A computer vision technique of detecting Hardhat for the safety of construction workers

Prepared By:

Mohammad Giasuddin

Student Id: 20052010

Programme: MSc Data Science

“This research was completed for the MSc in Data Science at the University of the West of England, Bristol. The work is my own. Where the work of others is used or drawn on it is attributed.”

Name & Signature

Mohammad Giasuddin

Word Count: 12,673

I. Abstract

Working in construction production is supposed to be one of the dangerous professions with the highest safety risk factor due to the complex interaction of workers with equipment. Therefore, most worker's safety regulations concern the correct use of personal protective equipment (PPE), especially hardhat to protect the workers from fatal injuries or most importantly brain damage. Therefore, computer vision techniques on automatic hardhat-wearing detection can reduce on-site fatal injuries.

To enhance the worker's safety, most of the recent studies used public datasets. However, these methods still have limitations in the context of the diverse representation of desired objects into the dataset that can improve the model performance detecting the variety of the objects available on-site image captures. This research presents how to prepare a balanced dataset by using the combination of image augmentation features and on-site learning examples that can increase the positive and negative features of the desired objects.

In this study, model YOLOv5 was used to train on the enhanced custom dataset. The improved resultant model achieved 0.890 mAP according to the training logs and achieved a 97% success rate in manual testing on the test dataset captured from the construction site, which outperforms several state-of-the-art methods on a public dataset.

II. Acknowledgments

I was working for the last 10 years in the software industry. I had the opportunity to work on the big dataset of several large companies in my country. I know the importance of the data how it can help the company from the insights of the big dataset. That's why I enrolled in MSc Data Science intending to be a future data scientist.

A year ago, I knew nothing about data science programming than I thought I knew about technology. I decided to fund this MSc for two reasons. Firstly, machine learning has always been my passion, but I never had a way of understanding how it worked. Secondly, I wanted to get into an exciting industry; hence the choice of a computer vision-based project. A year later, I can program with Python, R, and NoSQL, I understand much more about machine learning, deep learning, and computer vision have become a real passion of mine.

I would like to thank my lecturers, specifically Dr Thomas Win, Dr David Wyatt, and Dr Prakash Chatterjee. Dr David Wyatt and Dr Prakash Chatterjee taught me a lot about python programming, big dataset, SQL and NoSQL databases and inspired me to be passionate about python programming. Dr Thomas has been a constant guide for this research, not only helping me structure its execution, but encouraging me to go above and beyond the MSc limitations. Given my lack of deep learning experience, I did think twice about whether I could complete this research project. Dr Thomas was a reassuring and encouraging guide throughout. Beyond my lecturers, I would also thank a fellow student, Sadman Sakib, for the benefit of his experience in Machine Learning.

Finally, I must thank my wife. Without her support, I would not have achieved a fraction of what I have achieved. This dissertation would not exist without her.

III. Table of Contents

IV. List of Tables

V. List of Figures

1. Chapter One: Introduction

Construction sites are considered one of the most dangerous industries due to the complex interaction of workers with equipment, building materials, vehicles, etc. Hardhat is a crucial protective gear for the safety of workers on construction sites. Therefore, administrators need to identify the people who do not wear hardhats.

Construction plays an important role in the UK's economy –working 3.1 million people or over 9% of the workforce(GOV.UK, 2018). However, it is very important to the country that construction sector workers have the most dangerous occupations in the world. According to the data from the HSE Statistics, 142 workers lost their lives at workplace fatal injuries in Great Britain (HSE, 2020). Therefore, most workers safety regulations concern the correct use of personal protective equipment (PPE), especially hardhat. Most of the hardhat supervision still mainly relies on labor. This supervision is really difficult due to the complex site and large personnel. Therefore, computer vision techniques on the automatic hardhat-wearing detection can keep a huge contribution to improve the safety of the construction workers.

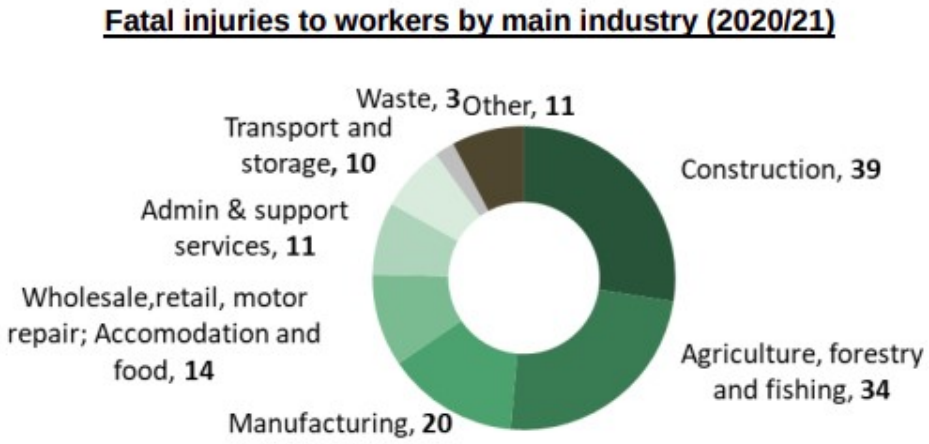


Figure-1.1: Workplace fatal injuries in Great Britain, 2021(HSE, 2020)

In 2008-2017, the fatal incidents caused 376 people to lose their lives (IOGP, no date). IOGP published a set of Life-Saving Rules to the industrial workers with the actions they can follow to save themselves from fatal injuries. According to the life saving rules, hardhat is one of the protective gears for the workers to wear in the industrial sites.

In the U.S. economy, construction is also considered one of the biggest sectors, with $1.598 trillion annual expenditure (US Census Bureau Construction Expenditures, 2021) and 7.53 million employees (~5% of the total workforce)(U.S. Bureau of Labor Statistics, 2021). The data from the US Bureau of Labor Statistics shows that the fatal injury rate was higher in the construction sector rather than other industries (U.S. Bureau of Labor Statistics, 2020).

Therefore, the automatic supervision of wearing personal protective equipment (PPE), especially hardhat, can reduce the fatal injuries in the construction sites.

1.1 Research Aim and Objectives

1.1.1 Research Aim

To prepare a balanced training dataset using image augmentation features and active learning from the on-site captured images can enhance the model performance for real-time detection of the construction worker hardhats.

1.1.2 Research Questions

1. How to prepare a well balanced training dataset for hardhat detection?

2. How image augmentation features can enhance the dataset as well as the model performance to detect the diverse instances of the hardhats?

3. How on-site learning examples(Active Learning) can enhance the model performance?

4. How to reduce the False Positive (FP) detection on-site?

1.1.3 Scope

This research will focus on preparing a balanced training dataset that can improve the model performance in detecting the diverse hardhats for the safety of construction workers.

1.1.4 Research Objectives

1. Research on improving training dataset for real-time object detection.

2. Research and analyze image augmentation features to enrich the object presentation in the dataset.

3. Develop the data collection pipeline for adding on-site learning examples and background images in the further training dataset.

