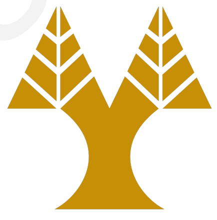
Thesis Dissertation

SPAM EMAIL CLASSIFICATION USING CONVOLUTIONAL NEURAL NETWORKS WITH HESSIAN-FREE OPTIMISATION

Kypros Ioannou

UNIVERSITY OF CYPRUS



COMPUTER SCIENCE DEPARTMENT

December 2021

Kypros Ioannou

UNIVERSITY OF CYPRUS
COMPUTER SCIENCE DEPARTMENT

Spam Email Classification using Convolutional Neural Networks with Hessian-Free Optimisation

Kypros Ioannou

Kypros Ioannou

Supervisor
Dr. Chris Christodoulou

Thesis submitted in partial fulfilment of the requirements for the award of degree of Bachelor in Computer Science at University of Cyprus

December 2021

Acknowledgments

I would like to thank my advisor, Dr Chris Christodoulou, for his continuous support of my thesis and over the courses I have had with him, as well as his patience, insightful feedback, and corrections. His guidance played a significant role in the completion of this dissertation. Moreover, I would like to extend my gratitude to Dr Michalis Agathokleous, who was always there to assist and share his knowledge.

Lastly, I am grateful to my family, for not only supporting financially, but also because they believed in me and gave me the chance to accomplish my dreams. Special thanks to my friend Eftychios Kaimakkamis for all the support during my thesis and in general.

Kypros Ioannou

Contents

1 Introduction **1**
1.1 Emails Attack Identification Problem 1
1.2 Email Threats Trends 2
1.2.1 Email Attacks with Covid-19 2
1.2.2 World's Most Dangerous Malware (Emotet) 3

2 Background and Literature Review **5**
2.1 E-mail Background 5
2.1.1 Email Threats 5
2.1.2 Structure of an E-mail 7
2.1.3 Email's Related Work 8
2.2 Origin of Artificial Neural Networks 10
2.3 Types of Artificial Neural Networks and Optimizers 11
2.3.1 McCulloch–Pitts (MCP) 11
2.3.2 Multi-Layer Perceptron (MLP) 14
2.3.3 Gradient Descent (GD) 17
2.3.4 Backpropagation Algorithm (BP) 18
2.3.5 Recurrent Neural Networks (RNN) 18
2.3.6 Convolutional Neural Networks (CNN) 20
2.3.7 Line Search 22
2.3.8 Conjugate Gradient (CG) 23
2.3.9 Newton's Method 23
2.3.10 Hessian Free Optimization (HFO) 24
2.3.11 Word Embedding 25
2.3.12 Subsampled Hessian Newton (SHN) Method 30
2.3.13 CNN With Hessian Free Optimisation Related Work 31
2.3.14 Ensemble Methods 32

3 Design and Implementation **34**
3.1 Data 34

Kypros Ioannou

3.2 Data Preprocessing . . . 36
3.3 Word Embedding on Email Data . . . 38
3.4 Network Implementation . . . 39
3.5 Metrics . . . 40
3.5.1 Training/Testing Set and Cross Validation . . . 41
3.5.2 Confusion Matrix . . . 43
3.5.3 Rates computed by Confusion Matrix . . . 44

4 Results and Discussion 46
4.1 Fine-tuning of Hyperparameters . . . 46
4.2 Cross-Validation Results with First Dataset . . . 49
4.3 Cross-Validation Results with Second Dataset . . . 52
4.4 Cross-Validation Results with Third Dataset . . . 53
4.5 Cross-Validation Results with Fourth Dataset . . . 55
4.6 Cross-Validation Results with Fifth Dataset . . . 57
4.7 Cross-Validation Results with Sixth Dataset . . . 59
4.8 Testing best model with the other 5 Datasets . . . 60
4.9 Comparison Between Gradient Descent and Hessian Free Optimisation 61
4.10 Comparison Between Original Dataset Work and CNN With Hessian Free Optimisation . . . 63

5 Conclusion and Future Work 66
5.1 Conclusion . . . 66
5.2 Future Work . . . 68

Appendix A - Source Code A-1
CNN With Hessian-Free Optimization Source Code . . . A-1
Create Folds for Cross-Validation Source Code . . . A-34
CNN with Gradient Descent Source Code . . . A-39

Appendix B - Confusion Matrices B-1
Confusion Matrices for the Whole Dataset . . . B-1
Remaining Confusion Matrices from Cross-Validation . . . B-3
First Dataset . . . B-3
Second Dataset . . . B-5
Third Dataset . . . B-8
Fourth Dataset . . . B-11
Fifth Dataset . . . B-13
Sixth Dataset . . . B-15

Kypros Ioannou

Confusion Matrices from when we use CNN with Gradient Descent B-18

Kypros Ioannou

List of Figures

1.1 The industries targeted by phishing of 2018 phishing trends & intelligence report [20]. 2

1.2 Covid-19, 2020 themed attacks vs all malwares [6]. 3

2.1 Format of an e-mail. We have the header and body of the e-mail. Both parts have multiple subparts. [3] 7

2.2 Fang et al. retrieve the emails and separate the email in two parts, Header and Body. They create the character and word vector and then proceed with the char/word level of header and body that will put in their network for training and testing. They use a Recurrent Convolutional Neural Network with attention to select the information that is more critical. The final result is a model that can do binary classification (Phishing/Legitimate Emails) [20]. 9

2.3 Biological representation of a neuron. 10

2.4 In this version, McCulloch and Pitts model has three inputs and is using the Threshold Function to either fire 1 or 0. [12] 11

2.5 Threshold Function 12

2.6 And and OR Gate 12

2.7 Perceptron Learning Algorithm 13

2.8 Xor Gate 14

2.9 Multilayer Perceptron 15

2.10 Types of decision regions that can be formed by single and Multilayer Perceptron with one or two hidden layers and two inputs [30]. 17

2.11 Backpropagation Algorithm [44] 18

2.12 Jordan Architecture with two 3 inputs and three States Units that propagate the previous output. [24] 19

2.13 Elman Architecture with an internal loop between input and hidden layer. [19] 19

Kypros Ioannou

2.14 A Convolutional Neural Network where we have the input introduced. We then extract features using convolution layer and we reduce the sizes by doing subsampling. Lastly we use a full connected neural network for the classification [28]. 21
2.15 Max and Average Pooling. 22
2.16 Gradient Descent (left) vs Conjugate Gradient (right) on a 2D problem. 23
2.17 The CBOW architecture predicts the current word based on the context, and the Skip-gram predicts surrounding words given the current word [35] 27
2.18 In CBOW, given the words (the quick brown box, over the lazy log), we would want to predict jump. In Skipgram just the opposite given the word jump, we would want to predict (the quick brown box, over the lazy log) [11]. 28
2.19 Architectures for CBOW (Left) and Skip-gram (Right) [11]. 28

3.1 An email with non-Lating characters. 36
3.2 Converting each word into a vector with real values. 38
3.3 Similar representation between similar words. 39
3.4 Cross Validation. We split our dataset into 5 pieces and each time we use a different piece in order to validate our model accuracy. . . . 42
3.5 Confusion Matrix. 44

4.1 Our Convolutional Neural Networks Architecture. 49
4.2 Dataset 1 Fold 5 Train Confusion Matrix.(Best Model) 50
4.3 Dataset 1 Fold 5 Valid Confusion Matrix. (Best Model) 51
4.4 Dataset 1 Fold 2 Train Confusion Matrix. (Worst Model) 51
4.5 Dataset 1 Fold 2 Valid Confusion Matrix. (Worst Model) 52
4.6 Dataset 2 Fold 2 Train Confusion Matrix(Worst Model) where we have 0 False Positives and 34 False Negatives 53
4.7 Dataset 2 Fold 2 Validation Confusion Matrix(Worst Model) have we scored 0 False Positives and 12 False Negatives. 53
4.8 Dataset 3 Fold 3 Train Confusion Matrix(Worst Model) where we have 3 False Positives and 4 False Negatives. 54
4.9 Dataset 3 Fold 3 Validation Confusion Matrix(Worst Model) where we have 1 False Positives and 3 False Negatives. 55
4.10 Dataset 4 Fold 4 Train Confusion Matrix(Worst Model) where we have 20 False Positives and 16 False Negatives. 56
4.11 Dataset 4 Fold 4 Validation Confusion Matrix(Worst Model) where we have 8 False Positives and 5 False Negatives. 56

Kypros Ioannou

4.12 Dataset 4 Fold 3 Train Confusion Matrix(Best Model) where we have 23 False Positives and 19 False Negatives. 57
4.13 Dataset 4 Fold 3 Validation Confusion Matrix(Best Model) where we have 0 False Positives and 2 False Negatives. 57
4.14 Dataset 5 Fold 3 Validation Confusion Matrix(Worst Model) where we have 0 False Positives and 62 False Negatives. 58
4.15 Dataset 5 Fold 3 Train Confusion Matrix(Worst Model) where we have 0 False Positives and 14 False Negatives. 58
4.16 Dataset 6 Fold 2 Validation Confusion Matrix(Worst Model) where we have 11 False Positives and 38 False Negatives. 59
4.17 Dataset 6 Fold 2 Validation Confusion Matrix(Worst Model) where we have 2 False Positives and 12 False Negatives. 60
4.18 CNNs with Hessian Free Optimization converges 2.5 times faster than the CNNs with Gradient Descent. 63

5.1 Confusion Matrix for the first dataset. B-1
5.2 Confusion Matrix for the second dataset. B-1
5.3 Confusion Matrix for the third dataset. B-2
5.4 Confusion Matrix for the fourth dataset. B-2
5.5 Confusion Matrix for the fifth dataset. B-2
5.6 Confusion Matrix for the sixth dataset. B-3
5.7 Dataset 1 Fold 1 Train Confusion Matrix B-3
5.8 Dataset 1 Fold 1 Validation Confusion Matrix B-4
5.9 Dataset 1 Fold 3 Train Confusion Matrix B-4
5.10 Dataset 1 Fold 3 Validation Confusion Matrix B-4
5.11 Dataset 1 Fold 4 Train Confusion Matrix B-5
5.12 Dataset 1 Fold 4 Validation Confusion Matrix B-5
5.13 Dataset 2 Fold 1 Train Confusion Matrix B-5
5.14 Dataset 2 Fold 1 Validation Confusion Matrix B-6
5.15 Dataset 2 Fold 3 Train Confusion Matrix B-6
5.16 Dataset 2 Fold 3 Validation Confusion Matrix B-6
5.17 Dataset 2 Fold 4 Train Confusion Matrix B-7
5.18 Dataset 2 Fold 4 Validation Confusion Matrix B-7
5.19 Dataset 2 Fold 5 Train Confusion Matrix B-7
5.20 Dataset 2 Fold 5 Validation Confusion Matrix B-8
5.21 Dataset 3 Fold 1 Train Confusion Matrix B-8
5.22 Dataset 3 Fold 1 Validation Confusion Matrix B-8
5.23 Dataset 3 Fold 2 Train Confusion Matrix B-9

Kypros Ioannou

5.24 Dataset 3 Fold 2 Validation Confusion Matrix B-9
5.25 Dataset 3 Fold 4 Train Confusion Matrix B-9
5.26 Dataset 3 Fold 4 Validation Confusion Matrix B-10
5.27 Dataset 3 Fold 5 Train Confusion Matrix B-10
5.28 Dataset 3 Fold 5 Validation Confusion Matrix B-10
5.29 Dataset 4 Fold 1 Train Confusion Matrix B-11
5.30 Dataset 4 Fold 1 Validation Confusion Matrix B-11
5.31 Dataset 4 Fold 2 Train Confusion Matrix B-11
5.32 Dataset 4 Fold 2 Validation Confusion Matrix B-12
5.33 Dataset 4 Fold 5 Train Confusion Matrix B-12
5.34 Dataset 4 Fold 5 Validation Confusion Matrix B-12
5.35 Dataset 5 Fold 1 Train Confusion Matrix B-13
5.36 Dataset 5 Fold 1 Validation Confusion Matrix B-13
5.37 Dataset 5 Fold 2 Train Confusion Matrix B-13
5.38 Dataset 5 Fold 2 Validation Confusion Matrix B-14
5.39 Dataset 5 Fold 4 Train Confusion Matrix B-14
5.40 Dataset 5 Fold 4 Validation Confusion Matrix B-14
5.41 Dataset 5 Fold 5 Train Confusion Matrix B-15
5.42 Dataset 5 Fold 5 Validation Confusion Matrix B-15
5.43 Dataset 6 Fold 1 Train Confusion Matrix B-15
5.44 Dataset 6 Fold 1 Validation Confusion Matrix B-16
5.45 Dataset 6 Fold 3 Train Confusion Matrix B-16
5.46 Dataset 6 Fold 3 Validation Confusion Matrix B-16
5.47 Dataset 6 Fold 4 Train Confusion Matrix B-17
5.48 Dataset 6 Fold 4 Validation Confusion Matrix B-17
5.49 Dataset 6 Fold 5 Train Confusion Matrix B-17
5.50 Dataset 6 Fold 5 Validation Confusion Matrix B-18
5.51 Confusion Matrix from when we train our model for the first dataset. B-18
5.52 Confusion Matrix from when we test our model for the first dataset. . B-19
5.53 Confusion Matrix from when we train our model for the second dataset. B-19
5.54 Confusion Matrix from when we test our model for the second dataset. B-20
5.55 Confusion Matrix from when we train our model for the third dataset. B-20
5.56 Confusion Matrix from when we test our model for the third dataset. B-21
5.57 Confusion Matrix from when we train our model for the fourth dataset. B-21
5.58 Confusion Matrix from when we test our model for the fourth dataset. B-22
5.59 Confusion Matrix from when we train our model for the fifth dataset. B-22
5.60 Confusion Matrix from when we test our model for the fifth dataset. . B-23
5.61 Confusion Matrix from when we train our model for the sixth dataset. B-23

Kypros Ioannou

5.62 Confusion Matrix from when we test our model for the sixth dataset. B-24

Kypros Ioannou

List of Tables

2.1 Truth table for AND 12
2.2 List of the most popular activation functions. [29] 16
2.3 One hot encoding where each word in its corresponding cell has a value of 1. 26
2.4 Comparison Between Tokenization and do One-Hot Encoding, Word2Vec Embedding and Bert Embedding for first Dataset. 29
2.5 Comparison Between Tokenization and do One-Hot Encoding, Word2Vec Embedding and Bert Embedding for second Dataset. 30

3.1 The table of the six different datasets that we retrieved from the article: [34].Each dataset is used separately and some datasets have common emails. Each dataset has more than 5000 email either as Spam or Ham. 35

4.1 Sixth dataset accuracy when we change the GNsize for fold 4. 47
4.2 Sixth dataset accuracy when we change the C hyperparameter for fold 4. 47
4.3 Sixth dataset accuracy using 3 different number of Convolutional and Pooling Layers. 48
4.4 Sixth dataset accuracy using 3 different number of filters fo the Convolutional Layer. 48
4.5 Cross-Validation for the First Dataset. We measure the accuracy of the model and also Spam and Ham Recall. 50
4.6 Cross-Validation for the Second Dataset. We measure the accuracy of the model and also Spam and Ham Recall. 52
4.7 Cross-Validation for the Third Dataset. We measure the accuracy of the model and also Spam and Ham Recall. 54
4.8 Cross-Validation for the Fourth Dataset. We measure the accuracy of the model and also Spam and Ham Recall. 55

Kypros Ioannou

4.9 Cross-Validation for the Fifth Dataset. We measure the accuracy of the model and also Spam and Ham Recall. 58
4.10 Cross-Validation for the Sixth Dataset. We measure the accuracy of the model and also Spam and Ham Recall. 59
4.11 Average accuracy for all the dataset(Dataset 3 is used to train CNNs/HFO). 60
4.12 Average Accuracy and Spam/Ham Recall from Cross-Validation for CNN/Gradient Descent Model 62
4.13 Average Accuracy and Spam/Ham Recall from Cross-Validation for CNN/Hessian Free Optimization Model 62
4.14 Dataset 1 Spam and Ham Recall from original paper and our model . 63
4.15 Dataset 2 Spam and Ham Recall from original paper and our model . 64
4.16 Dataset 3 Spam and Ham Recall from original paper and our model . 64
4.17 Dataset 4 Spam and Ham Recall from original paper and our model . 65
4.18 Dataset 5 Spam and Ham Recall from original paper and our model . 65
4.19 Dataset 6 Spam and Ham Recall from original paper and our model . 65

5.1 Metrics for all the folds for the first dataset B-18
5.2 Metrics for all the folds for the second dataset B-19
5.3 Metrics for all the folds for the third dataset B-20
5.4 Metrics for all the folds for the fourth dataset B-21
5.5 Metrics for all the folds for the fifth dataset B-22
5.6 Metrics for all the folds for the sixth dataset B-23

Kypros Ioannou

Abstract

In this dissertation we attempt to solve the Email Classification problem with a novel method using a second-order function with a Convolutional Neural Network (CNN). As far as the literature is concerned, currently there is no other method that uses Hessian Free Optimisation with CNN to solve the Email Classification problem.

We use CNN with Hessian Free Optimisation to distinguish between spam emails and ham (legitimate) emails. Word Embedding is applied to the data to convert them to a numerical form that the Neural Network model can understand. The Word Embedding we use is the Wor2Vec, and we achieve very satisfactory results. Furthermore, we use cross-validation to verify the model's good accuracy. We split the data five-fold, and used in total six different datasets.

We compare the model with other authors' works and a classic Convolutional Neural network with Gradient Descent (GD) which we also implement in this dissertation. We measure the efficacy of each model by calculating the Accuracy, and Spam/Ham Recall. The accuracy measurement was used just for the CNN with GD since the aforementioned authors only provided Ham and Spam Recall measurements.

We use the entire dataset for training when we compare this model with other authors' work. We achieve accuracy of 99.199%, and 97.39%, 99.94% for Spam and Ham Recall for the first dataset respectively. For the second dataset, we achieve accuracy of 99.227% and 96.98% and 100%, Spam and Ham Recall. The accuracy was 99.848% in the third dataset, and the Spam, Ham Recalls were 99.59% and 99.94%. The accuracy of the fourth dataset was 99.333%, with 99.58% for Spam and 98.59% for Ham Recall. For the fifth dataset, accuracy was 99.061%, with 98.69% for Spam and a perfect score (100%) for Ham Recall. Finally in the sixth dataset, the accuracy was 98.997%, with 98.93% Spam and 99.19% Ham Recall.

Kypros Ioannou

Lastly, we performed cross-validation and the average validation accuracy for each dataset was: Dataset 1 99.078%, Dataset 2 99.158%, Dataset 3 99.772%, Dataset 4 99.240%, Dataset 5 98.762% and Dataset 6 98.846%. The average Spam and Ham Recall for each dataset was similar to the ones mentioned in the previous paragraph, but we achieved two perfect scores in Ham Recall in the second and the fifth dataset. All other Spam and Ham Recalls from our implementation were between 96.72% and 99.88%. We also applied cross-validation for CNN with Gradient Descent, but the highest accuracy achieved was 76% in Dataset 4, and the lowest was in Dataset 5. The first three datasets have 0% Spam Recall, and the last three datasets have 0% Ham Recall.CNNs with Hessian Free Optimization do not just have better accuracy and ham/spam recall in every dataset, but also the model converges faster than the Gradient Descent model.We measure that with our best model; the HFO model converges 2.5 times faster than the Gradient Descent using the same dataset.

Kypros Ioannou

Chapter 1

Introduction

1.1 Emails Attack Identification Problem

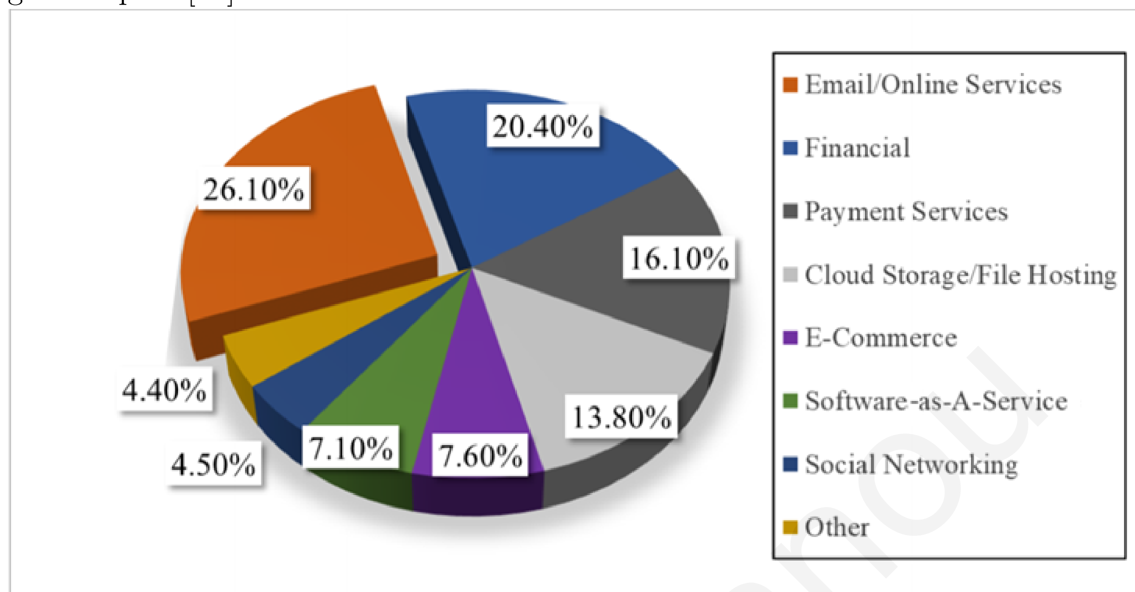
Emails are among the most valuable and usable tools that people use to communicate, and it is a cheap and fast way for many companies to do business. With the current pandemic still ongoing, the importance of effective email communication is perhaps greater than ever.

A malicious email can have a variety of purposes, as it is up to the attacker to decide what they want to get from the target. Those emails may promote a fake product with the intent to sell it to unsuspecting consumers, or a phishing email trying to steal the user's credentials. We expect that in 2021 there will be more cyberattacks that use psychological tricks on a variety of subjects to leverage the emotional fragility of users" [39]. Since the day the first email was sent, 48 years ago, attackers have developed many methods to bypass the email server filters and the standard email user.

In Figure 1.1 , we can see the chart from PhishLabs. In this chart, we have the top phishing targets [20]. The top two targets with the most attacks are the Email Online Service (EOS) and the Financial sector. The EOS sits at the top with more than a fourth of the total cyberattacks.

Kypros Ioannou

Figure 1.1: The industries targeted by phishing of 2018 phishing trends & intelligence report [20].



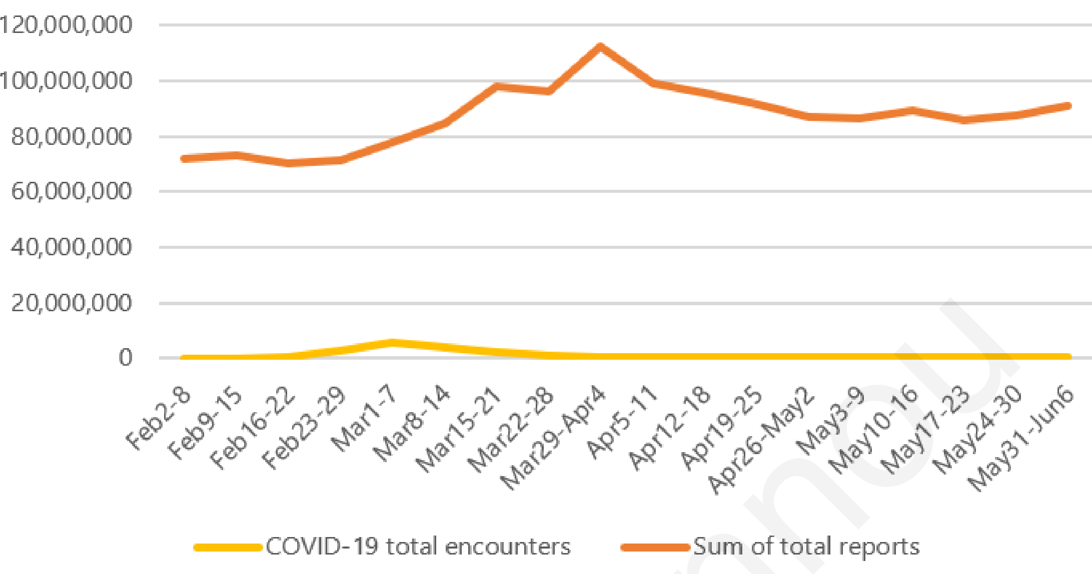
This dissertation aims to prove that a Convolutional Neural Network (CNN) with a second order optimization (Hessian-Free) can have comparable - perhaps even better - results to a more traditional version of CNN that uses the first-order optimizer (Gradient Descent) in a Natural Language processing problem as Email Classification. We will compare those two models in terms of Accuracy, Spam and Ham Recall, and Runtime. Furthermore, we will compare the CNN/HFO model with the authors from where we used the relevant data. Meltis et al. [34] use a variety of Naïve Bayes algorithms to solve the problem that we are trying to do in this dissertation. We wish to understand whether a more complex algorithm like CNN/HFO can achieve even higher results than the already satisfactory results presented in the original paper.

1.2 Email Threats Trends

1.2.1 Email Attacks with Covid-19

Microsoft examined how criminals behave during a current epidemic; they observed how easily they can change their basic concept plan of attacking to something relative to the current situation. In Figure 1.2, we can see how much attacks have increased during the COVID-19 pandemic [6].

Figure 1.2: Covid-19, 2020 themed attacks vs all malwares [6].



Microsoft has observed that COVID-19 themed attacks peaked worldwide in the first two weeks of March in 2020 and after that, the number had decreased. That is because during that period many nations began to take measures to reduce the spread of the virus, and travel restrictions came into effect. We can deduce from Figure 1.2 that attackers can easily change the theme of their emails based on the situation. Such attacks started before March; between February 9 and 15 attackers sent more than 500.000 emails per day. That number dramatically increased in March, reaching 5.5 million emails with a COVID-19 theme per day. After March 7, the number of COVID-19 emails steadily decreased until it reached 500.000 per day by May 31.

1.2.2 World's Most Dangerous Malware (Emotet)

As mentioned by Europol, Emotet is the world's most dangerous malware [9]. Emotet is nothing new, but it came to the forefront once again in July 2020. The attacks harbouring this malware use the Threat hijacking technique. This technique is very reliable since the email is sent from a trusted contact and thus "the context of the existing discussion lowers the targeted recipients' guard" [39]. With that in mind, the unaware victim may fall into the attacker's trap since it comes from a familiar email and the email provider may let this email pass since it is from a trusted user.

The remaining dissertation is split into 4 Chapters, starting from Chapter 2. We

Kypros Ioannou

elaborate in depth about different types of email attacks, and provide some Neural Network background to understand Neural Networks in general and the algorithm we used in this project. Continuing with Chapter 3, we discuss the data we use and all the preprocessing that is necessary to have so that they are in an appropriate form to feed them into the Neural Network. The fourth chapter compares our CNN/HFO model with original paper work [34] and CNN/GD. In the last chapter we go over conclusions drawn from this dissertation and suggest what work can be done in the future.

Kypros Ioannou

Chapter 2

Background and Literature Review

Kypros Ioannou

2.1 E-mail Background

2.1.1 Email Threats

With the current email filters that many big email companies support, like Google or Microsoft, attackers need to find a new way of sending malicious purpose emails. For example, an email may not even arrive in the spam folder because of the sender IP address. A kind new method that attackers use to bypass emails filters is called "image manipulation." Attackers are trying to hide malicious code behind an image, and the reason for that is that emails servers need more time to scan images instead of just text, and because of that, emails server may be scan only the text. Some of the top email attacks are listed below. A single attack may have more than one type of attacks.

- Malware
- Spamming
- Phishing
- Social Engineering

Malware

Malware is a piece of software written to damage a computer system or network intentionally. A variety of different Malware exists, such as Trojan, Worms, Ransomware, Spy-wares [41] and based on the type of the attack and what the attacker

wants to do different types of Malware can be used. Some common type of attacks is volumetric, zero-day Malware and URL attacks. Firstly, the volumetric attack has as a goal to spread to as many computers as possible. Malware tries to convert computers to zombies and create a bot network where all zombie machines propagate the Malware. In the second attack, zero-day, the attacker uses an undercover vulnerability of the system to execute his/shes malicious software. Lastly, with a URL attack, the attackers try to navigate the user to an unsafe web page to download malware.

Spamming

In 2021 most email providers like Google with Gmail will prevent spam mail from reaching the user. They can either move them to the junk folder or prevent the email from even come to the user account. A spam email does not necessarily mean that the sender has a purpose of attacking, but it may be an advertising email that promotes a product. On the other hand, we may have an advertise again about a product, but this time the product does not exist. Those products can be either cheaper than some official or promise things that other products cannot do.

Phishing

Phishing attacks are a subcategory of spam. When an attacker sends a phishing email, this email is for sure an email to do some damage. The damages maybe not be about the software of the user machine but are force to steal information from the user such as bank accounts or emails and passwords to use them for the next attacks or to steal money. Usually, phishing emails like spam emails are propagated to millions of accounts, and they do not have a different structure from one user to another.

We have another type of phishing attack, but instead of targeting large volumes of emails, it focuses on a small group of people. Spear-Phishing attacks targets are, for example, companies or organisations. Instead of sending them a generic email, they construct an email in such a way as to look like a legitimate email. The attackers scatter the internet to find information about their target group in order to create a more realistic/sophisticated email [45].

Social Engineering

Social Engineering techniques are a part of every email attack. The attacker needs to convince the user to either download malicious software or give some personal

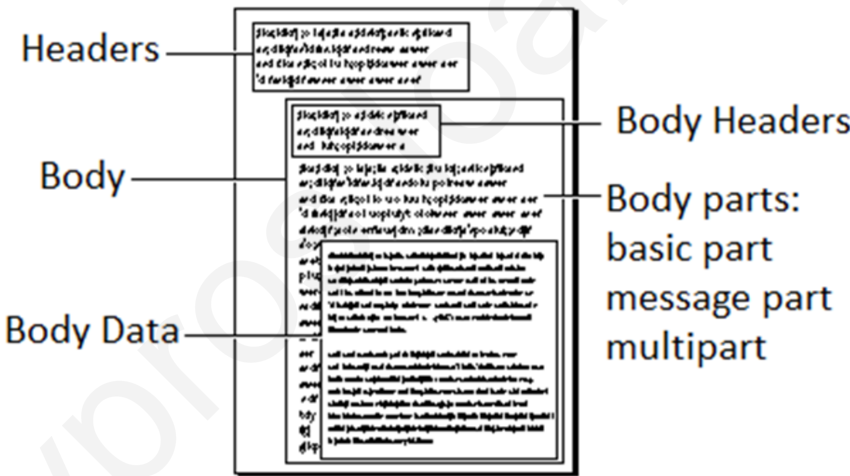
Kypros Ioannou

information.All of the attacks we mentioned above are required to trick the user in a case of Social Engineering. One of the main themes an email may have is the theme of fear. The attacker may send an email to a compromised account or a problem with a bank account to make the victim make a rush move. As we already mentioned above, attackers need to do some research to convince the victim. According to Cybint, “98% of Cybersecurity breaches are caused by human error” [18]. It is clear that even in 2021, humans can be manipulated to give access to attackers.

2.1.2 Structure of an E-mail

In order to visualize and understand in the following chapters how we convert data into a comprehensible form via a machine learning model, we first need to discuss the format of an e-mail. To get a clearer picture of said format, we look at the following figure:

Figure 2.1: Format of an e-mail. We have the header and body of the e-mail. Both parts have multiple subparts. [3]



When we extract an e-mail from an e-mail provider such as Google (Gmail) or Microsoft (Outlook), the e-mail is usually in Multipurpose Internet Mail Extension (MIME) format. The purpose of MIME is to enhance the e-mail message, and some of the extra features are listed below:

- The ability to send rich information through the Internet.
    - Add Audio, Video, Images, and Application Programs to the message.
- The ability to encode and attach binary (non-ASCII) content to messages.
- A framework for multipart mail messages that contain differing body parts.

As we can clearly observe from Figure 2.1, the two main parts of a MIME format are the Header and the Body. A body can have multiple sub-body parts, but they are the same in terms of structure. On the other hand, the header of an e-email includes a list of information, some of which is visible when sending or receiving an e-mail, while some other is not.

The information elements visible to the user during e-mail construction that the sender needs to fill are as follows:

- Received: Contains transit-related information of e-mail servers, IP addresses, dates.
- From: Sender's name; e-mail ID. "Name" e-mail@example.com.
- Recipient's name: e-mail ID. "Name"e-mail@example.com.
- Subject: String identifies the theme of the message placed by the sender.

The last elements we can extract from the header of an e-mail are the Return Path, Message-ID, X-Mailer, and the Content-type.

- Return-path encloses an optional address specification to use if an error is encountered (bounce).
- Message-ID has a unique message identifier that is designed by the mail system.
- X-Mailer can tell us the mail software sender used to construct the e-mail.
  - X-Mailer can be Mozilla Browser.
- Content-type refer as the name suggests, to the type of content.
  - Content-type can be Text/Plain.

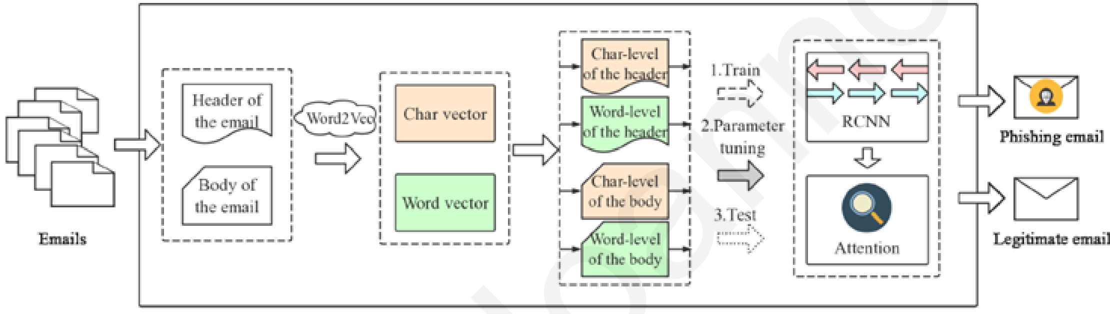
Kypros Ioannou

2.1.3 Email's Related Work

There is already plentiful pre-existing work on email classification [48] [27] [22].A variety of different methods has been used trying to achieve very good accuracy on each model. Vinayakumar et al. [48] use a variety of Deep Learning Methods such as Convolutional Neural Network(CNN), Recurrent Neural Network(RNN), Long-Short-Term Memory and Multiplayer-Perceptron trying to identify an email either as spam or legitimate. Kumar et al. [27] use 7 different classifiers for classifying email as spam or legit. They compare those 7 classifiers in terms of accuracy and the one that performed the best out of these was AdaBoost, an Ensemble Method that we

are going to explain in the next section. The other 6 classifiers were: Random Forest, Decision Trees, Bagging, K-Nearest Neighbor, Support Vector Machines, and Naïve Bayes. Lastly, we also have a very interesting combination of Convolutional Neural Networks and Recurrent Neural Networks [20]. In Figure 2.2, the architecture of Recurrent Convolutional Neural Network is visualized [20].

Figure 2.2: Fang et al. retrieve the emails and separate the email in two parts, Header and Body. They create the character and word vector and then proceed with the char/word level of header and body that will put in their network for training and testing. They use a Recurrent Convolutional Neural Network with attention to select the information that is more critical. The final result is a model that can do binary classification (Phishing/Legitimate Emails) [20].

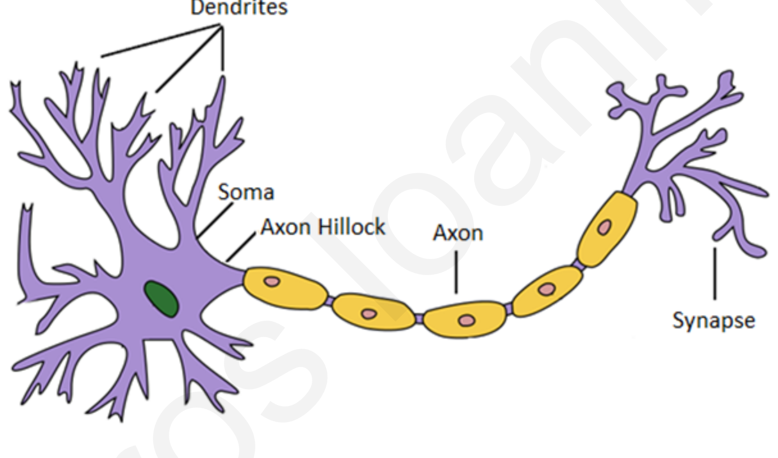


The first part is the header of the email that has all the information about the receiver. The second part is the actual body of the email.Fang et al. [20] retrieve both in word-level and in char-level using Word2Vec, a two-layer neural network "that processes text by"vectorizing" words" [2]. Some other articles we mentioned above [48] [27] focus only on word-level but here the char-level is investigated as well. That is done in order to take writing errors e.g., spelling mistakes and lowercase/uppercase letter writing into account. Such factors may constitute a sender's style of writing [20].After Fang et al. [20] retrieve Char and Word vectors they feed-in them in the RCNN. Instead of using a simple Recurrent Neural Network they [20] used an LTSM, which has been used by other researchers in the past [48]. This LSTM is slightly different because they use a bi-directional LSTM. The choice of LSTM is justified by the fact that an RNN cannot learn such "long-term dependencies" from the emails.Also, Fang et al. [20] mentioned in their paper that the "recurrent structure of RCNN can preserve longer contextual information" and we can have less noise.Lastly, a technique called "attention mechanism" is used for selecting the information that is more critical to achieving the current task, which is the classification of the emails.

2.2 Origin of Artificial Neural Networks

Many ideas in human history emerged by finding inspiration in the nature of known biological processes. This is precisely the case when it comes to creating an artificial neural network by drawing inspiration from the human brain; we mimic human brain behaviour in learning and passing information. However, an Artificial Neural Network does not have as much complexity as a neural network in our brain. A human brain has 10^{11} neurons that are connected to 10^4 other neurons each. Such a complex architecture is impossible to replicate with current means due to the limitation of the hardware. Figure 2.3 shows us the biological neural structure having only the most essential part of a neuron to understand how we transitioned from a biological neuron to an artificial one.

Figure 2.3: Biological representation of a neuron.



Dendrites receive an input signal from another neuron and transmit an electrical stimulus to the soma.The soma is the part of the neuron where all input signals are joined together and pass to the Axon Hillock. If the total strength of the signals exceeds the threshold, then it will fire a signal that will pass on through the axon. Lastly, a synapse is where two neurons are connected and pass information to each other.

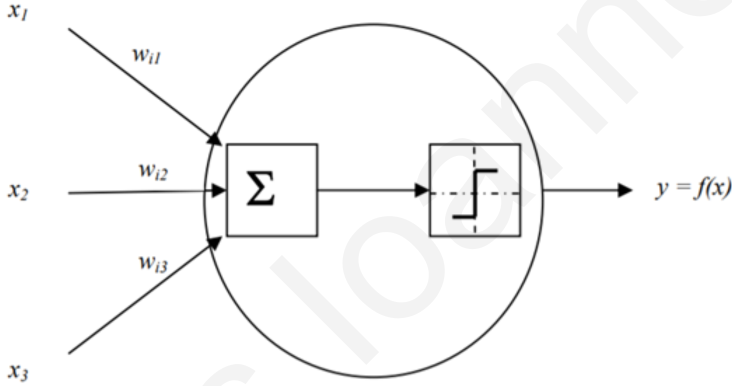
kyprosIoannou

2.3 Types of Artificial Neural Networks and Optimizers

2.3.1 McCulloch–Pitts (MCP)

In 1943, McCulloch and Pitts proposed the first Artificial Neuron. It is a straightforward model and consists of Xn as the number of inputs, Wn as the number of weight, an activation function, and y as output. Figure 2.4 shows the architecture of McCulloch and Pitts Neuron [33].

Figure 2.4: In this version, McCulloch and Pitts model has three inputs and is using the Threshold Function to either fire 1 or 0. [12]



Kypros Ioannou

We have an equal number of weights and inputs where we perform a simple multiplication between each pair. Moreover, the McCulloch sums all the products coming from the input/weights, and by using the Threshold Function(Equation 2.1 for the summation), the model either fires 1 if the sum is above a threshold, or 0. The McCulloch and Pitts model behaves similarly to the biological neuron we described in Section 2.2 by combining all the inputs, and the model fires if the final output is above a certain threshold.By adding bias, we bring the threshold to 0 (Figure 2.5), and we calculate the output according to Equation 2.2.

