2)\right)^2 \tag{5.2}$$

The input data should have values in the range [0,1] and thus input normalization is applied if necessary. The decoder structure of the autoencoder is denoted by P_φ , the encoder by Q_θ and the classifier by P_w. Accordingly, the output of the encoder is Z_1 and the classifier's hidden layers' output (merged to form a large vector) is Z_2. These two outputs are merged to a joint distribution and provided as input to the decoder as illustrated in Figure 5.1 in order to update the decoder. The objective function for the labeled data is the cross entropy loss in respect to the labels assigned. The model is trained with the ADAM [135] optimizer using mini-batches and dropout [42] applied to the classifier's layers. The exact training algorithm is shown in Algorithm 5.1.

Algorithm 5.1 : Training Procedure of the Conditional Generative Denoising Autoencoder

Data: batch size M, initial encoder parameters θ_0, initial decoder parameters φ_0, initial classifier parameters w_0

Result: encoder parameters θ, decoder parameters φ , classifier parameters w

---

Repeat until convergence

1. Sample a mini-batch $MB_{unlabeled}$ of size M such as $MB_{unlabeled} = \{x_1, x_2, \dots, x_M\}$
2. Sample a mini-batch $MB_{labeled}$ of size M such as
   $MB_{labeled} = \{(x_1, y_1), (x_2, y_2), \dots, (x_{M,} y_M)\}$
3. Apply noise corruption to both mini-batches such as
   $\widetilde{MB}_{unlabeled} = MB_{unlabeled} + \mathcal{N}(0,0.5)$
   $\widetilde{MB}_{labeled} = MB_{labeled} + \mathcal{N}(0,0.5)$
4. Perform a training step on the generative part of the model
   a. Forward propagate $\widetilde{MB}_{unlabeled}$ through the encoder and get Z_1
   b. Forward propagate Z_1 through the classifier and get Z_2
   c. Forward propagate $\{Z_1, Z_2\}$ through the decoder and get output x'
   d. Back propagate gradient information ∇_φ and update decoder parameters φ
   e. Calculate ∇_θ neglecting the gradients of the classifier to update encoder parameters θ (stop the gradients flowing back from the classifier).
5. Perform a training step on the classification structure of the model
   a. Forward propagate $\widetilde{MB}_{labeled}$ through the encoder and get Z_1
   b. Forward propagate Z_1 through the classifier and get a classifier output p
   c. Back propagate gradient information ∇_w and update the classifier weights. No gradient is propagated back to the encoder and thus no encoder update is performed.

During each training step one mini-batch is sampled from the unlabeled data $X_{unlabeled}$ and another mini-batch is sampled from the labeled data $X_{labeled}$. The first mini-batch is used to perform a forward pass through the encoder, classifier and decoder entities. More specifically, the unsupervised learning mini-batch is corrupted with noise and forwarded to the encoder producing the latent representation Z_1. Then Z_1 is passed through the classifier obtaining feature mappings Z_2. Both representations are forwarded to the decoder structure and an output is produced. Afterwards, the unsupervised learning gradient information is calculated and used to update the decoder. The encoder parameters are updated without considering the gradients of the classifier. This means that only the gradients coming from the decoder are considered for the calculation of ∇_θ. No gradient information coming from the classifier is considered during updating the parameters of the encoder. On the contrary, the classifier weights are updated using the second mini-batch which is sampled from labeled data. Again, this mini-batch is corrupted with noise and forwarded through the encoder and then through the classifier producing a prediction. During training with labeled data, no information flows through the decoder. The supervised learning gradient information flows through the classifier and stops at its input layer which means that no weight update is performed at the encoder component of the model. This means that during the supervised learning training step, the encoder feeds the classifier with its output but is not updated. During the unsupervised learning step, the encoder is updated by gradient information estimated without the contribution of the classifier. On the other hand, the decoder structure of the model interacts with both the classifier and the encoder by using their outputs to construct an image and is only modified during the unsupervised learning step. Gradient information from the classifier is not used to update the encoder because this would compromise the denoising criterion and cause a failure in training the autoencoder. The probabilistic properties, the manifold-attracting-vector-field minimization property and the score-matching learning that enables the data distribution approximation principle of the denoising autoencoder cease to apply if the denoising criterion is violated. To preserve the criterion, the encoder should not adapt to the classifier's gradients since the latter is trained on a solely different objective function. On the contrary, the proposed gradient information flow dictates the encoder's adaption in terms of reconstructing the input in a way that is synergically compatible to the classifier mappings. In other words, the encoder adapts in a way that enables it to provide the decoder with information that complements the information provided by the classifier. During experiments a small fraction of the classifier's gradient information was

allowed to reach the encoder to test the possibility of increasing the performance of the model without clashing the denoising criterion mechanisms. It was observed that even a small fraction of such gradient information degrades the performance significantly but most importantly makes the MCMC sampling process inapplicable. These results confirm that the probabilistic properties of the model are degrading when the denoising criterion is even partially compromised.

5.4 Sampling the Generative Model

The model comprises a conditional generative autoencoder that given a preferred class y_p produces a sample x according to

$$p(x|y_p) = \frac{p(x)\, p(y = y_p|x)}{p(y=y_p)} \tag{5.3}$$

$$p(x|y_p) \propto p(x)\, p(y_p|x) \tag{5.4}$$

Applying the Metropolis-adjusted Langevin method [136, 137], allows sampling with a MCMC process that has an iterative step defined by

$$x_{t+1} = x_t + a\, \nabla_x \log_2 p(x_t|y = y_p) + N(0, \sigma^2) \tag{5.5}$$

$$= x_t + a\, \nabla_x \log_2 p(x_t) + a\, \nabla_x \log_2 p(y = y_p|x_t) + N(0, \sigma^2) \tag{5.6}$$

Alain and Bengio [101] proved that denoising autoencoders trained with Gaussian noise of variance σ^2 are able to approximate the gradient of the log probability of the data distribution $\log p(x)$ such as

$$\frac{\partial \log_2 p(x_t)}{\partial x} \approx \frac{x_t' - x_t}{\sigma^2} \tag{5.7}$$

where x_t' is the reconstructed output of the autoencoder given an input x_t corrupted with noise. Using this property of denoising autoencoders into the Makov chain sampling and setting $a = \sigma^2$ the update step becomes

$$x_{t+1} = x_t' + \sigma^2\, \nabla_x \log_2 p(y = y_p|x_t) + N(0, \sigma^2) \tag{5.8}$$

where the reconstructed output x_t' is provided by the mechanisms of the denoising autoencoder given an input x_t

$$\tilde{x}_t \sim C\left(\tilde{X} \middle| X_t\right),\ z_t \sim Q_\theta(Z \mid \tilde{X}_t)\ ,\ x_t' \sim P_\varphi(X|Z_t) \tag{5.9}$$

The latent distribution $z_t = \{Z_1\ , Z_2\}$ is formed by merging the contribution of the encoder Z_1 with the internal representations of the classifier Z_2 . By providing information to the decoder, the classifier influences the sampling process. This influence is better understood by looking at Z_2 not as an extra latent representation component supplied to the decoder but as a feature vector of the encoder's output Z_1. According to this point of view the encoding of an input could be merged with the activations of a similar (in terms of Euclidian distance) image of the preferable class and be decoded to an output that is closer to the desired class. The activations or feature mappings must be in accordance to the input data and cannot be arbitrarily enforced during the sampling process. In other words, they must be calculated using the input data and the chosen class of the generated outcome. One way to apply a class preference by inducing appropriate features for Z_2 and do not cause a clash in the operation of the autoencoder is to strategically modify the input data in favor of the specific class. This is achieved using an activation-maximization operation by keeping the weights of the classifier fixed and using gradient ascent with backpropagation to modify the input of the model in such a way that increases the activation of the output neuron of the classifier that corresponds to the targeted class. The gradient information necessary for such an operation is the derivative of the desired output in respect to the input pattern. This gradient ascent operation completes the Metropolis-adjusted Langevin sampling process as follows

$$t = 0: \quad x_0 \sim U(0,1)$$

$$t > 0: \quad x_t = x_{t-1} + \sigma^2\, \nabla_{\mathrm{x}}\, \log_2\, p\left(y = y_p \,\middle|\, x_{t-1}\right) \tag{5.10}$$

$$\tilde{x}_t \sim C\left(\tilde{X} \middle| x_t\right)$$

$$z_t \sim Q_\theta(Z|\tilde{x}_t),\ \ x_t \sim P_\varphi(X|z_t) \tag{5.11}$$

For the initial step of the sampling process ($t = 0$), a random image is sampled from a uniform distribution of the range [0,1]. This design decision considers two important requirements: compatibility with the data distribution as proposed by [122] and enough input variance that

allows diversity among the generated images. Drawing a random image from the specified uniform distribution for initiating the sampling process satisfies these requirements. Furthermore, a gradient ascent step modifies the input in accordance to the desired/undesired labels on every iteration of the sampling process. This step attracts the input closer to the feature distribution and effectively facilitates the decoder on outputting an image closer to the data manifold. Consequently, this relaxes the need for particularly good initial values. The Gaussian noise injected has enough variance and appropriate value range to initiate the sampling process even in the extreme case of choosing a zero initial input. However, using a zero-valued initial image results in producing samples of less diversity. Experiments conducted using various types of initialization schemes showed that sampling an initial input from a uniform distribution $U(0,1)$ provides good sample diversity and requires less sampling iterations before converging to a good quality sample. Once an initial input is chosen, for each process step forward, the previously generated output is modified using gradient ascent based on the desired/undesired class/classes and then used as an input to the model. This results to an input data modification in such a way that is a little more likely for it to belong to the desired class. The resulting data is corrupted with noise and then is propagated to the network and the next $(t+1)$ output is produced. There are various termination criteria for the sampling process such as convergence to a stable output or high classification confidence. However, theoretically, the autoencoder should be sampled several times before converging to a stable state. The sampling process is shown in Algorithm 5.2.

Algorithm 5.2 : Sampling process of the Conditional Generative Denoising Autoencoder according to specific dataset label(s).

Data: dataset label y_d

Result : generated sample after t sampling steps x'_t

---

1. Sample a random initial input $x \sim U(0, 1)$
2. Repeat until convergence (step t)
3. Apply gradient ascent in respect to x_{t-1} in order to increase the activation of the desired class y_d such as $x_t = x_{t-1} + \sigma^2 \nabla_x \log_2 p(y = y_d | x_{t-1})$.
   For samples to match more than one class ($y_{d1}, y_{d2}, \ldots$) use $x_t = x_{t-1} + \sigma^2 \nabla_x \log_2 p(y = y_{d1}, y = y_{d2}, \ldots | x_{t-1})$
4. Corrupt x_t with noise , $\tilde{x}_t = x_t + \mathcal{N}(0, \sigma^2)$
5. Run $\tilde{x}_t$ through the encoder to get new latent code $z_1^t \sim Q_\theta(Z \mid \tilde{X}_t)$
6. Apply latent code to the classifier to get internal representations z_2^t
7. Run $\{z_1^t, z_2^t\}$ through the decoder and get output x'_t
8. Assign $x_t = x'_t$

The gradient ascent step is demonstrated in more details in Figure 5.3. The input pattern is applied to the model and the gradient ascent step computes the gradient of the preferred labels in respect to the input. This gradient information is used to update the input pattern so that it is a little more likely to belong to the specific class or classes. Then, the modified input is corrupted with noise and passed through the model for outputting a reconstructed pattern. During each iteration, the value of the input is set equal to the pattern of the previous iteration output and the two basic steps shown in Figure 5.3 are repeated.

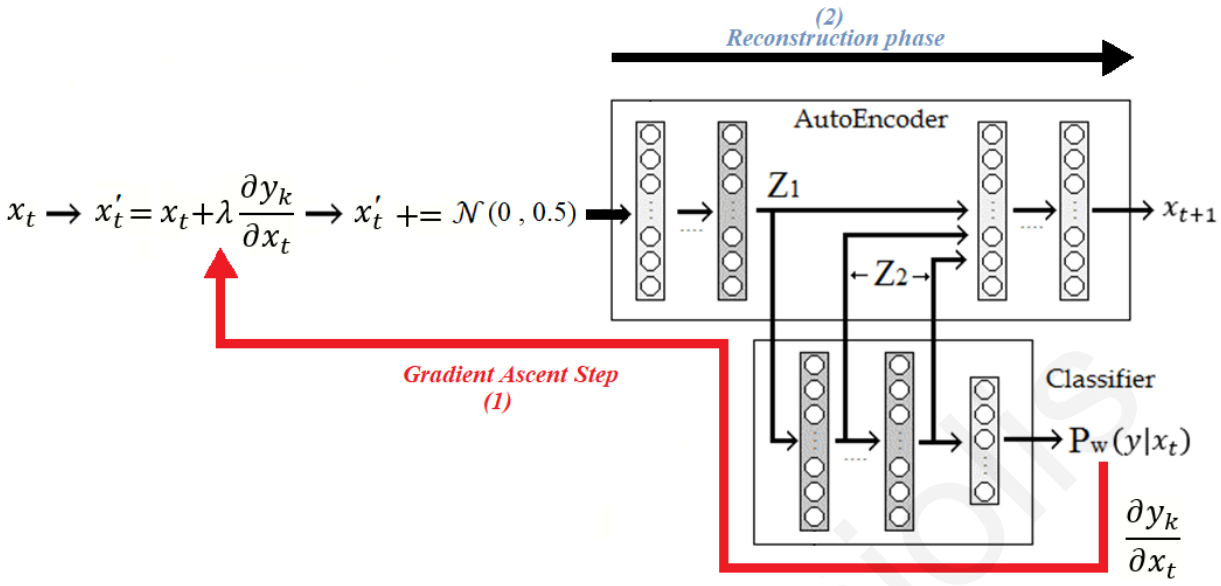


Figure 5.3. The two steps of the gradient ascent process. (1) The input is modified using the gradients of the labels of preference in respect to the input. (2) The modified pattern is noise-corrupted and forward propagated through the model to get the next sampling step output. Every iteration uses as input the output of the previous iteration of the MCMC process. Both steps are repeated during all iterations until the sampling procedure converges to the stationary distribution of the data conditioned on the preferred labels.

The sampling process takes an input of uniform noise and gradually transforms it to an image that belongs to the actual data distribution and has the characteristics reflected by the labels of preference used during the gradient ascent step. Figures 5.4 shows two illustrations of the sampling process steps and the way the initial random noise is evolving to meaningful images under the effect of some specific label of preference. The specific examples use a model trained on the CelebA (Celebrity Attributes) dataset and generate images according to the label "attractive". At the beginning of the sampling process the gradient ascent step provides a general form to the noisy input but as each of the sampling steps makes small adjustments towards the data manifold these adjustments take the form of detailed characteristics. At some point, the samples are drawn from an equilibrium distribution that is approximately equal to the distribution of the actual data. Any samples drawn from that point on, belong to this stationary distribution and have the required quality.

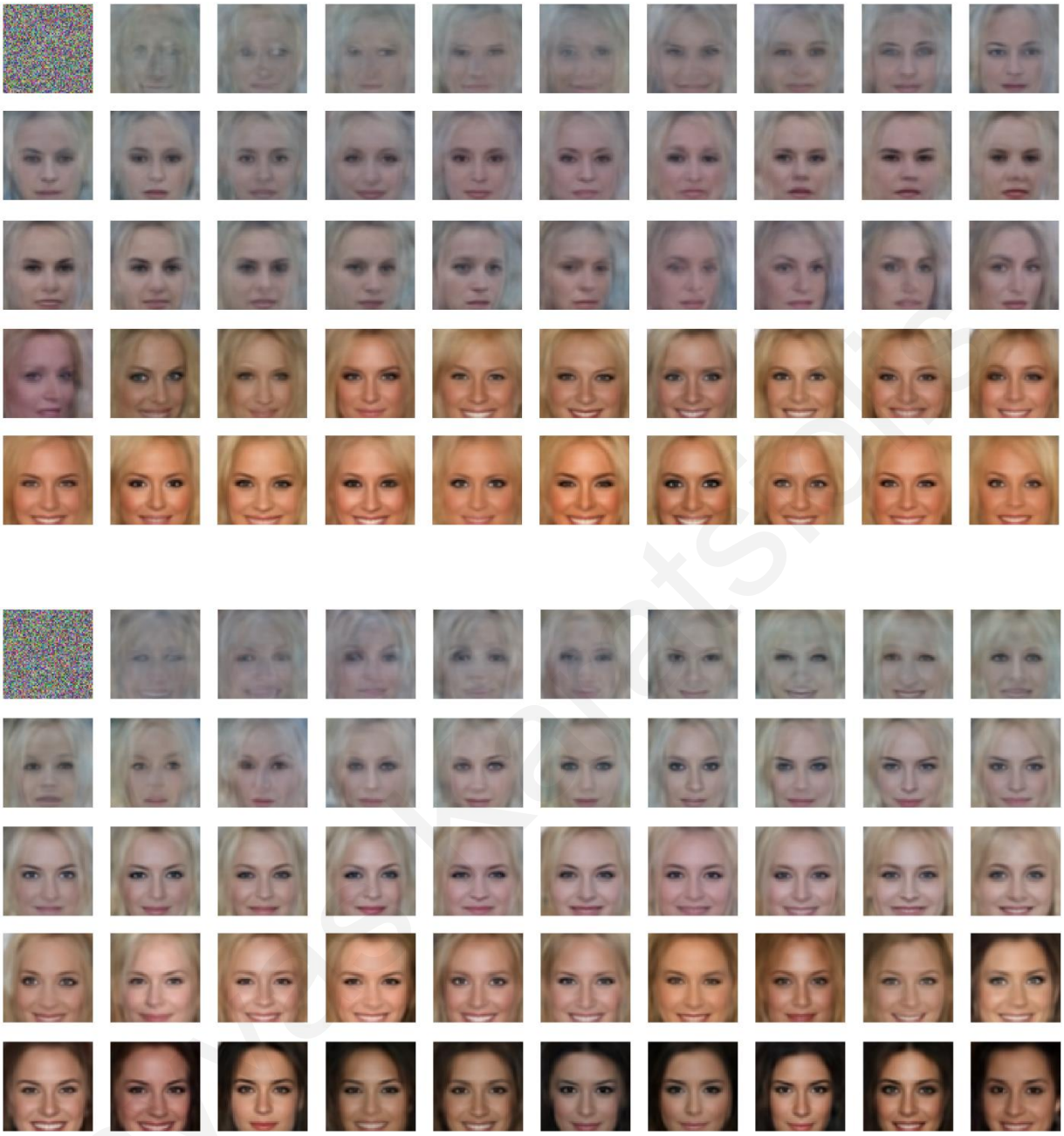


Figure 5.4. The evolution of initial random noise towards meaningful images during the sampling process of the CG-DAE. Both sampling processes use a model trained on the CelebA dataset and the "attractive" label. Initial noise turns from unmeaningful pixel-values to structures that reflect the label used during the sampling process. During the initial steps of the sampling sequence, we observe the formation of some high-level structures and eventually some more detailed components appear in the images. The sampling process approaches the actual data distribution at the beginning of the fourth rows for both image generations. Every new sample drawn after that point is different from the previous sample but still belongs to the stationary distribution reached by the MCMC process.

5.5 Experimental Results

Three popular datasets are used to test the proposed model: MNIST, Street View Home Numbers (SVHN) and Celebrity faces attributes (CelebA). The MNIST dataset is composed of 60000 grayscale images of handwritten digits and each pattern has a size of 28x28 pixels. The SVHN is a real-world image dataset obtained from house numbers in Google Street View images. Each color image has a size of 32x32 pixels and the dataset contains 73257 images. CelebFaces Attributes Dataset is a large-scale face attributes dataset with more than 200K celebrity images, each with 40 attribute annotations. The color images in this dataset cover large pose variations and background clutter and have a size of 218x178 pixels. To reduce memory and computational requirements, the CelebA images are initially cropped to a size of 108x108 and then resized to a 64x64 format while the other two datasets are kept in their original size. Samples from the three datasets are shown in Figure 5.5.

For the MNIST and the SVHN datasets, an autoencoder of three hidden layers (2 hidden layers for the encoder and 1 hidden layer for the decoder structure) is used along with a two hidden layer classifier. Each hidden layer of the model is composed of 1000 neurons. The specific architecture is shown in Figure 5.1. Since training a model with the CelebA dataset is more challenging, a bigger network is used. More specifically, a 6 hidden layer autoencoder with a 3 hidden layer classifier is put into effect and each layer is composed of 1200 neurons. Again, the input of the classifier is supplied by the output of the 3rd hidden layer (the encoder output) of the autoencoder while the outputs of the classifier's hidden layers are also forwarded to the 4th hidden layer of the Autoencoder which is the first decoder layer. For all cases, the fully connected network is trained with the ADAM optimizer and dropout is applied at the classifier component of the model. All neurons use the hyperbolic tangent activation function except from the output layer of the Autoencoder which uses sigmoid activations and the output layer of the classifier that uses either soft-max or sigmoid neurons. Experimentation revealed that even if the problem classes are mutually exclusive, models using the logistic function at their classifier's output layer produce more realistic images than the case of using the soft-max activation function. This implies that the cooperation of the Autoencoder and the classifier during training is better when the gradients of the classifier are not exaggerated by a soft-max activation which tends to reinforce the correct output while suppressing all other outputs. Furthermore, using a soft-max at the output of the classifier seems to slightly derail the training of the Autoencoder resulting in longer training times and slightly worse results. This is also

supported by the fact that using soft-max at the output stage of the classifier produces a model that tends to generate biased images in the sense that they lack broad diversity. The choice of using hyperbolic tangent activations considers the gradient ascent step performed during each sampling iteration. The specific activation function is differentiable in its whole value-range in contrast to the ReLU (rectified linear unit) activation that has a discontinuity at zero. It also extends to negative values and has a zero mean which makes it more suitable for performing the gradient ascent updates for applying a preference or disfavor for a specific set of labels. The networks are initialized using the Xavier method [110] and data is supplied in the form of mini-batches of 128 images each. The initial learning rates for the CelebA dataset was 1×10^{-4} and 1×10^{-3} for the other datasets. During training, the learning rate was reduced by an order of magnitude every time the error flattened for a significant amount of time. Some samples from the original datasets (Figure 5.5) and several samples generated by the trained models are shown below (Figure 5.6 for MNIST-SVHN and Figure 5.7 for CelebA). The generated samples are randomly selected and are not cherry picked by any means. The samples shown in the specific figures are generated using one class label which is self-evident for the MNIST and the SVHN samples and thus is not noted in the figures. However, for the CelebA samples, the applied label is shown on the left side of the generated images and the full label list is shown in Table 5.1.

Figure 5.5: Samples from the original datasets: MNIST (top and left), SVHN (top and right) and cropped-resized CelebA (bottom).

TABLE 5.1

Labels of the Celeba Dataset

1	5oClock Shadow	**21**	Male
2	Arched Eyebrows	**22**	Mouth Slightly Open
3	Attractive	**23**	Mustache
4	Bags Under Eyes	**24**	Narrow Eyes
5	Bald	**25**	No Beard
6	Bangs	**26**	Oval Face
7	Big Lips	**27**	Pale Skin
8	Big Nose	**28**	Pointy Nose
9	Black Hair	**29**	Receding Hairline
10	Blond Hair	**30**	Rosy Cheeks
11	Blurry	**31**	Sideburns
12	Brown Hair	**32**	Smiling
13	Bushy Eyebrows	**33**	Straight Hair
14	Chubby	**34**	Wavy Hair
15	Double Chin	**35**	Wearing Earrings
16	Eyeglasses	**36**	Wearing Hat
17	Goatee	**37**	Wearing Lipstick
18	Gray Hair	**38**	Wearing Necklace
19	Heavy Makeup	**39**	Wearing Necktie
20	High Cheekbones	**40**	Young

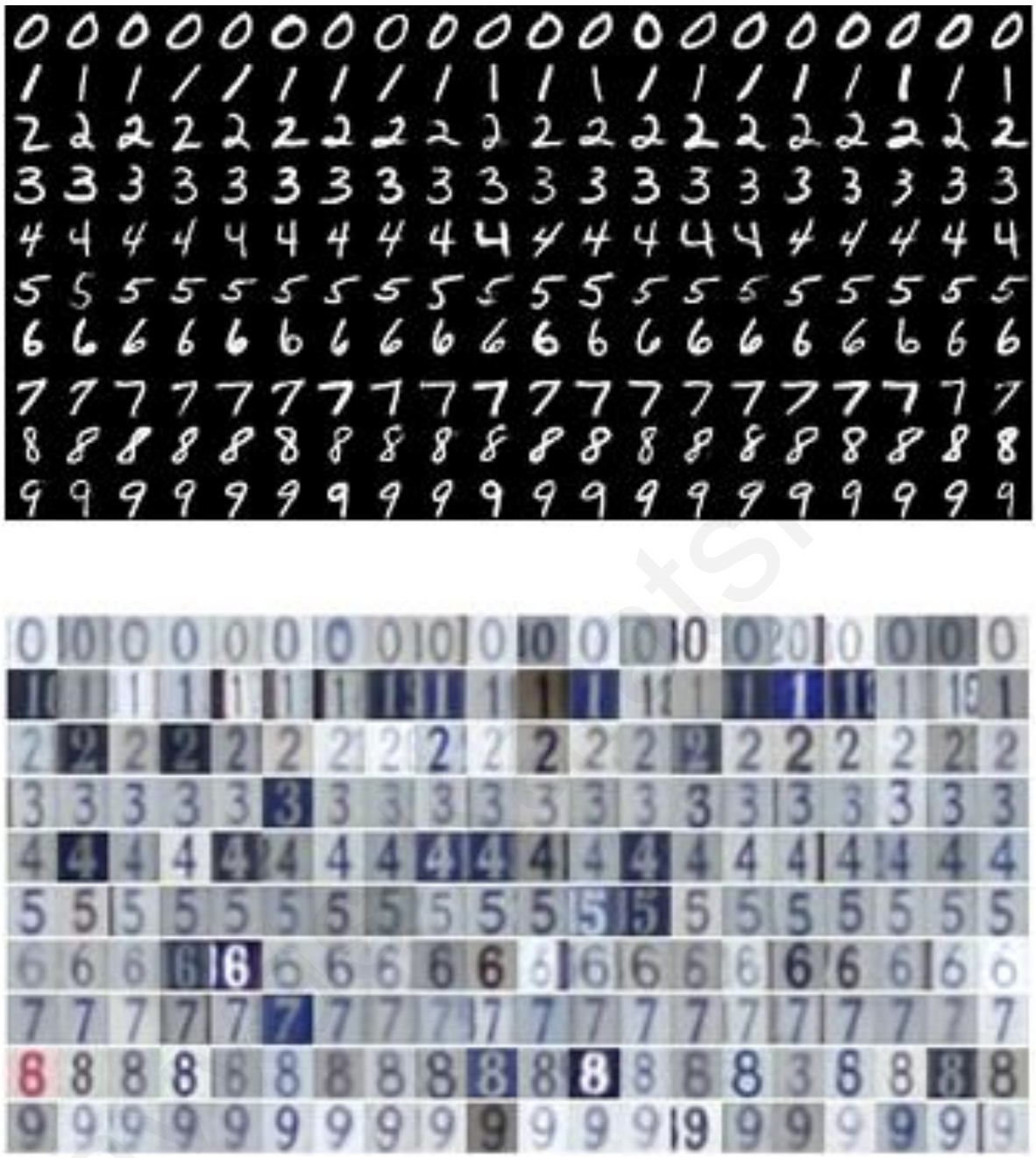
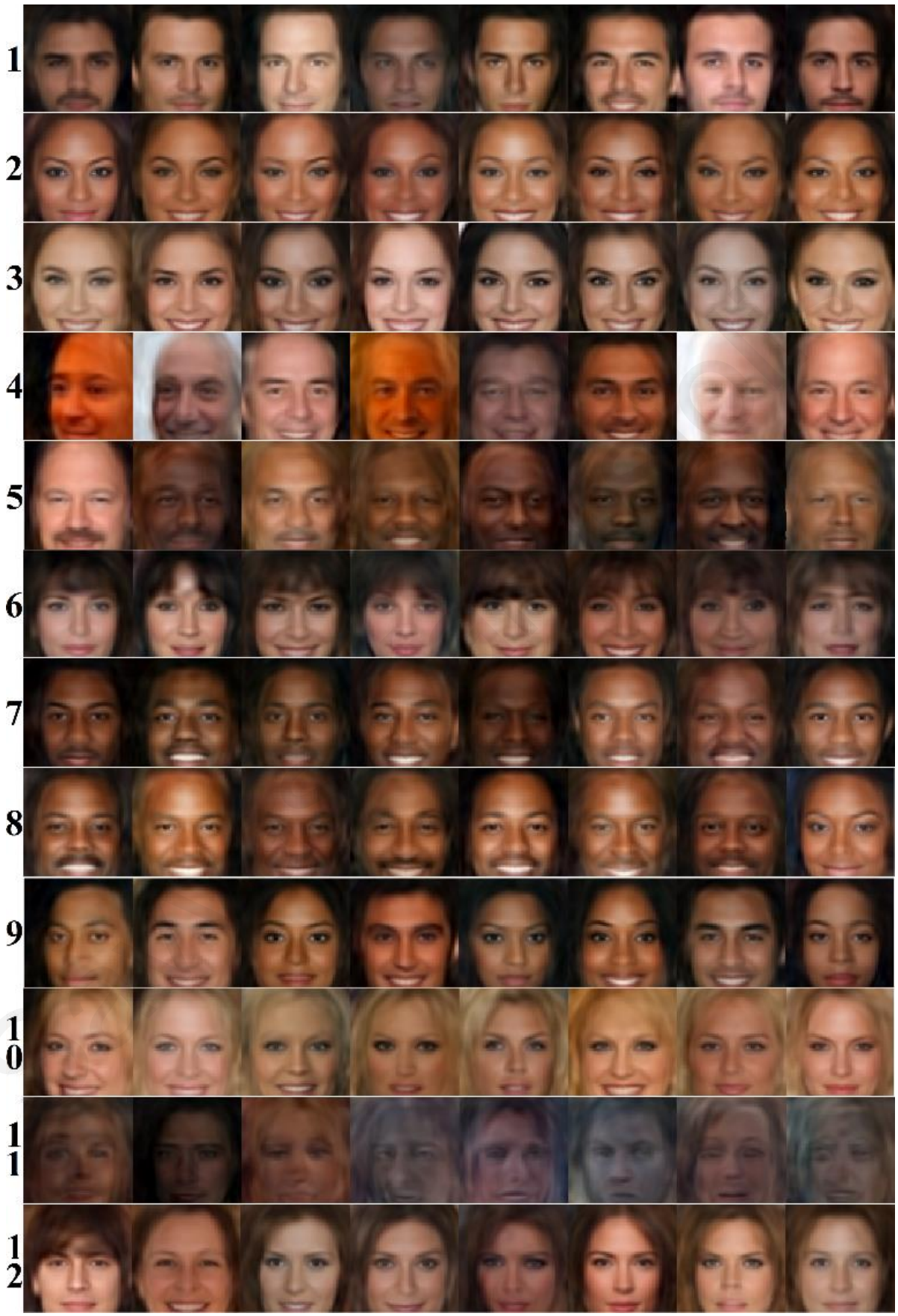
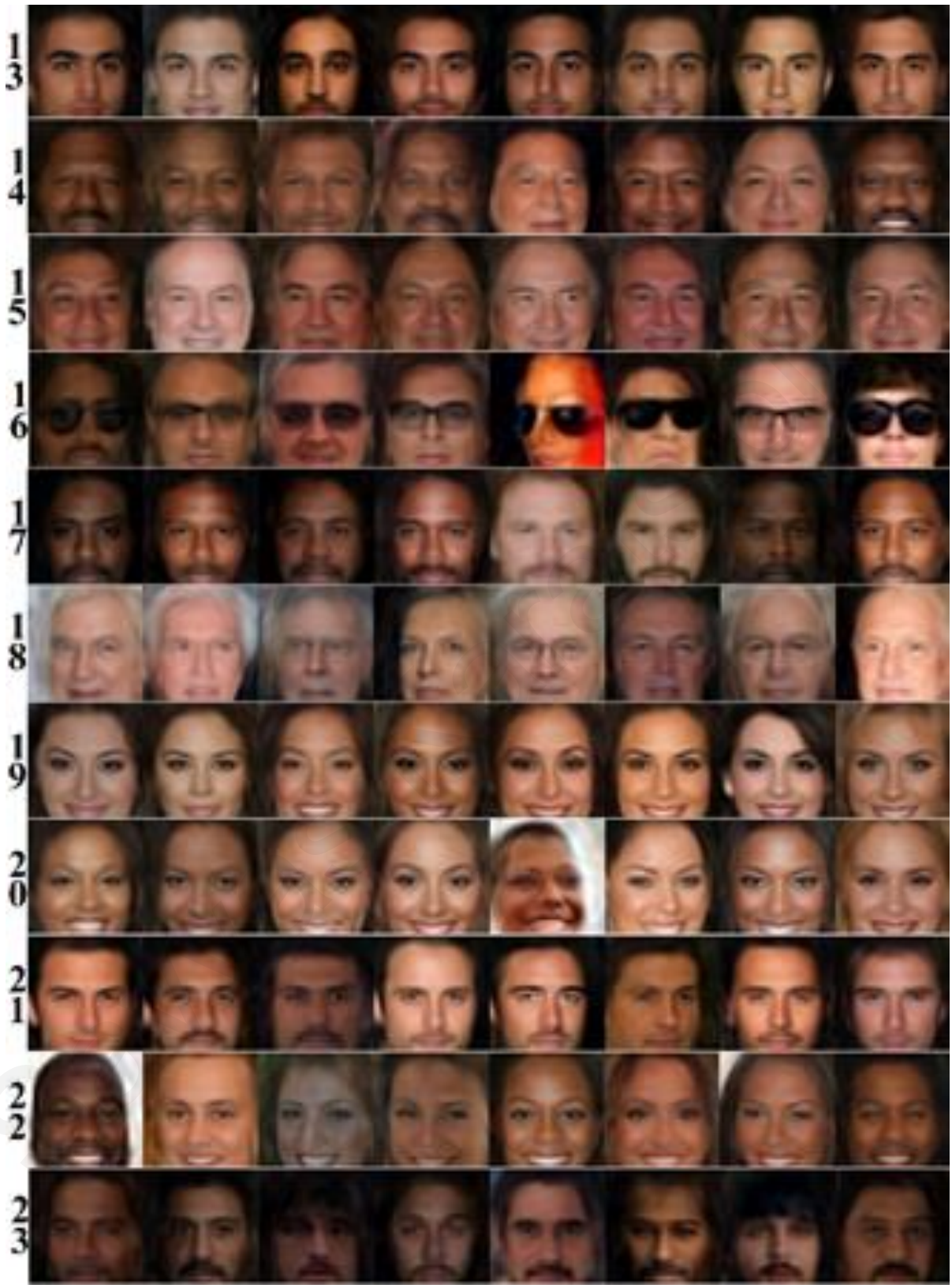


Figure 5.6: Samples from the generative models for the MNIST (top image) and the SHVN (bottom image) datasets.


