Master's Thesis

DETECTION AND CLASSIFICATION OF CONSTRUCTION OBJECTS BY USE OF MACHINE VISION AND DEEP LEARNING

Nicolas Nicolaou

Nicolas Nicolaou

Nicosia, May 2024

Approval Form

Master's Thesis

DETECTION AND CLASSIFICATION OF CONSTRUCTION OBJECTS BY USE OF MACHINE VISION AND DEEP LEARNING

Presented by

Nicolas Nicolaou

Supervisor: Symeon Christodoulou, Professor
Department of Civil and Environmental Engineering

_____

Member of the Committee: Loukas J Dimitriou, Associate Professor
Department of Civil and Environmental Engineering

_____

Member of the Committee:

_____

_____

Nicolas Nicolaou

Nicolas Nicolaou

Copyrights

©2024

Nicolas Nicolaou

ALL RIGHTS RESERVED

Acknowledgements

I would like to express my heartfelt gratitude to my supervisor, Dr. Symeon Christodoulou, for his invaluable assistance and guidance throughout the preparation and completion of my MSc thesis work. His support has been indispensable and greatly appreciated.

I would also like to extend my gratitude to my family for their unwavering support throughout this endeavor. Additionally, I am thankful to my colleagues and partners for granting me access to and supporting me in obtaining photographic material from construction sites.

Nicolas Nicolaou

ABSTRACT

The construction industry represents a continuously evolving environment where there is always room for process improvement and modernization. Artificial intelligence, as well as other emerging technologies such as machine vision (MV) and machine learning (ML), are indispensable in such an environment. Indeed, in recent years, their usefulness has become increasingly evident, along with the various possibilities for their application.

This research paper focuses on the detection and classification of objects at construction sites and analyzes the utility and potential of these detection and classification activities in the modern construction industry. Object detection and classification are performed by applying technologies such as machine vision (MV) and deep learning (DL) to image processing and/or in combination with object segmentation and labeling using bounding boxes. These activities have varied applications on construction sites, including but not limited to: (1) the monitoring of workers, machinery, and vehicles for productivity measurement and for the prevention of accidents and collisions; and (2) the monitoring and classification of procured of construction materials, to evaluate the progress of construction work. This serves as a valuable, low-cost measurement tool in the context of the management and monitoring of construction projects.

First, the presented research work examines the knowledge base from past work on the application of the aforementioned technologies to similar problems across varying professional domains, focusing on the construction sector. An extensive evaluation of their accuracy, reliability, and effectiveness is conducted, along with the identification of potential advantages/disadvantages of their application. Further presented is the methodology followed in pursuit of the research's scope of work. Specifically, nine specific 'construction classes' potentially found at a construction site are addressed. Subsequently, the corresponding results are provided and extensively discussed, along with an examination of any potential errors in the recognition and taxonomy of these classes.

Finally, snippets (in the Python programming language) of the programming code used during various phases of the overall methodology are provided.

Nicolas Nicolaou

ΠΕΡΙΛΗΨΗ

Ο κατασκευαστικός κλάδος αποτελεί ένα συνεχώς εξελισσόμενο περιβάλλον στο οποίο πάντα υπάρχει περιθώριο για ενέργειες βελτίωσης διεργασιών αλλά και εκσυγχρονισμού του. Η τεχνητή νοημοσύνη καθώς επίσης και άλλες νεοφανείς τεχνολογίες όπως είναι η μηχανική όραση (MV) και η μηχανική εκμάθηση (ML) δεν θα μπορούσαν να λείπουν από το περιβάλλον αυτό. Άλλωστε, τα τελευταία χρόνια γίνεται όλο και πιο εμφανής η χρησιμότητα τους αλλά και οι διάφορες δυνατότητες εφαρμογής τους.

Η παρούσα ερευνητική εργασία επικεντρώνεται στον εντοπισμό και την ταξινόμηση αντικειμένων σε εργοτάξια και αναλύει τη χρησιμότητα και τη δυναμική αυτών των δραστηριοτήτων εντοπισμού και ταξινόμησης στη σύγχρονη κατασκευαστική «βιομηχανία». Ο εντοπισμός και η ταξινόμηση αντικειμένων πραγματοποιείται μέσω της εφαρμογής τεχνολογιών όπως η μηχανική όραση (MV) και η βαθιά εκμάθηση (DL) στην επεξεργασία εικόνας ή/και σε συνδυασμό με την ανάθεση ετικετών και οριοθετημένων πλαισίων σε αντικείμενα. Αυτές οι δραστηριότητες έχουν διάφορες εφαρμογές στα εργοτάξια, καθώς συμβάλουν μεταξύ άλλων, στον έλεγχο του εργατικού προσωπικού, των μηχανημάτων και των οχημάτων που διακινούνται σε αυτό με σκοπό την πρόληψη ατυχημάτων και συγκρούσεων, καθώς επίσης στην παρακολούθηση και κατηγοριοποίηση των προμηθευομένων κατασκευαστικών υλικών, προκειμένου να αξιολογηθεί η πρόοδος των κατασκευαστικών εργασιών. Αυτό λειτουργεί ως ένα χαμηλού κόστους πολύτιμο εργαλείο επιμέτρησης-αξιολόγησης στα πλαίσια της διαχείρισης και παρακολούθησης κατασκευαστικών έργων.

Αρχικά, η παρούσα ερευνητική εργασία εξετάζει την ύπαρξη πρότερης εργασίας και εφαρμογής των εν λόγω τεχνολογιών για παρόμοιας φύσης ζητήματα σε οποιοδήποτε επάγγελμα και δη στον κατασκευαστικό τομέα. Γίνεται μία εκτενής αξιολόγηση της ακρίβειας, της αξιοπιστίας και της αποτελεσματικότητας τους καθώς και η διακρίβωση των πιθανών πλεονεκτημάτων/μειονεκτημάτων από την εφαρμογή τους. Επιπλέον, παρουσιάζεται η μεθοδολογία που ακολουθήθηκε στη βάση του σκοπού που περιεγράφηκε προηγουμένως. Συγκεκριμένα, εξετάζονται εννέα διαφορετικές κλάσεις αντικειμένων οι οποίες δυνητικώς εντοπίζονται σε ένα εργοτάξιο. Ακολούθως, δίνονται τα αντίστοιχα

Nicolas Nicolaou

αποτελέσματα και σχολιάζονται εκτενώς καθώς επίσης εξετάζεται η ύπαρξη τυχών σφαλμάτων.

Τέλος, δίνονται αποσπάσματα από κώδικα στη γλώσσα προγραμματισμού Python που χρησιμοποιήθηκε κατά τη διάρκεια διαφόρων φάσεων της μεθοδολογίας που ακολουθήθηκε.

Nicolas Nicolaou

TABLE OF CONTENTS

Nicolas Nicolaou

LIST OF FIGURES

Hellenic Open University

LIST OF TABLES

Nicolas Nicolaou

LIST OF ABBREVIATIONS

AI	Artificial Intelligence
MV	Machine Vision
DL	Deep Learning
IC	Image Classification
OD	Object Detection
NMS	Non-Maximum Suppression
IoU	Intersection over Union
mAP	mean Average Precision

Nicolas Nicolaou

1. INTRODUCTION

1.1 Literature review

The construction sector, even in modern times, faces numerous perennial problems and challenges related to the effective management and monitoring of construction projects.

Some of these challenges are, for example, associated with safety and health on construction sites. Despite significant progress achieved through the introduction of regulatory frameworks and legislation, a considerable number of occupational accidents are still recorded today. These accidents are not exclusively personal but are often linked to the reckless use of mechanical equipment and vehicles, as well as insufficient coordination, organization, and monitoring at construction sites. Therefore, a system using machine vision (MV) and deep learning (DL) technologies could, in this case, detect a worker in a restricted zone of the construction site, or determine whether the worker is wearing appropriate safety equipment or, even better, continuously appraise the ergonomic risks to workers (Lambrides and Christodoulou, 2023) [1]. Additionally, the system could be used to ensure the proper operation of mechanical equipment in designated areas and the maintenance of safe distances by the working personnel at the construction site.

Another critical issue commonly encountered at construction sites is resource management and construction progress monitoring. Delays, particularly in large construction projects, often occur due to material shortages and insufficient logistics in material delivery. Conflicts and construction delays are also frequent occurrences. Therefore, a mechanism based on the aforementioned technologies could be employed to monitor the transport of construction material to sites and visually document the construction process. This visual documentation can be valuable for reference and analysis.

In light of the aforementioned and other challenges faced by the modern construction industry, the use of machine vision and deep learning technologies is imperative. These technologies enable the automation of numerous technical processes on the construction site while facilitating the monitoring and resolution of various problems within it. The investment of the construction industry in technologies related to artificial intelligence has

Nicolas Nicolaou

taken significant dimensions in recent years. Specifically, according to Market.us (Figure 1.1) [2], for the year 2023, the value of Generative AI in the construction industry, on a global scale, has been estimated at around USD 142 million, while this value is expected to double by 2033, as depicted in the following graph. Additionally, a significant, if not the most significant, contribution to this value seems to be attributed to machine learning technology, something that is expected to continue happening in the immediate future. This once again confirms the prominent value of this technology in the activities of the construction sector.

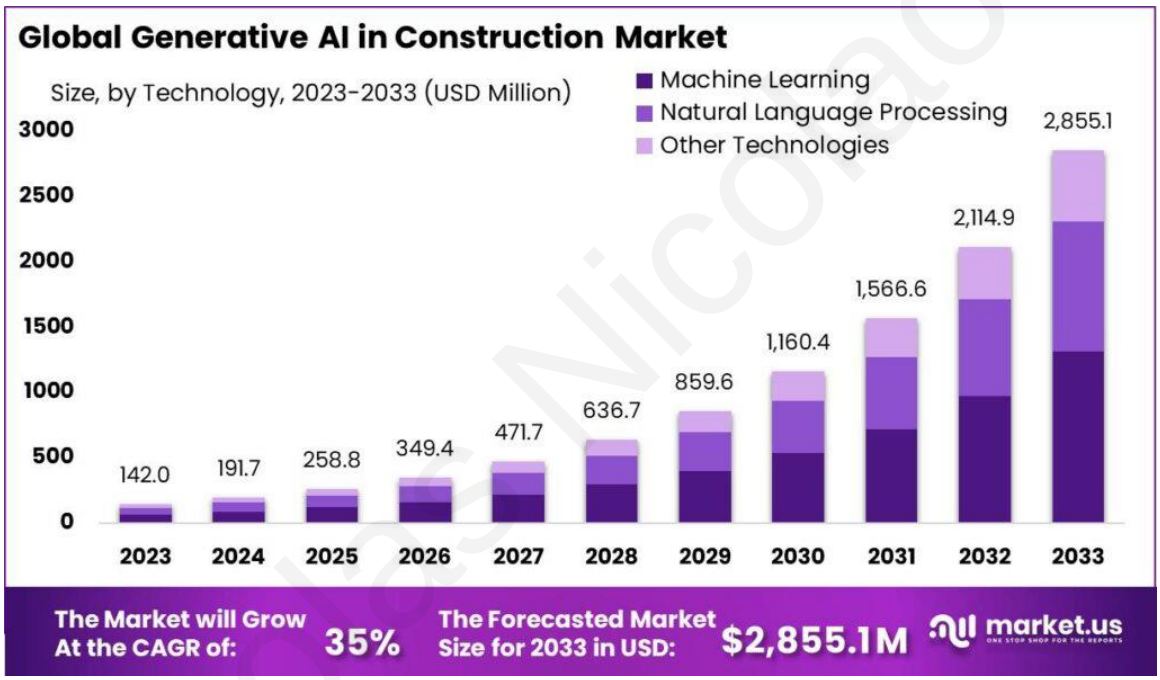


Figure 1.1: The value of global Generative AI in the construction industry

In recent years, a significant volume of research studies have been conducted on the use of machine vision (MV) and deep learning (DL) technologies for detecting and classifying construction elements at construction sites. This trend began in the early 2010s, with works on the automated generation of parametric BIMs (Brilakis et al., 2010) [3], and despite the significant improvements in relevant applications, there is still room for further development.

Nicolas Nicolaou

The article by Czerniawski and Leite (2020) [4] introduces the automation of digital modeling of existing buildings through reality capture devices and computer vision algorithms. The goal is to facilitate the use of digital building representation technologies, promoting new forms of simulation, automation, and information provision. The article provides an overview of endeavors spanning the last decade (2010-2020), with a primary focus on object recognition methodologies. Addressing limitations identified in previous review literature, the authors meticulously dissect the structure and variations of object recognition systems, accompanied by thorough quantitative performance evaluations. The research results suggest that achieving a more complete semantic coverage of building infrastructures will require a revision and intensification of relevant efforts. In conclusion, the study advocates for a reevaluation and heightened dedication to bridging existing gaps in object recognition technologies to propel advancements in automated digital modeling for existing buildings.

Nath and Behzadan (2020) [5] propose the validation of a genetic adversarial network (GAN) based on a deep convolutional neural network (CNN). The research involves photos taken, trained, and tested at the construction site from two internal datasets to increase image resolution when generating missing pixel information. Results demonstrate that using GAN-enhanced images can further improve the average accuracy of pre-trained models for object detection while maintaining overall processing time for real-time object detection.

A key aspect in leveraging DL methods for construction site data interpretation is the accurate identification of objects of interest. Achieving this accuracy requirement is essential, albeit often at the expense of computational speed. While lightweight DL algorithms like Mask R-CNN offer high accuracy in visual recognition tasks, their processing efficiency may hinder real-time decision-making capabilities. Notably, the YOLO (you-only-look-once) algorithm emerges as a promising solution due to its ability to strike a balance between speed and accuracy. The study introduces the Pictor-v2 dataset, comprising approximately 3,500 images and 11,500 instances of common construction site objects. Through transfer learning, the researchers trained YOLO-v2 and YOLO-v3

Nicolas Nicolaou

variants, assessing their performance on various combinations of data sources, including crowdsourced and web-mined images.

This review underscores the significance of employing DL-based approaches in construction site data interpretation and lays the groundwork for future research endeavors aimed at enhancing human capabilities through advanced assistive technology systems in complex visual data environments.

In a subsequent work, Paneru and Jeelani (2021) [6] provided an up-to-date and categorized overview of computer vision applications in construction by examining recent developments in the construction sector and the challenges that future research must address to maximize the benefits of computer vision. The authors focus on specific areas considered most likely to benefit significantly from computer vision, such as safety management on construction sites, progress and productivity monitoring, and work quality control.

This work provides an insightful and categorized overview of computational vision applications in construction, highlighting recent advancements in the field and identifying opportunities and challenges for future research. The focus is narrowed down to four key areas where computer vision can significantly enhance construction management: Safety Management, Progress Monitoring, Productivity Tracking, and Quality Control. Overall, the article offers a comprehensive examination of the potential benefits of integrating computer vision into construction management practices, shedding light on the opportunities and obstacles that lie ahead in fully harnessing the capabilities of this technology.

One year later, Duan et al. (2022) [7], focused on developing a large-scale image dataset specifically collected and processed for construction sites, named SODA (Site Object Detection Dataset). This dataset includes 15 types of objects categorized into mechanical means, materials, and labor personnel. Specifically, 20,000 images were collected from various construction sites, considering different construction site conditions, weather conditions, construction phases, and shooting angles. After careful examination and processing, 19,846 images were selected, containing 286,201 objects accompanied by corresponding labels from predefined categories.

Nicolas Nicolaou

An analysis conducted indicated that the developed dataset is advantageous in terms of diversity and volume. Further evaluation using two widely accepted object detection algorithms based on deep learning (YOLO v3 / YOLO v4) demonstrated the dataset's effectiveness in visualizing typical construction scenarios, achieving a maximum mean Average Precision (mAP) of 81.47%. This research contributes a large-scale dataset for the development of deep learning applications in object detection within the construction industry. It serves as a reference point for the further evaluation of corresponding algorithms in this field.

In their work, Wang et al. (2022) [8] proposed a new semantic method aiming to extract information by integrating deep learning object detection and image captioning. This method explores important information from construction images or videos. In the proposed approach, object detection serves as an encoder to extract features of construction objects and the holistic image. The image caption was selected as a decoder to extract the semantic information. A new post-processing technique has been suggested to assess semantic information in graph format, aiming to enhance accessibility and visualization. In experimental trials, the proposed approach yielded a Consensus Image Description Evaluation (CIDEr) score of 1.84, indicating its effectiveness. By adopting this method, semantic information from construction images can be presented to project managers as a valuable tool for making crucial decisions on the construction site.