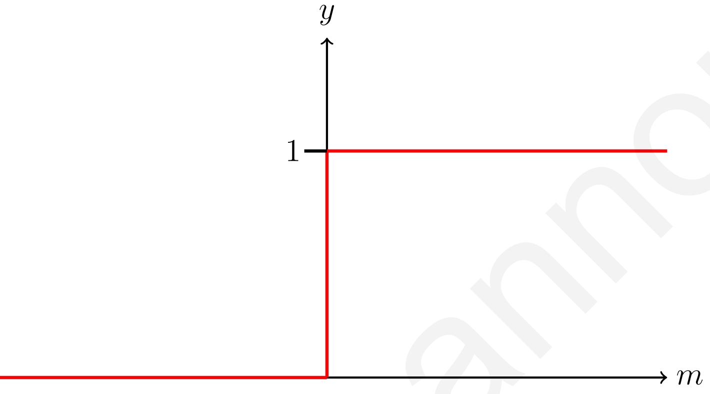
$$y = f(\sum_j W_i * X_j) \tag{2.1}$$

$$f(x) = \left\{ \begin{array}{ll} 1, & x > 0 \\ 0, & otherwise \end{array} \right\} \tag{2.2}$$

X(1) Inputs	X(2) Inputs	Output
0	0	0
0	1	0
1	0	0
1	1	1

Table 2.1: Truth table for AND

Figure 2.5: Threshold Function



Using McCulloch and Pitts, we can solve some 2D problems such as OR and AND Gate. For example, in Table 2.1, we have the AND Gate truth table where the three outputs are 0, and the last one is 1.
Looking at Figure 2.6, it is clear we can solve the AND Gate problem using only one decision line. By extension, we can solve OR Gate with only one decision line since the problem is similar to AND Gate.

Figure 2.6: And and OR Gate

(0,1) (1,1) (0,0) (1,0)

(0,1) (1,1) (0,0) (1,0)

Kypros Ioannou

For example, if we have two inputs X=[1,2] and weights W=[1,1] while the bias is 1.5, we can easily calculate the decision line following Equation 2.3 .

$$x_2 = -(\frac{x_1}{w_2}) * x_1 + (\frac{bias}{w_2}) \tag{2.3}$$

In order to make the model adaptable, the weights need to change based on the desired output. To make a better prediction, we move to Perceptron Networks and an algorithm that can adjust the weights. Figure 2.7 illustrates the Perceptron Learning Algorithm [15]. Firstly, we randomly initialize weights and bias, then inputs and output are presented where we calculate the actual output.We can decrease/increase the weights based on some simple rules (see Fig 2.7), or leave them untouched. When the desired output is one, we add to that output η times the corresponding input for that weight. On the other hand, when the desired output is 0, we subtract to that output η times the corresponding input for that weight.η is the learning rate, by which we define how fast or slow weights change.

Figure 2.7: Perceptron Learning Algorithm

Perceptron Learning Algorithm

1. Initiliaze weights and threshold
2. Present Inputs and desired Outputs
3. Calculate actual Output
4.Adapt Weights

If correct: Wi(t+1)=wi(t)

If output 0 should be 1: wi(t+1)= wi(t+1) + η Xi(t)

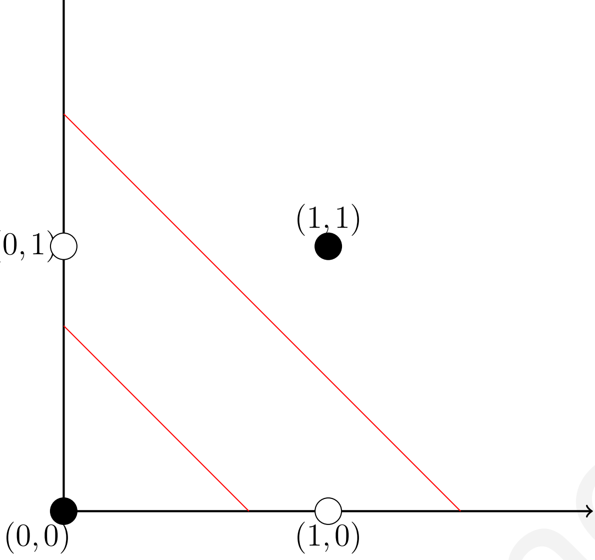
If output 1 should be 0: wi(t+1)= wi(t+1) - η Xi(t)

0 ≤ η ≤1

Kypros Ioannou

Using adaptive weights, we run into some problems we cannot solve, for example XOR Gate. Figure 2.8 helps us visualize the problem, as we need two decision lines in order to solve the problem.

Figure 2.8: Xor Gate



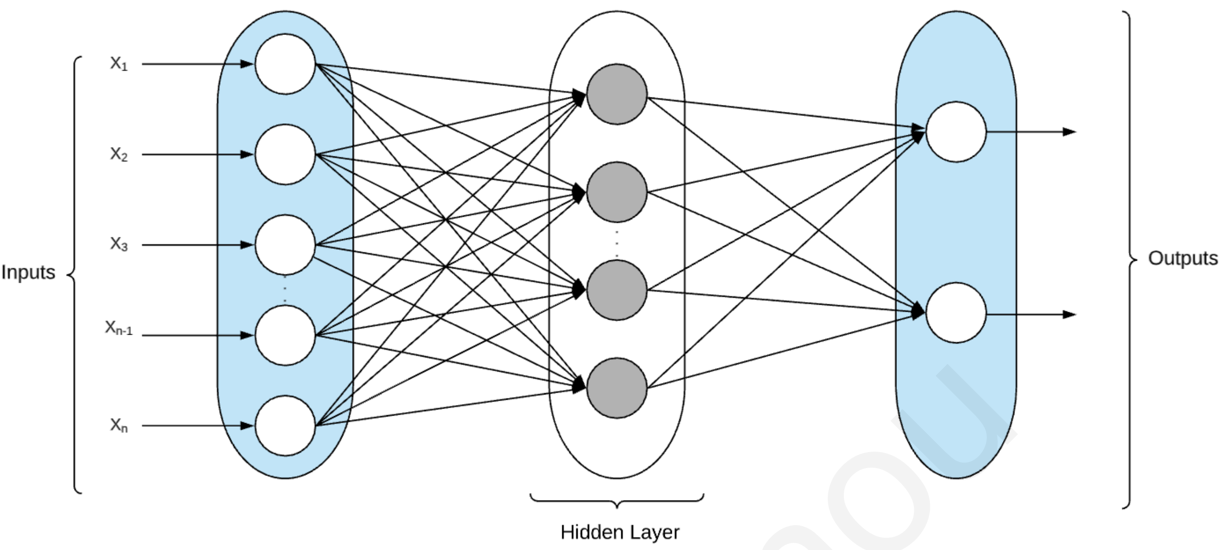
In order to solve a more complex problem, the next step is to use a Multilayer Perceptron network on which we shall elaborate in Section 2.3.2.

2.3.2 Multi-Layer Perceptron (MLP)

MLP is slightly different from the previous McCulloch and Pitts (MCP) model. That is because, instead of having only one neuron, we have a connected network of nodes (neurons). In the MCP model, we have a direct connection between inputs and outputs. Another crucial difference when solving more complex problems is the addition of an intermediate layer called "hidden layer". In Figure 2.9, the structure of MLP is present with N number of Inputs and Hidden nodes, and two outputs.

Kypros Ioannou

Figure 2.9: Multilayer Perceptron



The problem with standard perceptron is that it can solve only linear inseparable problems. Using an MLP, one perceptron may detect a different pattern and draw a different decision line than the other perceptron in the same network. By combining those decision lines, we may solve a problem such as XOR we described above. Nevertheless, we need to change the Heaviside Step Activation Function for a variety of reasons. First of all, since we have an intermediate layer or sometimes two, the hidden layer does not know the actual input. Second, since the Heaviside step function behaves like an on-off switch, we are only able to get “0” or “1”. The reason for behaving like an on-off switch is that is not differentiable. A derivative is a measure of how rapidly a function is decreasing or increasing. Thus, we lose almost all the information that we need in order to adjust the weights.Gladly, we have various arbitrary activation functions such as Sigmoid or Rectified Linear Unit (Relu) functions used frequently in many problems. The list of all activation functions is shown in Table 2.2.

Kypros Ioannou

Table 2.2: List of the most popular activation functions. [29]

Name	Plot	Equation	Derivative	Range
Heaviside		$f(x) = \begin{cases} 1 \; if \; x > 0 \\ 0 \; otherwise \end{cases}$	$f'(x) = \begin{cases} 0 \; if \; x \neq 0 \\ ? \; if \; x = 0 \end{cases}$	{0,1}
Logistic / Sigmoid		$f(x) = \frac{1}{1+e^{-x}}$	$f'(x) = f(x)(1 - f(x))$	(0,1)
TanH		$f(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}}$	$f'(x) = \frac{1}{x^2+1}$	(-1,1)
Rectified linear unit (ReLU)		$f(x) = \begin{cases} 0 \; if \; x < 0 \\ x \; otherwise \end{cases}$	$f'(x) = \begin{cases} 0 \; if \; x < 0 \\ 1 \; otherwise \end{cases}$	[0,∞)
SoftPlus		$f(x) = \ln(1 + e^x)$	$f'(x) = \frac{1}{1+e^{-x}}$	(0,∞)
Gaussian		$f(x) = e^{-x^2}$	$f'(x) = -2xe^{-x^2}$	(0,1)

Using an MCP, we could only perform binary Classification, but with MLP, we can solve Classification and Regression problems. Depending on the number of hidden layers and nodes (neurons) for each layer, we determine the complexity of the model. According to the Kolmogorov Theorem, three-layer perceptrons units can form arbitrary complex shapes and can separate any classes. Figure 2.10 illustrates the theorem starting with only a single perceptron with two inputs, then going up to two hidden layers with the same number of inputs.

Kypros Ioannou

Figure 2.10: Types of decision regions that can be formed by single and Multilayer Perceptron with one or two hidden layers and two inputs [30].

Structure	Types of Decision Regions	Exclusive-OR Problem	Classes with Meshed regions	Most General Region Shapes
Single-Layer	Half Plane Bounded By Hyperplane	A B B A	B A	
Two-Layer	Convex Open or Closed Regions	A B B A	B A	
Three-Layer	Arbitrary (Complexity Limited by Number of Nodes)	A B B A	B A	

We do not need massive changes compared to MCP when we want to calculate the output, as it can be readily seen from Equation 2.4. There we have the inner product of weights and inputs again, and the bias for each node. When the MLP calculates the output, it goes inside the activation function where the final output of that neuron is propagated to the next layer as input. This goes on until there are no more layers.

$$y = f(w^T \cdot x + bias) \tag{2.4}$$

2.3.3 Gradient Descent (GD)

After calculating the final output, it is necessary to understand how far we are from the desired output. We can calculate this using the Mean Square Error (Equation 2.5). We subtract each pair of an actual output (t_i) and predicted output (o_i), and we divide all of them by the number of the output neurons (n).

$$MSE = \frac{1}{n}(\sum_{i=1}^{n} t_i - o_i) \tag{2.5}$$

Gradient Descent is one of the most popular algorithms for training a neural network. In order to reduce the error function, we move each weight to the direction of the steepest descent. Furthermore, we adjust weights based on the negative derivative

Kypros Ioannou

of the error function pertaining to each weight (Equation 2.6).

$$\Delta w_{ij} = -\eta \frac{dE_p}{dw_{ij}} \tag{2.6}$$

2.3.4 Backpropagation Algorithm (BP)

One main problem we face is that we need two variables to adjust the weights each time; the actual and the predicted output. However, the only actual output we get is from the final node. The backpropagation algorithm is proposed for solving this problem, making the neural network hidden layers weights and input weights as adjustable as the final weights. Figure 2.11 describes the algorithm starting with the forward pass where we calculate the output for each node. We then proceed with the Backward pass with two phases. We calculate the error for each node, and then update the weights using Δw (Equation 2.6).The next step is to calculate the sum of the inner product for the hidden layers between the previously calculated errors and their corresponding weights instead of subtracting between the target and predicted output. Furthermore, after calculating all of the errors, we adjust each weight starting from the weights connected with the input and continue until we reach output weights. Lastly y is the actual output and d is the desired output.

Figure 2.11: Backpropagation Algorithm [44]

```
Backpropagation

Repeat:
      For each pattern :
            // Forward Pass
            Calculate the output
            // Backward Pass
            For each layer j, starting at the output:
                  For each unit i:
                         // Compute the error
                         If output neuron: δij = yij(1 − yij)(dij − yij)
                         If hidden neuron: δij = yij(1 − yij) Σ δik · Wjk
                         For each weight to this unit:
                              Compute and apply Δw
      Compute total error
      Increment epoch counter
Until small enough error or epoch counter exceeded
```

2.3.5 Recurrent Neural Networks (RNN)

We previously described the first generation of Artificial Neural Network [31]; we continue with the second generation starting with the Recurrent Neural Network.

Kypros Ioannou

An RNNs is helpful when the data at hand is in a specific order (sequential data). Sometimes the order of the data is helpful; for example, the sequence of words in a document is essential, or the stock price over time. Furthermore, we can use RNN to solve temporal context problems since RNN utilises internal memory. The next output mainly depends on the previous outputs and inputs. Jordan [24] and Elman [19] Networks are two basic RNN using a slightly different backpropagation algorithm called "Backpropagation Through Time Learning Algorithm" [50].

Figure 2.12: Jordan Architecture with two 3 inputs and three States Units that propagate the previous output. [24]

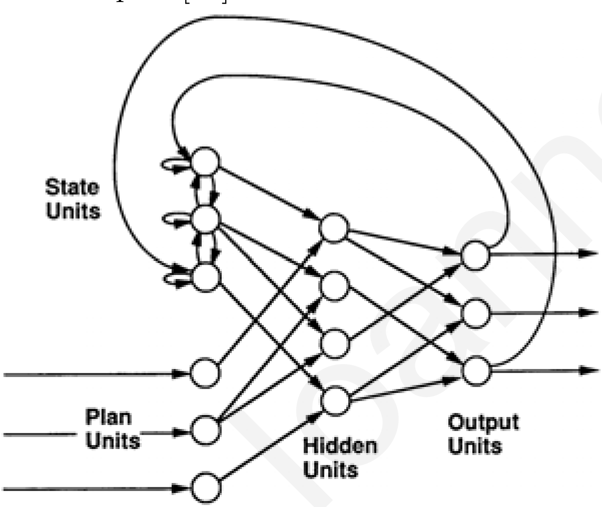
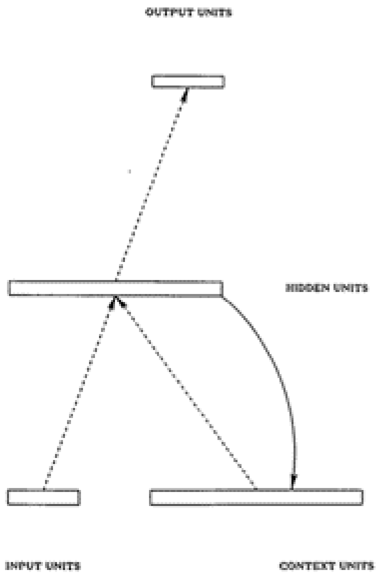


Figure 2.13: Elman Architecture with an internal loop between input and hidden layer. [19]



Kypros Ioannou

Figure 2.12 and 2.13 illustrate Jordan and Elman architecture respectively.Both models have an iterative stage.Beginning with Jordan, we have some extra inputs called "State units". Those units have the previously predicted output as an input and the value of 1 as weight. Since each time we feed the current output as new input, we realise that the subsequent output is affected by the previous one.Therefore, we have a loop between the input and hidden layers in the Elman structure instead of the output layer. Having the loop between those layers implies that the output does not depend on the previous output and can vary freely.

2.3.6 Convolutional Neural Networks (CNN)

The Convolutional Neural Network is the network we will use for this thesis and is in the same generation as RNN. We use it in many different areas such as image classification (in which performs very well), video recognition, medical analysis, and Natural Language Processing(NLP), such as in the case of this thesis.

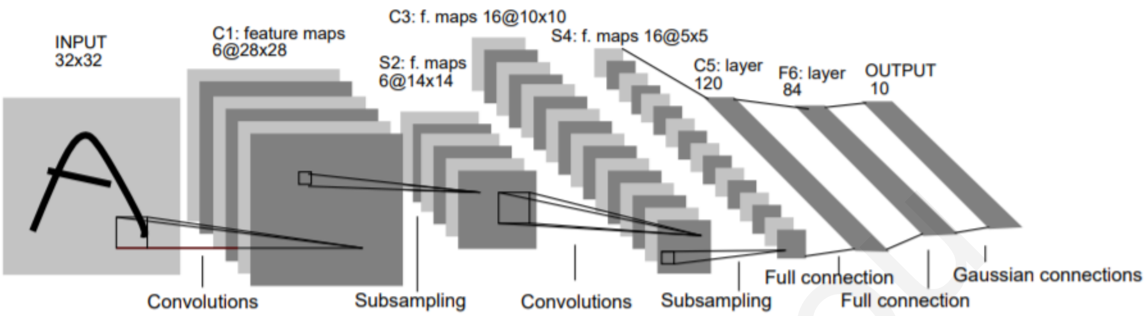
Multilayer Perceptron is used most of the time as a fully connected network, meaning that we connect each node(neuron) for each layer with every node of the next layer. Having a fully connected network may lead to overfitting which is expected and one of the reasons is the complexity of the model. However, CNN takes advantage of the hierarchical patterns in data and assembles patterns with high complexity with the help of smaller/simpler patterns.

Architecture of Convolutional Neural Network

The architecture of CNN has three main parts: the convolutional layer, the pooling layer, and a fully connected network for supervised learning. Figure 2.14 outlines the architecture of a CNN model with a convolutional and pooling layer followed by the same pair of layers. In a CNN, the input is a tensor with the shape of (number of train inputs)$\times$(input height) $\times$ (input width) $\times$ (input depth or channel).

Kypros Ioannou

Figure 2.14: A Convolutional Neural Network where we have the input introduced. We then extract features using convolution layer and we reduce the sizes by doing subsampling. Lastly we use a full connected neural network for the classification [28].



When the input of a CNN is an image, the first shape of the tensor is the number of images we feed the model with each time. Then we have image height and width, and if the image is not grayscale, we will have a value of three for the input channel. A grayscale has a value for one. The reason coloured images have the value of 3 is due to RGB (Red-Green-Blue). We need three tables to represent all the colours of the image.

Convolutional Layer

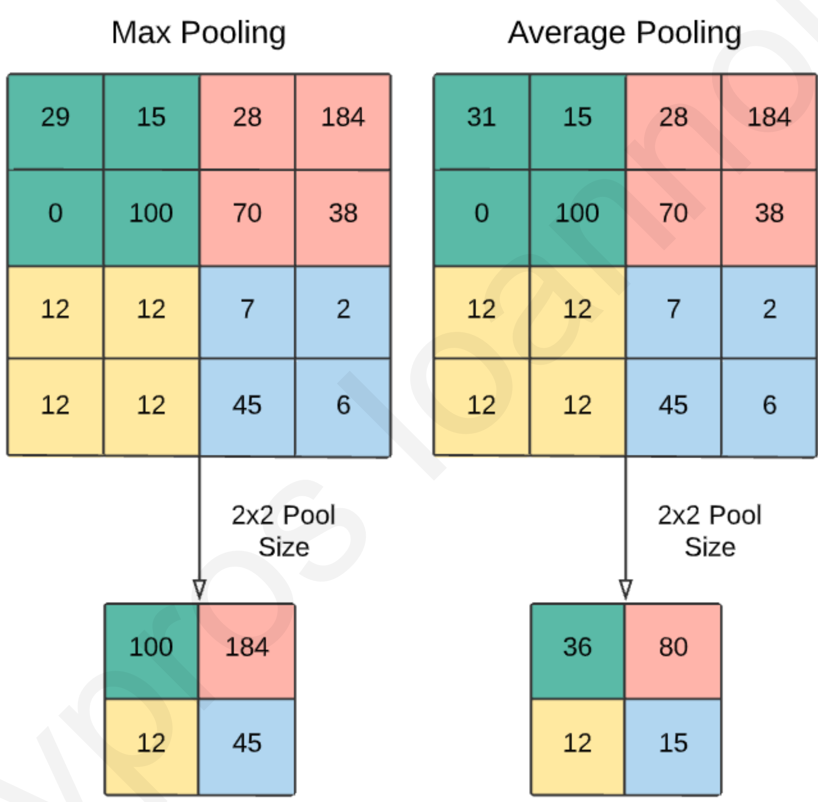
A convolutional layer has N number of filters(kernels), and each filter has a dimension of $k \times k \times m$. The first two are dimensions of the image that are smaller than the actual dimension. m as we have already mentioned is the number of channels; we can either have the same as the original image or less (3 for RGB and 1 for grayscale). When a convolutional layer convolves, the input is passed to the next layer, which is another convolutional layer or a pooling layer after first passing through the activation function (ReLu). The behaviour of the convolutional layer is similar to the response of neurons in the visual cortex to a specific stimulus [23]. Each convolutional neuron proceeds with its input for its receptive field. However, a fully connected neural network like an MLP receives inputs from all nodes of the previous layer, whereas a convolutional layer receives from only a restricted area. Usually this area is a square of dimensions 2 by 2 up to 5 by 5.

Kypros Ioannou

Pooling

The next important layer after the convolutional layer we are going to talk about is the pooling. The pooling layer (subsampling) is responsible for reducing the image's dimensions, free parameters of the network, and the network's complexity. We have various types of pooling, e.g. Max Pooling, Average, Min Pooling etc. By performing subsampling, we reduce the dimensionality of the data and we create invariance to small shifts. Figure 2.15 shows an example of Max and Average Pooling using a kernel size of 2 by 2.

Figure 2.15: Max and Average Pooling.



Kypros Ioannou

2.3.7 Line Search

The Line Search strategy is one of the two basic iterative approaches to find the minimum x^* of an objective function $f : R^n \to R$. Moreover, line search finds first the direction of descent along which the objective function f will be reduced, and then computes a step size that determines how far x should move along that direction (Equation 2.7 for Update rule) [7].

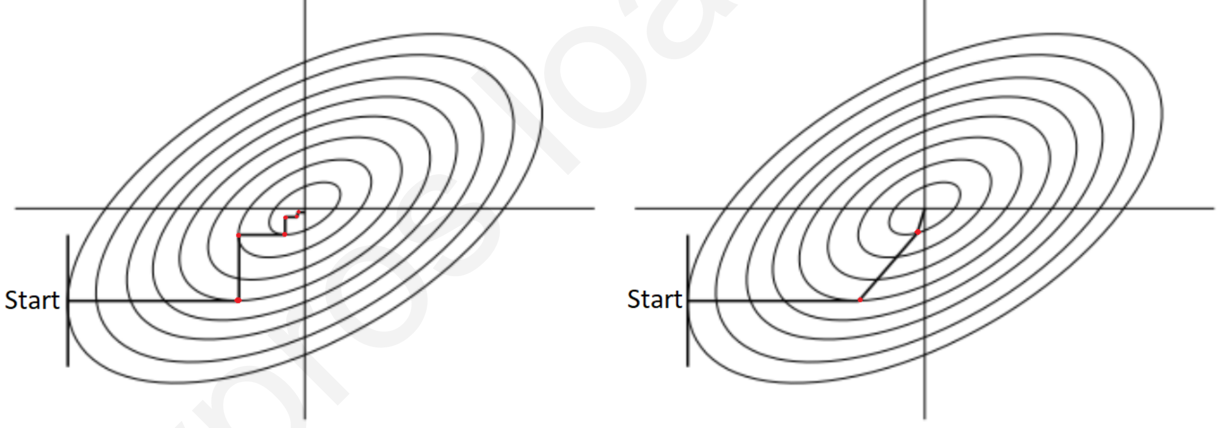
$$x_{k+1} = x_k + a_k p_k \tag{2.7}$$

Where: a_k : is the Step Size
p_k : is the Descent Direction

2.3.8 Conjugate Gradient (CG)

Conjugate Gradient is a line search algorithm that tries to minimise the error in one direction at a time. First it takes the direction of steepest descent and goes to the minimum, whence it may need to re-evaluate the error and the gradient in order to do that. Once it finds the minimum of the first direction, it goes to another direction. However, instead of affecting the gradients of the previous directions, what it does is to go in such a way that all the previous gradients are unaffected by the current move. In an N-Dimension problem, the CG can converge to an optimal solution in at most N steps [4]. Figure 2.16 shows an example of Conjugate Gradient(CG) and Gradient Descent(GD). As we can see, CG needs only three steps in order to converge, whereas GD needs more.

Figure 2.16: Gradient Descent (left) vs Conjugate Gradient (right) on a 2D problem.



2.3.9 Newton's Method

We already mentioned Gradient Descent, where it is used for minimising the error function. We keep moving in the direction of the negative derivative gradient until we eventually reach the local minimum. We need to add a fixed step size from the start, while we need to keep in mind that it is a first-order method. We assume that a neural network model's error surface looks and behaves like a plane by having a first-order method, ignoring any curvatures the surface may have. Even if Gradient Descent is a very popular optimiser, in practice it can be relatively slow. The Newton Method [40] is a second-order algorithm we can use in order to solve that problem.

If we want to find the minimum of a function $f : R \to R$, we first need to find the zero of its derivative which then signifies that $f(x)$ exhibits an extremum. We approximate this using a Taylor expansion about some point x_o (Equation 2.8).

$$f(x_o + x) \approx f(x_o) + f'(x_o)x + f''(x_o)\frac{x^2}{2} \tag{2.8}$$

Moreover, we need to choose x since we want $f(x_o + x)$ to be the minimum. We take the derivative of the expansion in regards to x and set it to zero, as seen in Equation 2.9 .

$$\frac{d}{dx}(f(x_o) + f'(x_o)x + f''(x_o)\frac{x^2}{2}) = f'(x_o) + f''(x_o)x = 0 \Longrightarrow x = -\frac{f'(x_o)}{f''(x_0)} \tag{2.9}$$

If we are trying to find the minimum in a nonlinear function f, we may not be able to calculate the minimum from the start. In order to find it, we need to repeat the process with an update rule (Equation 2.10).

$$x_{n+1} = x_n - \frac{f'(x_n)}{f''(x_n)} = x_n - (f''(x_n))^{-1} f'(x_n) \tag{2.10}$$

What we described above refers to a one-dimensional function; let us generalise by having a function $f : R^n \to R$. Here, we replace the first derivative with the gradient and the second derivative with the Hessian matrix (Equation 2.11,2.12).

$$f'(x) \to \nabla f(x) \tag{2.11}$$

$$f''(x) \to H(f)(x) \tag{2.12}$$

We update the update rule as well to (Equation 2.13):

$$x_{n+1} = x_n - (H(f)(x_n))^{-1} \nabla f(x_n) \tag{2.13}$$

2.3.10 Hessian Free Optimization (HFO)

Even though the Newton method is an excellent method to minimise the error function, we face some crucial problems when using it. The main problem is that we need to calculate the inverse of the Hessian Matrix for every update we have done (Equation 2.14: Hessian Matrix of the error with respect to their weights). The

Kypros Ioannou

Hessian Free Optimisation is proposed in order to solve this expensive computational problem [32].

$$H(e) = \begin{pmatrix} \frac{\partial^2 e}{\partial w_1^2} & \frac{\partial^2 e}{\partial w_1 \partial w_2} & \cdots & \frac{\partial^2 e}{\partial w_1 \partial w_n} \\ \frac{\partial^2 e}{\partial w_2 \partial w_1} & \frac{\partial^2 e}{\partial w_2^2} & \cdots & \frac{\partial^2 e}{\partial w_2 \partial w_n} \\ \vdots & \vdots & \ddots & \vdots \\ \frac{\partial^2 e}{\partial w_n \partial w_1} & \frac{\partial^2 e}{\partial w_n \partial w_1} & \cdots & \frac{\partial^2 e}{\partial w_n^2} \end{pmatrix} \tag{2.14}$$

Instead of calculating and inverting the Hessian (H) matrix as we would conventionally need to do (Equation 2.13), we compute the inner product between H and an arbitrary vector v ($H \cdot v$). The curvature matrix or Hessian can be approximated in many different ways, and one way is to use Gaussian (G)-Newton Matrix. Martens [32] has tested the Gaussian Newton matrix. In every test using either G or H, the Gaussian consistently resulted in much better search directions even where negative curvature was not present [32]. Another benefit of using the Gaussian is that it is two times faster than the Hessian while only using half the memory. Approximating the Hessian matrix instead of calculating and inverting it can help Machine Learning models to perform faster and be more efficient than using the original Newton Method. A detailed analysis of the HFO method is given by Charalambous [16].

2.3.11 Word Embedding

When dealing with a natural language problem (NLP), an important problem is transforming characters and words into a form that a machine learning model is able to understand. In the case of a photo, as we already mentioned in Section 2.3.6, we have the input in numerical value. There are various methods; some of them make no assumptions regarding semantics and the similarity of words while others do. One of the simplest methods is One-Hot Encoding, where we create a vocabulary size vector with 0s and 1s. For every word, only the corresponding column has the value of 1, and the others columns for the same words have 0. An example of that method is in Table 2.3, where we have four columns, and only the words corresponding to the cells of said words have the value of 1.

Kypros Ioannou

Table 2.3: One hot encoding where each word in its corresponding cell has a value of 1.

	car	man	plane	child
car	1	0	0	0
man	0	1	0	0
plane	0	0	1	0
child	0	0	0	1

On the other hand, another method of representing the words and converting them is using Word Embedding. We convert the words into real-value vectors, and we encode the meaning of that words. Also, using word embedding similars, words have similar representation. In this project, the main words embedding model we are going to use is the Word2Vec that was created by a team of researchers in Google led by Thomas Mikolov in 2013 [35] [36].As the name suggests, Word2Vec converts a word into a vector, a list of numbers representing the word.Wor2Vec model can capture both syntactic and semantic similarities between words. One famous example of the vector from a trained Word2Vec is shown below, where the vector of "king" minus the vector of "man" has the same meaning as the vector of "queen" minus the vector of "woman".

$$Vector("King") - Vector("Man") = Vector("Queen") - Vector("Woman")$$

The model has two layers; the purpose of the model is to reconstruct the words' linguistic context. It takes a large corpus of text as an input and has a vector space with hundred of dimensions as output. Then each word is assigned to a unique vector [8]. Word2Vec can generate a distributed representation of words using one of the two model architectures: continuous Bag-of-Words (CBOW) or continuous Skip-Gram. The model predicts the current word from a window of surrounding context words in the continuous bag-of-words architecture. The order of the words in the context has no bearing on the outcome (bag-of-words assumption). The model predicts the surrounding window of context words in the continuous skip-gram architecture based on the current word. Nearby context words are given more weight in the skip-gram architecture compared to more distant context words. Figure 2.17 illustrates the architecture of the two models.

Kypros Ioannou

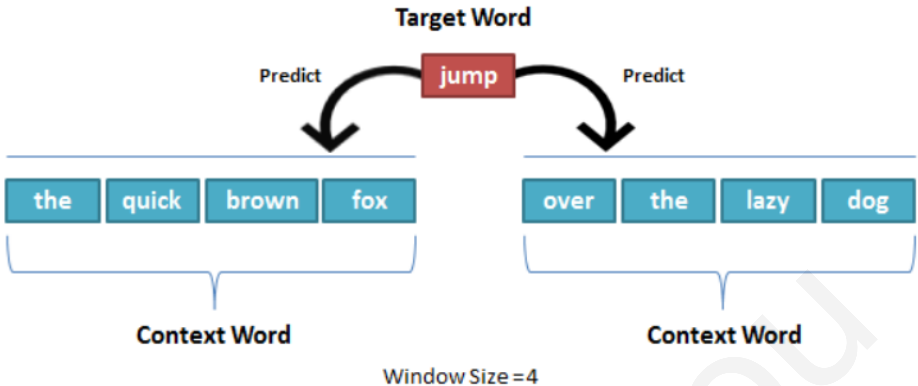
Figure 2.17: The CBOW architecture predicts the current word based on the context, and the Skip-gram predicts surrounding words given the current word [35]

.

Kypros Ioannou

Figure 2.18 shows an example of each model in order to better understand how each model works. We start with the sentence "the quick brown fox jump over the lazy dog". In CBOW, given the words ("the quick brown box, "over the lazy dog"), the model predicts the word in the red box ("jump"). However, Ski-gram does the opposite of CBOW; given the word "jump" we predict the other words in the sentence ("the quick brown box", "over the lazy dog").

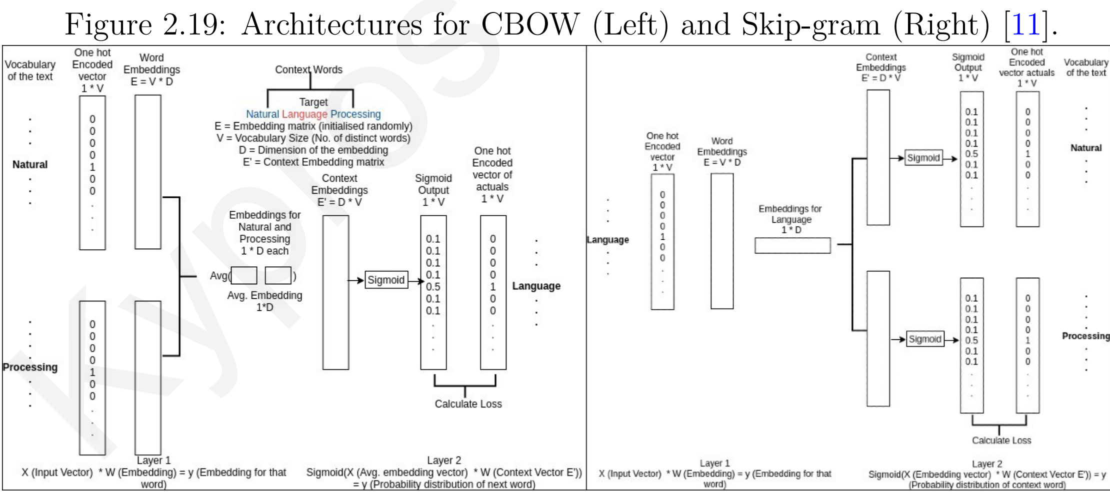
Figure 2.18: In CBOW, given the words (the quick brown box, over the lazy log), we would want to predict jump. In Skipgram just the opposite given the word jump, we would want to predict (the quick brown box, over the lazy log) [11].



How does each Model learn?

We begin with the CBOW model on the left side of Figure 2.19, and we use the "Natural Language Processing" as an example to study how it is able to learn. We use "Natural" and "Processing" as the context words and "Language" as the word we want to predict.

Figure 2.19: Architectures for CBOW (Left) and Skip-gram (Right) [11].



Kypros Ioannou

As we already explained at the start of subsection 2.3.11, the input will be a one-hot encoded vector of V terms (size of vocabulary/total number of unique words) only a single one. For this problem/example, the vocabulary consists of 5 words ("Natural", "Language", "Processing", "is", "Great"). Then each vector for each word will have a length of five, and for "Natural" it will be [1,0,0,0,0]. The next step is to initialise an embedding vector (E) randomly. The size of E will be $V \times D$, where D

is arbitrarily defined. This vector (E) will be the weight matrix for the input layer where we will multiply the input one-hot encoded vector with E. The product will give us the embedding vector for the context words (Natural and Processing) of size 1 D.

The model will average the embedding vector (E) of the context words in the hidden layer and then will be multiplied by another vector called Context Vector (E') whose size is again $D \times V$. The product of this multiplication will give as a vector of dimensions $1 \times V$ which when inserted in the sigmoid function to get the final output. The final output of the model will be compared with the one-hot encoded vector of the word ”Language” [0,1,0,0,0], and the model will calculate the loss function. This loss is backpropagated and the model is trained using Gradient Descent as we described in Subsection 2.3.3

For the Skip-gram it is just the opposite. We have the one-hot encoded vector for the middle word instead of the whole sentence multiplied with the weight/embedding vector E. The product will be the output of the input layer and the input of the hidden layer. Furthermore, we multiply with Context Vector E' and then we passed it through the sigmoid function to get the final output. We compare it with the context words, and again, the loss is calculated and backpropagated.

Why Word2Vec?

Word2Vec is not the only embedding tool that exists; Chatzimiltis [17] already uses a different data representation tool called Bert Embedding. More information about Bert can be found in Chatzimiltis’s dissertation. In Tables 2.4 and 2.5, we compare simple tokenisation/One-Hot Encoding, Word2Vec and Bert Embedding. We have better results with Word2vec. Furthermore, we had some problems using Bert embedding for our problem, as it needs a significant amount of time to load the Bert file before we start doing the data representation and then train our model.

Kypros Ioannou

Table 2.4: Comparison Between Tokenization and do One-Hot Encoding, Word2Vec Embedding and Bert Embedding for first Dataset.

CNN/HFO	Epochs	Accuracy	Loss
Tokenisation/One-Hot Encoding	75	0.8862	0.208
Word2Vec Embedding	75	0.9453	0.100
Bert Word Embedding	75	0.7900	0.322

Table 2.5: Comparison Between Tokenization and do One-Hot Encoding, Word2Vec Embedding and Bert Embedding for second Dataset.

CNN/HFO	Epochs	Accuracy	Loss
Tokenisation/One-Hot Encoding	75	0.8625	0.228
Word2Vec Embedding	75	0.9313	0.143
Bert Word Embedding	75	0.8095	0.303

2.3.12 Subsampled Hessian Newton (SHN) Method

The main goal of this dissertation is the use of Hessian-Free Optimisation (HFO) with a Convolutional Neural Network (CNN). HFO has addressed some problems of the original method (Netwon) by approximating the Hessian matrix. Stochastic Gradient Method (SGD) was and still is a very popular method, but as we describe in Section 2.3.8, it is much faster for Conjugate Gradient to converge. We can use SGD in various deep learning models, but up until 2020 we did not have any Netwon method for Convolutional Neural Networks. In most of the literature [13] [21] [25] [32] used a simpler model such as MLP with the Netwon method. The reason for avoiding using a Convolutional Neural Network is because of the complexity of the model. Combining CNN with the Netwon method we will involve a much more complicated operation [49] than using a simpler model with Netwon as well.

As we mentioned, until very recently we did not have a way of implementing Netwon and CNN. Wang et al. [49] introduced a new variation of Hessian-Free Optimisation called "Subsamples Hessian Newton" (SHN). We are not the first to use this variation of HFO with CNN to solve a problem. Leontiou [29] as well as Chatzimiltis [17], and Pafitis [38] already used that method (Section 2.3.13) in order to predict protein structures and classify Alzheimer MRI images.

The SHN algorithm is shown in Algorithm 1, where (36) is Equation 2.17, (35) is Equation 2.16, and lastly, (37) is Equation 2.18 Due to the high complexity of the algorithm, we refer the reader to the original paper of SHN in order to get a more in-depth understanding of the algorithm.G is the Gauss-Newton approximation and λ is a parameter decided by how good the function reduction is."Specifically, if θ + d is the next iterate after line search, we define ρ (Equation 2.15) as the ratio between the actual function reduction and the predicted reduction. By using ρ, the parameter λ_{next} for the next iteration is decided by where (drop,boost) are given constants. From Equation 2.18 we can clearly see that if the function value reduction is not satisfactory, then λ is enlarged and the resulting direction is closer to the negative gradient".

$$\rho = \frac{f(\theta + d) - f(\theta)}{\nabla f(\theta)^T d + \frac{1}{2}(d)^T G d} \tag{2.15}$$

Algorithm 1: A subsampled Hessian Newton method for CNN. [49]

Given initial θ. Calculate f (θ);
while $\nabla f(\theta) \neq 0$ **do**
    Choose a set $S \subset \{1, \ldots, l\}$;
    Compute $\nabla f(\theta)$ and the needed information for Gauss Newton matrix-vector products;
    Approximately solve the linear system in (36) by CG to obtain a direction $\boldsymbol{d}$;
    $\alpha = 1$
    while *True* **do**
        Compute $f(\theta + \alpha \boldsymbol{d})$;
        if *if (35) is satisfied* **then**
            break;
        end
        $\alpha \leftarrow \alpha/2$;
    end
    Update λ based on (37);
    $\theta \leftarrow \theta + \alpha \boldsymbol{d}$
end

$$f(\theta + \alpha \boldsymbol{d}) \leq f(\theta) + \eta \alpha \nabla f(\theta)^T \boldsymbol{d} \tag{2.16}$$

$$(G + \lambda I)\boldsymbol{d} = -\nabla f(\theta) \tag{2.17}$$

$$\lambda_{next} = \begin{cases} \lambda \times drop & \rho > \rho_{upper} \\ \lambda & \rho_{lower} \leq \rho \leq \rho_{upper}, \\ \lambda \times boost & \text{otherwise} \end{cases} \tag{2.18}$$

2.3.13 CNN With Hessian Free Optimisation Related Work

We are not the only ones that we are using Hessian Free Optimisation with Convolutional Neural Networks to solve problems. Three others University of Cyprus students are already done some work in different problems to this thesis.Leontiou [29] use CNN with HFO for Protein Structure Prediction.Proteins can be found in any

living organism and each protein contains smaller units named: amino-acids. Furthermore, proteins have different functionalities and what makes a protein different from other proteins is the amino-acids sequence. Moreover, the interactions between the amino acid's of a protein force the protein to fold into a specific three-dimensional structure. That structure determines the function of each protein [29].For analyzing the structure of the proteins more efficiently proteins are separate into 4 different categories: Primary, Secondary, Tertiary, and Quaternary structure.Leontiou [29] mostly focus on secondary structure and using the amino acid sequence as input.Chatzimiltis [17] also uses Convolutional Neural Networks with HFO as Leontiou to predict the secondary structure o protein. However, instead of feeding the model with raw inputs, it went a step further and used embedding as we do in the dissertation.