4. Annotate and validate the data quality.

5. Train the model YOLOv5 on the improved dataset and test the model with the test dataset to measure the model performance.

6. Develop API end-point to deploy the improved model.

7. Develop a user application for reporting the hardhat detection results.

1.2 Dissertation Structure

The dissertation starts with a broad literature review to outline the object detection and the comparative analysis of popular object detection algorithms. A series of recent researches on detecting hardhats are also outlined with the strengths and limitations.

Then, a proposed approach is outlined in detail including data collection pipeline, training pipeline, testing model to measure performance, model deployment, and user application development.

Finally, the experiment results analysis, discussion, and conclusion are provided against the research questions.

1.3 Conclusion

This chapter has outlined the research background, motivation, questions, goals, the structure for delivery. Ultimately, the end goal of this research is to prepare a balanced training dataset for improving the model detection performance.

2. Chapter Two: Literature Review

2.1 Object Detection

Object detection is that the identification of an object within the image together with its localization and classification. it's widespread applications and could be a critical component for vision-based software systems. Object detection detects visual objects of various classes (for instance, human, animal, cars, or building) in digital images like photos or video frames and It's an enormous accomplishment of deep learning and image processing. Creating localization with the assistance of bounding box could be a regular process of object detection. The goal of objection detection is to work out where objects are located in a very given image called object localization and therefore the category of those objects (Boesch, 2021)(Bharath K, 2021).

2.2 How object detection works

The first object detector was the Viola Jones Object Detector, and it came to enter 2001 (Enriquez, 2018). Although it absolutely was technically classified as an object detector, it's primary use case was for facial detection. It provided a true time solution and was adopted by many computer vision libraries at the time. The sector was substantially accelerated with the appearance of Deep Learning. The primary Deep Learning object detector model was called the Overfeat Network which used Convolutional Neural Networks (CNNs) together with a window approach. It classified each a part of the image as an object/non object and subsequently combined the results to come up with the ultimate set of predictions. This method of using CNNs to unravel detection led to new networks being introduced which pushed the state of the art even further. Object detection is worked using following methods (Boesch, 2021):

(1) Image processing techniques.

(2) Deep learning networks.

1. Image processing techniques – It is a method to perform operations on an image to extract information from it or enhance it. Image processing has a wide range of applications such as image restoration, medical imaging, remote sensing, image segmentation, etc. Every process requires a different technique. Figure-2.1 shows how image processing works for image segmentation to locate the desired objects or boundary information from an image.

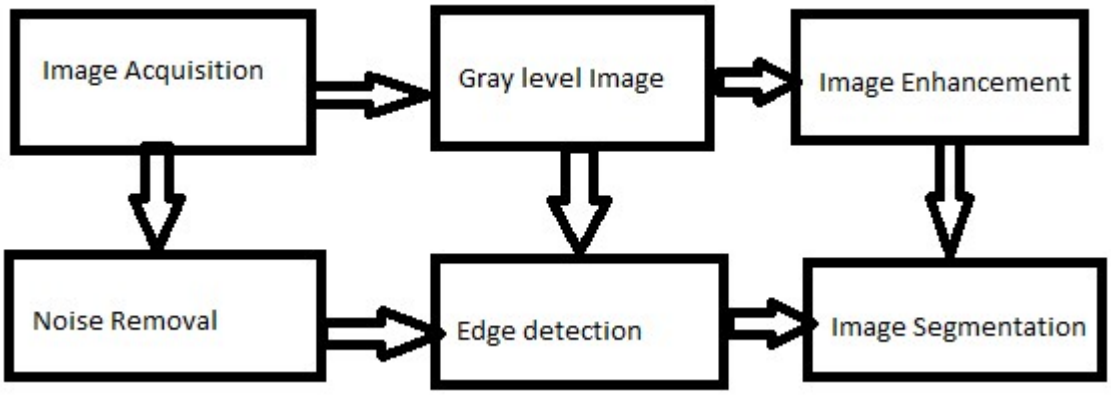


Figure-2.1 Image processing techniques (Nagalakshmi and Jyothi, 2015).

2. Deep learning networks – It extracts the features of images like corners and edges so as to form models of the assorted objects. It later uses these models to spot the objects and it's a subtype of machine learning. Machine learning doesn't require high-performance processors and more data. Deep learning is a machine learning technique that learns features and tasks directly from data. The data will be images, text files, or sound. According to Figure-2.2, the summation of each neuron inputs applies to an activation function to generate the output in the artificial neural network. It's inspired by the architecture of the human brain's signal processing with neurons.

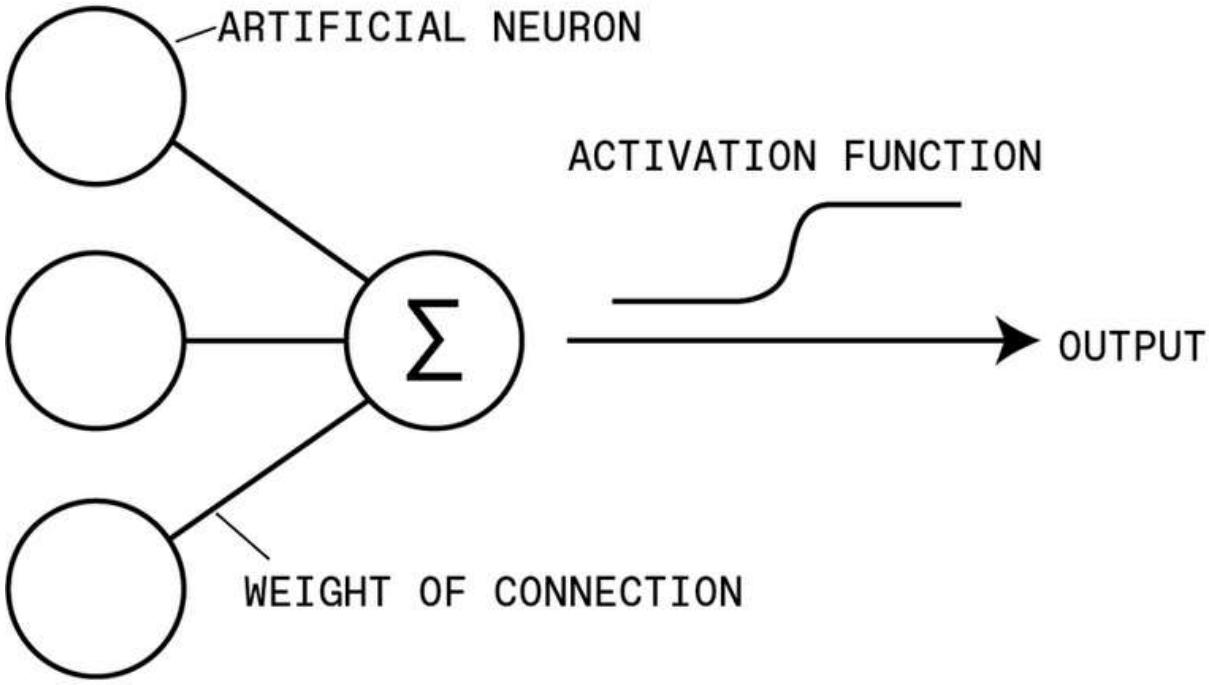


Figure-2.2 Deep learning network process (Moore, Schneider and Strickland, 2021).