In the research work of Hou et al. (2022) [9], a multi-object detection method based on the improved YOLOv4 model is proposed to overcome the problem of low detection accuracy. The method involves several key optimizations, including the utilization of the K-means algorithm for anchor box initialization, replacing pooling operations with dilated convolution to preserve feature map resolution, and integrating focus loss to address sample imbalance during model training. Research results indicate that the average accuracy (mAP) of the improved YOLOv4 model for many objects can reach 97.03%, which is 2.16% higher than that of the original YOLOv4 detection network. At the same time, the detection speed reached 31.11 fps, a decrease of 0.59 fps, a result quite satisfactory for real-time detection data.

Nicolas Nicolaou

Overall, this research marks a notable advancement in environment perception for construction machinery swarm operations. By addressing critical limitations in detection accuracy and speed, the proposed method lays a solid foundation for the unmanned and intelligent evolution of construction machinery operations, promising enhanced efficiency and safety in complex construction environments.

Zhou et al. (2022) [10] propose an object detection method based on an improved YOLOv5 model with high sorting accuracy of construction waste. It involves creating a dataset from images of construction waste taken in situ at construction sites. This improved model was trained, validated, and tested based on the collected images and compared with other conventional models such as Faster-RCNN, YOLOv3, YOLOv4, and YOLOv7. The YOLOv5 model recorded an average accuracy (mAP) on the test dataset of 0.9480, indicating better performance than other conventional models in object detection.

Overall, the study underscores the accuracy and practicality of the enhanced YOLOv5 model for sorting construction waste. By outperforming existing models, the proposed approach holds significant potential for optimizing waste management processes in construction settings, ultimately contributing to improved efficiency and resource utilization.

In a recent research paper by Jog et al. (2022) [11], full-scale validation experiments of a multi-object location tracking method for its application to resource tracking in large-scale, congested, outdoor construction sites are presented. The validation stage involved testing under harsh conditions on various large project sites. This research paper describes the process of data collection and testing, as well as the measurements and results obtained. The validation showed that the new vision tracking provides a good solution for tracking different entities in large and congested construction sites.

1.2 Thesis scope

As previously mentioned, this research work endeavor aims to achieve successful classification and detection of objects encountered at construction sites through the utilization of photographs and technologies in machine vision and deep learning. The object detection and classification tasks was focused on nine distinct classes of objects,

Nicolas Nicolaou

encompassing both load-bearing and non-load-bearing structural elements, excavators, human personnel, and individual protective equipment. Additionally, a capability was incorporated to examine safety and health issues at construction sites by issuing relevant warnings in case individuals without the required personal protective gear were identified in the photographs. For the training of classification and object detection models, two pertinent classes, namely "safety helmet" and "reflective jacket," were incorporated among the nine classes under examination to facilitate the aforementioned supplementary functionality. Furthermore, conclusions regarding the performance and accuracy of the new custom trained models were drawn through examples and confusion matrices, utilizing various measurement units and success rates.

To accomplish the aforenamed objective, models, codes (in Python), and algorithms were employed, which undertake the activities under examination with the assistance of computer vision and deep learning technologies. Following this, in this thesis a dedicated chapter (“Research Background”) elucidates the operational mechanisms of these technologies and the pertinent tasks they undertake. Furthermore, a chapter entitled "Research Methodology" is dedicated to providing a detailed explanation of the preparation and training process of the two models under study, as well as the examination of other functions. Following this chapter, the presentation and discussion of the relevant results ensue, along with a further evaluation of the effectiveness of the resultant models. In conclusion, the research work concludes with chapters on findings summary and conclusions, where a comprehensive and overarching commentary on the relevant findings is provided.

1.3 Thesis organization

Further to this introductory and brief literature review chapter, the thesis discusses the research background (Chapter 2) on machine vision (MV), deep learning (DL), image classification (IC) and object detection (OD), and the research methodology (Chapter 3).

Chapters 4 and 5 present the analysis performed and the findings, respectively, on object detection and classification at construction sites.

Nicolas Nicolaou

The thesis concludes with a chapter on key conclusions and an appendix with snippets of the programming code used for the analysis.

Nicolas Nicolaou

2. RESEARCH BACKGROUND

2.1 Machine Vision (MV)

Machine vision, also referred to as computer vision, pertains to a technological domain wherein computers are equipped to interpret and comprehend visual data, akin to the human visual system. It involves the development of algorithms, techniques, and systems that empower machines to extract, analyze, and understand meaningful insights from digital images or video feeds.

Typically, machine vision setups encompass cameras or similar image-capturing devices for acquiring visual data, alongside software and hardware components tasked with processing this data to execute various functions. These functions may entail tasks such as detecting, recognizing, classifying, tracking, analyzing motion, enhancing images, and reconstructing 3D representations. The application scope of machine vision spans numerous industries and disciplines, encompassing manufacturing, healthcare, agriculture, automotive, surveillance, robotics, augmented reality, and beyond. Its significance lies in its pivotal role in automation, quality control, inspection, monitoring, and decision-making processes by granting machines the capability to autonomously "see" and interpret visual information. An example of object detection task as part of machine vision technology is given in Figure 2.1 [12].

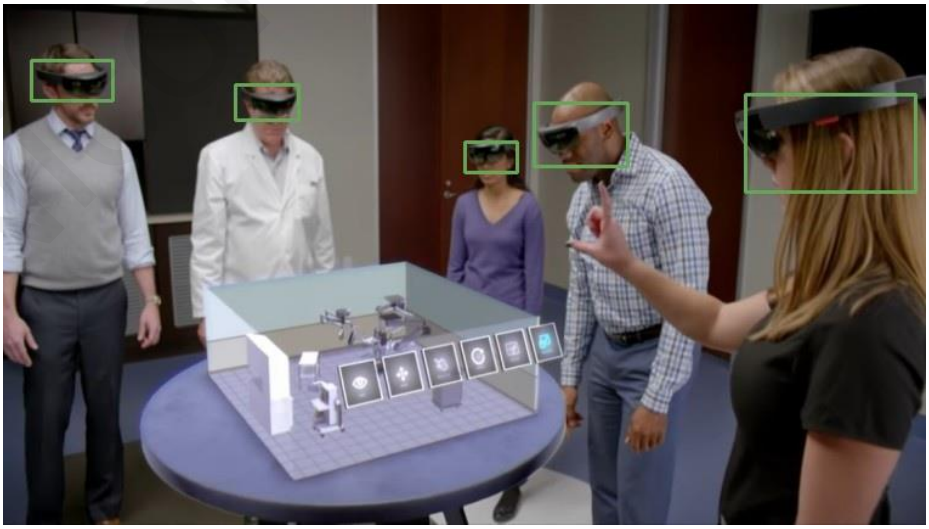


Figure 2.1: Object detection using Machine Vision

Nikolas Nicolaou

2.2 Deep Learning (DL)

Deep learning is a branch of machine learning technology focused on training complex artificial neural networks with multiple layers. These networks, inspired by the human brain's structure, process data through interconnected nodes called neurons across successive layers. Each layer extracts features from the input data, and through backpropagation, the network adjusts its internal parameters to minimize prediction errors during training.

Deep learning is particularly effective in tasks like image and speech recognition, natural language processing, and recommendation systems. It automatically learns hierarchical representations of data, eliminating the need for manual feature engineering. Advancements in hardware and the availability of large datasets have propelled deep learning's popularity and impact. It has revolutionized fields such as computer vision, speech processing, healthcare, finance, and autonomous systems. The following figure [13] explains graphically the difference between machine and deep learning.

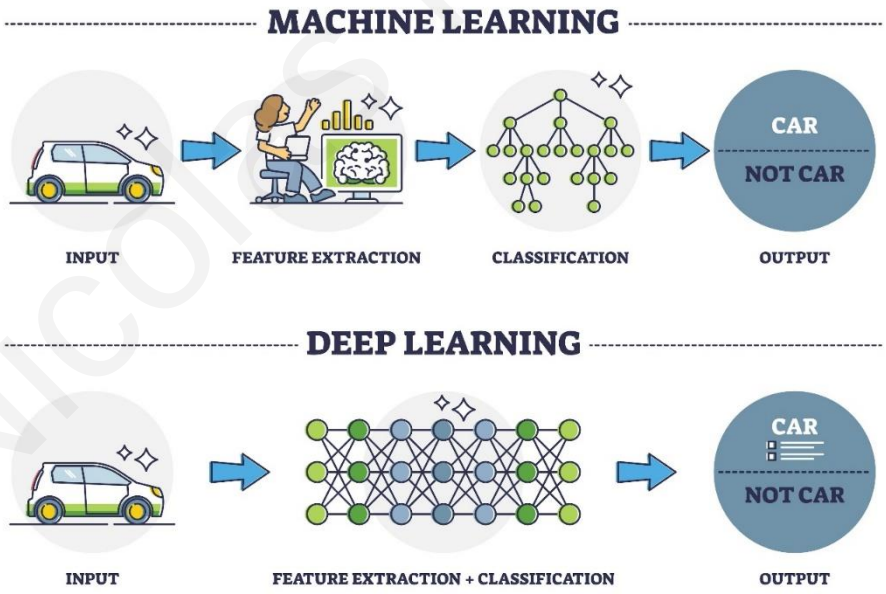


Figure 2.2: Functional difference between Machine Learning and Deep Learning

2.3 Image Classification (IC)

Image classification, a core concept in machine vision, refers to the process where images are sorted into predefined categories based on their visual features. By use of machine learning or deep learning algorithms, models are trained to discern patterns and characteristics that differentiate one category from another, such as shapes, colors, or textures. After training, these models can predict the most probable class for new images, often providing confidence scores or probability percentages for each predicted label. Image classification finds applications in various fields like object recognition, medical imaging, and satellite analysis. It serves as a foundational task in computer vision systems, providing essential insights into the content of images and enabling more complex tasks. In Figure 2.3 [12] a sample result of image classification implementation is provided through the usage of *ImageAI* [12] Python library, in which this thesis is based. Specifically, the classes with their corresponding probabilities of successful prediction are provided. The class with the highest percentage is considered to be the one that explains the content of the respective image to a greater extent according to the relevant trained model.

Figure 2.3: Sample result for image classification by ImageAI library

2.4 Object Detection (OD)

Object detection is a machine vision process that identification and localization of specific objects within images or videos. In contrast to image classification, which categorizes whole images into predefined classes, object detection not only identifies objects within an image but also determines their exact positions by outlining bounding boxes around them.

This task typically relies on machine learning or deep learning algorithms trained on datasets containing images labeled with annotated object bounding boxes. Object detection is vital for numerous applications, such as autonomous driving, surveillance, robotics, and medical imaging, as it allows machines to comprehend and interact with their environment by detecting and localizing relevant objects in visual data.

However, detection is frequently conflated with recognition, which refers to the process of identifying and comprehending objects or patterns within an image or scene. Unlike detection, recognition entails a more profound analysis of visual content, which may include grasping the context, identifying specific object features or traits, and drawing higher-level associations or inferences based on observed patterns.

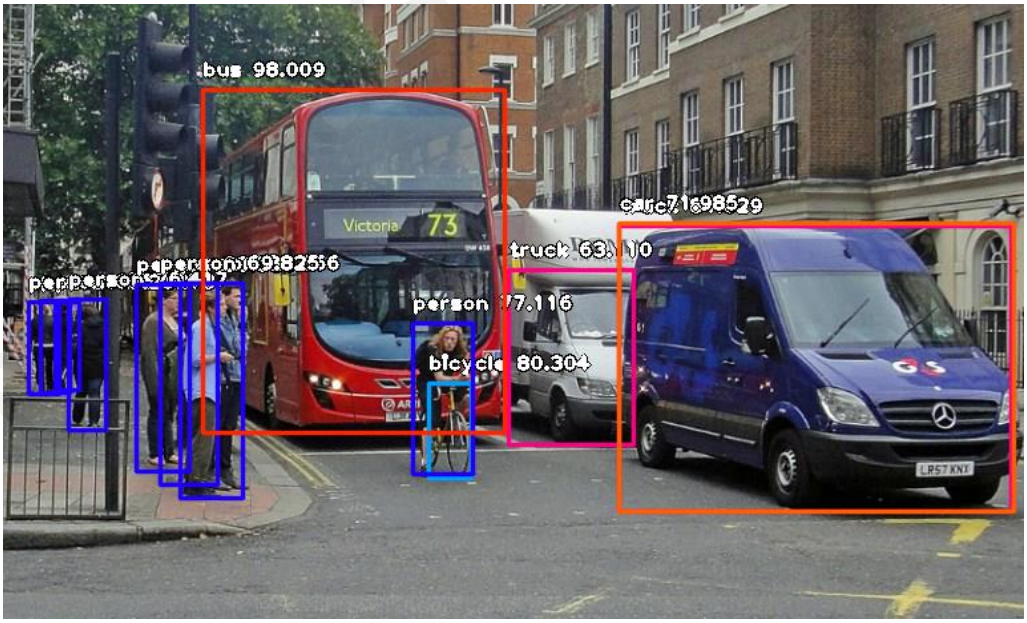


Figure 2.4: Sample result for object detection by ImageAI library

In the above figure a sample result of object detection implementation is provided through the usage of *ImageAI* [12] Python library. In contrast to image classification, in this activity, separate probabilities are provided, rather than complementary ones, for each object detected, along with additional information regarding the inclusion of the corresponding bounding boxes. Similarly, higher prediction percentages indicate greater confidence in the relevant model for successful localization, prediction, and classification of each respective object.

Nicolas Nicolaou

3. RESEARCH METHODOLOGY

3.1 Overview

The research work discussed herein focuses on the automated detection and classification of construction objects, and the applied research methodology was based on utilizing the Python programming language along with machine vision and deep learning technologies.

The goal was to create software, or leverage existing tools, capable of learning a series of construction objects present on a construction site. Subsequently, the software should successfully detect and classify these objects using either images from a dataset or random images. To achieve this objective, *ImageAI* (v.3.0.3) [12] was employed. *ImageAI* (Moses, 2018) is an open-source Python library that simplifies machine vision and deep learning tasks. It is built on other libraries such as *TensorFlow* and *Keras*. From the array of tasks offered by *ImageAI*, specific code libraries related to image classification and object detection were utilized - activities directly aligned with the focus of this research. The codes were divided into those dedicated to image classification and those dedicated to object detection. For each of the two tasks, a code was used for custom model training process based on the custom classes, resulting in the creation of a model. Additional codes were employed for result extraction, verification of the resulting accuracy-performance, and broader evaluation of the respective trained models, primarily through the utilization of unseen data.

Furthermore, a dataset was created for each task, incorporating photos of all the examined classes. These data resulted from a combination of own photos from construction sites, ready-made datasets from Kaggle [14] [15] [16] [17], which is a platform for data science and machine learning competitions, the GitHub web-based platform [18], and generally photos obtained by the Google Images search service. In the context of this research, the decision was made to initially explore two distinct classes to clarify the operational mode and compatibility of *ImageAI* with the research goals. These objects were the ‘column’ and the ‘excavator’. However, at a later stage, seven more classes (totaling 9) were added, which were as follows: ‘beam’, ‘masonry’, ‘slab’, ‘window’, ‘person’, ‘safety helmet’, and

Nicolas Nicolaou

'reflective jacket'. The choice of some of these object classes relates to the intent of using the developed algorithms and trained models for use in health & safety applications at construction sites. Furthermore, for the purpose of custom object detection training, numerical codes needed to be implemented for each class, as illustrated in the following table.

Table 3.1: Numerical codes of examined construction objects' classes

EXAMINED CONSTRUCTION OBJECTS' CLASSES	
Numerical Code	**Object Class**
0:	Column
1:	Excavator
2:	Beam
3:	Masonry
4:	Slab
5:	Window
6:	Person
7:	Safety Helmet
8:	Reflective Jacket

3.2 Image Classification Framework

For this task, a set of 6000+ images of the object classes to be examined was collected. Initially, a general folder was created, which contained two additional folders named 'train' and 'test,' respectively. Within each folder, a subfolder was created for each prediction class. The training photos, used to train the classification model, and the corresponding test photos, used to evaluate it, were placed in these subfolders.

In the 'train' folder/dataset, 500 photos were included for each class, while in the 'test' folder/dataset, 200 photos were included. This dataset was then utilized in the training code (Appendix) as provided by *ImageAI* [12], where various tasks were performed, including the selection of the algorithm. *ImageAI* offers the option to use four different algorithms for training custom image prediction models (*MobileNetV2*, *ResNet50*, *InceptionV3*, and *DenseNet*), each with different speed and prediction accuracy characteristics.