Furthermore, we are not using the same embedding method as Chatzimiltis for two reasons. The first reason is that Chatzimiltis uses the BERT model in order to embed his data. BERT is a very popular word embedding method, but for our problem did not perform well. We can see that in the 2.3.11 subsection of this thesis. The second reason that Chatzimiltis did not use the worldwide BERT versions is that ProBert can only be used for proteins. Therefore we cannot use it.

The last related work we are going to talk about is from Pafitis [38] where he uses Convolutional Neural Networks with HFO for an image classification problem. The problem was to classify MRI images from people with Alzheimer disease and images of normal cognitive people.A normal cognitive person is a person with no neurological or psychiatric problems whose cognitive abilities decline with normal aging. The main focus of his dissertation was to automate the identification as to which of these two categories an MRI image belongs to.

2.3.14 Ensemble Methods

An ensemble classifier consists of multiple weak models such as Decision Trees or Random Forest with only one objective: to combine their results in order to solve a problem [37]. In addition, an ensemble classifier can help to reduce variance and help to avoid overfitting. There is a variety of different types of Ensemble Methods, but here we shall restrict ourselves to just two of them.

- Bagging = Boostratp AGGregatING
- Boosting

Kypros Ioannou

One of the differences between Bagging and Boosting is how we perform the random sampling. Each algorithm has N number of learners (weak models). However, each learner is not guaranteed to have the same data as the others in the same model. In the case of Bagging, all of the data that have the same chance to occur to another learner. With Boosting on the other hand, we add some weights and some data will appear more frequently than others. Another difference between these two algorithms is the way in which we train each model. Bagging trains the learners in a parallel way, whereas Boosting does so sequentially. In the latter, we redistribute the weights at each training step. The weights increase when we deal with incorrectly classified data in order to focus on the more complex cases. Furthermore, since we have a sequential model, each learner takes the previous learner success into account. Lastly, Bagging results are obtained by taking the average of all learners output (majority vote). In Boosting, we assign an additional set of weights, which this time is for the learners and the data. We then take the weighted average of their estimates.

Kypros Ioannou

Chapter 3

Design and Implementation

3.1 Data

An essential part of every machine learning project is choosing the appropriate data to train the model at hand. Since we are training a model in order to classify emails, it is better to use real rather than constructed data. For that kind of problem, most of the times it is very difficult to find legitimate emails from different people due to privacy considerations. On the other hand, spam emails are much easier to find, since they are generic emails that reach millions of email accounts. It is therefore easy to set a "net" of email accounts and collect all the spam emails which the email provider sends to the spam folder.

The data we are going to feed to our model comes from 3 different sources. All of the ham (legit) emails are coming from the Enron Dataset. As we mentioned in the previous paragraph, we need realistic legit emails, and an excellent source is the Enron Dataset. Enron Corporation was an American energy, commodities, and services company based in Houston, Texas [5]. It was founded in 1985 and filed for bankruptcy in 2001 due to the infamous Enron Scandal where some of the employees were sending fraudulent emails to other employees. In the aftermath of an accounting fraud between the employees, the company shut down and emails became available to the public.

Compared to the ham datasets, the spam dataset is a blend of 3 different sources: Enron, SpamAssassin's, and Honeypot Datasets. SpamAssassin is the #1 Open Source anti-spam platform giving system administrators a filter to classify email and block spam [1]. For this dissertation, we are going to use six different datasets which we retrieve from an academic article [34]. In Table 3.1 we can see how many emails are contained in each dataset.

Kypros Ioannou

Table 3.1: The table of the six different datasets that we retrieved from the article: [34].Each dataset is used separately and some datasets have common emails. Each dataset has more than 5000 email either as Spam or Ham.

Dataset No:	Number of emails	Source
1	3673 Emails(Ham) 1500 Emails (Spam	Ham Emails ->Enron Dataset Spam Emails ->Enron Dataset Total ->5172
2	4361 Emails (Ham) 1496 Emails (Spam)	Ham Emails ->Enron Dataset Spam Emails ->Spam Assassin+ Honeypot Total ->5857
3	4012 Emails (Ham) 1500 Emails (Spam)	Ham Emails ->Enron Dataset Spam Emails ->Enron Dataset Total ->5512
4	1500 emails (Ham) 4500 emails (Spam)	Ham Emails ->Enron Dataset Spam Emails ->Enron Dataset Total ->6000
5	1500 Emails (Ham) 3675 Emails (Spam)	Ham Emails ->Enron Dataset Spam Emails ->SpamAssassin + HoneyPot Total ->5175
6	1500 Emails (Ham) 4500 Emails (Spam)	Ham Emails ->Enron Dataset Spam Emails ->Enron Dataset Total ->5512

We have six separate datasets that do not contain the same amount of emails. If we see the total number of emails, the difference is not significant, but we have a big disparity in spam and legit emails for each dataset. For the first three datasets, the legit emails are two to three times the number of spam emails. For the last three datasets, we have the opposite: the spam emails are two-three times the number of legit emails. In Chapter 4, we will see along the results how the lower number of spam or legit emails affects the model.

Lastly, the actual Enron Dataset has over 500,000 emails; neither we nor other researchers use all of them. Most of them construct a dataset by combining the Enron Dataset with other datasets [20] [14]. Metsis et al. [34] are not the only ones that use SpamAssassin and Enron Dateset, many other academic papers also use them to train their models [51] [20] [14]. Each dataset has 2 folders (Spam and Ham) where each contains the emails in txt format. However, the txt files may contain more than one email, and aside from the body, the file may also contain

Kypros Ioannou

the recipient, carbon copy (cc) and the subject. Because we want to do a direct comparison with work of Metsis et al. [34], we will train our model for each dataset without combining them into a big dataset. The only thing we will change is the algorithm.

3.2 Data Preprocessing

In order to prepare the data to be fed into the Convolutional Network, we need to convert the text (emails) into numerical values, and not just that. As we mentioned in Section 3.1, we keep each email in a different txt file. The first thing we need to do is to load all of the emails in order to combine them and create a dataset. All of the preprocessing we shall describe here and in the following sections can be found in the Appendix. We apply the following six steps to convert the data to the most appropriate form for the best possible results. The list of the steps is as follows:

- Load the data
- Add the class for each email
- Clean the data from nan(non-number) values and duplicates
- Remove Punctuation and Stop Words
- Tokenization
- Use Word Embedding tool to convert our data

Load Data

First, since the data was separated, we had to load and convert it into a type of file which we can use and with which other scientists can experiment in the future. A widespread type of that sort is the CSV. We then begin by loading the legit emails. We use the Parser function, which is an adept function at separating emails parts. Some parts of the email header mentioned in Section 2.1.2 remain in some of the emails. The best option is to remove them since they cannot be used for all of the data. That's the reason we use the Parser function, as we can retrieve just the body of the email fast and reliably. Before we save an email as a CSV file, we have to check for any non-Latin characters. Figure 3.1 has an example of a non-Latin character email that cannot be allowed into the network.

Figure 3.1: An email with non-Lating characters.

```
Subject: fw : ¶ w - èßô - ¶ üë? © ¡ aßô - ¶ ³ ] - p ³ ì » Ý - n?ß !
??? ? ? - ? ¶ ? ? ?  ? ? ? ?
```

Assign Class and Clean Data

When we load the data, we do not combine the legit and spam emails from the start. The reason for that is that we do not assign the class for each email, hence each category is in a different CSV file for the moment. In order for the model to understand the class, we cannot assign a class to each dataset that is not numeric. For example, Spam and Ham are good examples of distinguishing which email is spam and not, but the model cannot understand characters. Furthermore, we assign "1" as the class for the spam emails and the value of "0" for legit emails for each dataset.

The next step is to check whether we have any duplicates emails or non value emails. We do not want to let any duplicate since the model may behave better or worse based on how it learns the data. If for example we let the duplicates and the model behave well for those emails, we may have a model with better accuracy compared to removing them, but that is not the actual accuracy of the model. The more emails are correctly classified by the model, the better the accuracy will be. We increase the model's accuracy in this example by letting the model predict more than once the same email, but that is wrong. The same is applied for having duplicate emails which the model behaves badly when learning. Finally, we check for any emails that do not have a body, since if they do not the specific column will be empty.

Remove Punctuation and Stop Words

Before splitting the emails into tokens, we first need to perform some additional "cleaning". We will remove punctuation marks and stop words; The purpose of this process is to get rid of unhelpful parts of the data that do not provide any extra information that may help us classify the emails more accurately. A Stop Word usually refers to the most common words in a language e.g. “the”,” is”, “and” etc.

Tokenization

The final thing we need to do before converting the emails into numeric vectors is to tokenise them. By tokenise, we mean that we split the remaining words of each email into tokens. We use a very popular function for that called "word_tokenize" from the Natural Language Toolkit (NLTK) library.Just before the word_tokenize function sends back the tokenised emails, we make sure to have each word written in lowercase. That is done in order to handle two identical words the same way. Having

Kypros Ioannou

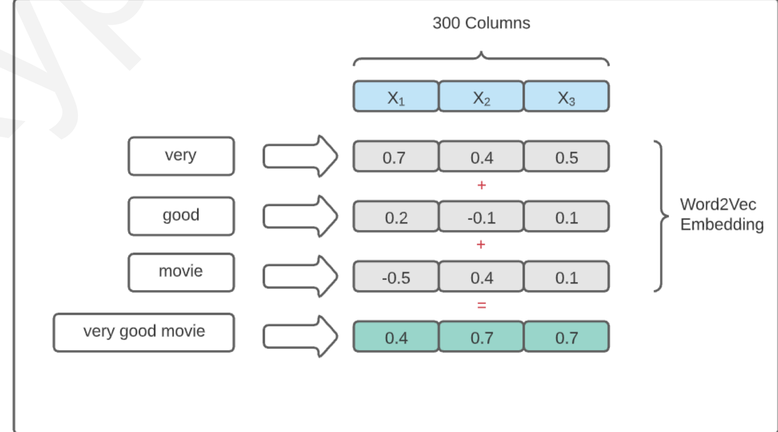
one of them start with a capital letter or containing only capital letters makes the model process it differently.

3.3 Word Embedding on Email Data

Gensim [43] is an open-source library for unsupervised topic modelling and natural language processing that gives us the flexibility, to develop word embedding.We already explained how Word2Vec works and how it can learn, but what we did not explain is what tool we shall use to convert the emails into numerical values. Using the Gensim library, we can either choose if we want to use the CBOW Model or Skip-gram to train the embedding model, but the default model is CBOW. Gensim also has a directory of pre-trained embeddings, trained on several documents like Wikipedia pages, Twitter tweets, and Google news. For this dissertation, we will be using a pre-trained embedding based on the Google News corpus (Can be found here: https://github.com/mmihaltz/ word2vec-GoogleNews-vectors) word vectors model (3 million 300 dimension English word vectors). We can replace each word with the dense vector of 300 real values.

In order to come up with a feature descriptor for each email, we take the sum of each word vector and have a representation based on Word2Vec embedding for the whole email. Figure 3.2 illustrates this by showing a part of each word vector that is summed up at the end as a whole sentence with real values. The results will be a vector of the 300 elements (features) for each email, and in the end we end up with a matrix of $M \times 300$ where M is the number of emails we have in each dataset.

Figure 3.2: Converting each word into a vector with real values.

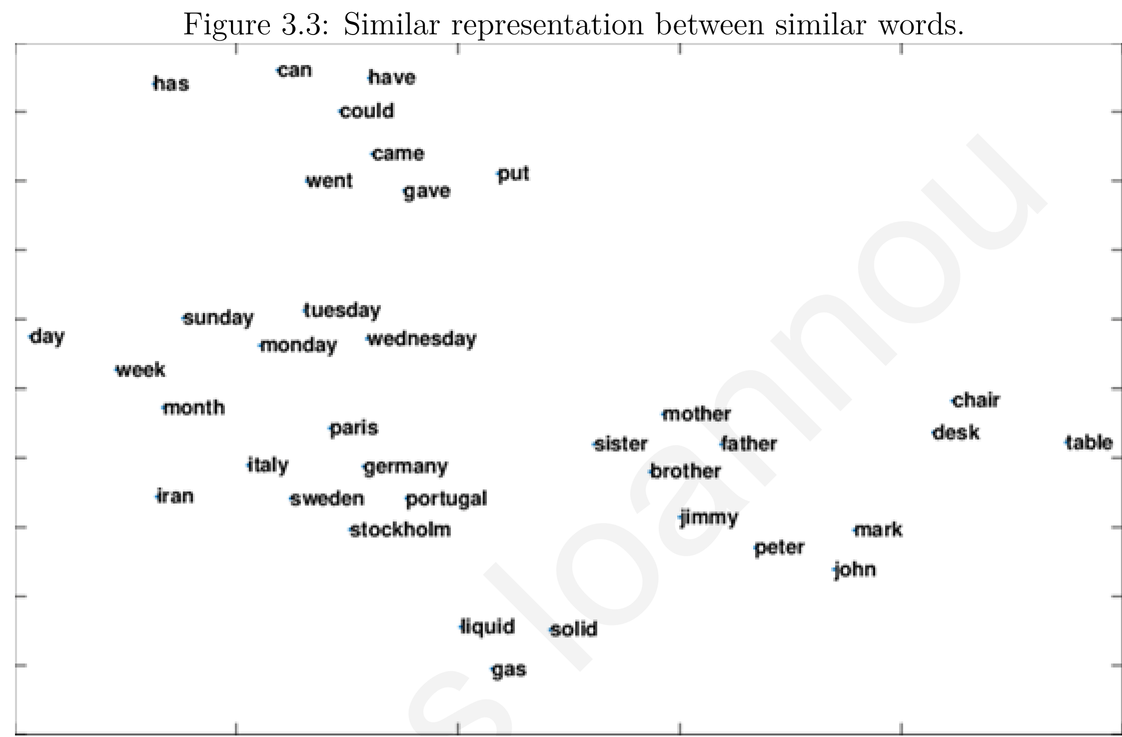


By converting each word into a vector of real values instead of just using the One-Hot Encoding, we attain a similar representation between similar words. So a word

Kypros Ioannou

that has a similar context with another will have a vector that will roughly point towards the same direction. An example of that can be seen in Figure 3.3. We have a 2-dimensional word embedding where “Sunday” has more similar values to other weekdays. Another example is the word “sister”, with similar values to other family words like "mother" and "father".

Figure 3.3: Similar representation between similar words.



3.4 Network Implementation

For this dissertation, we use a Convolutional Neural Network (CNN) with the Subsampled Hessian Network (SHN) method, which was implemented first in Matlab and then was converted to Python by Wang et al. [49]. The source for their code can be found here: [https://github.com/ cjlin1/simpleNN]. For the present implementation, we use the version in Python since we are more familiar with this programming language. The Python implementation uses Tensorflow [10]. Tensorflow is a machine learning framework, and the whole process is somewhat different from the original Matlab implementation.

One of the main problems of Newton is memory consumption due to the Hessian matrix. In order to solve the memory problem of the Newton method, the SHN technique uses a subset S of the train data to obtain the subsampled Gauss-Newton matrix that does what we already described in Hessian Free Optimisation (HFO) it

Kypros Ioannou

approximates the Hessian Matrix. As we increase the data, we increase the memory that we need to use to calculate the Hessian, thus by approximating it we reduce memory usage. Also, this technique reduces the execution time per iteration, but a side-effect is that we have a slightly less accurate direction.

As we have already mentioned, the initial implementation by Wang et al. [49] was in Matlab and because of that, the input dataset uses a Matlab format (.mat). After the code is converted to Python, the same type of dataset is used. The input file must contain a y variable (of size $N \times 1$), which includes all the target class labels, and a z variable (of size $N \times M$), which includes all the features.

Leontiou [29] created a script for his dissertation about Protein in order to convert his data into .mat format. So, instead of changing the code of Leontiou in order to convert the data into .mat format, we change the format the model is able to use. Anyone could load a file from a CSV file as it is the most frequently used format for a dataset, at least for the case of Python, and with a few lines of code and minor changes the dataset is ready to be fed into the model.

Furthermore, the implementation was modified so that we can execute it in a Jupyter Notebook Environment [26]. As mentioned in the previous paragraph, we need minor changes for each type of $2D$ problem i.e. different dataset. The reason for that is that we load the CSV file in a DataFrame which is a $2D$ labelled data structure with columns and rows. The minor change we have for each new dataset is specifying which column is the class. All the necessary scripts, programs, data files and instructions were uploaded in that repository, which can be found here: [https://gitlab.com/kypros_ioann/spam-filtering-with-cnn-hfo/-/tree/main/].

Lastly, we make a few modifications to the model's architecture since the initial version is not the best structure for the problem at hand. All the experiments that we are going to present in the next chapter run on a local machine with the following specifications:

- CPU: Intel Core i7 -7nd Generation
- 16 GB of Ram
- GPU: Nvidia GFORCE GTX 1070 Max-Q with 8GB

3.5 Metrics

Before we start talking about the results of this thesis implementation, it is vital to mention what metrics did we use in order to compare our model (Convolutional

Kypros Ioannou

Neural Network (CNN) with Hessian-Free Optimization (HFO)) with CNN with Gradient Descent (GD) and also to the original techniques from where we got the datasets.

The list of what we are going to compare using CNN/HFO and CNN/GD can be seen below.

- Accuracy
- Runtime to train the model
- False Positives

First, we compare the accuracy and validation accuracy for each model of how many emails have been classified correctly, either as spam or ham (legit). By validation accuracy, we mean that we split the datasets into a two-piece before training our model. We allocate 80% of the data for training and the other 20% in order to calculate model accuracy in data it has never seen before. For the Runtime, we use a library called "ipython-autotime" that can calculate the Runtime of each execution we want. Furthermore, for False Positives, we use the library "sklearn" in order to let us use Confusion Matrix [46] to calculate not only the False Positives, but also True Positives, True Negatives, and False Negatives. Lastly, we mention that we are also going to compare CNN/HFO with Metsis et al. work [34] where they use five different types of Naïve Bayes. Metsis et al. [34] use the Recall [47] technique for Spam and Ham emails separately which will be explained in the following subsection.

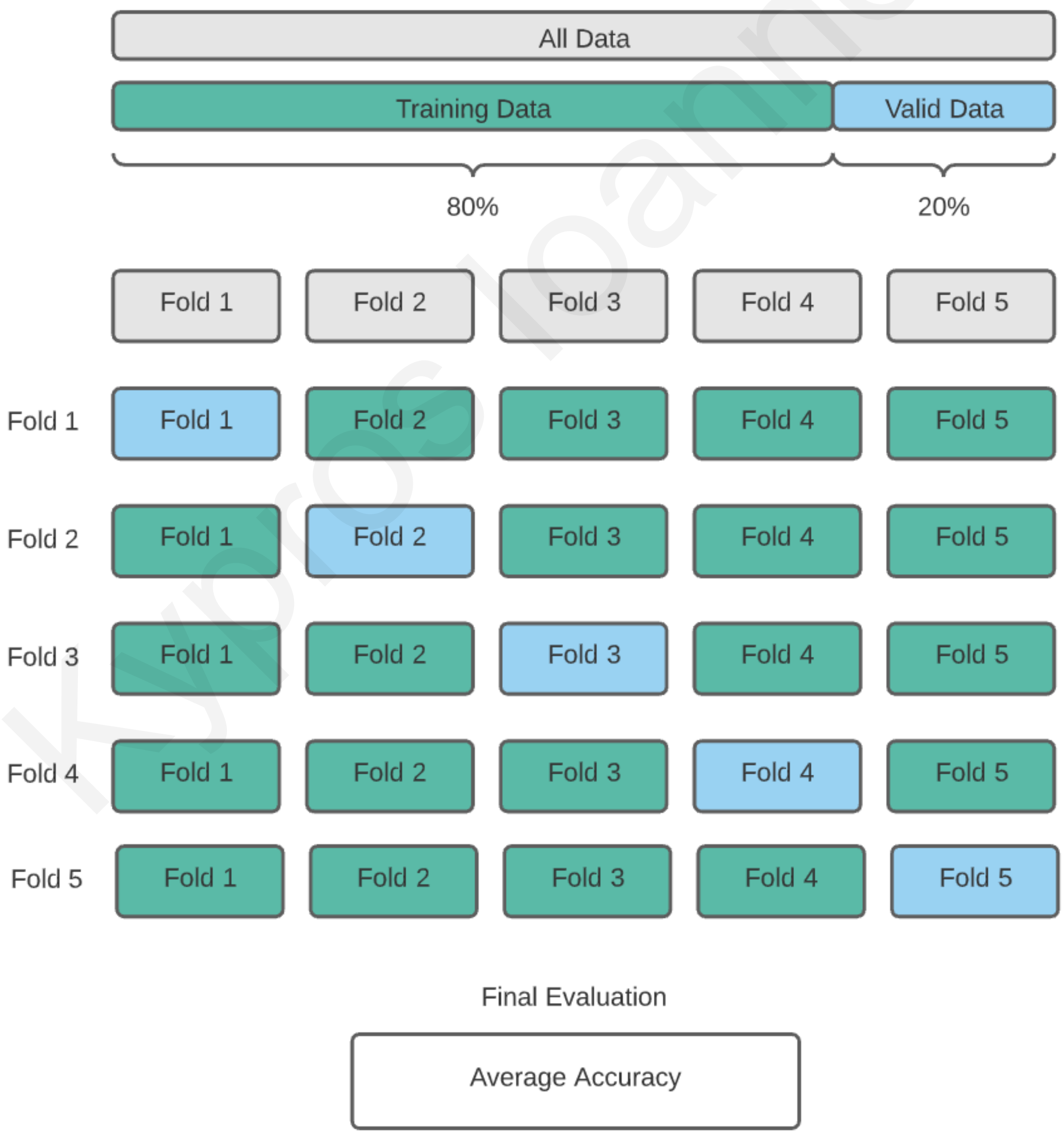
3.5.1 Training/Testing Set and Cross Validation

To evaluate each model's accuracy for each dataset, we use Cross Validation (CV) [42]. Cross-Validation is a technique where we split the dataset into x number of pieces, and we use one of the pieces each time for validation in order to understand how each model behaves with never-seen-before data. We can see in Figure 3.4 an example of Cross-Validation that is similar to what we do in this thesis. The dataset is split into five pieces with an equal number of data and we merge four of them each time, creating a "new dataset". The left one is used to validate the model. Each time the validation piece is different, we re-run the model as many times as the number of pieces. The CV may sound time-consuming, but with this technique we can better evaluate the model and understand whether the high accuracy we get is the result of luck or we are doing something right. We apply this technique for each dataset and then calculate the average accuracy of each model as well as the average validation

Kypros Ioannou

accuracy. To create those five pieces for the dataset, we use the sklearn library once more and the function called "KFold". Because this technique is time-consuming and every model is run on a private computer, it is tough to do Cross-Validation with ten pieces. That is because the private computer is heavily burdened by all the experiments we already did, since cross-validation is not the only experiment we need. However, we also need to find the Hyperparameter for the model to achieve the highest accuracy or the highest Recall. For more info on the Hyperparameters, we shall elaborate on it further in Chapter 4. We use cross-validation to test the Convolutional Neural Network (CNN) with Hessian-Free Optimisation and also the CNN with Gradient Descent.

Figure 3.4: Cross Validation. We split our dataset into 5 pieces and each time we use a different piece in order to validate our model accuracy.



Kyros Ioannou

3.5.2 Confusion Matrix

A confusion matrix [46] is a two-dimensional table that we use to visualise the performance of an algorithm. A confusion matrix is mainly used in supervised learning, whereas for unsupervised learning we use a matching matrix. Figure 3.5 visualises how a confusion matrix looks like in a binary classification problem such as the one pertaining to the current thesis. On the left side of the table is the actual (real) values, and on the bottom the predicted values from the model. As we can see from each cell, we have TP, TN, FP, FN, defined as follows for the current problem:

TP is the True Positive that indicates all the emails that are ham emails that are predicted correctly. Then we have the FP, which is the False Positive, and those are all the emails that are ham emails which are misclassified as spam emails. We have the FN, the False Negative, with all the spam emails which the model misclassifies as ham emails. Lastly we have the TN, the True Negative, with all the emails that are correctly classified as spam.

We can compute the accuracy of this model using Confusion Matrix, but also the Missclasification Rate in regards to how many emails in this problem are misclassified. Another rate we can compute is the Sensitivity (True Positive) rate or Recall, as Meltis el al mention and proceed to do in their paper [34]. Moreover, another rate is Specificity (True Negative Rate) which is the opposite version of Sensitivity. Instead of calculating how many times the model classifies ham emails correctly, we calculate how many times the model classifies spam emails correctly (Specificity). We will show in the following section how each rate is computed.

Kypros Ioannou

Figure 3.5: Confusion Matrix.

3.5.3 Rates computed by Confusion Matrix

To calculate our model's accuracy, we use Equation 3.1, where we divide the True Positives and True Negatives emails with the total number of emails.

$$Accuracy = \frac{(TP + TN)}{TotalNumberofEmails} \tag{3.1}$$

The next rate is the Misclassification Rate, as seen in Equation 3.2. We do the opposite of the accuracy where we add all the misclassified (False Positives and False Negatives) emails, and then divide them with the total number of emails:

$$MisclassificationRate = \frac{(FP + FN)}{TotalNumberofEmails} \tag{3.2}$$

For Recall or Sensitivity we use Equation 3.3 and we divide the False Positives with the True Positives:

$$Sensitivity(Recall) = \frac{FP}{TP} \tag{3.3}$$

Kypros Ioannou

Finally, we calculate the Specificity using Equation 3.4, by dividing the False Negatives with True Negatives:

$$Specificity = \frac{FN}{TN} \tag{3.4}$$

It is necessary to mention that in Metsis' work [34], Specificity and Sensitivity are referred as Spam Recall and Ham Recall respectively. In the next chapter, the model results shall be presented in detail. We will also compare it with Metsis's work and with Convolutional Neural Network with Gradient Descent.

Kypros Ioannou

Chapter 4

Results and Discussion

4.1 Fine-tuning of Hyperparameters

One of the most important things we have to do after properly preprocessing the data is choosing optimal parameters for the network. We perform many experiments changing only one parameter at a time so that we determine whether that parameter improves or worsens the accuracy of the model. For finding all the optimal hyperparameters, we experiment using fold four from the sixth dataset. The main reason for experimenting with fold four is that it has the lowest accuracy of all the folds from all datasets. In turn, the logic behind that decision is that if we can increase the accuracy of the lowest fold, we may find the optimal parameters to further increase the accuracy of other datasets.

The original number of iterations (epochs) we experiment with is 1000, the reasoning being that we want to let the network stabilise before we stop it. The model's accuracy increases radically from the first 10 epochs that were not stable, meaning that with each epoch the accuracy was slightly different from the previous iteration. We already mentioned that Word2Vec is the tool that we are going to use for data representation. Word2Vec extracts 300 values from each email, hence the input dimensions of CNN will be $15 \times 20 \times 1$.

The first parameter that we start experimenting with is the GNsize which is the number of samples used in the subsampled Gauss-Newton matrix. We use five different values starting from 50 ascending to 1024. We can see in Table 4.1 that having GNsize equal to 512, we achieve the highest accuracy of them all. Here we need to mention that we at first use the same Convolutional Neural Structure as used by Wang et al. [49]. We have 2 Convolutional layers with 64 filters, and also 2 Pooling layers.

Kypros Ioannou

Table 4.1: Sixth dataset accuracy when we change the GNsize for fold 4.

GNsize	C	BSize	Epochs	Num Conv	Num Filters	Num Pooling	Accuracy
50	0.5	32	1000	2	64	2	92.285%
100	0.5	32	1000	2	64	2	92.385%
200	0.5	32	1000	2	64	2	93.664%
512	0.5	32	1000	2	64	2	96.192%
1024	0.5	32	1000	2	64	2	95.491%

The next parameter that we choose is the regularisation parameter (C value), following the same process. We can see in Table 4.2 that we have the highest accuracy when C is equal to 0.5. Lastly, we have to choose the batch size, but there is no significant change, therefore we set it to 32.

Table 4.2: Sixth dataset accuracy when we change the C hyperparameter for fold 4.

GNsize	C	BSize	Epochs	Num Conv	Num Filters	Num Pooling	Accuracy
512	0.01	32	1000	2	64	2	92.185%
512	0.1	32	1000	2	64	2	93.192%
512	0.3	32	1000	2	64	2	94.293%
512	0.5	32	1000	2	64	2	96.444%
512	1	32	1000	2	64	2	95.921%

After we find the hyperparameter for the Subsampled Hessian Newton, the next step is to alter the structure of the CNN. We choose three different tasks toperform in order to conclude the most optimal parameter for this model.

- Number of Convolutional Layers
- Number of Filters
- Number of Pooling Layers

Convolutional Layers

We will only talk about the most important experiments, but for the sake of completion, most of the experiment tables can be found in the Appendix. As previously said, we first test our model with 2 Convolutional layers and 2 Pooling layers. We

Kypros Ioannou

proceed by choosing a different number of layers; if we look at Table 4.3 we can see that aside from 2×2 (2-Conv and 2 Pooling), we additionally use 3×3 and 1×1. In addition, we try to use only one convolutional layer without any pooling, as Chatzimiltis [17] uses in his thesis for a different problem, but that did not improve in terms of accuracy. We observe that, as long as we increase the complexity of the network by adding more layers, the model's accuracy is decreased. Therefore we choose to have a model with 1 Convolutional Layer and 1 Pooling Layer as a final model.

Table 4.3: Sixth dataset accuracy using 3 different number of Convolutional and Pooling Layers.

GNsize	C	BSize	Epochs	Num Conv	Num Filters	Num Pooling	Accuracy
512	0.5	32	1000	1	64	1	98.193%
512	0.5	32	1000	2	64	2	96.192%
512	0.5	32	1000	3	64	3	92.120%

Filters of the Convolutional Layer

The last hyperparameter which we will talk about is the number of filters that the model will have for each convolutional layer. In Table 4.4 we use three different numbers of filters that are mainly used, finding out that the model that achieves the highest accuracy is the one using 128 filters. We conclude that this model has the best result using only one convolutional layer from the previous section with 128 filters and one Pooling Layer.

Table 4.4: Sixth dataset accuracy using 3 different number of filters fo the Convolutional Layer.

GNsize	C	BSize	Epochs	Num Conv	Num Filters	Num Pooling	Accuracy
512	0.5	32	1000	1	32	1	98.193%
512	0.5	32	1000	1	64	1	98.233%
512	0.5	32	1000	1	128	1	98.413%

Our Convolutional Neural Networks Architecture

Our Convolutional Neural Network has an input of 15x20x1. Then the dimensions after passing through the first convolutional layer become 15x20x128 because we

Kypros Ioannou

set the number of filters to 128. The next layer which is the pooling layer will decrease the height and width dimension of the input, while keeping the features. The dimensions of the pooling layer are 7x10x128. The architecture of the model can be seen in Figure 4.1. Lastly, the output layer will either predict 1 for Spam Emails or 0 for Ham Emails (we have a binary classification model). In the following six sections, we will showcase the accuracy of each fold for each dataset, while we will further show how well each model behaves for Spam and Ham (Legit) emails.

Figure 4.1: Our Convolutional Neural Networks Architecture.

Kypros Ioannou

4.2 Cross-Validation Results with First Dataset

We begin with the first dataset, where we have six columns starting with the model's accuracy. The second column corresponds to the validation accuracy which is the model's accuracy for data it has never seen before. After those two columns, we have the Sensitivity (Ham Recall) and Specificity (Spam Recall) about which we have already talked in the Metrics section. Because in Meltis et al. [34] they call them Ham Recall and Spam Recall and we are eventually going to compare this current model to their work, we adopt the same names to prevent any confusion. Validation Recall refers to the data that the model has never seen before. The remaining two columns are the data for which that the model was trained.

We can observe from Table 4.5 that we cannot significantly distinguish between accuracy/validation accuracy and the recall/validation recall; all five-folds have almost the same accuracy. We observe a very important thing though, as it can be seen in the confusion matrix: we achieve 100% Ham Recall for three out of five folds on data that has never been seen before. If we take the last row where we average each metric, we realize that we have an almost perfect model that behaves very well with the first dataset. The average accuracy of the model is 99%, and the average Recall is almost 100%. We have good spam emails, but the model does not behave as

perfectly as for the ham emails, albeit a 97% accuracy is still more than adequate. We believe that having lower Recall for spam emails is due to having fewer spam emails than ham emails (3673 Ham compared to 1500 Spam).

Table 4.5: Cross-Validation for the First Dataset. We measure the accuracy of the model and also Spam and Ham Recall.

Dataset 1	Accuracy	Validation Accuracy	Spam Recall	Ham Recall	Validation Spam Recall	Validation Ham Recall
Fold 1	99.073%	98.899%	96.87%	99.96%	96.44%	100%
Fold 2	99.149%	98.998%	97.22%	99.96%	97.41%	99.58%
Fold 3	99.149%	99.299%	97.24%	99.92%	97.64%	100%
Fold 4	99.174%	98.998%	97.33%	99.92%	96.97%	99.85%
Fold 5	99.123%	99.198%	97.10%	99.96%	97.19%	100%
Average	99.133%	99.078%	97.15%	99.99%	97.13%	99.88%

We will not show all the folds confusion matrices in any of the six experiment sections. Instead we will show the best fold for each dataset as well as the worst based on the False Positives. The rest can be seen in the Appendix.

We can see the confusion matrices for the best model in Figures 4.2 and 4.3, and the "worst" model in Figures 4.4 and 4.5. Each fold has almost the same amount of emails, but they do not have precisely the same number of spam and ham emails. Note that by "worst" we do not mean that the model behaves badly, but that it has a higher number of misclassified ham emails. The highest number of misclassified ham emails is merely 3.

Figure 4.2: Dataset 1 Fold 5 Train Confusion Matrix.(Best Model)

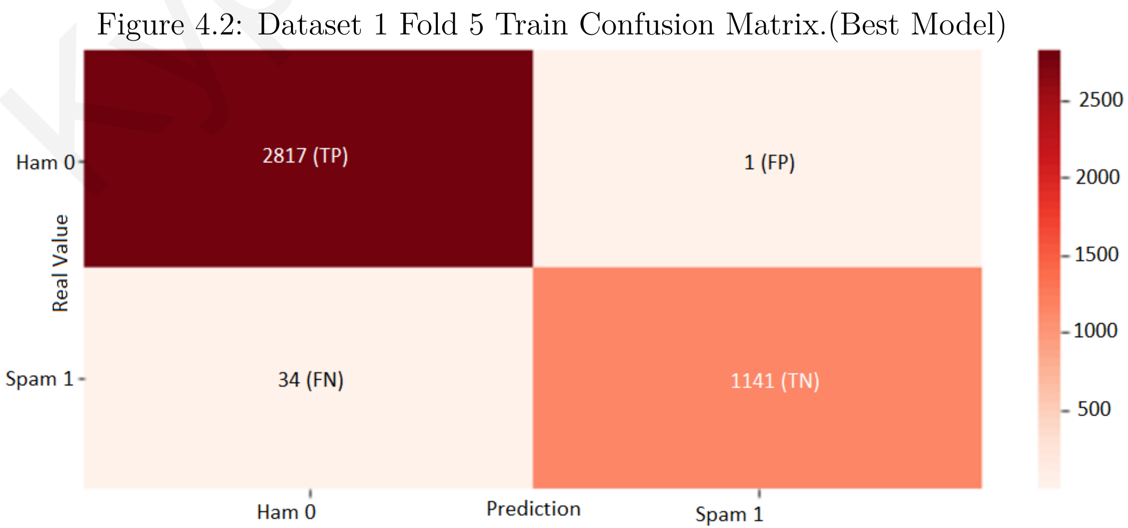


Figure 4.3: Dataset 1 Fold 5 Valid Confusion Matrix. (Best Model)

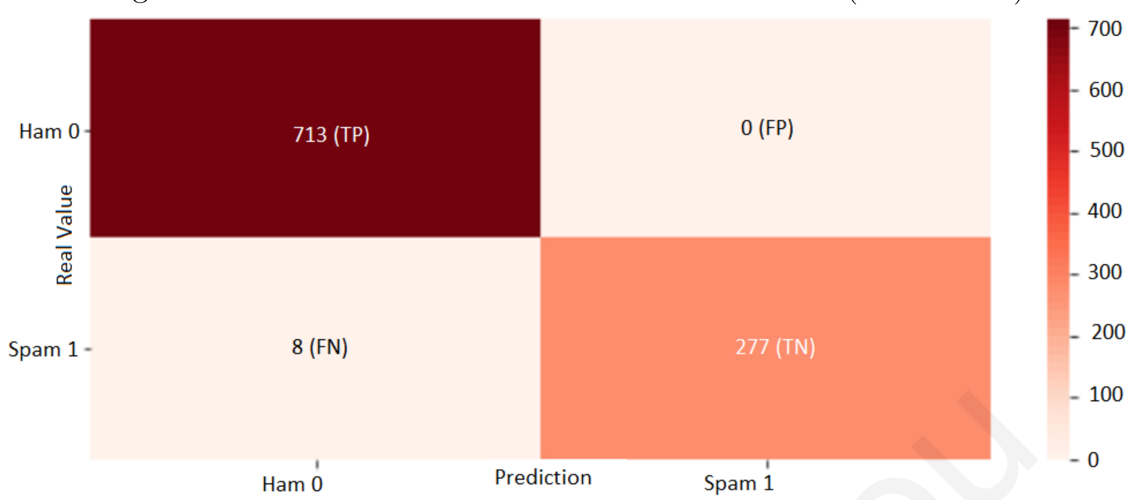
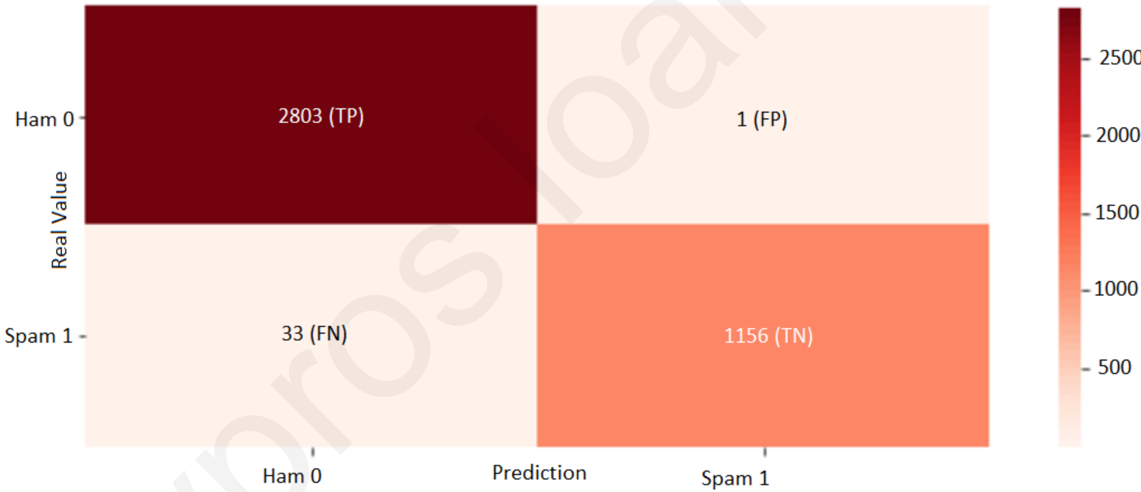
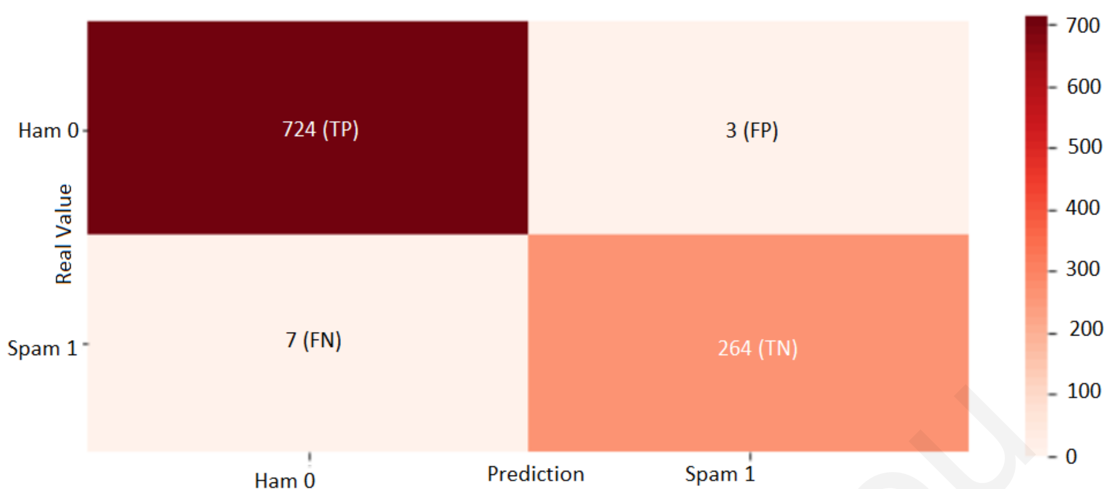


Figure 4.4: Dataset 1 Fold 2 Train Confusion Matrix. (Worst Model)



Kypros Ioannou

Figure 4.5: Dataset 1 Fold 2 Valid Confusion Matrix. (Worst Model)



4.3 Cross-Validation Results with Second Dataset

For the second dataset, we apply the same architecture and use the same hyperparameters. We performed cross-validation again for the second dataset, and in Table 4.6, we have the Accuracy/Validation Accuracy of the model and Spam and Ham Recall for both accuracies. The most important information that we can extract from Table 4.6 is that this model classified all the Ham email correctly in every fold. In addition, we score 99% accuracy/and validation accuracy for every fold except from Fold2, where we are just 0.030% below 99%.