Method	Pros	Cons
Image processing techniques	1. It doesn't require historical data for training and are unsupervised in nature. 2. This tasks do not require annotated images, where humans labelled data manually (for supervised training).	These techniques are restricted to multiple factors, like complex scenarios (without unicolor background), occlusion (partially hidden objects), illumination and shadows, and clutter effect.
Deep learning networks	1. All features are automatically and optimally tuned for desired outcome and features aren't required to be extracted earlier than time. This avoids time-consuming machine learning techniques. 2. Preferably, the neural network-based approach may be applied to several different applications and data types.	1. It requires very great amount of data so as to perform better than other techniques. 2. Deep learning networks increase the cost to the users due to expensiveness to coach thanks to complex data models and it requires expensive GPUs and many machines.

Table-2.1 Comparison of object detection methods(Boesch, 2021).

Now a days, Researchers and computer vision companies widely accepted deep learning object detection to build commercial products. Figure-2.3 shows an example of real-time object detection project.

Figure-2.3 Object detection for vehicles (cars, trucks, bikes, etc.) by deep learning. An example frame of a commercial real time application with AI recognition on the stream of IP cameras (Boesch, 2021).

2.3 Object detection algorithms

Object detection has been witnessing a rapid revolutionary change within the field of computer vision. The involvement of object classification with object localization makes it one of the foremost demanding topics within the domain area of computer vision. Following are the favored Algorithms of object detection.

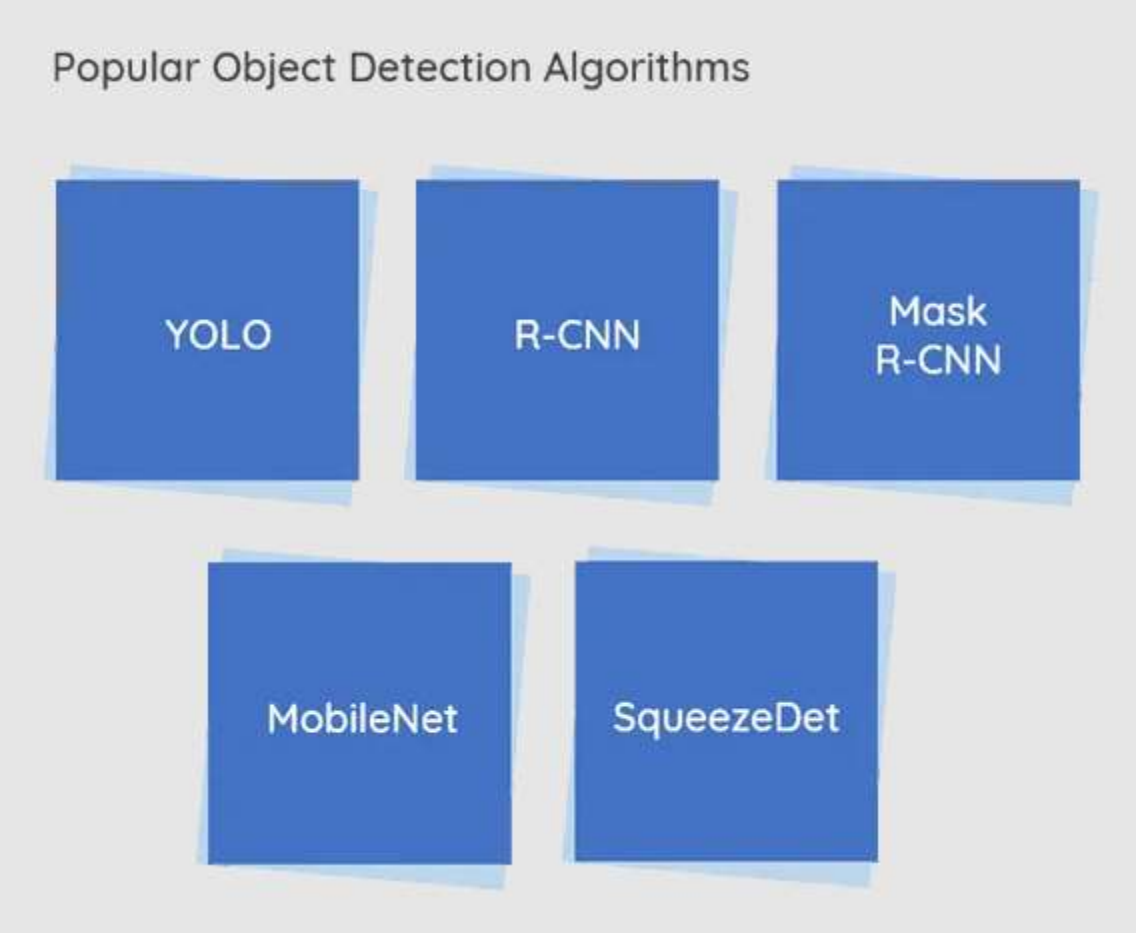


Figure-2.4 Object detection overview of popular algorithms (Boesch, 2021).

2.3.1 YOLO – You Only Look Once

YOLO is one in every of the popular algorithms in object detection utilized by researchers round the globe. YOLO could be a real-time object detection algorithm; it uses one neural network. In step with the researchers at Facebook AI Research, the uniform architecture of YOLO is excessively fast in manner. The bottom YOLO model handles images in real-time around 45 frames per second. The smaller version of the network, Fast YOLO can process an astonishing 155 frames per second while still achieving double the mAP of other real-time detectors (Redmon et al., 2016). This algorithm outperforms the opposite detection methods, including DPM and R-CNN, when generalising from natural images to other domains like artwork.

The latest release of ImageAI v2.1.0 now allows training a custom YOLO model to detect any type and number of objects. Convolutional neural networks are instances of classifier-based systems where the system repurposes classifiers or localizers to perform detection and applies the detection model to an image at multiple locations and scales. Using this process, “high

scoring" regions of the image are considered detections. Using this process, "high scoring" regions of the image are considered detections. Simply put, the regions which look most like the training images are given are identified positively.

As a single-stage detector (SSD), YOLO executes classification and bounding box regression in one step. It makes YOLO faster than most of the convolutional neural networks. For example, YOLO object detection system processes images 1000x faster than R-CNN and 100x faster than Fast R-CNN. The version of YOLOv3 earns 57.9% mAP on the MS COCO dataset compared to DSSD513 of 53.3% and RetinaNet of 61.1%. YOLOv3 utilizes multi-label classification with overlapping patterns for training. Therefore, it can be used in complex use cases for object detection. YOLOv3 can be used for small object classification due to its multi-class prediction capabilities. On the other hand, it shows worse performance for detecting large or medium-sized objects (Boesch, 2021).