Nicolas Nikolaou

Additionally, other parameters such as 'batch_size' (the number of images the network will process simultaneously) and 'num_experiments' (the number of network training iterations on all training images) were set in this code. For the purposes of this work, the MobileNetV2 algorithm was chosen due to its fastest prediction speed in compare with other algorithms.

Upon each execution of training code, the model attaining the highest accuracy was generated and subsequently stored in the dataset folder, accompanied by its corresponding generated *JSON* file. In this scenario, a *JSON* (JavaScript Object Notation) file functions as a structured data format for the storage and exchange of information pertaining to the custom detection and classification tasks conducted utilizing *ImageAI* or analogous frameworks. Additionally, the other parameters mentioned above were systematically varied during each run to elucidate their impact on the accuracy of the respective model. This measure was undertaken to facilitate the incremental enhancement of the model, which became evident with each successive iteration. In this context, accuracy represents the percentage probability that a detected object belongs to a specific class. The accuracy is calculated using the following formula:

$$\text{Accuracy} = \frac{\text{Number of Correctly Classified Images}}{\text{Total Number of Images}} * 100 \qquad [3.1]$$

This percentage reflects the model's confidence in the correctness of its prediction. Higher percentage probabilities generally indicate that the model is more confident in recognizing a particular class of object in the image.

At a later stage, this model was employed in another code, where its effectiveness in predicting the examined and subsequently trained classes was evaluated using both trained and random photos. In this code, the trained model, along with its corresponding *JSON* file, was imported, alongside the algorithm used for training the said model. Subsequently, the photograph to be examined by the image classification model was selected, concurrently with the configuration of certain parameters (some of which are optional). Among other settings, the number of classes to be displayed in the results was chosen, as well as the limitation of prediction display by setting a minimum threshold percentage for

Nicolas Nicolaou

predictions. By default, for the aforementioned code provided by *ImageAI*, this threshold is set at 30%. A schematic overview of this methodology is depicted in Figure 3.1.

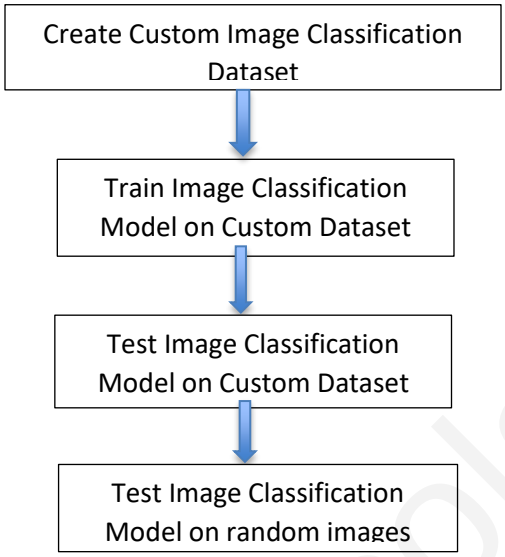


Figure 3.1: Custom image classification methodology flowchart

3.3 Object Detection Framework

For this task, a set of 2700+ images was collected for the examined classes. Initially, a general folder was created, which included two subfolders named ‘train’ and ‘validation,’ respectively. Within each of these folders, two additional subfolders were created. The first, named ‘images,’ contained photos - in jpg format - of the examined classes without separating them based on the object they depict. The second, named 'annotations,' contained the corresponding assignments for these classes, in txt format. The number of instances and images used for each class is given in Figure 3.2, although it is important to indicate that some images used for multiple classes.

For this task, a set of photos from the dataset collected for image classification was utilized. This dataset was then input into the training code as is given by ImageAI, where, among other tasks, algorithm selection was performed. ImageAI provides the option to use two different algorithms to train custom image object detection models, namely *YOLOv3* and *TinyYOLOv3*, each with varying speed and accuracy characteristics for prediction. In this

Nicolas Nicolaou

code, additional parameters such as ‘batch_size’ and ‘num_experiments’ were set, as previously explained.

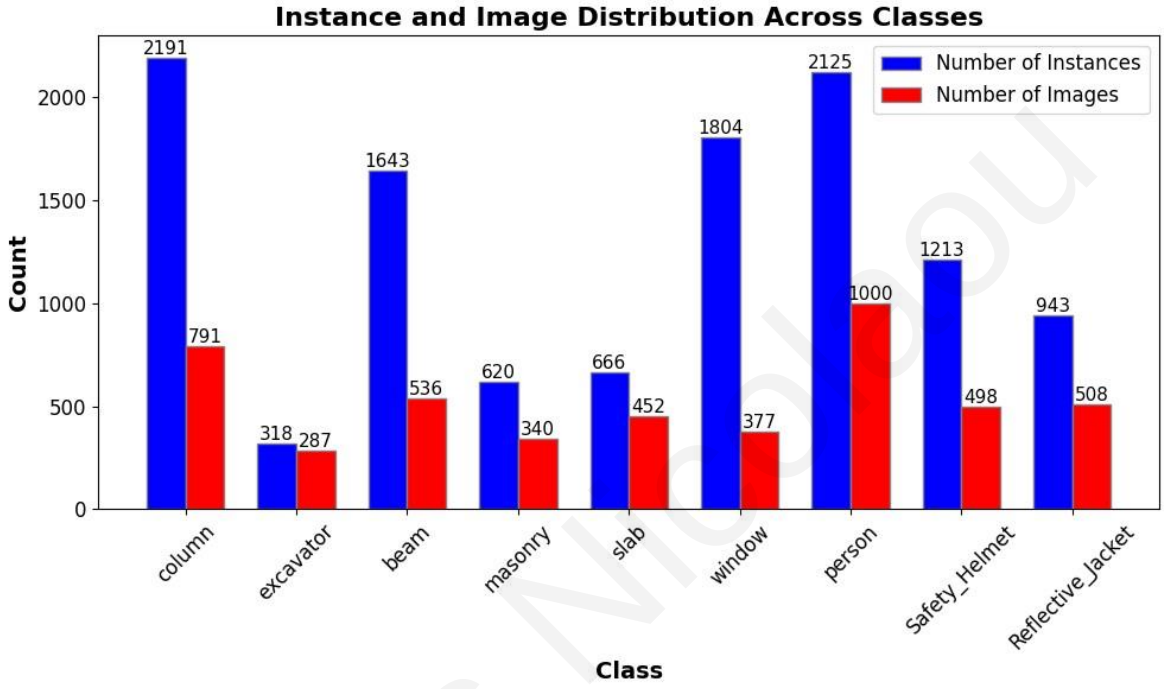


Figure 3.2: Instance and image distribution across custom classes

During the training process for object detection, the initial model used did not include specific objects such as those found on construction sites. The model training with additional construction site objects enriches the utilized pre-trained model and facilitates its use on construction-related image detection applications. Additionally, the option for training using a pre-trained *YOLOv3* model was specified. For the purposes of this work, both algorithms were employed. Future work shall aim the incorporation of newer releases of *YOLO* models (e.g., *YOLOv8*) and training datasets (e.g., *SODA*).

To create the assignments, an open-source graphic annotation tool for images, *Labelimg* [19], was employed. The process involved creating bounding boxes and labels in each photo and assigning them to each of the examined objects for the purpose of custom object detection training process. The associated annotation *.txt* files were generated in *YOLO*

format, wherein each row signifies a detected object and includes specific details such as the numerical class code and its corresponding coordinates. In Figure 3.3, an example of annotation creation in the *LabelImg* [19] interface is provided. As part of this task, 300+ photos were collected for each object, with 70-80% stored in the 'train' folder for training the detection model and the remainder in the 'validation' folder for evaluating the model's performance during training.

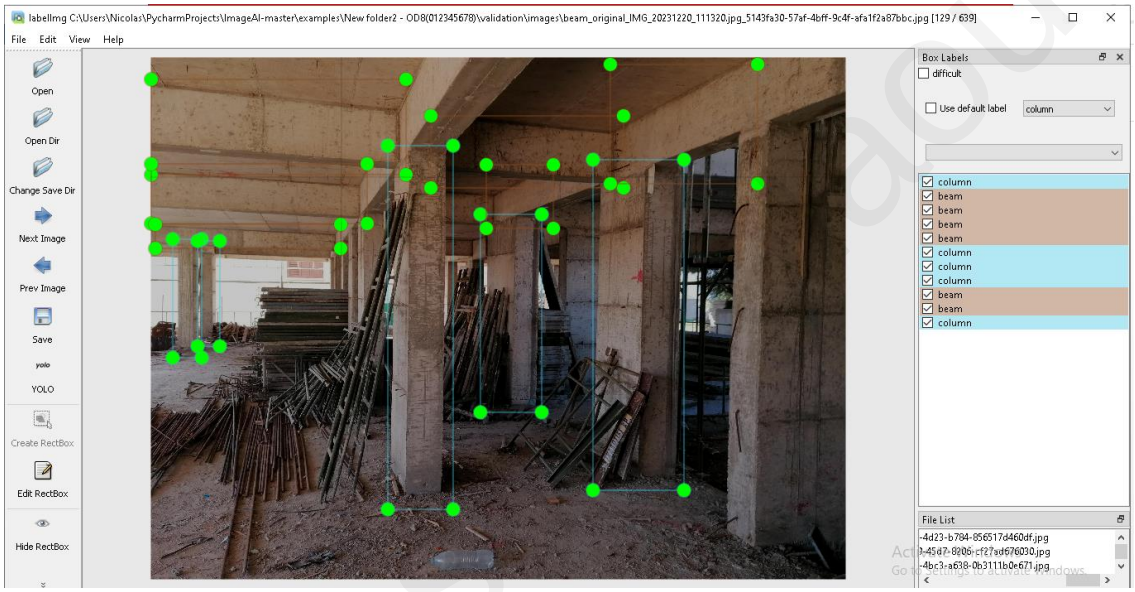


Figure 3.3: Labelimg workspace during annotation process

Each time the code was executed, the model with the highest accuracy in terms of mAP50 (mean Average Precision at 50%), along with its corresponding JSON file were generated and stored in the dataset folder. Additionally, the other parameters mentioned above were systematically varied - in conjunction with the practical application of non-maximum suppression (NMS) - during each run to elucidate their impact on the accuracy of the respective model. NMS is a technique applied after object detection to sift through numerous overlapping bounding boxes generated by a model, retaining only the most assured ones.

In the context of object detection, Non-Maximum Suppression (NMS) is crucial for refining the predictions made by a model. When an object detection model analyzes an

image, it often generates multiple bounding boxes that overlap, resulting in redundant detections of the same object. NMS addresses this issue by filtering out less accurate bounding boxes and retaining only the most confident ones.

The NMS process involves several steps. First, the model assigns a confidence score to each bounding box, and these boxes are then sorted from highest to lowest based on their scores. Starting with the highest-scoring box, NMS iteratively selects this box and suppresses all other overlapping boxes whose Intersection over Union (IoU) with the selected box exceeds a certain threshold. This suppression process continues until all boxes are either selected or discarded.

NMS is significant for several reasons. It reduces redundancy by eliminating overlapping bounding boxes, thus enhancing the model's precision. Retaining only the most confident detections helps improve the overall accuracy of the model. This is particularly important in construction site monitoring, where accurately detecting objects such as safety helmets and reflective jackets is critical. Furthermore, applying NMS can lead to significant improvements in performance metrics like mean Average Precision (mAP) and F1 score, as it ensures that detections are more reliable and less cluttered.

In this research work, the NMS threshold was systematically varied to observe its impact on model performance. The results indicated that adjusting the NMS value could significantly affect the precision, recall, and overall F1 score of the detection model. For instance, selecting an NMS threshold of 0.4 resulted in a notable improvement in the model's accuracy, highlighting the importance of fine-tuning this parameter.

In summary, NMS is an indispensable technique in object detection, enhancing the reliability and accuracy of models by ensuring that only the most probable detections are considered. This study's findings underscore the critical role of NMS in achieving robust performance, particularly in the context of construction object detection models.

During the training of each model, several key metrics were evaluated, including mAP50, precision, recall, and mAP50-95. These metrics provided valuable insights into the model's performance. However, it is important to note that some of these metrics were not automatically saved during the training process.

Nicolas Nicolaou

Specifically, precision serves as a fundamental metric for evaluating the accuracy of positive predictions made by the model. It quantifies the model's ability to correctly identify positive instances among all instances predicted as positive. In the context of construction object detection, precision is crucial as it helps ensure that identified objects such as beams, columns, and safety gear are indeed present, thereby reducing false alarms that could lead to unnecessary inspections or safety checks. Precision is derived from the following relationship:

$$Precision = \frac{True\ Positives}{True\ Positives + False\ Positives} \quad [3.2]$$

On the other hand, recall, also known as sensitivity or true positive rate, assesses the model's capacity to identify all relevant instances of a particular class. It measures the proportion of true positives that the model correctly identifies out of all actual positives. The recall score is computed by dividing the number of true positives by the sum of true positives and false negatives (*Equation 3.3*). High recall is essential in construction applications where missing a true positive, such as failing to detect a safety hazard, could have significant implications for site safety and compliance.

$$Recall = \frac{True\ Positives}{True\ Positives + False\ Negatives} \quad [3.3]$$

By using these two terms, it is possible to calculate another widely used metric for evaluating classification models. The F1 score, often referred to as the harmonic mean of precision and recall, provides a balanced assessment of the model's performance. It captures the trade-off between precision and recall, offering a single metric to evaluate a model's effectiveness. This metric is particularly useful in construction applications where both precision and recall are equally critical, ensuring the model accurately identifies and classifies construction objects and hazards. The F1 score is computed using the formula:

$$F1\ Score = 2 * \frac{Precision * Recall}{Precision + Recall} \quad [3.4]$$

Nicolas Nicolaou

The term ‘mAP’ (mean Average Precision) is a metric that assesses the precision-recall tradeoff of a model. It evaluates how well a model performs at different confidence levels in its predictions. Specifically, ‘mAP50’ evaluates the model’s precision and recall at a specific 50% Intersection over Union (IoU) threshold. Higher mAP50 values indicate better performance, with a maximum value of 1.0 representing perfect precision and recall at the specified IoU threshold. IoU is a metric that measures the overlap between the predicted bounding box and the actual location of the object. A 50% IoU means there is at least a 50% overlap between the predicted and actual contexts. The accuracy metric of mAP offers insights into the model's ability to detect construction objects of varying sizes and complexities within an image, such as identifying smaller items like safety helmets in addition to larger objects like excavators.

Understanding and optimizing these metrics are essential for improving the effectiveness of custom trained models in practical applications. Precision, recall, and the F1 score allow for a balance between minimizing false positives and false negatives, while mAP provides a comprehensive evaluation of the object detection models' performance under various detection challenges. Utilizing these metrics enables the refinement of the models, ensuring they are well-suited for a range of use cases and environments.

This evaluation system is commonly used in assessing object detection models, including those trained for custom object detection tasks. At a later stage, this model was employed in another code, where its effectiveness in custom object detection examined and subsequently trained classes was evaluated using mainly unseen data, alongside the selection of a specific value for NMS and the adjustment of various rendering options. A summary flowchart of this methodology is presented in Figure 3.4.