1
2
3
4
5
6
7
8
9
10
11
12

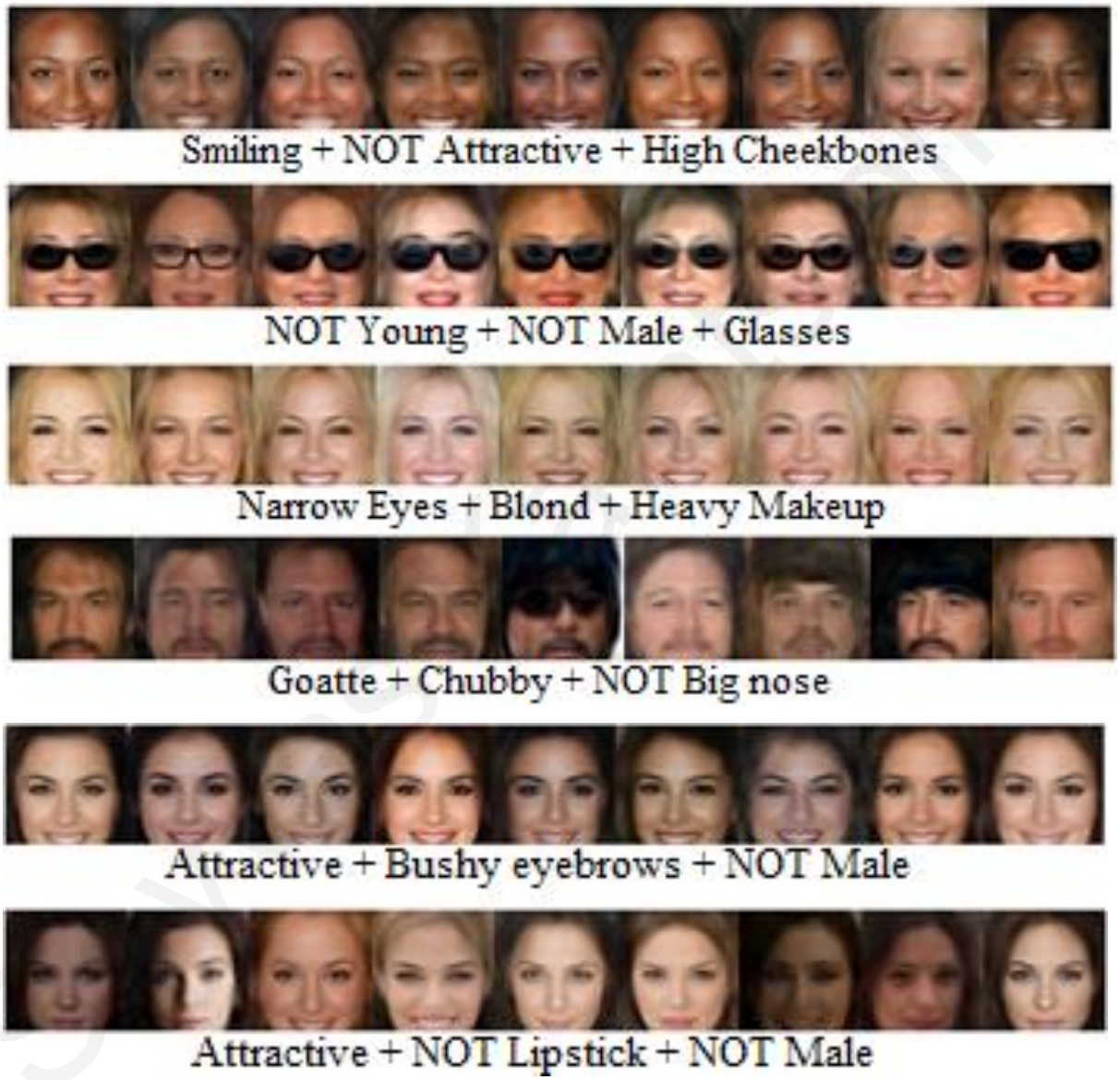

13
14
15
16
17
18
19
20
21
22
23


24
25
26
27
28
29
30
31
32
33
34

Figure 5.7: Generated samples from the model trained on the CelebA dataset. Each row corresponds to a specific class of data indicated by a number label on its left side. The label-class associations are shown in Table 5.1.

Problem labels can be considered as properties that a generated image may or may not possess. This is especially applicable in the case of the CelebA generative model since the labels of the specific dataset are not mutually exclusive. For example, a male person (label number 21) may also wear glasses (label number 16) or a young person (label number 40) may also be smiling (label number 32) and have a big nose (label number 8) at the same time. Accordingly, a person may wear glasses and not have brown hair. Some image properties may be specifically requested to be absent in the generated image by having that label output provide negative gradient information (instead of positive) during the sampling process. More specifically, during model sampling the classifier performs steps of gradient descent in respect to an undesired class instead of gradient ascent updates. Several generated images using desired and undesired properties are shown in Figure 5.8. The undesirable properties are marked with a “NOT” keyword and the

combination process merging the sampling requirements is illustrated by a "+" sign. In practice, labels can be characterized as 'desired', 'undesired' and 'don't care'. Respectively, the gradient of each class-output in respect to the input during the sampling process is positive, negative or zero.



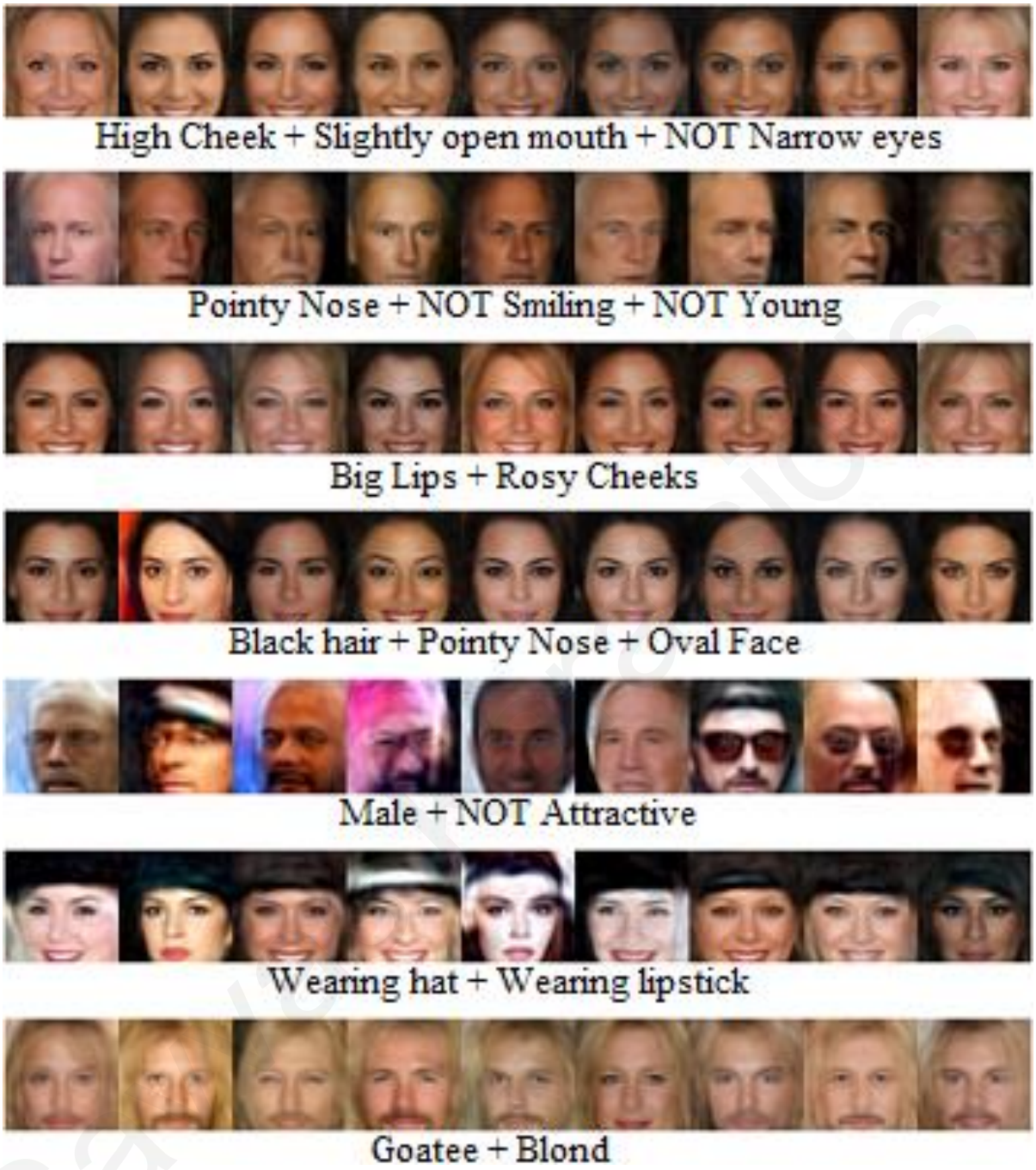


Figure 5.8: Samples generated using multiple labels for the CelebA dataset. The applied labels may imply a positive or a negative effect on the sample and the negative property is indicated by a "NOT" keyword in front of the associated property.

To further evaluate the performance of the model, linear latent space interpolation is applied in order to observe the transition between the generated images in terms of smoothness and cosmetic appearance. An initial and a transitional image are randomly selected from the dataset.

The two images are passed through the model and their latent representation $\{Z_1, Z_2\}$ is noted. Starting from the latent vector of the initial image and interpolating towards the target latent vector, each generated image is collected. Figure 5.9 shows several linear interpolations with the first image of each sequence being the initial image and the last image being the transitional target. The images in-between reveal how the model transitions between internal representations and how it practically responds to small variations starting from one concept and moving towards a different one.

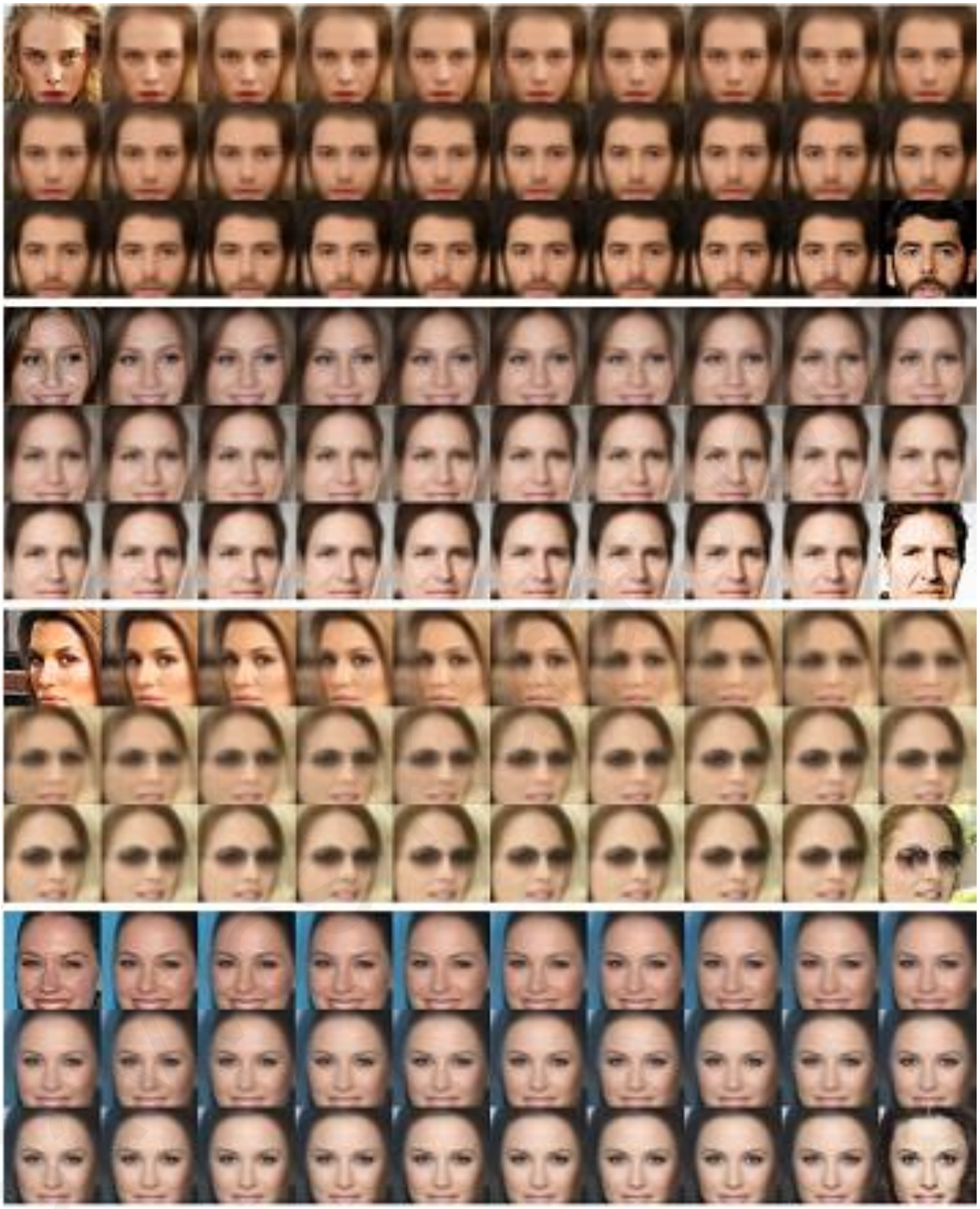


Figure 5.9: Several latent space interpolations for the model trained on the CelebA dataset. The first image of each sequence is the initial image while the last image of the sequence is the target image. The transitions of the in-between images towards the final image are quite smooth.

5.6 Convolutional CG-DAE

A pure convolutional implementation of the CG-DAE is not expected to be as efficient as the fully connected version of the algorithm. This is due to the principle of the local receptive field which applies in convolutional architectures. According to this principle, every neuron in the feature maps has a constrained visibility regarding the data carried by the previous feature maps and the input image. When performing the gradient ascent step the input image is modified such as to increase feature detection by a large number of filters (convolutional kernels) without obeying a strong collaboration regime like the one found in fully connected architectures. Gradient ascent updates the input image in order to strengthen the attributes that the convolutional kernels are expecting to match for a given class. For example, a mustache detector formed by higher-level feature map activations provides a positive detection signal if the structure of a moustache is found anywhere in the image. This means that spatial coherence is not directly applied as in the case of fully connected models where each neuron depends on the activation of every previous neuron in the network. In convolutional CG-DAE spatial coherence is only implied by the image attributes that increase high-level feature detection: detecting an eye implies increased probability of locating another eye at about the same height. In fully connected models this is self-evident by allowing the detection signal of each neuron to influence all next neurons in the model. In convolutional models this is just implied by learned higher-level structures that represent low detail human faces and need to be active when making the gradient ascent step for generating human faces. The problem is that higher-level structure representations formed in the feature maps of convolutional neural networks are automatically learned during training and cannot be directly accessed (or hand-engineered). Since these high-level structures are responsible for providing spatial coherence in generating images, it is very difficult to account for every different configuration in the input data. The basic structures are easily regenerated while structures of high variability like hair or beards are very difficult to generate.

In fully connected models each neuron contributes to the gradient ascent step by acting as a collaborative feature detector that accounts for information found in every input pixel. This means that it calculates its activation considering signals coming from all neurons of the previous layer. In convolutional neural networks each neuron represents the application of a specific kernel on an area of a previous feature map and thus the gradient ascent step is performed in a self-contained way neglecting information from other regions of the feature

maps. The gradient ascent step essentially increases the detection signals of the convolutional kernels. This allows a great degree of freedom towards obtaining an updated image that satisfies the increased feature detection requirement but at the same time the updated image may greatly deviate from the data distribution. For example, a gradient ascent step using a label “wearing hat” may easily produce an image that is comprised of many hats or hat components because this greatly satisfies the urge of the kernel for detecting the specific attribute anywhere in the image. On the other hand, fully connected networks must account for the interaction of a hat with all other image components so it is highly unlikely to produce such an image. This peculiarity based on the increased degree of freedom that convolutional models possess is the reason that they produce unexpectedly good results at classification tasks when the convolutional kernels are just randomly chosen instead of being learned parameters while the only trainable layer(s) in the model is the final stage classifier [9]. Random kernels still match various useful features because of the nature of the convolution process and its flexibility in detecting meaningful structures.

Regardless their natural weaknesses in performing a “proper” gradient ascent step, convolutional CG-DAE models are able to generate images based on labels. However, they fail to provide fine details on components of high variability like hair. The architecture used in the experiments is described in Figure 5.10 and Tables 5.2, 5.3 and 5.4.

TABLE 5.2

The architecture of the Classifier structure of the convolution CG-DAE model. Each layer has a size of 1024 neurons except the final layer which has 40 neurons matching the number of the problem classes

CLASSIFIER	
LAYER	OUTPUT
FC1	1024
FC2	1024
FC3	1024
OUT	40

TABLE 5.3

The architecture of the Encoder structure of the convolution CG-DAE model. The kernel of each convolutional layer is shown in brackets. The middle column indicates whether down sampling is implemented in the specific layer for reducing the size of the feature maps by a factor of 2. Each layer uses 256 convolutional kernels.

ENCODER		
LAYER	RESAMPLE	OUTPUT
$X + \mathcal{N}(0, \sigma)$	-	$\hat{X}$
Conv1 [5,5]	D	$32 \times 32 \times 256$
Conv2 [3,3]	D	$16 \times 16 \times 256$
Conv3 [3,3]	-	$16 \times 16 \times 256$
Conv4 [3,3]	-	$16 \times 16 \times 256$
Conv5 [3,3]	D	$8 \times 8 \times 256$
Conv6 [3,3]	D	$4 \times 4 \times 256$
Conv7 [3,3]	-	$4 \times 4 \times 256$

TABLE 5.4

The architecture of the Decoder structure of the convolution CG-DAE model. The kernel of each convolutional layer is shown in brackets. The middle column indicates whether up sampling is implemented in the specific layer for expanding the size of the feature maps by a factor of 2.

DECODER		
LAYER	RESAMPLE	OUTPUT
DeConv1 [3,3]	U	$8 \times 8 \times 256$
Conv2 [3,3]	-	$8 \times 8 \times 256$
DeConv3 [3,3]	U	$16 \times 16 \times 256$
Conv4 [3,3]	-	$16 \times 16 \times 256$
DeConv5 [3,3]	U	$32 \times 32 \times 256$
Conv6 [3,3]	-	$32 \times 32 \times 256$
DeConv7 [3,3]	U	$64 \times 64 \times 256$
Conv8 [3,3]	-	$64 \times 64 \times 3$

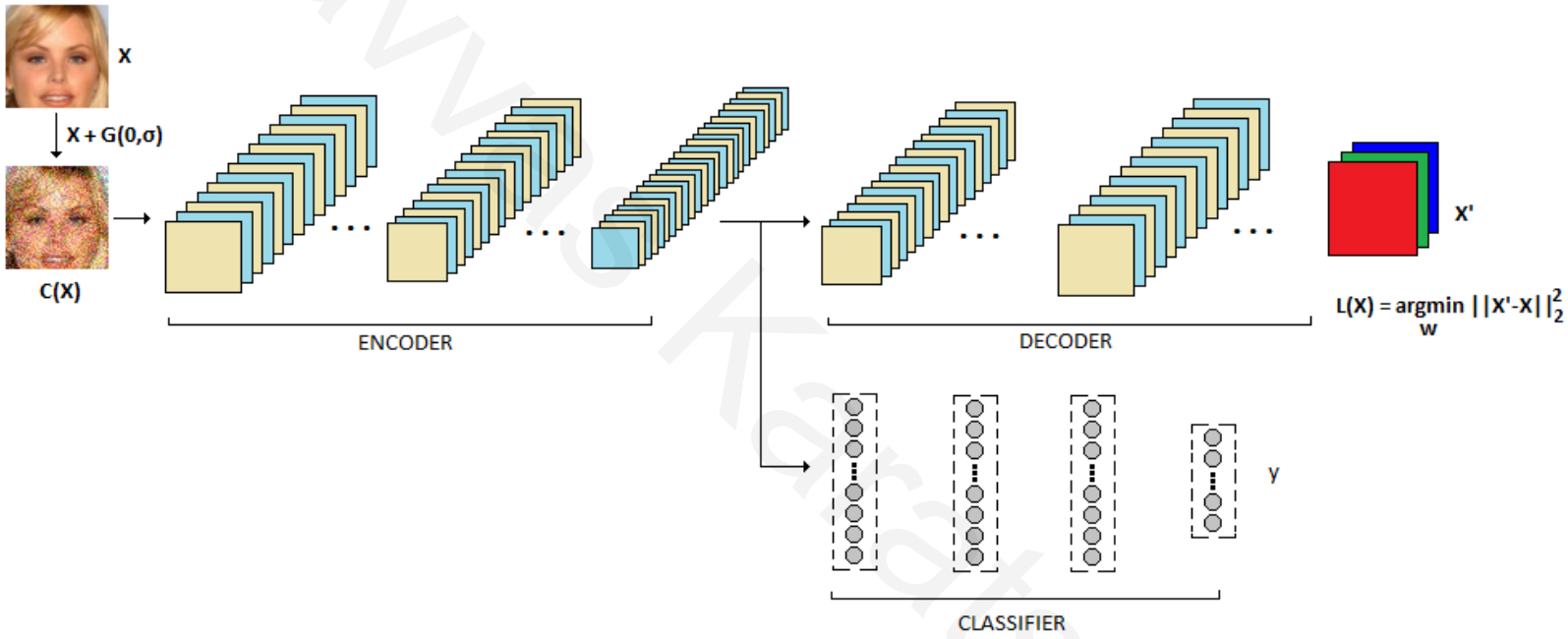
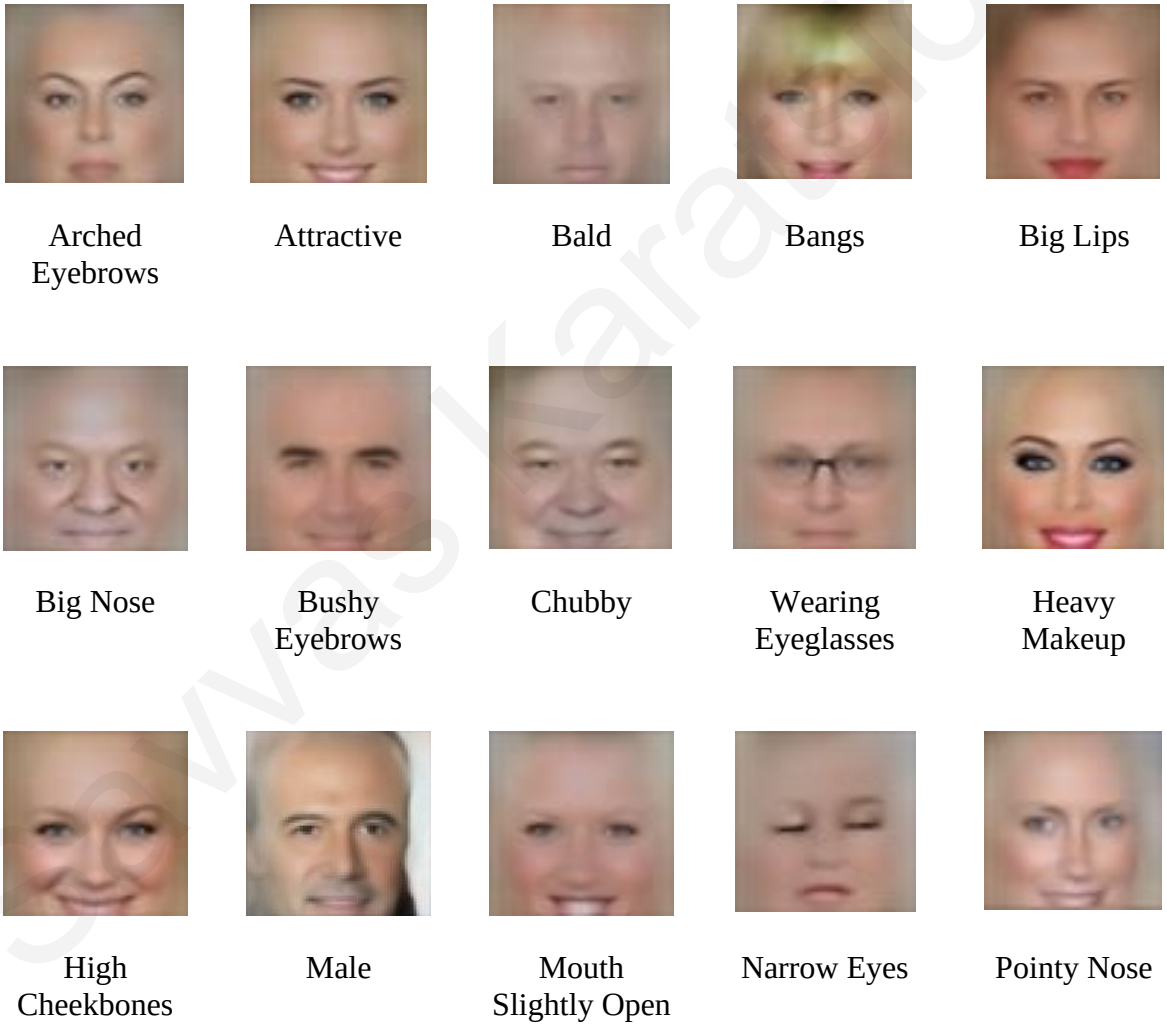


Figure 5.10: The architecture used for implementing the convolutional flavor of the CG-DAE. Like in the fully connected version, the input image is corrupted with Gaussian noise and passed through several convolutional layers to obtain the embeddings. The embeddings are forwarded to the classifier and the decoder structure in order to get the classification output and the reconstructed image respectively. The reconstruction error is the MSE between the original(uncorrupted) image and the output image of the decoder.

5.6.1 Samples drawn from the convolutional CG-DAE

Indicative samples drawn from the convolutional version of the CG-DAE method are shown in Figure 5.11. A close investigation of the samples reveals the tendency of the convolutional model to exaggerate on the features that relate to the applied labels. For example, when applying the "Narrow eyes" label, the model tends to generate images of persons having fully closed eyes. This tendency for exaggeration seems to come at a cost related to the regions of the images that are less important or require a higher degree of detail like the edges of the faces or hair.



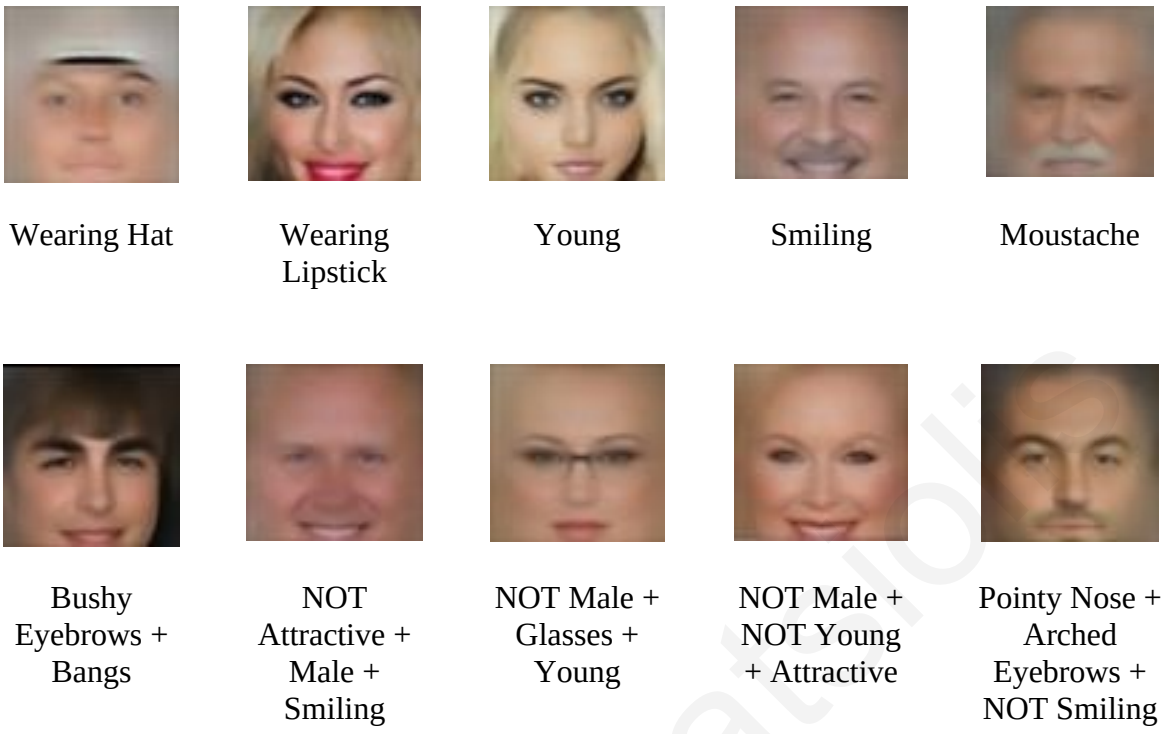


Figure 5.11: Samples drawn from the convolutional CG-DAE. It is evident that the convolutional version tends to exaggerate on the characteristics of the images at the cost of more blurry reconstruction of regions that are of less importance like hair and face edges.

The model seems to greedily trying to activate the output of its detectors (convolutional kernels) individually without applying any degree of collaboration in the way the gradient ascent step is performed. The most profound impact of this behavior is the small diversity of the drawn samples. Using the same magnitude learning rate during the ascent step and the same magnitude for the noise corrupting process produces similar samples. Sample diversity can be improved by using different learning rates for the gradient ascent step.

An analogues reasoning applies for using a convolutional decoder instead of fully connected architecture. The deconvolutional operations fixate themselves on certain detectable sparse patterns introducing the tendency to generate images of low diversity. To conform this belief, a different experiment is conducted by constraining the gradient ascent step only up to the embeddings' layer, namely the output of the encoder. This strategy allows for an update guided only by fully connected layers in order to test the belief that the convolutional decoder is also contributing to the reduced image diversity. The sampling process is modified as shown in

Algorithm 5.3. The algorithm ensures that only fully connected layers contribute to the gradient information and allows for studying the behavior of the convolutional model without the concern of performing a gradient ascent step through a convolutional network path.

Algorithm 5.3 : The modified sampling process for applying the gradient ascent step up to the encoder's output.

Data: dataset label y_d

Result : generated sample after t sampling steps x'_t

---

1. Sample a random embeddings vector $z \sim U[-1, +1]$
2. Repeat until convergence (step t)
    3. Apply gradient ascent in respect to z_{t-1} in order to increase the activation of the desired class y_d such as $z_t = z_{t-1} + \sigma^2 \nabla_z \log_2 \mathrm{p}\,(\, y = y_d \,|z_{t-1})$.
    For samples to match more than one class (y_{d1} , y_{d2} , ...) use
    $z_t = z_{t-1} + \sigma^2 \nabla_z \log_2 \mathrm{p}\,(\, y = y_{d1,}\, y = y_{d2}\, , \ldots \,|z_{t-1})$
    4. Run z_t through the decoder an get new output image x_t
5. Run $x_t + N(0, \sigma)$ through the encoder to get new latent code $z_t \sim Q_\theta(Z \mid x_t)$

The latent code is randomly initialized in the range $[-1, +1]$ to reflect the possible values the hyperbolic tangent units of the encoder's output layer can output. Noise is added to the reconstructed image x_t before it is used to calculate the new embeddings' values z_t. Samples generated using the above strategy have similar characteristics as the ones generated using a gradient ascent step reaching the network's input layer. This confirms that the gradient ascent step through the convolutional layers is not the only factor contributing to the lower image diversity. The use of a convolutional decoder also plays a role in generating biased samples.

5.7 Discussion – analysis

5.7.1 Related Work and comparative results

The idea of combining a generator and a classifier to effectively train the former has been explored before. Generative adversarial networks (GANs) [21] use a discriminator attached to the output of the generator in order to back-propagate gradient information that trains the generator to produce realistic images. The discriminator in its original form tries to classify true and fake (produced by the generator) images. Among several notable variations of the original GAN architecture was the proposed replacement of the single output (fake or true) discriminator with a classifier that is aware of all the problem classes [89]. In this case the classifier has K+1 outputs with K of them being the problem classes and the remaining output corresponding to the "fake" class. The conditional version of GANs (CGAN) was explored by Mirza and Osindero [87]. To convert a GAN to its conditional version they added an extra input to both the generator and the discriminator that holds the desired label. This enables the construction of a multi-modal model able to generate samples conditioned on a specific label. The Parzen window log-likelihood estimate for the MNIST test data of our model surpasses the performance reported by GAN [21], CGAN [87], DBN (Deep Belief Network) [133] and GSN (Generative Stochastic Network) [138]. The estimate is constructed by drawing 1000 samples from each problem class and fit the samples with a Gaussian Parzen window as described in [131]. The results of this comparison are shown in Table 5.6.