Table 4.6: Cross-Validation for the Second Dataset. We measure the accuracy of the model and also Spam and Ham Recall.

Dataset 2	Accuracy	Validation Accuracy	Spam Recall	Ham Recall	Validation Spam Recall	Validation Ham Recall
Fold 1	99.055%	99.399%	96.34%	100%	97.59%	100%
Fold 2	99.270%	98.970%	97.15%	100%	96.01%	100%
Fold 3	99.098%	99.056%	96.44%	100%	96.48%	100%
Fold 4	99.077%	99.141%	96.41%	100%	96.59%	100%
Fold 5	99.206%	99.227%	96.91%	100%	96.95%	100%
Average	99.141%	99.158%	96.65%	100%	96.72%	100%

Because every confusion matrix is precisely the same for the False Positive for Dataset 2, we will show only the “worst” case where the model achieves 98.970%. Figures 4.6 and 4.7 show the confusion matrices for Dataset 2, where we can see

Kypros Ioannou

0 False Positives(FP), and 34 and 12 misclassified Spam emails for Accuracy and Validation Accuracy respectively. Once again, we have more ham emails than spam emails (more than twice the amount).

Figure 4.6: Dataset 2 Fold 2 Train Confusion Matrix(Worst Model) where we have 0 False Positives and 34 False Negatives

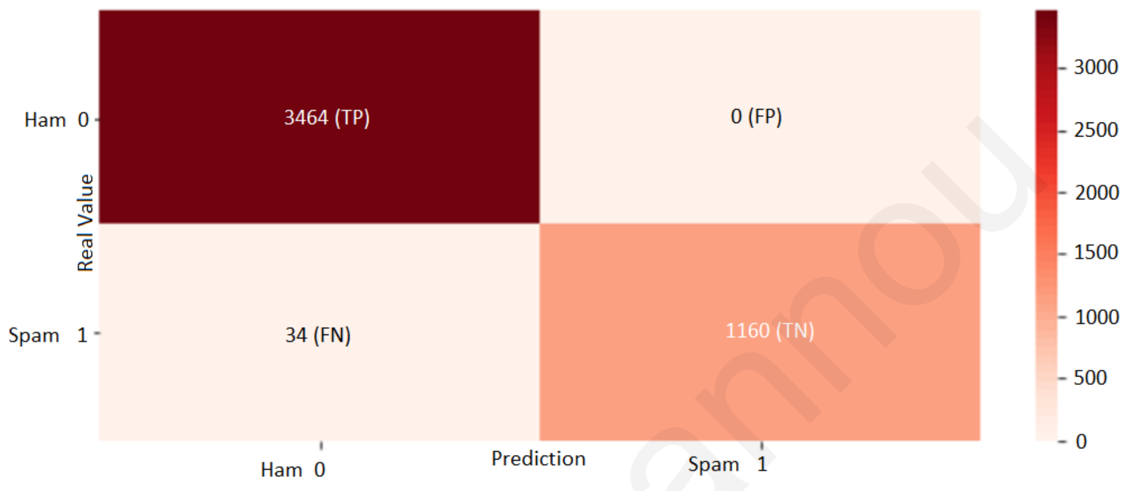
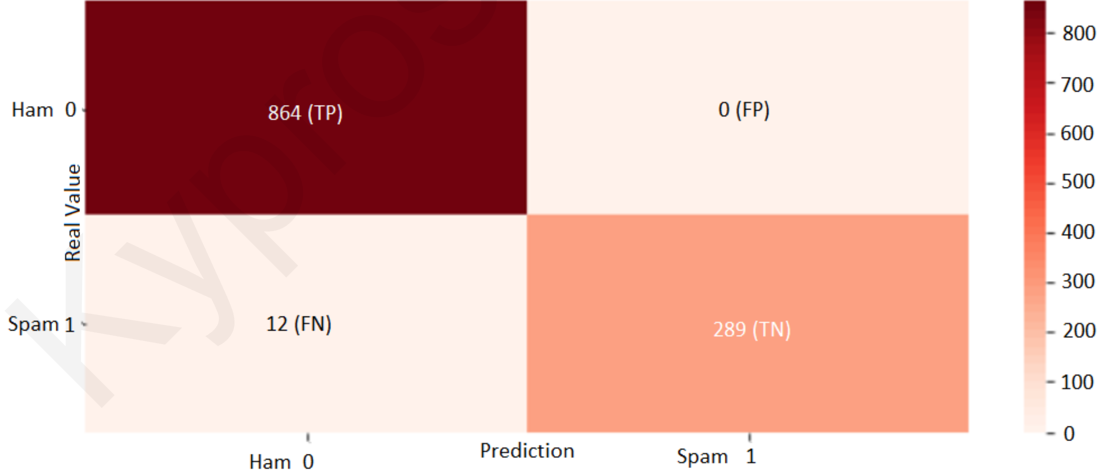


Figure 4.7: Dataset 2 Fold 2 Validation Confusion Matrix(Worst Model) have we scored 0 False Positives and 12 False Negatives.



4.4 Cross-Validation Results with Third Dataset

Table 4.7 shows the accuracy and Recall of the third dataset. One primary difference we have from the other two datasets is that there is not much difference between the Spam and Ham Recalls. In the previous two datasets, the Ham Recall was 2-3%

higher than the Spam Recall most of the time. With the third dataset, we achieve 99% accuracy and validation accuracy for all the folds, and the difference between spam and ham recall is for the most part less than 1%. With the exception of Fold 4 where the Validation Spam Recall is just 0.12 below the 99%, all the others have Recall higher than 99%.

Table 4.7: Cross-Validation for the Third Dataset. We measure the accuracy of the model and also Spam and Ham Recall.

Dataset 3	Accuracy	Validation Accuracy	Spam Recall	Ham Recall	Validation Spam Recall	Validation Ham Recall
Fold 1	99.787%	99.905%	99.33%	99.96%	99.65%	100%
Fold 2	99.763%	99.905%	99.15%	100%	99.68%	100%
Fold 3	99.834%	99.621%	99.66%	99.90%	99.02%	99.86%
Fold 4	99.763%	99.620%	99.26%	99.96%	98.88%	99.87%
Fold 5	99.787%	99.810%	99.40%	99.93%	99.68%	99.86%
Average	99.787%	99.772%	99.36%	99.95%	99.38%	99.91%

Fold 4 had the lowest Validation Spam Recall (98.88%), but we will show in confusion matrices of Fold 3 with higher misclassified ham emails as we already did for the second dataset. Figure 4.8 shows the confusion matrix of the model where the model misclassified 3 Ham and 4 Spam emails. Also, Figure 4.9 presents the model's confusion matrix for data which it had never seen before where the model misclassified 1 Ham email and 3 Spam emails.

Figure 4.8: Dataset 3 Fold 3 Train Confusion Matrix(Worst Model) where we have 3 False Positives and 4 False Negatives.

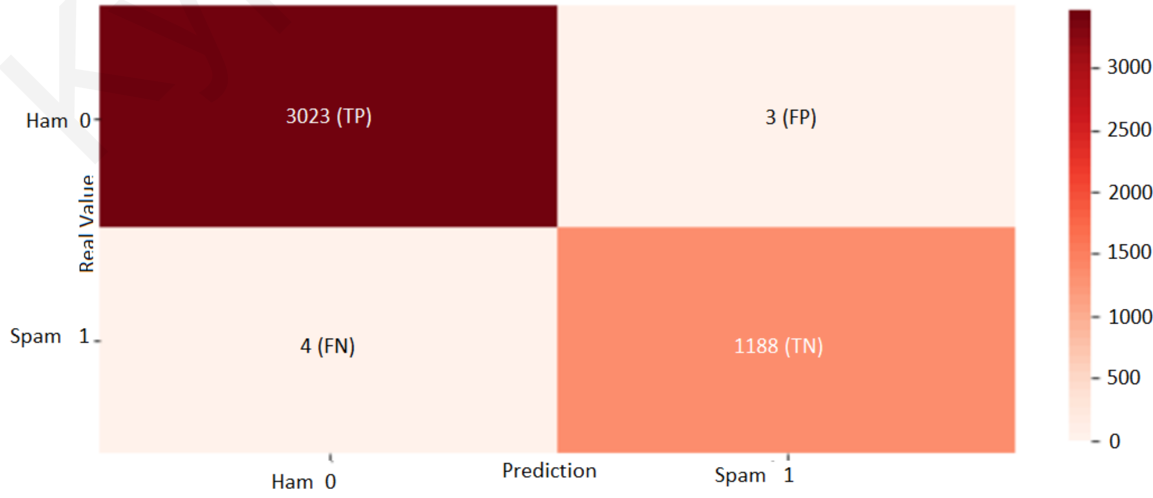
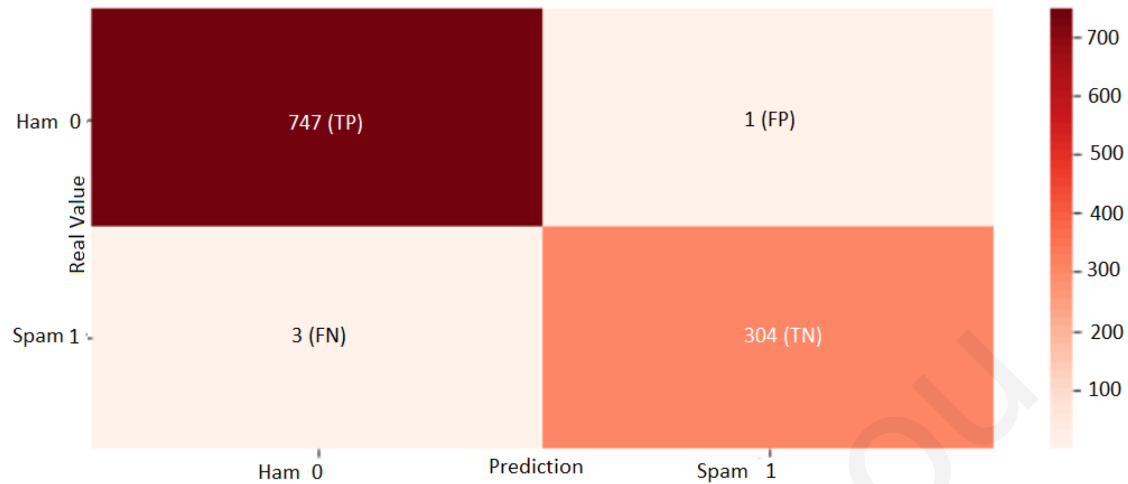


Figure 4.9: Dataset 3 Fold 3 Validation Confusion Matrix(Worst Model) where we have 1 False Positives and 3 False Negatives.



4.5 Cross-Validation Results with Fourth Dataset

In the first three datasets, we had a higher Ham Recall than Spam recall in every fold. For Dataset 4, we have the opposite as we clearly see that, aside from Fold 3 for which the validation Ham Recall is 100% , all the other folds have higher spam than ham Recall. The results (Table 4.8) in this dataset are as good as the previous three datasets. However, we may have just a bit lower Ham Recall than Spam due to the number of spam emails being three times larger than that of ham emails (1500 Ham compared to 4500 Spam emails).

Table 4.8: Cross-Validation for the Fourth Dataset. We measure the accuracy of the model and also Spam and Ham Recall.

Dataset 4	Accuracy	Validation Accuracy	Spam Recall	Ham Recall	Validation Spam Recall	Validation Ham Recall
Fold 1	99.251%	99.316%	99.51%	98.49%	99.53%	98.68%
Fold 2	99.337%	98.973%	99.54%	98.72%	99.41%	97.79%
Fold 3	99.102%	99.829%	99.45%	98.11%	99.77%	100%
Fold 4	99.230%	98.888%	99.54%	98.32%	99.42%	97.37%
Fold 5	99.380%	99.230%	99.62%	98.67%	99.54%	98.28%
Average	99.26%	99.24%	99.532%	98.65%	99.53%	98.42%

Dataset 4 has the most misclassified Ham emails.In Figures 4.10 and 4.11, we present the confusion matrix for the lowest fold, while Figures 4.12 and 4.13 show the best

Kypros Ioannou

fold for this dataset, which is fold 3. Starting from the fold that scored the lowest, we had 20 misclassified ham, and 16 misclassified spam emails for the model. When we tested it to no-train data, we had eight misclassified Ham and 5 Spam emails. On the other hand, the best fold (Fold 3) for dataset 4 has three more misclassified Ham emails for train emails, but had 0 misclassified Ham emails when we test it for new data. Fold three had a nearly perfect prediction with 99.829% accuracy for new data, boasting just two wrongly classified spam emails.

Figure 4.10: Dataset 4 Fold 4 Train Confusion Matrix(Worst Model) where we have 20 False Positives and 16 False Negatives.

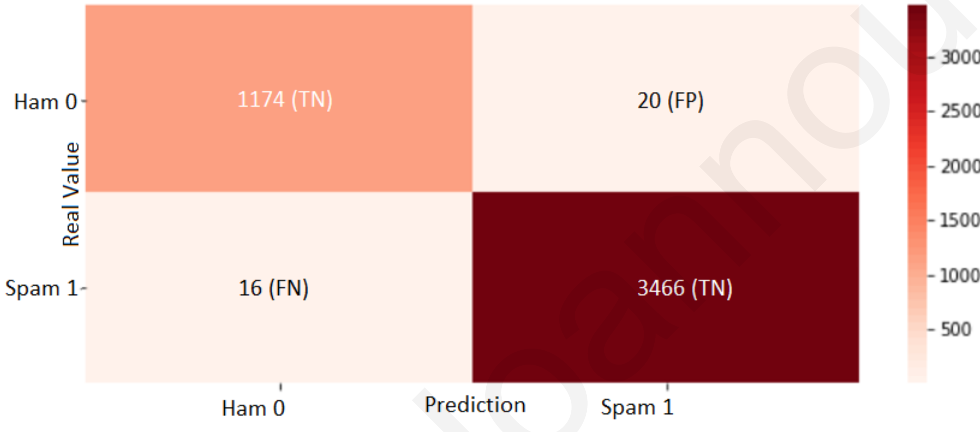


Figure 4.11: Dataset 4 Fold 4 Validation Confusion Matrix(Worst Model) where we have 8 False Positives and 5 False Negatives.

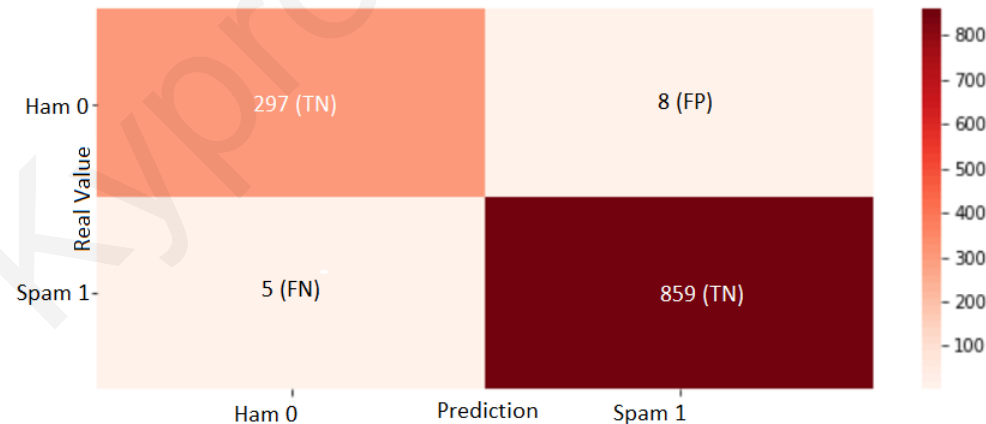


Figure 4.12: Dataset 4 Fold 3 Train Confusion Matrix(Best Model) where we have 23 False Positives and 19 False Negatives.

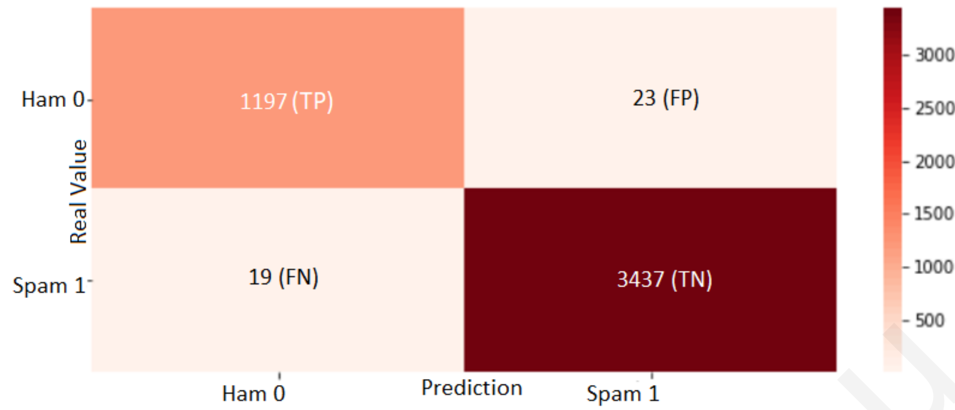
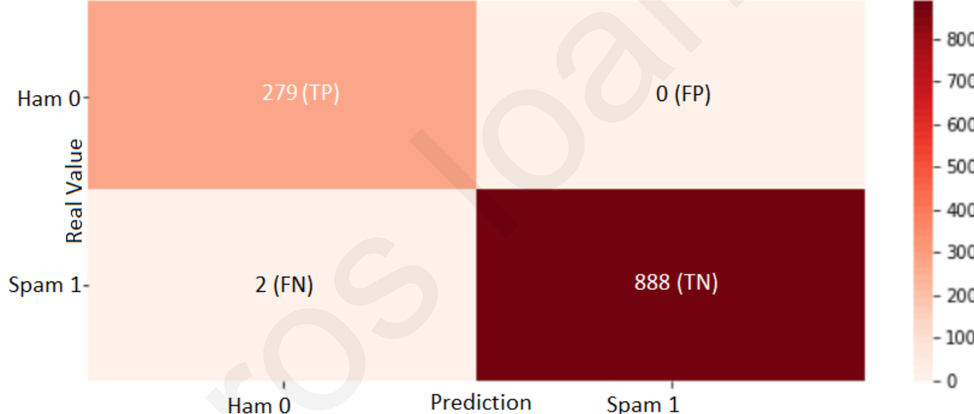


Figure 4.13: Dataset 4 Fold 3 Validation Confusion Matrix(Best Model) where we have 0 False Positives and 2 False Negatives.



4.6 Cross-Validation Results with Fifth Dataset

In Dataset 5 we have once more achieved perfect prediction for all the ham emails. The fold with the lowest accuracy and Spam Recall was fold 3 with an accuracy on new data of 98.630%, additionally having the lowest Spam Recall at 97.89% . Table 4.9 shows all the results from all the folds for Dataset 5, whereas we only analyze fold 3 in Figures 4.14 and 4.15. All the others confusion matrices, as mentioned already, can be found in the Appendix. Finally, in dataset five we have yet again more Spam than Ham emails (two times as many), however we have higher Ham Recall than Spam Recall.

Kypros Ioannou

Table 4.9: Cross-Validation for the Fifth Dataset. We measure the accuracy of the model and also Spam and Ham Recall.

Dataset 5	Accuracy	Validation Accuracy	Spam Recall	Ham Recall	Validation Spam Recall	Validation Ham Recall
Fold 1	98.875%	99.022%	98.42%	100%	98.67%	100%
Fold 2	98.997%	99.217%	98.69%	100%	98.90%	100%
Fold 3	98.484%	98.630%	97.89%	100%	98.06%	100%
Fold 4	98.973%	98.826%	98.56%	100%	98.37%	100%
Fold 5	98.851%	99.119%	98.40%	100%	98.89%	100%
Average	98.836%	98.762%	98.39%	100%	98.57%	100%

Figure 4.14: Dataset 5 Fold 3 Validation Confusion Matrix(Worst Model) where we have 0 False Positives and 62 False Negatives.

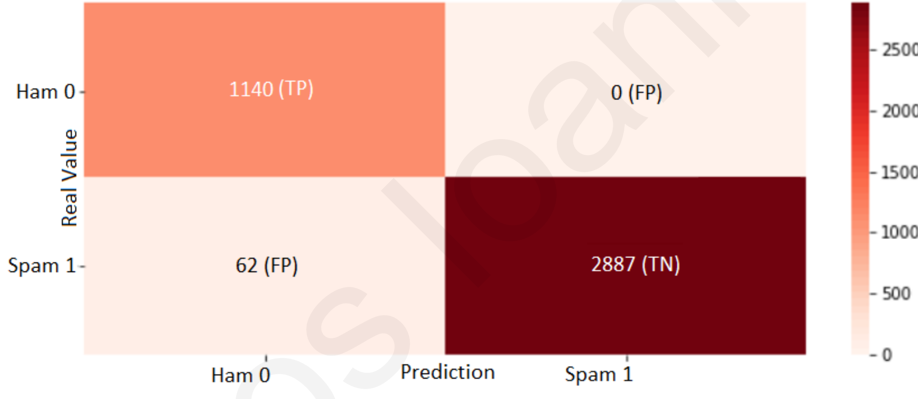
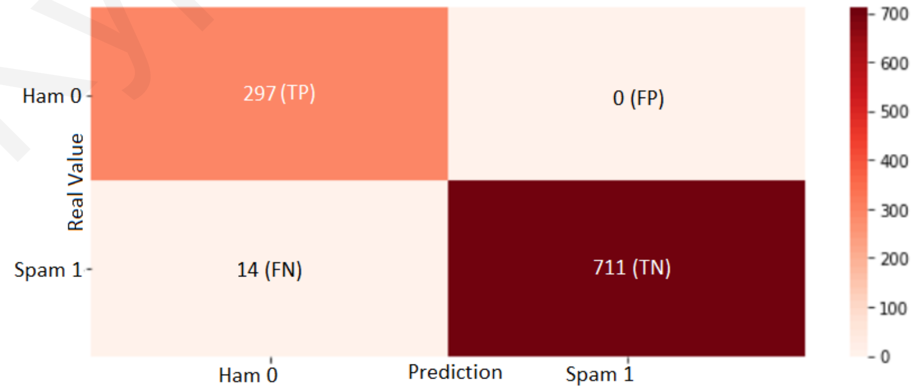


Figure 4.15: Dataset 5 Fold 3 Train Confusion Matrix(Worst Model) where we have 0 False Positives and 14 False Negatives.



Kypros Ioannou

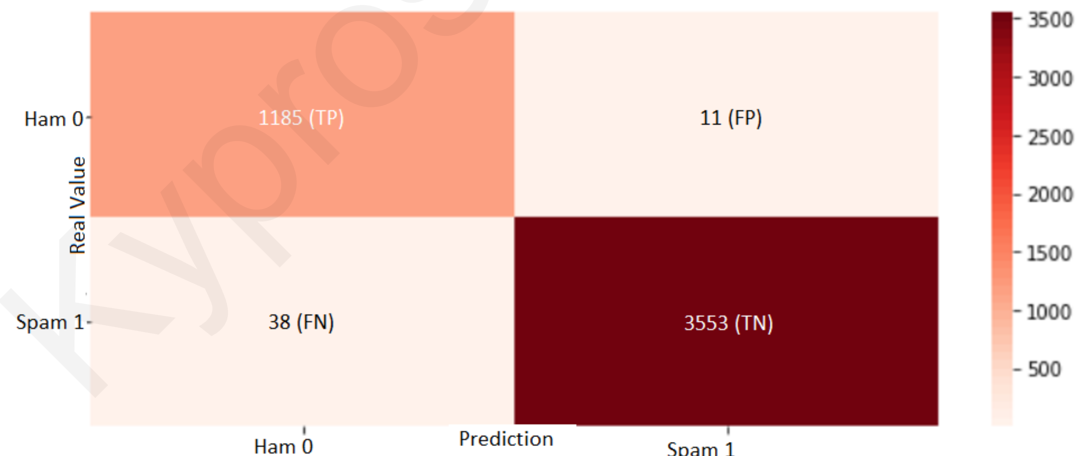
4.7 Cross-Validation Results with Sixth Dataset

We proceed with the last dataset, with Table 4.10 listing all the results for each of the five folds. We have more spam than ham emails (1500 Ham and 4500 Spam emails) as per usual. Dataset 6 is the only one where we did not score 100% on any of the folds for Spam Recall. In Figures 4.16 and 4.17, we can see the confusion matrix of the dataset six with the lowest Ham Recall score, which is Fold 2.

Table 4.10: Cross-Validation for the Sixth Dataset. We measure the accuracy of the model and also Spam and Ham Recall.

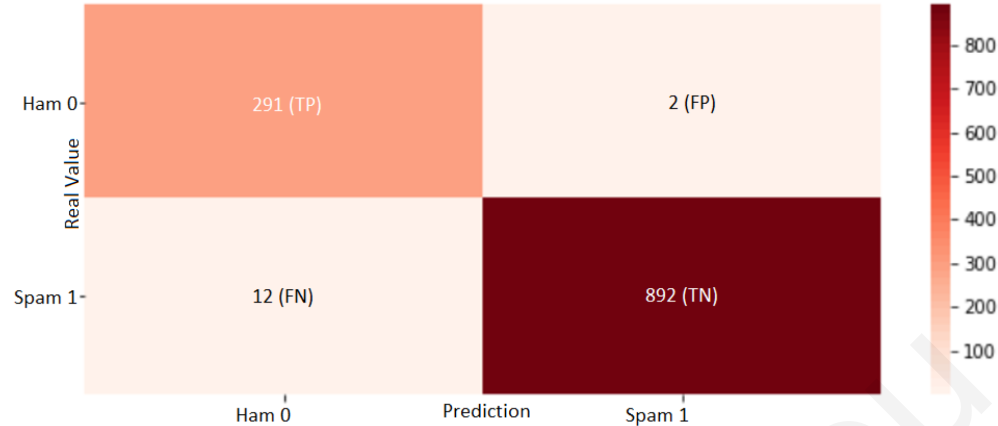
Dataset 6	Accuracy	Validation Accuracy	Spam Recall	Ham Recall	Validation Spam Recall	Validation Ham Recall
Fold 1	98.475%	99.248%	98.19%	99.32%	99.20%	99.35%
Fold 2	98.976%	98.830%	98.94%	99.08%	98.67%	99.31%
Fold 3	98.663%	98.496%	98.38%	99.49%	98.10%	99.66%
Fold 4	99.081%	98.413%	98.94%	99.49%	98.34%	98.63%
Fold 5	98.580%	99.247%	98.41%	99.08%	99.22%	99.31%
Average	98.755%	98.846%	98.52%	99.09%	98.70%	99.25%

Figure 4.16: Dataset 6 Fold 2 Validation Confusion Matrix(Worst Model) where we have 11 False Positives and 38 False Negatives.



Kypros loannou

Figure 4.17: Dataset 6 Fold 2 Validation Confusion Matrix(Worst Model) where we have 2 False Positives and 12 False Negatives.



4.8 Testing best model with the other 5 Datasets

Having produced very good results for each dataset and for each fold in the previous sections, we decide to test the highest-performing model (model 3) with the other 5 datasets. What we do is to use said model that was trained with a fold from Dataset 3, to test each fold from every other dataset. Then we calculate the average validation accuracy for each dataset. In Table 4.11 we present the average accuracy for each dataset using CNNs/HFO with Dataset 3.

Table 4.11: Average accuracy for all the dataset(Dataset 3 is used to train CNNs/HFO).

Dataset	Average Accurracy
First	61.75%
Second	45.65%
Fourth	86.71%
Fifth	84.34%
Sixth	95.60%

We can see that the first dataset has an average accuracy of 61.75%, while the second dataset has the lowest accuracy of all the 5 five dataset (45.65%). For dataset 4 it increases, reaching an accuracy of 86.71%. The accuracy of the fifth dataset is slightly lower than the fourth dataset (84.34%). With the last dataset we have an accuracy of 95.60% which is was the highest out of all of the 5 datasets tested.

Kypros Ioannou

Just as we mentioned above, the second dataset had the lowest accuracy out of all the datasets using the dataset 3 model. Taking a look at Table 3.1, Dataset 2 used Enron emails for Ham emails and Honeypot and Spam Assassins emails for Spam emails. We have the same thing in Dataset 5, the only difference being that we have twice as many spam emails compared to ham emails. Dataset 2 exhibits the opposite by having 2.5 times more ham emails than spam emails. We speculate that the model may have underperformed because the ham emails of Dataset 2 are not so similar to the ones we used to train our model. The same thing applies to Dataset 1 for which we had the second lowest accuracy. Metsis et al. [34] used emails from a variety of Enron Company employees, hence we believe that the spam emails will have more similarities between them as opposed to Ham emails. That is because spam emails are not tailored specifically for each person or target a specific group of people, thus they will be very similar with others. By contrast, ham emails differ based on the writing style and specific subject pertaining to the sender and recipient in each case. If the lower accuracy stems from that, then it is an indication that the model used does not generalize as well as we initially thought.

4.9 Comparison Between Gradient Descent and Hessian Free Optimisation

In this section, we are going to compare our Convolutional Neural Network (CNN) with Hessian Free Optimization (HFO) model with the CNN with Gradient Descent (GD) model. The difference between the comparison that we are going to do for the last section of Chapter 4 is that we run this model rather than compare our results based on others researchers work as we do with original paper data [34].

For CNN/GDwill also did Cross-Validation. Table 4.12 shows the average Accuracyand Ham/Spam Recall for CNN/GD, and Table 4.13 is for CNN with Hessian Free Optimisation. The CNN with Gradient Descent is not able to classify most of the emails correctly. It is essential to mention that the average accuracy here is the accuracy of each model based on data that has never been seen before. We already show how our model behaves, and we are not going to show them again. Here we simplify the tables by showing only the accuracy and the spam and ham Recall.

Looking at the last two columns from Table 4.12, it is obvious that CNN/GD classifies all emails either as spam or ham. After checking the number of ham and spam emails we have in each dataset, we conclude that the model classified all of their emails as spam when the dataset has more spam emails than ham. The same is

Kypros Ioannou

applied when we have more ham emails than spam.

Table 4.12: Average Accuracy and Spam/Ham Recall from Cross-Validation for CNN/Gradient Descent Model

Dataset Average	Accuracy	Spam Recall	Ham Recall
Dataset 1	71.44%	0%	100%
Dataset 2	74.57%	0%	100%
Dataset 3	72.51%	0%	100%
Dataset 4	76.13%	100%	0%
Dataset 5	71.14%	100%	0%
Dataset 6	75.25%	100%	0%

Table 4.13: Average Accuracy and Spam/Ham Recall from Cross-Validation for CNN/Hessian Free Optimization Model

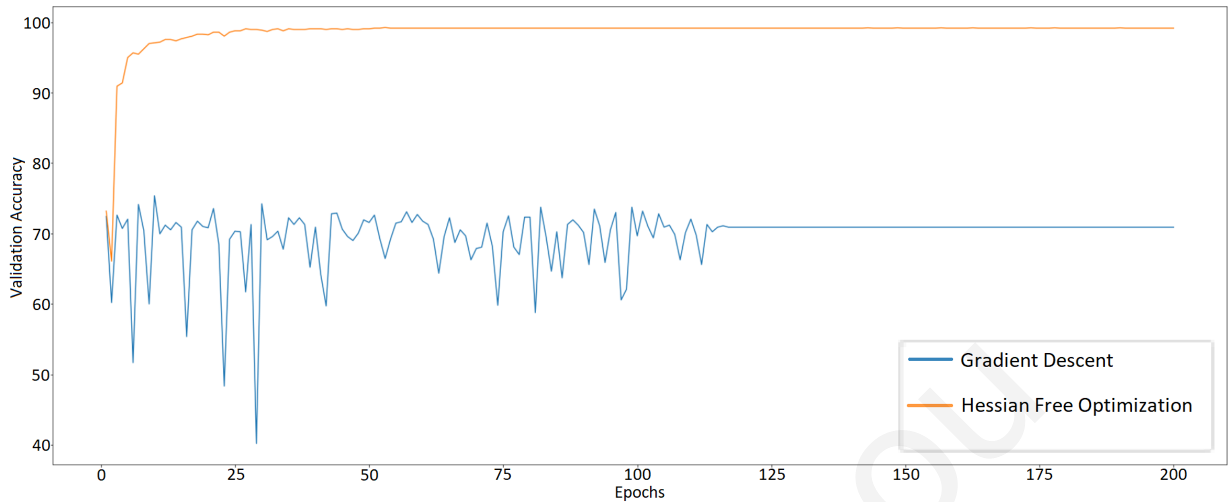
Dataset Average	Accuracy	Spam Recall	Ham Recall
Dataset 1	99.078%	97.13%	99.88%
Dataset 2	99.141%	96.72%	100%
Dataset 3	99.772%	99.38%	99.918%
Dataset 4	99.24%	99.534%	98.424%
Dataset 5	98.762%	98.57%	100%
Dataset 6	98.846%	98.70%	99.25%

On the other hand, our model behaves very well and is able to achieve 99% from the first 50-70 epochs. Overall, if we let our model run for 1000 epoch, the Gradient model is faster than our HFO model, around 25 minutes, but we do not want a model that cannot learn even if it is faster than ours.

Before we proceed with the comparison between the CNNs/HFO and the original paper's models, it is important to mention that our model converges faster than the Convolutional Neural Networks with Gradient Descent (GD). We can see that in Figure 4.18, where HFO model needs at most 50 epochs to converge whereas GD needs almost 2 and a half times more than that.For Figure 4.18 we use one of the folds from dataset 3 to train HFO and GD models.Since the model was using second order optimizer, we knew from the start that it would probably converge faster than a first order optimizer, and with Figure 4.18 we prove this point in section 2.3.8.This deserves further investigation.

Kypros Ioannou

Figure 4.18: CNNs with Hessian Free Optimization converges 2.5 times faster than the CNNs with Gradient Descent.



4.10 Comparison Between Original Dataset Work and CNN With Hessian Free Optimisation

Finally, we are going to compare our model with Meltis et al. [34] work. Since the authors did not split their dataset to see how each model behaves in new data, the Spam and Ham Recall we show here is based on the whole dataset. We use all the data from each dataset to train our model. Table 4.14 shows the Spam and Ham Recall for all of the 5 different types of Naives Bayes and Algorithm and the Recall for our CNN/HFO model. Our model has higher Spam and Ham Recall from any of Meltis Naïve Bayes algorithms.

Table 4.14: Dataset 1 Spam and Ham Recall from original paper and our model

Dataset 1	Spam Recall	Ham Recall
FB	90.50%	97.64%
MV Gauss	93.08%	94.83%
MN TF	95.66%	94.00%
MV Bern.	97.08%	93.19%
MN Bool.	96.00%	95.25%
CNNs/HFO	97.39%	99.94%

We continue with the second dataset in Table 4.15, and again, our model has higher

Kypros Ioannou

Spam and Ham Recall from any of the 5 Naïve Algorithms. Also, it achieved 100% for Ham emails which is the main focus of this dissertation.

Table 4.15: Dataset 2 Spam and Ham Recall from original paper and our model

Dataset 2	Spam Recall	Ham Recall
FB	93.63%	98.83%
MV Gauss	95.80%	96.97%
MN TF	96.81%	96.78%
MV Bern.	91.05%	97.22%
MN Bool.	96.68%	97.83%
CNNs/HFO	96.98%	100%

The third dataset it shows in Table 4.16 and has almost have a perfect score for Ham and Spam Recall. Again our model outbid the other algorithms.

Table 4.16: Dataset 3 Spam and Ham Recall from original paper and our model

Dataset 3	Spam Recall	Ham Recall
FB	96.94%	95.36%
MV Gauss	97.55%	88.81%
MN TF	95.04%	98.83%
MV Bern.	97.42%	75.41%
MN Bool.	96.94%	98.88%
CNNs/HFO	99.59%	99.94%

For Dataset four(Table 4.17), the spam recall is higher than every naïve Bayes algorithm. Even though the ham recall from our model is not higher, we have to consider that our model in total is better if we took MV Gauss or MN. Bool algorithms that they have higher Ham Recall than our model have spam Recall lower than us. For the first, the difference is 19%, and it is almost 2% for the second.

Kypris Ioannou

Table 4.17: Dataset 4 Spam and Ham Recall from original paper and our model

Dataset 4	Spam Recall	Ham Recall
FB	95.78%	96.61%
MV Gauss	80.14%	99.39%
MN TF	97.79%	98.30%
MV Bern.	97.70%	95.86%
MN Bool.	97.79%	99.05%
CNNs/HFO	99.58%	98.59%

For dataset five (Table 4.18), we have the opposite where our model has the highest Ham Recall but no the higher spam Recal but again in total, our model Recall is better than any of the other models.

Table 4.18: Dataset 5 Spam and Ham Recall from original paper and our model

Dataset 5	Spam Recall	Ham Recall
FB	99.56%	90.76%
MV Gauss	95.42%	97.28%
MN TF	99.42%	95.65%
MV Bern.	97.95%	90.08%
MN Bool.	99.69%	95.65%
CNNs/HFO	98.69%	100%

Dataset 6 (Table 4.19) is the last, and we achieve higher Ham Recall than any of the five other algorithms but our Spam Recall is not the highest in general. However, our model in total is better than any of the other models.

Table 4.19: Dataset 6 Spam and Ham Recall from original paper and our model

Dataset 6	Spam Recall	Ham Recall
FB	99.55%	89.97%
MV Gauss	91.95%	95.87%
MN TF	98.08%	95.12%
MV Bern.	97.92%	82.52%
MN Bool.	98.10%	96.88%
CNNs/HFO	98.93%	99.19%

Kypros Ioannou

Chapter 5

Conclusion and Future Work

5.1 Conclusion

The primary purpose of this dissertation was to prove that a second-order algorithm such as Hessian Free Optimization (HFO) can achieve very good results when we combine it with a Convolutional Neural Network (CNN). We also seek to understand if a CNN/HFO model can achieve better results than a classic CNN with Gradient Descent in a Natural Language Problem as Email Classification. Lastly, since we had the results from the original paper that provided us with the data, we also compared our model with the six different Naïve Bayes algorithms used in said paper [34].