Nicolas Nicolaou

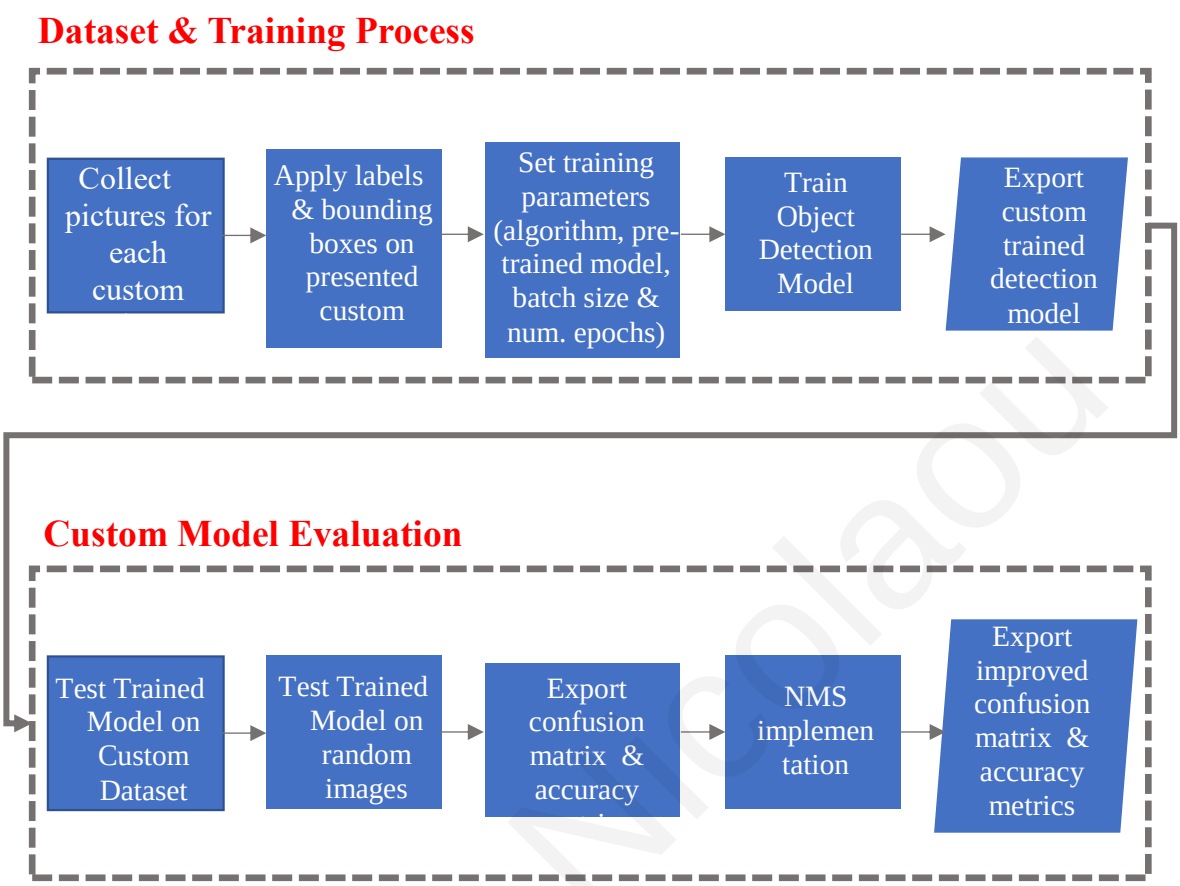


Figure 3.4: Custom object detection methodology flowchart

Additionally, the capability was provided by creating a Python code in combination with trained detection models to identify health and safety issues in photographs from construction sites, specifically printing a relevant warning message for the absence of a part or of all of the necessary safety equipment (protective helmet and reflective jacket) in case a person is detected in those areas. Specifically, the code initially examined the presence or absence of the aforementioned objects, and then examined the potential high overlap (IoU > 90%) of their defined bounding boxes with those of the detected human figure/s (indicating the adoption of this specific security measure by the individual). The high effectiveness of this additional function was verified through the execution of various tests on trained and random photos including some or all of the relevant classes.

Nicolas Nicolaou

4. ANALYSIS AND RESULTS

In the pursuit of fulfilling the objectives of this work, a series of tests were conducted through the execution of custom training Python programming language codes, as previously described. Throughout these tests, specific parameters were systematically varied in each training code, including the dataset itself, to generate two models - one for each task - with the highest accuracy and optimal performance. These models aimed to best fulfill the intended purpose for which they were created.

4.1 Image Classification Results

The final analysis results for the nine classes described in the previous stage are as follows. For image prediction, considering the case of nine classes, a *MobileNetV2* model achieved an accuracy of 81.06%. This relatively high accuracy indicates the near certainty of the model in the correctness of its predictions, specifically in successfully predicting the nine trained classes in any given photo. This result was further validated by the model's performance on various photos, consistently yielding generally high probabilities for correctly predicting the depicted object. The model was tested on both trained and random images, and during the conducted tests, no significant change in performance was observed between these two categories of images. It is thus evident that the model's performance was proportional to its accuracy rate. It should be noted that the listed per image object-classification probabilities add up to 100%. Thus, when multiple object classes are detected within an image, the reported class probabilities are lower in value compared to cases where object classes are fewer. As a result, in several tests performed on both trained and random images, notable fluctuations were observed in the prediction rates between the displayed classes, with some classes showing low rates. Some example results are provided below.

Nicolas Nicolaou

EXAMPLE 1:

```
beam : 52.2404
column : 26.02
slab : 21.3805
masonry : 0.1741
person : 0.1149
Safety_Helmet : 0.0305
excavator : 0.0174
Reflective_Jacket : 0.0142
window : 0.0081
```

Figure 4.1: Prediction percentages per class by custom IC model (example 1)

Discussion:

In the initial instance of image classification, the presented figure depicts the outcomes derived from analyzing a random image showcasing various objects on a construction site, notably columns, beams, and slabs. Employing the aforementioned image classification model, probabilities were assigned to the objects identified in the photo, with a beam receiving a probability of 52.24%, a column 26.02%, and a slab 21.38%. Predictions for the remaining classes were notably low, aligning with expectations given that only these three classes were prominently featured in the image under examination.

The variance in probabilities assigned to the three main predicted classes can be attributed to factors such as the viewing angle of each class within the photo and the inherent similarity between certain classes in appearance or features (e.g., beam versus column). Such similarities may challenge the model's ability to accurately distinguish between them, resulting in lower confidence scores for certain classes.

Nevertheless, the overall outcome underscores the model's robust predictive capabilities, despite the nuanced challenges presented by the image's composition and the similarities between certain classes.

Nicolas Nicolaou

EXAMPLE 2:

Figure 4.2: Prediction percentages per class by custom IC model (example 2)

Discussion:

Figure 4.2 presents the outcomes derived from analyzing a random photograph featuring masonry and window elements, both of which are equally prominent. Notably, the prediction percentage for these two primary classes is relatively high and nearly equal, while predictions for other classes not represented in the image are minimal. Specifically, the model assigned a probability of 50.79% to the window class and 44.41% to the masonry class. In this instance, the custom image classification model demonstrated its commendable accuracy, maintaining balance in the prediction probabilities between the two primary classes, due to the absence of significant similarities in appearance or features between masonry and window elements, along with the low number of detected classes.

Nicolas Nicolaou

EXAMPLE 3:

```
Reflective_Jacket : 49.1006
person : 38.3086
Safety_Helmet : 12.3392
excavator : 0.2281
beam : 0.0159
masonry : 0.0066
slab : 0.0006
window : 0.0003
column : 0.0001
```

Figure 4.3: Prediction percentages per class by custom IC model (example 3)

Discussion:

The image depicted in Figure 4.3 pertains to the presence of a worker adorned with appropriate personal protective equipment at a construction site. The model's analysis of this scenario yielded successful predictions for the three primary classes evident in the photograph, albeit with notable fluctuations in prediction percentages.

Specifically, the reflective jacket class garnered the highest prediction percentage at 49.10%, followed by the person class at 38.31%, and the safety helmet class registering a significantly lower percentage at 12.34% among these three categories. The substantial variance in these probability values can be attributed to the inherent similarity between the images used for training the model across these three classes. Notably, person, safety helmet, and reflective jacket are classes frequently observed together within a construction site environment.

However, it is to some extent expected that there will be significant variations in percentage values and relatively low prediction rates for some classes. This occurs because the prediction rates for all detected objects in an image collectively add up to 100%, leading to lower individual prediction rates when multiple objects are present.

Nicolas Nicolaou

EXAMPLE 4:

```
excavator : 75.1408
Reflective_Jacket : 23.9794
Safety_Helmet : 0.5158
beam : 0.3572
masonry : 0.0047
person : 0.0012
slab : 0.0007
column : 0.0002
window : 0.0001
```

Figure 4.4: Prediction percentages per class by custom IC model (example 4)

Discussion:

Figure 4.4 illustrates the outcomes derived from an image featuring a worker equipped with appropriate personal protective gear in proximity to an excavator. The model accurately identified the presence of the excavator and the safety vest; however, it failed to recognize the worker and the safety helmet.

One potential explanation for the notably low prediction rates pertaining to the aforementioned classes within the corresponding photograph could be attributed to intense lighting conditions, particularly around the area where the safety helmet is situated, compounded by the posture of the human subject. Additionally, the similarity between the classes "person," "safety helmet," and "reflective jacket" may pose a challenge for the model in distinguishing between them, consequently resulting in diminished confidence scores for some of these classes.

Nicolas Nicolaou

4.2 Object Detection Results

Accordingly, for object detection, a *YOLOv3* model with an average accuracy (mAP) of 67.41 % was achieved. The obtained model recorded also F1-Score accuracy roughly above 65% as shown in Table 4.1. This figure indicates the relatively average to good accuracy of the specific model in terms of detecting, bounding and successfully classifying the objects under study in examined photographs, however efforts are being made to enhance the performance of this model to achieve even higher success rates. This result was further validated by the model's performance on various photos, where several satisfactory results were observed in terms of the true positive detection and classification of objects. The *YOLOv3* model was tested on both trained and random images, and during the conducted tests, no significant change was observed in terms of the model's performance between these two categories of images. Therefore, in this case as well, it is evident that the performance of the model is proportional to its accuracy rate. Some example results are provided below.

In object detection models, it is common to encounter overlapped bounding boxes for each detected object. To address this issue, a common practice is the implementation of Non-Maximum Suppression (NMS), which, according to its threshold, removes overlapped bounding boxes and retains only those with the highest confidence scores for each detected object. To improve the outcomes with the implementation of the custom object detection model, various NMS thresholds were applied, either to each image result or to the overall validation set. This was done to understand how different values of NMS affect the performance and accuracy of the trained model across different metrics.

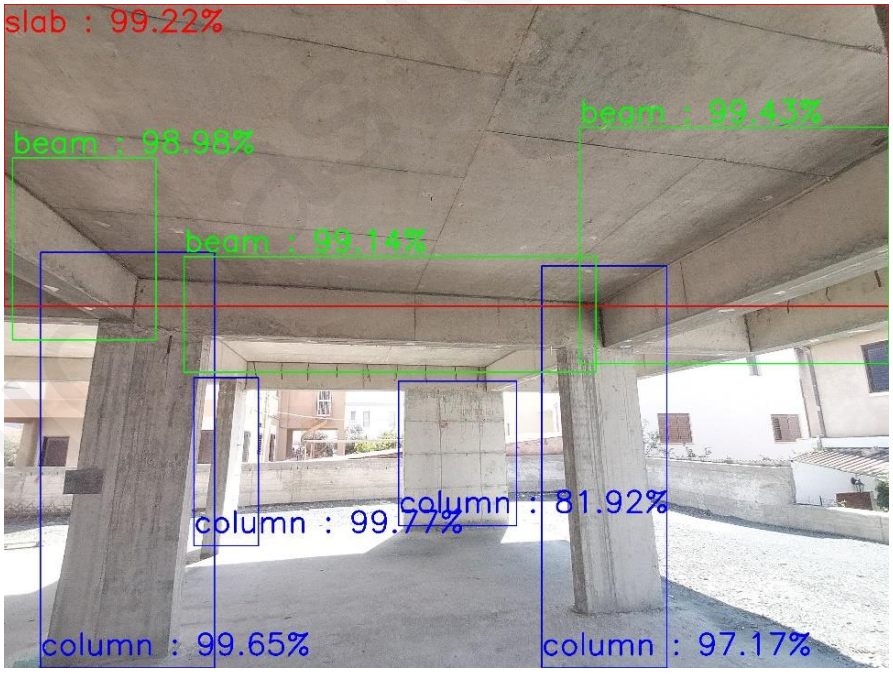
Table 4.1: Custom object detection model's metrics

MODEL ALGORITHM	PRECISION (%)	RECALL (%)	F1-SCORE (%)	mAP0.5 (%)
YOLOV3	61.82	68.99	65.21	67.41

Nicolas Nicolaou

EXAMPLE 1:

(a)



(b)

Figure 4.5: Custom OD model implementation (example 1): (a) Initial random photo, (b) Detected objects by custom OD model

Discussion:

The figure above presents the results derived from a random image depicting various objects observed within a construction site, encompassing columns, beams, and slabs. Employing the aforementioned object detection model, the analysis revealed the successful detection of four columns, three beams, and one slab, each with a classification probability exceeding 97%, thus affirming the model's robust performance. Notably, only one column yielded a positive detection probability below 97%; however, even in this case, the probability remains relatively high, indicating a favorable outcome.

However, certain other objects anticipated to be detected by the model were not identified. This discrepancy could potentially be attributed to occlusion phenomena resulting from the specific angle at which the photograph was captured. Nonetheless, the overall results remain largely consistent with those obtained using the classification model, thus warranting a satisfactory evaluation. NMS application with a threshold value of 0.05 was also necessary for obtaining the presented results.

Table 4.2: Detected objects by custom OD model (example 1)

No. OBJECT DETECTED	CLASS	CONFIDENCE SCORE (%)
1	COLUMN	99.65
2	COLUMN	99.77
3	COLUMN	81.92
4	COLUMN	97.17
5	BEAM	98.98
6	BEAM	99.14
7	BEAM	99.43
8	SLAB	99.22

Nicolas Nicolaou

EXAMPLE 2:

(a)

(b)

Figure 4.6: Custom OD model implementation (example 2): (a) Initial random photo, (b) Detected objects by custom OD model

Table 4.3: Detected objects by custom OD model (example 2)

No. OBJECT DETECTED	CLASS	CONFIDENCE SCORE (%)
1	SAFETY HELMET	98.90
2	REFLECTIVE JACKET	98.76
3	PERSON	96.94

Discussion:

Figure 4.6 illustrates the outcomes derived from a random photo capturing a worker on a construction site equipped with appropriate personal protection measures. The model successfully detected the three main classes depicted in the photo, with corresponding confidence scores exceeding 96%. This aligns with the results obtained from the classification model applied to the same photo, as previously presented. Notably, in the case of custom object detection, no issues were observed with the trained model, despite the inherent similarity among photos featuring these three classes in the training dataset. During the annotation process, all objects of different classes observed in each photo were duly annotated, contributing to the model's effective performance. Additionally, the NMS technique was applied with a threshold set to 0.35 to enhance the model's outcomes.

Nicolas Nicolaou

EXAMPLE 3:

(a) (b)

Figure 4.7: Custom OD model implementation (example 3): (a) Initial random photo, (b) Detected objects by custom OD model

Discussion:

The image showcased in Figure 4.7 captures a scene from a construction site, focusing on a window encased by brickwork. Through the utilization of the trained object detection model on this photo, both masonry structures and the window were successfully identified, each boasting a confidence score surpassing 94%. This outcome underscores the effectiveness of the model under these particular conditions, reaffirming its capability to accurately discern and classify objects within complex and detailed construction environments. The overall result mentioned above was achieved in conjunction with the implementation of NMS with a threshold value of 0.2.

Table 4.4: Detected objects by custom OD model (example 3)

No. OBJECT DETECTED	CLASS	CONFIDENCE SCORE (%)
1	MASONRY	99.78
2	MASONRY	99.74
3	MASONRY	94.82
4	WINDOW	98.13

Nicolas Nicolaou

EXAMPLE 4:

(a)

(b)

Figure 4.8: Custom OD model implementation (example 4): (a) Initial random photo, (b) Detected objects by custom OD model.

Table 4.5: Detected objects by custom OD model (example 4)

No. OBJECT DETECTED	CLASS	CONFIDENCE SCORE (%)
1	SAFETY HELMET	99.39
2	REFLECTIVE JACKET	95.24
3	PERSON	72.86
4	EXCAVATOR	97.94

Discussion:

The preceding figure illustrates an example portraying various objects pertinent to the examination, with the custom-trained object detection model delivering commendable outcomes for the majority of these objects. Particularly, it demonstrates a moderate to good result for a singular class, namely "person." Notably, an enhanced performance of the detection model is discernible in this image when juxtaposed with its classification counterpart (Figure 4.4), showcasing notably higher prediction rates. This instance serves as additional confirmation that the trained detection model remains unaffected by the resemblance among the train photos used for three of the four identified classes above (“safety helmet”, “reflective jacket” and “person”). In the aforementioned case, a NMS threshold of 0.3 was implemented to attain sharper results.

Nicolas Nicolaou

NMS CORRECTION EXAMPLE:

(a)

(b)

(c)

Figure 4.9: Custom OD model implementation (NMS example): (a) Initial random photo, (b) Detected objects by custom OD model without NMS, (c) Detected objects by custom OD model with NMS and rendering settings

Nicolas Papadopoulos

Discussion:

In Figure 4.9, a comparison is made between the output of a custom object detection model on a random image with and without the application of NMS (Non-Maximum Suppression). The initial output exhibits numerous predicted bounding boxes overlapping multiple times on the detected objects, each bearing a high confidence score for the overlapping detected object, labeled with the same class and accompanied by reduced readability of the results. To refine the final outcome, NMS was applied in conjunction with several rendering settings to achieve sharper and accurately adjusted results on the specified image. In this particular case, the NMS threshold was set to 0.2 to attain the clear and satisfactory result.