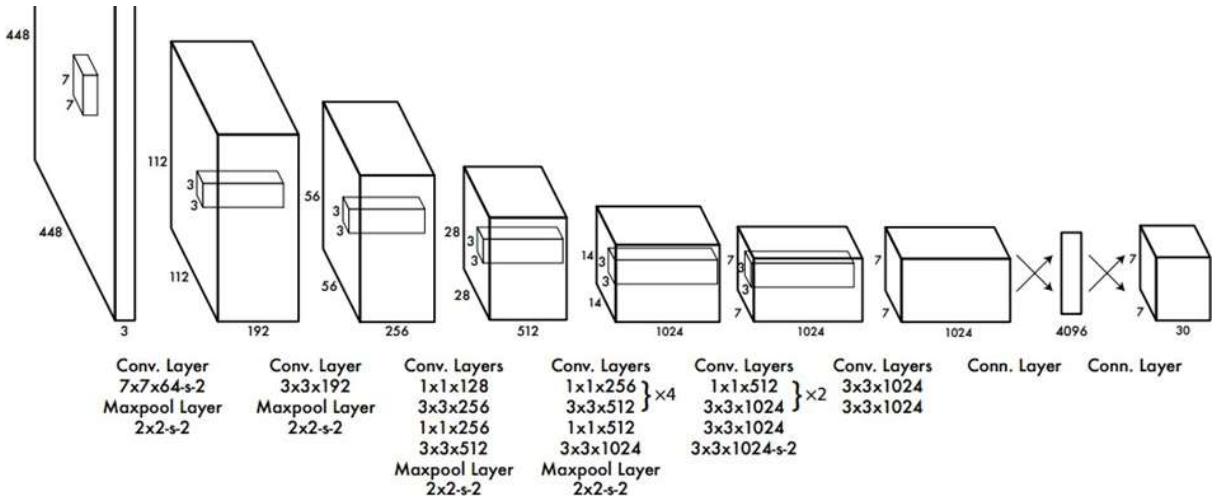


Figure-2.5 You Only Look Once (Redmon *et al.*, 2016).

Figure-2.5 shows the basic YOLO architecture. The architecture consists of 24 convolution layers. Then it's followed by two fully connected layers. Alternating 1 × 1 convolutional layers decrease the features space from the foregoing layers. In the ImageNet classification task, YOLO pre-train the convolutional layers at half the image resolution and then double the resolution for detection (Redmon *et al.*, 2016).

The YOLOv5 is an enhanced version of YOLOv4. The principal additions are mosaic data improvement, self-adversarial training, and cross mini-batch normalization.

2.3.2 SSD – Single-shot detector

SSD could be a method for detecting objects in images employing a single deep neural network. It's a preferred one-stage detector which will predict multiple classes. The SSD approach discretises the output space of bounding boxes into a group of default boxes over different aspect ratios. After discretising, the tactic scales per feature map location. The SSD network combines predictions from multiple feature maps with different resolutions to naturally handle objects of varied sizes. (Choudhury, 2020).

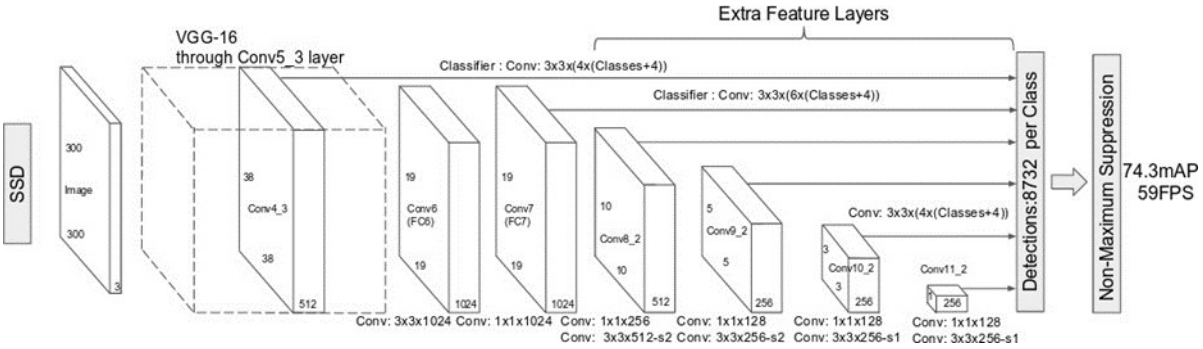


Figure-2.6 Single Shot Detector (SSD)(Khandelwal, 2019)

As can be seen in Figure-2.6, SSD architecture has a base VGG-16 network followed by multi-box convolution layers. The base neural network works for features extraction. Then additional convolution layers perform the object detection.

The SSD creates the confidence scores for the existence of each object class in each default box and modifies the box to better fit the object shape. The network also merges the predictions from the numerous feature maps with distinct resolutions to handle objects of various sizes. This approach is easy to train and incorporate into software systems that need an object detection feature. Compared with other single-stage methods, SSD achieves much better accuracy not only in large input image but also in smaller input image (Boesch, 2021).

2.3.3 R-CNN – Region-based Convolutional Neural Networks

The region-based Convolutional Network method (RCNN) contains region proposals with Convolution Neural Networks (CNNs). R-CNN assists to find out the location of the objects with a deep network and training a high-capable model with only a small amount of annotated

detection data. First, the R-CNN approach selects the proposed regions from an image (for example, anchor boxes are one type of selection method) and then annotates their categories and bounding box information (e.g., offsets). R-CNN has excellent object detection accuracy because of uses a deep convolution network to categorize object proposals. R-CNN can rise to thousands of object classes without resorting to approximate techniques, including hashing (Choudhury, 2020).

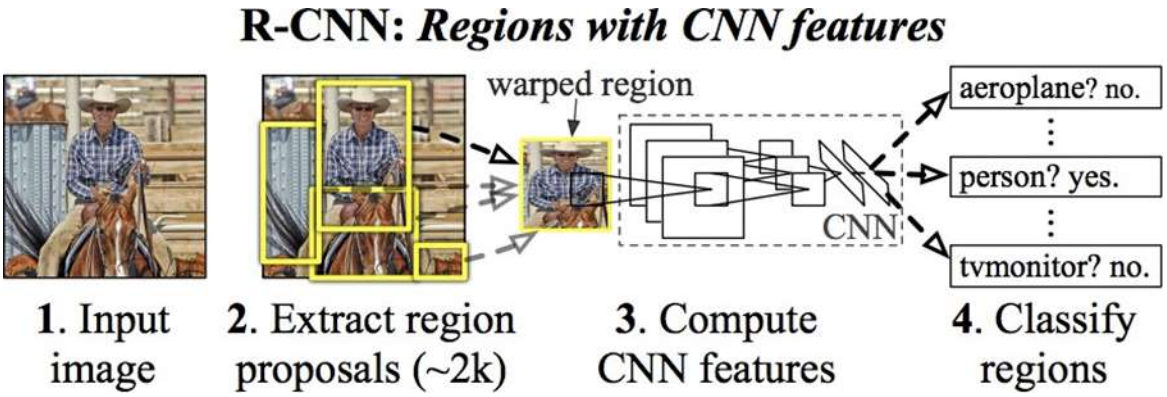


Figure-2.7 Region based convolutional neural networks (RCNN)(Horak and Sablatnig, 2019)