The current model was measured in Accuracy and Ham and Spam Recall to be compared with Meltis et al. [34] and with a Convolutional Neural Network with Gradient Descent. We tested the model in six different datasets, and in order to compare it with the six different Naïve Bayes algorithms that the original paper used, we used the entire Dataset each time to train it.

For the first Dataset, we achieve Accuracy of 99.199% and Spam and Ham Recall 97.39% and 99.94%, respectively. Our model has higher Spam and Ham Recall than any Meltis et al. [34] Naïve Bayes algorithms with higher MN. Bool with 96% and 95.25%. For the second email, we achieve an accuracy of 99.227% and 96.98% and 100% Spam and Ham Recall. Again their best model was MN. Bool with 96.68% and 97.83% Ham and Spam Recall. With Dataset 3, we achieve a nearly perfect score with an accuracy of 99.848% and with 99.59% and 99.94% Ham and Spam Recall. Again we have the same algorithm as the best model from their work with 96.94% and 98.88% for Spam and Ham Recall.

This model did not outperform every Naïve Bayes algorithm in every Ham and Spam Recall with the last three dataset models. However, comparing which model

Kypros Ioannou

achieves the most correctly classified emails each time, our model surpasses that of the original paper. We start with the fourth Dataset, where we achieve accuracy of 99.333% and 99.58% and 98.59% Ham and Spam Recall. MN. Bool is once again their highest-performing model with 97.79% Spam and 99.05% Ham Recall. We proceed with the fifth Dataset, where we achieve accuracy of 99.061% and 98.69% Spam and a perfect score(100%) for Ham Recall. MN. Bool persists as the best model of Meltis et al. with 99.69% Spam and 95.65% Ham Recall. With the last model, we have the lowest accuracy of all other datasets, but it still produces commendable results. We achieved an accuracy of 98.997% and 98.93% Spam, and 99.19% Ham Recall. Their best model did not change, with 98.10% for Spam and 96.88% for Ham Recall.

In addition to comparing the model of this project with the model in the original paper, we also did cross-validation five-folds for our model with the Convolutional neural network (CNN) that uses Gradient Descent (GD). CNN with GD could not identify spam and ham emails correctly, yet our model achieves impressive results in every fold for every Dataset. The average accuracy we achieve with the classic CNN was 71.44% up to 76.13%, while with the other model the average accuracy ranged from 98.762% to 99.742%. We achieve similar Accuracy, and Spam and Ham Recall both when we split the dataset, and when we use the entire dataset to train the model. On the other hand, CNN with GD had 0% Spam Recall for the first three datasets, and 0 Ham Recall the last three datasets. Overall, the Convolutional Neural Network with Hessian Free Optimisation has achieved excellent results and performed well in every comparison we made. Lastly, we need to mention that since we used a second order optimizer we with the CNNs, we expected the model would converge faster compared to CNNs with Gradient Descent. CNNs with HFO converge 2.5 times faster than the one with Gradient Descent which we have shown in section 4.8 when by training both models for 200 epochs.

Kypros Ioannou

5.2 Future Work

We experiment with a small group of datasets, around 5000 emails per dataset. Meltis et al. who use the original dataset, also achieve very good results comparable to our algorithm. Using such simple algorithms we may have very good performance on a small dataset, but those algorithms will probably fail in a high volume of data, and a deep Neural Network can be endured and cope in such a high volume of data.

An attacker can very quickly change how they construct the email in order trick the email provider's filter and the user. An obvious example of that is the current pandemic where the attackers quickly adapted the emails' theme to deceive users. The email theme is not the only thing an attacker can change; it could also be where they hide the malware or the link to navigate the user to a malicious site.

Since it is effortless for an attacker to change tactics to achieve their goal, we believe that a Deep Neural Network such as CNN can be used to adapt to those changes and correctly predict other emails that have never been seen before. With the conclusion of this dissertation, the next step is to find and train the outlined model with a high volume of data with various emails to test when it can adapt and learn. It is not easy to find legit emails since most people and companies do not disclose their personal emails with the public.Furthermore, it is important to note that our data is outdated, since it is from 2006. As we already mentioned, the attackers' tactics change often and with ease, so we need to find newer datasets compared to those used in this dissertation, or possibly combine "old-fashioned" tactics with new ones.

Another thing we could do is to test how the model reacts in a real scenario situation. After we train the model with a high volume of data, we can use either Microsoft Email Provider (Microsoft Outlook) or Google's (Gmail) API to filter emails using our model. That way we may see whether it performs as well as an already tried-and-tested filter by Microsoft and Google, or whether it achieves different results.

Having shown that our model converges really fast, it is natural that such a model could been used in the business industry. It is well-behaved and learns quite fast, hence a company can use such a model in order to create a personal email filter for all customers.

Another thing to mention is that we do not need to only use this model for the spam classification problem, but for a variety of different Natural Language Processing problems as well. The reasoning behind this is that all the preprocessing steps we

Kypros Ioannou

undertook for our spam classification problem can also be applied for other text processing problems. An example of that is Sentiment Analysis for determining whether the sentiment towards any topic or products etc. is positive, negative or neutral, based on the text corpus. Sentiment analysis may also be used for predicting elections based on social media activity. By analyzing tweets from Twitter where millions of people express their thoughts, we are able to make predictions on the performance for each candidate.

As a conclusion, our model can be used for document processing. Documents contain valuable information and using a machine learning model such as the one presented in this thesis instead of doing it manually, we are able to reduce the time needed to read and extract that information from the documents. Plus, it is less prone to human error and costs less to automate this process.

Kypros Ioannou

Bibliography

[1] Apache spam assassins. https://spamassassin.apache.org/ (Accessed: 13-05-2021).

[2] A beginner's guide to word2vec and neural word embeddings. https://wiki.pathmind.com/word2vec (Accessed: 13-05-2021).

[3] Building and parsing mime messages. https://docs.oracle.com/cd/E19957-01/816-6028-10/asd3j.html (Accessed: 15-06-2021).

[4] Conjugate gradient methods. https://www.math.uci.edu/~chenlong/226/CG.pdf (Accessed: 22-06-2021).

[5] Enron comporation. https://en.wikipedia.org/wiki/enron (Accessed: 13-05-2021).

[6] Exploiting a crisis: How cybercriminals behaved during the outbreak. https://www.microsoft.com/security/blog/2020/06/16/exploiting-a-crisishow-cybercriminals-behaved-during-the-outbreak/ (Accessed: 15-06-2021).

[7] Line search. https://en.wikipedia.org/wiki/Line_search (Accessed: 21-06-2021).

[8] Word2vec. https://en.wikipedia.org/wiki/Word2vec (Accessed: 24-06-2021).

[9] World's most dangerous malware emotet disrupted through global action. https://www.europol.europa.eu/newsroom/news/world\T1\textquoterights-most-dangerous-malware-emotet-disrupted-through-global-actio (Accessed: 15-06-2021).

[10] M. Abadi, A. Agarwal, P. Barham, E. Brevdo, Z. Chen, C. Citro, G. S. Corrado, A. Davis, J. Dean, M. Devin, et al. Tensorflow: Large-scale machine learning on heterogeneous distributed systems. *arXiv preprint arXiv:1603.04467*, 2016.

Kypros Ioannou

[11] S. Anirban. Text classification — from bag-of-words to bert — part 2 (word2vec). https://medium.com/analytics-vidhya/text-classification-from-bag-of-words-to-bert-part-2-word2vec-35c8c3b34ee3 (Accessed: 07-12-2021).

[12] A. Bari, A. Martin, B. Boulouha, D. Barranco, J. Gonzalez-Andujar, I. Trujillo, and G. Ayad. Image feature extraction combined with a neural networks approach for the identification of olive cultivars. In *Proceedings of the 3rd IASTED international conference on visualization, imaging and image processing*, pages 613–620, 2003.

[13] A. Botev, H. Ritter, and D. Barber. Practical gauss-newton optimisation for deep learning. In *International Conference on Machine Learning*, pages 557–565. PMLR, 2017.

[14] E. Castilloa, S. Dhaduvai, P.Liu, K. Thakur, A.Dalton, and T.Strzalkowski. Email threat detection using distinct neural network approaches. In *Proceedings for the First International Workshop on Social Threats in Online Conversations: Understanding and Management,Marseille, France*, pages 48–55, 2020.

[15] S. Chakraverty, D. M. Sahoo, and N. R. Mahato. Perceptron learning rule. In *Concepts of Soft Computing*, pages 183–188. Springer, 2019.

[16] C. Charalambous. Protein secondary structure prediction using bidirectional recurrent neural networks and hessian free optimisation, 2018. BSc Thesis, Department of Computer Science, University of Cyprus.

[17] S. Chatzimiltis. Continuous input vector representation through embeddings from language models for protein structure prediction using convolutional neural networks with hessian free optimisation, 2021. BSc Thesis, Department of Computer Science, University of Cyprus.

[18] D.Milkovich. 5 alarming cyber security facts and stats. https://www.cybintsolutions.com/cyber-security-facts-stats (Accessed: 15-06-2021).

[19] J. L. Elman. Finding structure in time. *Cognitive Science*, 14(2):179–211, 1990.

[20] Y. Fang, C. Zhang, C. Huang, L. Liu, and Y. Yang. Phishing email detection using improved rcnn model with multilevel vectors and attention mechanism. *IEEE Access*, 7:56329–56340, 2019.

Kypros Ioannou

[21] X. He, D. Mudigere, M. Smelyanskiy, and M. Takác. Distributed hessian-free optimization for deep neural network. In *Workshops at the Thirty-First AAAI Conference on Artificial Intelligence*, 2017.

[22] M. Hiransha, N. Unnithan, R. Vinayakumar, and K. Soman. Deep learning based phishing e-mail detection cen-deepspam. *In: R. Verma, A. Das (eds.): Proceedings of the 1st AntiPhishing Shared Pilot at 4th ACM International Workshop on Security and Privacy Analytics (IWSPA 2018), Tempe, Arizona, USA, CEUR Workshop Proceedings*, Vol. 2124:Pages 16–20, 2018.

[23] D. H. Hubel and T. N. Wiesel. Receptive fields, binocular interaction and functional architecture in the cat's visual cortex. *The Journal of Physiology*, 160(1):106–154, 1962.

[24] M. I. Jordan. Chapter 25 - serial order: A parallel distributed processing approach. In J. W. Donahoe and V. Packard Dorsel, editors, *Neural-Network Models of Cognition*, volume 121 of *Advances in Psychology*, pages 471–495. North-Holland, 1997.

[25] R. Kiros. Training neural networks with stochastic hessian-free optimization. *arXiv preprint arXiv:1301.3641*, 2013.

[26] T. Kluyver, B. Ragan-Kelley, F. Pérez, B. E. Granger, M. Bussonnier, J. Frederic, K. Kelley, J. B. Hamrick, J. Grout, S. Corlay, et al. *Jupyter Notebooks-a publishing format for reproducible computational workflows.*, volume 2016. 2016.

[27] N. Kumar, S. Sonowal, et al. Email spam detection using machine learning algorithms. In *2020 Second International Conference on Inventive Research in Computing Applications (ICIRCA)*, pages 108–113. IEEE, 2020.

[28] Y. LeCun, P. Haffner, L. Bottou, and Y. Bengio. Object recognition with gradient-based learning. In *Shape, Contour and Grouping in Computer Vision*, pages 319–345. Springer Berlin Heidelberg, Berlin, Heidelberg, 1999.

[29] P. Leontiou. Protein secondary structure prediction using convolutional neural networks and hessian free optimisation, 2020. BSc Thesis, Department of Computer Science, University of Cyprus.

[30] R. Lippmann. An introduction to computing with neural nets. *IEEE ASSP Magazine*, 4(2):4–22, 1987.

Kypros Ioannou

[31] J. L. Lobo, J. Del Ser, A. Bifet, and N. Kasabov. Spiking neural networks and online learning: An overview and perspectives. *Neural Networks*, 121:88–100, 2020.

[32] J. Martens. Deep learning via hessian-free optimization. In *Proceedings of the 27th International Conference on International Conference on Machine Learning*, ICML'10, page 735–742, Madison, WI, USA, 2010. Omnipress.

[33] W. S. McCulloch and W. Pitts. A logical calculus of the ideas immanent in nervous activity. *The bulletin of mathematical biophysics*, 5(4):115–133, 1943.

[34] V. Metsis, I. Androutsopoulos, and G. Paliouras. Filtering with naive bayes - which naive bayes? *"Proceedings of the 3rd Conference on Email and Anti-Spam (CEAS 2006), Mountain View, CA, USA*, 2006.

[35] T. Mikolov, K. Chen, G. Corrado, and J. Dean. Efficient estimation of word representations in vector space. In Y. Bengio and Y. LeCun, editors, *1st International Conference on Learning Representations, ICLR 2013, Scottsdale, Arizona, USA, May 2-4, 2013, Workshop Track Proceedings*, 2013.

[36] T. Mikolov, I. Sutskever, K. Chen, G. Corrado, and J. Dean. Distributed representations of words and phrases and their compositionality. In *Proceedings of the 26th International Conference on Neural Information Processing Systems - Volume 2*, NIPS'13, page 3111–3119, Red Hook, NY, USA, 2013. Curran Associates Inc.

[37] D. Opitz and R. Maclin. Popular ensemble methods: An empirical study. *Journal of Artificial Intelligence Research*, 11:169–198, 1999.

[38] M. Pafitis. Convolutional neural networks with hessian-free optimization for alzheimer's diagnosis through mri images, 2021. BSc Thesis, Department of Computer Science, University of Cyprus.

[39] N. Petitto. Email security predictions 2021: 6 ways hackers will target businesses. `https://www.vadesecure.com/en/blog/email-security-predictions-6-wayshackers-will-target-businesses/` (Accessed: 15-06-2021).

[40] B. T. Polyak. Newton's method and its use in optimization. *European Journal of Operational Research*, 181(3):1086–1096, 2007.

Kypros Ioannou

[41] K. Radhakrishnan, R. R. Menon, and H. V. Nath. A survey of zero-day malware attacks and its detection methodology. In *TENCON 2019 - 2019 IEEE Region 10 Conference (TENCON)*, pages 533–539, 2019.

[42] P. Refaeilzadeh, L. Tang, and H. Liu. *Cross-Validation*, pages 532–538. Springer US, Boston, MA, 2009.

[43] R. Rehurek and P. Sojka. Gensim–python framework for vector space modelling. *NLP Centre, Faculty of Informatics, Masaryk University, Brno, Czech Republic*, 3(2), 2011.

[44] D. E. Rumelhart, G. E. Hinton, and R. J. Williams. Learning representations by back-propagating errors. *nature*, 323(6088):533–536, 1986.

[45] N. Stembert, A. Padmos, M. S. Bargh, S. Choenni, and F. Jansen. A study of preventing email (spear) phishing by enabling human intelligence. In *2015 European intelligence and security informatics conference*, pages 113–120. IEEE, 2015.

[46] K. M. Ting. *Confusion Matrix*, pages 260–260. Springer US, Boston, MA, 2017.

[47] M. Vakili, M. Ghamsari, and M. Rezaei. Performance analysis and comparison of machine and deep learning algorithms for iot data classification. *arXiv preprint arXiv:2001.09636*, 2020.

[48] R. Vinayakumar, H. Barathi Ganesh, M. Anand Kumar, K. Soman, and P. Poornachandran. Deep anti-phishnet: Applying deep neural networks for phishing email detection. *In: R. Verma, A. Das (eds.): Proceedings of the 1st AntiPhishing Shared Pilot at 4th ACM International Workshop on Security and Privacy Analytics (IWSPA 2018), Tempe, Arizona, USA, CEUR Workshop Proceedings*, Vol. 2124:Pages 39–49, 2018.

[49] C.-C. Wang, K. L. Tan, and C.-J. Lin. Newton methods for convolutional neural networks. *ACM Transactions on Intelligent Systems and Technology (TIST)*, 11(2):1–30, 2020.

[50] P. J. Werbos. Backpropagation through time: what it does and how to do it. *Proceedings of the IEEE*, 78(10):1550–1560, 1990.

[51] W. Xiujuan, Z. Chenxi, Z. Kangfeng, T. Haoyang, and T. Yuanrui. Detecting spear-phishing emails based on authentication. In *2019 IEEE 4th International Conference on Computer and Communication Systems (ICCCS)*, pages 450–456, 2019.

Kypros Ioannou

Appendix A - Source Code

CNN With Hessian-Free Optimization Source Code

```
import numpy as np # linear algebra
import pandas as pd # data processing, CSV file I/O (e.g. pd.read_csv)
import os
import matplotlib.pyplot as plt
import seaborn as sns
import math
import pdb
import tensorflow.compat.v1 as tf
tf.disable_v2_behavior()

from gensim.models import Word2Vec, KeyedVectors
from email.parser import Parser
from nltk.tokenize import word_tokenize
from nltk.corpus import stopwords
from string import punctuation
from tqdm.notebook import tqdm

from sklearn.multioutput import MultiOutputClassifier
from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import train_test_split
from sklearn.metrics import roc_auc_score
from statistics import mean
from sklearn.metrics import classification_report, confusion_matrix,
    accuracy_score, roc_auc_score

from tensorflow.python.client import device_lib
import numpy as np
import tensorflow.compat.v1.keras as keras
from keras.regularizers import l2

%load_ext autotime

#Load Test and Validation Dataset
fold = pd.read_csv('Fold4_6.csv')
test_fold=pd.read_csv('Valid4_6.csv')
```

Kypros Ioannou

```
#Separate class from features
train_labels=fold.Class
train_texts=fold.drop(['Class'], axis=1)
test_labels=test_fold.Class
test_texts=test_fold.drop(['Class'], axis=1)

#Convert all of them into numpy type
train_labels=train_labels.to_numpy()
test_labels=test_labels.to_numpy()
train_texts=train_texts.to_numpy()
test_texts=test_texts.to_numpy()


#FOR using the whole MELTIS DATASET
# train_texts=pd.concat([fold, test_fold], ignore_index=True)
# train_labels=train_texts.Class
# train_texts=train_texts.drop(['Class'], axis=1)
# train_texts=train_texts.to_numpy()
# train_labels=train_labels.to_numpy()

def acc_loss(data,labels):
    args = parse_args()

    sess_config = tf.compat.v1.ConfigProto()
    sess_config.gpu_options.allow_growth = True

    with tf.compat.v1.Session(config=sess_config) as sess:
        graph_address = args.model_file + '.meta'
        imported_graph =
            tf.compat.v1.train.import_meta_graph(graph_address)
        imported_graph.restore(sess, args.model_file)
        mean_param = [v for v in tf.compat.v1.global_variables() if
            'mean_tr:0' in v.name][0]
        label_enum_var = [v for v in tf.compat.v1.global_variables() if
            'label_enum:0' in v.name][0]

        sess.run(tf.compat.v1.variables_initializer([mean_param,
            label_enum_var]))
        mean_tr = sess.run(mean_param)
        label_enum = sess.run(label_enum_var)
```

Kypros Ioannou

```
        test_batch, num_cls, _ = read_data(train_texts,train_labels,
            dim=args.dim, label_enum=label_enum)
        test_batch[0], _ = normalize_and_reshape(test_batch[0],
            dim=args.dim, mean_tr=mean_tr)

        x = tf.compat.v1.get_default_graph().get_tensor_by_name
            ('main_params/input_of_net:0')
        y = tf.compat.v1.get_default_graph().get_tensor_by_name
            ('main_params/labels:0')
        outputs =
            tf.compat.v1.get_default_graph().get_tensor_by_name('output_of_net:0')

        if args.loss == 'MSELoss':
            loss = tf.reduce_sum(input_tensor=tf.pow(outputs-y, 2))
        else:
            loss =
                tf.reduce_sum(input_tensor=tf.nn.softmax_cross_entropy_with_logits
                (logits=outputs, labels=tf.stop_gradient(y)))

        network = (x, y, loss, outputs)

        avg_loss, acc, results = predict(sess, network, test_batch,
            args.bsize)

        # convert results back to the original labels
        inverse_map = dict(zip(np.arange(num_cls), label_enum))
        results = np.expand_dims(results, axis=1)
        results = np.apply_along_axis(lambda x: inverse_map[x[0]],
            axis=1, arr=results)

    print('In test phase, average loss: {:.3f} | accuracy:
        {:.3f}%'.format(avg_loss, acc*100))
    return results

    def confusion_matrix(actual_labels,results):
    fig = plt.figure(figsize=(10,4))
    heatmap = sns.heatmap(data =
        pd.DataFrame(confusion_matrix(actual_labels, results)), annot =
        True, fmt = "d", cmap=sns.color_palette("Reds", 50))
```

Kypros Ioannou

```
    heatmap.yaxis.set_ticklabels(heatmap.yaxis.get_ticklabels(),
        rotation=0, ha='right', fontsize=14)
    heatmap.xaxis.set_ticklabels(heatmap.xaxis.get_ticklabels(),
        rotation=45, ha='right', fontsize=14)
    plt.ylabel('Ground Truth')
    plt.xlabel('Prediction')
    plt.show()

# CNN_4layers is the main CNN structure for this experiments.
def CNN_4layers(input_shape, output_shape):

    layers = [

        keras.layers.Conv2D(128, [5, 5],kernel_regularizer=l2(0.03),
            padding='same', activation=tf.nn.relu6,
            input_shape=input_shape),
        keras.layers.MaxPool2D([2, 2], strides=2),
#         keras.layers.Conv2D(64, [5, 5],kernel_regularizer=l2(0.03),
    padding='same', activation=tf.nn.relu),
#         keras.layers.MaxPool2D([2, 2], strides=2),
#         keras.layers.Conv2D(32, [5, 5],kernel_regularizer=l2(0.03),
    padding='same', activation=tf.nn.relu),
#         keras.layers.MaxPool2D([2, 2], strides=2),
        keras.layers.Dropout(0.5),
        keras.layers.Flatten(),
        keras.layers.Dense(output_shape),

    ]
    return keras.models.Sequential(layers)

def CNN_7layers(input_shape, output_shape):
   layers = [
      keras.layers.Conv2D(32, [5, 5], padding='same',
          activation=tf.nn.relu, input_shape=input_shape),
      keras.layers.Conv2D(32, [3, 3], padding='same',
          activation=tf.nn.relu),
      keras.layers.MaxPool2D([2, 2], strides=2),
      keras.layers.Conv2D(64, [3, 3], padding='same',
          activation=tf.nn.relu),
```

Kypros Ioannou

```
        keras.layers.Conv2D(64, [3, 3], padding='same',
            activation=tf.nn.relu),
        keras.layers.MaxPool2D([2, 2], strides=2),
        keras.layers.Conv2D(64, [3, 3], padding='same',
            activation=tf.nn.relu),
        keras.layers.Conv2D(128, [3, 3], padding='same',
            activation=tf.nn.relu),
        keras.layers.MaxPool2D([2, 2], strides=2),
        keras.layers.Flatten(),
        keras.layers.Dense(output_shape),
    ]
    return keras.models.Sequential(layers)

def CNN_model(net, input_shape, output_shape):
    return globals()[net](input_shape, output_shape)

def CNN(net, num_cls, dim):
    _NUM_CLASSES = num_cls
    _IMAGE_HEIGHT, _IMAGE_WIDTH, _IMAGE_CHANNELS = dim

    with tf.name_scope('main_params'):
        x = tf.placeholder(tf.float32, shape=[None, _IMAGE_HEIGHT,
            _IMAGE_WIDTH, _IMAGE_CHANNELS], name='input_of_net')
        y = tf.placeholder(tf.float32, shape=[None, _NUM_CLASSES],
            name='labels')

    outputs = CNN_model(net, dim, num_cls)(x)
    outputs = tf.identity(outputs, name='output_of_net')

    return (x, y, outputs)


import pdb
import time
import os
import math
def Rop(f, weights, v):
    """Implementation of R operator
    Args:
        f: any function of weights
```

Kypros Ioannou

```
    weights: list of tensors.
    v: vector for right multiplication
  Returns:
    Jv: Jaccobian vector product, length same as
      the number of output of f
  """
  if type(f) == list:
    u = [tf.zeros_like(ff) for ff in f]
  else:
    u = tf.zeros_like(f) # dummy variable
  g = tf.gradients(ys=f, xs=weights, grad_ys=u)
  return tf.gradients(ys=g, xs=u, grad_ys=v)

def Gauss_Newton_vec(outputs, loss, weights, v):
  """Implements Gauss-Newton vector product.
  Args:
    loss: Loss function.
    outputs: outputs of the last layer (pre-softmax).
    weights: Weights, list of tensors.
    v: vector to be multiplied with Gauss Newton matrix
  Returns:
    J'BJv: Guass-Newton vector product.
  """
  # Validate the input
  if type(weights) == list:
    if len(v) != len(weights):
      raise ValueError("weights and v must have the same length.")

  grads_outputs = tf.gradients(ys=loss, xs=outputs)
  BJv = Rop(grads_outputs, weights, v)
  JBJv = tf.gradients(ys=outputs, xs=weights, grad_ys=BJv)
  return JBJv


class newton_cg(object):
  def __init__(self, config, sess, outputs, loss):
    """
    initialize operations and vairables that will be used in newton
    args:
      sess: tensorflow session
```

Kypros Ioannou

```
        outputs: output of the neural network (pre-softmax layer)
        loss: function to calculate loss
    """
    super(newton_cg, self).__init__()
    self.sess = sess
    self.config = config
    self.outputs = outputs
    self.loss = loss
    self.param = tf.compat.v1.trainable_variables()

    self.CGiter = 0
    FLOAT = tf.float32
    model_weight = self.vectorize(self.param)

    # initial variable used in CG
    zeros = tf.zeros(model_weight.get_shape(), dtype=FLOAT)
    self.r = tf.Variable(zeros, dtype=FLOAT, trainable=False)
    self.v = tf.Variable(zeros, dtype=FLOAT, trainable=False)
    self.s = tf.Variable(zeros, dtype=FLOAT, trainable=False)
    self.g = tf.Variable(zeros, dtype=FLOAT, trainable=False)
    # initial Gv, f for method minibatch
    self.Gv = tf.Variable(zeros, dtype=FLOAT, trainable=False)
    self.f = tf.Variable(0., dtype=FLOAT, trainable=False)

    # rTr, cgtol and beta to be used in CG
    self.rTr = tf.Variable(0., dtype=FLOAT, trainable=False)
    self.cgtol = tf.Variable(0., dtype=FLOAT, trainable=False)
    self.beta = tf.Variable(0., dtype=FLOAT, trainable=False)

    # placeholder alpha, old_alpha and lambda
    self.alpha = tf.compat.v1.placeholder(FLOAT, shape=[])
    self.old_alpha = tf.compat.v1.placeholder(FLOAT, shape=[])
    self._lambda = tf.compat.v1.placeholder(FLOAT, shape=[])

    self.num_grad_segment =
        math.ceil(self.config.num_data/self.config.bsize)
    self.num_Gv_segment =
        math.ceil(self.config.GNsize/self.config.bsize)

    cal_loss, cal_lossgrad, cal_lossGv, \
```

Kypros Ioannou

```
    add_reg_avg_loss, add_reg_avg_grad, add_reg_avg_Gv, \
    zero_loss, zero_grad, zero_Gv = self._ops_in_minibatch()

    # initial operations that will be used in minibatch and newton
    self.cal_loss = cal_loss
    self.cal_lossgrad = cal_lossgrad
    self.cal_lossGv = cal_lossGv
    self.add_reg_avg_loss = add_reg_avg_loss
    self.add_reg_avg_grad = add_reg_avg_grad
    self.add_reg_avg_Gv = add_reg_avg_Gv
    self.zero_loss = zero_loss
    self.zero_grad = zero_grad
    self.zero_Gv = zero_Gv

    self.CG, self.update_v = self._CG()
    self.init_cg_vars = self._init_cg_vars()
    self.update_gs = tf.tensordot(self.s, self.g, axes=1)
    self.update_sGs = 0.5*tf.tensordot(self.s,
        -self.g-self.r-self._lambda*self.s, axes=1)
    self.update_model = self._update_model()
    self.gnorm = self.calc_norm(self.g)



def vectorize(self, tensors):
    if isinstance(tensors, list) or isinstance(tensors, tuple):
        vector = [tf.reshape(tensor, [-1]) for tensor in tensors]
        return tf.concat(vector, 0)
    else:
        return tensors

def inverse_vectorize(self, vector, param):
    if isinstance(vector, list):
        return vector
    else:
        tensors = []
        offset = 0
        num_total_param = np.sum([np.prod(p.shape.as_list()) for p in
            param])
        for p in param:
```

Kypros Ioannou

```
            numel = np.prod(p.shape.as_list())
            tensors.append(tf.reshape(vector[offset: offset+numel],
               p.shape))
            offset += numel

        assert offset == num_total_param
        return tensors

    def calc_norm(self, v):
        # default: frobenius norm
        if isinstance(v, list):
            norm = 0.
            for p in v:
                norm = norm + tf.norm(tensor=p)**2
            return norm**0.5
        else:
            return tf.norm(tensor=v)

    def _ops_in_minibatch(self):
        """
        Define operations that will be used in method minibatch
        Vectorization is already a deep copy operation.
        Before using newton method, loss needs to be summed over training
            samples
        to make results consistent.
        """

        def cal_loss():
            return tf.compat.v1.assign(self.f, self.f + self.loss)

        def cal_lossgrad():
            update_f = tf.compat.v1.assign(self.f, self.f + self.loss)

            grad = tf.gradients(ys=self.loss, xs=self.param)
            grad = self.vectorize(grad)
            update_grad = tf.compat.v1.assign(self.g, self.g + grad)

            return tf.group(*[update_f, update_grad])

        def cal_lossGv():
```

Kypros Ioannou

```
      v = self.inverse_vectorize(self.v, self.param)
      Gv = Gauss_Newton_vec(self.outputs, self.loss, self.param, v)
      Gv = self.vectorize(Gv)
      return tf.compat.v1.assign(self.Gv, self.Gv + Gv)

   # add regularization term to loss, gradient and Gv and further
       average over batches
   def add_reg_avg_loss():
      model_weight = self.vectorize(self.param)
      reg = (self.calc_norm(model_weight))**2
      reg = 1.0/(2*self.config.C) * reg
      return tf.compat.v1.assign(self.f, reg +
          self.f/self.config.num_data)

   def add_reg_avg_lossgrad():
      model_weight = self.vectorize(self.param)
      reg_grad = model_weight/self.config.C
      return tf.compat.v1.assign(self.g, reg_grad +
          self.g/self.config.num_data)

   def add_reg_avg_lossGv():
      return tf.compat.v1.assign(self.Gv, (self._lambda +
          1/self.config.C)*self.v
       + self.Gv/self.config.GNsize)

   # zero out loss, grad and Gv
   def zero_loss():
      return tf.compat.v1.assign(self.f, tf.zeros_like(self.f))
   def zero_grad():
      return tf.compat.v1.assign(self.g, tf.zeros_like(self.g))
   def zero_Gv():
      return tf.compat.v1.assign(self.Gv, tf.zeros_like(self.Gv))

   return (cal_loss(), cal_lossgrad(), cal_lossGv(),
         add_reg_avg_loss(), add_reg_avg_lossgrad(),
            add_reg_avg_lossGv(),
         zero_loss(), zero_grad(), zero_Gv())

def minibatch(self, data_batch, place_holder_x, place_holder_y, mode):
   """
```

Kypros Ioannou

```
A function to evaluate either function value, global gradient or
    sub-sampled Gv
"""
if mode not in ('funonly', 'fungrad', 'Gv'):
   raise ValueError('Unknown mode other than funonly & fungrad &
      Gv!')

inputs, labels = data_batch
num_data = labels.shape[0]
num_segment = math.ceil(num_data/self.config.bsize)
x, y = place_holder_x, place_holder_y

# before estimation starts, need to zero out f, grad and Gv
    according to the mode

if mode == 'funonly':
   assert num_data == self.config.num_data
   assert num_segment == self.num_grad_segment
   self.sess.run(self.zero_loss)
elif mode == 'fungrad':
   assert num_data == self.config.num_data
   assert num_segment == self.num_grad_segment
   self.sess.run([self.zero_loss, self.zero_grad])
else:
   assert num_data == self.config.GNsize
   assert num_segment == self.num_Gv_segment
   self.sess.run(self.zero_Gv)

for i in range(num_segment):

   load_time = time.time()
   idx = np.arange(i * self.config.bsize, min((i+1) *
      self.config.bsize, num_data))
   batch_input = inputs[idx]

   batch_labels = labels[idx]

   batch_input = np.ascontiguousarray(batch_input)
   batch_labels = np.ascontiguousarray(batch_labels)
   self.config.elapsed_time += time.time() - load_time
```

Kypros Ioannou

```
        if mode == 'funonly':

            self.sess.run(self.cal_loss, feed_dict={
                        x: batch_input,
                        y: batch_labels,})

        elif mode == 'fungrad':

            self.sess.run(self.cal_lossgrad, feed_dict={
                        x: batch_input,
                        y: batch_labels,})

        else:

            self.sess.run(self.cal_lossGv, feed_dict={
                        x: batch_input,
                        y: batch_labels})

    # average over batches
    if mode == 'funonly':
        self.sess.run(self.add_reg_avg_loss)
    elif mode == 'fungrad':
        self.sess.run([self.add_reg_avg_loss, self.add_reg_avg_grad])
    else:
        self.sess.run(self.add_reg_avg_Gv,
            feed_dict={self._lambda: self.config._lambda})


def _update_model(self):
    update_model_ops = []
    x = self.inverse_vectorize(self.s, self.param)
    for i, p in enumerate(self.param):
        op = tf.compat.v1.assign(p, p + (self.alpha-self.old_alpha) *
             x[i])
        update_model_ops.append(op)
    return tf.group(*update_model_ops)

def _init_cg_vars(self):
    init_ops = []
```

Kypros Ioannou

```
        init_r = tf.compat.v1.assign(self.r, -self.g)
        init_v = tf.compat.v1.assign(self.v, -self.g)
        init_s = tf.compat.v1.assign(self.s, tf.zeros_like(self.g))
        gnorm = self.calc_norm(self.g)
        init_rTr = tf.compat.v1.assign(self.rTr, gnorm**2)
        init_cgtol = tf.compat.v1.assign(self.cgtol, self.config.xi*gnorm)

        init_ops = [init_r, init_v, init_s, init_rTr, init_cgtol]

        return tf.group(*init_ops)

    def _CG(self):
        """
        CG:
            define operations that will be used in method newton
        Same as the previous loss calculation,
        Gv has been summed over batches when samples were fed into Neural
            Network.
        """

        def CG_ops():

            vGv = tf.tensordot(self.v, self.Gv, axes=1)

            alpha = self.rTr / vGv
            with tf.control_dependencies([alpha]):
                update_s = tf.compat.v1.assign(self.s, self.s + alpha *
                    self.v, name='update_s_ops')
                update_r = tf.compat.v1.assign(self.r, self.r - alpha *
                    self.Gv, name='update_r_ops')

                with tf.control_dependencies([update_s, update_r]):
                    rnewTrnew = self.calc_norm(update_r)**2
                    update_beta = tf.compat.v1.assign(self.beta, rnewTrnew /
                        self.rTr)
                    with tf.control_dependencies([update_beta]):
                        update_rTr = tf.compat.v1.assign(self.rTr, rnewTrnew,
                            name='update_rTr_ops')
```

Kypros Ioannou

```
            return tf.group(*[update_s, update_beta, update_rTr])

        def update_v():
            return tf.compat.v1.assign(self.v, self.r + self.beta*self.v,
                name='update_v')

        return (CG_ops(), update_v())


    def newton(self, full_batch, val_batch, saver, network,
         test_network=None):
        """
        Conduct newton steps for training
        args:
            full_batch & val_batch: provide training set and validation set.
                The function will
              save the best model evaluted on validation set for future
                  prediction.
            network: a tuple contains (x, y, loss, outputs).
            test_network: a tuple similar to argument network. If you use
                layers which behave differently
              in test phase such as batchnorm, a separate test_network is
                  needed.
        return:
            None
        """
        # check whether data is valid
        full_inputs, full_labels = full_batch
        assert full_inputs.shape[0] == full_labels.shape[0]

        if full_inputs.shape[0] != self.config.num_data:
            raise ValueError('The number of full batch inputs does not agree
                with the config argument.\
                        This is important because global loss is averaged
                            over those inputs')

        x, y, _, outputs = network

        tf.compat.v1.summary.scalar('loss', self.f)
        merged = tf.compat.v1.summary.merge_all()
```

Kypros Ioannou

```
train_writer = tf.compat.v1.summary.FileWriter('./summary/train',
    self.sess.graph)

print(self.config.args)
if not self.config.screen_log_only:
   log_file = open(self.config.log_file, 'w')
   print(self.config.args, file=log_file)

self.minibatch(full_batch, x, y, mode='fungrad')
f = self.sess.run(self.f)
output_str = 'initial f: {:.3f}'.format(f)
print(output_str)
if not self.config.screen_log_only:
   print(output_str, file=log_file)

best_acc = 0.0

total_running_time = 0.0
self.config.elapsed_time = 0.0
total_CG = 0

for k in range(self.config.iter_max):

   # randomly select the batch for Gv estimation
   idx = np.random.choice(np.arange(0, full_labels.shape[0]),
         size=self.config.GNsize, replace=False)

   mini_inputs = full_inputs[idx]
   mini_labels = full_labels[idx]

   start = time.time()

   self.sess.run(self.init_cg_vars)
   cgtol = self.sess.run(self.cgtol)

   avg_cg_time = 0.0
   for CGiter in range(1, self.config.CGmax+1):

      cg_time = time.time()
      self.minibatch((mini_inputs, mini_labels), x, y, mode='Gv')
```


Kypros Ioannou

```
        avg_cg_time += time.time() - cg_time

        self.sess.run(self.CG)

        rnewTrnew = self.sess.run(self.rTr)

        if rnewTrnew**0.5 <= cgtol or CGiter == self.config.CGmax:
          break

        self.sess.run(self.update_v)

      print('Avg time per Gv iteration: {:.5f}
          s\r\n'.format(avg_cg_time/CGiter))

      gs, sGs = self.sess.run([self.update_gs, self.update_sGs],
          feed_dict={
            self._lambda: self.config._lambda
        })

      # line_search
      f_old = f
      alpha = 1
      while True:

        old_alpha = 0 if alpha == 1 else alpha/0.5

        self.sess.run(self.update_model, feed_dict={
          self.alpha:alpha, self.old_alpha:old_alpha
          })

        prered = alpha*gs + (alpha**2)*sGs

        self.minibatch(full_batch, x, y, mode='funonly')
        f = self.sess.run(self.f)

        actred = f - f_old

        if actred <= self.config.eta*alpha*gs:
          break
```

Kypros Ioannou

```
    alpha *= 0.5

# update lambda
ratio = actred / pred
if ratio < 0.25:
    self.config._lambda *= self.config.boost
elif ratio >= 0.75:
    self.config._lambda *= self.config.drop

self.minibatch(full_batch, x, y, mode='fungrad')
f = self.sess.run(self.f)

gnorm = self.sess.run(self.gnorm)

summary = self.sess.run(merged)
train_writer.add_summary(summary, k)

# exclude data loading time for fair comparison
end = time.time()

end = end - self.config.elapsed_time
total_running_time += end-start

self.config.elapsed_time = 0.0

total_CG += CGiter

output_str = '{}-iter f: {:.3f} |g|: {:.5f} alpha: {:.3e} ratio:
    {:.3f} lambda: {:.5f} #CG: {} actred: {:.5f} prered: {:.5f}
    time: {:.3f}'.\
            format(k, f, gnorm, alpha, actred/prered,
                self.config._lambda, CGiter, actred, prered,
                end-start)
print(output_str)
if not self.config.screen_log_only:
    print(output_str, file=log_file)

if val_batch is not None:
    # Evaluate the performance after every Newton Step
    if test_network == None:
```