In more detail, the initial output without NMS shows a cluttered visualization, where some of the detected object are surrounded by multiple bounding boxes. This can cause confusion and makes it difficult to accurately interpret the results, as the same object might appear multiple times in different positions, each with a slight variation in the bounding box coordinates. This issue arises because the object detection model, by default, generates bounding boxes for every potential detection with a confidence score above a certain threshold. However, these overlapping boxes can obscure the true position and scale of the detected objects. To mitigate this problem, NMS is employed. This technique works by retaining the bounding box with the highest confidence score for each detected object and suppressing all other overlapping boxes with lower confidence scores. In addition, NMS ensures that only the most relevant and accurate bounding boxes are displayed, significantly improving the readability and interpretability of the results. By setting the NMS threshold to 0.2, the model discards any bounding boxes that overlap significantly with the highest confidence box, ensuring a cleaner and more precise detection.

Moreover, the rendering settings were fine-tuned to complement the NMS application. These settings include adjusting the line thickness, color, and transparency of the bounding boxes and labels to enhance visibility. As a result, the final output provides a clear, concise, and accurate representation of the detected objects, making it easier to assess the model's performance. The approach of using NMS with varying thresholds and rendering settings was consistently applied across other images analyzed in this section.

Nicolas Nicolaou

HEALTH & SAFETY EXAMPLE:

(a) (b)

Figure 4.10: Custom OD model implementation (H&S example): (a) Initial random photo, (b) Detected objects by custom OD model.

Figure 4.11: Python code output (H&S example)

Discussion:

Using the same detection model, results were obtained from other construction site photographs, with an additional capability introduced: the detection and notification of safety and health issues concerning the necessary and recommended personal protective measures on the construction site. Specifically, if a person was detected in these photographs without either or both of the reflective jacket and safety helmet, a corresponding warning was issued, as effectively demonstrated in the subsequent figure.

In Figure 4.10, two persons are depicted within a construction site, one wearing a protective helmet and the other not wearing any personal protective equipment. Therefore, this case serves as a prime example of a situation where issues regarding compliance with safety and health regulations may arise on the construction site. Upon the introduction of the photograph and the utilization of the trained object detection model, the two individuals and the protective helmet were correctly identified, with classification success rates exceeding 98%. Through the execution of the specialized Python code tailored for cases such as this, utilizing the aforementioned photograph, a warning (Figure 4.11) was appropriately issued regarding the presence of safety and health concerns, as one or more individuals failed to adhere to all required measures of personal protection (i.e., "safety helmet" and "reflective jacket").

Table 4.6: Detected objects by custom OD model (H&S example)

No. OBJECT DETECTED	CLASS	CONFIDENCE SCORE (%)
1	SAFETY HELMET	99.01
2	PERSON	98.62
3	PERSON	99.59

Nicolas Nicolaou

CONFUSION MATRIX:

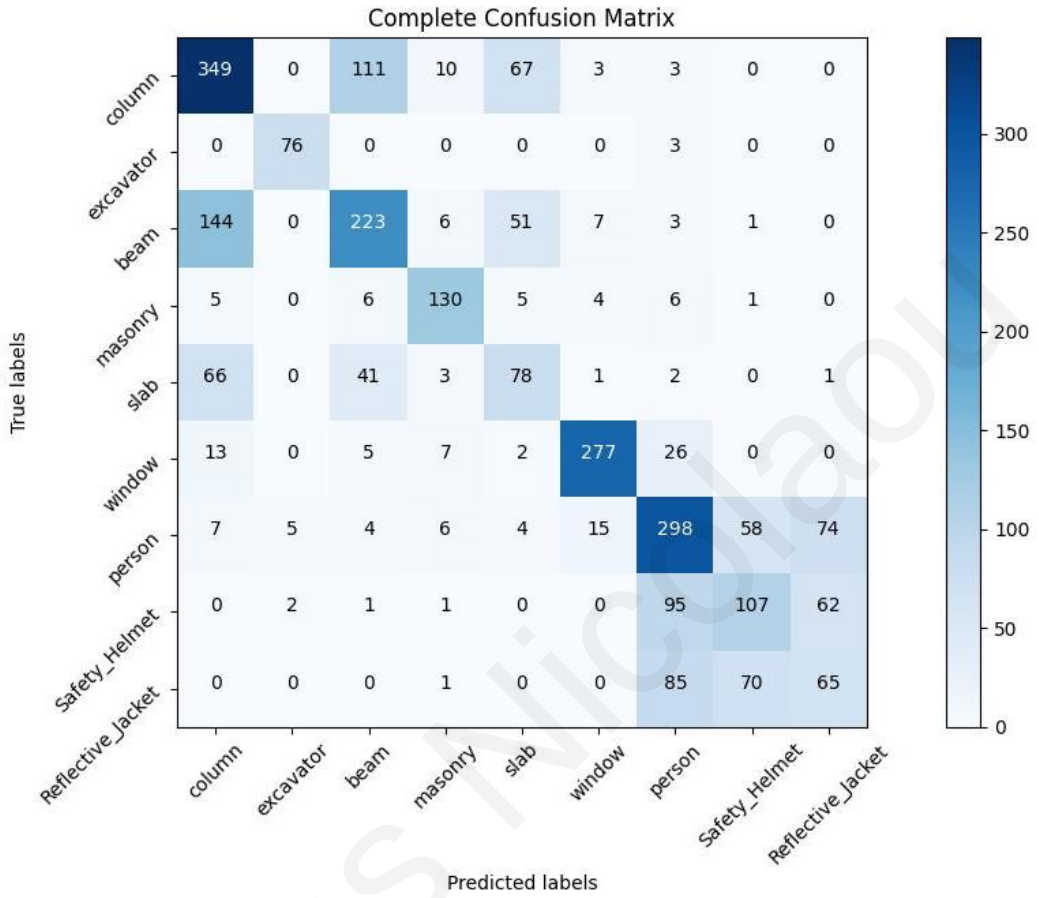


Figure 4.12: Results of confusion matrix based on custom OD model

Discussion:

Figure 4.12 presents the confusion matrix of the detection model examined in the study. In a confusion matrix, each row represents the actual labels depicted in the validation set images, while the columns represent the corresponding labels predicted by the detection model. From the presented matrix, it is observed that some classes are positively evaluated due to a high number of true positive detections (e.g., excavator, window, etc.), while others are characterized as moderate to negative. The results of the matrix are to some extent expected, as the examined detection model did not achieve particularly high levels of accuracy. Essentially, τhe matrix provides insights into which classes the model

Nicolas Nicolaou

struggles to predict accurately and can guide further improvements in the model, such as fine-tuning class-specific features or collecting more diverse training data for those classes. Therefore, there is room for significant future improvements.

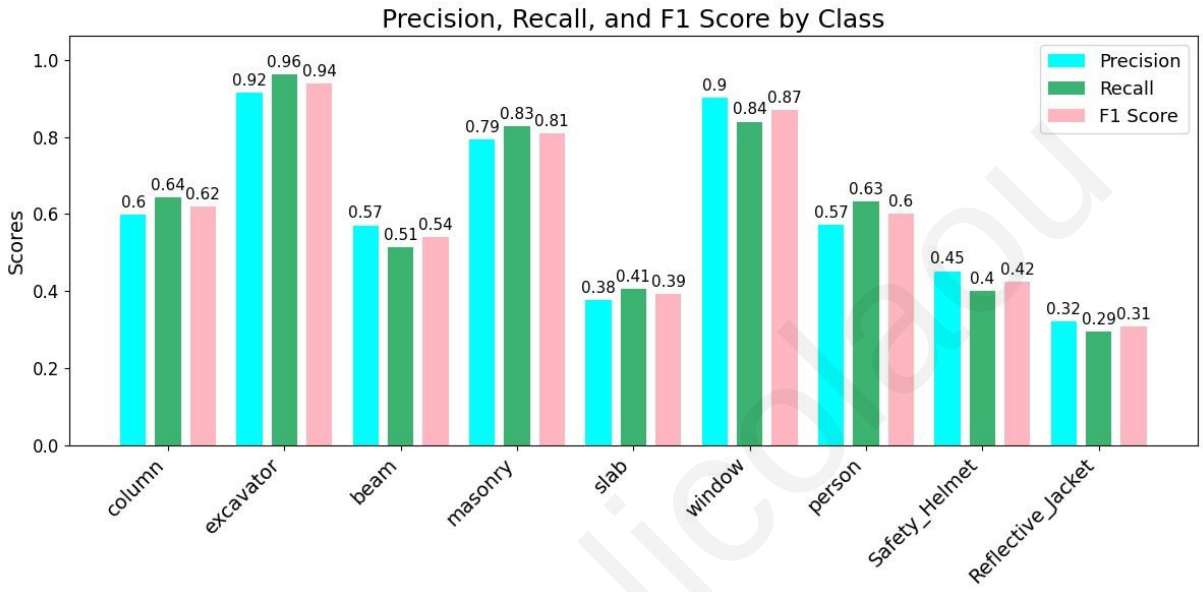


Figure 4.13: Custom object detection model's metrics (per class and overall)

A better visualization of the results and information contained in the confusion matrix presented in Figure 12 is provided in Figure 13. Using the extracted matrix and a specially configured Python code, the precision values of the trained model against each class were calculated, utilizing measurement metrics such as precision, recall, and the derived F1-Score. Specifically, a higher F1-Score value was recorded for the class "excavator" (94%), while the lowest was for the class "Reflective Jacket". High precision values were also recorded for the classes "window" and "masonry", moderate values for the classes "column", "person", and "beam", while low precision F1-Score values were recorded for the remaining classes. From the above prediction graph of the trained model against the true labels of the validation set used for training purposes, an overall F1-Score of 61.11% was obtained, a result deemed moderate to good.

With the aim of improving the accuracy and performance of the custom object detection trained model, the practice of NMS was applied—initially at various values—on the initial

Nicolas Nicolaou

predictions of the model on the validation set. The use of different NMS values aimed at a deeper understanding of its impact on the number of model predictions as well as the overall accuracy in terms of precision, recall, and F1-Score, and consequently by extracting corresponding confusion matrices.

From the experiments conducted for this particular model, an increase in its accuracy values was observed with the decrease in NMS values, which is highly positive and entirely expected. However, this positive impact of low NMS values was counteracted by the dramatic decrease in the instances for each class, as this practice removes a large number of instances and leaves only those with the highest confidence scores, resulting in even fewer instances than the truth labels.

Upon the conclusion of the experiments, the decision was made to select an NMS value of 0.4. This value led to an increase in the overall accuracy of the model in terms of F1-Score by approximately 30%. As we can see in graph below, instantly the predicted labels for each class and overall were more than truth labels, especially for some classes like column and person.

Although, after the implementation of NMS practice with a threshold = 0.4, a significant drop of instances per class was observed as many overlapped bounding boxes and mainly those with the relatively lowest confidence scores were removed. However, the certain value of NMS has led to have less predicted instances than truth labels. This drop between truth and NMS predicted labels was relatively high for all classes, except class “excavator”.

Nicolas Nicolaou

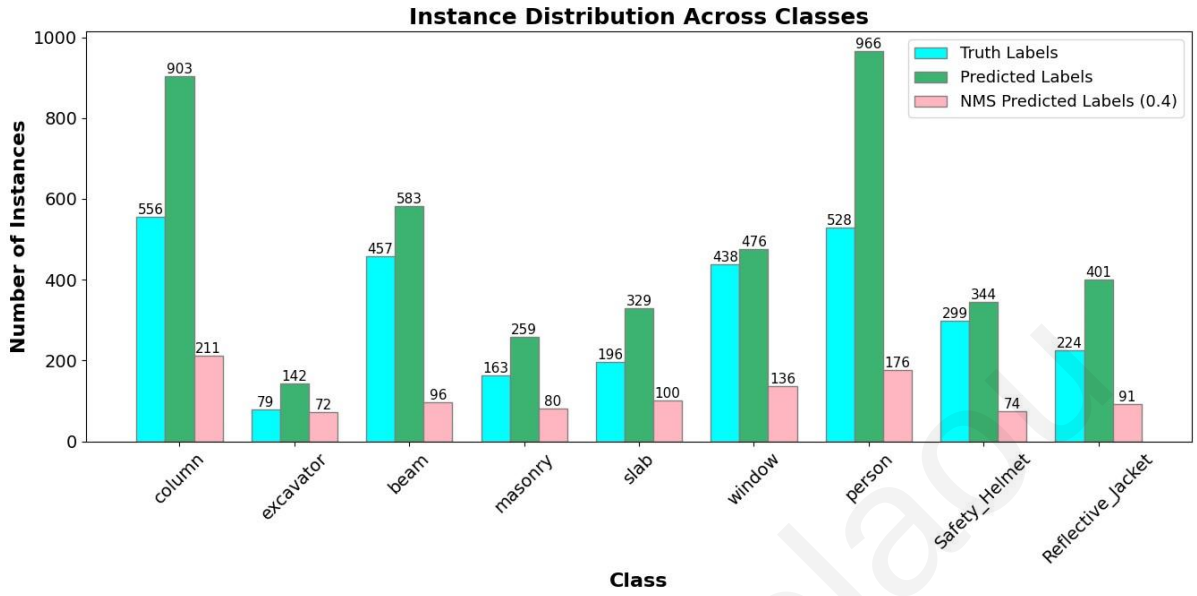


Figure 4.14: Instance distribution before and after NMS application

The impact of applying NMS with a threshold of 0.4 on the accuracy and performance of the custom trained model was observed through the confusion matrix and the resulting metrics derived from its values. As illustrated in Figure 4.14, the total instances per class are notably fewer compared to the confusion matrix depicted in Figure 4.12. Additionally, the matrix exhibits a more diagonal pattern (Figure 4.15), indicating that more classes have predominantly higher accuracies (around 70%), while fewer classes exhibit moderate accuracies (around 50-60%). This observation suggests that the application of NMS with a threshold of 0.4 has led to a refinement in the model's performance, with a clearer delineation between classes and improved overall accuracy.

The analysis of the confusion matrix shown in Figure 4.15, reveals a noteworthy enhancement in the model's overall performance, quantified by an F1-Score of 79.69% (Figure 4.16). This improvement is particularly significant when considering the individual classes. Notably, classes that previously exhibited lower accuracy with the model's initial predictions experienced substantial boosts in their F1-Score values. Specifically, both "masonry" and "window" classes achieved notably higher F1-Score values, reaching an impressive 94%. This indicates a marked improvement in the model's ability to accurately detect and classify these objects within the images. Furthermore, the majority of the

Nicolas Nicolaou

remaining classes also demonstrated commendable F1-Score values, surpassing the 70% threshold. This indicates a consistent improvement across multiple object categories, reflecting the efficacy of the adjustments made to the model.