In R-CNN, the input image is first split into almost two thousand region proposals, and then it applies a convolutional neural network for each region, respectively. Then, The size of the regions is calculated, and the correct region is added to the neural network. The whole process can produce time constraints. Model training time is much higher compared to YOLO because it categorizes first and calculates bounding boxes individually, and a neural network is applied to one region at a time. Fast R-CNN was developed in 2015 to cut down significantly on train time. The original R-CNN alone computed the neural network features on each of as many as 2 thousand regions of interest. Fast R-CNN executes the neural network once on a full image. It is almost comparable to YOLO's architecture. However, YOLO is still a faster option than Fast R-CNN because of the simplicity of the code. The network could be a novel method called Region of Interest (ROI) Pooling, which slices out each Region of Interest from the network's output tensor, reshapes, and classifies it. It makes Fast R-CNN more accurate than the original R-CNN. However, because of this recognition technique, fewer data inputs are required to train Fast R-CNN and R-CNN detectors (Boesch, 2021).

2.3.4 Mask R-CNN

It is written in Python and C++ (Caffe), and Mask R-CNN is an advancement of Fast R-CNN. Fast Region-based Convolutional Network method or Fast R-CNN may be a training algorithm for object detection. This algorithm mainly corrects the drawbacks of R-CNN and SPPnet, while

improving on their speed and accuracy (Choudhury, 2020). The dissimilarity between both is that Mask R-CNN added a branch for predicting an object mask in parallel with the prevailing branch for bounding box recognition (Boesch, 2021). Mask R-CNN is straightforward to coach and adds only a little overhead to Faster R-CNN; it can run at 5 fps.

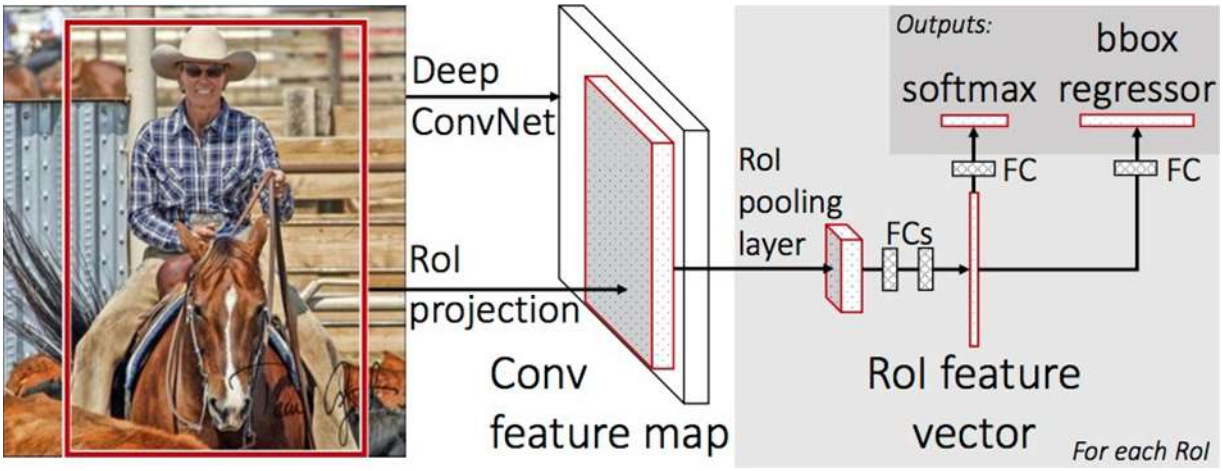


Figure-2.8 Mask RCNN (Girshick, 2015)

As can be seen in Figure-2.8, the input image is fed into CNN to output the feature map. Then the regions of proposals are placed and converted into squares. Next, these regions are given to the RoI Pooling layer, where each region is pooled into a specific-sized feature map. It's mapped to a feature vector by fully connected layers. Each RoI is separated into 2 vectors. One is SoftMax probabilities and another is regression offsets.

2.3.5 MobileNet

As the following Figure-2.9, MobileNet architecture uses depthwise separable convolutions to construct lightweight deep convolutional neural networks and provides an efficient model for mobile and embedded vision applications. It works with single-shot multi-box detection network wont to run object detection tasks and this model is implemented using the Caffe framework and also the output may be a typical vector containing the tracked object data (Boesch, 2021).

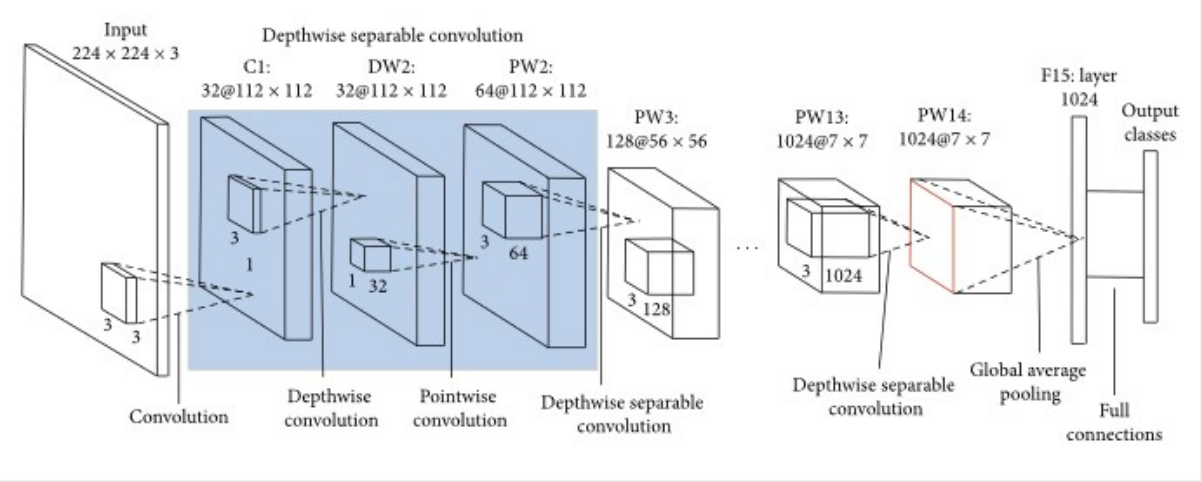


Figure-2.9 MobileNet (Pujara, 2020)

2.3.6 SqueezeDet

SqueezeDet is one of the deep neural networks for computer vision that was released in 2016. It was specifically designed for autonomous driving. It works for object detection using computer vision techniques. This is concentrated on the reduction of network parameters. For this objective, it utilizes plenty of 1x1 filters to massively bring down the number of weights. Like YOLO, it's a single-shot detector algorithm. In SqueezeDet, convolutional layers are employed only to extract feature maps but also because the output layer to calculate bounding boxes and sophistication probabilities. The detection pipeline of SqueezeDet models only includes single forward passes of neural networks, allowing them to be excessively fast. The following Figure-2.10 shows the basic architecture of SqueezeDet. (Boesch, 2021).