TABLE 5.6

Parzen-window log-likelihood estimates for the MNIST test data

MODEL	**LOG-LIKELIHOOD**
DBN	138 ± 2
Deep GSN	214 ± 1.1
GAN	225 ± 2
CGAN	132 ± 1.8
CG-DAE	$\mathbf{298 \pm 1.4}$

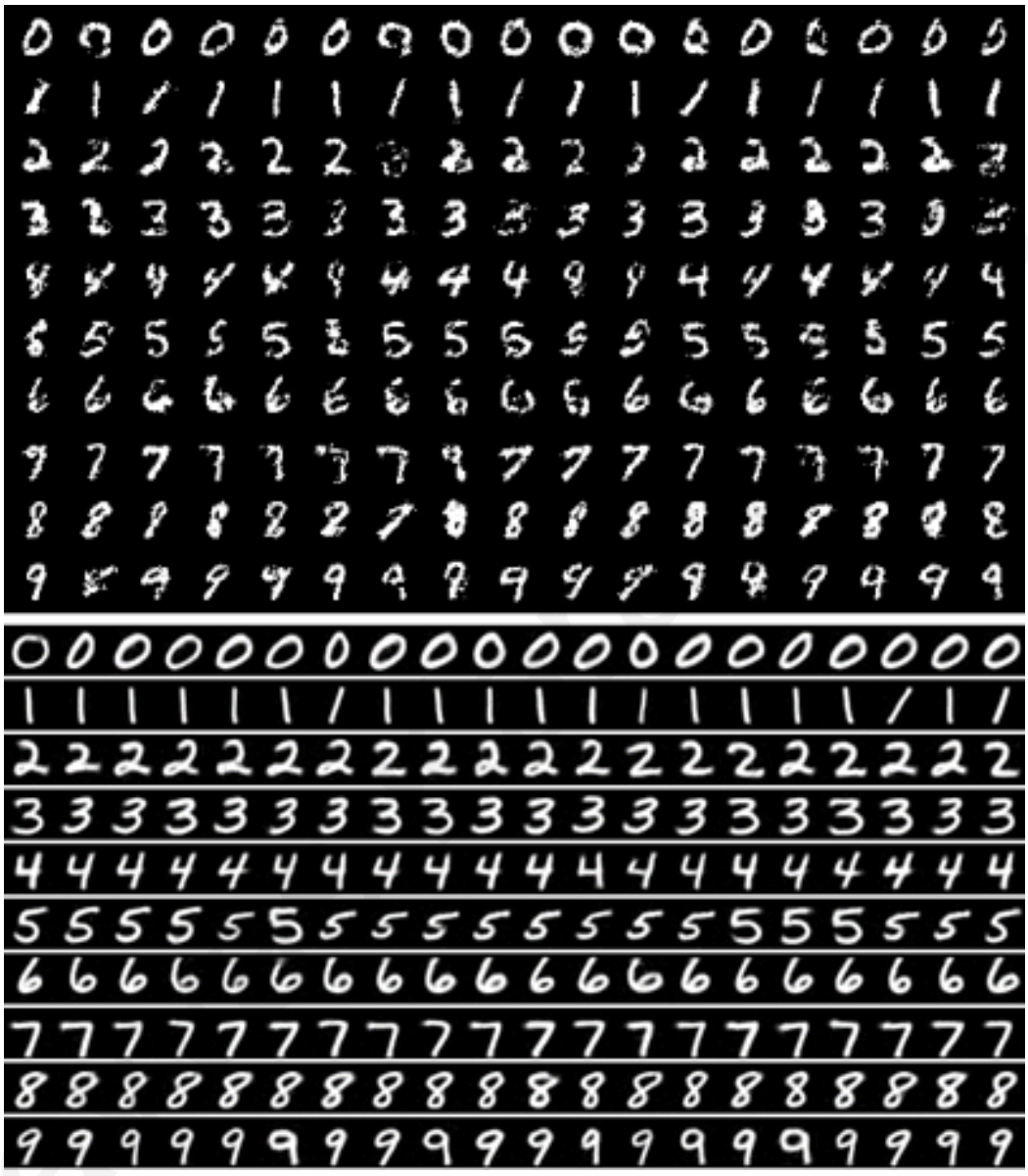


Figure 5.12: Samples generated by the CGAN model trained with the MNIST dataset are shown at the top of the figure [87] while samples generated from our model are shown at the bottom. Samples are grouped per problem class.

Figure 5.12 compares the patterns generated by the CGAN and CG-DAE. The significant difference in the log-likelihood estimates between our model and CGAN is clearly reflected in the quality of the generated samples. While GANs and CG-DAE only share the design

characteristic of employing a different-role classifier in their architecture, there are other methodologies proposed through the recent years that share more implementation and theoretic similarities. One such model by Dosovitskiy and Brox [139] uses a generator, a comparator and a discriminator and is trained with an objective function that considers perceptual similarity metrics reflecting feature loss, adversarial loss and image space loss (in terms of Euclidean distance). Strictly speaking, while these generators produce good results they do not comply with the theoretical requirements of generative models since they do not impose a prior on their latent representation nor they apply noise to their input to ensure 'data manifold learning' according to the denoising autoencoders theoretical framework [98]. In turn, this prevents them from obeying a sampling procedure. However, these generators were used by Nguyen et al [140] to supply a prior for producing preferred images using activation maximization in a deep architecture that connects an inverted deep generative network to a deep neural classifier. Activation maximization has also been used extensively, especially in the field of visualizing the representations learned by a deep neural network [141, 26, 22]. The main idea is to modify an input image into another image that indicates which input space structures excite the network feature mapping operations the most. In other words, the task is to discover the dominant image characteristics that mostly influence the functionality of the classifier network. The most concrete method proposed is the optimization method that uses gradient ascent to determine the directions of input modification in order to achieve activation increase in any neuron of the network. Activation maximization tends to create high frequency artifacts in the input images which becomes problematic when multi-step gradient ascent is used instead of a single step process. This can be solved by applying Gaussian blurring [141] between each iteration. According to CG-DAE's sampling process, Gaussian blurring is not necessary since each iterative step contains a noise corruption process which greatly decreases the hazard of building up such high frequency artifacts. CG-DAE applies activation maximization through an Autoencoder that relies its operation on the accompanying classifier that drives this process. In this way, directing the classifier towards increasing class-specific internal representations also drives the Autoencoder towards producing class-specific images.

The Variational Autoencoder performs well and is theoretically elegant. These qualities establish it as a frequent choice for building generative models. However, it suffers from two major drawbacks: its samples tend to be blurry and it does not utilize the latent space effectively in the sense that it tends to use only a fraction of the available latent dimensions (Goodfellow

2015). The blurriness in the generated samples of a Variational Autoencoder may be caused by the fact that the model minimizes $D_{KL}(Q_\theta(Z|X) \parallel P(Z))$ in order to perform maximum likelihood. This issue is also applicable to generative models that use log-likelihood optimization. The use of a Gaussian distribution for $P_\varphi(X|Z)$ also contributes to the problem. Several attempts have been made in order to improve its performance.

Hue et al [85] replace the pixel-wise loss of a Variational Autoencoder with a feature consistency loss. More specifically, they forward pass the original and the generated image through a VGG-19 network [43]. Then they evaluate the similarity of the intermediate layers feature mappings produced by the two images. This feature-perceptual-loss is measured in squared Euclidean distance and comprises the new loss used for training their deep feature consistent VAE. This approached produces two models that are based on the loss applied on different intermediate VGG layers: VAE123 applies the loss on the first 3 ReLU layers and VAE345 applies the loss on ReLU layers 3-5. Their results are compared with the results of CG-DAE in Figure 5.13. Junbo Zhao, Michael Mathieu and Yann LeCun introduced the "Energy-based generative adversarial network" [142] which interprets the discriminator as an energy function. Energy levels assigned to regions in the vicinity of the data manifold are low while all other regions are assigned a high energy. This drives the generator towards producing samples of very low energy and also allows a more stable training process. The results of the Energy Based Generative Adversarial Network (EBGAN) model are also compared to the results of CG-DAE in Figure 5.13.

PVAE

DCGAN

VAE-123

VAE-345

EBGAN

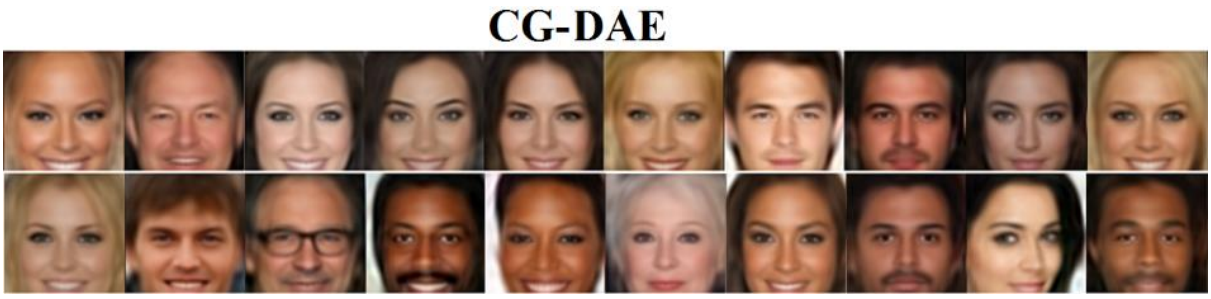


Figure 5.13: Comparative results of various methods with samples drawn from CG-DAE. The first two rows are images generated by a Plain Variational Autoencoder (PVAE). Generally, VAEs tend to produce blurry images due to the Gaussian prior enforced in the latent space which inevitably introduces a l_2 loss on the objective function [143]. The second pair of rows show the results of a deep convolutional GAN (DCGAN) [144]. The 5^{th} – 8^{th} rows show the results of the VAE123 and VAE345 models trained with feature consistent VAEs and the second last pair of rows shows the results of an Energy-based GAN. Samples from the CG-DAE are shown in the last pair of rows.

5.7.2 Exploring the connection between the classifier and the Autoencoder

The interaction between the Autoencoder and the classifier in the proposed model requires some deeper investigation in order to be sufficiently explained. In order to get an insight on the internal mechanisms of the model we pass images from the training set through the network and obtain the reconstructed images at the output after interchangeably masking the latent code of the encoder and the classifier. More specifically, we obtain the output ignoring the contribution of the classifier to the model's decoder by masking it out (making it zero) and also obtain the output ignoring the encoder's latent code by also masking it out. It is acknowledged that this process is not, theoretically speaking, suitable for isolating either component of the model since the contributions of each structural component (encoder or classifier) are normally combined to produce a linear activation followed by a nonlinear function in the first layer of the decoder. Consequently, masking out a model component's latent code does not produce an output image that specifically represents what the non-masked component actually contributes to the reconstruction process. Nevertheless, we consider this experiment as having some useful value in trying to study the association of the model's components. Figure 5.14 shows the results of masking out (omitting) either the classifier or the encoder's contributions to the decoder. The input images are provided to the network in their original form without the addition of noise. Figure 5.15 shows the case that the images are corrupted by a significant amount of noise. The

first column holds the original image, the second column the reconstructed output with the encoder's output masked, the third column the reconstructed output with the classifier masked and the final column is the combined output with no masking on either component.

Figure 5.14: The output of the model after masking out (omitting) either the contribution of the classifier (3rd column) or the contribution of the encoder (2nd column). The normal output without any masking is shown in the 4th column.

Figure 5.15: The output of the model when the input is corrupted with Gaussian noise (1st column) and either the contribution of the classifier is masked (3rd column) or the contribution of the encoder is masked (2nd column). The normal output without any masking is shown in the fourth column.

From Figures 5.13 and 5.14, it is evident that the Autoencoder performs, as expected, most of the image reconstruction operation. The classifier's contribution is better characterized as a supplementary representation that improves the quality of the final image by adding some fine-adjusting-details to the Autoencoder's perception of what the output should be. Both representations are affected by the introduction of noise but the classifier seems to maintain a higher corruption tolerance most likely because it produces a considerably more general perception of the input without being considered too much with image details. The classifier's representation seems to concentrate more on specific characteristics like the eyes' structure and the nose-mouth morphology. It is important to note that the surface of eye glasses is more evidently captured by the autoencoder which leaves the classifier focus extensively on the features of the eyes. This observation is also important in proving some cooperation between the two model components because it seems that the presence of eye glasses in the reconstructions is generally decided by both parties acting in a very interesting way: sometimes the classifier representation stresses the contour of the eye glasses and sometimes negates the

‘attempt’ of the autoencoder to add eye glasses to the reconstruction. This is better shown in Figure 5.16 where we focus on three of the cases presented in Figures 5.14 and 5.15. The manifested cooperation seems to be a natural thing to occur when two such systems are joining forces under the specific architecture.

Figure 5.16: The classifier and the encoder are reconstructing the output through an interesting cooperation that allows the classifier to focus on eye characteristics. For example, in the first and second rows the encoder representation (3rd column) contains an eye glasses contour which is negated by the contribution of the classifier (2nd column) through appropriate pixel values at the coordinates that the encoder suggests the presence of eye glasses. In the last row, the classifier reinforces the presence of eye glasses through darker pixel values located around the eyes.

The classifier does not care for any feature that is irrelevant to the task of differentiating between problem classes. This is also proved by the fact that its representations in Figures 5.14, 5.15 and 5.16 do not account for face orientation since this is not a strong feature for the classification task. On the contrary it does what it knows best: detecting features and elaborate on their structure and formation in order to get the job done. However, it is interesting that the encoder

steps back and gets involved with constructing representations that allow for a good reconstruction by taking advantage of the feature mappings of the classifier. Another reason this is a rational thing to do is that the encoder by itself is a shallower network that is also feeding the classifier. This makes it, by definition, unable to provide the feature mapping depth of the classifier. On the contrary, the classifier is allowed to process the representations of the encoder some layers further and find features based on these mappings. In this notion, the encoder is destined to find representations that allow for a good reconstruction and to serve the classifier in finding good representations that allow for good classification results.

Another point of view for analyzing the model mechanics is through the notion of merging multiple learning tasks into the same network. Again, two different components contribute different kinds of information to the network regarding the same data space. As long as the induced information from one task does not interfere with the ability of the model to act upon the other task, given sufficient capacity and training time, this approach can be advantageous.

5.8 Practical considerations for training the model

Training deep models with large datasets requires more memory and computational resources than training a shallow network on a significantly smaller dataset. The CG-DAE additionally applies two optimizers instead of one which means even more increased computational and memory requirements. The ADAM optimizers applied by the CG-DAE require significantly more memory because they maintain information regarding a finite history of updates throughout the training process. They also have a higher computational complexity than plain mini-batch gradient descent.

Using colored images comprising of three information channels of considerable size is also stressing the training procedure. The experimental models presented in the Thesis were trained on either of two GPUS (NVIDIA GTX2070 and GTX970) in Tensorflow [145]. Generative models additionally contain a high-dimensional output layer that provides an image of the same dimensionality to the images found in the dataset. This requirement adds an extra burden in managing the available hardware resources because it increases the network parameters.

The process of feeding the model with data during training requires some design care in order to achieve low delays in reading and passing data to the training process. Mini-batch training

with a batch-size of 128 images requires periodically passing a large chunk of data to the GPU. The whole dataset usually consists of thousands of batches and each batch is used to update the model thousands of times. Consequently, any delays in the model's data feeding process have a considerable impact on the training time. To mitigate these delays, the data passed to the CG-DAE is stored in files with each file containing 2026 images. The pixel values of the images are of the unsigned integer data type format in order to require only one byte per pixel instead of the 8 bytes required by the floating point (64-bit) data type. The data type is casted to the 32-bits floating point type after the images are read from the disk into the memory. Disk accesses of large files (of the order of Gigabytes) are much more time consuming than arithmetic operations or simple casting operations in the memory. One more "trick" used to speed up the data feeding process by reducing delays during disk accesses, is to pre-fetch the next data file into the CPU's cache or into the disk's cache. This is done by spawning a process after a file access that reads the next file to be used in the near future and dispose the values to a null system device. In this way, the next file data does not consume memory since it is just discarded but (at least most of it) is kept in the cache pages of the CPU or the cache of the disk. The next file access is performed much quicker than the case of a fresh disk read request.

5.9 Recapitulation

We propose a conditional generative model that synergistically combines a classifier with a denoising autoencoder in order to produce good quality data samples. We showed that this kind of network can be efficiently trained on both labeled and unlabeled patterns and provided the theoretical foundations of the sampling process. The samples generated by the model are at least comparable to the results of other generative models at the cost of a more expensive sample process that requires several iterations towards converging to a high quality output. Furthermore, future work could use the proposed model as a building block for much deeper networks. This work reflects the belief of the authors that unsupervised learning seats in the core of true-human-like intelligence and that supervised learning may additionally serve as an auxiliary form of learning that serves the former. Nevertheless, it would be interesting to investigate whether the proposed model could provide a basis for semi-supervised learning that uses extremely few labels to train a well-performing model.

Chapter 6

Modular Domain-to-Domain Translation Network

Domain-to-domain translation methods map images from a source domain to corresponding images from a target domain. The two domains contain images from the same classes but these images look different. Recent approaches use Generative Adversarial Networks in various configurations and architectures to perform the translation. By using GANs they inevitably inherit their problems like training instability and mode collapse. Modular Domain-to-Domain translation network is a novel approach to the problem that does not use a GAN. Instead, it relies on an hierarchical architecture that encapsulates information of the target domain by using individually trained networks. This hierarchical architecture is then trained as one unified deep network. Using this approach, images from the original domain are translated to the target domain both for the case when there is a one-to-one correspondence between the images of the two domains and the case that such correspondence information is absent. The translation from one information domain to the other is visually inspected. Furthermore, the proposed model's relation to the Conditional Generative Adversarial Networks is discussed and the argument that deep learning can benefit from the proposed hierarchical architecture is examined.

5.1 Introduction

Domain translation aims at producing a pattern according to the way data is represented in the target domain, based on an input pattern coming from a source domain. The two domains contain instances of the same classes but are composed of images that are generated in different ways and look different. Assuming a source domain Z and a target domain X we need to find a mapping $M: Z \rightarrow \hat{X}$ such that the output $\hat{x} = M(z),\ z \in Z$ matches the distribution of X. An optimal mapping M translates Z to a domain $\hat{X}$ that contains images similar to the images in X. Figure 6.1 shows a domain translation example where patterns of the Street View Home Numbers dataset [154] are translated to corresponding patterns from the MNIST dataset [153].

Figure 6.1: An example of translating images from one domain to another. Both domains are composed of digits and have the same classes. The first row shows patterns from the SVHN dataset and the second row shows corresponding patterns from the MNIST dataset. Each column forms a translation pair and consists of images that have a different appearance but belong to the same class.

Domain translation is important in many application fields: artistic style image translation (converting an input image into an image that exhibits a certain drawing style), rendering images to a similar context and producing counterpart patterns for a given input. Besides such applications, domain translation is an interesting research field: humans easily identify when the same data is represented in different ways but the task proves very challenging to achieve in the Machine Learning framework. For example, humans can effortlessly identify a digit regardless whether it is handwritten on a paper or engraved on sand. On the contrary, Machine Learning algorithms are inefficient when tested on data that looks different from the training data. Accomplishing near human performance on the domain translation task and promoting

understanding of its underlying mechanisms may shed some light to human cognitive abilities like visual semantic understanding, imagination and concept transferring.

Several approaches for domain translation were developed in the recent years and the great majority of them uses Generative Adversarial Networks (GANs) [21]. For example, Kim et al [145] use two generators, one for each of the domains. These generators learn the mappings from one domain to the other. This means that the model also learns the inverse mapping for reconstructing the input image from an image in the target domain. The input data and their mappings are fed to a discriminator that applies the adversarial loss. This coupled model based on two GANs is called DiscoGAN and is able to discover relations between two unpaired and unlabeled datasets.

Benaim and Wolf [146] proposed a method that does not require any inverse mapping for reconstructing the data of the input domain. Instead, their model learns a mapping that maintains the distance between a pair of samples. Their approach is named One-sided Unsupervised domain mapping due to the absence of inverse mapping in their model. Benaim and Wolf [147] also proposed a method for translating an image from a previously unseen domain to a target domain. Interestingly, their approach considers only a single image from the source domain and is called One-shot domain translation. More specifically, they train a variational autoencoder for the target domain. Another variational autoencoder for the source domain is trained by only adapting the layers that are representationally closer to the source domain image.

A different approach for discovering the relations between two domains was proposed by Liu and Tuzel [148]. Their strategy is to learn a joint distribution of multi-domain images by enforcing a weight-sharing constraint on a pair of GANs that each generate images from one of the considered domains. The weight-sharing constraint allows the model to learn a joint distribution for the two domains without requiring tuples of corresponding image pairs. The model is called Coupled GAN (CoGAN).

A more straight-forward approach for image-to-image translation was introduced more recently by Isola et al [149] and uses Conditional GANs. While generating target domain images conditioned on source domain images is not a new idea for GANs, Isola et al applied some architectural modifications that enhance the model's performance. More specifically, they use a U-Net architecture [150] applying skip connections between the encoder and the decoder of the generator. They additionally use a Markovian discriminator which they call PatchGAN that

applies the loss function at the scale of image patches. This approach mitigates the weakness of $L2$ and $L1$ losses on neglecting high-frequency crispness. Image to image translation with GANs produces good results but deals with an admittedly easier problem since every image pair carries very similar information.

The same problem is addressed by Zhu et al [151]. Their proposed model called Cycle-consisted Generative Adversarial Network (CycleGAN) deals with the problem without requiring aligned image pairs. This is done by ensuring that the output lies in the same low-dimensional embedding space. Such a property is achieved by the introduction of two cycle consistency losses that in addition to the translation from the source domain to the target domain, they also allow for the translation of the target domain image back to the original input image. Finally, cross-domain image generation has been studied by Taigman, Polyak and Wolf [152] by implementing a modified multiclass GAN that they called Domain Transfer Network to produce an image that is relevant to an input image.

Apparently, state-of-the-art approaches implement domain translation through the use of GANs. While GANs are popular generative models due to their expressiveness, they suffer from instability due to unstable gradients and are often difficult to train. Mode collapse is another common problem occurring during training a GAN and results in a generator that outputs a single data point for the different modes of the domain. In the extreme case, the generator collapses to a single output mode which makes the model practically useless since it can only generate extremely similar patterns regardless the input. The problem becomes even greater when more than one GANs are used in a single architecture, as most of the domain translation state-of-the-art models do. Using multiple GANs in a model requires a great deal of effort to synchronize their training and tune them into cooperating in a productive manner.

We propose an alternative approach for domain translation that does not rely on GANs and thus avoids the problems accompanying their training. Our method integrates the operation of distinct network structures in a unified architecture that performs the task efficiently, while dealing with both cases of having a *1:1* and *1:M* correspondence of input and target domain images. It also applies the same representational space for both domains in order to efficiently learn the mapping from the input domain to the target domain.

6.2 The model

The proposed model aims at the translation of patterns from a source domain to patterns belonging to a different domain such that both domains share the same pattern classes. For our experiments we use two widely known datasets, MNIST [153] and Street View Home Numbers (SVHN) [154]. We also create three synthetic datasets from the MNIST dataset: the Noisy MNIST (MNIST_Noisy), the rotated MNIST (MNIST_Rot) and the rotated plus background image MNIST (MNIST_Rot_Bck). The MNIST dataset consists of gray-scale handwritten digits each centered in a *28x28* pixels bounding box having a black background. The MNIST dataset contains *60000* training images and *10000* testing images. The SVHN dataset contains digits and numbers obtained from house numbers in Google Street View images and contains more than *73257* training images and *26032* testing images. For our model the MNIST dataset represents the target domain, while the source domain is either the SVHN dataset or one of the distorted variants (synthetic datasets) of the MNIST dataset. Having a well-performing classifier for the target domain at hand is important for the proposed model because, during training, it provides vital information for performing the translation from one domain to the other. Domain to domain translation may deal with two scenarios:

1. Cross-domain pattern correspondence is the case of having the same information expressed by very similar but still different domains (patterns have a *1:1* correspondence). For example, for a target domain pattern x, the input domain may contain a modified version of this pattern x' which is obtained after x has been altered by a transformation t or distorted by a signal n, that is, $z = x' = t(x) + n$.

2. Cross-domain categorical correspondence is the case of having samples from two quite different domains having the same pattern classes. This kind of relation will be referred to as *1:M* correspondence because one sample from the target domain is related to many samples of the input domain because of class resemblance.

The 1:1 correspondence is identified in the case of having the MNIST domain as the target domain and a distorted variant of MNIST (e.g. rotated-MNIST) as the input domain. An example of the *1:M* correspondence is the case of using the SVHN as the input domain because the images of this specific dataset are quite different from the images of the MNIST domain in the sense that the digits to be recognized are accompanied with other digits placed on their sides,

there is color information and there is a lot of distortion, blurring, spatial shifting etc. Still, the pattern classes are the same for both datasets.

6.2.1 The modular domain-to-domain translation architecture

We propose a model that is composed of three distinct components: the input-domain to target-domain mapping network, the decoding of this representation to the target domain and a well performing target domain classifier. All these stages (representation, decoding and classifier) are embedded into a deep architecture that is trained with backpropagation to produce a translation network from the input domain to the target domain. However, the three distinct stages must be trained before being embedded in the final deep architecture. This pre-training strategy places the unified architecture in the vicinity of a good initial training state that maintains the objective function and prevents overfitting due to the increased number of model parameters. It also prevents underfitting caused by the vanishing gradients phenomenon caused by the deep architecture. The detailed steps for the construction of the model are shown in Algorithm 6.1 and 6.2. Figure 6.2 shows the model architecture and the distinguishable stages that we train before their unification into a deep network architecture.

Algorithm 6.1: Constructing the deep domain-to-domain translation network

Assuming an input domain dataset of the form $\{(z_1, y_1), (z_2, y_2), ©, (z_M, y_M)\}$ and a target domain dataset of the form $\{(x_1, y_1), (x_2, y_2), ©, (x_C, y_C)\}$, with z_i, x_i being the individual patterns of the source and target domains respectively and y_i being their label. For 1:1 pattern correspondence between Z and X, it should also hold that $M = C$.

1. Construct an auto-encoder for the target domain (shown as (a) in Figure 6.2) such as $f: X \to L$, $g: L \to X$ with f being the encoder, g the decoder and $L \in \Re^K$, for some K, is the latent representation. The auto-encoder should use sigmoid functions to maintain a direct probabilistic interpretation for the latent representations.
2. Train a representation network (shown as (b) in Figure 6.2) $T: Z \to R$ with the M input domain patterns with cross-entropy loss function. The training targets are binary vectors $\vec{r}_m$ sampled from the latent representations calculated in step (1) such as $r_m^k \in \{0,1\}$. The sampling is performed for every mini-batch of the training. Algorithm 6.2 describes this step in detail.
3. Train a well performing classifier D for the target domain (shown as © in Figure 6.2) such as $D: X \to Y$
4. Create a deep architecture model by embedding the model from step (2), the decoder g from step (1) and the classifier D from step (3) in this exact order as shown in Figure 6.2.
5. Train the unified model $F = \{T, g, D\}$ with tiny learning rates for stages g and D such as $F: Z \to Y$
6. The final translation model S is formed by discarding the classifier D stage and keeping $S = \{T, g\}$

Algorithm 6.2: Training algorithm for the representation network

Assuming an input domain dataset of the form $\{z_1, z_2, z_3, z_4, \dots, z_M\}$, the corresponding target domain latent representations $\{l_1, l_2, l_3, l_4, \dots, l_M\}$ as calculated from step (1) of Algorithm 6.1 and the binary form of these representations as the formal problem targets $\{t_1, t_2, \dots, t_M\}$ such as

$$t_i = \begin{cases} 1 & l_i \geq 0.5 \\ 0 & l_i < 0.5 \end{cases}$$

Repeat until convergence

Repeat until the whole dataset is examined

1. Sample a k-size mini batch such as $\{z_1, z_2, \dots, z_k\}$ associated with binary targets $\{r_1, r_2, \dots, r_k\}$ sampled from the probabilities $\{l_1, l_2, \dots, l_k\}$ such as

$$r_i = \begin{cases} 1 & p \leq l_i \\ 0 & p > l_i \end{cases}, \quad p \sim U(0,1)$$

2. Perform batch-normalization training with cross-entropy loss on the current mini batch and $\{r_1, r_2, \dots, r_k\}$ as targets.

End

3. Evaluate network for dataset $\{z_1, z_2, z_3, \dots, z_M\}$ with targets $\{t_1, t_2, \dots, t_M\}$.

End

Every distinct stage of the model is not restricted to a specific architecture and can be shallow or deep according to the application's complexity. However, the width and depth of every stage affects the final model size accordingly. As soon as we construct the individual networks, we embed them into a deep architecture and further train them as a unified network with tiny learning rates for the final two stages in order to preserve the information transferred from the pre-training procedure. If we train the target-domain-dependent stages with anything but tiny learning rates, then the network will not necessarily maintain its prior knowledge. During the experiments we assigned these learning rates a value of $1e^{-8}$. Since we initialize the last stage to be a well-performing target-domain classifier allowed to undergo only slight changes due to a very small learning rate, we expect it to maintain a great portion of this ability after the training of the model is over. Furthermore, we expect it to guide (through the backpropagation of its gradient information) the early stages of the unified architecture in adapting their feature mapping in such a way that the constructed features are a match for the last stage feature space.

Consequently, this drives the early stages of the architecture to figure out a way to construct features that are related to the target domain. We did not apply batch normalization in the final model because this could destroy the pre-training and provoke the loss of the target domain information encapsulated in the network. Training the stages provided by the target domain with learning rates that are not tiny can have the same effect. In turn, we trained the first three hidden layers with a small learning rate and the next layers with a tiny learning rate. The final domain translation model's performance could be enhanced by adding the available target domain autoencoder as an extra stage at the output of the model. This approach produces sharper and more detailed images but we did not use it in the experimental results because the main purpose is to evaluate the proposed model in its basic form. Performance enhancements and output image improvements may be explored and implemented in future work.

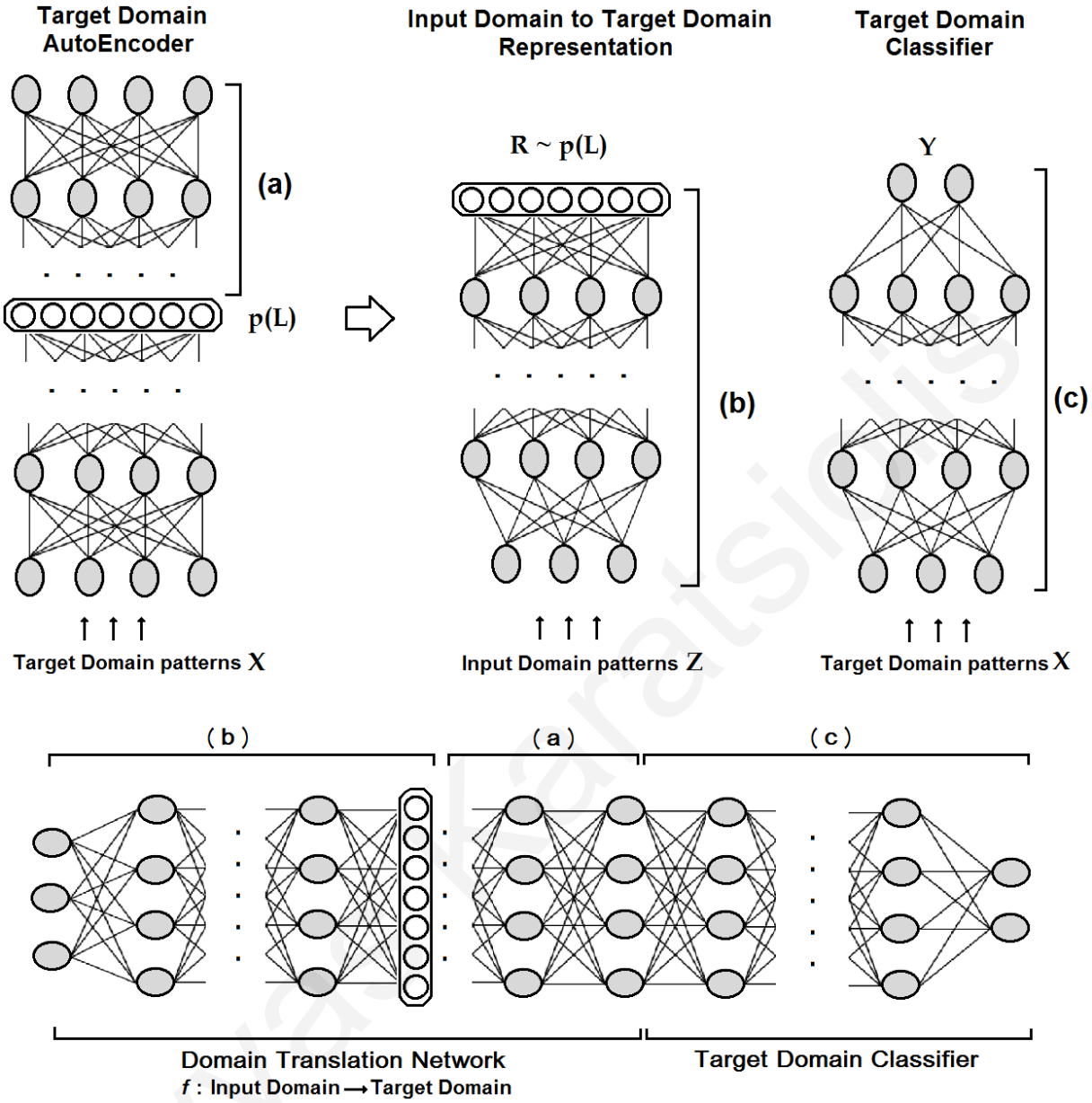


Figure 6.2: The proposed model architecture. Three distinct networks are pre-trained and then placed into the final deep architecture: (a) a target domain auto-encoder (b) an input-domain to target-domain representation network trained on cross entropy loss of sampled binary values from the latent variables of the target domain auto-encoder and (c) a well performing pre trained classifier of the target domain. After these stages are placed as shown in the final deep architecture model, they are trained with very small learning rates in stages (a) and (c) in order to avoid destruction of the knowledge transferred from the target domain. The final model effectively consists of the domain translation network formed by stages (b) and (a). At the output of the domain translation network, just before the final classifier, it is expected to observe patterns belonging to the target domain.