Kypros Ioannou

```
        val_loss, val_acc, _ = predict(
            self.sess,
            network=(x, y, self.loss, outputs),
            test_batch=val_batch,
            bsize=self.config.bsize,
            )
    else:
        # A separat test network part has not been done...
        val_loss, val_acc, _ = predict(
            self.sess,
            network=test_network,
            test_batch=val_batch,
            bsize=self.config.bsize
            )

    output_str = '\r\n {}-iter val_acc: {:.3f}% val_loss
        {:.3f}\r\n'.\
        format(k, val_acc*100, val_loss)
    print(output_str)
    if not self.config.screen_log_only:
        print(output_str, file=log_file)

    if val_acc > best_acc:
        best_acc = val_acc
        checkpoint_path = self.config.model_file
        save_path = saver.save(self.sess, checkpoint_path)
        print('Best model saved in {}\r\n'.format(save_path))

if val_batch is None:
    checkpoint_path = self.config.model_file
    save_path = saver.save(self.sess, checkpoint_path)
    print('Model at the last iteration saved in
        {}\r\n'.format(save_path))
    output_str = 'total_#CG {} | total running time
        {:.3f}s'.format(total_CG, total_running_time)
else:
    output_str = 'Final acc: {:.3f}% | best acc {:.3f}% | total_#CG
        {} | total running time {:.3f}s'.\
        format(val_acc*100, best_acc*100, total_CG,
            total_running_time)
```

```
        print(output_str)
        if not self.config.screen_log_only:
            print(output_str, file=log_file)
            log_file.close()

import scipy.io as sio
import os


class ConfigClass(object):
    def __init__(self, args, num_data, num_cls):
        super(ConfigClass, self).__init__()
        self.args = args
        self.iter_max = args.iter_max

        # Different notations of regularization term:
        # In SGD, weight decay:
        #  weight_decay <- lr/(C*num_of_training_samples)
        # In Newton method:
        #  C <- C * num_of_training_samples

        self.seed = args.seed

        if self.seed is None:
            print('You choose not to specify a random seed.'+\
                'A different result is produced after each run.')
        elif isinstance(self.seed, int) and self.seed >= 0:
            print('You specify random seed {}.'.format(self.seed))
        else:
            raise ValueError('Only accept None type or nonnegative integers
                 for'+\
                   ' random seed argument!')

        self.train_set = args.train_set
        self.val_set = args.val_set
        self.num_cls = num_cls
        self.dim = args.dim

        self.num_data = num_data
        self.GNsize = min(args.GNsize, self.num_data)
```

Kypros Ioannou

```
self.C = args.C * self.num_data
self.net = args.net

self.xi = 0.1
self.CGmax = args.CGmax
self._lambda = args._lambda
self.drop = args.drop
self.boost = args.boost
self.eta = args.eta
self.lr = args.lr
self.lr_decay = args.lr_decay

self.bsize = args.bsize
if args.momentum < 0:
  raise ValueError('Momentum needs to be larger than 0!')
self.momentum = args.momentum

self.loss = args.loss
if self.loss not in ('MSELoss', 'CrossEntropy'):
  raise ValueError('Unrecognized loss type!')
self.optim = args.optim
if self.optim not in ('SGD', 'NewtonCG', 'Adam'):
  raise ValueError('Only support SGD, Adam & NewtonCG optimizer!')

self.log_file = args.log_file
self.model_file = args.model_file
self.screen_log_only = args.screen_log_only

if self.screen_log_only:
  print('You choose not to store running log. Only store model to
     {}'.format(self.log_file))
else:
  print('Saving log to: {}'.format(self.log_file))
  dir_name, _ = os.path.split(self.log_file)
  if not os.path.isdir(dir_name):
    os.makedirs(dir_name, exist_ok=True)

dir_name, _ = os.path.split(self.model_file)
if not os.path.isdir(dir_name):
  os.makedirs(dir_name, exist_ok=True)
```

Kypros Ioannou

```
        self.elapsed_time = 0.0


def read_data(enron_data,data_class, dim, label_enum=None):
    """
    args:
        filename: the path where .mat files are stored
        label_enum (default None): the list that stores the original
            labels.
            If label_enum is None, the function will generate a new list
                which stores the
            original labels in a sequence, and map original labels to [0, 1,
                ... number_of_classes-1].
            If label_enum is a list, the function will use it to convert
            original labels to [0, 1,..., number_of_classes-1].
    """

#   mat_contents = sio.loadmat(filename)
    images, labels = enron_data,data_class
    images=np.array(enron_data)
    labels=list(map(lambda el:[el], data_class))
    labels=np.array(labels)
    labels = labels.reshape(-1)
    images = images.reshape(images.shape[0], -1)

    _IMAGE_HEIGHT, _IMAGE_WIDTH, _IMAGE_CHANNELS = dim
    zero_to_append = np.zeros((images.shape[0],
        _IMAGE_CHANNELS*_IMAGE_HEIGHT*_IMAGE_WIDTH-np.prod(images.shape[1:])))
    images = np.append(images, zero_to_append, axis=1)

    # check data validity
    if label_enum is None:
        label_enum, labels = np.unique(labels, return_inverse=True)
        num_cls = labels.max()+1


        if len(label_enum) != num_cls:
            raise ValueError('The number of classes is not equal to the
                number of\
```

Kypros Ioannou

```
                labels in dataset. Please verify them.')
    else:
        num_cls = len(label_enum)
        forward_map = dict(zip(label_enum, np.arange(num_cls)))
        labels = np.expand_dims(labels, axis=1)
        labels = np.apply_along_axis(lambda x:forward_map[x[0]], axis=1,
            arr=labels)

    # convert groundtruth to one-hot encoding

    labels = np.eye(num_cls)[labels]
    labels = labels.astype('float32')
    return [images, labels], num_cls, label_enum

def normalize_and_reshape(images, dim, mean_tr=None):
    _IMAGE_HEIGHT, _IMAGE_WIDTH, _IMAGE_CHANNELS = dim
    images_shape = [images.shape[0], _IMAGE_CHANNELS, _IMAGE_HEIGHT,
        _IMAGE_WIDTH]

    # images normalization and zero centering
    images = images.reshape(images_shape[0], -1)

    images = images/255.0

    if mean_tr is None:
        print('No mean of data provided! Normalize images by their own
            mean.')
        # if no mean_tr is provided, we calculate it according to the
            current data
        mean_tr = images.mean(axis=0)
    else:
        print('Normalize images according to the provided mean.')
        if np.prod(mean_tr.shape) != np.prod(dim):
            raise ValueError('Dimension of provided mean does not agree with
                the data! Please verify them!')

    images = images - mean_tr

    images = images.reshape(images_shape)
    # Tensorflow accepts data shape: B x H x W x C
```


Kypros Ioannou

```
    images = np.transpose(images, (0, 2, 3, 1))
    return images, mean_tr


def predict(sess, network, test_batch, bsize):
    x, y, loss, outputs = network

    test_inputs, test_labels = test_batch
    batch_size = bsize

    num_data = test_labels.shape[0]
    num_batches = math.ceil(num_data/batch_size)

    results = np.zeros(shape=num_data, dtype=int)
    infer_loss = 0.0

    for i in range(num_batches):
        batch_idx = np.arange(i*batch_size, min((i+1)*batch_size,
             num_data))
#       batch_idx = np.arange(i*batch_size, min((i+1)*batch_size, 1459))
        batch_input = test_inputs[batch_idx]
        batch_labels = test_labels[batch_idx]

        net_outputs, _loss = sess.run(
            [outputs, loss], feed_dict={x: batch_input, y: batch_labels}
            )
#       print(net_outputs)
        results[batch_idx] = np.argmax(net_outputs, axis=1)
        # note that _loss was summed over batches
        infer_loss = infer_loss + _loss

    acc = (np.argmax(test_labels, axis=1) == results).mean()
    avg_loss = infer_loss/num_data

    return avg_loss, acc, results

# By setting HFO=True we use HFO. If we set it to False then we use SGD
HFO=True
if HFO==True:
```


Kypros Ioannou

```
    train_args = ("--optim NewtonCG --GNsize 512 --C 0.5 --net
        CNN_4layers --bsize 16 --iter_max 1000 " +
          "--train_set ./" + "TRAIN_FILE" + " --val_set ./" +
              "VALID_FILE" + " --dim " +
          str(15) + " " + str(20)+" "+"1" ).split()
else:
    train_args = ("--optim SGD --lr 0.5 --net CNN_4layers --bsize 16
        --epoch_max 1000 " +
              "--train_set ./" + 'TRAIN_FILE' + " --val_set ./" +
                  'VALID_FILE' + " --dim " +
              str(15) + " " + str(20) + " 1").split()

import pdb

# tf.compat.v1.disable_eager_execution()

import argparse

def parse_args():
   parser = argparse.ArgumentParser(description='Newton method on DNN')
   parser.add_argument('--C', dest='C',
                help='regularization term, or so-called weight decay
                    where'+\
                    'weight_decay = lr/(C*num_of_samples) in this
                        implementation' ,
                default=0.01, type=float)

   # Newton method arguments
   parser.add_argument('--GNsize', dest='GNsize',
                help='number of samples for estimating Gauss-Newton
                    matrix',
                default=4096, type=int)
   parser.add_argument('--iter_max', dest='iter_max',
                help='the maximal number of Newton iterations',
                default=100, type=int)
   parser.add_argument('--xi', dest='xi',
                help='the tolerance in the relative stopping condition
                    for CG',
                default=0.1, type=float)
   parser.add_argument('--drop', dest='drop',
```

Kypros Ioannou

```
                help='the drop constants for the LM method',
                default=2/3, type=float)
parser.add_argument('--boost', dest='boost',
                help='the boost constants for the LM method',
                default=3/2, type=float)
parser.add_argument('--eta', dest='eta',
                help='the parameter for the line search stopping
                    condition',
                default=0.0001, type=float)
parser.add_argument('--CGmax', dest='CGmax',
                help='the maximal number of CG iterations',
                default=250, type=int)
parser.add_argument('--lambda', dest='_lambda',
                help='the initial lambda for the LM method',
                default=1, type=float)

# SGD arguments
parser.add_argument('--epoch_max', dest='epoch',
                help='number of training epoch',
                default=500, type=int)
parser.add_argument('--lr', dest='lr',
                help='learning rate',
                default=0.01, type=float)
parser.add_argument('--decay', dest='lr_decay',
                help='learning rate decay over each mini-batch update',
                default=0, type=float)
parser.add_argument('--momentum', dest='momentum',
                help='momentum of learning',
                default=0, type=float)

# Model training arguments
parser.add_argument('--bsize', dest='bsize',
                help='batch size to evaluate stochastic gradient, Gv,
                    etc. Since the sampled data \
                for computing Gauss-Newton matrix and etc. might not fit
                    into memeory \
                for one time, we will split the data into several
                    segements and average\
                over them.',
                default=1024, type=int)
```

Kypros Ioannou

```
    parser.add_argument('--net', dest='net',
                        help='classifier type',
                        default='CNN_4layers', type=str)
    parser.add_argument('--train_set', dest='train_set',
                        help='provide the directory of .mat file for training',
                        default='data/mnist-demo.mat', type=str)
    parser.add_argument('--val_set', dest='val_set',
                        help='provide the directory of .mat file for validation',
                        default=None, type=str)
    parser.add_argument('--model', dest='model_file',
                        help='model saving address',
                        default='./saved_model/model.ckpt', type=str)
    parser.add_argument('--log', dest='log_file',
                        help='log saving directory',
                        default='./running_log/logger.log', type=str)
    parser.add_argument('--screen_log_only', dest='screen_log_only',
                        help='screen printing running log instead of storing it',
                        action='store_true')
    parser.add_argument('--optim', '-optim',
                        help='which optimizer to use: SGD, Adam or NewtonCG',
                        default='NewtonCG', type=str)
    parser.add_argument('--loss', dest='loss',
                        help='which loss function to use: MSELoss or
                            CrossEntropy',
                        default='MSELoss', type=str)
    parser.add_argument('--dim', dest='dim', nargs='+', help='input
        dimension of data,'+\
                  'shape must be: height width num_channels',
                        default=[32, 32, 3], type=int)
    parser.add_argument('--seed', dest='seed', help='a nonnegative
        integer for reproducibility',
                        default=0, type=int)
    args = parser.parse_args(args=train_args)
    return args

args = parse_args()

def init_model(param):
    init_ops = []
    for p in param:
```

Kypros Ioannou

```
        if 'kernel' in p.name:
            weight = np.random.standard_normal(p.shape)* np.sqrt(2.0 /
                ((np.prod(p.get_shape().as_list()[:-1]))))
            opt = tf.compat.v1.assign(p, weight)
        elif 'bias' in p.name:
            zeros = np.zeros(p.shape)
            opt = tf.compat.v1.assign(p, zeros)
        init_ops.append(opt)
    return tf.group(*init_ops)

def gradient_trainer(config, sess, network, full_batch, val_batch,
     saver, test_network):
    x, y, loss, outputs, = network

    global_step = tf.Variable(initial_value=0, trainable=False,
        name='global_step')
    learning_rate = tf.compat.v1.placeholder(tf.float32, shape=[],
        name='learning_rate')

    # Probably not a good way to add regularization.
    # Just to confirm the implementation is the same as MATLAB.
    reg = 0.0
    param = tf.compat.v1.trainable_variables()
    for p in param:
        reg = reg + tf.reduce_sum(input_tensor=tf.pow(p,2))
    reg_const = 1/(2*config.C)
    batch_size = tf.compat.v1.cast(tf.shape(x)[0], tf.float32)
    loss_with_reg = reg_const*reg + loss/batch_size

    if config.optim == 'SGD':
        optimizer = tf.compat.v1.train.MomentumOptimizer(
                learning_rate=learning_rate,
                momentum=config.momentum).minimize(
                loss_with_reg,
                global_step=global_step)
    elif config.optim == 'Adam':
        optimizer =
            tf.compat.v1.train.AdamOptimizer(learning_rate=learning_rate,
                        beta1=0.9,
                        beta2=0.999,
```

Kypros Ioannou

```
                  epsilon=1e-08).minimize(
                  loss_with_reg,
                  global_step=global_step)

train_inputs, train_labels = full_batch
num_data = train_labels.shape[0]
num_iters = math.ceil(num_data/config.bsize)


if not config.screen_log_only:
   log_file = open(config.log_file, 'w')
   print(config.args, file=log_file)
sess.run(tf.compat.v1.global_variables_initializer())


print('-------------- initializing network by methods in He et al.
    (2015) --------------')
param = tf.compat.v1.trainable_variables()
sess.run(init_model(param))

total_running_time = 0.0
best_acc = 0.0
lr = config.lr

for epoch in range(0, args.epoch):

   cumulative_loss = 0.0
   cumulative_size = 0
   start = time.time()

   for i in range(num_iters):

      load_time = time.time()
      # randomly select the batch
      idx = np.random.choice(np.arange(0, num_data),
            size=config.bsize, replace=False)

      batch_input = train_inputs[idx]
      batch_labels = train_labels[idx]
      batch_input = np.ascontiguousarray(batch_input)
```


Kypros Ioannou

```
        batch_labels = np.ascontiguousarray(batch_labels)
        config.elapsed_time += time.time() - load_time

        step, _, batch_loss= sess.run(
            [global_step, optimizer, loss_with_reg],
            feed_dict = {x: batch_input, y: batch_labels, learning_rate:
                lr}
            )

        # print initial loss
        if epoch == 0 and i == 0:
            output_str = 'initial f (loss with reg of 1st batch):
                {:.3f}'.format(batch_loss)
            print(output_str)
            if not config.screen_log_only:
                print(output_str, file=log_file)

        cumulative_loss = cumulative_loss + batch_loss *
             batch_input.shape[0]
        cumulative_size = cumulative_size + batch_input.shape[0]
        # print log every 10% of the iterations
        if i % math.ceil(num_iters/10) == 0:
            end = time.time()
            output_str = 'Epoch {}: {}/{} | loss with reg {:.4f} | lr
                {:.6} | elapsed time {:.3f}'\
                .format(epoch, i, num_iters, batch_loss , lr, end-start)
            print(output_str)
            if not config.screen_log_only:
                print(output_str, file=log_file)

        # adjust learning rate for SGD by inverse time decay
        if args.optim != 'Adam':
            lr = config.lr/(1 + args.lr_decay*step)

    # exclude data loading time for fair comparison
    epoch_end = time.time() - config.elapsed_time
    total_running_time += epoch_end - start
    config.elapsed_time = 0.0

    if val_batch is None:
```

Kypros Ioannou

```
        output_str = 'In epoch {} train loss with reg: {:.3f} | epoch time {:.3f}'\
            .format(epoch, cumulative_loss / cumulative_size, epoch_end-start)
    else:
        if test_network == None:
            val_loss, val_acc, _ = predict(
                sess,
                network=(x, y, loss_with_reg*batch_size, outputs),
                test_batch=val_batch,
                bsize=config.bsize
                )
        else:
            # A separat test network part have been done...
            val_loss, val_acc, _ = predict(
                sess,
                network=test_network,
                test_batch=val_batch,
                bsize=config.bsize
                )

        output_str = 'In epoch {} train loss with reg: {:.3f} | val loss with reg: {:.3f} | val accuracy: {:.3f}% | epoch time {:.3f}'\
            .format(epoch, cumulative_loss / cumulative_size, val_loss, val_acc*100, epoch_end-start)

        if val_acc > best_acc:
            best_acc = val_acc
            checkpoint_path = config.model_file
            save_path = saver.save(sess, checkpoint_path)
            print('Saved best model in {}'.format(save_path))

    print(output_str)
    if not config.screen_log_only:
        print(output_str, file=log_file)

if val_batch is None:
    checkpoint_path = config.model_file
    save_path = saver.save(sess, checkpoint_path)
```

Kypros Ioannou

```
        print('Model at the last iteration saved in
            {}\r\n'.format(save_path))
        output_str = 'total running time
            {:.3f}s'.format(total_running_time)
    else:
        output_str = 'Final acc: {:.3f}% | best acc {:.3f}% | total
            running time {:.3f}s'\
           .format(val_acc*100, best_acc*100, total_running_time)

    print(output_str)
    if not config.screen_log_only:
        print(output_str, file=log_file)
        log_file.close()

def newton_trainer(config, sess, network, full_batch, val_batch, saver,
    test_network):

    _, _, loss, outputs = network
    newton_solver = newton_cg(config, sess, outputs, loss)
    sess.run(tf.compat.v1.global_variables_initializer())

    print('-------------- initializing network by methods in He et al.
        (2015) --------------')
    param = tf.compat.v1.trainable_variables()
    sess.run(init_model(param))
    newton_solver.newton(full_batch, val_batch, saver, network,
        test_network)


def main():
    full_batch, num_cls, label_enum = read_data(train_texts,train_labels,
        dim=args.dim)
    if args.val_set is None:
        print('No validation set is provided. Will output model at the
            last iteration.')
        val_batch = None
    else:
        val_batch, _, _ = read_data(train_texts,train_labels,
            dim=args.dim, label_enum=label_enum)
```

```
    # val_batch, _, _ = read_data(test_texts,test_labels,
        dim=args.dim, label_enum=label_enum)

num_data = full_batch[0].shape[0]

config = ConfigClass(args, num_data, num_cls)

if isinstance(config.seed, int):
    tf.compat.v1.random.set_random_seed(config.seed)
    np.random.seed(config.seed)

if config.net in ('CNN_4layers', 'CNN_7layers', 'VGG11', 'VGG13',
     'VGG16','VGG19'):
    x, y, outputs = CNN(config.net, num_cls, config.dim)
    test_network = None
else:
    raise ValueError('Unrecognized training model')

if config.loss == 'MSELoss':
    loss = tf.reduce_sum(input_tensor=tf.pow(outputs-y, 2))
else:
    loss =
        tf.reduce_sum(input_tensor=tf.nn.softmax_cross_entropy_with_logits
        (logits=outputs, labels=y))

network = (x, y, loss, outputs)

sess_config = tf.compat.v1.ConfigProto()
sess_config.gpu_options.allow_growth = True

with tf.compat.v1.Session(config=sess_config) as sess:

    full_batch[0], mean_tr = normalize_and_reshape(full_batch[0],
        dim=config.dim, mean_tr=None)
    if val_batch is not None:
        val_batch[0], _ = normalize_and_reshape(val_batch[0],
            dim=config.dim, mean_tr=mean_tr)

    param = tf.compat.v1.trainable_variables()
```

Kypros Ioannou

```
        mean_param = tf.compat.v1.get_variable(name='mean_tr',
            initializer=mean_tr, trainable=False,
                validate_shape=True, use_resource=False)
        label_enum_var=tf.compat.v1.get_variable(name='label_enum',
            initializer=label_enum, trainable=False,
                validate_shape=True, use_resource=False)
        saver = tf.compat.v1.train.Saver(var_list=param+[mean_param])

        if config.optim in ('SGD', 'Adam'):
            gradient_trainer(
                config, sess, network, full_batch, val_batch, saver,
                    test_network)
        elif config.optim == 'NewtonCG':
            newton_trainer(
                config, sess, network, full_batch, val_batch, saver,
                    test_network=test_network)

main()

pred_args = ("--bsize 32 --valid_set ./" + "VALID_FILE" + " --train_set
    ./" + "TRAIN_FILE" +
                " --model ./saved_model/model.ckpt --dim " +
            str(15) + " " + str(20) + " 1").split()

#Train Model
train_results=acc_loss(train_text,train_labels)
#Shows Confusion Matrix for train data
confusion_matrix(train_labels,train_results)
#Test Model
test_results=acc_loss(test_texts,test_labels)
#Shows Confusion Matrix for test data
confusion_matrix(test_labels,test_results)
```

Kypros Ioannou

Create Folds for Cross-Validation Source Code

```
import numpy as np # linear algebra
import pandas as pd # data processing, CSV file I/O (e.g. pd.read_csv)
import os
import math
import pdb
from gensim.models import Word2Vec, KeyedVectors
from email.parser import Parser
from nltk.tokenize import word_tokenize
from nltk.corpus import stopwords
from string import punctuation
from tqdm.notebook import tqdm
from sklearn.model_selection import train_test_split
from statistics import mean
from gensim.models import KeyedVectors
from sklearn.model_selection import KFold, StratifiedKFold,
    cross_val_score
from sklearn import linear_model, tree, ensemble\
%load_ext autotime

ham_src=r'.\Enron Dataset\enron6\enron6\ham'
spam_src=r'.\Enron Dataset\enron6\enron6\spam'

#Load every ham email on ham list
def data_preprocessing(src):
    clean_list=[]
    directory=src
    for filename in os.listdir(directory):
        try:
             with open(os.path.join(directory, filename), "r") as f:
                data = f.read()
                email = Parser().parsestr(data)
                clean_list.append(email.as_string())
        except:
            print("Unclear File",filename)
            continue
    return clean_list

#Remove stopwords and puncutation. Also tokenize each email.
```

Kypros Ioannou

```python
def preprocess_corpus(texts):
    #importing stop words like in, the, of so that these can be removed
        from texts
    #as these words dont help in determining the classes(Whether a
        sentence is toxic or not)
    mystopwords = set(stopwords.words("english"))
    def remove_stops_digits(tokens):
        #Nested function that lowercases, removes stopwords and digits
            from a list of tokens
        return [token.lower() for token in tokens if token not in
            mystopwords and not token.isdigit()
               and token not in punctuation]
    #This return statement below uses the above function and tokenizes
        output further.
    return [remove_stops_digits(word_tokenize(text)) for text in
        tqdm(texts)]

#Function that takes in the input text dataset in form of list of lists
    where each sentence is a list of words all the sentences are
#inside a list
def embedding_feats(list_of_lists, DIMENSION, w2v_model):
    zeros_vector = np.zeros(DIMENSION)
    feats = []
    missing = set()
    missing_sentences = set()
    #Traverse over each sentence
    for tokens in tqdm(list_of_lists):
        # Initially assign zeroes as the embedding vector for the
            sentence
        feat_for_this = zeros_vector
        #Count the number of words in the embedding for this sentence
        count_for_this = 0
        #Traverse over each word of a sentence
        for token in tokens:
            #Check if the word is in the embedding vector
            if token in w2v_model:
                #Add the vector of the word to vector for the sentence
                feat_for_this += w2v_model[token]
                count_for_this +=1
```

Kypros Ioannou

```
            #Else assign the missing word to missing set just to have a
                look at it
            else:
                missing.add(token)
        #If no words are found in the embedding for the sentence
        if count_for_this == 0:
            #Assign all zeroes vector for that sentence
            feats.append(feat_for_this)
            #Assign the missing sentence to missing_sentences just to
                have a look at it
            missing_sentences.add(' '.join(tokens))
        #Else take average of the values of the embedding for each word
            to get the embedding of the sentence
        else:
            feats.append(feat_for_this/count_for_this)
    return feats, missing, missing_sentences

#Call data_preprocessing function to load the ham and spam emails
ham_list=data_preprocessing(ham_src)
spam_list=data_preprocessing(spam_src)
#Remove emails with less than 5 words
ham_list=[x for x in ham_list if len(x) >= 5]
spam_list=[x for x in spam_list if len(x) >= 5]
#assign the class to each dataframe (HAM->0 Spam->1)
df_ham = pd.DataFrame({"Mails":ham_list})
df_ham['Class']=0
df_spam = pd.DataFrame({"Mails":spam_list})
df_spam['Class']=1
#Drop nan values and duplicates
df_ham=df_ham.dropna()
df_spam=df_spam.dropna()
df_ham=df_ham.drop_duplicates()
df_spam=df_spam.drop_duplicates()
#Combine datasets
data_set=pd.concat([df_ham, df_spam], ignore_index=True)
#set which column is the class and which are the features
train_texts = list(data_set["Mails"].values)
train_labels=data_set['Class'].values
#save data in CSV format
data_set.to_csv("Enron6_0.csv",index=False)
```

Kypros Ioannou

```
filename='Enron6_0.csv'
data_set = pd.read_csv(filename)
train_texts = list(data_set["Mails"].values)
train_labels=data_set['Class'].values
train_texts_processed = preprocess_corpus(train_texts)


#load pretrain wor2vec model from GoogleNews
w2v_google_news =
    KeyedVectors.load_word2vec_format('\GoogleNews-vectors-negative300.bin',
    binary=True)

#Remove the subject and cc and also delete emails tha have less than 2
    words
stopwords = ['subject', 'cc']
results=[]
for x in train_texts_processed:
    resultwords = [word for word in x if word.lower() not in stopwords]
    results.append(resultwords)
results_new=[]
for x in results:
    resultwords = [word for word in x if len(word)>1]
    results_new.append(resultwords)

train_vectors, missing, missing_sentences = embedding_feats(results_new,
    300, w2v_google_news)
#split dataset into train and test data (80% train and 20% testing)
train_data, val_data, train_cats, val_cats =
    train_test_split(train_vectors, train_labels,test_size=0.20)
f = pd.DataFrame(train_data)
df2=pd.DataFrame(val_data)
data_set=pd.concat([df, df2], ignore_index=True)
classes=np.concatenate((train_cats, val_cats))
data_set['Class']=classes
y = data_set.Class # Target variable
X = data_set.copy()
# Lets split the data into 5 folds.
# We will use this 'kf'(KFold splitting stratergy) object as input to
    cross_val_score() method
kf =KFold(n_splits=5)
```


Kypros Ioannou

```
cnt = 1
# split() method generate indices to split data into training and test
    set.
for train_index, test_index in kf.split(X, y):
    print(f'Fold:{cnt}, Train set: {len(train_index)}, Test
        set:{len(test_index)}')
    p=data_set.iloc[train_index]
    p.to_csv(f'Fold{cnt}_6.csv'.format(),index=False)
    p=data_set.iloc[test_index]
    p.to_csv(f'Valid{cnt}_6.csv'.format(),index=False)
    cnt += 1
```

Kypros Ioannou

CNN with Gradient Descent Source Code

```
# Importing libraries
import tensorflow as tf
import keras
from keras.datasets import imdb
from keras.models import Sequential
from keras.layers import Dense
from keras.layers import Flatten
from keras.layers import Dropout
from keras.layers.convolutional import Conv2D
from keras.layers.convolutional import MaxPooling2D
from keras.layers.embeddings import Embedding
from keras.preprocessing import sequence
from keras.optimizers import Adam
from keras.optimizers import SGD
import pandas as pd
import numpy as np
from keras.regularizers import l2
from statistics import mean
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.metrics import classification_report, confusion_matrix,
    accuracy_score, roc_auc_score
%load_ext autotime

#load train and testing dataset
fold = pd.read_csv(r'.\Validation Fold\D1Folds\Fold5_1.csv')
test_fold=pd.read_csv(r'.\Validation Fold\D1Folds\Valid5_1.csv')

#Split each dataset to feature columns and class
train_labels=fold.Class
train_texts=fold.drop(['Class'], axis=1)
test_labels=test_fold.Class
test_texts=test_fold.drop(['Class'], axis=1)
#convert them in numpy format
train_labels=train_labels.to_numpy()
test_labels=test_labels.to_numpy()
train_texts=train_texts.to_numpy()
```

Kypros Ioannou

```
test_texts=test_texts.to_numpy()

# Convert each dataset in order to feed them to CNN
def preprocess_cnn(data,labels_class):
    #features
    features = data.reshape(data.shape[0], -1)
    _IMAGE_HEIGHT, _IMAGE_WIDTH, _IMAGE_CHANNELS = 15,20,1
    zero_to_append = np.zeros((features.shape[0],_IMAGE_CHANNELS*
        _IMAGE_HEIGHT*_IMAGE_WIDTH-np.prod(features.shape[1:])))
    features = np.append(features, zero_to_append, axis=1)
    _IMAGE_HEIGHT, _IMAGE_WIDTH, _IMAGE_CHANNELS = 15,20,1
    images_shape = [features.shape[0], _IMAGE_CHANNELS, _IMAGE_HEIGHT,
        _IMAGE_WIDTH]
    # images normalization and zero centering
    features = features.reshape(images_shape[0], -1)
    features = features/255.0
    mean_tr = features.mean(axis=0)
    features = features - mean_tr
    features = features.reshape(images_shape)
    # Tensorflow accepts data shape: B x H x W x C
    features = np.transpose(features, (0, 2, 3, 1))

    #LABELS
    data_class=list(map(lambda el:[el], labels_class))
    data_class=np.array(labels_class)
    data_class = labels_class.reshape(-1)

    #TRAIN DATA
    label_enum, labels = np.unique(data_class, return_inverse=True)
    num_cls = data_class.max()+1

    forward_map = dict(zip(label_enum, np.arange(2)))
    data_class = np.expand_dims(data_class, axis=1)
    data_class = np.apply_along_axis(lambda x:forward_map[x[0]], axis=1,
        arr=data_class)

    #convert groundtruth to one-hot encoding
#     data_class = np.eye(2)[data_class]
    data_class = data_class.astype('float32')
    return features,data_class
```

Kypros Ioannou

```
train_data,class_data=preprocess_cnn(train_texts,train_labels)
test_data,class_test=preprocess_cnn(test_texts,test_labels)

cnn_model = Sequential([
    keras.layers.Conv2D(128, [5, 5],kernel_regularizer=l2(0.03),
        padding='same', activation=tf.nn.relu6, input_shape=[15,20,1]),
    keras.layers.MaxPool2D([2, 2], strides=2),
    keras.layers.Dropout(0.5),
    keras.layers.Flatten(),
    keras.layers.Dense(2,activation='softmax')])

cnn_model.summary()
cnn_model.compile(loss ='sparse_categorical_crossentropy',
    optimizer=SGD(lr=0.5),metrics =['accuracy'])

model_checkpoint_callback = tf.keras.callbacks.ModelCheckpoint(
    filepath=r'.\Validation Fold',
    save_weights_only=True,
    monitor='val_accuracy',
    mode='max',
    save_best_only=True)

history = cnn_model.fit(
    train_data,
    class_data,
    batch_size=32,
    epochs=1000,
    verbose=1,
    validation_data=(test_data,class_test),
    callbacks=[model_checkpoint_callback]
)

#Confusion Matrix for train data
train_predict = cnn_model.predict_classes(train_data)
fig = plt.figure(figsize=(10,4))
heatmap = sns.heatmap(data = pd.DataFrame(confusion_matrix(class_data,
    train_predict)), annot = True, fmt = "d",
    cmap=sns.color_palette("Reds", 50))
```

Kypros Ioannou

```
heatmap.yaxis.set_ticklabels(heatmap.yaxis.get_ticklabels(), rotation=0,
    ha='right', fontsize=14)
heatmap.xaxis.set_ticklabels(heatmap.xaxis.get_ticklabels(),
    rotation=45, ha='right', fontsize=14)
plt.ylabel('Ground Truth')
plt.xlabel('Prediction')
plt.show()
```

Kypros Ioannou

Appendix B - Confusion Matrices

In this section we present all confusion matrices used while training our model. The first subsection will show all of the matrices used when feeding the entire dataset to the model. The second subsection will present all confusion matrices which were omitted from Chapter 4 for the sake of brevity. Finally, we will show confusion matrices from training and testing the Convolutional Neural Network, as well as the accuracy and Spam/Ham Recall for every fold for the classic CNN.

Confusion Matrices for the Whole Dataset

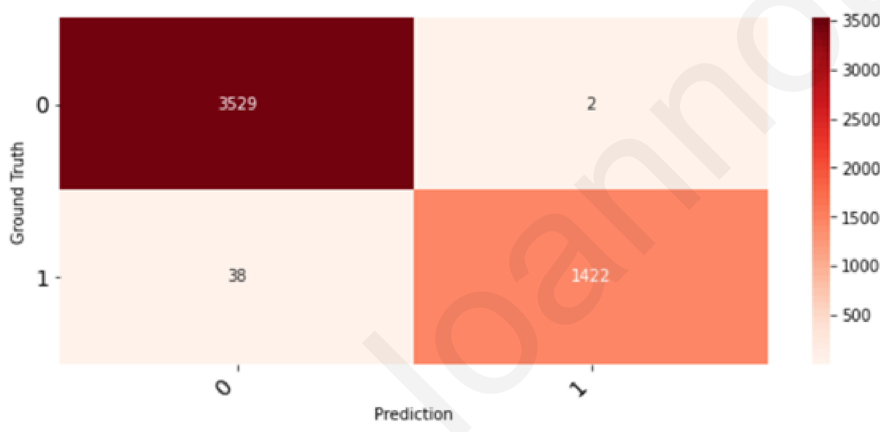


Figure 5.1: Confusion Matrix for the first dataset.

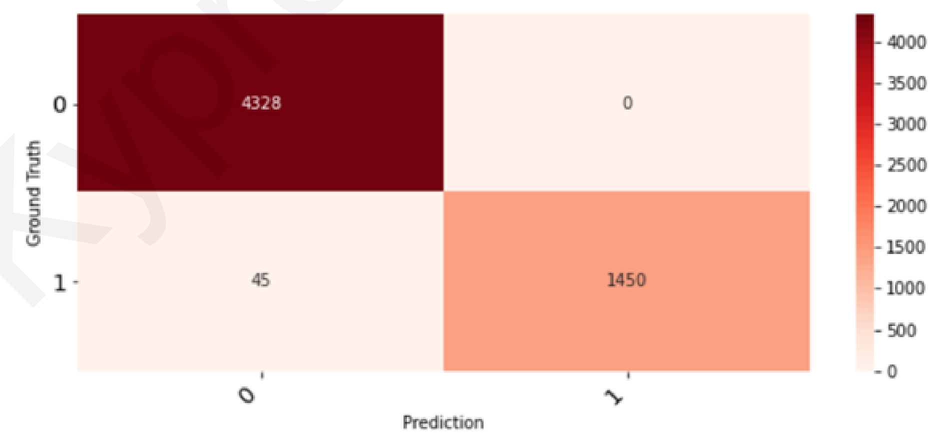


Figure 5.2: Confusion Matrix for the second dataset.

Kypros Ioannou

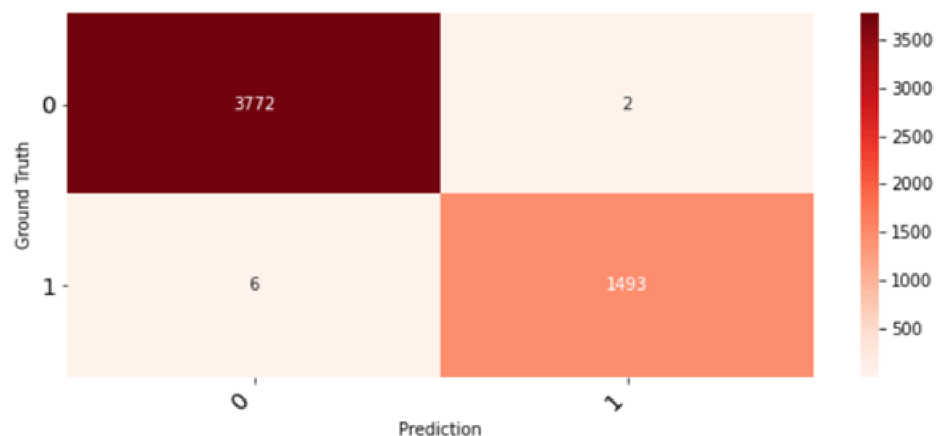


Figure 5.3: Confusion Matrix for the third dataset.

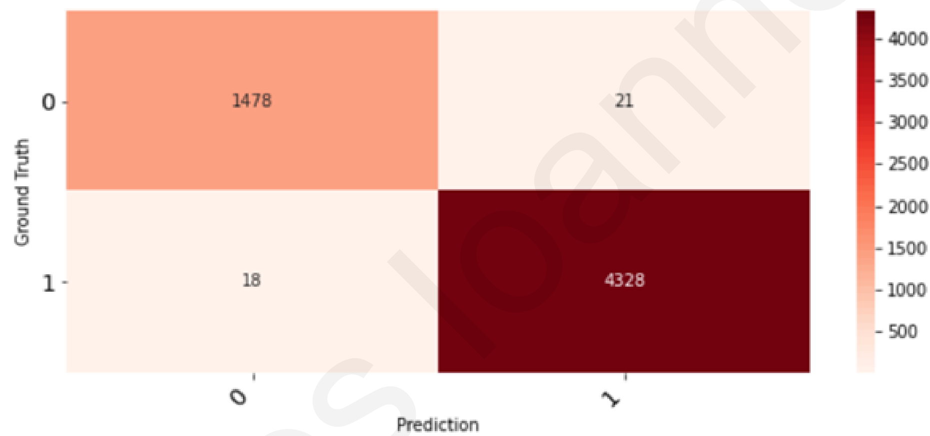


Figure 5.4: Confusion Matrix for the fourth dataset.

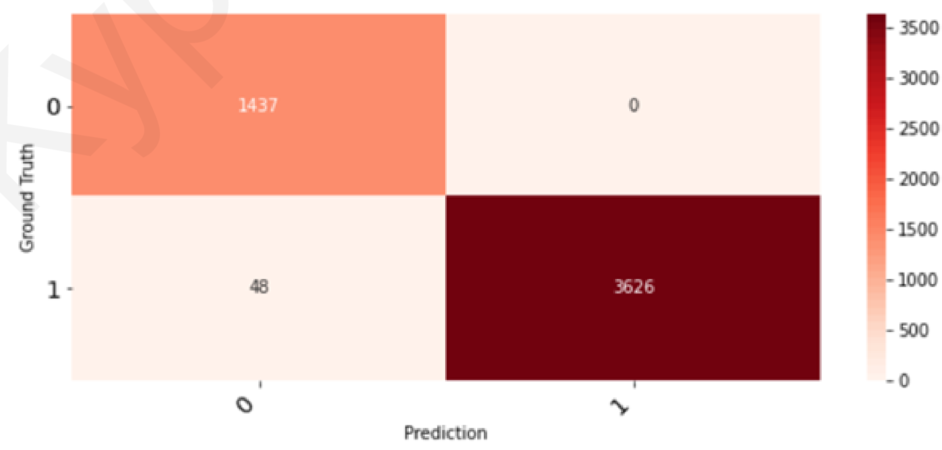


Figure 5.5: Confusion Matrix for the fifth dataset.

Kypros Ioannou

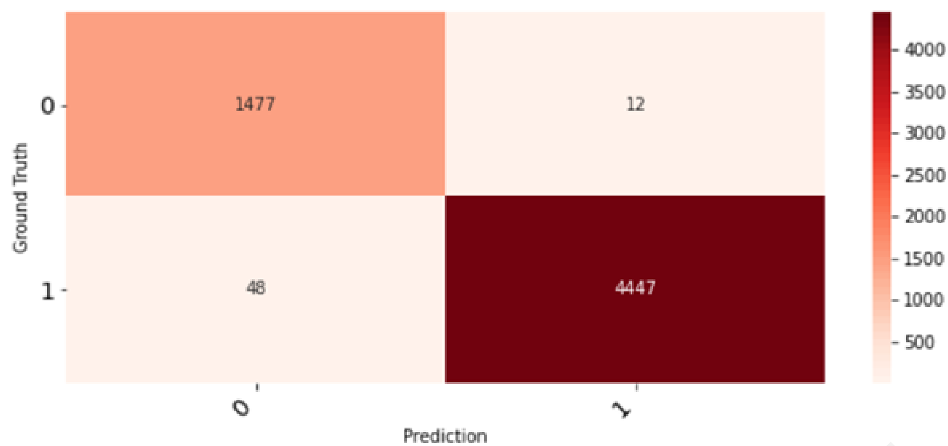


Figure 5.6: Confusion Matrix for the sixth dataset.