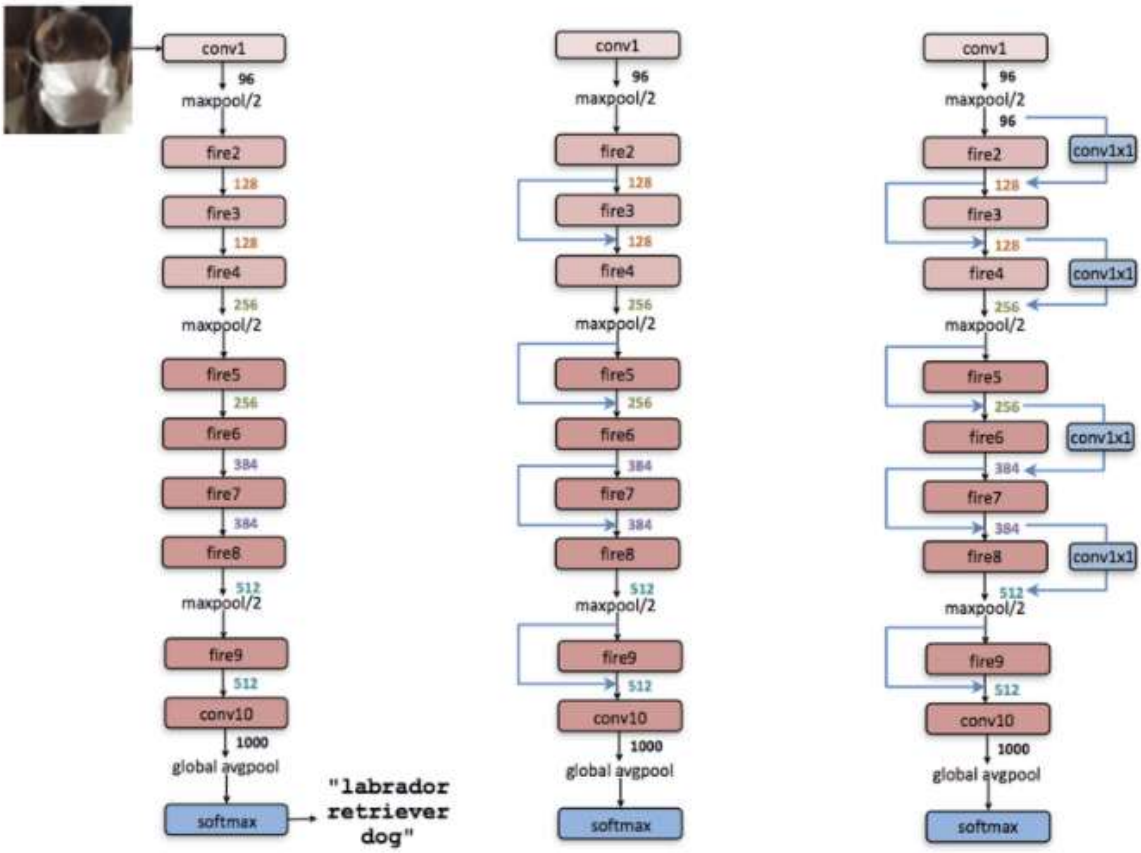


Figure-2.10 SqueezeDet (Villanueva *et al.*, 2019)

2.3.7 Retina Net

The RetinaNet could be a single step object detector that boasts the state of the art results at this time in time by introducing a novel loss function. This model depicts the primary instance where one step detectors have surpassed two step detectors in accuracy while retaining superior speed. The authors realized that the rationale why one step detectors have lagged behind 2 step detectors in accuracy was an implicit class imbalance problem that was encountered while training. The RetinaNet sought to unravel this problem by introducing a loss function coined Focal Loss (Chahal and Dey, 2018).

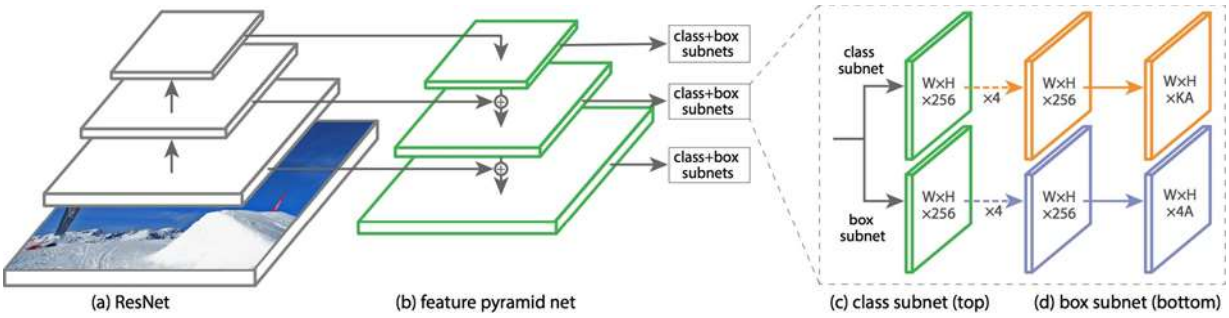


Figure-2.11 Retina Net (Chahal and Dey, 2018)

Figure-2.11 shows the architecture of Retina Net. It could be a single unified network composed of a backbone network and it's two task-specific subnetworks. One subnetwork predicts the category of the item area and another subnetwork predicts the boundary box offsets.

2.3.8 Histogram of Oriented Gradients (HOG)

Histogram of oriented gradients (HOG) is largely a feature descriptor that's utilized to detect objects in image processing and other computer vision techniques. The Histogram of oriented gradients descriptor technique contains the occurrences of gradient orientation in localized portions of a picture, like the detection window, the region of interest (ROI), among others. One advantage of HOG-like features is their simplicity, and it's easier to know the knowledge they carry (Choudhury, 2020).