Every distinct stage of the model is not restricted to a specific architecture and can be shallow or deep according to the application's complexity. However, the width and depth of every stage affects the final model size accordingly. As soon as we construct the individual networks, we embed them into a deep architecture and further train them as a unified network with tiny learning rates for the final two stages in order to preserve the information transferred from the pre-training procedure. If we train the target-domain-dependent stages with anything but tiny learning rates, then the network will not necessarily maintain its prior knowledge. During the experiments we assigned these learning rates a value of $1e^{-8}$. Since we initialize the last stage to be a well-performing target-domain classifier allowed to undergo only slight changes due to a very small learning rate, we expect it to maintain a great portion of this ability after the training of the model is over. Furthermore, we expect it to guide (through the backpropagation of its gradient information) the early stages of the unified architecture in adapting their feature mapping in such a way that the constructed features are a match for the last stage feature space. Consequently, this drives the early stages of the architecture to figure out a way to construct features that are related to the target domain. We did not apply batch normalization in the final model because this could destroy the pre-training and provoke the loss of the target domain information encapsulated in the network. Training the stages provided by the target domain with learning rates that are not tiny can have the same effect. In turn, we trained the first three hidden layers with a small learning rate and the next layers with a tiny learning rate. The final domain translation model's performance could be enhanced by adding the available target domain autoencoder as an extra stage at the output of the model. This approach produces sharper and more detailed images but we did not use it in the experimental results because the main purpose is to evaluate the proposed model in its basic form. Performance enhancements and output image improvements may be explored and implemented in future work.

6.2.2 The input domain representation network

For the construction of the representation network (Figure 6.2b) which is the first stage of the deep unified model, a target domain auto-encoder must be trained first (Figure 6.2a). This auto-encoder has an encoder and a decoder stage. The output of the encoder provides the latent variables of the target domain's patterns. We used these representations as probabilistic targets to train the input-domain representation network. More specifically, we used the latent representations produced by the target domain auto-encoder to sample binary targets in order to

train a model to map the input domain to the desired target domain features. Sampling the targets of the mapping model from the latent representations of the target domain enables the representation network to learn the most essential elements required to reconstruct the target domain. In essence, the input representation network translates the source domain to the fundamental structures that make up the target domain. In other words, this approach trains the representation network into a state in which it processes knowledge of the target domain data manifold. After we trained the representation network we placed it in the final architecture. Starting the training of the deep model from the point that the initial stage of the architecture lies around the target domain manifold highly promotes learning of the domain translation task. However, for the latent representations to provide quality information for the input domain representation network they should be free of constant values. Some of the target-domain latent codes are always *1* or *0* for all the target domain patterns. In practice, we considered two threshold values, *0.99* and *0.01*. Any neuron having an output greater than *0.99* or smaller than *0.01* for all input patterns is considered to have a constant output. Maintaining these constant values is of course useful for decoding the digits through the decoder stage of the auto-encoder but conveys no special information in terms of training a model that maps the input domain patterns to the latent representations of the target domain. On the contrary, these always switched on or off neurons will constantly provide the same output target values after latent code sampling (either *1*s or *0*s). We discovered that this reduces the quality of the representation network by compromising the impact of other meaningful latent representations that provide different target values through sampling. It is important for the input representation network to stay in the vicinity of the target domain manifold. This means that the latent constant-valued variables must be exempted from these representation codes but the decoder stage should keep their effect in order to sustain its data reconstruction ability. It turns out that these constant-valued variables can be safely removed from the latent representations while their effect in the calculation of the output activations is maintained and thus the decoder function is preserved intact. To achieve this, the latent variables that are always *1* have their output weights added to the bias input of the affected neuron in the next layer, while constant *0* latent variables are just ignored since they have no impact to the calculation of the output activation. This concept is shown in Figure 6.3. The specific modification yielded faster and better training for the input domain representation network.

Once the latent variables that maintain a constant value for all input patterns are removed, the remaining variables provide the probabilistic target domain representations. More specifically, we sample these variables for obtaining binary targets for the input domain representation network. For the cases when there is 1:1 correspondence between target domain examples and input domain examples, the training of the representation network is straight forward: just sample the latent variables' vector $\vec{l}_m$ of each target domain example $\vec{x}_m$ and create binary targets $\vec{r}_m \sim p(\vec{l}_m),\ r_m \in \{0,1\}$ for the corresponding input domain example in each mini-batch. In the case that the input domain and the target domain are not related with a 1:1 pattern correspondence, the probabilistic target domain representations require a further processing step to be finalized.

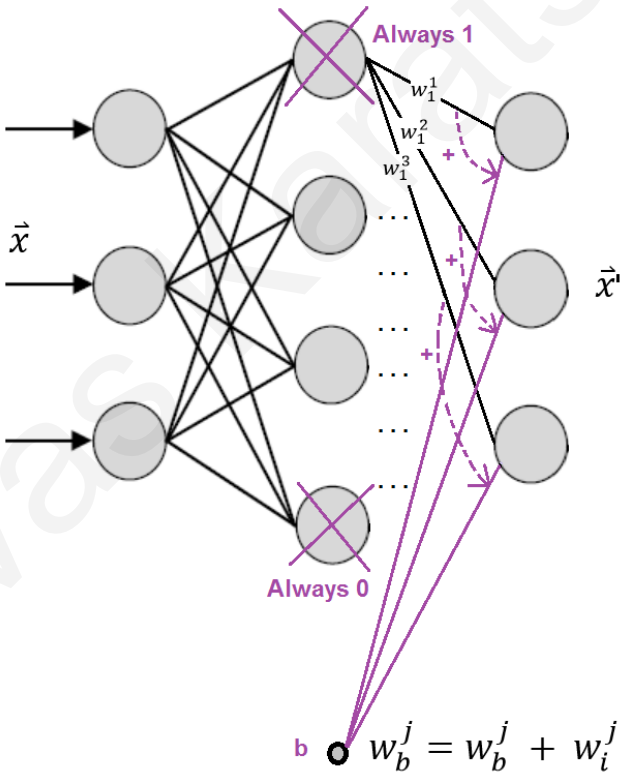


Figure 6.3: Latent variables of the target domain auto-encoder that are always 1 or 0 are omitted. The case of constant 0 neuron output is trivially omitted since the neuron has not effect on the model's functionality. The effect caused by the neurons that have constant output 1 over all target domain patterns may be embedded in the bias terms of the next layer neurons and thus the decoder's functionality is kept intact. This is accomplished by adding the weights of the corresponding neurons to the bias weight of the affected output neurons which results in an equivalent model function.

This further step is necessary because under the 1:1 pattern correspondence regime, the same information is just expressed in different but similar domains, also viewed as 'information channels' that are linked by simple transformations or domain-transition operations. This means that same-source information mappings are also linked in a pairwise (input to target domain) fashion. That is why sampling the latent representations of the target domain is sufficient to construct the feature vectors of the representation network. On the contrary, when a *1:1* pattern correspondence does not exist between the two domains of the ongoing translation, using latent representation statistics in a pairwise fashion may be misleading and difficult. First of all, there is no easy way for deciding an appropriate cross-domain pairing between the patterns, since cross-domain similarity metric construction is itself a problem of its own merit.

Under these circumstances, we use a more general association: the mean of the target domain latent variable vectors belonging to the same class. In other words, the latent representations of the target domain examples are aggregated based on their class and a mean representation is calculated for each class. These mean representations define the probabilistic targets for training the input domain representation network. For example, assuming the MNIST to be the target domain and SVHN to be the input domain, the *1:1* correspondence between the patterns is absent. In order to create a more general correspondence, namely *1:M*, we aggregate the latent representations of the target domain patterns per problem class and calculate their mean value. The mean value represents the latent probabilities for each problem class and we use these probabilities to sample the binary target values for the input-domain patterns according to their class during training of the representation network. The aggregation of the latent representations for the specific class "1" in MNIST is demonstrated in Figure 6.4.

Assuming there are M target domain examples belonging to a certain problem class C and the auto-encoder calculates a K number of latent variables l, then the binary target's sampling probability y for a specific latent variable k for the input domain representation network is

$$p\,(\,y^k \mid C) = \frac{1}{M}\sum_{m=1}^{M} l_m^k = \overline{l^k}$$

After we calculate these mean representations, we sample them for each mini batch of the training procedure of the input domain representation network. This general mapping surprisingly works better than expected and performs similarly to the more informative *1:1* correspondence. Matching a target domain example to an input domain example provides

information that should promote learning. However, for the specific translation model it suffices to initialize the input domain representation network with a mapping function which lies at the vicinity of the target domain manifold. Consequently, the more general *1:M* mapping between input domain and target domain patterns performs equally well.

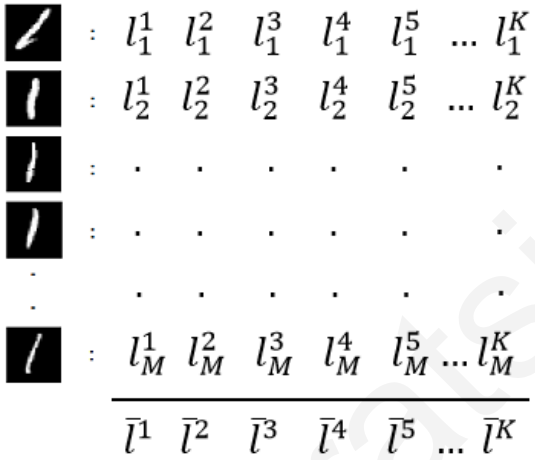


Figure 6.4: Aggregating the autoencoder's latent variables l_i^j for various instances of digit "1" in the dataset. The index of each digit instance is represented by i while each element of the latent representation is indexed by j. Different writing styles produce different latent vectors whose elements are averaged in order to calculate a mean representation for the specific class. This representation is interpreted as a vector of probabilities and each corresponding element is sampled to form binary targets for the input domain representation network.

6.3 Results

6.3.1 Data sets

Having two forms of information describing the same or similar observations is not a rare situation. For example, a digit recognition problem is described by two datasets, the MNIST and the SVHN (Street View Home Numbers). In the case of the SVHN-MNIST pair every example in one of the datasets can be associated with many examples in the other dataset linked by their common class label. Thus, their correspondence is of a general nature (simply categorical) since it is difficult or meaningless to link the examples with a more detailed relation like style, orientation, displacement etc. In other words there is not a one-to-one correspondence between

the examples of the MNIST dataset and the examples of the SVHN dataset. In order to explore such a one-to-one correspondence between the examples of the two domains, we introduce three datasets which are corrupted versions of the original MNIST dataset. These MNIST variants are listed below:

- Rotated MNIST (MNIST_Rot): the original digits are rotated by an angle generated randomly in the range $0\ to\ 2\pi$.
- Noisy MNIST (MNIST_Noisy): each pixel in the original image is replaced with *20%* probability by a random number uniformly sampled from the range $0\ to\ 1$.

Rotated + Background image (MNIST_Rot_Bck): the digits are randomly rotated as in rotated MNIST and the background is replaced with a black and white image patch. These 28×28 patches are extracted from random nature pictures downloaded from the internet and screened for having a minimum level of pixel variation. More specifically, we discard patches that have a variance less than *0.01*. Samples from the random images used to provide these patches are shown in Figure 6.5.

Figure 6.5: Samples of images from which some 28x28 patches are extracted to use as background texture for the MNIST_Rot_Bck dataset.

While creating the MNIST variant datasets, we noted the image correspondence between the original image and its variant and thus is available during the training of the translation model. Figure 6.6 shows some samples from the available datasets and indicates the direct *1:1* correspondence between MNIST and its distorted versions. The goal is to train models that can translate a pattern belonging to either the SVHN, the MNIST_Noisy, the MNIST_Rot or the MNIST_Rot_Bck domain to an appropriate pattern belonging to the MNIST domain. Additionally, for each MNIST variant we study the two correspondence relations (*1:1* and *1:M*).

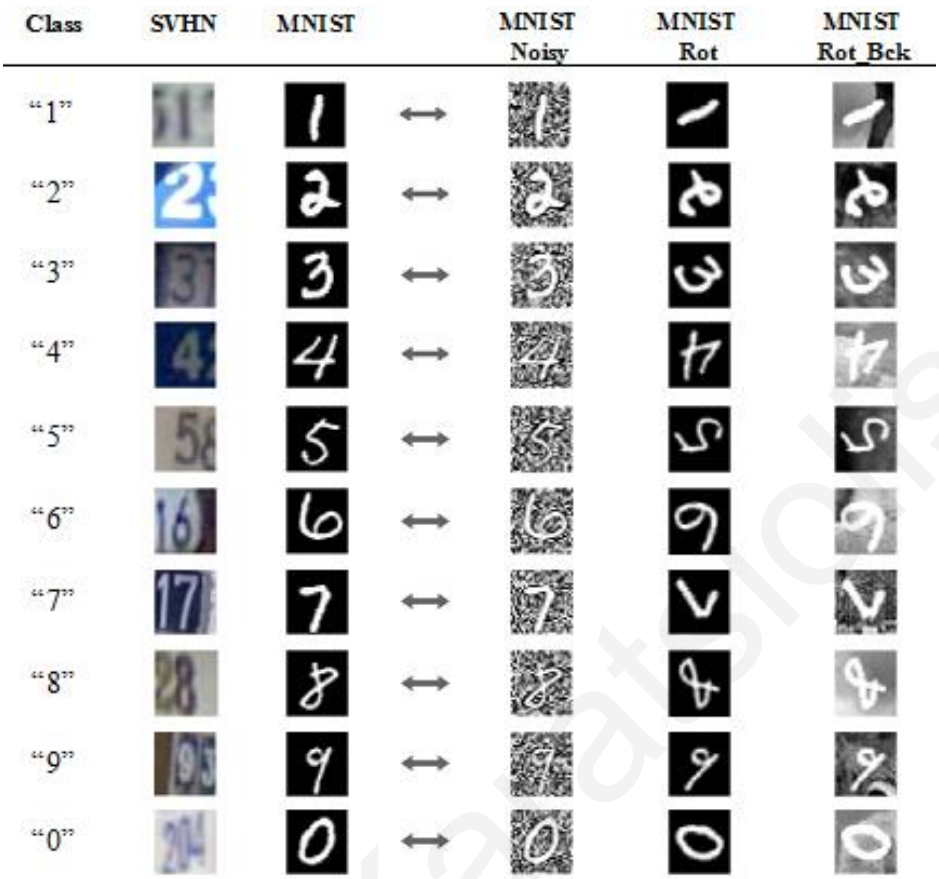


Figure 6.6: Samples from the SVHN, MNIST, MNIST_Rot, MNIST_Noisy and MNIST_Rot_Back datasets. The arrows represent a 1:1 correspondence between the patterns of MNIST and its variants. This correspondence is not applicable when a domain translation from SVHN to MNIST domain is applied or vice versa.

We explore two experimental pathways: domain translation with cross-domain pattern correspondence and domain translation without cross-domain pattern correspondence. We study both pathways for the MNIST variants translation to the MNIST domain while the translation of the SVHN domain to the MNIST domain is performed only according to the latter pathway. According to the proposed methodology we train the MNIST auto-encoder making use of the whole *60000* available patterns in the dataset. Next, we express the MNIST dataset by the latent representation of the auto-encoder and all constant valued variables (*1* or *0*) are removed by making the necessary bias weights' adjustments for the decoder stage where necessary. Furthermore, we calculate the mean latent representations per class for the case of the SVHN to

MNIST translation. Finally, we place the individual stages defined by the proposed model in a unified architecture which is trained with a tiny learning rate for the final stages.

6.3.2 The target domain Autoencoder

For the construction of the MNIST auto-encoder we use a binary restricted Boltzmann machine with *1000* neurons in the hidden layer, providing a latent representation of *1000* variables. Since the MNIST image size is *28x28*, the auto-encoder has an architecture of *784-1000*. During the training we applied a weight decay of $\lambda = 1e^{-5}$ and used the full MNIST dataset comprising of *60000* samples. We next unfolded the trained autoencoder according to its encoding and decoding parts in an architecture of *784-1000-784* with the weights of the second layer (decoder) being the transposed weights of the first layer (encoder).

6.3.3 The target domain classifier

Training on the full *60000* patterns MNIST dataset (*50000* patterns for training and *10000* for evaluation) provides classification results that exceed *99%* for the *10000* patterns test set. Hinton et al [42] reported a successful classification rate of *99.21%* by using Dropout and pre-training of the network layers with restricted Boltzmann Machines. Rasmus et al [93] used the Ladder network (semi-supervised learning) and obtained results that were slightly better, namely *99.43%*. We trained a *784-800-800-10* MNIST classifier with dropout and used it in all experiments. The classifier's hidden layers contain rectified linear units (ReLUs) and the final layer is a soft-max function. We accomplished a classification success rate of *99.17%* on the *10000* patterns of the test set which is comparable to the state-of-the-art results for the MNIST dataset.

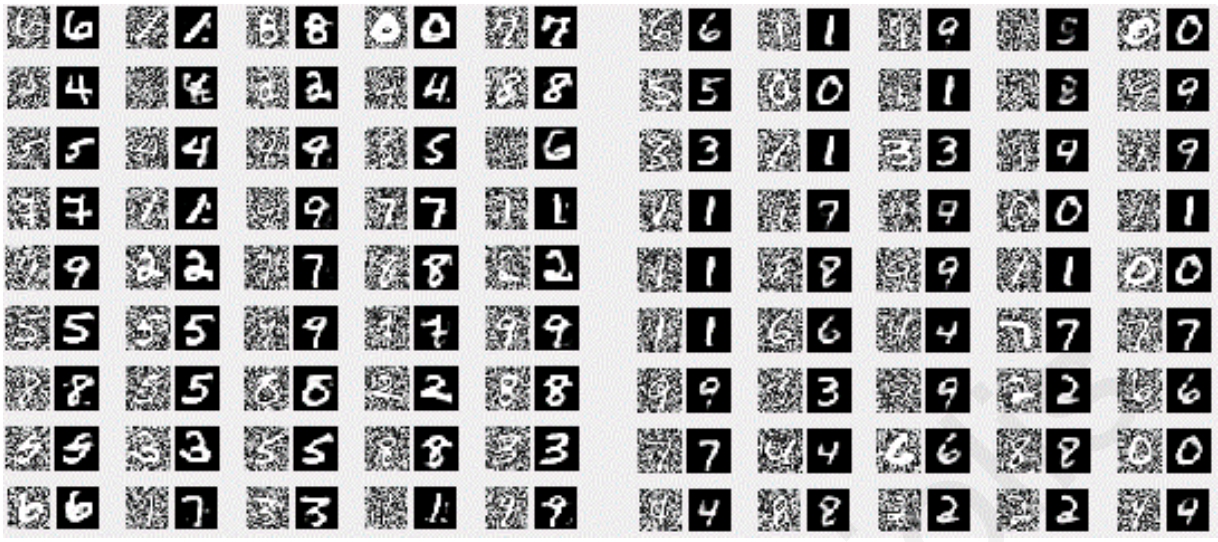
6.3.4 The input domain representation network

The latent representations for the MNIST target domain consisted of *1000* variables. After exempting the constant-valued latent representations only *922* remained. We used the latent variables provided by the target domain auto-encoder to train a *2000-2000-922* network, with the output stage corresponding to the latent representations $L \in \Re^{922}$. For the distorted MNIST datasets the input images are of size *28x28* so the network input has a size of *784*. For the SVHN case the input image is *32x32*, so we used a network input layer of size *1024*. The hidden layers neurons use rectified linear activations (RLUs) and the output layer units are sigmoidal. We

trained the network with batch normalization and mini-batches of *1000* patterns, a learning rate of $\lambda_w = 0.5$, and a tiny weight decay since the targets' sampling occurring on every mini batch creation adds a lot of gradient noise. We used two metrics to evaluate the performance of the network during training: the cross-entropy loss based on the targets and the actual outputs and the precision score of both states (*1* or *0*) of the binary outputs. The second metric is rather necessary in order to detect misleading performance scores obtained by matching many of the outputs on one binary state while heavily neglecting the other state.

6.3.5 Constructing and training the unified architecture

After the unified model resulted from the embedding of the various stages, we trained it with mini batch gradient descent. According to the networks used to form the model, the final architecture for the MNIST variants is *784-2000-2000-922-784-800-800-10* and for the SVHN is *1024-2000-2000-922-784-800-800-10*. As expected, every layer size reflects the underlying individual network embedded in the architecture. For example the 4^{th} hidden layer of both models is of size *784* because it lies at the position where the decoder part of the auto-encoder is placed and thus it must have an output equal to the MNIST image size. The 3^{rd} hidden layer of the networks reflects the final number of the latent variables of the target domain auto-encoder after the constant-valued variables are removed. Finally, the last two hidden layers correspond to the architecture of the target-domain classifier. We trained the first three hidden layers with a learning rate of $\lambda = 0.05$ and for the next layers we used a learning rate of $\lambda = 1e^{-8}$. The final results are shown in Figure 6.7 for the MNIST_Noisy, MNIST_Rot, MNIST_RotBck and SVHN datasets respectively. For the MNIST variants, the left side of the figures show the results of a model trained with the mean latent class representations. For the SVHN domain, the 1:1 pattern relation is not applicable.



(a)

(b)

©

(d)

Figure 6.7: Random samples of the (a) MNIST_Noisy, (b) MNIST_Rot, (c) MNIST_RotBck and (d) SVHN domain to the MNIST domain translation. Each pair of images is formed by the input domain pattern on the left and the model's output on the right. The left side of the figures (a), (b) and (c) shows the 1:1 cross-domain pattern correspondence case while the right side shows the translation results for the 1:M case. For the case of the SVHN to MNIST translation (d), the 1:1 cross-domain pattern correspondence is not applicable.

According to the results, the input domain patterns are translated to a pattern in the target domain that belongs to the same class and has an image quality that generally reflects the quality of the input image. All models take advantage of the existence of prior information on how to feature map the input domain to the target domain and seem to build a concept of what defines a MNIST sample. This prior knowledge is embedded to the model through the pre-trained input domain representation network, the embedded target domain decoder stage and the target domain classifier lying in the last stage of the model. This information guides the training of the initial stages of the network towards learning how to translate an image coming from a different domain to the specific format. For the specific implementation which uses the MNIST domain as the target domain this means disposing the noise in the image as efficiently as possible, reconstructing or rearranging the digit areas that are lost or modified and maintaining a uniform dark background. None of this information was given to the model explicitly but it is implied by the embedded stages that were trained based on the target domain. The results are particularly interesting for the SVHN to MNIST translation setup since the model learns implicitly an efficient way to reconstruct the MNIST format of the number in the middle of the image ignoring

any numbers appearing at its sides. Additionally, this model trains itself on dealing with distorted digit images and many variations of the original patterns. During experimentation it was noted that the quality of the target domain auto-encoder and the training of the input domain representation network were more crucial for the performance of the final model than extreme tuning of the final stage classifier. Another, rather unexpected, observation extracted from the results, is the fact that cross-domain pattern correspondence is not as advantageous as expected. The resulted quality of both experimental pathways (*1:1* correspondence and *1:M* correspondence) is not that different which suggests that during training of the input domain representation model, an adequate approximation of the target domain manifold is sufficient in terms of enabling the unified architecture to learn how to perform the domain translation.

6.4 Discussion

We have introduced a method for performing cross-domain pattern translation which transfers a pattern from one domain to a similar domain that spans the same classification categories. Figure 6.8 compares our model's results for the translation of the MNIST_Rot dataset to the MNIST dataset with the results of a CoGAN [148] trained to output MNIST images rotated by *90* degrees specifically. Our model produces at least comparable results while performing a significantly harder task since the rotation of each pattern is not fixed but varies in the range of $0\ to\ 2\pi$.

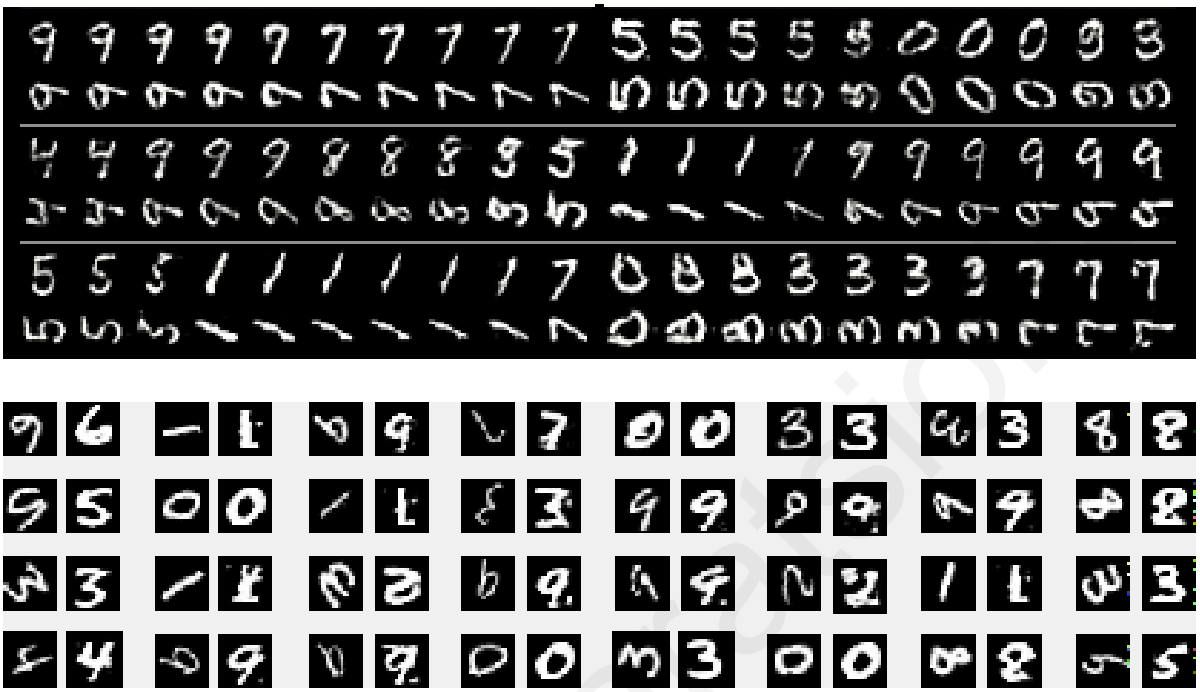


Figure 6.8: Comparison of the proposed model to the CoGAN [148] for the rotation operation. The top figure shows the results obtained by CoGAN with the top row of each group containing unrotated MNIST patterns while the second row of each group shows their counterpart rotated by 90 degrees. The bottom image shows the results of our model when trained on the rotation-inversion task with the MNIST_Rot dataset. Our model performs a more complicated task because the rotation applied is not of a constant value like in CoGAN but significantly varies in the range $[0, 2\pi]$.

Figure 6.9 shows the results of our model, CycleGAN [151] and One-sided Unsupervised domain mapping [146] when performing translation from the SVHN domain to the MNIST domain.

Figure 6.9: The results of translating SVHN domain patterns to the MNIST domain for the CycleGAN [151] (top left), the One-sided Unsupervised Domain Mapping [146] (top right) and our model (bottom figure). The results of our model are at least comparable to the other methods' results.

The results are also interesting from another point of view: the stages of the model that perform the domain translation and the classifier at the final stage of the unified network resemble the mechanisms of a conditional GAN. The former group of stages represents the GAN generator and the latter represents the GAN discriminator that outputs the K problem classes. In respect to Odena's semi-supervised GAN [155] which implements $K + 1$ discriminator outputs, the proposed model omits the fake image output class. This output is rather safely omitted because the generator is pre-trained to produce images in the vicinity of the domain information format and is prevented from deviating away from it since the training gradients come from a well performing classifier acting on that domain. The concept served by the feature mapping property added to the objective function of Tim Salivan et al method is also served by the pre-training of the input domain representation network proposed by our approach: instead of enforcing a similarity on how an intermediate layer of the discriminator is mapping real and generated

information, our model consolidates this representation through the pre-training process, the embedding of the decoder stage of the target domain decoder and the tiny learning rate applied on this stage. Extending this rational, the proposed model is acting similarly to a conditional GAN with $K + 1$ outputs, starting from an advanced training point, after the generator has started performing "reasonably" by producing images that should mostly be classified as belonging to one of the K problem classes. Hypothetically, the discriminator is constantly fooled by the generator in identifying one of the available K classes for every fake image it examines. Consequently, the fake output is omitted and not considered as an output option. Jost Tobias Springenber [156] also uses K output classes for training a GAN in an unsupervised or semi-supervised manner by omitting the fake image output. His strategy is to train an artificial image generator which produces fake images that seem real and uses a uniform distribution of samples in terms of their associating label. At the same time the discriminator must be trained to perform well on real data, to raise classification uncertainty when dealing with fake data and to choose the output class uniformly. From the requirements it is obviously assumed that the model deals with uniform class priors. Our model complies with all the requirements stated above both for the generator and the discriminator. Uniform distribution of sample generation is satisfied by the generator due to equal class priors. The discriminator also satisfies this requirement and the one for performing well on real data because of its pre-training on the target domain data. The raised uncertainty condition for the discriminator is naturally satisfied due to the noise injected during sampling of the latent probabilities. The proposed model also shares some theoretic principles with the semi-supervised GAN described by Odena [155] which uses $K + 1$ discriminator outputs.

An obvious modelling deviation of the proposed model from GANs is the absence of a noise generator at the input. Noise is important for the unconditional GAN for supplying the necessary variance at the input of the model. For conditional GANs, to which the proposed model is parallelized, this noise is not critical or even necessary since there is enough input variation due to the input domain images applied to the network and are acting like a condition for the generative process. This is also supported by Isola et al [149] and Mathieu et al [157].

Besides the conditional GAN parallelization, there is another notable characteristic of the proposed model. It suggests that deep networks perform well when the layers obey an hierarchy of functionality. Of course this is not a new idea since modern deep networks are built upon a function-specific layer architecture with various types of layers (convolutional, pooling and

fully connected layers). However, a stricter sectional embedding paradigm in the form proposed might worth further investigation and experimentation. Embedding domain information to a model in the form of whole network blocks, may produce networks that learn in a more efficient way. It could also provide the structural foundation of combining information from many similar domains to construct high level concepts that are transferable between these domains and are used to build more sophisticated models.

6.5 Conclusions

The modular domain-to-domain translation network is a novel approach for performing domain translation without incorporating a GAN in the model architecture like the great majority of the previously proposed models do. The specific approach uses an hierarchical architecture composed of individually trained modules. The architecture is trained as a whole (fine-tuned) and is able to achieve results that are at least comparable to the ones provided by the state-of-the-art models that use one or more GANs in their architecture. By not using GANs, the proposed architecture avoids deficiencies related to the training of GANs such as high instability and mode-collapse.

Chapter 7

Region based Support Vector Machine Algorithm for medical diagnosis on the Pima Indian Diabetes dataset

The problem of diagnosing Pima Indian Diabetes from data obtained from the UCI (University of California Irvine) Repository of Machine Learning Databases is handled with a modified Support Vector Machine strategy. Performance comparison with previous studies is presented in order to demonstrate the proposed algorithm's advantages over various classification methods. The goal of this study is to provide the grasp of a methodology that can be efficiently used to raise classification success rates obtained by the use of conventional approaches such as Neural Networks, RBF (Radial Basis Function) networks and K-nearest neighbors. The suggested algorithm divides the training set into two subsets: one that arises from the joining of coherent data regions and one that is composed of the data portion that is difficult to be clustered. Consequently, the first subset is used to train a Support Vector Machine with an RBF kernel and the second subset is used to train another Support Vector Machine with a polynomial kernel. During classification the algorithm is capable of identifying which of the two Support Vector Machine models to use. The intuition behind the suggested algorithm relies on the expectation that the RBF Support Vector Machine model is more appropriate for data sets of different characteristics than the polynomial kernel. In the specific study case the suggested algorithm

raised average classification success rate to 82.2% while the best performance obtained by previous studies was 81% achieved by a fine tuned, highly complex ARTMAP-IC (Adaptive resonance theory mapping with instance counting) model.