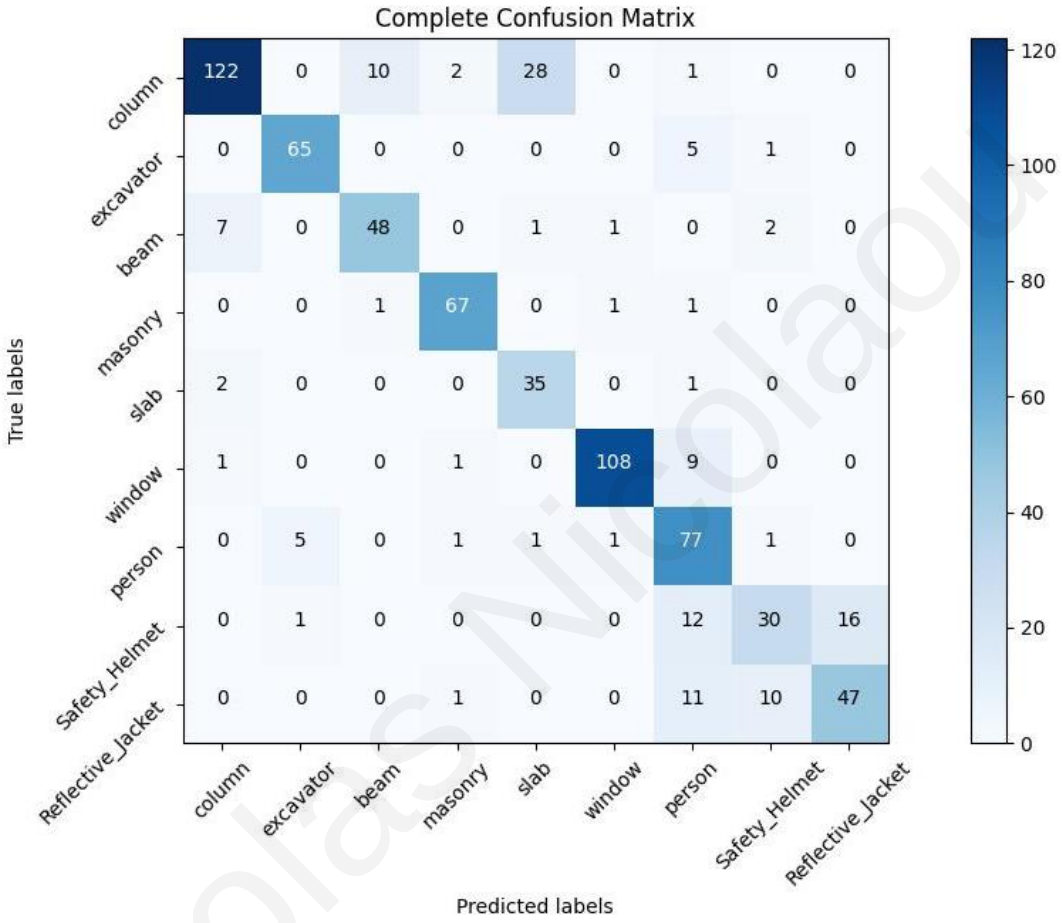
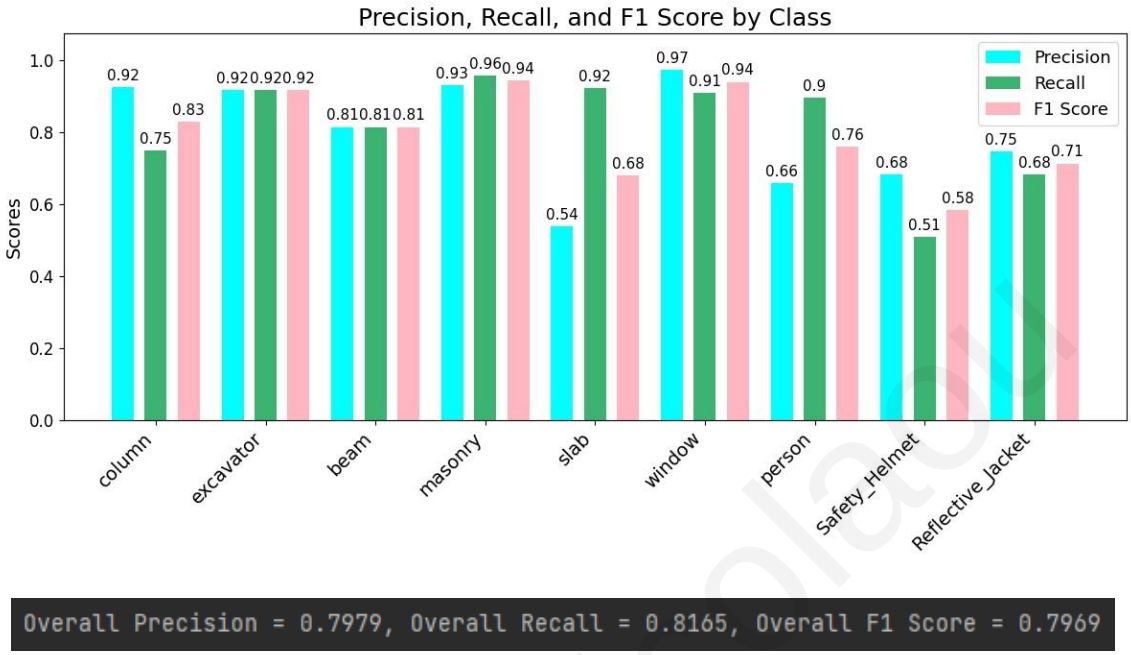


Figure 4.15: Results of confusion matrix based on custom OD model after NMS application

However, it is worth noting that the class "safety helmet" registered the lowest F1-Score value in this analysis, recording a value of 58%. While this falls within the realm of moderate accuracy, it underscores the ongoing need for refinement, particularly in accurately detecting and classifying instances of safety helmets within the images. Overall, these findings underscore the effectiveness of the model refinement efforts, leading to substantial enhancements in accuracy across various object classes.

Nicolas Nicolaou




```
Overall Precision = 0.7979, Overall Recall = 0.8165, Overall F1 Score = 0.7969
```

Figure 4.16: Custom object detection model's metrics after NMS application (per class and overall)

Nicolas Nicolaou

5. SUMMARY OF FINDINGS

Upon examining the results derived from the two trained models (image classification & object detection) in the previous section, several observations and conclusions regarding their accuracy, performance, and reliability can be made.

Firstly, in the image classification model, which achieved a high accuracy rate (81%) during training, a significant validation of this accuracy was observed in the four examples examined. Specifically, in all cases except Example 4, the depicted objects exhibited significant prediction rates, with each object class receiving the majority share of the one unit of prediction (100%) available each time, thereby avoiding the presence of significant percentages in classes that were not depicted in each photograph.

Some significant challenges identified during the application of the trained model included the notable fluctuations in prediction rates among the depicted classes, and more specifically, false negative predictions of classes in the photographs. One possible explanation for both phenomena appears to be the model's inability to accurately describe the content of the examined photograph when depicting classes with similar characteristics, such as beams and columns, or the combination of classes for which similar photographs were used in the trained dataset, as they often coexist on the construction site, such as the classes "safety helmet", "reflective jacket", and "person". Another possible explanation seems to be the presence of obstacles in front of the depicted objects, specific lighting conditions of the images, and even the angle of capture. Another significant factor that influenced both the degree of variation and the prediction percentages for each class was that these percentages cumulatively summed to 100% for the entire content of each examined photograph. This effect became more pronounced as the number of identified classes increased.

In any case, to address the aforementioned issues in the results of the trained image classification model, it is recommended, as part of future work, to modify or add to the existing trained dataset in order to achieve a better balance between the number of instances and greater variety of images for each class.

Nicolas Nicolaou

Similarly, with the implementation of the trained object detection model with an accuracy in terms of mAP0.5 of 67.41%, which is considered moderate to high, a plethora of results emerged that significantly validated the reliability and effectiveness of the model using random photographs, most of which were also used in image classification for result comparison. In this model, very positive results were recorded as the highest number of depicted classes were identified in all examples, even in cases where a large number of classes were depicted, and objects of the same class were identified in the same photograph. Additionally, prediction rates for each detected object exceeded 90% in the majority of cases.

However, similar to the image classification model, the object detection model also encountered a series of issues, comparable in magnitude to the accuracy of the model. Specifically, it was found that objects covering a smaller portion of the examined photograph or objects with significant obstacles preventing their full depiction recorded lower rates of successful prediction compared to other objects of the same class, or worse, were not detected at all (see windows in example 1). One possible explanation appears to be the methodology followed during the process of assigning bounded boxes during the preparation of the training data, where objects under poor lighting conditions, with obstacles, or with relatively small covering surface areas were avoided in order to simplify and facilitate the model training on the examined objects.

Nonetheless, unlike the image classification model, the results of the object detection model were not influenced by similar characteristics or training images among the detected classes. This was mainly because, for this activity, the training images were not separated per object; instead, bounding boxes were applied to objects of all depicted classes simultaneously. Additionally, this model was used for detecting safety and health issues on construction sites, achieving satisfactory results.

Furthermore, an extensive application of the Non-Maximum Suppression (NMS) technique was conducted on the extracted results to remove any overlapping bounding boxes of the same class and to achieve more distinct results. Moreover, precision, recall, and F1-Score were evaluated on the validation set using various NMS values, where it was demonstrated

Nicolas Nicolaou

that using NMS=0.4 achieves the optimal precision result and number of instances for this trained model.

In conclusion, both prediction models, one for each activity, demonstrate relatively good accuracy values, as verified by their application to random photographs. However, they are accompanied by a significant number of challenges and observations that require thorough study and addressing in future work. This aims to simultaneously increase the achieved accuracies and utilize more recent and advanced learning algorithms (e.g., *YOLOv8*).

Nicolas Nicolaou

6. CONCLUSIONS

The utilization of artificial intelligence, particularly technologies such as Machine Vision (MV) and Deep Learning (DL), in the construction industry is deemed imperative. The applications and benefits that can arise from these technologies are crucial, especially during the transition to a new era fraught with challenges. Activities such as image classification and object detection consistently prove to be extremely valuable tools, enhancing, improving, and facilitating numerous practical and technical tasks on construction sites. The real-time application of these technologies can enhance the monitoring of safety and health issues on construction sites, extending to the broader and more essential oversight of labor management, mechanical equipment, vehicles, and materials, all while considering the relatively low costs resulting from the use of these technologies.

The present study focused on the automated detection and classification of construction elements at construction sites using the *ImageAI* library, built on the foundation of Python's *TensorFlow* and *Keras* libraries. The entire process was based on the integration of Machine Vision and Deep Learning technologies, combined with a dataset collected for the objects under consideration. The extracted results were analyzed in relation to the accuracy of the corresponding models from which they were derived.

As part of future work, the following actions are to be taken to enhance the performance and accuracy of the relevant models based on *ImageAI*:

- Quantitative and qualitative expansion of the dataset, encompassing a greater variety of objects. To incorporate more construction classes, the methodology employed for the nine classes examined in this research could be extended. These additional classes might encompass various items categorized into groups such as mechanical equipment, construction materials, vehicles, and personnel. Procuring images depicting these new classes would be essential. However, this endeavor would entail augmented computational resources and incur higher costs, particularly contingent upon the quantity of additional classes. Mitigating this, existing training images featuring these classes could be utilized, or new images

Nicolas Nicolaou

could be generated using the Augmentor Python library (a code example is provided in the Appendix), ensuring diversity within the dataset without redundancy. Additionally, the use of a balanced dataset with respect to all examined objects would be crucial to prevent overfitting and the memorization of specific objects by the trained model for each task.

- In future research, a systematic approach could categorize object classes into purpose-specific groups, enhancing model applicability in construction management. For instance, machinery items like excavators, cranes, and bulldozers could be tracked for productivity analysis, while materials such as bricks, steel beams, and piping could aid in quantity/cost estimation. Additionally, structural elements like columns and walls could be monitored for construction scheduling purposes. All these applications can complement the existing capability of health and safety object tracking provided by this research, to ensure adherence to safety protocols on construction sites, which can also be expanded including more health & safety classes.

- Improvement of the annotation functions for bounded frames and labels, using advanced rendering practices in conjunction with the practical application of non-maximum suppression (NMS) to produce sharper and accurately adjusted results. Additionally, exploration and testing of other custom activities offered by the ImageAI library for accuracy and usefulness, particularly by using video streams of related content, would be beneficial.

- Exploring alternative methods, software libraries, and datasets to enhance speed and accuracy is a pivotal aspect of future research. This entails delving into newer iterations of object detection algorithms, like YOLOv8, and exploring cutting-edge architectures such as feature pyramid networks (FPN) and EfficientDet. Additionally, integrating diverse training datasets like SODA can enrich the model's capabilities by offering a broader range of training examples. Continuously assessing and adopting the latest techniques, libraries, and datasets ensures that custom object detection models remain at the forefront of innovation, delivering optimal results in terms of speed, accuracy, and reliability.

Nicolas Nicolaou

REFERENCES

[1] Lambrides, E., & Christodoulou, S.E. (2023). Human action detection and ergonomic risk assessment at construction sites, by use of machine vision and deep learning. In: EC3 Conference 2023 (Vol. 4). European Council on Computing in Construction, Crete, Greece.

[2] Market.US (2024). Specializes in in-depth market research and analysis. https://market.us/report/generative-ai-in-construction-market/

[3] Brilakis, I., Lourakis, M., Sacks, R., Savarese, S., Christodoulou, S., Teizer, J. and Makhmalbaf, A. (2010). Toward automated generation of parametric BIMs based on hybrid video and laser scanning data. Advanced Engineering Informatics, 24(4), pp.456-465.

[4] Czerniawski, T. & Leite, F. (2020). Automated digital modeling of existing buildings: A review of visual object recognition methods. Automation in Construction, 113, p.103131.

[5] Nath, N. & Behzadan, A.H. (2020). Deep generative adversarial network to enhance image quality for fast object detection in construction sites. In: 2020 Winter Simulation Conference (WSC) (pp. 2447-2459). IEEE.

[6] Paneru, S. & Jeelani, I. (2021). Computer vision applications in construction: Current state, opportunities & challenges. Automation in Construction, 132, p.103940.

[7] Duan, R., Deng, H., Tian, M., Deng, Y. & Lin, J. (2022). SODA: site object detection dataset for deep learning in construction. arXiv preprint arXiv:2202.09554.

[8] Wang, Y., Xiao, B., Bouferguene, A., Al-Hussein, M. & Li, H. (2022). Vision-based method for semantic information extraction in construction by integrating deep learning object detection and image captioning. Advanced Engineering Informatics, 53, p.101699.

[9] Hou, L., Chen, C., Wang, S., Wu, Y. & Chen, X. (2022). Multi-object detection method in construction machinery swarm operations based on the improved YOLOv4 model. Sensors, 22(19), p.7294.

[10] Zhou, Q., Liu, H., Qiu, Y. & Zheng, W. (2022). Object Detection for Construction Waste Based on an Improved YOLOv5 Model. Sustainability, 15(1), p.681.

[11] Jog, G.M., Brilakis, I.K. & Angelides, D.C. (2011). Testing in harsh conditions: Tracking resources on construction sites with machine vision. Automation in construction, 20(4), pp.328-337.

[12] Moses, O. (2018). ImageAI, an open source python library built to empower developers to build applications and systems with self-contained computer vision capabilities. https://github.com/OlafenwaMoses/ImageAI.

[13] Turing (2024). A combination of internal experts, global talent, and proprietary AI technology working together to accelerate and innovate companies and careers more efficiently. https://www.turing.com/kb/ultimate-battle-between-deep-learning-and-machine-learning

[14] Umer Yasin, M. (2022). Bricks Under Construction or Old Building / Houses, an image dataset that contains pictures of buildings and houses under construction. https://www.kaggle.com/datasets/mumeryasin/bricks-under-construction-or-old-building-houses/data
[15] Ahmadzada, A. (2020). People Image Dataset, many pictures of people performing different activities. https://www.kaggle.com/datasets/ahmadahmadzada/images2000/data
[16] B Naik, N. (2023). Safety Helmet and Reflective Jacket, images of Individuals Wearing Safety Helmets and Reflective Jackets. https://www.kaggle.com/datasets/niravnaik/safety-helmet-and-reflective-jacket
[17] Deshmukh, R., Wenguang, M. & Wei, M. (2020). Window Detection in Street Scenes, selected images from Paris Street-View Dataset with Window Annotations. https://www.kaggle.com/datasets/rude009/window-detection-in-street-scenes
[18] B Naik, N. (2022). PPU1dataset, a test dataset of concrete column and concrete beam annotated images for developing custom column and beam object detection model. https://github.com/febrifahmi/PPU1dataset
[19] Tzutalin (2015). LabelImg, a graphical image annotation tool. https://github.com/HumanSignal/labelImg

Nicolas Nicolaou

APPENDIX

Python code for custom image classification training

```
from imageai.Classification.Custom import ClassificationModelTrainer

model_trainer = ClassificationModelTrainer()
model_trainer.setModelTypAsMobileNetV2()
model_trainer.setDataDirectory("structural elements - ver.2 - IP9")
model_trainer.trainModel(num_experiments=1000, batch_size=4)
```

Python code for custom image classification prediction

```
from imageai.Classification.Custom import CustomImageClassification
import os

execution_path = os.getcwd()
prediction = CustomImageClassification()
prediction.setModelTypeAsMobileNetV2()
prediction.setModelPath(os.path.join(execution_path, "mobilenet_v2-
structural elements - ver.2 - IP9-test_acc_0.81056_epoch-257.pt"))
prediction.setJsonPath(os.path.join(execution_path, "structural
elements - ver.2 - IP9_model_classes.json"))
prediction.loadModel()
predictions, probabilities =
prediction.classifyImage(os.path.join(execution_path,
"IMG_20240410_101352.jpg"), result_count=9)
for eachPrediction, eachProbability in zip(predictions, probabilities):
    print(eachPrediction + " : " + str(eachProbability))
```