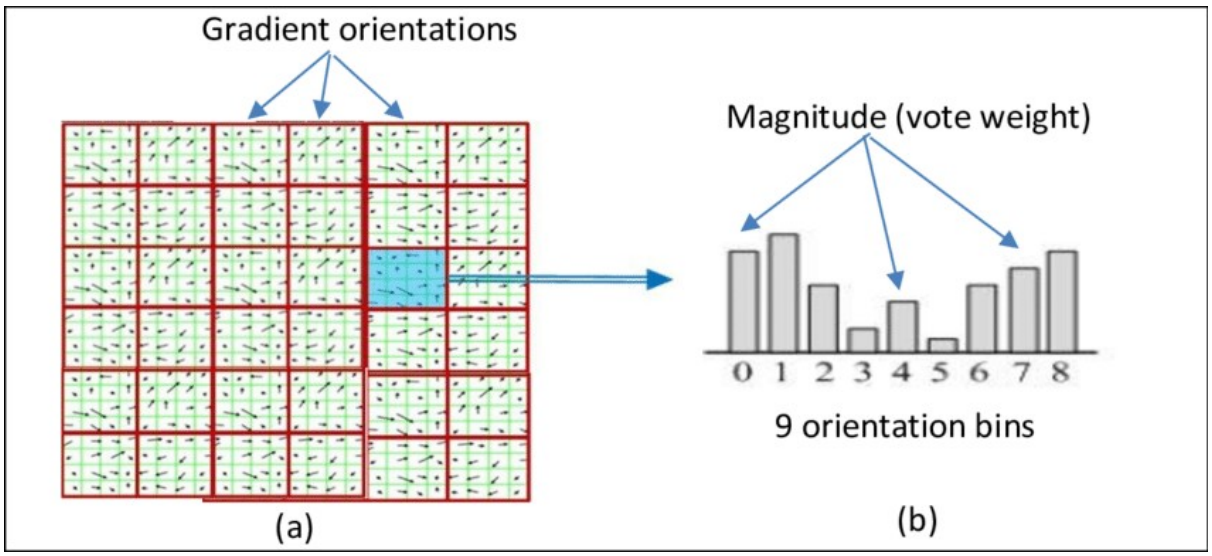


Figure-2.12 Histogram of Oriented Gradients (Kadhm, 2017).

As can be seen in Figure-2.12, the HOG descriptor uses magnitude in addition to the angle of the gradient to compute the features. For the regions of the image, it generates histograms using the magnitude and orientations of the gradient.

2.3.9 YOLOR

In 2021, the novel object detection YOLOR was introduced. The algorithm uses implicit and explicit knowledge to the model training at an identical time. Hence, YOLOR can learn a general representation and complete multiple tasks through this general representation. Through kernel space alignment, Implicit knowledge is integrated into explicit knowledge. Prediction refinement, and multi-task learning. Via this method, YOLOR gains vastly enhanced object detection performance results. Comparison between other object detection methods on the COCO dataset benchmark, the MAP of YOLOR is 3.8% beyond the PP-YOLOv2 at the identical inference speed. Compared with the Scaled-YOLOv4, the inference speed has been improved by 88%, making it the fastest real-time object detector available today. Read more in our dedicated article about YOLOR – you merely Learn One Representation (Boesch, 2021).

2.4 Comparing object detection algorithms

For the result given below, the models were trained on both PASCAL VOC 2007 and 2012 data. The Mean Average Precision (mAP) was estimated on the PASCAL VOC 2012 testing dataset. For SSD, the chart exhibits results for 300 × 300 and 512 × 512 input images. For YOLO, it's results for 288 × 288, 416 ×461 and 544 × 544 images. Higher resolution images for the identical model have better mAP but are slower to process (Hui, 2019).

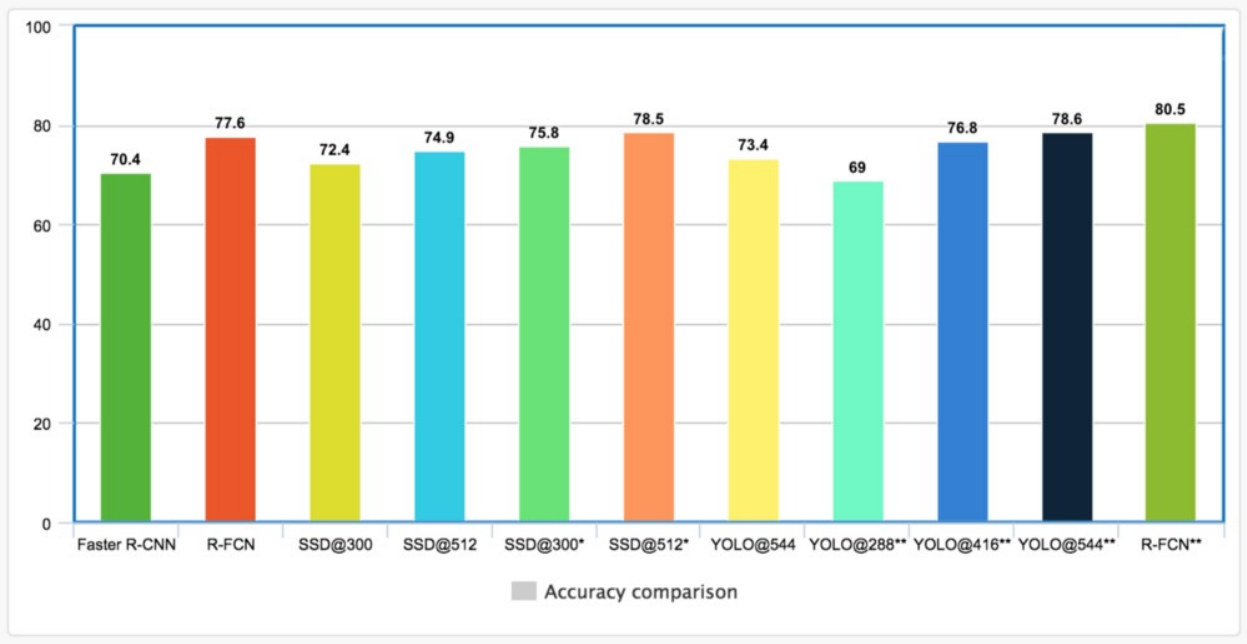


Figure-2.13 Object detection algorithms accuracy comparison (Hui, 2019).

* indicates that small object data augmentation was applied.

** means the results are measured on VOC 2007 testing dataset.

Input image resolutions and have extractors impact speed. Below is that the highest and lowest FPS reported by the corresponding papers. Yet, the result below may be highly biased particularly they're measured at different mAP (Hui, 2019).