7.1 Introduction

Medical applications have been pushing the advancement of Computational Intelligence for some decades now, mainly because of the need to improve the accuracy of diagnostic processes and the need to reduce the accompanied cost. At the same time the medical sector provides plenty of structured information for the researchers to experiment upon. It is an indicative fact that the data sets belonging to "life sciences" category currently available on UCI Repository are more than 28% of the total data sets available. In turn, this amplifies the need to constantly improve available prediction algorithms and come up with new efficient models for data manipulation. The Pima Indian Diabetes (PID) data set includes information gathered from at least 21 years old Pima Indian females. It's a relatively popular set probably because of the difficulty it opposes towards achieving good classification results, a challenge that seems to be responsible for the data set's quite few citations. As a result of the challenging nature of the specific dataset, a variety of simple and complex models have been tried to achieve improved classification results [158, 159, 160,161], but still the success rates remain only around 80%. Smith et al. [161] used the PID data set to evaluate the perceptron-like Adaptive learning routine (ADAP). This study had 576 cases in the training set and 192 cases in the test set. Using 576 training instances, the sensitivity and specificity of their algorithm was 76% on the remaining 192 instances. The same number of random training and test sets was used to compare the simulation results. On the Pima Indian Diabetes (PID) database fuzzy ARTMAP test set performance was similar to that of the ADAP algorithm [161] but with far fewer rules and faster training. An ARTMAP pruning algorithm [159] further reduces the number of rules by an order of magnitude and also boosts test set accuracy to 79%.

An instance counting algorithm ARTMAP-IC [158] improves accuracy to 81%, at the expense of added model complexity. Table 7.1 [158, 159, 160, 161] compares the performance of several studies on the PID dataset.

TABLE 7.1

Performance results of various studies on the PID dataset [158, 159, 160, 161]

Methodology	Classification success rates(%)
Logistic Regression	77
MLP (Multi-layer Perceptron trained with Levenberg-Marquardt)	77
ADAP	76
RBF	68.23
General Regression NN (Neural Network)	80.21
ARTMAP-IC	**81**
KNN	77
ARTMAP	66

The relatively low success achieved by different angles of attack is frustratingly opposing claims that the specific data set is complete and correct. As a matter of fact, this is the general case when a bunch of machine learning algorithms fail to come up with a model that achieves satisfactory recall results. Their results seem to approach a solid upper bound that seems hard to overcome. At the same time, theoretically weaker models like linear classifiers and K-Nearest Neighbor (k-NN) models achieve similar success rates. Analyzing this fact, one may come to the conclusion that the combination of two disappointing omissions constraint machine learning algorithms from generalizing well over the dataset: available attributes are not appropriate or the variation of attributes over time is highly contributive to the information basis of the problem and is not examined or is not correctly embedded in any form to the dataset. These two problematic phenomena are explained in the following paragraphs.

In order to gain a better understanding of the first constraint let's consider a diagnostician who is able to analyze a person's genome and consequently, to directly check the integrity of a gene (or genes) responsible for a specific medical condition. This approach would cancel out the need for a machine learning algorithm to recognize a disease, since an observation of a directly linked “quantity” (responsible gene's structure) provides the answer with zero or negligible uncertainty. Adding levels of abstraction by observing the phenotype of a gene's expression in

proteins, raises the need to combine measurements of phenotypes and uncertain indicators to generalize over these observations. The higher the level of abstraction, the more carefully the attributes must be selected in order to encapsulate the required information necessary to drive the learning process. Usually, a higher level of abstraction contributes in the lower expense of data collection but at the same time it fades away the strength of the immediate link and the causality between observation-conclusion. This leads to the use of less appropriate but cheaper to obtain attributes.

The reasoning behind the second omission (not using time varying data relations) when referring to medical problems relies partly on the diversity of DNA (Deoxyribonucleic acid) and mainly on the interrelations of living organisms body chemistry. A measurement of a physical quantity is affected by factors that do not have to do with its immediate effectors. For example, blood sugar may be increased by factors uncorrelated to diet or pathology, like stress or infections. Taking various measurements over time so as to average out some unwanted conditions may reveal some information that would normally be overseen by one-time measurements or make it possible to avoid unwanted measurement noise. Also a side effect of the dataset attributes' abstraction may be the delayed effect of a condition on the values of the attributes selected, which could be misleading. For example, the stage of a disease may have different impact on the attributes, especially on early and later stages of manifestation.

When a specific dataset is used to train a classification model there is little to do about the second omission regarding time varying data relations. This should probably be considered when conducting data collection. As far as the first problem is concerned, regarding attributes' appropriateness, the algorithm proves that mapping a carefully selected subset of the available data set to a constrained VC (Vapnik Chervonenkis) dimensional feature space (through the polynomial kernel SVM) could reveal some relations that would be otherwise unseen when mapping the whole data set to a high or infinite dimensional feature space at once (for example through an RBF SVM). This fact suggests that some cases in the data set have large projections on a group of features that overwhelm weaker projections on other features that can provide the means to improve classification rates. The goal is to separate the data set into subsets in a way that this overwhelming phenomenon is reduced or even minimized.

7.2 The reasoning behind clustering the dataset

A data set is considered to be of reasonable value if its information basis is maintained and is not malformed by noise to a grade of no separability and the distribution of data is more or less imprinted in collected measurements. Given that theses prerequisites are met, most of the time one can identify two types of data within the dataset in respect to potential feature space separability: the set of data that can be classified correctly in an easy or moderately easy way and a set of cases that is more challenging or very hard to separate using a single model. When dealing with classification problems, it is usual to try out several models towards an improved and approved solution, for example neural networks with different architectures and different optimizers, support vector machines with various kernels, KNN classifiers, clustering algorithms like k-means, discriminant models etc. It is common to observe that specific test cases in the validation set are always classified correctly. Easy classification means a safe distance from the separation boundary. A more accurate expression would be "classified correctly with high confidence" but the sloppier definition is preferred for the sake of the expressing argument. On the other hand, specific cases may almost always be misclassified. The above scenario is mainly true when high success rates cannot be achieved on the validation set in contrast to a significantly lower training error without this phenomenon being a consequence of over-fitting. It could be an effect of erroneous or noisy measurements and in some cases could be caused by the use of an inappropriate or incomplete attributes' set. When the latter case concentrates the researcher's suspicions, a principal component analysis of the data set may shed some light to the investigation.

This study concentrates on the event that none of the above problematic cases applies but rather a more disturbing scenario is at hand: some cases in the dataset are by nature difficult to separate by a single machine learning model. It may be the case that the majority of the examples in the validation set are separated correctly by a number of models but a smaller portion of the test cases is terribly processed. The end result is, of course, the degradation of the overall classification performance. An anticipated side effect of this phenomenon is the fact that a linear model acting on the same dataset may provide very similar results to a non-linear model. This surprising fact is based on the strong belief that the dataset is not linearly separated. The intuition behind this situation is that while the algorithm is trying to minimize the training error by constructing concepts that service the majority of the cases, these concepts are far from applicable to a respectable (in terms of size) minority. It will eventually favor the largest of the

training cases group because minimizing the objective function is apparently only feasible through the favoring of the majority. At the same time the separation hyper surface is probably very unsmooth while smoothness is a necessary quality of good generalization. In turn, this makes a linear separation of the data perform quite satisfactory relatively to a non-smooth hyper surface with large linear regions that may look like a very noisy linear boundary.

The observation that a linear model can have results that are similar to a well-tuned nonlinear model is worthy of more examination. Avoiding feature mapping in a higher dimensional space and working in input space instead and still get better results on a classification problem, can only mean one disappointing situation: feature mapping is done poorly. On the other hand, when dealing with the RBF kernel in SVMs, feature space has infinite dimensions and does not constraint the production of any appropriate feature. So feature dimensionality is not an issue. The only possibility left is the situation that some cases are discriminated by features that are not really helpful for discriminating other cases located in different subspaces of the feature space. So it sounds rational to detect the cases that get a discrimination advantage from one large feature space (like the feature space of a SVM with RBF kernel) and work with the rest of the cases in a very different but constrained feature space. By following this methodology, confusing cases are constrained from preventing a smooth generalization boundary to be formed out of reliable data while at the meantime these less reliable data cases are processed explicitly by polynomial features formation. The concept is illustrated in Figure 7.1.

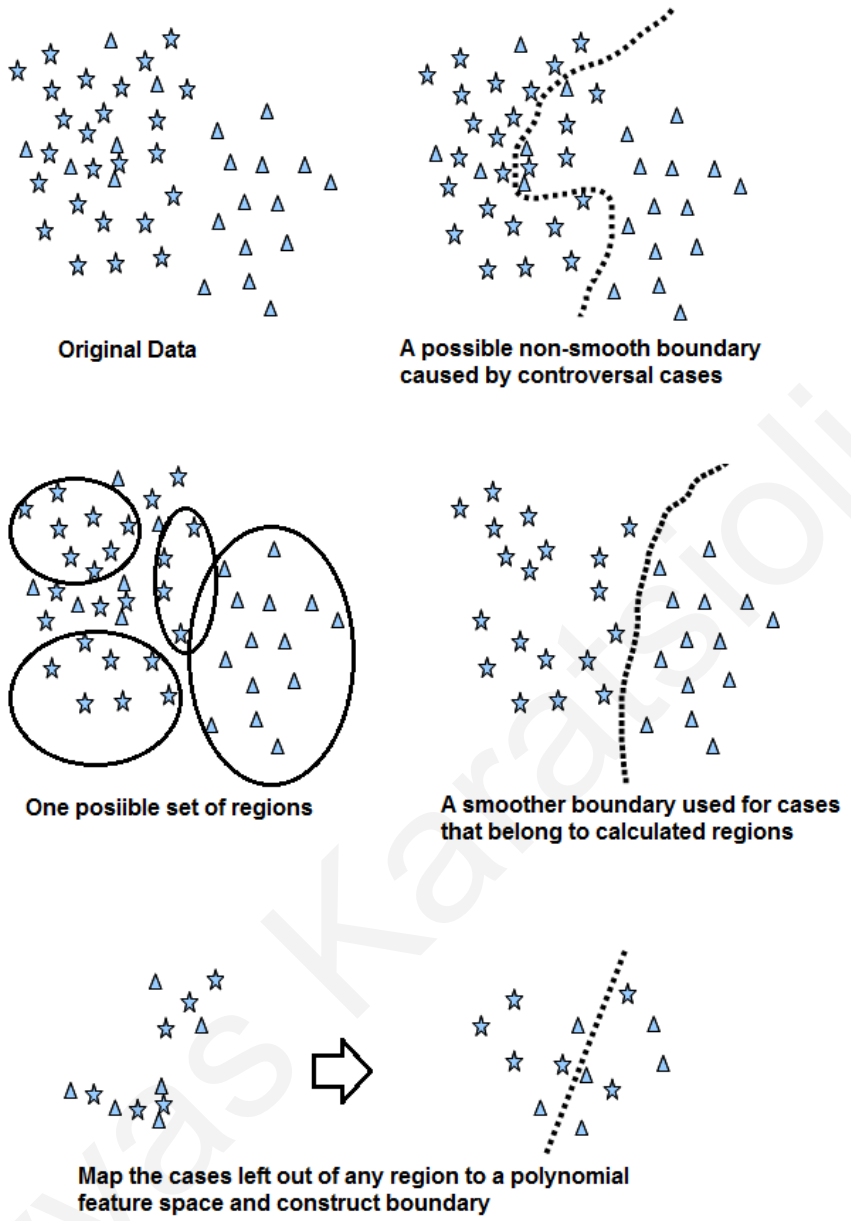


Figure 7.1: Controversial cases decrease smoothness of the classification boundary which in turn reduces generalization. A boundary calculated based on highly reliable regions of classes tends to be smoother. The cases that are not included in any region are mapped to a polynomial feature space for a separate classification model.

In addition, the suggested methodology considers another rather significant characteristic of the data distribution. Data belonging to the same class may be distributed in considerably sparse clusters. As long as these clusters are at least loosely defined, they are discovered by the proposed algorithm because of its natural ability to identify data clusters. Based on the above

discussion it may seem rational to cluster the training set in a way that the training examples that require different or even conflicting feature mapping functions are assigned to a different modeling cluster. Classifying an unseen case becomes a two stage process, first recognizing the cluster a specific pattern belongs to, and then applying the correct model for separation.

7.3 The Algorithm

From the previous discussion, it seems rational to have a number of machine learning models derived during the training phase and then use the most appropriate one, depending on the test case under consideration. The selected model for each occasion is based on its suitability to operate on the subspace of the problem that the test case belongs to. However, classifying the cases to problem subspaces and training a model for each subspace is by itself a modified version of the original problem. There are two major problems one has to overcome in order to apply this point of view to the classification problem. Making the clustering according to input space metrics will not help since the problem described in the previous section is laying in feature space mapping, so using input-space-based clustering is not an option. What is more, as explained above, splitting the feature space into subspaces and maintaining the ability to distinguish what model to use for an unseen case is a classification problem of its own merit. Regional SVM algorithm deals with these two problems by using multiple SVMs decision boundaries to form unified areas (regions) that include example cases belonging only to one class and exactly zero example cases belonging to other class(es). This results in the formation of confident positive regions and confident negative regions since numerically the classes of an SVM dichotomization are either *+1* or *-1*. Positive regions include only a number of positive class cases and respectively negative regions include only a number of negative class cases. To be more specific, every cluster is defined by the union of the spaces that a group of SVM classifiers learn to classify as class *+1* (positive class region) or class *-1* (negative class region). By detecting one class region after the other and the respective data cases that define them, the algorithm builds up a set of negative and positive class cases that can be used to train a SVM with an RBF kernel to construct a classification boundary. The rest of the cases that during the calculation of the region-detection SVMs are left orphan, meaning they are not part of any region, are used to construct a second model that will try to discriminate between the classes by using a reduced feature space kernel (polynomial kernel). This reduced feature space kernel will try to create polynomial features of the input space and operates quite differently from the spatial

concept that an RBF kernel is using through the Euclidean distance from selected support vectors. A neural network with one hidden layer is also capable of calculating polynomial features of the input space and could be used instead of a polynomial kernel SVM. However, the latter is preferred due to lower execution times when the data set size is not large. Evaluating regions of positive and negative cases provides the means to solve the problem of recognizing which model is appropriate for an input case: first investigate whether the examined case falls into any of the regions and if so, apply the RBF kernel SVM, else use the polynomial kernel SVM.

In order to calculate positive and negative regions of the data set, a genetic algorithm is put to work with the objective to evolve a population of representations that reflect the information region (and consequently the region-detection SVM) that the individual cases of the dataset actually belong to. When a pattern belongs to a specific region, this automatically means that it is used in the training set of the corresponding region-detection SVM. In this specific case (PID dataset), three region-detection SVMs are trained over evolving data subsets. After the training of the SVMs using the dataset cases represented by each individual in the genetic algorithm population, the union of the space of the positive (or the negative) side of the boundaries of the SVMs is checked to detect whether it includes a satisfactory number of only positive (or only negative) examples of the dataset. If it does, the corresponding data cases are removed from the working data set and the region is saved in memory. A region is defined by its sign (positive or negative class), by the set of the SVMs whose unions of boundary areas define the region and by the data cases that are found in the region. It must be noted that all positive regions are calculated first and when it becomes very hard for the genetic algorithm to detect more positive regions, the whole process is repeated again so as to calculate the negative regions. Every time a region is detected, the enclosed data cases are extracted from the search and the algorithm is initialized (initial random population) according to the resulting smaller data set. The fitness function of the genetic algorithm reflects the necessity to include only single class cases. This is accomplished by constructing a fitness function that depends on the ratio of the defined region's enclosed cases belonging to the targeted class over the enclosed cases belonging to the undesired class. Thus, by maximizing this ratio through genetic operations, the genetic algorithm tries to exclude any undesired class cases. When the positive and negative regions are collected they are used in two ways. First, all data cases included in both positive and negative regions are combined to produce a somehow reliable data set that is used to train an SVM with

an RBF kernel. The result is a model that has a smooth boundary and generalizes well over the data that falls into the constructed regions. To determine whether a test data case falls into one of the regions learned, requires the classification of the case with the SVMs that define the regions one at a time. If a pattern is classified as positive by all region-detection SVMs then it belongs to that area (union of individual regions). The same analogy holds for the negative class area. All data left out (not belonging to a region) is collected and a data set is constructed that trains a polynomial kernel SVM. A priori we expect that the separation of this dataset is very hard, which means that the success rate of its classification is lower than the success rates achieved by the RBF model for the cases belonging to the constructed regions. The important thing is that when the two models are combined into a unified classification system they provide better results compared to the case of training a single model for the whole original dataset. The algorithm is described by the following step-by-step procedure:

1. Initialize the genetic algorithm with a population P containing N individuals with random genes and train initial region-detection SVMs accordingly. Each individual p_i holds a set of indexes x_k , $0 \leq x_k \leq j$, $0 \leq k < M$, with j being the number of region-detection SVMs and M being the number of training patterns. Each index x_k points to one of the region-detection SVMs or indicates that the corresponding pattern should be included in the dataset of the polynomial kernel SVM. More specifically, each region-detection SVM has an identification number in the range $[1, j]$ and these identification numbers are used by the indexes in each population individual to show the region that a training pattern belongs to. An index may also have a zero value instead of the identification of a region, which means that the pattern does not belong to a region and should be classified by the polynomial kernel SVM. The population has the following form

$$P = \{[x_1, x_2, x_3, \dots, x_M]_1 , [x_1, x_2, x_3, \dots, x_M]_2 , \dots , [x_1, x_2, x_3, \dots, x_M]_N\}$$

2. Fitness Function reflects the search for solely positive regions. Each individual's fitness function is the number of positive cases that are classified as positive by all SVM models over the number of negative cases that are classified as negative by all of its SVM models. So the cases belonging to the positive area $Cases^+$ also belong to the positive problem class while all cases in the negative area $Cases^-$ belong to the negative problem class.

$$f(p_i) = \frac{|\ Cases^{+}\ |}{|\ Cases^{-}\ | + 0.001}$$

$$Cases^{+} \in \{\ SVM_1^{+} \cap \ldots . \cap SVM_j^{+}\ \}$$

$$Cases^{-} \in \{\ SVM_1^{-} \cap \ldots . \cap SVM_j^{-}\ \}$$

3. The population is evolved through *2*-point crossover and mutation (*0.02*%) for maximizing fitness function $f(p_i)$. After a genetic operator is applied to an individual, the region-detection SVMs are trained with the resulting datasets.

4. If after an epoch an individual has a very high fitness function (meaning zero negative class cases) and the number of the positive cases is more than or equal to *5%* of the total positive data set examples, then the region is saved and the included cases are removed from the total training set. The process is repeated from step *1*.

5. When the algorithm cannot detect any more valid positive regions, the search for positive regions terminates. Steps *1* and *4* are repeated for the negative class cases, with the reversal of the fitness function to reflect the goal of searching clusters for negative cases. The process is the same as searching for positive-patterns regions but this time, the regions detected by the SVMs correspond to the negative class patterns. More precisely, the cases belonging to the positive area $Cases^{+}$ also belong to the negative problem class while all cases in the negative area $Cases^{-}$ belong to the positive problem class.

6. Cases included in all regions (either positive or negative) form a training set that is used to train a final RBF SVM. All left-out cases (not belonging to a positive or a negative region) are used to train a polynomial kernel SVM.

7. During the classification phase, an unknown case is passed through the region-detection SVMs and if it belongs to one such area (union of regions), it is classified with the main RBF SVM. Otherwise it is classified by the polynomial kernel SVM. In practice, first a decision is made regarding whether a pattern belongs to the union of regions that are composed of easier problem cases or to the area of more controversial cases. If the former is true and a more straight forward pattern is identified, then the pattern is classified with the final RBF SVM. Otherwise, the pattern is classified with the polynomial kernel SVM.

7.4 Results

The algorithm was used to divide the data set to two subsets: the one that is used to train an RBF Support Vector Machine and the one that is used to train a polynomial Support Vector Machine. The complete data set consists of 500 normal cases and 268 abnormal cases. Region based SVM algorithm results in the creation of an RBF training set of 376 normal and 177 abnormal cases and of a polynomial training set consisting of 124 normal and 91 abnormal cases. In turn, these training sets are used to train both SVMs by a 5-fold cross validation technique. Each validation set pair is shown in Table 7.2 and the final results are shown in Table 7.3.

TABLE 7.2

Sizes of the Cross Validation Datasets

RBF SVM				**POLYNOMIAL SVM**			
Training Set Normal	*Training Set Abnormal*	*Test Set Normal*	*Test Set Abnormal*	*Training Set Normal*	*Training Set Abnormal*	*Test Set Normal*	*Test Set Abnormal*
131	123	245	54	62	54	62	37

TABLE 7.3

Final results on various validation test sets. The overall success rate is calculated using the success rates of both the RBF and the polynomial SVMs.

	RBF SVM		**POLYNOMIAL SVM**		**Overall Final Test Results**	
Validation Set number	***Normal Cases Success Rate (%)***	***Abnormal Cases Success Rate (%)***	***Normal Cases Success Rate (%)***	***Abnormal Cases Success Rate (%)***	***Normal Cases Success Rate (%)***	***Abnormal Cases Success Rate (%)***
1	98.38	86.94	67.75	64.87	**83.06**	**82.41**
2	83.68	92.6	64.52	67.57	**79.98**	**82.41**
3	85.3	92.6	70.97	64.87	**82.41**	**82.41**
4	86.94	90.74	64.52	70.27	**82.41**	**81.31**
5	88.57	88.89	64.52	72.97	**83.71**	**82.41**

7.5 Conclusions

The overall results are compared with the ones of several previous studies shown in Table 7.1 and a small improvement on the average performance is achieved by the suggested algorithm. It is important to note that the suggested algorithm is able to outperform much more complex algorithms like ARTMAP-IC and achieves satisfactory performance while avoiding excess tuning.

On the other hand there are some limitations in applying the suggested algorithm. The most obvious one is the limitation raised by the data set size. A small dataset is not eligible for solving with the proposed algorithm because of the algorithm's approach to divide the dataset to clusters which in turn reduces the size of the test set. Extremely small-sized test sets make the test phase unreliable and prone to over fitting. Consequently the available dataset must contain at least some hundred examples. Another issue that region based SVM must deal with, is the execution time of the genetic algorithm when the dataset is large. Special programming skills must be used to improve performance with the use of parallel execution being the most efficient approach to apply. Finally, the algorithm's performance was not tested on multi class problems.

Chapter 8

Using adaptive neural network targets for dealing with extremely imbalanced datasets

An original classification algorithm is proposed for dealing with extremely imbalanced datasets that often appear in biomedical problems. Its originality comes from the way a neural network is trained in order to get a decent hypothesis out of a dataset that is composed of a huge-size majority class and a tiny-size minority class. Imbalanced datasets are especially probable when forming machine learning databases describing rare medical conditions. The algorithm is tested on a large dataset in order to predict the risk of preeclampsia in pregnant women. Conventional machine learning algorithms tend to provide poor hypothesis for extremely imbalanced datasets by favoring the majority class. The proposed algorithm is not trained on the basis of the mean squared error or cross entropy loss objective functions and thus avoids the overwhelming effect of the highly asymmetric class sizes. The methodology provides preeclampsia detection rate of 49% and normal case detection rate slightly above 76%.

8.1 Introduction

Many biomedical machine learning problems are very difficult to tackle because of the rareness of the medical condition they are dealing with. For example, preeclampsia only occurs in approximately 2% of all pregnancies. As anticipated, any database constructed in order to feed a machine learning algorithm that will predict the occurrence of this unfortunate disease to an individual, contains a very small number of preeclampsia cases compared to the number of normal cases. In turn, this phenomenon greatly affects the quality of the constructed hypothesis especially if the learning procedure is based on the cross-entropy loss function. Through the years many machine learning researchers have proposed quite a few workarounds towards overcoming this problem but in their majority these techniques are empirical and problem dependent. Foster et al [162] and Weiss et al [163] studied the imbalanced dataset situation in machine learning classification and analyzed different aspects of the problem. Foster claims that standard Machine Learning algorithms trained with an imbalanced dataset should have their decision threshold adjusted accordingly. Another way to favor the minority class identification without adjusting the threshold is to add lower weights to the majority class examples during training. Weiss tries to explain why the minority class error rate is high and proposes the implementation of a progressive sampling strategy that improves the balance between the problem classes.

Japkowicz et al [164, 167, 168] also studied the problem using artificial data and suggest that the performance of a learning algorithm trained with imbalanced datasets depends on the complexity of the problem, the training set size and the degree of the imbalance. Additionally, linearly separated problems are much less sensitive to imbalanced datasets and the main source of the poor performance on such sets is the presence of small isolated minority class clusters. Many researchers claim that the best way to deal with the problem is to preprocess the data through sampling. The majority class could be reduced in size by under-sampling while the minority class size could be increased through over-sampling. Such sampling processes might be performed randomly or heuristically. Batista et al [169], among other researchers, note that random over-sampling may cause over fitting while under-sampling may cause loss of useful information. Some popular heuristic-driven sampling methods are Neighborhood Cleaning [170, 171], Tomek links [172], One Sided Selection [173], Condensed Nearest Neighbor [174] and Smote [166]. Application of the Smote algorithm with the incorporation of Tomek links seems to provide a good basis for dealing with imbalanced datasets. Such an approach was

successfully implemented in the annotation of proteins in Bioinformatics [175]. Finally, Drummond [165] and Chawla [166] claim that experimental results prove that under-sampling usually proves to be more effective while, on the contrary, Japkowicz et al [164] claim the opposite.

It is evident that the solutions proposed are of empirical nature, are heavily problem-dependent and require a lot of fine-tuning through trial and error. The proposed algorithm deals with the imbalance of the dataset and constructs a decent hypothesis for the classification task.

8.2 Dealing with the Dataset Imbalance Problem

Training a neural network on a loss function that is governed by concrete probabilistic properties like cross-entropy or mean square error, inevitably introduces a strong tendency to consider the data priors. This in turn, has a devastating effect on the classifier since it is forced to construct more and stronger feature detectors for the dominating class. Generally, representation learning models have a hard time generalizing well when trained with probabilistic loss functions on greatly imbalanced datasets: imbalanced priors are encoded to the extend that problem features and the learned representations are internally constructed. The problem can be tackled with several techniques which can be summarized in three major approaches.

The first approach is to avoid end-to-end representation learning and provide sophisticated hand-engineered features for the problem. The process of constructing the low-level problem features is the one that suffers the most from imbalanced dataset situations. Of course, hand-designing the problem features is costly, difficult and time consuming. On the other hand, if probabilistic loss functions are to be used without any intervention on the way data is fed into the model, there is no other alternative than to ensure that the model is trained on input features of high quality. Well-engineered problem features may be linearly separable and linearly separable problems are less sensitive to the imbalanced datasets phenomenon.

The second approach is to trick the model into experiencing modified priors. This may be achieved by introducing an intermediate data-processing step before actually feeding the data to the model. Such a step may be a data sampling process (over-sampling in terms of the tiny class and under-sampling for the huge class), a strategic dataset reduction or a data augmentation process. The specific approach is heavily based on empirical results and usually needs a lot of

experimentation to detect the "sweet spot" between making things worse and improving model's generalization ability.

The third approach is to use a loss function that is of "raw form" in the sense that its probabilistic interpretation does not overwhelm the process of learning the internal representations. This approach is actually our contributing proposal and works by relaxing the model output to local minimums or maximums (depending on the class label of each data pattern) during training and using these values as target-values for the various patterns in a second training phase that returns a regression model predicting these outcomes. Having variable target values not only disentangles the model and the data class priors but also helps the model-function become more convex shaped at each iteration.

8.3 Adaptive Neural Network target values

In order to avoid fixation of the training procedure to the class priors, the neural network is not restricted to pursuit some fixed labels. Instead, it is trained to maximize its output for the first class and to minimize its output for the other class. Under this approach, supervised learning paradigm relaxes to the point of just providing the desired direction of search regarding each training example. To be more specific, in the problem under study, preeclampsia training patterns are accompanied with a maximization flag and the normal cases are accompanied with a minimization flag. Normally in supervised learning we provide a fixed target value which is usually *+1* for the positive class patterns and *-1* or *0* for the negative class patterns. The neural network is initialized with random weights but the training process does not aim in adapting the weights. The maximization-minimization is performed by varying the inputs, not the weights. Each training example is modified using gradient information of the output in respect to the input in order to find a local maximum or a local minimum depending on the target flag. This locally optimal value is stored and used as the calculated target value for the specific example in the second phase of training. In this way, the model targets are not predetermined by the data acquirer/provider. Instead, they are calculated by the neural network. In the second training phase, the data set and the calculated targets are used to train a neural network regression model initialized with the weights used during the first phase of training (during calculating appropriate target-values for the problem). The Broyden-Fletcher–Goldfarb–Shanno (BFGS) algorithm is used to reclaim the provided gradient information to search for a local optimal. Due to the

variability of the target values, the output layer of the Neural Network does not use a squashing function but a linear function instead while the hidden layers use the sigmoid activation function. Algorithm 8.1 shows the pseudo-code of the methodology.

Algorithm 8.1: Pseudo-code of the adaptive target values algorithm

```
1. weights = InitializeRandom()
2. for iter = 1:MaxIterations
3.     for i=1:M
           if  label(i) == true
               targets(i)  =  BFGSLocalSearchMax(data(i),weights)
           else
               targets(i)  =  BFGSLocalSearchMin(data(i),weights)
4.     trainedWeights =  BGFSRegression(data,targets,weights)
```

The calculation of the gradients for the single and the dual hidden layer Neural Network is shown below. For the single hidden layer network we define y to be the output of the model and z_k to be the activation of the k hidden unit when pattern x^i is applied to the input of the model, where $x^i \in \Re^n,\ 1 \leq i \leq M$. The total number of hidden units is K and the output layer weights are denoted as w_k.

$$y = \sum_{k=1}^{K} w_k \, h(z_k)$$

$$z_k = \sum_{j=1}^{n} w_{kj} \, x_j + b_j$$

$$y = \sum_{k=1}^{K} w_k \, h \, (\sum_{j=1}^{n} w_{kj} x_j + b_k)$$

$$\frac{\partial y}{\partial x_j} = \sum_{k=1}^{K} w_k \, w_{kj} \, \frac{\partial h(z)}{\partial z} \; = \; \sum_{k=1}^{K} w_k \, w_{kj} h(z_k)\big(1 - h(z_k)\big)$$

For the two hidden layer architecture we use K^1 and K^2 instead of just K, z^2 and z^1 instead of just z and w_{ij}^2, w_{ij}^1 instead of just w_{ij} .

$$y = \sum_{k=1}^{K_2} w_k \, h(z_k^2)$$

$$z_k^2 = \sum_{l=1}^{K1} w_{kl}^2 \, h(z_l^1) + b_k$$

$$z_l^1 = \sum_{j=1}^{n} w_{lj}^1 \, x_j + b_l$$

$$y = \sum_{k=1}^{K_2} w_k \; h\,(\sum_{l=1}^{K_1} w_{kl}^2 \, h(\sum_{j=1}^{n} w_{lj}^1 \, x_j + b_l) + b_k\,)$$

$$\frac{\partial y}{\partial x_j} = \sum_{k=1}^{K_2} w_k \sum_{l=1}^{K_1} w_{kl}^2 \, w_{lj}^1 \, \frac{\partial h(z_k^2)}{\partial z_k^2} \, \frac{\partial h(z_l^1)}{\partial z_l^1}$$

$$\frac{\partial y}{\partial x_j} = \sum_{k=1}^{K_2} w_k \sum_{l=1}^{K_1} w_{kl}^2 \, w_{lj}^1 \, h(z_k^2)\Big(1 - h(z_k^2)\Big)h(z_l^1)\Big(1 - h(z_l^1)\Big)$$

For most problems the one hidden layer model is sufficient to provide a decent solution but a model with a significantly larger VC dimension may become useful in certain cases. Figure 8.1 demonstrates the operation of the algorithm in one dimensional space. Of course in real problems the input space is multi-dimensional but this simplification is inevitable for visualization purposes. The data contains positive class patterns (numbered *5* and *7*) and negative class patterns (numbered *1, 2, 3, 4* and *6*). The first graph (a) shows the output of the model after random weights initialization and marks the operations of the algorithm for each pattern during the first iteration. According to (a), points *1*, *2* and *6* must be reduced to their

local minimum while points *3,4* are not altered because they are already situated on their local minimums with their gradients being zero. Points *5*, *7* must be elevated to their local maximum because they belong to the positive problem class.

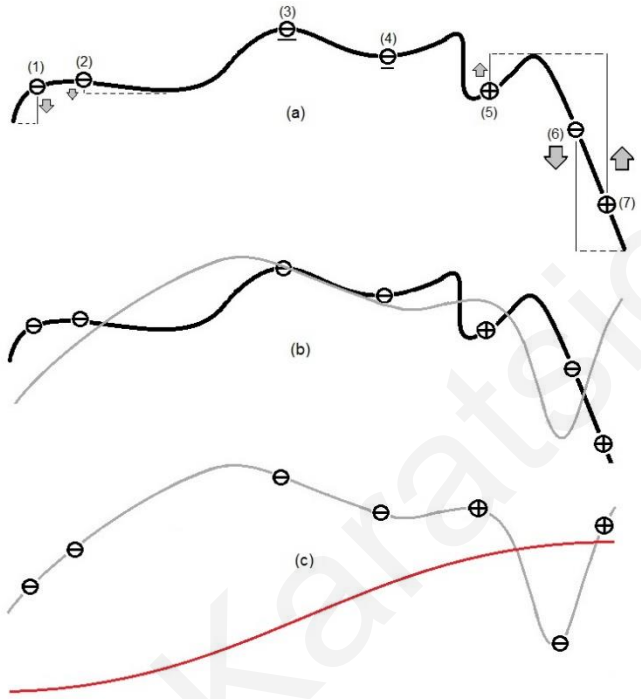


Figure 8.1: The algorithm intuition demonstrated on a one dimensional dataset

(a) Initial model function and example data patterns. The problem classes are either illustrated as positive (+) or negative (-)

(b) The model function after one iteration. The negative class patterns are given the function value of the local minimum and the positive class patterns are given the function value of the local maximum

(c) After several iterations, the model function is smoother, convex-shaped and has the qualities of a good classification function

After the data target values are evaluated they are used to train the regression model. Graph (b) shows the result of the training along with the initial function in order to become more evident that the initial function is improved in terms of the problem at hand in the sense that it becomes smoother and starts to favor the mapping of the negative class at lower values and the mapping of the positive class at higher function values. Graph (c) shows the result of this behavior after several iterations. The function is now really smooth and has become a nice hypothesis for

decision making. Positive patterns tend to be mapped at the high value region of the function while negative class patterns tend to be mapped on the low value region. The outlying pattern *6* regulates the maximum value of the function in its vicinity, reducing the degree of belief for the positive class region. Of course this reduction is reversely analogous to the number of positive class patterns that are present in the area.