Remaining Confusion Matrices from Cross-Validation

First Dataset

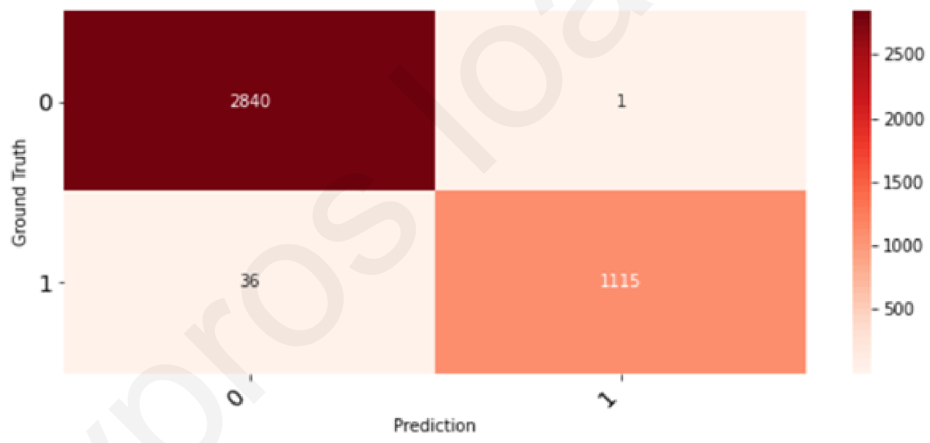


Figure 5.7: Dataset 1 Fold 1 Train Confusion Matrix

Kyproslоannou

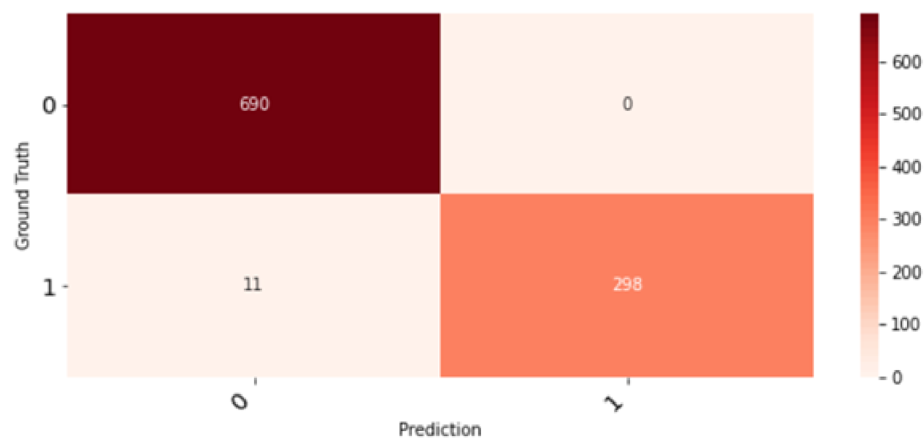


Figure 5.8: Dataset 1 Fold 1 Validation Confusion Matrix

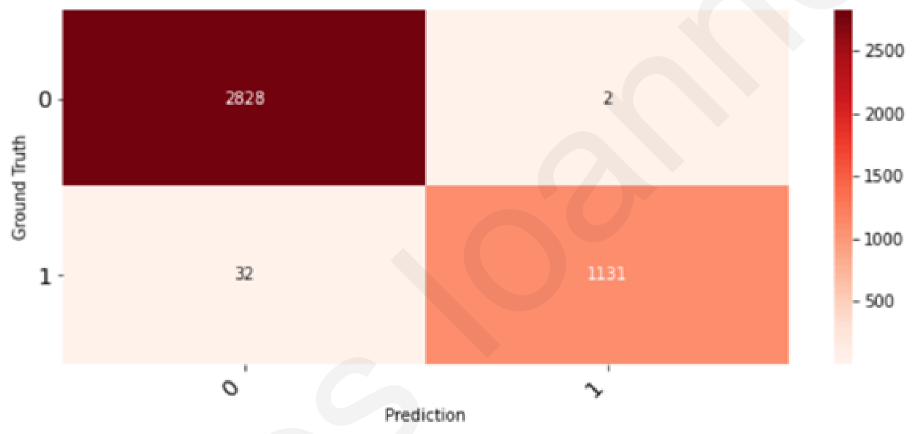


Figure 5.9: Dataset 1 Fold 3 Train Confusion Matrix

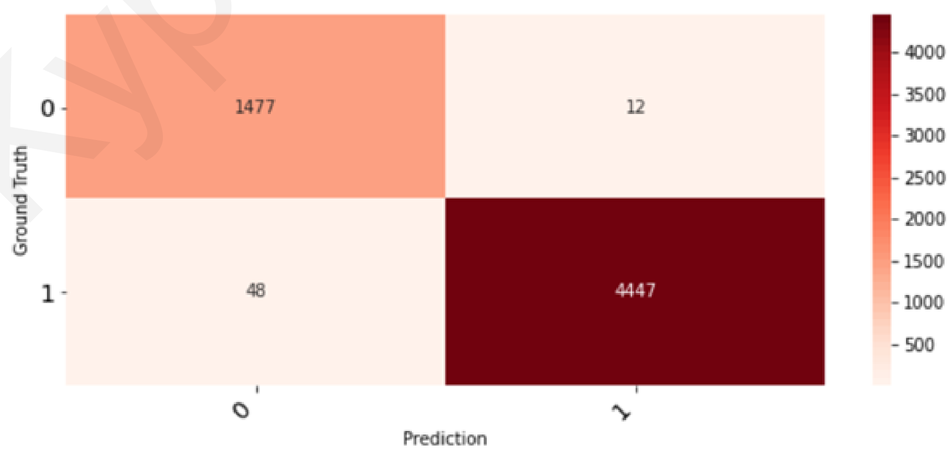


Figure 5.10: Dataset 1 Fold 3 Validation Confusion Matrix

Kypros Ioannou

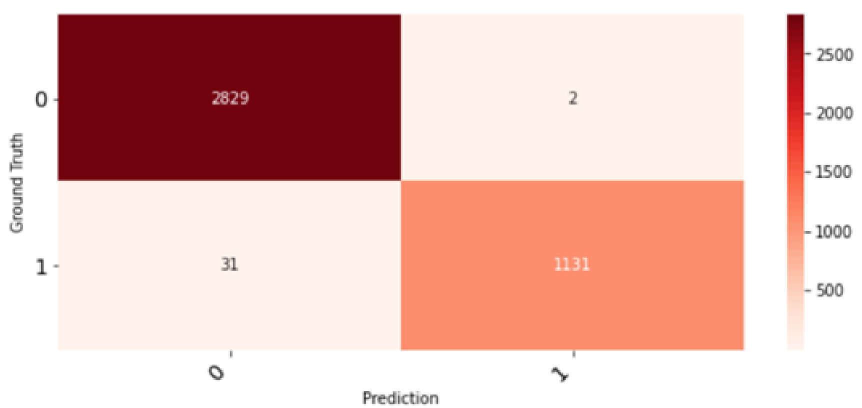


Figure 5.11: Dataset 1 Fold 4 Train Confusion Matrix

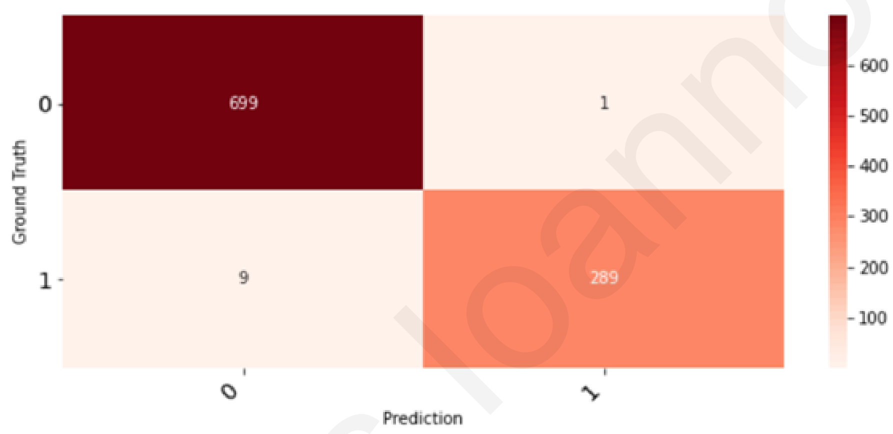


Figure 5.12: Dataset 1 Fold 4 Validation Confusion Matrix

Second Dataset

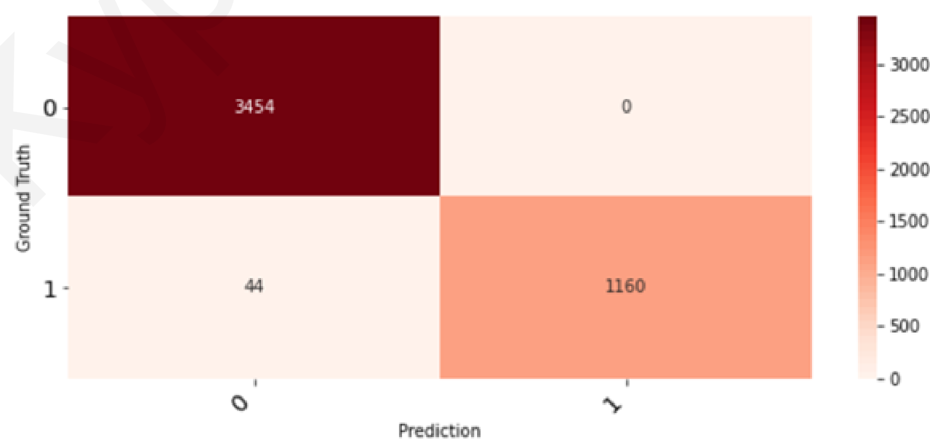


Figure 5.13: Dataset 2 Fold 1 Train Confusion Matrix

Kypros Ioannou

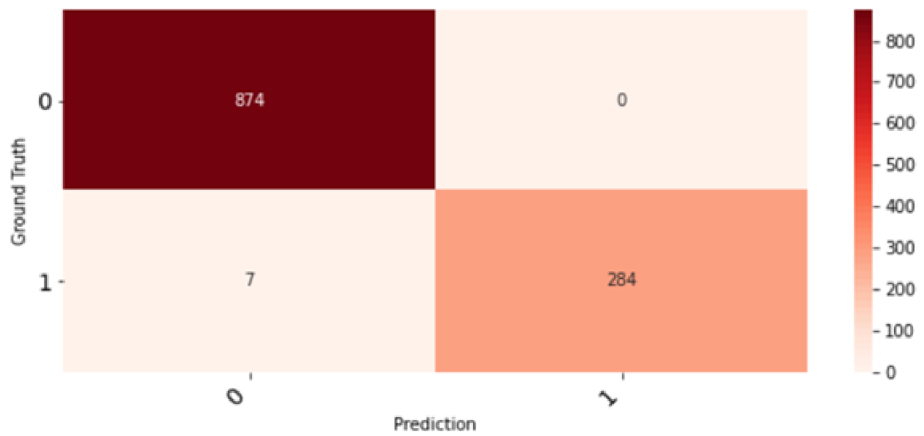


Figure 5.14: Dataset 2 Fold 1 Validation Confusion Matrix

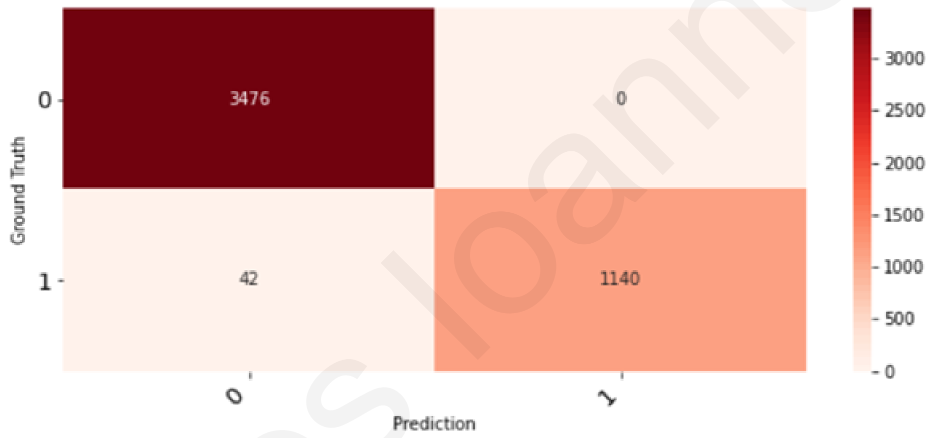


Figure 5.15: Dataset 2 Fold 3 Train Confusion Matrix

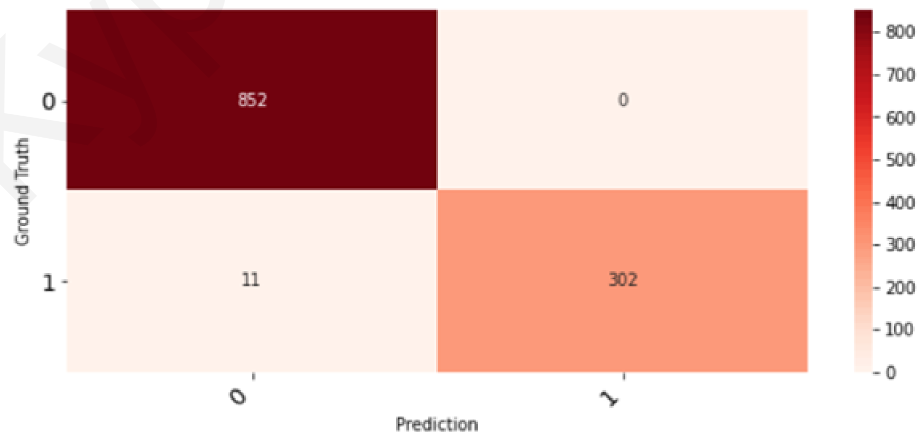


Figure 5.16: Dataset 2 Fold 3 Validation Confusion Matrix

Kypros Ioannou

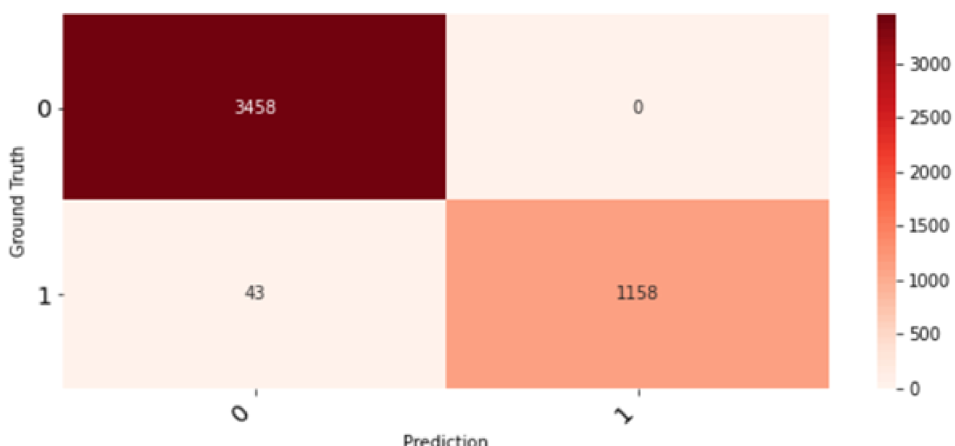


Figure 5.17: Dataset 2 Fold 4 Train Confusion Matrix

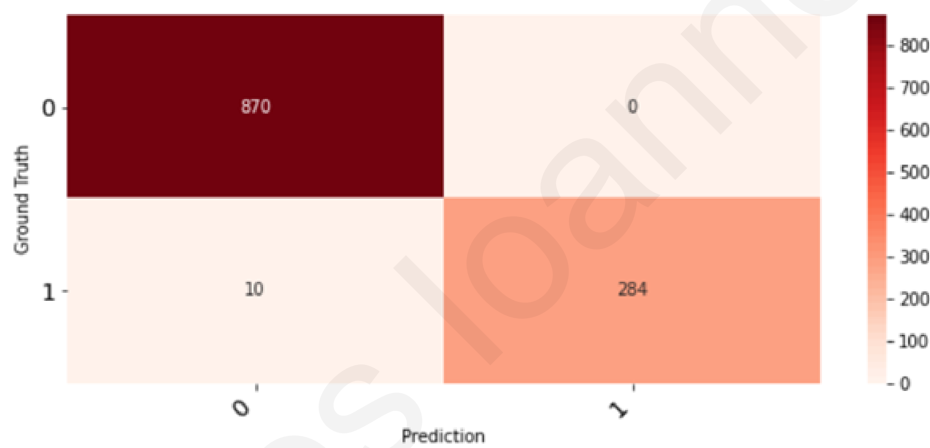


Figure 5.18: Dataset 2 Fold 4 Validation Confusion Matrix

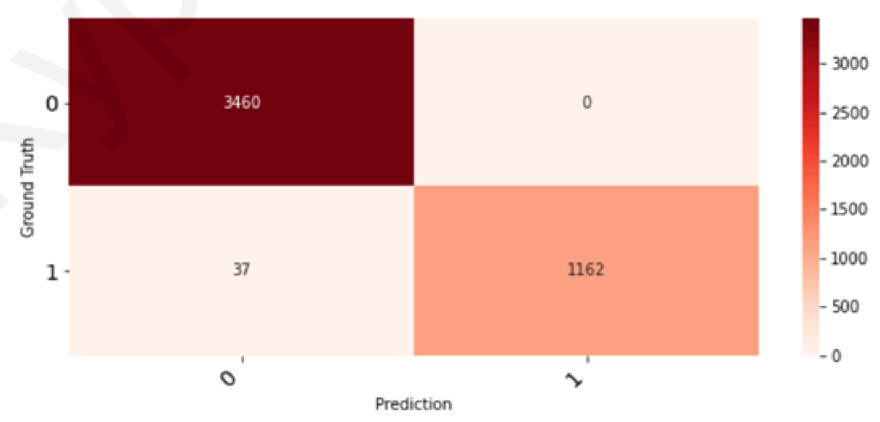


Figure 5.19: Dataset 2 Fold 5 Train Confusion Matrix

Kypros Ioannou

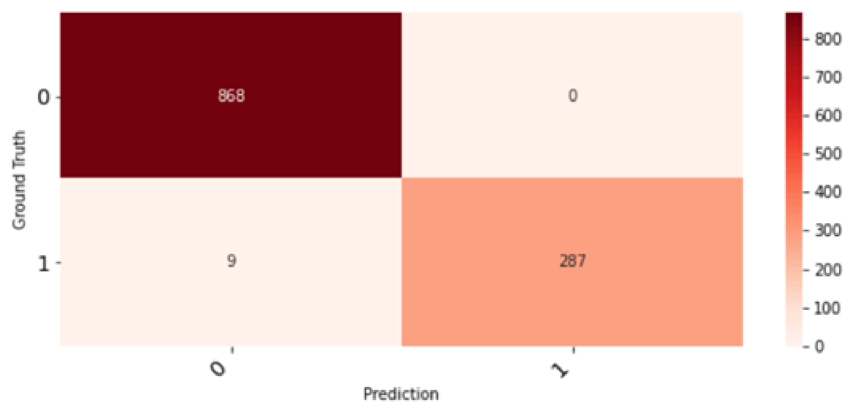


Figure 5.20: Dataset 2 Fold 5 Validation Confusion Matrix

Third Dataset

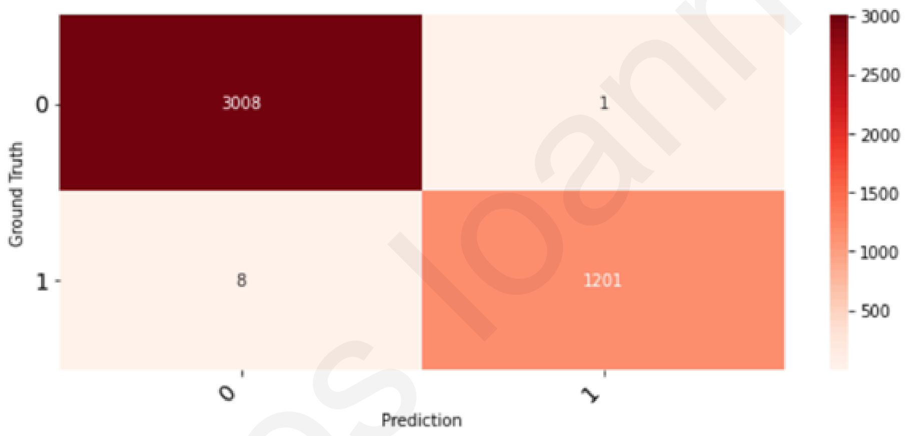


Figure 5.21: Dataset 3 Fold 1 Train Confusion Matrix

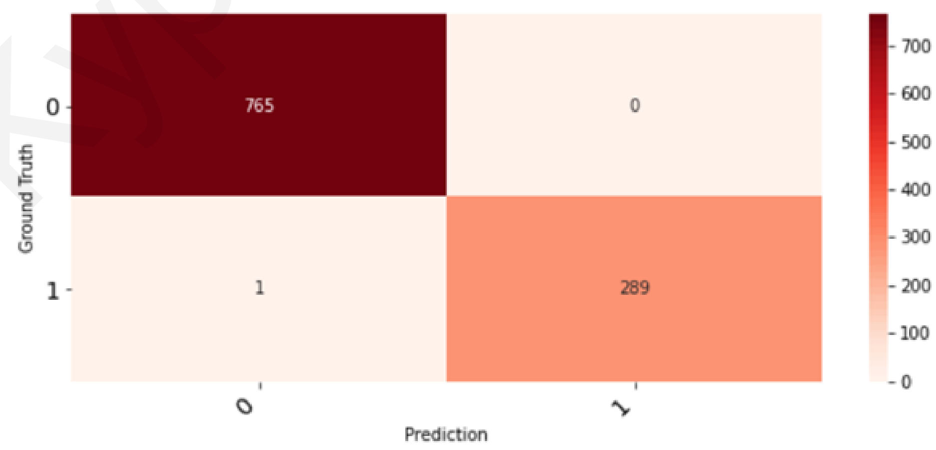


Figure 5.22: Dataset 3 Fold 1 Validation Confusion Matrix

Kypros Ioannou

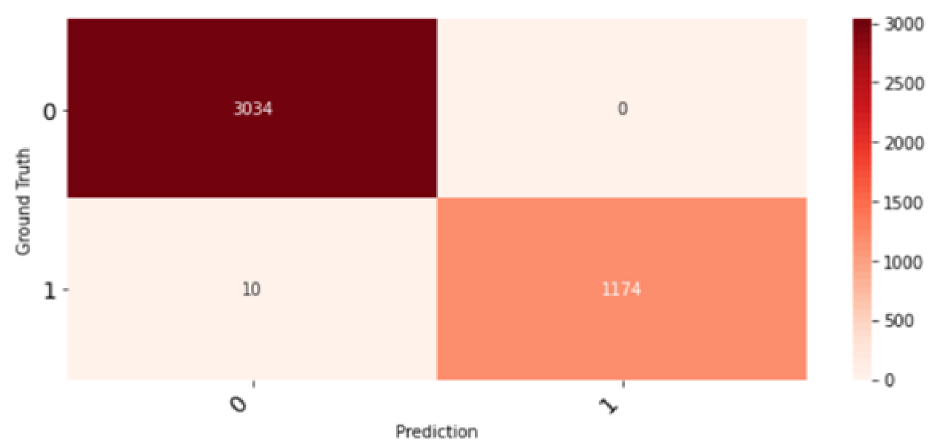


Figure 5.23: Dataset 3 Fold 2 Train Confusion Matrix

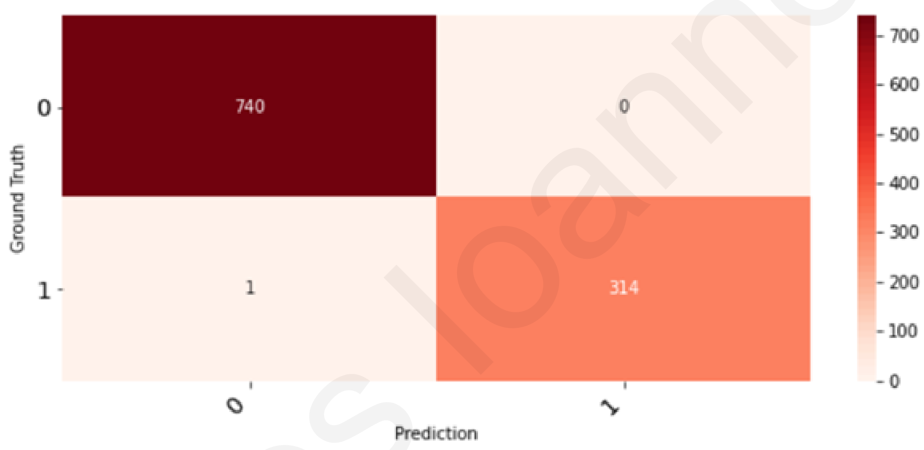


Figure 5.24: Dataset 3 Fold 2 Validation Confusion Matrix

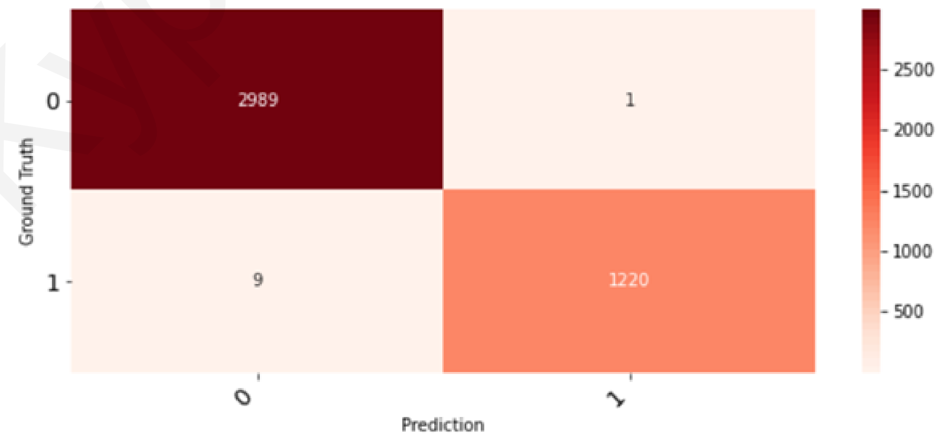


Figure 5.25: Dataset 3 Fold 4 Train Confusion Matrix

Kypros Ioannou

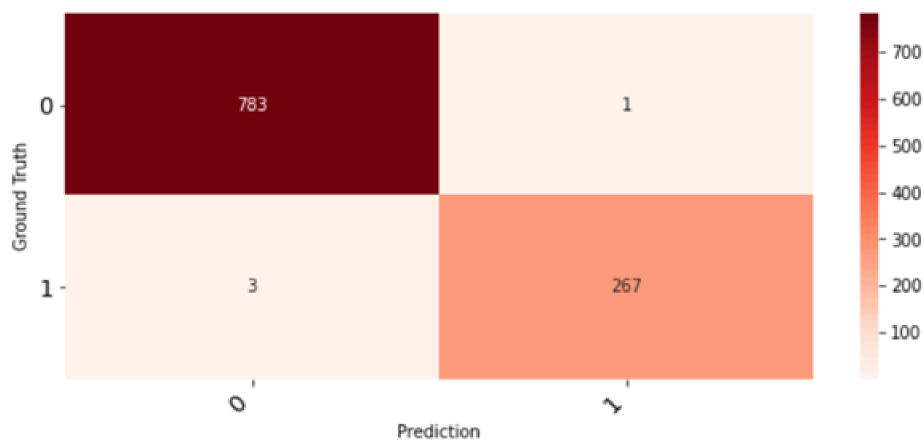


Figure 5.26: Dataset 3 Fold 4 Validation Confusion Matrix

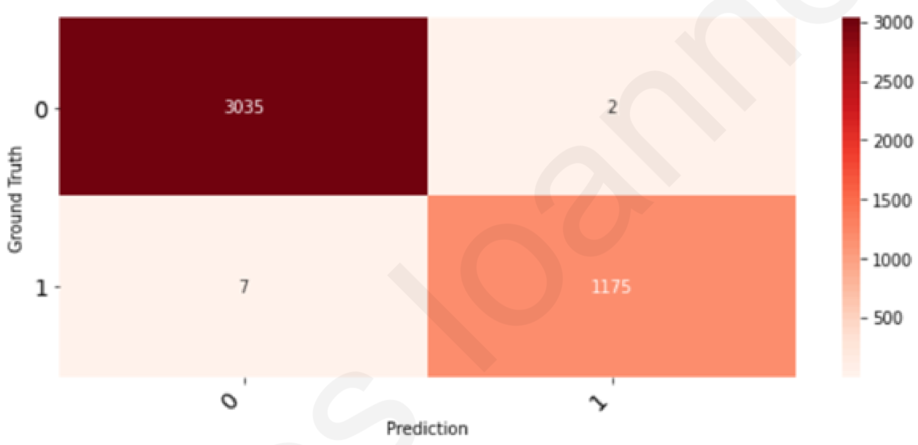


Figure 5.27: Dataset 3 Fold 5 Train Confusion Matrix

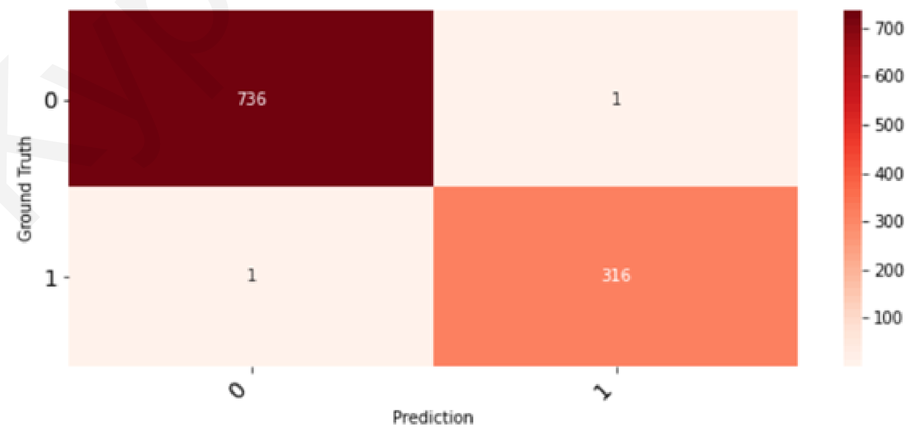


Figure 5.28: Dataset 3 Fold 5 Validation Confusion Matrix

Kypros Ioannou

Fourth Dataset

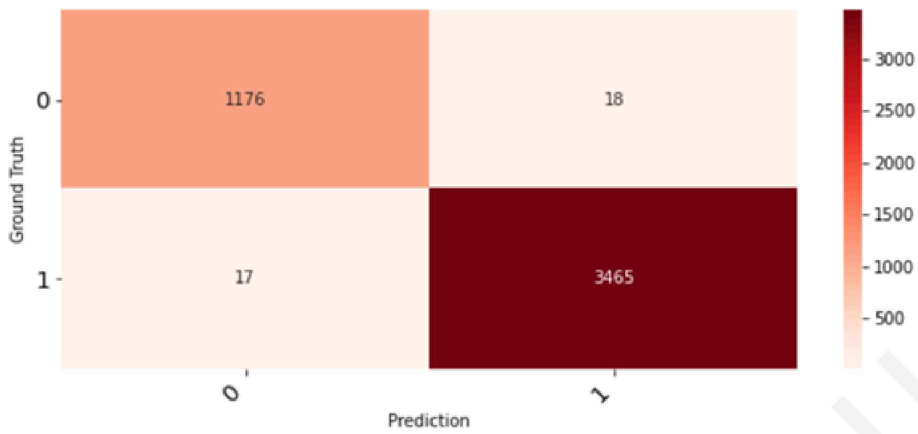


Figure 5.29: Dataset 4 Fold 1 Train Confusion Matrix

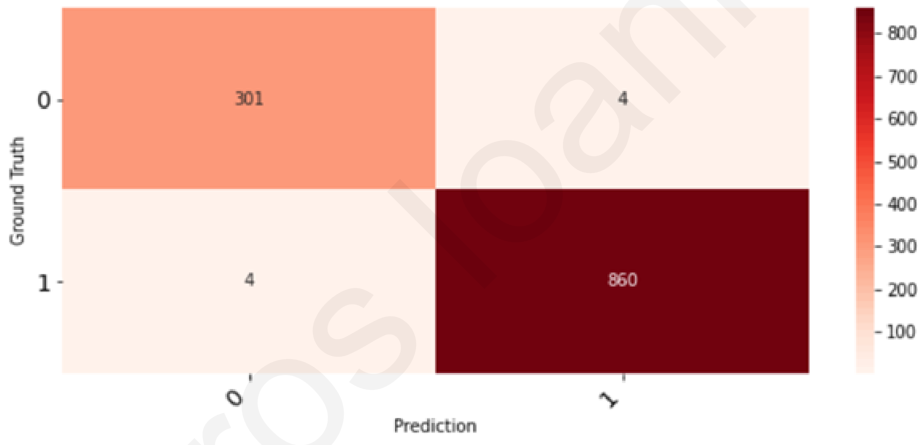


Figure 5.30: Dataset 4 Fold 1 Validation Confusion Matrix

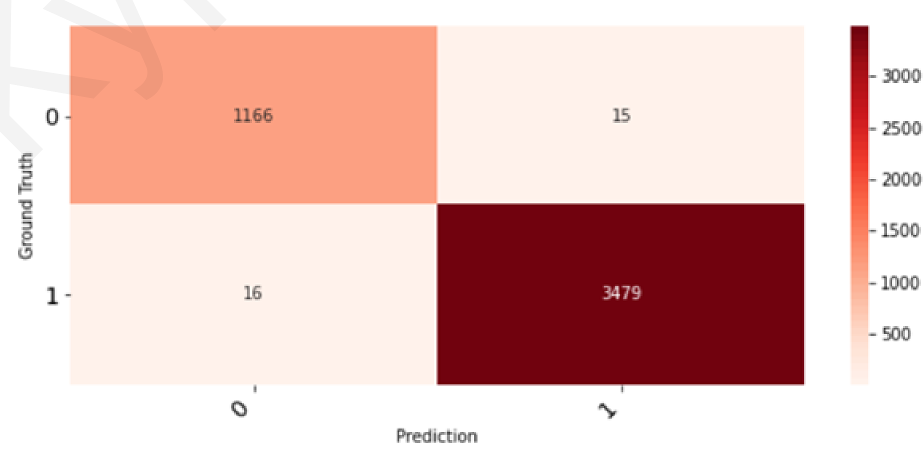


Figure 5.31: Dataset 4 Fold 2 Train Confusion Matrix

Kypros Ioannou

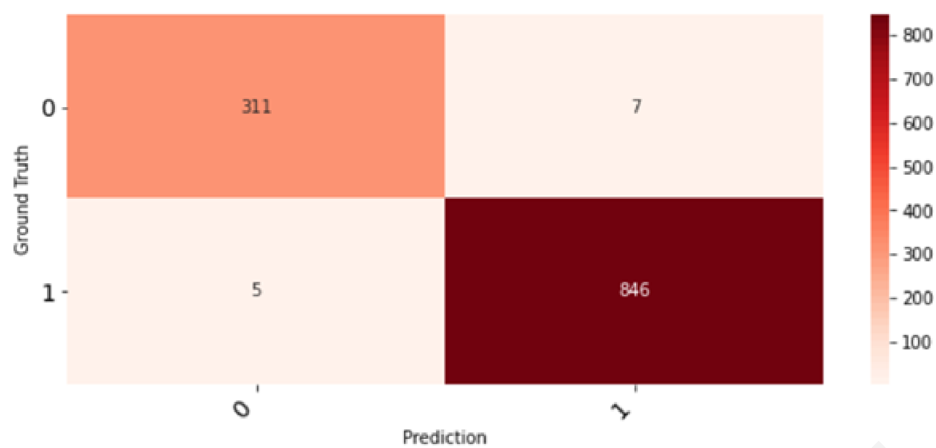


Figure 5.32: Dataset 4 Fold 2 Validation Confusion Matrix

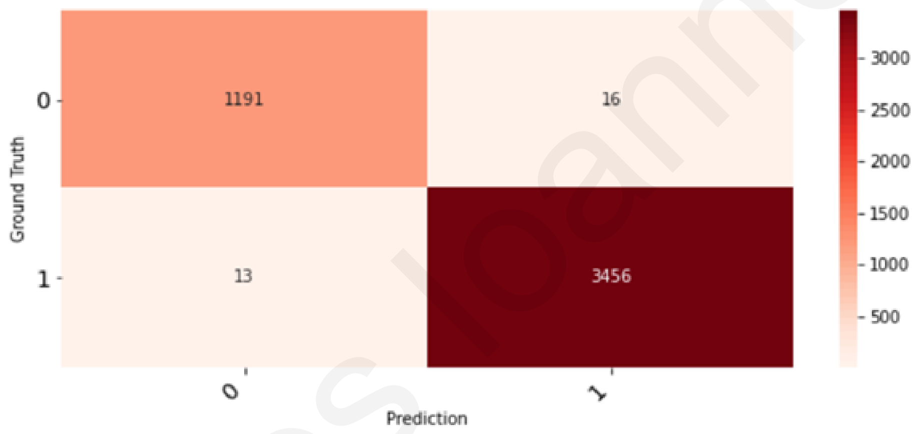


Figure 5.33: Dataset 4 Fold 5 Train Confusion Matrix

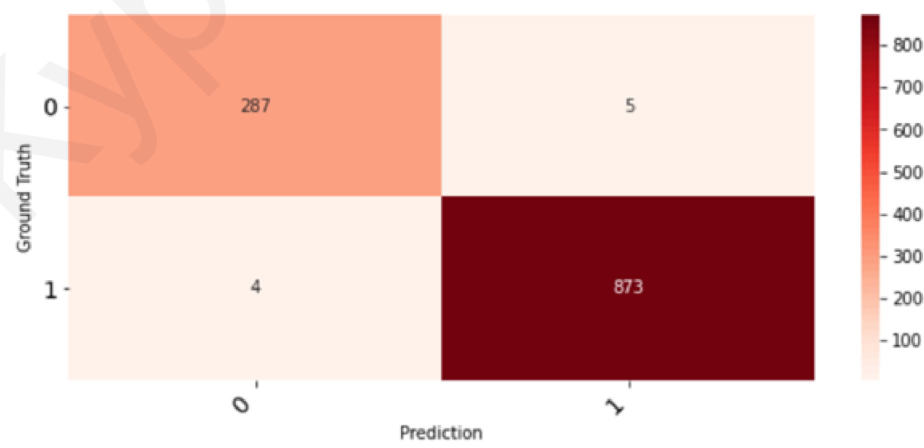


Figure 5.34: Dataset 4 Fold 5 Validation Confusion Matrix

Kypros Ioannou

Fifth Dataset

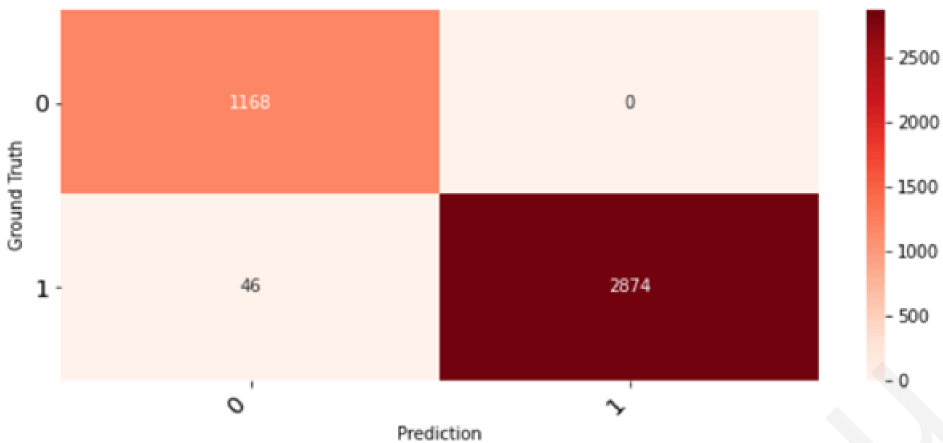


Figure 5.35: Dataset 5 Fold 1 Train Confusion Matrix

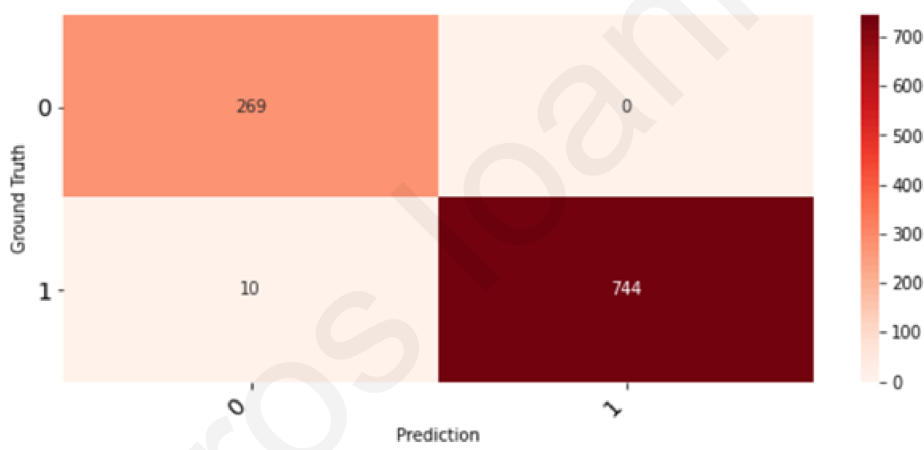


Figure 5.36: Dataset 5 Fold 1 Validation Confusion Matrix

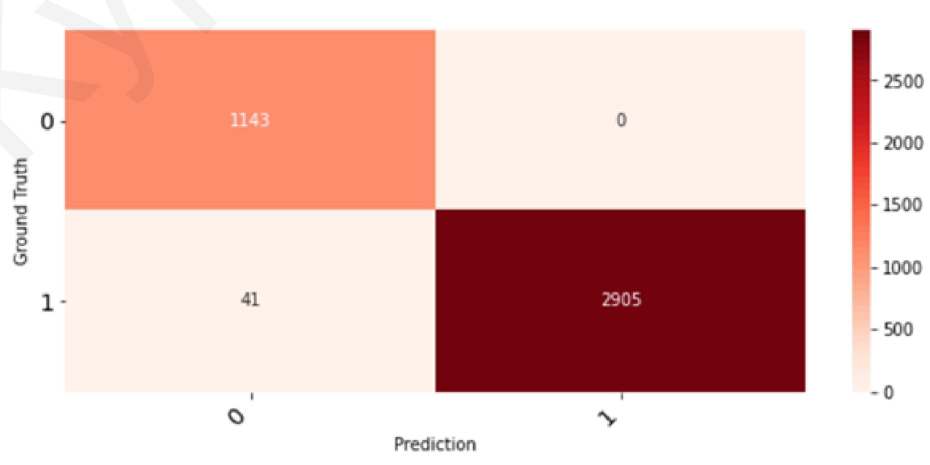


Figure 5.37: Dataset 5 Fold 2 Train Confusion Matrix

Kypros Ioannou

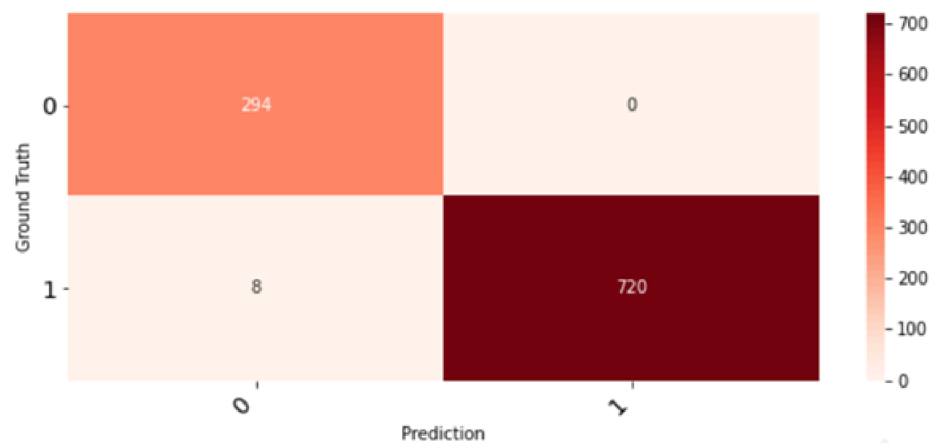


Figure 5.38: Dataset 5 Fold 2 Validation Confusion Matrix

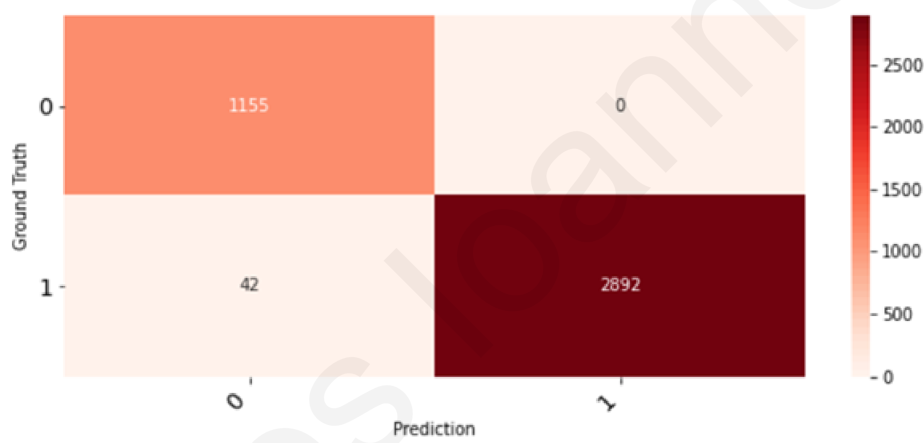


Figure 5.39: Dataset 5 Fold 4 Train Confusion Matrix

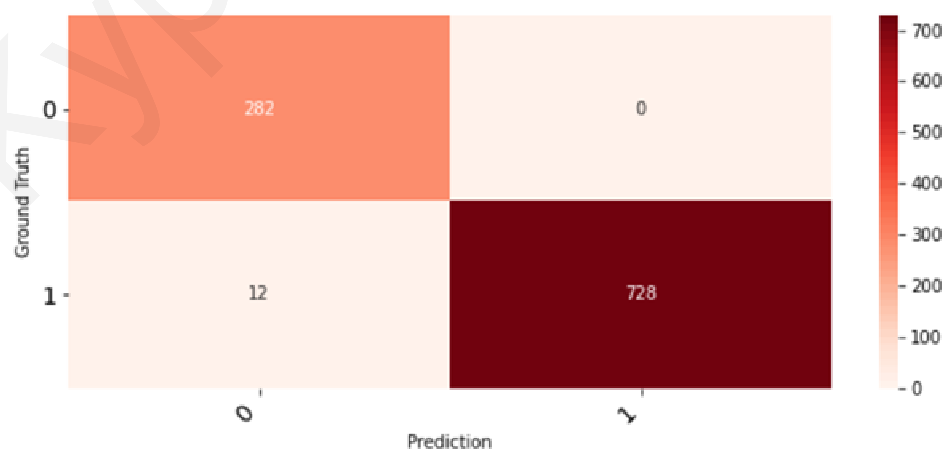


Figure 5.40: Dataset 5 Fold 4 Validation Confusion Matrix

Kypros Ioannou

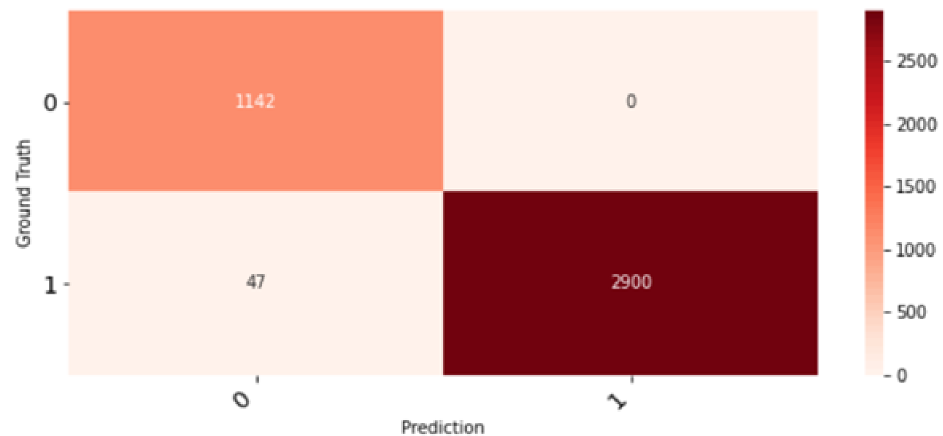


Figure 5.41: Dataset 5 Fold 5 Train Confusion Matrix

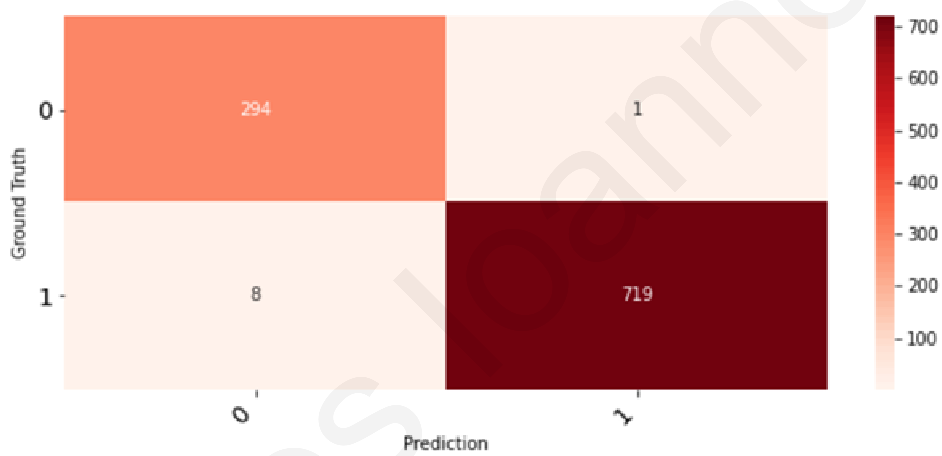


Figure 5.42: Dataset 5 Fold 5 Validation Confusion Matrix

Sixth Dataset

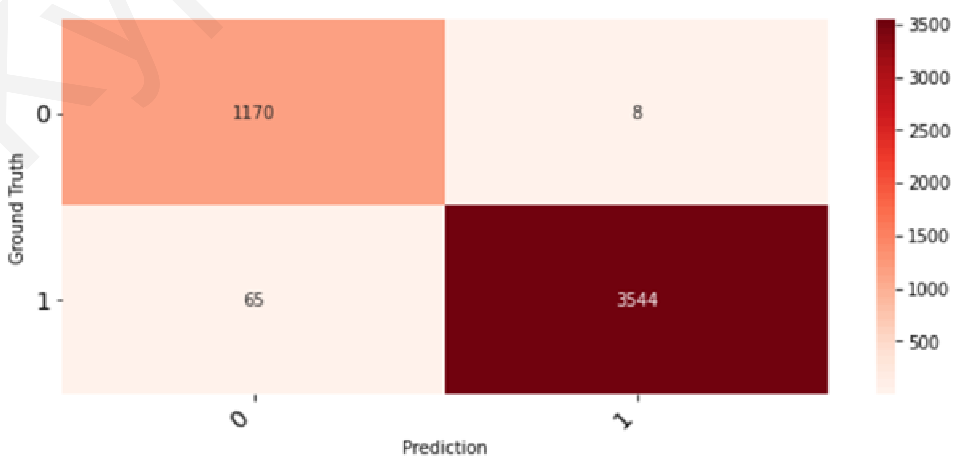


Figure 5.43: Dataset 6 Fold 1 Train Confusion Matrix

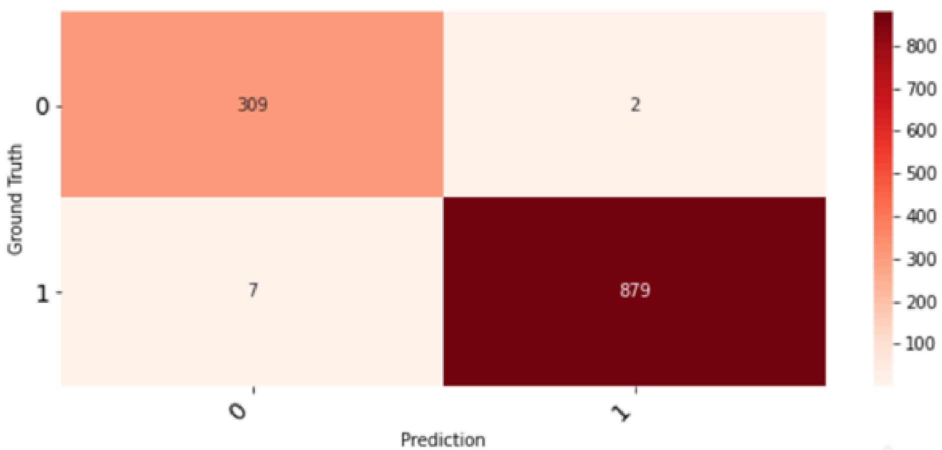


Figure 5.44: Dataset 6 Fold 1 Validation Confusion Matrix

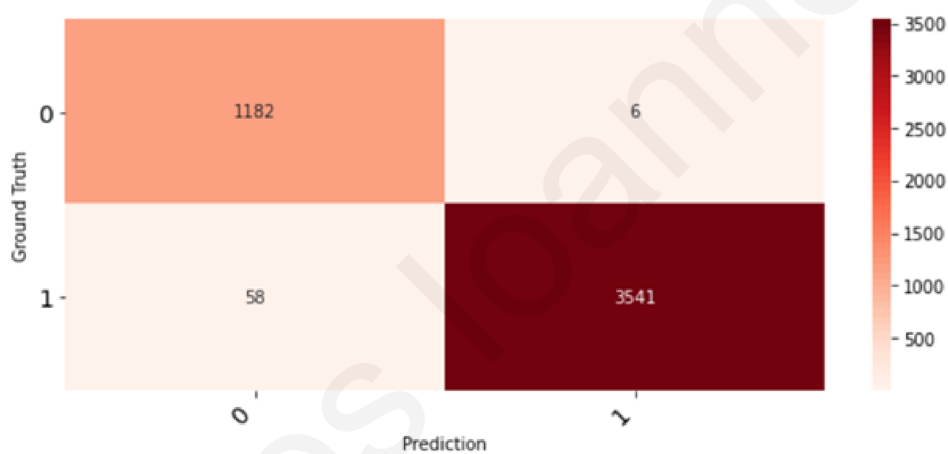