Python code for custom object detection training

```
from imageai.Detection.Custom import DetectionModelTrainer

trainer = DetectionModelTrainer()
trainer.setModelTypeAsYOLOv3()
trainer.setDataDirectory(data_directory="CustomObjectDetection")
trainer.setTrainConfig(object_names_array=['column','excavator','beam',
'masonry','slab','window','person','Safety_Helmet','Reflective_Jacket']
, batch_size=4
                       , num_experiments=200,
train_from_pretrained_model="yolov3_hololens-yolo_mAP-0.82726_epoch-
73.pt")
#download pre-trained model via
https://github.com/OlafenwaMoses/ImageAI/releases/download/3.0.0-
pretrained/yolov3.pt
# If you are training to detect more than 1 object, set names of
objects above like object_names_array=["hololens", "google-glass",
"oculus", "magic-leap"]
trainer.trainModel()
```

Nicolas Nikolaou

Python code for augmented pictures production

```
import os
import Augmentor

# Specifies the path to the directory containing original images
input_directory = r"C:\Users\Nicolas\PycharmProjects\ImageAI-
master\examples\final_entry\BEAM\RENAME JPG"

# Specifies the output directory where augmented images will be saved
output_directory = r"C:\Users\Nicolas\PycharmProjects\ImageAI-
master\examples\final_entry\BEAM\aug2"

# Checks if the output directory exists, otherwise creates it
if not os.path.exists(output_directory):
    os.makedirs(output_directory)

# Creates an Augmentor pipeline for the input directory
pipeline = Augmentor.Pipeline(input_directory, output_directory)

# Defines augmentation operations
pipeline.rotate(probability=0.7, max_left_rotation=3,
max_right_rotation=3)
pipeline.flip_left_right(probability=0.7)
pipeline.flip_top_bottom(probability=0.7)
pipeline.zoom_random(probability=0.7, percentage_area=0.8)
pipeline.flip_random(probability=0.7)

# Color Jittering
pipeline.random_color(probability=0.7, min_factor=0.7, max_factor=1.3)

# Brightness Adjustment
pipeline.random_brightness(probability=0.7, min_factor=0.7,
max_factor=1.3)

# Contrast Adjustment
pipeline.random_contrast(probability=0.7, min_factor=0.7,
max_factor=1.3)

# Sets the number of augmented images to generate
num_augmented_images = 100

# Executes the augmentation process
pipeline.sample(num_augmented_images)
```

Nicolas Nicolaou

Python code for custom object detection with NMS and rendering settings

```
import cv2
from imageai.Detection.Custom import CustomObjectDetection

# Defines rendering settings for each class
class_rendering_settings = {
    "excavator": {"color": (255, 0, 0), "label_position": "top_left"},
    "person": {"color": (204, 204, 0), "label_position":
"bottom_right"},
    "Safety_Helmet": {"color": (0, 255, 0), "label_position":
"bottom_left"},
    "Reflective_Jacket": {"color": (0, 0, 255), "label_position":
"top_right"},
}

detector = CustomObjectDetection()
detector.setModelTypeAsYOLOv3()
detector.setModelPath("yolov3_CustomObjectDetection_mAP-0.67410_epoch-
20.pt")
detector.setJsonPath("CustomObjectDetection_yolov3_detection_config.jso
n")
detector.loadModel()

# Detection with adjusted rendering options
detections = detector.detectObjectsFromImage(
    input_image="ud6.jpg",
    output_image_path="ud6_output.jpg",
    minimum_percentage_probability=30,
    display_percentage_probability=False,  # Disables displaying
percentage probability
    display_object_name=False,  # Disables displaying object names
    display_box=True,
    nms_treshold=0.35
)

# Loads the image
image = cv2.imread("ud6_output.jpg")

# Iterates through detections
for detection in detections:
    class_name = detection["name"]
    rendering_settings = class_rendering_settings.get(class_name, {})
# Gets rendering settings for the class
    color = rendering_settings.get("color", (255, 255, 255))
    label_position = rendering_settings.get("label_position",
"top_left")

    # Draws the bounding box
    left, top, right, bottom = detection["box_points"]
    cv2.rectangle(image, (left, top), (right, bottom), color, 2)

    # Calculates the center of the bounding box

    center_y = (top + bottom) // 2
```

Nicolas Nicolaou

```
    # Draws the label
    label = detection["name"] + " :
{:.2f}%".format(detection["percentage_probability"])
    label = detection["name"] + " :
{:.2f}%".format(detection["percentage_probability"])
    label_size, _ = cv2.getTextSize(label, cv2.FONT_HERSHEY_SIMPLEX, 1,
1)
    if label_position == "bottom_right":
        label_position = (right -95, top-(-60))
    elif label_position == "bottom_left":
        label_position = (right , top)
    elif label_position == "top_left":
        label_position = (right -95, center_y + label_size[1] // 2)
    elif label_position == "top_right":
        label_position = (right - 125, center_y + label_size[1] // 2)
    else:
        label_position = (right, top - 100)
    cv2.putText(image, label, label_position, cv2.FONT_HERSHEY_SIMPLEX,
0.3, color, 1)

# Saves the image with modified rendering
cv2.imwrite("ud6-detected-labeled.jpg", image)
```

Nicolas Nicolaou

Python code for custom object detection on .txt format (class, bounding boxes coordinates, confidence score)

```
from imageai.Detection.Custom import CustomObjectDetection
import os
from PIL import Image

# Creates a CustomObjectDetection instance
detector = CustomObjectDetection()

# Sets the model type to YOLOv3
detector.setModelTypeAsYOLOv3()

# Sets the path to the trained YOLOv3 model file
detector.setModelPath("yolov3_CustomObjectDetection_mAP-0.67410_epoch-
20.pt")

# Sets the path to the JSON file containing detection configuration
detector.setJsonPath("CustomObjectDetection_yolov3_detection_config.jso
n")

# Loads the YOLOv3 model
detector.loadModel()

# Path to the directory containing validation set images
validation_set_path = r"C:\Users\Nicolas\PycharmProjects\ImageAI-
master\examples\CustomObjectDetection - Results\validation\images"

# Path to the directory where the output text files with bounding box
annotations will be saved
output_annotations_path = r"C:\Users\Nicolas\PycharmProjects\ImageAI-
master\examples\CustomObjectDetection - Results\validation\predictions"

# Dictionary to map class names to numerical codes
class_mapping = {"column": 0, "excavator": 1, "beam": 2, "masonry": 3,
"slab": 4, "window": 5, "person": 6,
                 "Safety_Helmet": 7, "Reflective_Jacket": 8}

# Iterates over each image in the validation set
for image_filename in os.listdir(validation_set_path):
    if image_filename.endswith(".jpg"):
        # Builds the full path to the input image
        input_image_path = os.path.join(validation_set_path,
image_filename)

        # Opens the image and gets its dimensions
        with Image.open(input_image_path) as img:
            image_width, image_height = img.size

        # Performs object detection on the current image
        detections =
detector.detectObjectsFromImage(input_image=input_image_path,
output_image_path=None)

        # Builds the full path to save the output text file with
bounding box annotations
```

Nicolas Nicolaou

```
        annotation_filename = image_filename.replace(".jpg", ".txt")
        output_annotation_path = os.path.join(output_annotations_path,
annotation_filename)

        # Opens the output text file for writing
        with open(output_annotation_path, 'w') as annotation_file:
            # Writes each detection in YOLO format to the text file
            for detection in detections:
                class_label = detection["name"]
                numerical_code = class_mapping.get(class_label)
                if numerical_code is not None and numerical_code in
range(9):
                    x_min, y_min, x_max, y_max =
detection["box_points"]
                    x_center = (x_min + x_max) / 2 / image_width
                    y_center = (y_min + y_max) / 2 / image_height
                    box_width = (x_max - x_min) / image_width
                    box_height = (y_max - y_min) / image_height
                    confidence_score =
detection["percentage_probability"] / 100.0
                    annotation_str = f"{numerical_code} {x_center:.6f}
{y_center:.6f} {box_width:.6f} {box_height:.6f}
{confidence_score:.6f}\n"
                    annotation_file.write(annotation_str)
                else:
                    print(f"Invalid class label '{class_label}' for
{image_filename}. Skipping annotation.")
```

Nicolas Nicolaou

Python code for custom object detection on .txt format (NMS implementation)

```
import numpy as np
import os
def non_max_suppression(boxes, scores, threshold):
    # If no boxes, returns an empty list
    if len(boxes) == 0:
        return []

    # Converts boxes to numpy array
    boxes = np.array(boxes)

    # Initializes list to store the picked boxes
    picked_boxes = []

    # Extracts coordinates of bounding boxes
    x1 = boxes[:, 0]
    y1 = boxes[:, 1]
    x2 = boxes[:, 2]
    y2 = boxes[:, 3]

    # Computes the area of each bounding box
    area = (x2 - x1 + 1) * (y2 - y1 + 1)

    # Sorts the bounding boxes by their confidence scores (in
descending order)
    idxs = np.argsort(scores)[::-1]

    while len(idxs) > 0:
        # Picks the bounding box with the highest confidence score
        last = len(idxs) - 1
        i = idxs[last]
        picked_boxes.append(i)

        # Calculates the intersection over union (IoU) of the picked
box with other boxes
        xx1 = np.maximum(x1[i], x1[idxs[:last]])
        yy1 = np.maximum(y1[i], y1[idxs[:last]])
        xx2 = np.minimum(x2[i], x2[idxs[:last]])
        yy2 = np.minimum(y2[i], y2[idxs[:last]])

        w = np.maximum(0, xx2 - xx1 + 1)
        h = np.maximum(0, yy2 - yy1 + 1)

        intersection = w * h

        iou = intersection / (area[i] + area[idxs[:last]] -
intersection)

        # Discards the boxes with IoU greater than the threshold
        idxs = np.delete(idxs, np.concatenate(([last], np.where(iou >
threshold)[0])))

    # Returns the indices of the picked boxes
    return picked_boxes
```

Nicolas Nicolaou

```
# Function to apply NMS to bounding box predictions in a TXT file
def apply_nms_to_txt_file(txt_file_path, output_file_path,
nms_threshold):
    # Opens the input TXT file
    with open(txt_file_path, 'r') as input_file:
        lines = input_file.readlines()

    # Parses each line in the input file and apply NMS
    refined_lines = []
    boxes = []
    scores = []
    for line in lines:
        # Parses the line to extract bounding box coordinates and
confidence score
        class_label, x_center, y_center, box_width, box_height,
confidence_score = map(float, line.strip().split())
        # Appends the bounding box details to the lists
        boxes.append([x_center - box_width / 2, y_center - box_height /
2, x_center + box_width / 2, y_center + box_height / 2])
        scores.append(confidence_score)

    # Applies NMS to the bounding box predictions
    picked_boxes = non_max_suppression(boxes, scores, nms_threshold)

    # Writes the refined bounding box predictions to the output TXT
file
    with open(output_file_path, 'w') as output_file:
        for idx in picked_boxes:
            output_file.write(' '.join(map(str,
lines[idx].strip().split())) + '\n')

# Path to the folder containing the TXT files with bounding box
predictions
input_folder_path = r"C:\Users\Nicolas\PycharmProjects\ImageAI-
master\examples\CustomObjectDetection - Results\validation\predictions"
# Path to the folder to save the refined TXT files with NMS applied
output_folder_path = r"C:\Users\Nicolas\PycharmProjects\ImageAI-
master\examples\CustomObjectDetection - Results\validation\predictions-
nms0.4"

# NMS threshold
nms_threshold = 0.4  # Adjust as needed

# Iterates over each TXT file in the input folder
for txt_file_name in os.listdir(input_folder_path):
    if txt_file_name.endswith(".txt"):
        # Builds the full path to the input TXT file
        input_txt_file_path = os.path.join(input_folder_path,
txt_file_name)
        # Builds the full path to the output TXT file
        output_txt_file_path = os.path.join(output_folder_path,
txt_file_name)

        # Applies NMS to the bounding box predictions in the current
TXT file
        apply_nms_to_txt_file(input_txt_file_path,
output_txt_file_path, nms_threshold)
```

Nicolas Nicolaou

Python code for custom object detection model confusion matrix

```
from sklearn.metrics import confusion_matrix
import os
from collections import Counter
import numpy as np
import matplotlib.pyplot as plt

# Custom class labels
class_labels = {
    0: "column", 1: "excavator", 2: "beam", 3: "masonry",
    4: "slab", 5: "window", 6: "person", 7: "Safety_Helmet", 8:
"Reflective_Jacket"
}

# Paths
ground_truth_path = r"C:\Users\Nicolas\PycharmProjects\ImageAI-
master\examples\CustomObjectDetection - Results\validation\annotations"
predictions_path = r"C:\Users\Nicolas\PycharmProjects\ImageAI-
master\examples\CustomObjectDetection - Results\validation\predictions-
nms0.4"

# Initializes variables to store confusion matrices
conf_matrices = []

# Processes files one by one
for batch_index in range(0, len(os.listdir(ground_truth_path)), 1):
    batch_ground_truth_labels, batch_predicted_labels = [], []

    # Loads ground truth and predicted labels for the current batch
    for filename in
os.listdir(ground_truth_path)[batch_index:batch_index + 1]:
        if filename.endswith(".txt"):
            with open(os.path.join(ground_truth_path, filename), 'r')
as annotation_file:
                # Reads ground truth labels
                batch_ground_truth_labels.extend(int(line.split()[0])
for line in annotation_file.readlines())

            with open(os.path.join(predictions_path, filename), 'r') as
prediction_file:
                # Reads predicted labels
                batch_predicted_labels.extend(int(line.split()[0]) for
line in prediction_file.readlines())

    # Counts occurrences of each class in ground truth and predicted
labels
    ground_truth_counts = Counter(batch_ground_truth_labels)
    predicted_counts = Counter(batch_predicted_labels)

    # Determines the minimum count for each class
    class_min_counts = {cls: min(ground_truth_counts[cls],
predicted_counts[cls]) for cls in class_labels.keys()}

    # Creates lists to store filtered ground truth and predicted labels
    filtered_ground_truth_labels = []
```

Nicolas Nicolaou

```
    filtered_predicted_labels = []

    # Iterates over each label and keep only the required number of
instances for each class
    for label, cls in zip(batch_predicted_labels,
batch_ground_truth_labels):
        if class_min_counts[cls] > 0:
            filtered_ground_truth_labels.append(cls)
            filtered_predicted_labels.append(label)
            class_min_counts[cls] -= 1

    # Updates batch lists with filtered ones
    batch_ground_truth_labels = filtered_ground_truth_labels
    batch_predicted_labels = filtered_predicted_labels

    # Computes confusion matrix for the current batch
    conf_matrices.append(confusion_matrix(batch_ground_truth_labels,
batch_predicted_labels, labels=range(9)))

# Merges confusion matrices to create the complete confusion matrix
complete_conf_matrix = sum(conf_matrices)

# Displays confusion matrix with numbers in each cell
plt.figure(figsize=(10, 8))
plt.imshow(complete_conf_matrix, interpolation='nearest',
cmap=plt.cm.Blues)

# Adds color bar
plt.colorbar()

# Adds numbers in each cell
thresh = complete_conf_matrix.max() / 2.
for i in range(complete_conf_matrix.shape[0]):
    for j in range(complete_conf_matrix.shape[1]):
        plt.text(j, i, format(complete_conf_matrix[i, j], 'd'),
                 horizontalalignment="center",
                 color="white" if complete_conf_matrix[i, j] > thresh
else "black")

# Sets axis labels and title
plt.xlabel('Predicted labels')
plt.ylabel('True labels')
plt.title('Complete Confusion Matrix')

# Sets x and y axis ticks and labels
plt.xticks(np.arange(len(class_labels)), class_labels.values(),
rotation=45)
plt.yticks(np.arange(len(class_labels)), class_labels.values(),
rotation=45)

plt.tight_layout()
plt.show()
```