A quality characteristic of the algorithm is its natural tendency to smooth the model function. During each iteration local maximums and minimums are consumed by the algorithm and thus their number is progressively shrunk. This characteristic is preserved by using a good regularization term in the regression model which enables the algorithm to camber the function and perform appropriate "convexification" during each iteration.

8.4 Experimental Results

The preeclampsia dataset consists of clinical data collected from *33602* pregnant women and contains *11* attributes: age, race, height, weight, conception type (spontaneous or ovulation drugs), smoking, previous preeclampsia condition during past pregnancies, heredity factor (mother experiencing the condition), chronic hypertension presence, PAP blood test value and β-hCG (human chorionic gonadotropin) blood test value. Out of these cases only *752* were eventually diagnosed with preeclampsia which is translated to the expected *2%-3%* frequency of appearance for this disease. This low frequency constitutes a highly imbalanced dataset.

Approximately *66%* of the normal cases (no preeclampsia) comprising *22000* cases are used for training with the remaining *10850* cases are used for evaluation. The preeclampsia cases are split into *652* cases for training and *100* for evaluation. Using conventional neural networks, the results were really bad because of the imbalanced dataset. Even after extended fine-tuning and application of many different models and architectures the best results obtained did not exceed *38%* for the preeclampsia cases and *78%* for the normal cases. The Support Vector Machine with an RBF kernel produced better results with *42%* success rate for preeclampsia cases and *75%* for the normal cases under the best model (sigma=*4*, box-constraint=*1*). Gaussian discriminant analysis produced the worst rates with *22%* for the preeclampsia and nearly *82%* for the normal cases. These results are concentrated in table 8.2. All results are the outcomes of a *10*-fold cross validation strategy.

TABLE 8.2

Comparison between the test set success rates of the adaptive neural network target values algorithm and other models. The proposed model closes the gap between the small-sized class and the dominant problem class.

Training Model	**Preeclampsia success rate (%)**	**Normal cases Success rate (%)**
SVM (RBF)	41.8	74.7
NN (sigmoid)	29.6	80.6
NN (Gaussian)	37.9	77.8
Gaussian Discriminative Analysis	22.3	82.4
Adaptive NN target values	**49.3**	**76.2**

The proposed algorithm was used to train a neural network having a hidden layer of *8* nodes and during the regression phase an L^2-norm regularization term with a coefficient value of *0.0002* was introduced.

The whole training phase was repeated for *10* iterations and a classification success rate of *49%* for the preeclampsia cases and *76.2%* for the normal cases was accomplished. Compared to the results produced with the conventional approaches, they may be considered quite satisfactory.

8.5 Conclusions

The proposed algorithm seems to provide a reasonable resort for solving highly imbalanced machine learning problems that are difficult to tackle with conventional approaches. While other algorithms seem to have huge difficulties in learning the general structure of the minority class, adaptive neural network target values algorithm is less exposed to the overwhelming effect of the huge majority class. However, the most important problem of such problems is the

unsatisfactory specificity of the calculated model. For every misclassified pattern that belongs to the majority class the overall performance falls drastically in comparison to the successful rate improvement gained by a correctly classified pattern that belongs to the minority class. This is inevitable because of the high imbalance ratio. Especially in the medical field, the impact of having low specificity causing the initiation of expensive thorough examinations and diagnostic procedures can make an otherwise decently performing algorithm useless. Under this in mind, having an imbalanced problem at hand it must be decided whether the bad performance observed from the training procedure is caused by the imbalance itself or the inseparable dataset. If the latter is suspected, then a quest for better problem attributes or a modification of the current ones could be a reasonable approach. After all, a linearly separable dataset experiences a much lower impact from dataset imbalance [164]. Despite that, using adaptation of the target values of the neural network can overcome the imbalance and provide a compromise between learning the underlying data structure of both classes.

Chapter 9

Feature Mapping through maximization of the Atomic Interclass Distances

The method implements feature mapping for classification problems by fitting the given data with a set of functions composed of a mixture of convex functions. In this way, a certain pattern's potential of belonging to a certain class is mapped in a way that promotes interclass separation, data visualization and understanding of the problem's mechanics. In terms of enhancing separation, the algorithm can be used in two ways: to construct problem features for feeding a classification algorithm or to detect a subset of problem attributes that could be safely ignored. In terms of problem understanding, the algorithm can be used for constructing a low dimensional feature mapping in order to make problem visualization possible. The whole approach is based on the derivation of an optimization objective which is solved with a genetic algorithm. The algorithm was tested under various datasets and proved successful in providing improved results. Specifically for Wisconsin breast cancer problem, the algorithm has a success rate of 98% while for Pima Indian diabetes it has a success rate of 82%.

9.1 Introduction

During the many decades of Machine Learning research, a plethora of algorithms have been developed and introduced. While the hypothesis of some algorithms like the Bayes classifier may be considered as self-explanatory and naturally deduced, a discriminative model's hypothesis like the one calculated by a Neural Network or a Support Vector Machine is difficult to explain. This phenomenon is really obvious when we deal with a non-linear boundary hypothesis. Real world machine learning problems are very rarely restricted to a few attributes. On the contrary, recently defined Machine Learning problems tend to consist of hundreds or thousands of attributes and a large number of patterns. This phenomenon is stressed by the lately developed belief among the Machine Learning community that is not the best algorithm that wins at the end of the day but the one trained with the largest dataset. Fortunately, nowadays collecting data is much easier than it was two decades ago. At the same time, Machine Learning has provided ways to reduce data dimensionality thus allowing partial visualization and data analysis. The problem of using many of such methods is that they reduce the dimensionality of the input space meaning that the decision of which part of the information could be thrown away is taken in the data space and not the feature space. Another concern with this approach is the fact that significant information may be inevitably disposed, towards the goal of ending up with two or three (transformed) attributes in order to visualize a portion of the data. This high information disposal may prevent the emergence of important data relationships. Nevertheless, it is generally useful to have a feature mapping that can be constrained to two or three dimensions for visualization purposes. By definition, mapping a rather difficult problem into only a few features means that these features must be highly expressive and reflect the outcome of complex functions in order to be able to encapsulate a great amount of the data inter-relations.

The proposed approach relies on derived data representations (feature mappings) that can efficiently support a classification hypothesis for the problem at hand. Effectively, the data is visualized based on a powerful hypothesis that has success rates that are very close to or even better than the success rates of state of the art Machine Learning algorithms.

9.2 The feature mapping optimization objective

Let's assume a training data set $D = \{(x^1, y^1), (x^2, y^2), \ldots, (x^M, y^M)\}$ consisting of M labeled patterns such as $x^i \in R^n$ and $y^i \in \{1, 2, 3, \ldots, k\}$ meaning that there are k different labels (classes). The basic task is to construct a hypothesis $h: R^n \rightarrow \{1, 2, \ldots, k\}$ that can predict the label of an unseen data pattern. This hypothesis is based on features that map the characteristics of the different classes of the problem. These features are chosen to be a mixture of convex functions having the problem attributes as variables. The decision of using convex functions has to do with the properties of this special group of functions in the sense that they provide mathematical adequacy to the approach. The feature mapping optimization objective is derived assuming only two classes for reasons of simplicity so for this discussion holds that $k = 2$. We will also refer to these classes as the positive and the negative class. To keep the graphs simple, the data patterns are restricted to be one-dimensional so it further holds that $n = 1$. Finally, we define two random variables X_+ and X_-, one for each problem class. Assuming a feature mapping function f then a certain pattern's feature value is in the form of $f(x^i)$. In order to promote inter class separation the expected feature mapping values of the patterns of each class should be separated with a distance d that is as large as possible.

$$| E[f(X_+)] - E[f(X_-)] | \geq d \tag{9.1}$$

The optimization objective should reflect the maximization of separating distance d and is formally defined as

$$\max_f | E[f(X_+)] - E[f(X_-)] | \equiv \max_f d \tag{9.2}$$

Figure 9.1 demonstrates the rationale behind the defined objective function.

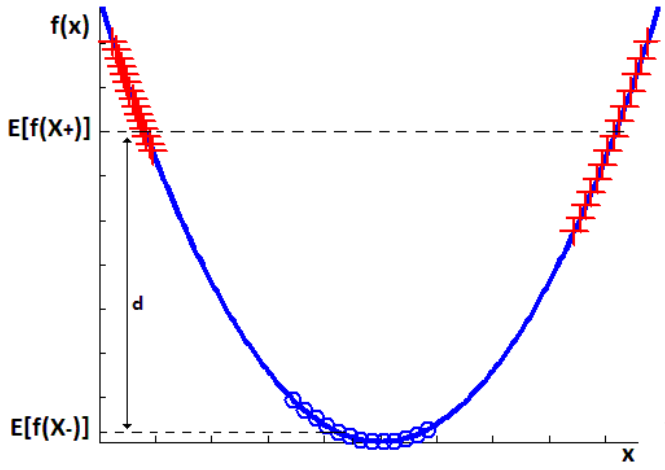


Figure 9.1: The feature mapping of a two class problem with the circles representing the negative class patterns and crosses representing the positive class patterns. The feature function $f(x) = x^2$ maps the patterns in a way that distance d between the expected values of each mapped class is high enough in order to promote class separation. To achieve this goal the feature function must be selected carefully through an optimization process concentrating on getting as large separation as possible.

If the feature functions used in the derived solution are convex or mixtures of convex functions then we can take advantage of Jensen's inequality to establish more strict requirements. More specifically, according to Jensen's inequality for convex functions it holds that

$$f(E[X_+]) \leq E[f(X_+)]\,,\ f(E[X_-]) \leq E[f(X_-)] \tag{9.3}$$

Using this upper bound definition the objective function (9.2) can be redefined as

$$\max_f \ |\, f(E[X_+]) - f(E[X_-])\,| \tag{9.4}$$

The objective function must be made more stringent and more relevant to the desired inter-class separation quality. For this reason we introduce the more sophisticated inter-class distance metric shown in (9.5) which is called Total Atomic Inter-class Distance (TAID). It will be proved that TAID is a stronger objective function than (9.4).

$$\frac{1}{M}\sum_{i=1}^{M} \underset{j}{argmin}\,|\, f(x_+^i) - f(x_-^j)| + \frac{1}{N}\sum_{i=1}^{N} \underset{j}{argmin}\,|\, f(x_+^i) - f(x_-^j)| \tag{9.5}$$

In (9.5), M is the number of the positive class patterns and N is the number of the negative class patterns in the training set. In practice, this distance metric is the sum of the absolute distances

of all feature mappings of the positive patterns to their nearest negative class mapping plus the total individual absolute distances of the feature mappings of the negative patterns to their closest positive class mapping. This distance metric is called the Total Atomic Inter-class Distance (TAID). The proof that the objective function in (9.4) is a subset of the more enriched Total Atomic Inter-class Distance is fairly simple. Given a dataset of two classes the expected value for the positive class data is $E[X_+]$ and the expected value for the negative class data is $E[X_-]$. Their corresponding feature mappings are $f(E[X_+])$ and $f(E[X_-])$ respectively as shown in Figure 9.2. The expected input space values of the two problem classes imply that there are at least four samples placed at minimum distances α, β, γ or δ away from the expected pattern values and have a lower or higher feature value respectively. This increment or decrement in their feature value in respect to the feature value of the expected classes makes them closer to patterns of the opposite class that are placed on the direction of their displacement from the expected values. For example, a positive pattern mapped on a value less than $f(E[X_-])$ is far more likely to be closer to $f(E[X_-] + \delta)$. A positive pattern mapped on a value greater than $f(E[X_-])$ is far more likely to be closer to $f(E[X_-] - \gamma)$. Let's assume that the pattern having a value of $E[X_+] + \beta$ constitutes the closest opposite class feature value to the set of negative class patterns indicated as B that have a lower feature value than $f(E[X_+])$. Symmetrically $f(E[X_+] - \alpha)$ is the closest positive class feature value to the set of negative class patterns indicated as A that have a higher feature value than $f(E[X_+])$. It should be stated again that there may be a number of negative patterns that have feature values that are closer to $f(E[X_+])$ instead of $f(E[X_+] + \beta)$ or $f(E[X_+] - \alpha)$, but this number is expected to be much smaller than the number of the negative patterns that are indeed closer to the latter two patterns. Furthermore, at the vicinity of the solution the number of negative class patterns having feature values closer to $f(E[X_+] - \alpha)$ should be as small as possible in respect to the number of the negative patterns that are closer to $f(E[X_+] + \beta)$. Accordingly, we assume that $f(E[X_-] + \delta)$ constitutes the closest opposite class feature value to the D positive class patterns that have a lower feature value than $f(E[X_-])$ and $f(E[X_-] - \gamma)$ is the closest negative class feature value for the C positive patterns that have a higher feature value than $f(E[X_-])$. Following the latter definitions we divide the training set into four sub areas and further define their corresponding random variables representing the patterns drawn from these areas. As shown in Figure 9.2 we have four random variables one for each area of the data distribution, namely $X_{+\gamma}$, $X_{+\delta}$, $X_{-\alpha}$and $X_{-\beta}$.

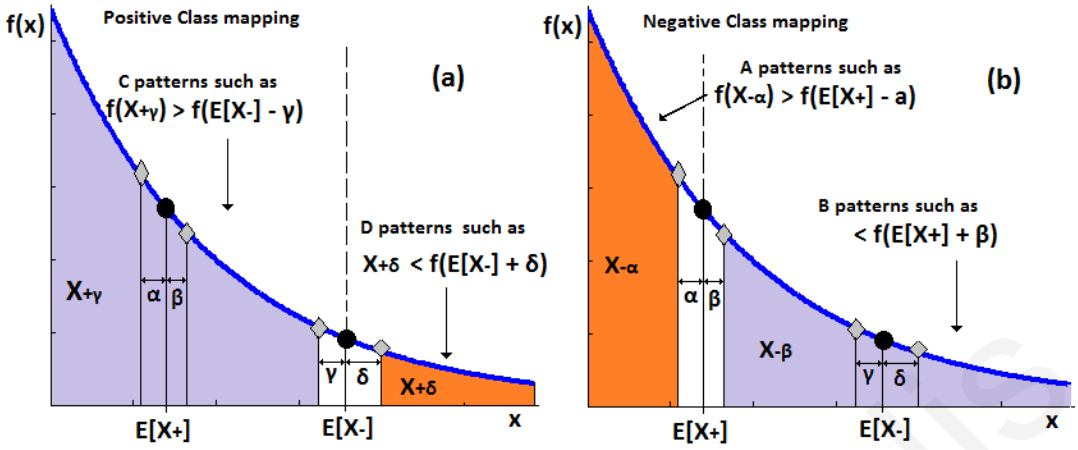


Figure 9.2: In order to examine the properties of equation (9.5) we divide the feature mapping distribution into four areas consisting of A, B, C and D patterns and represented by the random variables $X_{+\gamma}$, $X_{+\delta}$, $X_{-\alpha}$ and $X_{-\beta}$. The first subscript of the random variables represents the class and the second the furthest opposite class pattern.

The first subscript of the random variables represents its class and the second subscript the furthest opposite class pattern. Using these definitions the maximum Total Atomic Inter-class Distance equation (9.5) may be expanded as in (9.6). It is noted that (9.6) reflects the maximum TAID because $f(E[X_+] - k)$, $k \in \{\alpha, \beta, \gamma, \delta\}$ are the furthermost opposite class feature mappings values given $E[X_+]$ and $E[X_-]$.

$$\frac{1}{C}\sum_{i=1}^{C}[f(x_{+\gamma}^{i}) - f(E[X_-] - \gamma)] + \frac{1}{D}\sum_{i=1}^{D}[f(E[X_-] + \delta) - f(x_{+\delta}^{i})] +$$
$$\frac{1}{A}\sum_{i=1}^{A}[f(x_{-\alpha}^{i}) - f(E[X_+] - \alpha)] + \frac{1}{B}\sum_{i=1}^{B}[f(E[X_+] + \beta) - f(x_{-\beta}^{i})] =$$

$$E[f(X_{+\gamma})] - f(E[X_-] - \gamma) + f(E[X_-] + \delta) - E[f(X_{+\delta})] + E[f(X_{-\alpha})] -$$
$$f(E[X_+] - \alpha) + f(E[X_+] + \beta) - E[f(X_{-\beta})] \qquad (9.6)$$

$$f(E[X_-] - \gamma) + f(E[X_+] - \alpha) > f(E[X_-] + \delta) + f(E[X_+] + \beta) \rightarrow$$
$$-f(E[X_-] - \gamma) - f(E[X_+] - \alpha) + f(E[X_-] + \delta) + f(E[X_+] + \beta) \equiv \Sigma < 0$$

Replacing the above quantity back to (9.6) we have

$$TAID = E[f(X_{+\gamma})] - E[f(X_{+\delta})] + E[f(X_{-\alpha})] - E[f(X_{-\beta})] - \Sigma, \quad \Sigma > 0 \qquad (9.7)$$

Equation (9.7) can be further simplified using the following facts

$$E[f(X_{+\gamma})] = f(E[X_{+}]) + \delta_1, \delta_1 \geq 0$$
$$E[f(X_{+\delta})] = f(E[X_{-}]) - \delta_2, \delta_2 > 0$$
$$E[f(X_{-a})] = f(E[X_{+}]) + \delta_3, \delta_3 > 0$$
$$E[f(X_{-\beta})] = f(E[X_{+}]) - \delta_4, \delta_4 \geq 0$$

Substituting in (9.7)

$$\begin{aligned} TAID &= f(E[X_{+}]) + \delta_1 - f(E[X_{-}]) + \delta_2 + f(E[X_{+}]) + \delta_3 - f(E[X_{+}]) + \delta_4 - \Sigma \\ &= 2f(E[X_{+}]) - 2\mathrm{f}(E[X_{-}]) + \Delta - \Sigma \end{aligned} \quad (9.8)$$

Inspecting (9.8), reveals that it essentially includes equation (9.4) as an embedded sub-goal. Since the TAID metric calculates the sum of the minimum feature mapping distance of the patterns of each class to its nearest opposite class feature mapping for a problem having two classes, it is rational to get the multiplier of two for the sub-goal of equation (9.4) in the first term of equation (9.8). Armed with the basic form of the objective function we are ready to begin formalizing the final expression of the problem. There are still some issues to be solved and these concerns are listed below:

- Feature mapping functions should be bounded in order to avoid over fitting and maintain sanity. Fixed upper and lower bounds are implemented which results in rejecting candidate solutions that map any of the data outside this acceptable range. In all experimental runs of the algorithm the bounds are symmetrical in the range $[-1000, 1000]$.

- Regularization should be added in order to avoid over fitting. In this spirit, a regularization term r is added that makes sure that the Σ term never goes to zero or becomes extremely low during the search. In practice, the regularization term reflects the number of the boundary range integer values on which at least one positive or one negative pattern is mapped.

$$r = \sum_{i=\text{-Bound}}^{+\text{Bound}} E(i) \quad , \quad E(i) = \begin{cases} 1 & if \quad \exists x^j : \lfloor f(x^j) \rfloor = i \\ 0 & otherwise \end{cases} \quad (9.9)$$

- The algorithm should have the ability to construct multiple features in order to deal with problems of high complexity. Several features are combined to provide a decent feature mapping. Having a feature f_k already constructed using the objective function, a new feature calculation should consider this feature and perform its own search in a way that consummates all previous work done so far. Considering previous constructed features drives the algorithm

towards applying a new perspective to the problem by examining a different point of view upon the training patterns. To maintain a different angle of view upon the problem, f_{k+1} should map a specific pattern x^j to a different location on the feature range than f_k already did. Towards this direction the distance of the two mappings is considered in the objective. In a formal definition, in order to calculate f_{k+1} the objective function of (9.5) should be modified to reflect the contribution of f_k such as

$$\frac{1}{M}\sum_{i=1}^{M} \underset{\text{j}}{argmin}\ |f_{k+1}(x_+^i) - f_{k+1}(x_-^j)| + \frac{1}{N}\sum_{i=1}^{N} \underset{\text{j}}{argmin}\ |f_{k+1}(x_-^i) - f_{k+1}(x_+^j)| + \frac{1}{M}\sum_{i=1}^{M} \underset{\text{j}}{argmin}\ |f_{k+1}(x_+^i) - f_k(x_-^j)| + \frac{1}{N}\sum_{i=1}^{N} \underset{\text{j}}{argmin} |f_{k+1}(x_-^i) - f_k(x_+^j)| \quad (9.10)$$

Every time a new feature is calculated, new sub-goals in the form of function (9.5) must be included so as to achieve current feature's diversity from the previously constructed features. This increases the complexity of the algorithm making it very inefficient to the extend of losing its practical use. For this reason, the multi- feature objective function (9.10) is modified in such a way that the requirements of the above discussion are preserved without overwhelming the performance of the solver. The new objective function is using Euclidean distance for the calculation of the current feature function. Every feature mapping at a given point is represented by a vector $\mathring{f}(x^i) = [f_1(x^i) f_2(x^i) \dots f_k(x^i), f_{k+1}(x^i)]$ whose elements are all previous feature mappings and the one to be calculated by the current iteration $f_{k+1}(x^i)$. Using this approach we have the following objective function. The last term r is the regularization term defined in (9.9).

$$\frac{1}{M}\sum_{i=1}^{M} \underset{\text{j}}{argmin}\ \|\mathring{f}(x_+^i) - \mathring{f}(x_-^j)\|_2 + \frac{1}{N}\sum_{i=1}^{N} \underset{\text{j}}{argmin}\ \|\mathring{f}(x_-^i) - \mathring{f}(x_+^j)\|_2 + r \quad (9.11)$$

- The last modification to the objective function is a term that adds selectivity on the attributes used to perform the feature mappings. A subset of the input attributes may be selectively used for the calculation of a specific feature based on attributes' suitability, information content and contribution in respect to all available attributes. This means that only a subset of the problem attributes are used for calculating a feature. Limiting the number of involved attributes in the feature construction process, favors the usage of the strongest feature elements during each feature mapping iteration and consequently advances the concentration on a certain angle of view for the problem. This approach is more appropriate when the problem at hand is difficult in the sense that a powerful classification algorithm like a Support Vector Machine or a Neural

Network provides somehow lower than expected success rates. There are also situations that a problem has a great number of attributes and it seems rational to construct many features each of which encapsulates a narrow spectrum of the characteristics carried inside the attributes. In such cases the degree of sparseness may be large in order to promote usage of different attributes during the calculation of each problem feature. This approach is similar to sparse auto-encoders currently used by researchers in deep learning algorithms [176]. In this way, some attributes may not be used at all, implying that they could be neglected in the training of a classification algorithm. The final objective function intended for maximization is shown below

$$\frac{1}{M}\sum_{i=1}^{M} \underset{\mathrm{j}}{argmin}\{\left\|\mathring{f}(x_{+}^{i}) - \mathring{f}(x_{-}^{j})\right\|_2\} + \frac{1}{N}\sum_{i=1}^{N} \underset{\mathrm{j}}{argmin}\{\left\|\mathring{f}(x_{-}^{i}) - \mathring{f}(x_{+}^{j})\right\|_2\} \; + \\ r - s\frac{l}{n}, x \in R^n, f_k \in R^l, 0 \le s \le UpperBound \qquad (9.12)$$

The coefficient of sparseness s defines the degree of attributes negligence and the overall sparseness term depends on how many attributes l are actually involved in a specific feature f_k calculation over the total attributes of the problem. The sparseness coefficient can be tuned accordingly depending on the problem at hand.

9.3 Algorithm Implementation

Having the objective function fully defined in (9.12), this section deals with how to implement an efficient search method. A gradient based method cannot be used because the objective is not a well-defined continuous function with variable parameters and its form cannot be known a priori. The search process should combine and test a great number of convex functions until it reaches a good quality solution. By definition genetic programming algorithms are the best fit for this type of problems. The representation scheme of the genetic programming algorithm consists of a set of convex functions, some primitive operators and some squashing functions. More specifically, the operators used are addition, subtraction, multiplication, division, entropy information, pattern normal value, power function, Gaussian function, logistic function and greater than Boolean functions. The algorithm essentially combines the building blocks of the problem's genome and the goal is to eventually come up with a good feature mapping representation. When a calculation for a pattern falls outside the defined range of the feature mapping, the candidate solution is allocated a very small fitness. An initial population goes

through the genetic operations of crossover and mutation for several generations until convergence to a state that the best population fitness value does not change any more.

9.4 Experimental Results

To test the capabilities of the algorithm we first use *2*-dimensional artificial datasets because of the natural capability of visualizing the feature mapping surface directly. This provides a better understanding of the behavior of the method and reveals its strengths and weaknesses. Then we test the algorithm on real world problems, namely the banknotes dataset, the Wisconsin breast cancer dataset and the Pima Indian diabetes dataset. The first artificial dataset generated consists of two inseparable classes of data distributed as in Figure 9.3. The two features calculated are shown with letters *A* and *B*. We observe that features are of satisfactory quality and map the variability of the data classes. One of the greatest advantages of the proposed algorithm is the smoothness of its mapping. Tangled data is given a feature value that implies less confidence than the feature value of patterns placed far away from the opposite class clusters. This smoothness quality is a characteristic of good solutions that generalize well on unseen data. The feature surface of *B* contributes to the overall problem by emphasizing on specific feature mappings that inspect different relations from the relations examined by *A*.

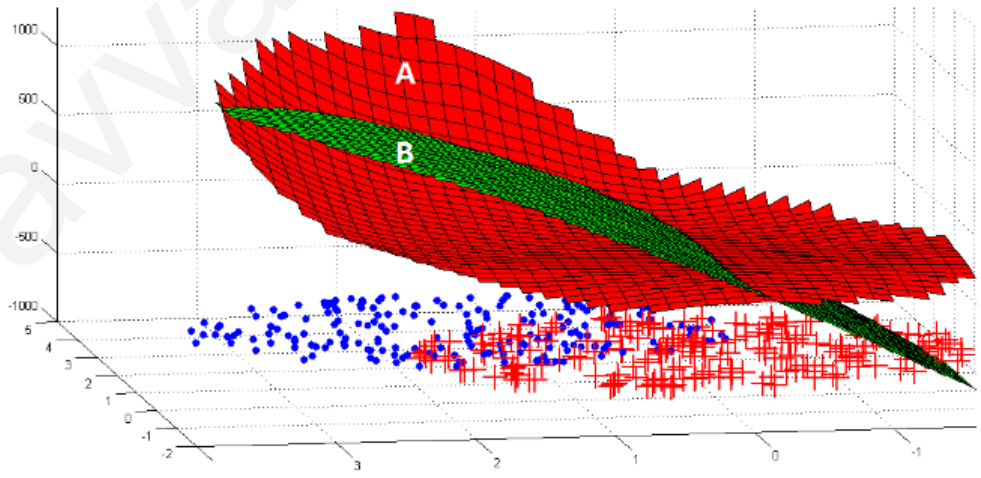


Figure 9.3: Two non-separated classes distributed in a two dimensional space are feature mapped with two features *A* and *B* using Total Atomic Inter-class Distance.

Constructing only a few enriched features and plotting them in respect to each other provides an insight view to the problem. Besides visualization, the features calculated can be fed directly into a machine learning classification algorithm. Especially when the problem has a great number of attributes, highly sparse feature mapping can make a significant difference in terms of classification results as seen later. The second artificial dataset tested consists of three clusters of inseparable data with the positive class placed between two negative class clusters. Figure 9.4 shows the data distribution and the feature mapping results. The third dataset consists of multiple separable clusters of data and is shown in Figure 9.5.

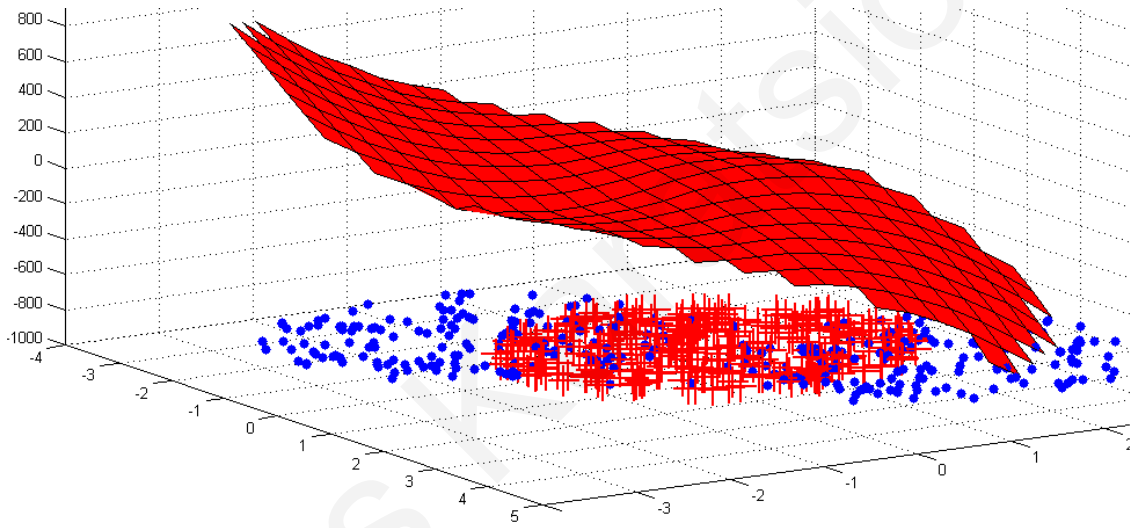


Figure 9.4: The mapping results of a two dimensional inseparable problem with three clusters of data, one belonging to the negative class and the other two to the positive class.

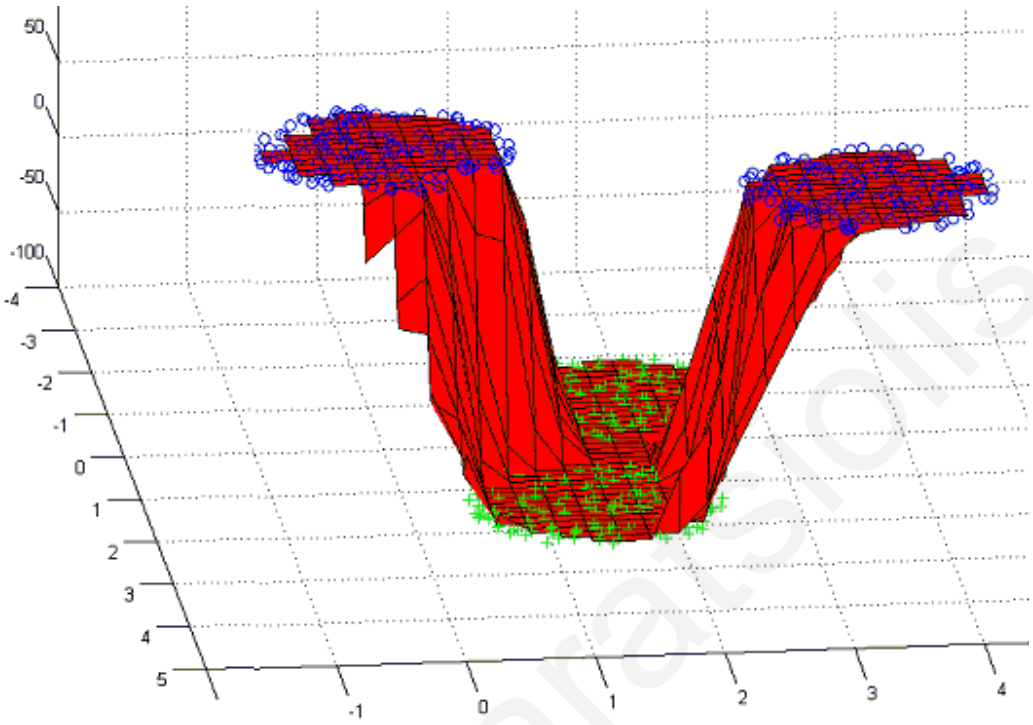


Figure 9.5: The mapping results of a two dimensional problem with four separable clusters of data, two belonging to the negative class and the other two to the positive class.

Besides the use of artificial datasets, the algorithm was applied to real life problems in order to study its behavior under realistic circumstances. For the separable banknotes dataset the algorithm came up with a single feature function that is doing the job effortless. The histogram of the feature mapping function is shown in Figure 9.6.