Figure 5.45: Dataset 6 Fold 3 Train Confusion Matrix

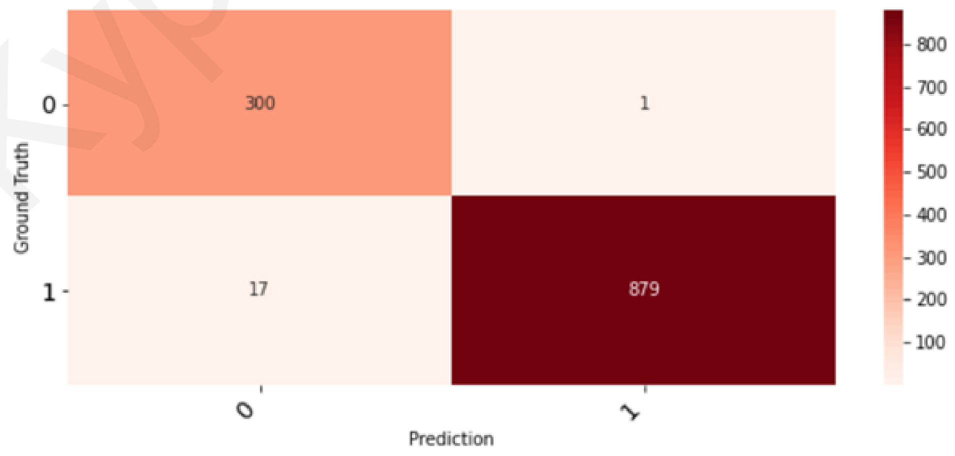


Figure 5.46: Dataset 6 Fold 3 Validation Confusion Matrix

Kypros Ioannou

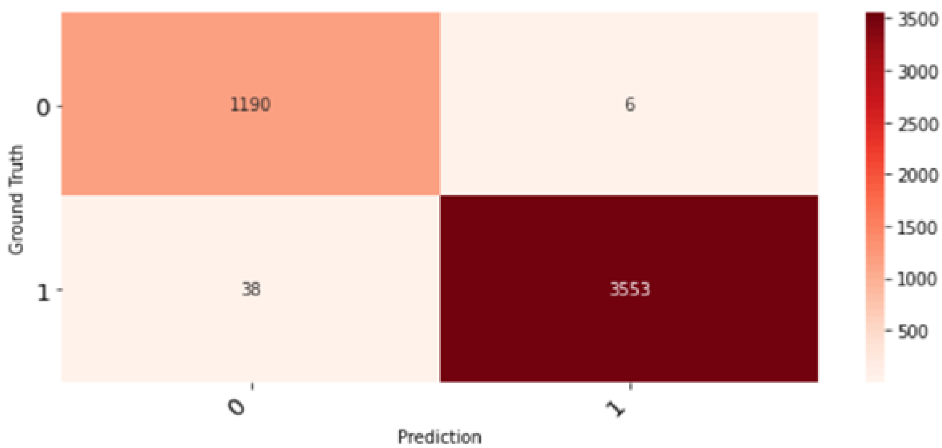


Figure 5.47: Dataset 6 Fold 4 Train Confusion Matrix

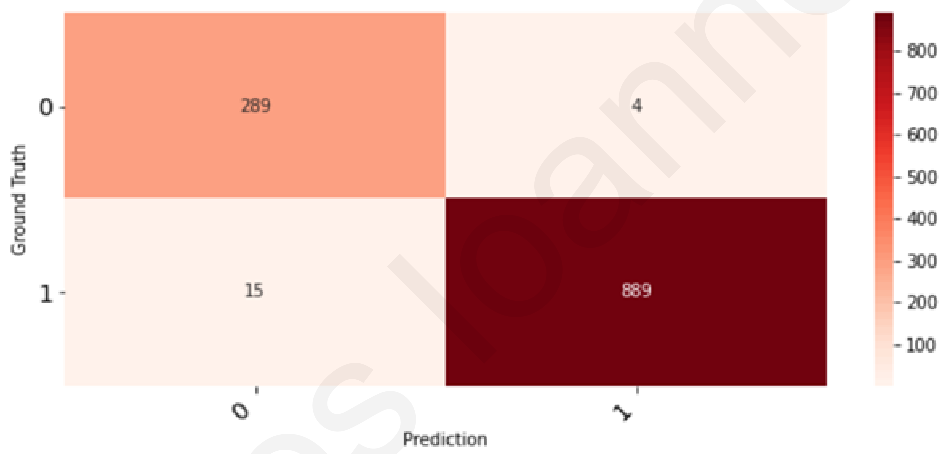


Figure 5.48: Dataset 6 Fold 4 Validation Confusion Matrix

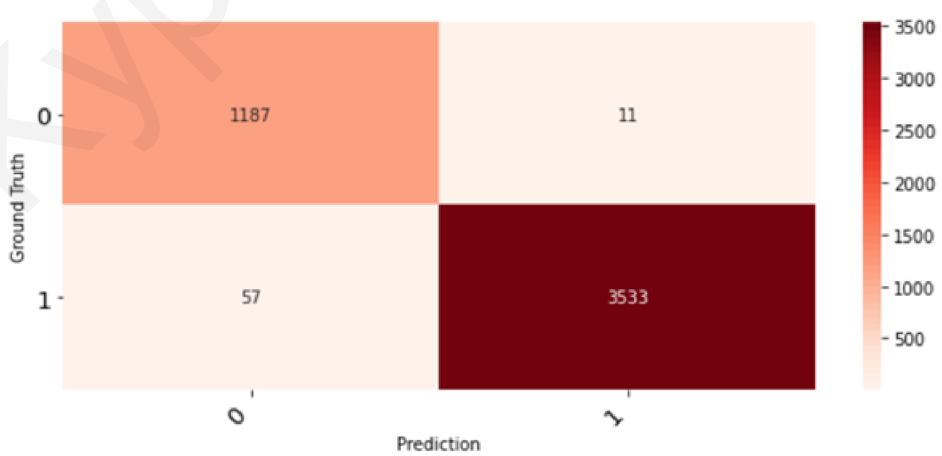


Figure 5.49: Dataset 6 Fold 5 Train Confusion Matrix

Kypros Ioannou

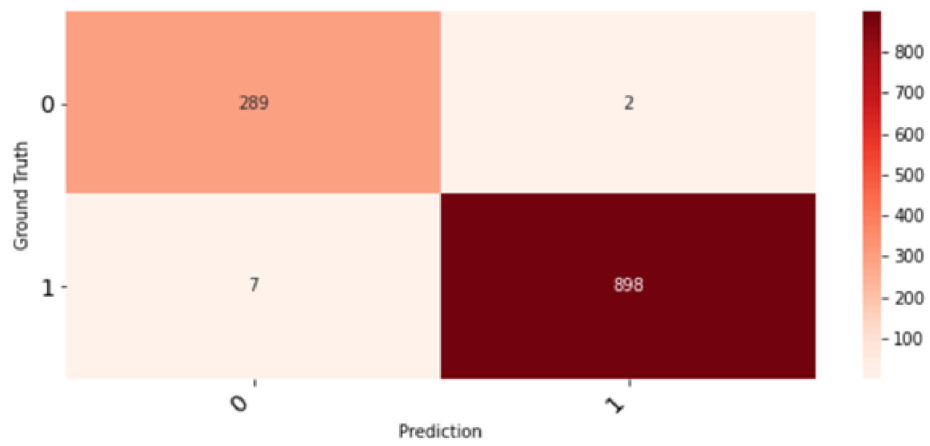


Figure 5.50: Dataset 6 Fold 5 Validation Confusion Matrix

Confusion Matrices from when we use CNN with Gradient Descent

Dataset1	Accuracy	Validation Accuracy	Spam Recall	Ham Recall	V Spam Recall	V Ham Recall
Fold 1	0.7117	0.6907	0	100	0	100
Fold 2	0.7022	0.7285	0	100	0	100
Fold 3	0.7087	0.7024	0	100	0	100
Fold 4	0.7090	0.7014	0	100	0	100
Fold 5	0.7057	0.7144	0	100	0	100

Table 5.1: Metrics for all the folds for the first dataset

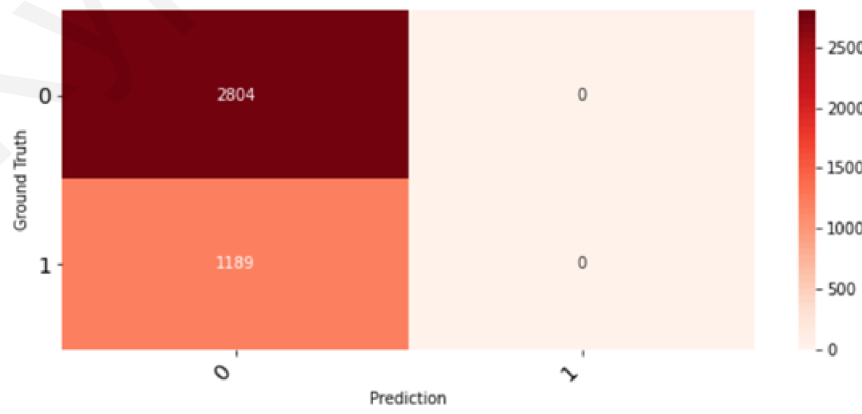


Figure 5.51: Confusion Matrix from when we train our model for the first dataset.

Kyprosloannou

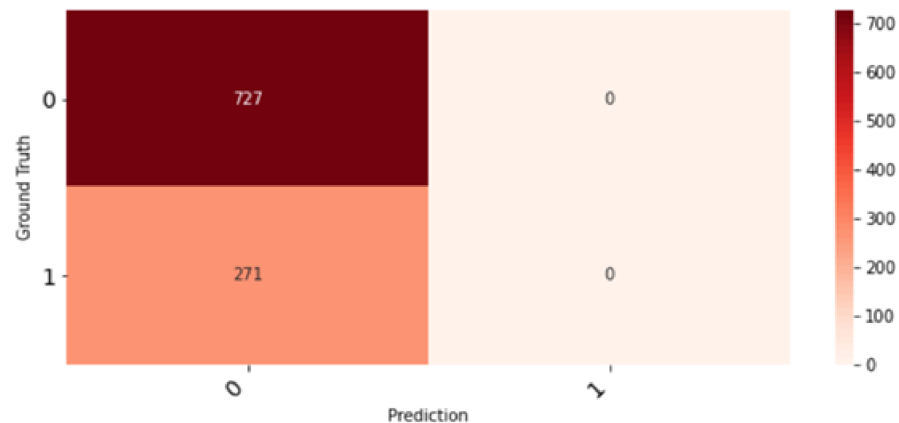


Figure 5.52: Confusion Matrix from when we test our model for the first dataset.

Dataset2	Accuracy	Validation Accuracy	Spam Recall	Ham Recall	V Spam Recall	V Ham Recall
Fold 1	0.7415	0.7502	0	100	0	100
Fold 2	0.7437	0.7416	0	100	0	100
Fold 3	0.7462	0.7885	0	100	0	100
Fold 4	0.7401	0.7474	0	100	0	100
Fold 5	0.7409	0.7457	0	100	0	100

Table 5.2: Metrics for all the folds for the second dataset

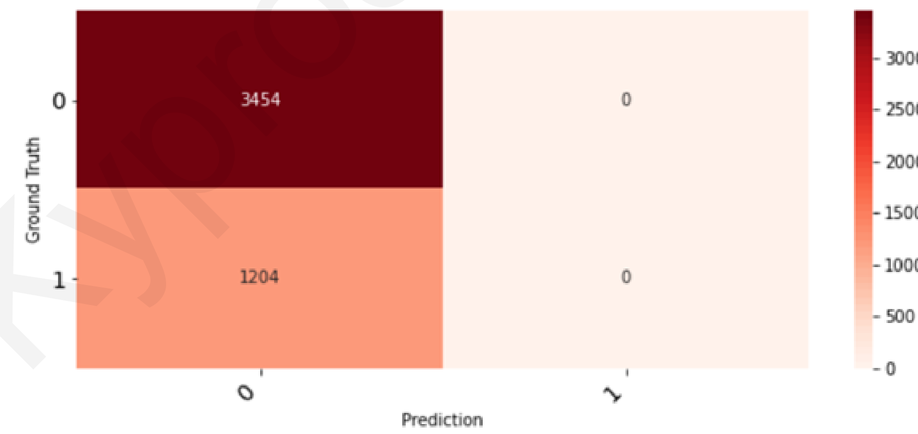


Figure 5.53: Confusion Matrix from when we train our model for the second dataset.

Kypros Ioannou

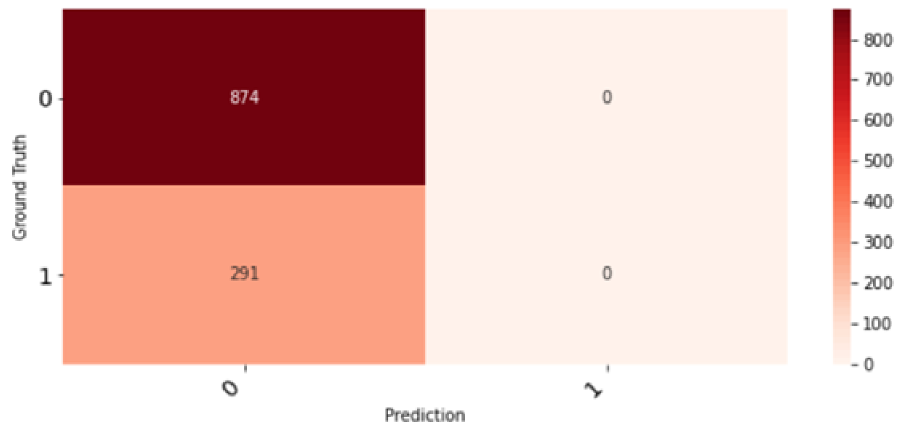


Figure 5.54: Confusion Matrix from when we test our model for the second dataset.

Dataset3	Accuracy	Validation Accuracy	Spam Recall	Ham Recall	V Spam Recall	V Ham Recall
Fold 1	0.7134	0.7251	0	100	0	100
Fold 2	0.7150	0.7014	0	100	0	100
Fold 3	0.7155	0.7090	0	100	0	100
Fold 4	0.7035	0.7438	0	100	0	100
Fold 5	0.7160	0.6992	0	100	0	100

Table 5.3: Metrics for all the folds for the third dataset

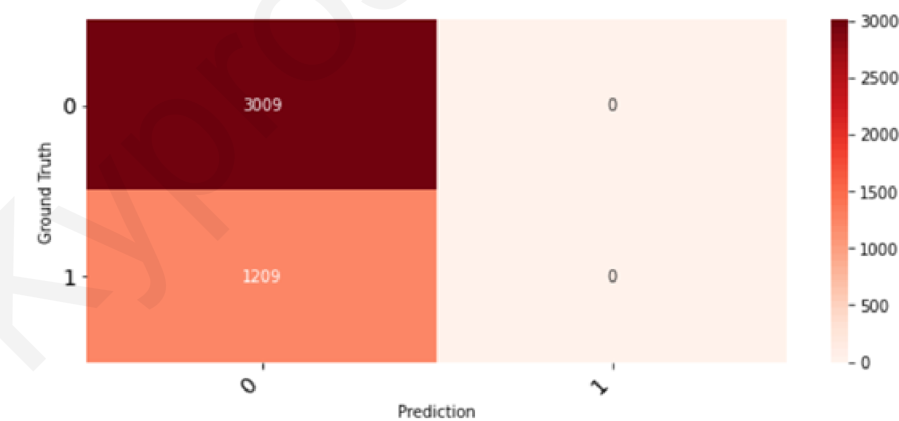


Figure 5.55: Confusion Matrix from when we train our model for the third dataset.

Kypros Ioannou

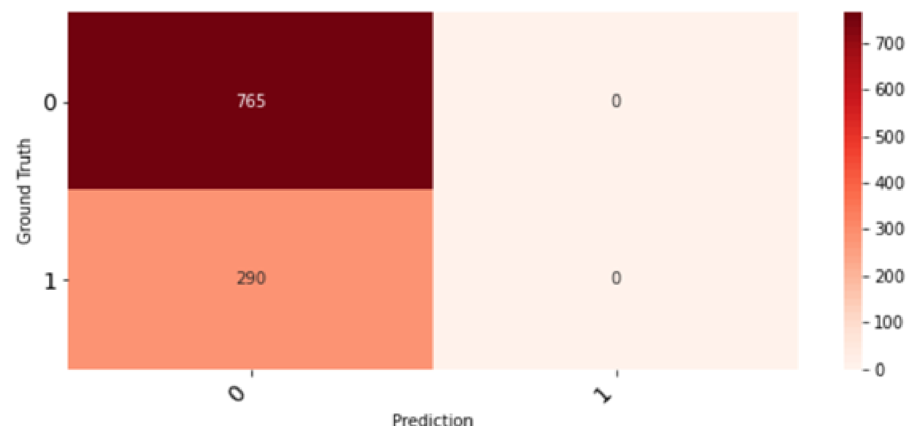


Figure 5.56: Confusion Matrix from when we test our model for the third dataset.

Dataset4	Accuracy	Validation Accuracy	Spam Recall	Ham Recall	V Spam Recall	V Ham Recall
Fold 1	0.7412	0.7391	100	0	100	0
Fold 2	0.7474	0.7280	100	0	100	0
Fold 3	0.7391	0.7613	100	0	100	0
Fold 4	0.6163	0.7391	100	0	100	0
Fold 5	0.6223	0.7502	100	0	100	0

Table 5.4: Metrics for all the folds for the fourth dataset

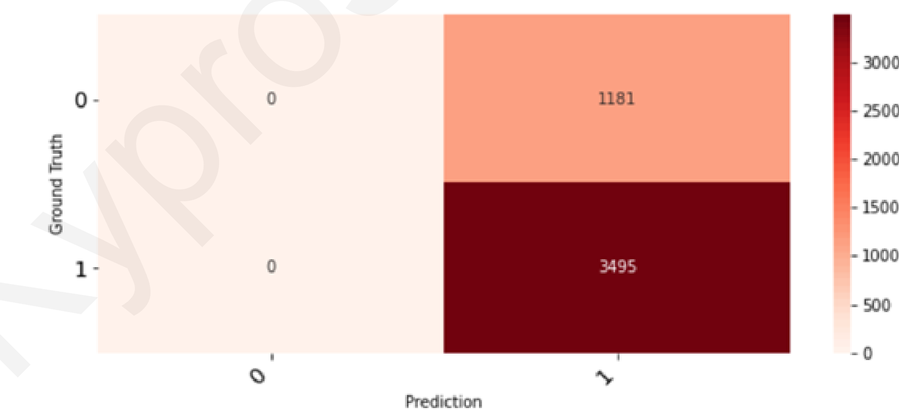


Figure 5.57: Confusion Matrix from when we train our model for the fourth dataset.

Kypros Ioannou

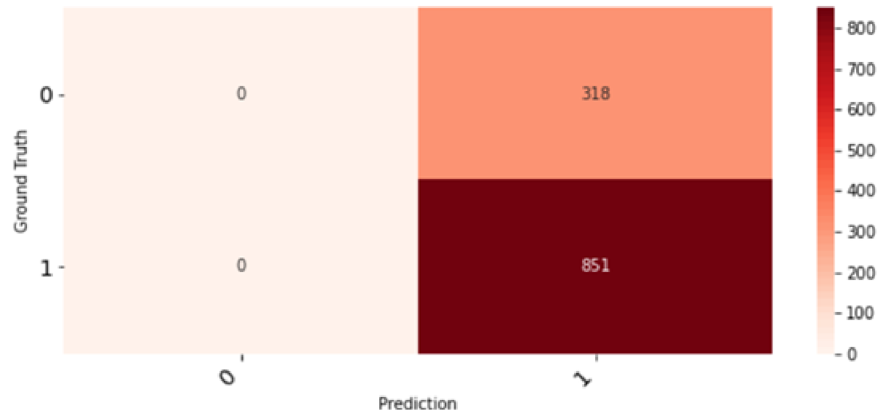


Figure 5.58: Confusion Matrix from when we test our model for the fourth dataset.

Dataset5	Accuracy	Validation Accuracy	Spam Recall	Ham Recall	V Spam Recall	V Ham Recall
Fold 1	0.7109	0.7370	100	0	100	0
Fold 2	0.7205	0.7123	100	0	100	0
Fold 3	0.7212	0.7094	100	0	100	0
Fold 4	0.7606	0.8542	100	0	100	0
Fold 5	0.7207	0.7114	100	0	100	0

Table 5.5: Metrics for all the folds for the fifth dataset

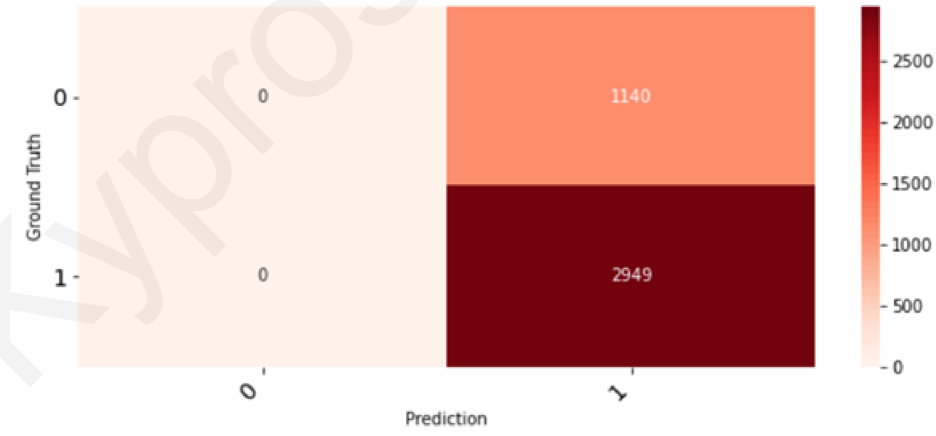


Figure 5.59: Confusion Matrix from when we train our model for the fifth dataset.

Kypros Ioannou

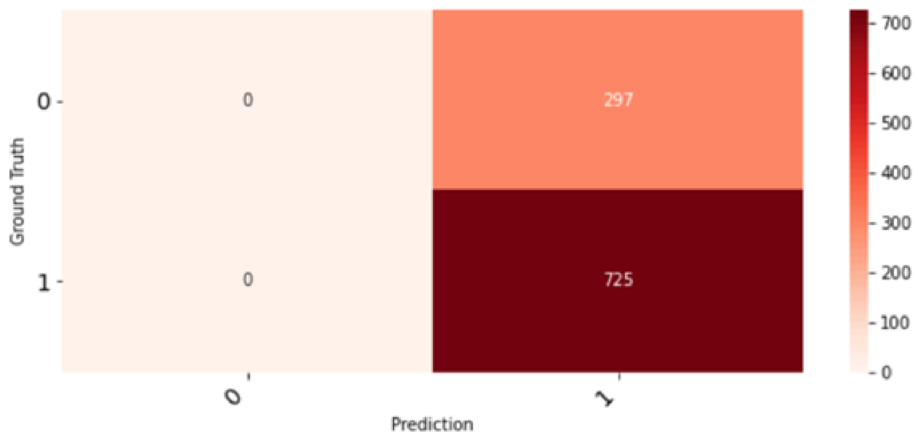


Figure 5.60: Confusion Matrix from when we test our model for the fifth dataset.

Dataset6	Accuracy	Validation Accuracy	Spam Recall	Ham Recall	V Spam Recall	V Ham Recall
Fold 1	0.7539	0.7402	100	0	100	0
Fold 2	0.7502	0.7552	100	0	100	0
Fold 3	0.7493	0.7432	100	0	100	0
Fold 4	0.7542	0.7593	100	0	100	0
Fold 5	0.7498	07532	100	0	100	0

Table 5.6: Metrics for all the folds for the sixth dataset

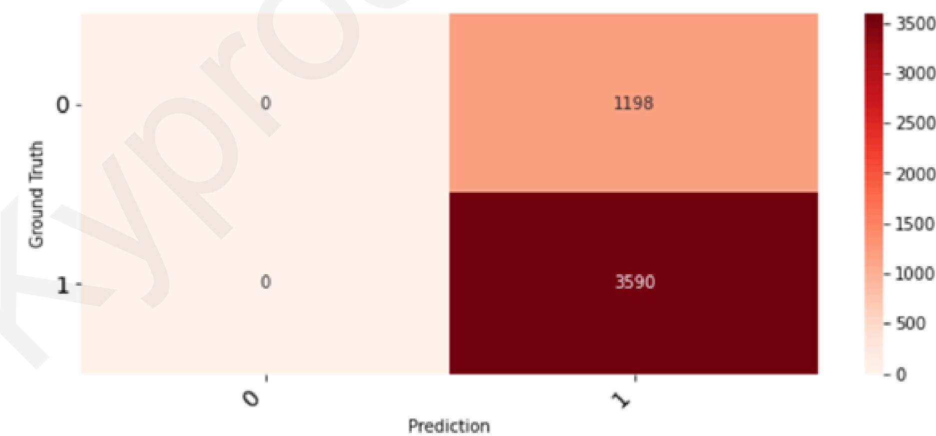


Figure 5.61: Confusion Matrix from when we train our model for the sixth dataset.

Kypros Ioannou

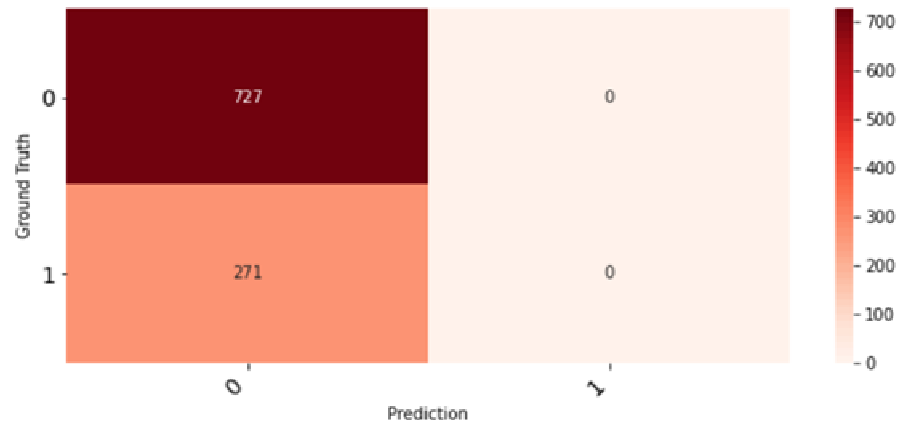


Figure 5.62: Confusion Matrix from when we test our model for the sixth dataset.

Kypros Ioannou