Nicolae Nicolaou

Python code for custom object detection model accuracy metrics

```
import numpy as np
import matplotlib.pyplot as plt

# Custom class labels
class_labels = {
    0: "column", 1: "excavator", 2: "beam", 3: "masonry",
    4: "slab", 5: "window", 6: "person", 7: "Safety_Helmet", 8:
"Reflective_Jacket"}

def calculate_metrics(conf_matrix):
    num_classes = conf_matrix.shape[0]

    # Initializes arrays to store precision, recall, and F1 score for
each class
    precision = np.zeros(num_classes)
    recall = np.zeros(num_classes)
    f1_score = np.zeros(num_classes)

    for i in range(num_classes):
        # True positives: diagonal element
        tp = conf_matrix[i, i]
        # False positives: sum of column i (excluding tp)
        fp = np.sum(conf_matrix[:, i]) - tp
        # False negatives: sum of row i (excluding tp)
        fn = np.sum(conf_matrix[i, :]) - tp
        # True negatives: sum of all values except row i and column i
        tn = np.sum(conf_matrix) - tp - fp - fn

        # Calculates precision
        precision[i] = tp / (tp + fp) if (tp + fp) > 0 else 0
        # Calculates recall
        recall[i] = tp / (tp + fn) if (tp + fn) > 0 else 0
        # Calculates F1 score
        f1_score[i] = 2 * (precision[i] * recall[i]) / (precision[i] +
recall[i]) if (precision[i] + recall[i]) > 0 else 0

    # Calculates overall metrics
    overall_precision = np.mean(precision)
    overall_recall = np.mean(recall)
    overall_f1_score = np.mean(f1_score)
    return precision, recall, f1_score, overall_precision,
overall_recall, overall_f1_score

# Provided confusion matrix
conf_matrix = np.array([
    [122, 0, 10, 2, 28, 0, 1, 0, 0],
    [0, 65, 0, 0, 0, 0, 5, 1, 0],
    [7, 0, 48, 0, 1, 1, 0, 2, 0],
    [0, 0, 1, 67, 0, 1, 1, 0, 0],
    [2, 0, 0, 0, 35, 0, 1, 0, 0],
    [1, 0, 0, 1, 0, 108, 9, 0, 0],
    [0, 5, 0, 1, 1, 1, 77, 1, 0],
    [0, 1, 0, 0, 0, 0, 12, 30, 16],
    [0, 0, 0, 1, 0, 0, 11, 10, 47]])
```

Nicolae Nicolaou

```
# Calculates precision, recall, and F1 score
precision, recall, f1_score, overall_precision, overall_recall,
overall_f1_score = calculate_metrics(conf_matrix)

# Plots the results for each class
classes = list(class_labels.values())
x = np.arange(len(classes))
width = 0.2
space = 0.1
fig, ax = plt.subplots(figsize=(12, 6))

# Calculates the maximum value among precision, recall, and f1_score
arrays
max_value = max(max(precision), max(recall), max(f1_score))

# Sets the upper limit of the y-axis slightly higher than the maximum
value
ax.set_ylim(0, max_value + 0.1)

rects1 = ax.bar(x - width - space, precision, width, label='Precision',
color='cyan')
rects2 = ax.bar(x, recall, width, label='Recall',
color='mediumseagreen')
rects3 = ax.bar(x + width + space, f1_score, width, label='F1 Score',
color='#FFB6C1')  # Light pink color

# Adds labels, title, and legend
ax.set_ylabel('Scores', fontsize=14)
ax.set_title('Precision, Recall, and F1 Score by Class', fontsize=18)
ax.set_xticks(np.arange(len(classes)))
plt.yticks(fontsize=12)
ax.set_xticklabels(list(class_labels.values()), rotation=45,
ha='right', fontsize=14)
ax.legend(fontsize=13)

# Adds value annotations to each bar
def autolabel(rects):
    for rect in rects:
        height = rect.get_height()
        ax.annotate('{}'.format(round(height, 2)),
                    xy=(rect.get_x() + rect.get_width()/2, height),
                    xytext=(0, 3),
                    textcoords="offset points",
                    ha='center', va='bottom', fontsize=11)
autolabel(rects1)
autolabel(rects2)
autolabel(rects3)

# Shows the plot
plt.tight_layout()
plt.show()

# Prints overall metrics
print(f"\nOverall Precision = {overall_precision:.4f}, Overall Recall =
{overall_recall:.4f}, Overall F1 Score = {overall_f1_score:.4f}")
```

Nicolas Nicolaou

Python code for custom object detection (H&S VERSION)

```
import cv2
from imageai.Detection.Custom import CustomObjectDetection

def calculate_iou(box1, box2, image_width, image_height):
    x1, y1, w1, h1 = box1
    x2, y2, w2, h2 = box2

    # Converts YOLO coordinates to pixel coordinates
    x1, y1 = int(x1 * image_width), int(y1 * image_height)
    w1, h1 = int(w1 * image_width), int(h1 * image_height)
    x2, y2 = int(x2 * image_width), int(y2 * image_height)
    w2, h2 = int(w2 * image_width), int(h2 * image_height)

    # Calculates intersection rectangle coordinates
    x_start = max(x1, x2)
    y_start = max(y1, y2)
    x_end = min(x1 + w1, x2 + w2)
    y_end = min(y1 + h1, y2 + h2)

    # Calculates width and height of intersection rectangle
    intersection_width = max(0, x_end - x_start)
    intersection_height = max(0, y_end - y_start)

    # Calculates area of intersection rectangle
    intersection_area = intersection_width * intersection_height

    # Calculates areas of individual bounding boxes
    area_box1 = w1 * h1
    area_box2 = w2 * h2

    # Calculates area of union
    union_area = area_box1 + area_box2 - intersection_area

    # Calculates IoU
    iou = intersection_area / union_area if union_area > 0 else 0
    return iou

detector = CustomObjectDetection()
detector.setModelTypeAsYOLOv3()
detector.setModelPath("yolov3_CustomObjectDetection_mAP-0.67410_epoch-
20.pt")
detector.setJsonPath("CustomObjectDetection_yolov3_detection_config.jso
n")
detector.loadModel()

# Loads the image to get its dimensions
input_image_path = "hj93.jpg"
image = cv2.imread(input_image_path)
image_height, image_width, _ = image.shape

# Detection with adjusted rendering options
detections = detector.detectObjectsFromImage(
    input_image=input_image_path,
    output_image_path="hj93_output.jpg",
```

Nicolas Nicolaou

```
    minimum_percentage_probability=30,
    display_percentage_probability=False,  # Disable displaying
percentage probability
    display_object_name=True,  # Disable displaying object names
    display_box=True,
    nms_treshold=0.05  )

class_6_detected = False
class_7_detected = False
class_8_detected = False

# Count of objects for each class
class_6_count = 0
class_7_count = 0
class_8_count = 0

# Stores bounding boxes for each class
class_6_boxes = []
class_7_boxes = []
class_8_boxes = []

# Counters for checkpoints
warning_count_1 = 0
warning_count_2 = 0
warning_count_3 = 0

# Iterates through detections
for detection in detections:
    class_name = detection["name"]
    bbox = detection["box_points"]
    if class_name == "person":
        class_6_detected = True
        class_6_count += 1
        class_6_boxes.append(bbox)
    elif class_name == "Safety_Helmet":
        class_7_detected = True
        class_7_count += 1
        class_7_boxes.append(bbox)
    elif class_name == "Reflective_Jacket":
        class_8_detected = True
        class_8_count += 1
        class_8_boxes.append(bbox)

if not class_6_detected:
    print("No action required")
else:
    if not class_7_detected or not class_8_detected:
        warning_count_1 += class_6_count
    else:
        if class_7_count < class_6_count or class_8_count <
class_6_count:
            warning_count_2 += class_6_count -  min(class_7_count,
class_8_count)
        else:
            # Check for full overlapping
            full_overlap = True
            class_6_without_full_overlap = 0
```

Ioannis Nicolaou

```
            for class_6_box in class_6_boxes:
                overlap_7 = False
                overlap_8 = False
                for class_7_box in class_7_boxes:
                    if calculate_iou(class_6_box, class_7_box,
image_width, image_height) >= 0.9:
                        overlap_7 = True
                        break
                for class_8_box in class_8_boxes:
                    if calculate_iou(class_6_box, class_8_box,
image_width, image_height) >= 0.9:
                        overlap_8 = True
                        break
                if not overlap_7 or not overlap_8:
                    full_overlap = False
                    class_6_without_full_overlap += 1
            if not full_overlap:
                warning_count_3 += class_6_without_full_overlap
            else:
                print("No action required")

# ANSI escape code for red color
RED = '\033[91m'
# ANSI escape code for underlining text
UNDERLINE = '\033[4m'
# ANSI escape code for resetting underline
RESET_UNDERLINE = '\033[24m'

# Prints the checkpoint with the highest count
max_count = max(warning_count_1, warning_count_2, warning_count_3)

if max_count == 1:
    if max_count == warning_count_1:
        print(f"{RED}{UNDERLINE}HEALTH & SAFETY
ISSUE:{RESET_UNDERLINE}\n{warning_count_1} person without proposed
safety equipment\nin the construction site!!!")
    if max_count == warning_count_2:
        print(f"{RED}{UNDERLINE}HEALTH & SAFETY
ISSUE:{RESET_UNDERLINE}\n{warning_count_2} person without proposed
safety equipment\nin the construction site!!!")
    if max_count == warning_count_3:
        print(f"{RED}{UNDERLINE}HEALTH & SAFETY
ISSUE:{RESET_UNDERLINE}\n{warning_count_3} person without proposed
safety equipment\nin the construction site!!!")
else:
    if max_count == warning_count_1:
        print(f"{RED}{UNDERLINE}HEALTH & SAFETY
ISSUE:{RESET_UNDERLINE}\n{warning_count_1} persons without proposed
safety equipment\nin the construction site!!!")
    if max_count == warning_count_2:
        print(f"{RED}{UNDERLINE}HEALTH & SAFETY
ISSUE:{RESET_UNDERLINE}\n{warning_count_2} persons without proposed
safety equipment\nin the construction site!!!")
    if max_count == warning_count_3:
        print(f"{RED}{UNDERLINE}HEALTH & SAFETY
ISSUE:{RESET_UNDERLINE}\n{warning_count_3} persons without proposed
safety equipment\nin the construction site!!!")
```

Nicolas Nicolaou

Python code for number of instances & images per class counter

```
import os
import matplotlib.pyplot as plt
import numpy as np

# Custom class labels
class_labels = {
    0: "column", 1: "excavator", 2: "beam", 3: "masonry",
    4: "slab", 5: "window", 6: "person", 7: "Safety_Helmet", 8:
"Reflective_Jacket"
}

# Function to parse YOLO format truth labels and count instances for
each class
def count_instances(truth_labels_dir):
    class_counts = {label: 0 for label in range(len(class_labels))}
    class_files = {label: 0 for label in range(len(class_labels))}

    # Iterates through each truth label file
    for filename in os.listdir(truth_labels_dir):
        if filename.endswith(".txt"):
            with open(os.path.join(truth_labels_dir, filename), 'r') as
file:
                # Reads lines and counts instances for each class
                lines = file.readlines()
                found_classes = set()
                for line in lines:
                    class_id = int(line.split()[0])
                    if class_id in class_labels:
                        class_counts[class_id] += 1
                        found_classes.add(class_id)
                # Counts the files that contain at least one instance
of each class
                for class_id in found_classes:
                    class_files[class_id] += 1

    return class_counts, class_files

# Function to plot bar graph of instance distribution
def plot_instance_distribution(class_counts, class_files):
    # Sorts class counts and class files by class ID
    sorted_counts = [class_counts[label] for label in
sorted(class_labels)]
    sorted_files = [class_files[label] for label in
sorted(class_labels)]
    class_names = [class_labels[label] for label in
sorted(class_labels)]

    # Sets bar width
    bar_width = 0.35

    # Sets position of bars on X axis
    r1 = np.arange(len(class_names))
    r2 = [x + bar_width for x in r1]
```

Nicolae Nicolaou

```
    # Plots bars
    plt.figure(figsize=(10, 6))
    plt.bar(r1, sorted_counts, color='blue', width=bar_width,
edgecolor='grey', label='Number of Instances')
    plt.bar(r2, sorted_files, color='red', width=bar_width,
edgecolor='grey', label='Number of Images')

    # Adds labels and title with custom font size
    plt.xlabel('Class', fontweight='bold', fontsize=14)  # Adjust
fontsize as needed
    plt.ylabel('Count', fontweight='bold', fontsize=14)  # Adjust
fontsize as needed
    plt.xticks([r + bar_width / 2 for r in range(len(class_names))],
class_names, rotation=45,
               fontsize=12)  # Adjust fontsize as needed
    # Sets y-axis scale
    plt.yticks(fontsize=12)  # Adjust fontsize as needed
    plt.title('Instance and Image Distribution Across Classes',
fontweight='bold',
              fontsize=16)  # Adjust fontsize as needed

    # Adds values on each bar
    for i, count in enumerate(sorted_counts):
        plt.text(i, count + 0.1, str(count), ha='center',
va='bottom',fontsize=11)

    for i, files in enumerate(sorted_files):
        plt.text(i + bar_width, files + 0.1, str(files), ha='center',
va='bottom',fontsize=11)

    # Adds legend
    plt.legend(fontsize=12)
    plt.tight_layout()
    plt.show()

# Path to directory containing truth label files
truth_labels_dir = r'C:\Users\Nicolas\PycharmProjects\ImageAI-
master\examples\CustomObjectDetection - Results\totals'

# Counts instances and files for each class
class_counts, class_files = count_instances(truth_labels_dir)

# Plots instance and file distribution
plot_instance_distribution(class_counts, class_files)
```

Nicolas Nicolaou

Python code for number of instances per class counter (3 folders comparison version)

```
import os
import matplotlib.pyplot as plt
import numpy as np

# Custom class labels
class_labels = {
    0: "column", 1: "excavator", 2: "beam", 3: "masonry",
    4: "slab", 5: "window", 6: "person", 7: "Safety_Helmet", 8:
"Reflective_Jacket"
}

# Function to parse YOLO format truth labels and count instances for
each class
def count_instances(truth_labels_dir):
    class_counts = {label: 0 for label in range(len(class_labels))}
    class_files = {label: 0 for label in range(len(class_labels))}

    # Iterates through each truth label file
    for filename in os.listdir(truth_labels_dir):
        if filename.endswith(".txt"):
            with open(os.path.join(truth_labels_dir, filename), 'r') as
file:
                # Reads lines and counts instances for each class
                lines = file.readlines()
                found_classes = set()
                for line in lines:
                    class_id = int(line.split()[0])
                    if class_id in class_labels:
                        class_counts[class_id] += 1
                        found_classes.add(class_id)
                # Counts the files that contain at least one instance
of each class
                for class_id in found_classes:
                    class_files[class_id] += 1

    return class_counts, class_files

# Function to plot bar graph of instance distribution
def plot_instance_distribution(class_counts, class_files):
    # Sorts class counts and class files by class ID
    sorted_counts = [class_counts[label] for label in
sorted(class_labels)]
    sorted_files = [class_files[label] for label in
sorted(class_labels)]
    class_names = [class_labels[label] for label in
sorted(class_labels)]

    # Sets bar width
    bar_width = 0.35

    # Sets position of bars on X axis
    r1 = np.arange(len(class_names))
    r2 = [x + bar_width for x in r1]
```

Nicolas Nicolaou

```
    # Plots bars
    plt.figure(figsize=(10, 6))
    plt.bar(r1, sorted_counts, color='blue', width=bar_width,
edgecolor='grey', label='Number of Instances')
    plt.bar(r2, sorted_files, color='red', width=bar_width,
edgecolor='grey', label='Number of Images')

    # Adds labels and title with custom font size
    plt.xlabel('Class', fontweight='bold', fontsize=14)  # Adjust
fontsize as needed
    plt.ylabel('Count', fontweight='bold', fontsize=14)  # Adjust
fontsize as needed
    plt.xticks([r + bar_width / 2 for r in range(len(class_names))],
class_names, rotation=45,
               fontsize=12)  # Adjust fontsize as needed
    # Sets y-axis scale
    plt.yticks(fontsize=12)  # Adjust fontsize as needed
    plt.title('Instance and Image Distribution Across Classes',
fontweight='bold',
              fontsize=16)  # Adjust fontsize as needed

    # Adds values on each bar
    for i, count in enumerate(sorted_counts):
        plt.text(i, count + 0.1, str(count), ha='center',
va='bottom',fontsize=11)

    for i, files in enumerate(sorted_files):
        plt.text(i + bar_width, files + 0.1, str(files), ha='center',
va='bottom',fontsize=11)

    # Adds legend
    plt.legend(fontsize=12)
    plt.tight_layout()
    plt.show()

# Path to directory containing truth label files
truth_labels_dir = r'C:\Users\Nicolas\PycharmProjects\ImageAI-
master\examples\CustomObjectDetection - Results\totals'

# Counts instances and files for each class
class_counts, class_files = count_instances(truth_labels_dir)

# Plots instance and file distribution
plot_instance_distribution(class_counts, class_files)
```

Nicolas Nicolaou