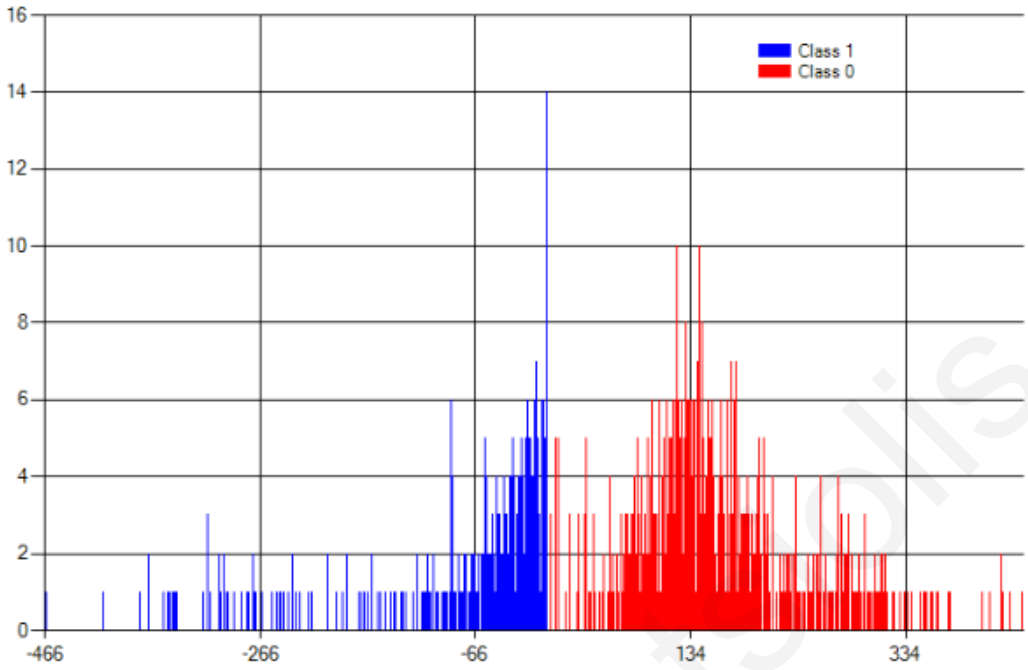


Figure 9.6: The feature mapping histogram of the separable banknotes dataset.

Applying the algorithm to the Pima diabetes dataset with high sparseness reveals that three really strong data attributes (1^{st} , 4^{th}, 8^{th}) can be used with a classifier while ignoring the rest of the attributes. An SVM using only these attributes can solve the problem with an 82% success rate on the test set which is a significant performance boost over the 75.5% of using all attributes. A comparative survey on various classification approaches for the diabetes dataset can be found in [177]. The Wisconsin breast cancer dataset is solved by constructing two features that achieve an improved classification success rate of 98%. The two features and the boundary is shown in Figure 9.7. The experimental results show that the method effectively performs a suitable feature mapping than can be used to advance problem understanding and support the construction of a good classification hypothesis. The next step is to study the possibility of using the algorithm in self-taught feature learning approaches by allowing the absence (partly or totally) of data labeling. In this way, the features will be constructed in respect to naturally occurring underlying relationships and will be enforced by the definition of the problem. A collaborative unsupervised algorithm, for example Expectation-Maximization, could direct the allocation of pseudo-labels during the search, thus discovering data features that govern the problem space. After constructing a number of these inter-relations, supervised learning using the derived problem labels could take over the classification task.

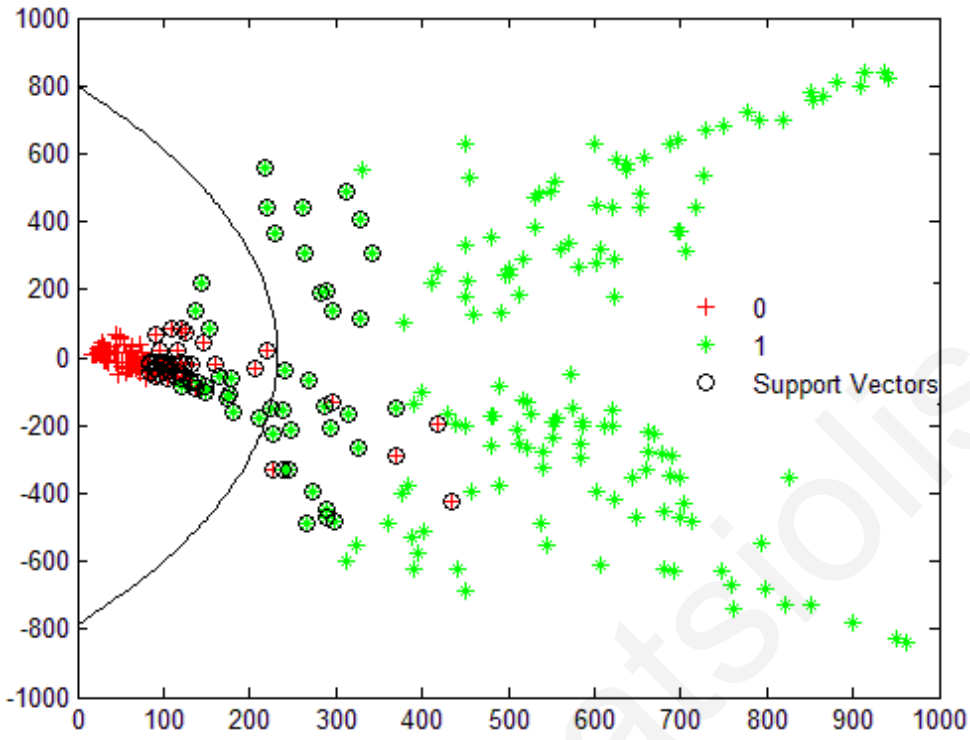


Figure 9.7: Feature mapping for WBC dataset with its SVM decision boundary.

9.5 Conclusions

Studying the outcomes of the implementations, it is evident that the suggested algorithm could be useful during the analysis of a classification problem and can provide some insight on the problem's mechanics that are otherwise well hidden and deeply encapsulated into the data space. The algorithm also enables the construction of a state of the art hypothesis provided that a good tuning of its parameters is accomplished.

Chapter 10

Conclusions and Future Work

10.1 Conclusions

Generative models aim at learning the true data distribution of the training set so as to generate new data points with some variation. This task is often very difficult because of the complex distributions of data in real-life problems. Learning the exact data distribution either explicitly or implicitly may be impossible. Most of the time, neural networks learn a function which only approximates the real distribution. This is one of the main reasons that while supervised learning applied on deep networks is achieving human level performance, at the same time this is not accomplished by unsupervised learning models. Supervised learning is partly easier because learning discrete distributions conditioned on labels is far easier than learning continuous real world distributions. In the case of CG-DAEs (Chapter 5), the model's data distribution approximation function is enriched with supervised learning features. These features enable the generative model to distinguish between the different modes of the data. They also enable the model to perform steps towards matching or differentiating from them.

Usually, conditional generative models directly apply a label for conditioning the sampling process. CG-DAE uses an information-enriched approach to condition the samples drawn from the model. Instead of having a one-shot conditioning based on the label, the model performs a walk towards and around a region of the distribution that satisfies the established criteria. The CG-DAE moves towards the direction of mapping complex distributions in more enriched feature space combining information from two worlds: unsupervised and supervised learning. Data labels provide a means to discover meaningful semantic relations that are important to the problem. Unsupervised learning features are blended with these informative structures into a latent distribution that better captures the data variation. In turn, this helps in overcoming well-known problems of popular generative models like the instability issue of GANs and the blurriness of VAEs.

Since the model is built over the theoretical grounds of denoising autoencoders it is able to perform efficient manifold learning while considering more informative features. It must be noted that the features learned with supervised learning are not directly influenced by the detector structure and only depend on the features of the encoder of the model as shown in Figure 5.1 and Figure 5.2. Likewise, the encoder is not directly influenced by the classifier but is updated using the gradients of the decoder. This means that the classifier influences the encoder indirectly, through the interpretation of the decoder upon the merged features partly provided by the classifier. Consequently, the calculated manifold consists of variables that are not too correlated which would compromise their contribution.

Performing “educated” walks on the data distribution helps revealing more complicated variations like the case of producing samples that have the “Goatee” and “Blond” characteristics at the bottom row of Figure 5.9. Some of these samples tend to capture a rather feminine face with a beard, a combination not learned from the dataset (since it contains no such images) but it is inferred from the labels’ features. The model constructs the “blond” characteristic manifold near the “feminine” characteristics manifold because blond men are much more rare in the dataset. This results in a higher probability that the model chooses feminine characteristics when generating images according to the labels “blond” and “goatee”. This phenomenon does not happen when we ask the model to generate images according to the labels “blond” and “male” since it is explicitly asked to avoid feminine characteristics. However, this proves that the model does a descent job learning the data distribution and the data manifolds and does not memorize the dataset by any means.

The CG-DAE model's performance is at least comparable to the state of the art methods like the GAN and the VAE. It is notable that the image samples obtained from a fully-connected CG-DAE model are comparable to the results of deep convolutional generative networks. This fact alone suggests that the CG-DAE provides a promising direction towards building better generative models.

Image translation from one domain to the other (Chapter 6) is a relatively recent research field mainly because it depends on the advancement of generative models. Most image translation models rely on Generative Adversarial Networks (GAN) for performing the operation. Thus, they inherit the disadvantages of the specific model: mode-collapsing and training instability. The modular domain-to-domain translation network does not rely its operation on a GAN model but it uses pretrained modules that are embedded into a single unified architecture that is fine-tuned with small learning rate. The specific approach of integrating self-contained modules achieves comparative performance with other state-of-the-art models. It also proves that the integration of individually trained modules that perform different tasks can be beneficial for performing a more complex task. Strategically combining several modules capable of performing simple sub-tasks, can lead to models that build upon embedded knowledge and perform complex tasks.

Region based Support Vector Machine algorithm (Chapter 7) builds a model of enhanced generalization ability by deciding and applying the appropriate processing required for a specific pattern. The proposed approach detects two subsets of the dataset, one corresponding to the easy problem cases that are easily separable and the other that contains the most controversial problem cases that are harder to deal with. By using two different models for dealing with these two data subsets, the model disentangles the ways these subsets are conceived by the classifier. The different nature of these subsets requires different feature mapping strategies that recognize solely different data relations.

Adaptive neural network targets (Chapter 8) is an alternative approach for training neural networks and is fundamentally different from any known methodology. The algorithm iteratively updates the input patterns so that the output of the model due to these updated patterns resides on the local minima of the current model function. The local minima of the model function defines the behavior of the classifier because they are controlling the nature of the

discriminating function. By iteratively applying this strategy the model function becomes smoother, gets a convex form and consequently becomes more capable of discriminating between the problem classes.

Feature mapping through maximization of the Atomic Interclass Distances (Chapter 9) produces feature mappings that effectively promote data separation. The algorithm identifies the most important data attributes and builds representations that output values separated by a large margin when patterns belong to different classes.

10.2 Future Work

The proposed CG-DAE model (Chapter 5) may be used for applications requiring the generation of images according to some pre-defined characteristics or categories. For example CG-DAE may be used in the law-enforcement field for assisting investigators in producing a detailed sketch of a suspect. In the field of arts, the CG-DAE could be trained with images of paintings accompanied by artistic characteristics as labels and thus generate art of a specific artistic trend. It could also be used as a means of data augmentation by generating considerably more data for training high performance classifiers. Another interesting application of the CG-DAE may generate images of persons based on photos taken in the past. In this way, the generated images can provide visual information on how the person looks in the time being which may be several years after the original photos were taken. This application can help authorities locate long-missing persons and gather information from witnesses that otherwise could not use a means of describing the appearance of a person in detail.

The most interesting research direction that the CG-DAE could contribute in, is towards semantic understanding of the various image components. Instead or in addition to image categories and characteristics, the CG-DAE could be trained with labels revealing spatial relations or high-level concepts. For example, an object's property of containing another object could be encoded in a label. This would allow the generation of images comprised of objects that contain other objects. In the same sense, any spatial property like "located on top of an object", "next to another object" and "only partially observed" or physical properties like "small" and "big" could allow for a more expressive description of images and their context. High-level concepts inferred from the image content could also act as meaningful pre-conditions

for image generation. An implementation of the CG-DAE of this nature may contribute in making progress towards accomplishing true machine learning understanding.

Modular domain-to-domain translation network (Chapter 6) may provide the stepping stone for building modular networks. Each module may have a high specialization on a subtask of a more complex task and the integration of these smaller tasks can form a highly sophisticated entity. One of the limitations of neural networks is the lack of understanding of how they work and how they build their internal representations. Modular networks can contribute to the direction of solving this problem since a model can be decomposed to smaller self-explanatory modules. Another interesting future research direction for the specific model is the development of its convolutional version. The model in its current form deals exclusively with images and this makes its convolutional implementation very intriguing.

Future work on the region based Support Vector Machine algorithm (Chapter 7) could focus on building a model that separates the data into more than two groups (not just the separable patterns group and the group of more difficult cases). Thus, many hypotheses can act on subsets of a given problem which are created based on the simple criterion that each allocated subset is separable. This may further improve the generalization of the model by enforcing many different angles of attack and exploring relations that would otherwise go unnoticed by the model.

Adaptive neural network targets (Chapter 8) could be applied to problems that do not face the imbalanced dataset problem. Explicitly considering the local minimum and maximum of a model function for fitting the data could be generally beneficial. While the BFGS algorithm seems the best choice for reaching these zero gradient points, gradient descent/ascent could also be tested as the optimizer of the approach, because is simpler and requires less computations and memory. Successful use of the gradient descent/ascent optimizer enables the implementation of the algorithm on much wider and deeper networks, where the BFGS is very expensive to apply (in terms of computations and memory). The algorithm could also be modified to deal with multiclass problems.

Feature mapping through maximization of the Atomic Interclass Distances (Chapter 9) may support a methodology that generates reusable features which can be directly embedded into a new machine learning project without training. The features calculated by the algorithm are easily comprehended functions in the form of mathematical expressions. This enables a researcher to choose between readily available functions that were calculated once and stored

in a database of features according to their outcome and functionality. For example, a specific feature detector like a hair or eyes detector could be calculated and reused for projects that need this kind of feature detector among other types of detectors. Each feature is a self-contained entity and does not depend on other calculations. Thus, these features could comprise the representation network that feeds a neural network classifier.

10.3 Dissemination of Work

The "Conditional Generative Denoising AutoEncoder" (Chapter 5) been accepted for publication in the IEEE Transactions on Neural Networks and Learning Systems (TNNLS) Journal (acceptance letter received on 02/11/2019) and is currently being processed to be published.

The "Modular Domain-to-Domain Translation Network" (Chapter 6) was published in the Journal of Neural Computing and Applications (NCAA) in 2019. It was also previously published in the Proceedings of the 27th International Conference on Artificial Neural Networks and Machine Learning (ICANN) in 2019.

The "Variable Target Values Neural Network for Dealing with Extremely Imbalanced Dataset" (Chapter 7) was published in the Proceedings of the 14th Mediterranean Conference on Medical and Biological Engineering and Computing in 2016.

The "Feature Mapping Through Maximization of the Atomic Interclass Distances" (Chapter 8) was published in the Proceedings of the Conference on Statistical Learning and Data Sciences (SLDS) in 2015.

The "Region Based Support Vector Machine algorithm for medical diagnosis on Pima Indian Diabetes dataset" was published in the Proceedings of the 12th IEEE International Conference on Bioinformatics and Bioengineering (BIBE) n 2012. The specific algorithm also implements a software cache for genetic programming algorithms that was published under the title "Implementing a software cache for genetic programming algorithms for reducing execution time" in the Proceedings of the International Conference on Evolutionary Computation Theory and Applications (ECTA) in 2014.

All published work, either included in this thesis or not, is presented in Appendix A.

References

[1] G. Hatfield, “René Descartes,” The Stanford Encyclopedia of Philosophy, 2018. [Online]. Available: https://plato.stanford.edu/cgi-bin/encyclopedia/archinfo.cgi. [Accessed: 20-Apr-2019].

[2] R. A. Geiger, B. Rudzka-Ostyn, “Conceptualizations and mental processing in language”, De Gruyter, Berlin, Germany, pp. 25–26, 1993.

[3] W. Clark, J. Golinski, S. Schaffer, "Enlightened Automata", The Sciences in Enlightened Europe, University of Chicago Press, pp. 126-165, 1999.

[4] Hebb, D.O. “The Organization of Behavior”, New York, Wiley & Sons,1949.

[5] K. Fukushima, “Neocognitron: A Self-Organizing Neural Network Model for a Mechanism of Pattern Recognition Unaffected by Shift in Position,” Biological Cybernetics, vol. 36, pp. 193–202, 1980.

[6] Y. LeCun, L. Bottou, Y. Bengio and P. Haffner, “Gradient-Based Learning Applied to Document Recognition,” Proceedings of IEEE, 86(11):2278-2324, 1998.

[7] K. Fukushima, “Cognitron: A Self-Organizing Multilayer Neural Network,” Biological Cybernetics, vol. 20, pp. 121–136, 1975.

[8] X. Glorot, A. Bordes, and Y. Bengio, “Deep Sparse Rectifier Neural Networks,” in G. Gordon, D. Dunson, and M. Dudík (Eds.). International Conference on Artificial Intelligence and Statistics 14 (AISTATS), PMLR (Proceedings of Machine Learning Research) W&CP, vol. 15, pp. 315–323, 2011.

[9] K. Jarrett, K. Kavukcuoglu, M. Ranzato, and Y. LeCun, “What is the best multi-stage architecture for object recognition?,” Proceedings of IEEE International Conference on Computer Vision (ICCV), Kyoto, Japan , pp. 2146–2153, 2009.

[10] W. S. Mcculloch and W. H. Pitts, “A Logical Calculus of the Ideas Immanent in Nervous Activity,” Bulletin of Mathematical Biophysics, vol. 5, pp. 115–133, 1943.

[11] A. Hodgkin and A. Huxley, “Currents carried by sodium and potassium ions through the membrane of the giant axon of Loligo,” The Journal of Physiology, vol. 116, no. 4, pp. 449–472, 1952.

[12] F. Rosenblatt, “The Perceptron: A Probabilistic Model for Information Storage and Organization in the Brain,” Psychological Review, vol. 65, no. 6, pp. 386–408, 1958.

[13] D. E. Rumelhart, G. E. Hinton, and R. J. Williams, “Learning representations by back-propagating errors,” Nature, vol. 323, no. 6088, pp. 533–536, 1986.

[14] C. Cortes and V. Vapnik, “Support-Vector Networks.,” Machine Learning, vol. 20, no. 3, pp. 273–297, 1995.

[15] G. E. Hinton, S. Osindero, and Y. W. Teh, "A fast learning algorithm for deep belief nets," Neural Computation, vol. 18, pp. 1527–1554, 2006.

[16] Y. Bengio, P. Lamblin, D. Popovici, and H. Larochelle, "Greedy Layer-Wise Training of Deep Networks," in B. Schölkopf and J.C. Platt and T. Hoffman (Eds.). Advances in Neural Information Processing 19 (Neural Information Processing Systems - NIPS), MIT Press, Cambridge, MA, pp. 153–160, 2006

[17] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR), Las Vegas, NV, pp. 770–778, 2016.

[18] J. Long, E. Shelhamer, and T. Darrell, "Fully convolutional networks for semantic segmentation," Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR), Boston, MA, pp. 3431–3440, 2015.

[19] H A Haenssle, C Fink, R Schneiderbauer, F Toberer, T Buhl, A Blum, A Kalloo, A Ben Hadj Hassen, L Thomas, A Enk, L Uhlmann, Reader study level-I and level-II Groups, "Man against machine: diagnostic performance of a deep learning convolutional neural network for dermoscopic melanoma recognition in comparison to 58 dermatologists", Annals of Oncology, vol. 29, Issue 8, Pages 1836–1842, 2018

[20] B. Lorica and P. Nathan, "AI Adoption in the Enterprise How Companies Are Planning and Prioritizing AI Projects in Practice," 2019. [Online]. Available: https://www.oreilly.com/data/free/ai-adoption-in-the-enterprise.csp. [Accessed: 05-Mar-2019].

[21] I. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville and Y. Bengio, "Generative Adversarial Nets," in Z. Ghahramani and M. Welling and C. Cortes and N. D. Lawrence and K. Q. Weinberger (Eds.) Advances in Neural Information Processing 27 (Neural Information Processing Systems - NIPS). Curran Associates, Inc., pp. 2672–2680, 2014.

[22] M. D. Zeiler and R. Fergus, "Visualizing and Understanding Convolutional Networks," The Computing Research Repository (CoRR), abs/1311.2901, 2013. [Online]. Available: http://arxiv.org/abs/1311.2901. [Accessed: 12-May-2019].

[23] L. van der Maaten and G. Hinton, "Visualizing Data using t-SNE," Journal of Machine Learning Research (JMLR), vol. 9, pp. 2579–2605, 2008.

[24] J. T. Springenberg, A. Dosovitskiy, T. Brox, and M. A. Riedmiller, "Striving for Simplicity: The All Convolutional Net," Proceedings of the 3rd International Conference on Learning Representations (ICLR), Workshop Track Proceedings, San Diego, CA, 2015.

[25] K. Simonyan, A. Vedaldi, and A. Zisserman, “Deep Inside Convolutional Networks: Visualising Image Classification Models and Saliency Maps,” in 2nd International Conference on Learning Representations (ICLR), Workshop Track Proceedings, Banff, AB, Canada, 2014.

[26] A. Mahendran and A. Vedaldi, “Understanding deep image representations by inverting them,” Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR), Boston, MA, pp. 5188–5196, 2015.

[27] C. Szegedy, W. Zaremba, I. Sutskever, J. Bruna, D. Erhan, I. Goodfellow, and R. Fergus,, “Intriguing properties of neural networks,” Proceedings of the 2nd International Conference on Learning Representations (ICLR), 2014. [Online]. Available: http://arxiv.org/abs/1312.6199. [Accessed: 12-May-2019].

[28] T. B. Brown, D. Mané, A. Roy, M. Abadi, and J. Gilmer, “Adversarial Patch,” The Computing Research Repository (CoRR), abs/1712.09665, 2017. [Online]. Available: http://arxiv.org/abs/1712.09665. [Accessed: 12-May-2019].

[29] G. E. Hinton, “Ask me anything,” 2014. [Online]. Available: https://www.reddit.com/r/MachineLearning/comments/2lmo0l/ama_geoffrey_hinton/. [Accessed: 01-Apr-2019].

[30] Y. Bengio, “What it the best level to model the Mind-brain?,” Conference of Cognitive Computational Neuroscience (CNN), Closing Keynote Debate, 2017. [Online]. Available: https://www.youtube.com/watch?v=FhRW77rZUS8. [Accessed: 02-Apr-2019].

[31] Y. LeCun, “How does the brain learn so much so quickly?,” Conference of Cognitive Computational Neuroscience (CNN), Opening Keynote Debate, 2017. [Online]. Available: https://www.youtube.com/watch?v=cWzi38-vDbE. [Accessed: 02-Apr-2019].

[32] V. Natoli, “Why 2016 is the most important year in HPC in over two decades,” HPCWire, 2016. [Online]. Available: https://www.hpcwire.com/2016/08/23/2016-important-year-hpc-two-decades/#. [Accessed: 02-Apr-2019].

[33] D. H. Hubel and T. N. Wiesel, “Receptive Fields of Single Neurons in the Cat’s Striate Cortex,” Journal of Physiology, vol. 148, pp. 574–591, 1959.

[34] D. H. Hubel and T. N. Wiesel, “Receptive Fields, Binocular Interaction and Functional Architecture in the Cat’s Visual Cortex,” Journal of Physiology , vol. 160, pp. 106–154, 1962.

[35] M. Lin, Q. Chen, and S. Yan, “Network In Network,” in Proceedings of International Conference on Learning Representations (ICLR), vol. abs/1312.4400, 2013. [Online]. Available: http://arxiv.org/abs/1312.4400. [Accessed: 12-May-2019].

[36] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner, “Gradient-based learning applied to document recognition,” Proceedings of the IEEE, vol. 86, no. 11, pp. 2278–2324, 1998.

[37] A. Krizhevsky, I. Sutskever and G. E. Hinton, ‘ImageNet Classification with Deep Convolutional Neural Networks’, in F. Pereira and C. J. C. Burges and L. Bottou and K. Q. Weinberger (Eds.). Advances in Neural Information Processing 25 (Neural Information Processing Systems – NIPS), Curran Associates, Inc., pp. 1097–1105, 2012.

[38] K. He, X. Zhang, S. Ren, and J. Sun, “Spatial Pyramid Pooling in Deep Convolutional Networks for Visual Recognition.,” Journal of IEEE Transactions on Pattern Analysis and Machine Intelligence, vol. 37, no. 9, pp. 1904–1916, 2015.

[39] I. Goodfellow, Y. Bengio, and A. Courville, Deep Learning. MIT Press, MA, United States, 2016.

[40] S. Sabour, N. Frosst, and G. E. Hinton, “Dynamic Routing Between Capsules,” in I. Guyon and U. V. Luxburg and S. Bengio and H. Wallach and R. Fergus and S. Vishwanathan and R. Garnett (Eds.). Advances in Neural Information Processing 30 (Neural Information Processing Systems - NIPS), Curran Associates, Inc., pp. 3856–3866, 2017.

[41] G. E. Hinton, S. Sabour, and N. Frosst, “Matrix capsules with EM routing.,” International Conference on Learning Representations (ICLR), 2018. [Online]. Available: https://openreview.net/forum?id=HJWLfGWRb. [Accessed: 12-May-2019].

[42] N. Srivastava, G. E. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov, “Dropout: A Simple Way to Prevent Neural Networks from Overfitting,” Journal of Machine Learning Research (JMLR), vol. 15, pp. 1929–1958, 2014.

[43] K. Simonyan and A. Zisserman, “Very Deep Convolutional Networks for Large-Scale Image Recognition,” International Conference on Learning Representations (ICLR), vol. abs/1409.1556, 2015. [Online]. Available: http://arxiv.org/abs/1409.1556.

[44] C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. E. Reed, D. Anguelov, D. Erhan, V. Vanhoucke and A. Rabinovich, “Going Deeper with Convolutions,” The Computing Research Repository (CoRR), vol. abs/1409.4842, 2014. [Online]. Available: http://arxiv.org/abs/1409.4842. [Accessed: 12-May-2019].

[45] K. He, X. Zhang, S. Ren, and J. Sun, “Deep Residual Learning for Image Recognition,” in Conference on Computer Vision and Pattern Recognition (CVPR), IEEE Computer Society, Las Vegas, NV, pp. 770–778, 2016.

[46] K. He, X. Zhang, S. Ren, and J. Sun, “Identity Mappings in Deep Residual Networks.,” The Computing Research Repository (CoRR), vol. abs/1603.05027, 2016. [Online]. Available: http://arxiv.org/abs/1603.05027. [Accessed: 12-May-2019].

[47] G. Urban, K. Geras, S. E. Kahou, Ö. Aslan, S. Wang, R. Caruana, A. Mohamed, M. Philipose, and M. Richardson, “Do Deep Convolutional Nets Really Need to be Deep (Or Even Convolutional)?,” The Computing Research Repository (CoRR), vol. abs/1603.05691, 2016. [Online]. Available: http://arxiv.org/abs/1603.05691. [Accessed: 12-May-2019].

[48] C. Szegedy, S. Ioffe, and V. Vanhoucke, “Inception-v4, Inception-ResNet and the Impact of Residual Connections on Learning,” The Computing Research Repository (CoRR), vol. abs/1602.07261, 2016. [Online]. Available: http://arxiv.org/abs/1603.07261. [Accessed: 12-May-2019].

[49] X. Zhang, Z. Li, C. C. Loy, and D. Lin, “PolyNet: A Pursuit of Structural Diversity in Very Deep Networks.,” Proceedings of the Conference on Computer Vision and Pattern Recognition (CVPR), Honolulu, HI, pp. 3900–3908, 2017.

[50] A. Veit, M. J. Wilber, and S. J. Belongie, “Residual Networks are Exponential Ensembles of Relatively Shallow Networks,” The Computing Research Repository (CoRR), vol. abs/1605.06431, 2016. [Online]. Available: http://arxiv.org/abs/1605.06431. [Accessed: 12-May-2019].

[51] K. N. Thanh, C. Fookes, A. Ross, and S. Sridharan, “Iris Recognition With Off-the-Shelf CNN Features: A Deep Learning Perspective.,” IEEE Access, vol. 6, pp. 18848–18855, 2018.

[52] G. Huang, Z. Liu, L. van der Maaten, and K. Q. Weinberger, “Densely Connected Convolutional Networks,” Proceedings of Conference on Computer Vision and Pattern Recognition (CVPR), Honolulu, HI, pp. 2261–2269, 2017.

[53] G. Larsson, M. Maire, and G. Shakhnarovich, “FractalNet: Ultra-Deep Neural Networks without Residuals,” The Computing Research Repository (CoRR), vol. abs/1605.07648, 2016. [Online]. Available: http://arxiv.org/abs/1605.07648. [Accessed: 12-May-2019].

[54] Z. Chen and X. Huang, “End-to-end learning for lane keeping of self-driving cars.,” Intelligent Vehicles Symposium, Los Angeles, CA, pp. 1856–1860, 2017.

[55] M. Bojarski et al., “End to End Learning for Self-Driving Cars,” The Computing Research Repository (CoRR), vol. abs/1604.07316, 2016. [Online]. Available: http://arxiv.org/abs/1605.07316. [Accessed: 12-May-2019].

[56] B. Huval et al., “An Empirical Evaluation of Deep Learning on Highway Driving,” The Computing Research Repository (CoRR), vol. abs/1504.01716, 2015. [Online]. Available: http://arxiv.org/abs/1605.01716. [Accessed: 12-May-2019].

[57] N. Lee, W. Choi, P. Vernaza, C. B. Choy, P. H. S. Torr, and M. K. Chandraker, "DESIRE: Distant Future Prediction in Dynamic Scenes with Interacting Agents," Proceedings of Conference on Computer Vision and Pattern Recognition (CVPR), Honolulu, HI, pp. 2165--2174, 2017.

[58] A. Rajkomar et al., "Scalable and accurate deep learning for electronic health records," The Computing Research Repository (CoRR), vol. abs/1801.07860, 2018. [Online]. Available: http://arxiv.org/abs/1801.07860. [Accessed: 12-May-2019].

[59] R. Miotto, F. Wang, S. Wang, X. Jiang, and J. T. Dudley, "Deep learning for healthcare: review, opportunities and challenges," Briefings in Bioinformatics, vol. 19, no. 6, pp. 1236–1246, 2018.

[60] P. Thodoroff, J. Pineau, and A. Lim, "Learning Robust Features using Deep Learning for Automatic Seizure Detection," Proceedings of Machine Learning in Health Care (MLHC), Los Angeles, CA, pp. 178–190, 2016.

[61] M. Chen, H. Yixue, K. Hwang, L. Wang, and L. Wang, "Disease Prediction by Machine Learning Over Big Data From Healthcare Communities," IEEE Access, vol. 5, pp. 8869--8879, 2017.

[62] I. J. Goodfellow, D. Warde-Farley, M. Mirza, A. Courville, and Y. Bengio, "Maxout Networks," Proceedings of the 30th International Conference on Machine Learning (ICML), Atlanta, GA, vol. 28, pp. 1319–1327, 2013.

[63] I. J. Goodfellow, J. Shlens, and C. Szegedy, "Explaining and Harnessing Adversarial Examples," Proceedings of International Conference on Learning Representations (ICLR), vol. abs/1412.6.572, 2015. [Online]. Available: http://arxiv.org/abs/1605.01716. [Accessed: 12-May-2019].

[64] N. Papernot, P. D. McDaniel, S. Jha, M. Fredrikson, Z. B. Celik, and A. Swami, "The Limitations of Deep Learning in Adversarial Settings," Proceedings of IEEE European Symposium on Security and Privacy, Saarbrucken, Germany, pp. 372–387, 2016.

[65] NVIDIA, "Scalable AI Platform for Autonomous Driving," NVIDIA Drive, 2019. [Online]. Available:https://www.nvidia.com/en-us/self-driving-cars/drive-platform/%0D. [Accessed: 10-May-2019].

[66] L. Chen, S. Hou, and Y. Ye, "SecureDroid: Enhancing Security of Machine Learning-based Detection Against Adversarial Android Malware Attacks," Proceedings of Computer Security Applications Conference, Vancouver, BC, pp. 362–372, 2017.

[67] G. E. Dahl, J. W. Stokes, L. Deng, and D. Yu, "Large-scale malware classification using random projections and neural networks," Proceedings of IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP), Las Vegas, NV, pp. 3422–3426, 2013.

[68] S.-M. Moosavi-Dezfooli, A. Fawzi, and P. Frossard, "DeepFool: A Simple and Accurate Method to Fool Deep Neural Networks," Proceedings of IEEE Conference on Computer Vision and Pattern Recognition (CVPR), Las Vegas, NV, pp. 2574–2582, 2016.

[69] S.-M. Moosavi-Dezfooli, A. Fawzi, O. Fawzi, and P. Frossard, "Universal adversarial perturbations," The Computing Research Repository (CoRR), vol. abs/1610.08401, 2016. [Online]. Available: http://arxiv.org/abs/1605.08401. [Accessed: 12-May-2019].

[70] N. Papernot, P. D. McDaniel, X. Wu, S. Jha, and A. Swami, "Distillation as a Defense to Adversarial Perturbations Against Deep Neural Networks," Proceedings of IEEE Symposium on Security and Privacy, San Jose, CA, pp. 582–597, 2016.

[71] J. Su, D. V. Vargas, and K. Sakurai, "One Pixel Attack for Fooling Deep Neural Networks," Journal of IEEE Transactions on Evolutionary Computation, vol. 23, no. 5, pp. 828–841, 2019.

[72] G. Hinton, O. Vinyals, and J. Dean, "Distilling the Knowledge in a Neural Network," The Computing Research Repository (CoRR), vol. abs/1503.02531, 2015. [Online]. Available: http://arxiv.org/abs/1503.02531. [Accessed: 12-May-2019].

[73] J. Ba and R. Caruana, "Do Deep Nets Really Need to be Deep?," in Z. Ghahramani and M. Welling and C. Cortes and N. D. Lawrence and K. Q. Weinberger (Eds.). Advances in Neural Information Processing 27 (Neural Information Processing Systems - NIPS), Curran Associates, Inc., pp. 2654–2662, 2014.

[74] D. G. Mayer, B. P. Kinghorn, and A. A. Archer, "Differential evolution – an easy and efficient evolutionary algorithm for model optimization," Agricultural Systems, vol. 83, no. 3, pp. 315–328, 2005.

[75] J. T. Springenberg, A. Dosovitskiy, T. Brox, and M. A. Riedmiller, "Striving for Simplicity: The All Convolutional Net," Proceedings of the 3rd International Conference on Learning Representations (ICLR), Workshop Track Proceedings, San Diego, CA, 2015.

[76] K. Hornik, M. Stinchcombe, and H. White, "Multilayer feedforward networks are universal approximators," Neural Networks, vol. 2, no. 5, pp. 359–366, 1989.

[77] H. Hosseini, B. Xiao, M. Jaiswal, and R. Poovendran, "On the Limitation of Convolutional Neural Networks in Recognizing Negative Images," Proceedings of International Conference on Machine Learning and Applications (ICMLA), Cancun, Mexico, pp. 352–358, 2017.

[78] C. Zhang, S. Bengio, M. Hardt, B. Recht, and O. Vinyals, "Understanding deep learning requires rethinking generalization," ArXiv e-prints, 2016. [Online]. Available: http://arxiv.org/abs/1611.03530. [Accessed: 12-May-2019].

[79] D. Arpit, S. Jastrzębski, N. Ballas, D. Krueger, E. Bengio, M. S. Kanwal, T. Maharaj, A. Fischer, A. Courville, Y. Bengio and S. Lacoste-Julien, "A Closer Look at Memorization in Deep Networks," in D. Precup and Y. W. Teh (Eds.). Proceedings of International Conference on Machine Learning 34 (ICML), PMLR, Sydney, NSW, Australia, vol. 70, pp. 233–242, 2017.

[80] D. Krueger, N. Ballas, S. Jastrzkebski, D. Arpit, M. S. Kanwal, T. Maharaj, E. Bengio, A. Fischer and A. C. Courville, "Deep Nets Don't Learn via Memorization," International Conference on Learning Representations (ICLR), 2017. [Online]. Available: https://openreview.net/forum?id=rJv6ZgHYg. [Accessed: 12-May-2019].

[81] X. Zhu, Z. Ghahramani, and J. D. Lafferty, "Semi-Supervised Learning Using Gaussian Fields and Harmonic Functions," in International Conference on Machine Learning (ICML), Washington, DC, pp. 912–919, 2003.

[82] G. E. Hinton, "'What is wrong with convolutional neural nets ?,'" Brain & Cognitive Sciences, MIT Fall Colloquium Series, 2014. [Online]. Available: https://www.youtube.com/watch?v=rTawFwUvnLE. [Accessed: 12-May-2019].

[83] D. P. Kingma and M. Welling, "Auto-Encoding Variational Bayes," International Conference on Learning Representations (ICLR), 2014. [Online]. Available: http://arxiv.org/abs/1312.6114. [Accessed: 12-May-2019].

[84] D. J. Im, S. Ahn, R. Memisevic, and Y. Bengio, "Denoising Criterion for Variational Auto-Encoding Framework," The Computing Research Repository (CoRR), abs/1511.06406, 2015. [Online]. Available: http://arxiv.org/abs/1511.06406. [Accessed: 12-May-2019].

[85] X. Hou, L. Shen, K. Sun, and G. Qiu, "Deep feature consistent variational autoencoder," Proceedings of IEEE Winter Conference on Applications of Computer Vision (WACV), Santa Rosa, CA, pp. 1133–1141, 2017.

[86] B. Blagojevic, "Generating Letters Using Generative Adversarial Networks," ml-everything, 2018. [Online]. Available: https://medium.com/ml-everything/generating-letters-using-generative-adversarial-networks-gans-161b0be3c229. [Accessed: 12-May-2019].

[87] M. Mirza and S. Osindero, "Conditional Generative Adversarial Nets," The Computing Research Repository (CoRR), abs/1411.1784 2014. [Online]. Available: http://arxiv.org/abs/1411.1784. [Accessed: 12-May-2019].

[88] E. Denton, S. Chintala, A. Szlam, and R. Fergus, "Deep Generative Image Models using a Laplacian Pyramid of Adversarial Networks," Advances in C. Cortes and N. D. Lawrence and D. D. Lee and M. Sugiyama and R. Garnett (Eds.). Advances in Neural Information Processing 28 (Neural Information Processing Systems - NIPS), Curran Associates, Inc., pp. 1486–1494, 2015.

[89] E. L. Denton, S. Gross, and R. Fergus, “Semi-Supervised Learning with Context-Conditional Generative Adversarial Networks,” The Computing Research Repository (CoRR), abs/1611.06430 2016. [Online]. Available: http://arxiv.org/abs/1611.06430. [Accessed: 12-May-2019].

[90] P. Isola, J.-Y. Zhu, T. Zhou, and A. Efros, “Image-to-Image Translation with Conditional Adversarial Networks,” Proceedings of IEEE Conference on Computer Vision and Pattern Recognition (CVPR), Honolulu, HI, pp. 5967–5976, 2017.

[91] L. Wolf, Y. Taigman, and A. Polyak, “Unsupervised Creation of Parameterized Avatars,” Proceedings of International Conference on Computer Vision (ICCV), Venice, Italy, pp. 1539–1547, 2017.

[92] T. Salimans, I. Goodfellow, W. Zaremba, V. Cheung, A. Radford, and X. Chen, “Improved Techniques for Training GANs,” in D. D. Lee and M. Sugiyama and U. V. Luxburg and I. Guyon and R. Garnett (Eds.). Advances in Neural Information Processing 29 (Neural Information Processing Systems - NIPS), Curran Associates, Inc., pp. 2226–2234, 2016.

[93] A. Rasmus, H. Valpola, and M. Berglund, “Semi-Supervised Learning with Ladder Network,” in C. Cortes and N. D. Lawrence and D. D. Lee and M. Sugiyama and R. Garnett (Eds.). Advances in Neural Information Processing 28 (Neural Information Processing Systems - NIPS), Curran Associates, Inc., pp. 3546–3554, 2015.

[94] M. Arjovsky, S. Chintala, and L. Bottou, “Wasserstein Generative Adversarial Networks,” Proceedings of International Conference on Machine Learning (ICML), Sydney, NSW, Australia, pp. 214–223, 2017.

[95] I. Gulrajani, F. Ahmed, M. Arjovsky, V. Dumoulin, and A. C. Courville, “Improved Training of Wasserstein GANs,” in I. Guyon and U. V. Luxburg and S. Bengio and H. Wallach and R. Fergus and S. Vishwanathan and R. Garnett (Eds.). Advances in Neural Information Processing 30 (Neural Information Processing Systems - NIPS), Curran Associates, Inc., pp. 5767--5777, 2017.

[96] M. Ranzato, Y.-L. Boureau, and Y. LeCun, “Sparse Feature Learning for Deep Belief Networks,” in J. C. Platt and D. Koller and Y. Singer and S. T. Roweis (Eds.), Advances in Neural Information Processing 20 (Neural Information Processing Systems - NIPS), Curran Associates, Inc., pp. 1185–1192, 2007.

[97] S. Rifai, P. Vincent, X. Muller, X. Glorot, and Y. Bengio, “Contractive Auto-Encoders: Explicit Invariance During Feature Extraction,” Proceedings of International Conference on Machine Learning 28 (ICML), Bellevue, WA, pp. 833–840, 2011.

[98] P. Vincent, H. Larochelle, Y. Bengio, and P.-A. Manzagol, “Extracting and composing robust features with denoising autoencoders,” Proceedings of International conference on Machine learning 25 (ICML), Helsinki, Finland, pp. 1096–1103, 2008.

[99] P. Vincent, H. Larochelle, I. Lajoie, Y. Bengio, and P.-A. Manzagol, "Stacked Denoising Autoencoders: Learning Useful Representations in a Deep Network with a Local Denoising Criterion," Journal of Machine Learning Research (JMLR), vol. 11, pp. 3371–3408, 2010.

[100] A. Hyvärinen, "Estimation of Non-Normalized Statistical Models by Score Matching," Journal of Machine Learning Research (JMLR), vol. 6, pp. 695–709, 2005.

[101] G. Alain and Y. Bengio, "What regularized auto-encoders learn from the data-generating distribution," Journal of Machine Learning Research (JMLR), vol. 15, no. 1, pp. 3563–3593, 2014.

[102] G. E. Hinton and T. J. Sejnowski, "Learning and Relearning in Boltzmann Machines," in D. E. Rumelhart and J. L. Mcclelland, (Eds.). Proceedings of Parallel Distributed Processing: Explorations in the Microstructure of Cognition, MIT Press, Cambridge, MA, vol. 1, pp. 282–317, 1986.

[103] G. E. Hinton, S. Osindero, and Y. W. Teh, "A fast learning algorithm for deep belief nets," Neural Computation, vol. 18, pp. 1527–1554, 2006.

[104] "Google trends, Explore what the world is searching," 2019. [Online]. Available: https://trends.google.com/trends/. [Accessed: 12-May-2019].

[105] D. J. Bartholomew, Principal Components Analysis, International Encyclopedia of Education, 3rd Edition, Elsevier Science, pp. 374-377, 2010.

[106] G. E. Hinton and R. R. Salakhutdinov, "Reducing the Dimensionality of Data with Neural Networks," Science, vol. 313, no. 5786, pp. 504–507, 2006.

[107] R. Salakhutdinov and G. E. Hinton, "Semantic hashing," Journal of Approximate Reasoning (JAR), vol. 50, no. 7, pp. 969–978, 2009.

[108] A. Krizhevsky and G. E. Hinton, "Using Very Deep Autoencoders for Content-Based Image Retrieval," Proceedings of European Symposium on Artificial Neural Networks (ESANN), 2011. [Online]. Available: https://www.elen.ucl.ac.be/Proceedings/esann/esannpdf/es2011-10.pdf. [Accessed: 12-May-2019].

[109] K. He, X. Zhang, S. Ren, and J. Sun, "Delving Deep into Rectifiers: Surpassing Human-Level Performance on ImageNet Classification," Proceedings of International Conference on Computer Vision (ICCV), Santiago, Chile, pp. 1026–1034, 2015.

[110] X. Glorot and Y. Bengio, "Understanding the difficulty of training deep feedforward neural networks," Proceedings of International Conference on Artificial Intelligence and Statistics (AISTATS), Sardinia, Italy, vol. 9, pp. 249–256, 2010.

[111] J. Ma, R. P. Sheridan, A. Liaw, G. E. Dahl, and V. Svetnik, “Deep Neural Nets as a Method for Quantitative Structure-Activity Relationships.,” Journal of Chemical Information and Modeling (JCIM), vol. 55, no. 2, pp. 263–274, 2015.

[112] D. Erhan, P.-A. Manzagol, Y. Bengio, S. Bengio, and P. Vincent, “The Difficulty of Training Deep Architectures and the Effect of Unsupervised Pre-Training,” Journal of Machine Learning Research (JMLR), vol. 5, pp. 153–160, 2009.

[113] J. B. Tenenbaum, V. Silva, and J. C. Langford, “A Global Geometric Framework for Nonlinear Dimensionality Reduction,” Science, vol. 290, no. 5500, pp. 2319–2323, 2000.

[114] P. Z. Peebles, Probability Random variables, and random signal principles. McGraw-Hill Education, OH, United States, 2001.

[115] O. Perron, “Zur Theorie der Matrices,” Mathematische Annalen, vol. 64, no. 2, pp. 248–263, 1907.

[116] G. Frobenius, “Über Matrizen aus nicht negativen Elementen,” in Göttingen : Königliche Gesellschaft der Wissenschaften, ETH-Bibliothek Zürich, Rar 1524, pp. 456–477, 1912.

[117] G. Strang, Introduction to Linear Algebra, Fourth. Wellesley, MA: Wellesley-Cambridge Press, Wellesley, MA, 2009.

[118] G. Casella and E. I. George, “Explaining the Gibbs Sampler,” The American Statistician, vol. 46, no. 3, pp. 167–174, 1992.

[119] A. E. Gelfand and A. F. M. Smith, “Sampling-Based Approaches to Calculating Marginal Densities,” Journal of the American Statistical Association (JASA), vol. 85, no. 410, pp. 398–409, 1990.

[120] P. Vincent, H. Larochelle, Y. Bengio, and P.-A. Manzagol, “Extracting and composing robust features with denoising autoencoders,” Proceedings of International conference on Machine learning (ICML), Helsinki, Finland, pp. 1096–1103, 2008.

[121] P. Vincent, H. Larochelle, I. Lajoie, Y. Bengio, and P.-A. Manzagol, “Stacked Denoising Autoencoders: Learning Useful Representations in a Deep Network with a Local Denoising Criterion,” Journal of Machine Learning Research (JMLR), vol. 11, pp. 3371–3408, 2010.

[122] Y. Bengio, L. Yao, G. Alain, and P. Vincent, “Generalized Denoising Auto-Encoders as Generative Models,” in C. J. C. Burges and L. Bottou and M. Welling and Z. Ghahramani and K. Q. Weinberger (Eds.). Advances in Neural Information Processing 26 (Neural Information Processing Systems - NIPS), Curran Associates, Inc.,NY, pp. 899–907, 2013.

[123] A. Makhzani and B. J. Frey, “A Winner-Take-All Method for Training Sparse Convolutional Autoencoders,” The Computing Research Repository (CoRR), abs/1409.2752, 2014. [Online]. Available: http://arxiv.org/abs/1409.2752. [Accessed: 12-May-2019].

[124] K. Arulkumaran, A. Creswell, and A. A. Bharath, “Improving Sampling from Generative Autoencoders with Markov Chains,” The Computing Research Repository (CoRR), abs/1610.09296, 2016. [Online]. Available: http://arxiv.org/abs/1610.09296. [Accessed: 12-May-2019].

[125] G. E. Hinton, “Training Products of Experts by Minimizing Contrastive Divergence,” Neural Computation, vol. 14, no. 8, pp. 1771–1800, 2002.

[126] A. Goyal, N. R. Ke, S. Ganguli, and Y. Bengio, “Variational Walkback: Learning a Transition Operator as a Stochastic Recurrent Net,” in I. Guyon and U. V. Luxburg and S. Bengio and H. Wallach and R. Fergus and S. Vishwanathan and R. Garnett (Eds.). Advances in Neural Information Processing 30 (Neural Information Processing Systems - NIPS), Curran Associates, Inc., NY, pp. 4392–4402, 2017.

[127] J. Rudy and G. W. Taylor, “Generative Class-conditional Autoencoders,” International Conference on Learning Representations (ICLR), abs/1412.7009, 2015. [Online]. Available: http://arxiv.org/abs/1412.7009. [Accessed: 12-May-2019].

[128] R. Memisevic, “Gradient-based learning of higher-order image features,” Proceedings of International Conference on Computer Vision (ICCV), Barcelona, Spain, pp. 1591–1598, 2011.

[129] L. Theis, A. van den Oord, and M. Bethge, “A note on the evaluation of generative models,” International Conference on Learning Representations (ICLR), abs/1511.01844, 2016. [Online]. Available: http://arxiv.org/abs/1511.01844. [Accessed: 12-May-2019].

[130] O. Breuleux, Y. Bengio, and P. Vincent, “Quickly Generating Representative Samples from an Rbm-derived Process,” Neural Computation, vol. 23, no. 8, pp. 2058–2073, Aug. 2011.

[131] Y. Bengio, E. Laufer, G. Alain, and J. Yosinski, “Deep Generative Stochastic Networks Trainable by Backprop,” Proceedings of International Conference on Machine Learning (ICML), Beijing, China, pp. 226–234, 2014.

[132] Y. Bengio, G. Mesnil, Y. N. Dauphin, and S. Rifai, “Better Mixing via Deep Representations,” Proceedings of International Conference on Machine Learning (ICML), Atlanta, GA, pp. 552–560, 2013.

[133] S. Rifai, Y. Bengio, Y. Dauphin, and P. Vincent, “A Generative Process for Sampling Contractive Auto-Encoders,” Proceedings of the 29th International Conference on Machine Learning (ICML) 2012. [Online]. Available: https://icml.cc/Conferences/2012/papers/910.pdf. [Accessed: 12-May-2019].

[134] D. Kingma and J. Ba, “Adam: a method for stochastic optimization,” ArXiv e-prints, 2014. [Online]. Available: http://arxiv.org/abs/1412.6980. [Accessed: 12-May-2019].

[135] G. O. Roberts and J. S. Rosenthal, “Optimal scaling of discrete approximations to Langevin diffusions.,” Journal of the Royal Statistical Society Series B: Statistical Methodology, vol. 60, no. 1, pp. 255–268, 1998.

[136] M. Girolami and B. Calderhead, “Riemann manifold Langevin and Hamiltonian Monte Carlo methods,” Royal Statistical Society Series B, Statistical Methodology, vol. 73, no. 2, pp. 123–214, 2011.

[137] Y. Bengio, E. Laufer, G. Alain, and J. Yosinski, “Deep Generative Stochastic Networks Trainable by Backprop,” Proceedings of International Conference on Machine Learning (ICML), Beijing, China, vol. 32, pp. 226–234, 2014.

[138] A. Dosovitskiy and T. Brox, “Generating Images with Perceptual Similarity Metrics based on Deep Networks,” in D. D. Lee and M. Sugiyama and U. V. Luxburg and I. Guyon and R. Garnett (Eds.). Advances in Neural Information Processing 29 (Neural Information Processing Systems - NIPS), Curran Associates, Inc., NY, pp. 658–666, 2016.

[139] A. M. Nguyen, A. Dosovitskiy, J. Yosinski, T. Brox, and J. Clune, “Synthesizing the preferred inputs for neurons in neural networks via deep generator networks,” in D. D. Lee and M. Sugiyama and U. V. Luxburg and I. Guyon and R. Garnett (Eds.). Advances in Neural Information Processing 29 (Neural Information Processing Systems - NIPS), Curran Associates, Inc., NY, pp. 3387–3395, 2016.

[140] J. Yosinski, J. Clune, A. M. Nguyen, T. J. Fuchs, and H. Lipson, “Understanding Neural Networks Through Deep Visualization,” The Computing Research Repository (CoRR), abs/1506.06579, 2015. [Online]. Available: http://arxiv.org/abs/1506.06579. [Accessed: 12-May-2019].

[141] J. Zhao, M. Mathieu, and Y. LeCun, “Energy-based Generative Adversarial Network,” The Computing Research Repository (CoRR), abs/1609.03126, 2016. [Online]. Available: https://arxiv.org/abs/1609.03126. [Accessed: 12-May-2019].

[142] L. Cai, H. Gao, and S. Ji, “Multi-Stage Variational Auto-Encoders for Coarse-to-Fine Image Generation,” Proceedings of International Conference on Data Mining (ICDM), Alberta, Canada, pp. 630–638, 2019.

[143] A. Radford, L. Metz, and S. Chintala, “Unsupervised Representation Learning with Deep Convolutional Generative Adversarial Networks,” Proceedings of International Conference on Learning Representations (ICLR), 2016. [Online]. Available: http://arxiv.org/abs/1511.06434. [Accessed: 12-May-2019].

[144] "Tensorflow, an end-to-end open source Machine Learning Framework", 2019. [Online]. Available: https://www.tensorflow.org/ . [Accessed: 12-May-2019].

[145] T. Kim, M. Cha, H. Kim, J. K. Lee, and J. Kim, "Learning to Discover Cross-Domain Relations with Generative Adversarial Networks," Proceedings of International Conference on Machine Learning 34 (ICML), Beijing, China, vol. 11, pp. 323–336, 2017.

[146] S. Benaim and L. Wolf, "One-sided Unsupervised Domain Mapping," in I.Guyon, U. v. Luxburg, S. Bengio, H. M. Wallach, R. Fergus, S. V. N. Vishwanathan, R. Garnett (Eds.). International Conference on Neural Information Processing Systems 31 (Neural Information Processing Systems - NIPS). Curran Associates, Inc., NY, pp. 752–762, 2017.

[147] S. Benaim and L. Wolf, "One-shot Unsupervised Cross Domain Translation," in I.Guyon, U. v. Luxburg, S. Bengio, H. M. Wallach, R. Fergus, S. V. N. Vishwanathan and R. Garnett (Eds.). International Conference on Neural Information Processing Systems 32 (Neural Information Processing Systems - NIPS). Curran Associates, Inc., NY, pp. 2108–2118, 2018.

[148] M.-Y. Liu and O. Tuzel, "Coupled Generative Adversarial Networks," in D. D. Lee, M. Sugiyama, U. v. Luxburg, I. Guyon and R. Garnett (Eds.) Advances in Neural Information Processing Systems 29 (Neural Information Processing Systems - NIPS). Curran Associates, Inc., NY, pp. 469–477, 2016.

[149] P. Isola, J.-Y. Zhu, T. Zhou, and A. Efros, "Image-to-Image Translation with Conditional Adversarial Networks," Proceedings of IEEE Conference on Computer Vision and Pattern Recognition (CVPR), Honolulu, HI, pp. 5967–5976, 2017.

[150] O. Ronneberger, P. Fischer, and T. Brox, "U-Net: Convolutional Networks for Biomedical Image Segmentation.," in N. Navab, J. Hornegger, W. M. Wells and A. F. Frangi (Eds.). International Conference of Medical Image Computing and Computer-Assisted Intervention 18 (MICCAI), pp. 234–241, 2015.

[151] J. Y. Zhu, T. Park, P. Isola, and A. A. Efros, "Unpaired Image-to-Image Translation using Cycle-Consistent Adversarial Networks," Proceedings of IEEE International Conference on Computer Vision (ICCV), Venice, Italy, pp. 2242—2251, 2017.

[152] Y. Taigman, A. Polyak, and L. Wolf, "Unsupervised Cross-Domain Image Generation," The Computing Research Repository (CoRR), abs/1611.02200, 2016. [Online]. Available: http://arxiv.org/abs/1611.02200. [Accessed: 12-May-2019].

[153] Y. LeCun, C. Corinna, and C. Burges, "MNIST Handwritten Digit Database," 1998. [Online]. Available: http://yann.lecun.com/exdb/mnist. [Accessed: 12-May-2019].

[154] Y. Netzer, T. Wang, A. Coates, A. Bissacco, B. Wu, and A. Y. Ng, "Reading Digits in Natural Images with Unsupervised Feature Learning," in NIPS Workshop on Deep Learning and Unsupervised Feature Learning, 2011.

[155] A. Odena, “Semi-Supervised Learning with Generative Adversarial Networks,” The Computing Research Repository (CoRR), abs/1606.01583, 2016. [Online]. Available: http://arxiv.org/abs/1606.01583. [Accessed: 12-May-2019].

[156] J. T. Springenberg, “Unsupervised and Semi-supervised Learning with Categorical Generative Adversarial Networks,” International Conference on Learning Representations 4 (ICLR), San Juan, Puerto Rico, 2016. [Online]. Available: https://arxiv.org/abs/1511.06390. [Accessed: 12-May-2019].

[157] M. Mathieu, C. Couprie, and Y. LeCun, “Deep multi-scale video prediction beyond mean square error,” Track Proceedings of International Conference on Learning Representations 4 (ICLR), San Juan, Puerto Rico, 2016. [Online]. Available: http://arxiv.org/abs/1511.05440. [Accessed: 12-May-2019].

[158] G. A. Carpenter and N. Markuzon, “ARTMAP-IC and medical diagnosis: Instance counting and inconsistent cases,” Neural Networks, vol. 11, no. 2, pp. 323–336, 1998.

[159] G. A. Carpenter and A.-H. Tan, “Rule Extraction: From Neural Architecture to Symbolic Representation,” Connection Science, vol. 7, no. 1, pp. 3–27, 1995.

[160] D. Deng and N. Kasabov, “On-line pattern analysis by evolving self-organizing maps,” Neurocomputing, vol. 51, pp. 87–103, 2003.

[161] J. W. Smith, J.E. Everhart, W.C. Dickson, W.C. Knowler, and R.S. Johannes, “Using the ADAP Learning Algorithm to Forecast the Onset of Diabetes Mellitus,” Proceedings of the Annual Symposium on Computer Applications in Medical Care, Washington, D.C., pp. 261-265, 1988.

[162] F. Provost, “Machine learning from imbalanced data sets 101,”Workshop of Association for the Advancement of Artificial Intelligence, 2000. [Online]. Available: https://pdfs.semanticscholar.org/c9d6/0858f1cbe6b7eb36473b7d37ff4b73c31af8.pdf, [Accessed: 12-May-2019].

[163] G. Weiss and F. Provost, “The effect of class distribution on classifier learning,”, Technical report ML-TR-43, Department of Computer Science, Rutgers University, 2001. [Online]. Available: https://rucore.libraries.rutgers.edu/rutgers-lib/59623/. [Accessed: 12-May-2019].

[164] N. Japkowicz and S. Stephen, “The class imbalance problem: A systematic study,” Intelligent Data Analysis, vol. 6, no. 5, pp. 429–449, 2002.

[165] C. Drummond and R. Holte, “Class Imbalance, and Cost Sensitivity: Why Under-Sampling beats Over Sampling,” Proceedings of the International Conference on Machine Learning 20 (ICML), Workshop on Learning from Imbalanced Datasets, Washington, D.C., 2003.

[166] N. Chawla, "C4.5 and Imbalanced datasets: Investigating the effect of sampling method, probabilistic estimate and decision tree structure," Proceedings of the International Conference on Machine Learning (ICML) Workshop on Learning from Imbalanced Datasets, Washington, D.C., 2003.

[167] N. Japkowicz, "Class Imbalances: Are We Focusing on the Right Issue?,", Proceedings of the International Conference on Machine Learning (ICML), Workshop on Learning from Imbalanced Datasets, Washington, D.C., 2003.

[168] N. Japkowicz, "The class imbalance problem: Significance and strategies," Proceedings of the International Conference on Artificial Intelligence (IC-AI), Las Vegas, Nevada, pp. 111-117, 2000.

[169] G. E. A. P. A. Batista, R. C. Prati, and M. C. Monard, "A study of the behavior of several methods for balancing machine learning training data," Association for Computing Machinery's Special Interest Group on Knowledge Discovery and Data Mining Explorations (SIGKDD), vol. 6, no. 1, pp. 20–29, 2004.

[170] J. Laurikkala, "Improving Identification of Difficult Small Classes by Balancing Class Distribution," in S. Quaglini, P. Barahona and S. Andreassen (Eds.). Proceedings of Conference on Artificial Intelligence in Medicine in Europe 8 (AIME), Springer, Berlin, vol. 2101, pp. 63–66, 2001.

[171] D. L. Wilson, "Asymptotic Properties of Nearest Neighbor Rules Using Edited Data," IEEE Transactions on Systems, Man and Cybernetics, vol. 2, no. 3, pp. 408–421, 1972.

[172] I. Tomek, "Two Modifications of CNN," IEEE Transactions on Systems, Man and Cybernetics, vol. 7(2), pp. 679–772, 1976.

[173] M. Kubat, R. C. Holte, and S. Matwin, "Learning When Negative Examples Abound," in M. v. Someren and G. Widmer (Eds.). Proceedings of European Conference on Machine Learning 9 (ECML), Lecture Notes in Computer Science, Springer, vol. 1224, pp. 146–153, 1997.

[174] P. E. Hart, "The condensed nearest neighbor rule," IEEE Transactions on Information Theory, vol. 14, no. 3, pp. 515–516, 1968.

[175] G. E. A. P. A. Batista, A. L. C. Bazzan, and M. C. Monard, "Balancing Training Data for Automated Annotation of Keywords: a Case Study," Proceedings of Brazilian Workshop on Bioinformatics 2, Brazil, pp. 10–18, 2003.

[176] R. Raina, A. Battle, H. Lee, B. Packer, and A. Y. Ng, "Self-taught learning: transfer learning from unlabeled data," in Z. Ghahramani (Ed.). Proceedings of International Conference on Machine Learning 24 (ICML), ACM, NY, vol. 227, pp. 759–766, 2007.

[177] R. Rahman and F. Afroz, "Comparison of Various Classification Techniques Using Different Data Mining Tools for Diabetes Diagnosis," Journal of Software Engineering and Applications, Vol. 6 No. 3, pp. 85-97, 2013.

[178] K. Pearson, “On Lines and Planes of Closest Fit to Systems of Points in Space,” Philosophical Magazine, vol. 2, no. 6, pp. 559–572, 1901.

[179] Y. Taigman, M. Yang, M. Ranzato, and L. Wolf, “DeepFace: Closing the gap to human-level performance in face verification,”, Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR), IEEE Computer Society, Columbus, OH, pp. 1701–1708, 2014.

[180] F. Schroff, D. Kalenichenko, and J. Philbin, “FaceNet: A unified embedding for face recognition and clustering,” in Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR), Boston, MA, pp. 815–823, 2015.

[181] J. Redmon, S. Divvala, R. Girshick, and A. Farhadi, “You Only Look Once: Unified, Real-Time Object Detection,” in Proceedings of IEEE Conference on Computer Vision and Pattern Recognition (CVPR), Las Vegas, NV, pp. 779–788, 2016.

[182] A. Dosovitskiy and T. Brox, “Generating Images with Perceptual Similarity Metrics based on Deep Networks,” in D. D. Lee, M. Sugiyama, U. v. Luxburg, I. Guyon and R. Garnett (Eds.), Proceedings of Neural Information Processing Systems 29 (Neural Information Processing Systems - NIPS), Barcelona, Spain, pp. 658–666, 2016.

[183] H. Larochelle and I. Murray, “The Neural Autoregressive Distribution Estimator,” in G. J. Gordon, D. B. Dunson and M. Dud (Eds.), Proceedings of International Conference on Artificial Intelligence and Statistics 14 (AISTATS), Fort Lauderdale, USA, vol. 15, pp. 29–37, 2011.

[184] K. Gregor, I. Danihelka, A. Graves, D. J. Rezende, and D. Wierstra, “DRAW: A Recurrent Neural Network For Image Generation,” in F. R. Bach and D. M. Blei (Eds.), Proceedings of International Conference on Machine Learning 32 (ICML), Lille, France, pp. 1462–1471, 2015.

[185] A. Nguyen, J. Clune, Y. Bengio, A. Dosovitskiy, and J. Yosinski, “Plug & Play Generative Networks: Conditional Iterative Generation of Images in Latent Space,” in in Proceedings of Conference on Computer Vision and Pattern Recognition (CVPR), Honolulu, HI, pp. 3510–3520, 2017.

[186] D. Erhan, A. Courville, and P. Vincent, “Why Does Unsupervised Pre-training Help Deep Learning ?,” Journal of Machine Learning Research (JMLR), vol. 11, pp. 625–660, 2010.

[187] Y. Bengio, P. Lamblin, D. Popovici, and H. Larochelle, “Greedy Layer-Wise Training of Deep Networks.,” in B. Scholkopf, J. C. Platt and T. Hofmann (Eds.). Proceedings of Neural Information Processing Systems 19 (Neural Information Processing Systems - NIPS), MIT Press, Cambridge, MA, pp. 153–160, 2006.

[188] T. Miyato, S. Maeda, M. Koyama, K. Nakae, and S. Ishii, “Distributional Smoothing by Virtual Adversarial Examples,” Conference Track Proceedings of International Conference on Learning Representations 4 (ICLR), 2016. [Online]. Available: http://arxiv.org/abs/1507.00677. [Accessed: 12-May-2019].

Appendix A

Publications during PhD work

[1] S. Karatsiolis and C. N. Schizas, “Conditional Denoising Autoencoder,” IEEE Transactions on Neural Networks and Learning Systems (TNNLS), 2019. [Accepted for publication on 2/11/2019, publication proofing submitted on 24/11/2019].

[2] S. Karatsiolis, C. N. Schizas, and N. Petkov, “Modular Domain-to-Domain Translation Network,” Neural Computing and Applications (NCAA), pp. 1–13, 2019.

[3] A. Kamilaris, C. van den Brink, and S. Karatsiolis, “Training Deep Learning Models via Synthetic Data: Application in Unmanned Aerial Vehicles,”, in M. Vento, G. Percannella, S. Colantonio, D. Giorgi, B. J. Matuszewski, H. Kerdegari and M. Razaak (Eds.). Proceedings of Computer Analysis of Images and Patterns 18 (CAIP), Workshop on Deep learning based computer vision for UAV, Springer, vol. 1089, pp. 81–90, 2019.

[4] S. Karatsiolis, C. N. Schizas, and N. Petkov, “Modular Domain-to-Domain Translation Network,” in V. Kurkova, Y. Manolopoulos, B. Hammer, L. S. Iliadis and I. Maglogiannis (Eds.).Proceedings of Artificial Neural Networks and Machine Learning 27 (ICANN), Springer, vol. 11141, pp. 425–435, 2018.

[5] S. Karatsiolis, C. N. Schizas, "Variable Target Values Neural Network for Dealing with Extremely Imbalanced Datasets,", in E. Kyriacou, S. Christofides, C. Pattichis (Eds.). 14th Mediterranean Conference on Medical and Biological Engineering and Computing. IFMBE Proceedings, Springer, vol 57. pp.525-528, 2016.

[6] S. Karatsiolis, C. N. Schizas, “Feature Mapping Through Maximization of the Atomic Interclass Distances,”, in A. Gammerman, V. Vovk, H. Papadopoulos (Eds.), Statistical Learning and Data Sciences (SLDS), Lecture Notes in Computer Science, vol 9047, pp 75-85, Springer, NY, 2015.

[7] S. Karatsiolis and C. N. Schizas, “Implementing a software cache for genetic programming algorithms for reducing execution time,”, Proceedings of the International Conference on Evolutionary Computation Theory and Applications (ECTA), Rome, Italy, pp. 259–265, 2014.

[8] S. Karatsiolis and C. N. Schizas, “Region based Support Vector Machine algorithm for medical diagnosis on Pima Indian Diabetes dataset,” Proceedings of the IEEE International Conference on Bioinformatics and Bioengineering 12 (BIBE), IEEE Computer Society, Washington, DC, pp. 139–144, 2012.