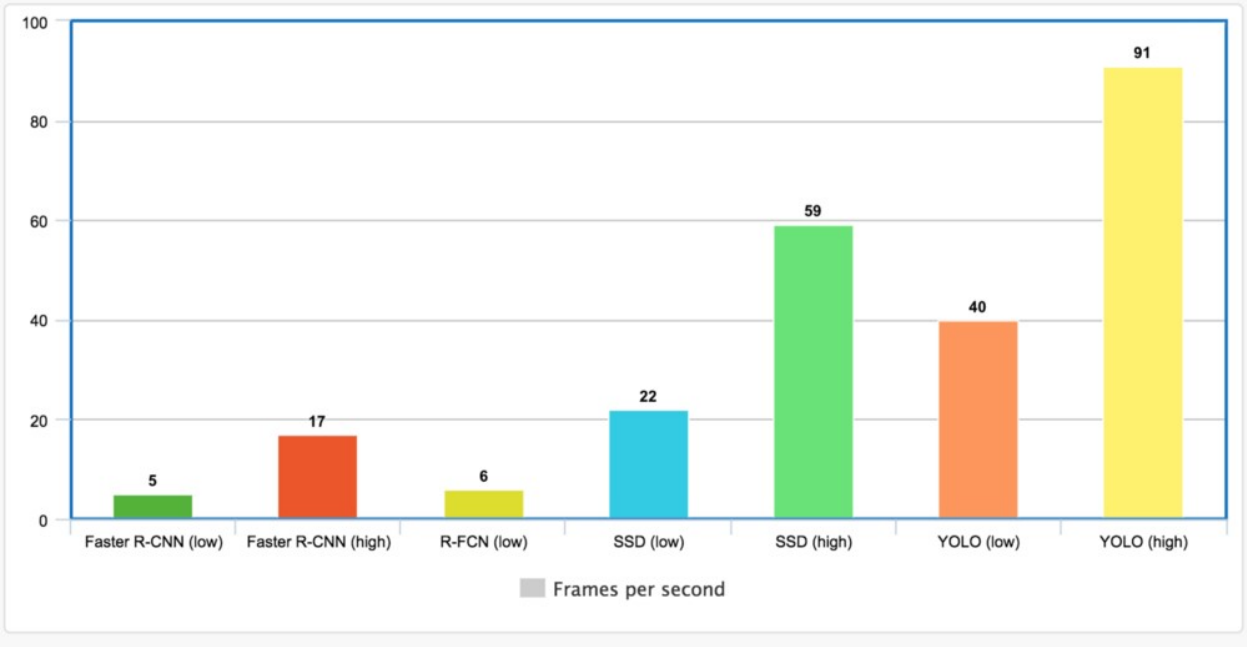


Figure-2.14 Object detection algorithms speed comparison (Hui, 2019).

The above comparative revision of the well-known object detection algorithms helped to consider YOLO as the base algorithm for this study in terms of accuracy and speed. YOLOv5 training pipeline was installed in the AWS platform for frequent training on the continuously improved datasets.

2.5 Related works

The literature review to frame this study focused on the balanced dataset preparation for detecting real-time diverse instances of the desired objects, especially head and hardhat for the safety of construction workers. This research also focuses on the object representation in the training dataset by using the image augmentation features(such as rotation, exposure, contrast, brightness, crop, etc.), learning examples from on-site(Active learning). In this study, the latest YOLO pre-trained model is considered to train the custom dataset for detecting workers hardhat. The relevant papers reviewed in these areas are summarized as follows.

Bhadeshiya, Brahmbhatt and Pitroda(2021) used the object detection algorithm You Only Look Once (YOLOv4) for hard-hat detection and safety concerns at construction sites. The training dataset, which is made up of classes of hard-hat, head, and person captured in colored and monochrome images. The dataset is made with varying light conditions. According to this research, the yolov4 model can detect the hardhat with an approximate accuracy of 99% for images and 98% for videos. Figure-2.15 shows how object detection works in this approach.

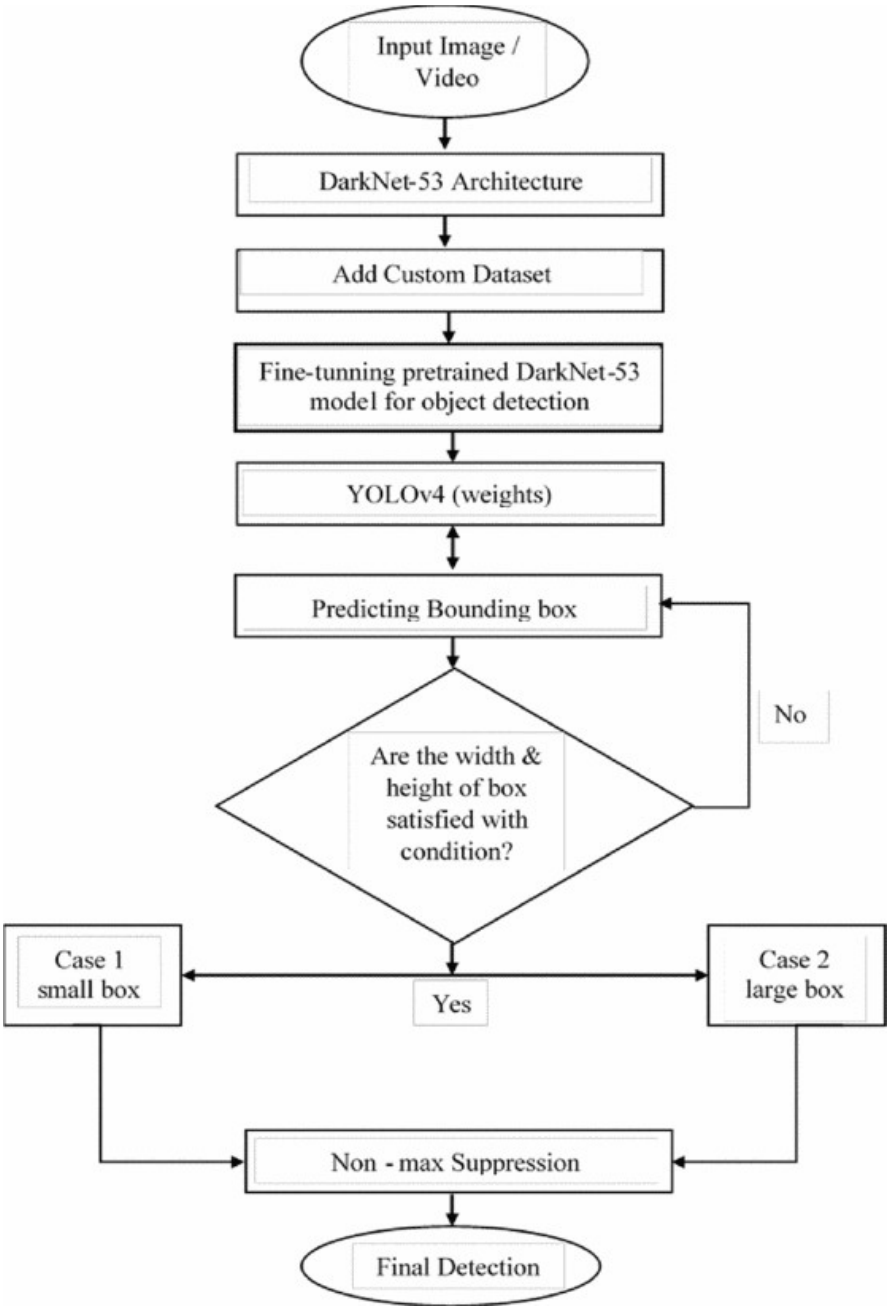


Figure-2.15 Flow-chart for multiple object detection (Bhadeshiya, Brahmbhatt and Pitroda, 2021).

According to Figure-2.15, YOLOv4 was straightway trained on a public dataset in this study. However, achieving 98% to 99% precision in real-time object detection for multiple objects in a single image is challenging. No evidence or guidance was found on how to improve the diverse object features that can enhance the performance of the model for detecting the variety of object instances from on-site captures. Even no discussion was found on how to continuously enhance the training dataset as well as model performance.

Casuat et al.(2020) developed a hard hat detection system by using YOLOv3 to determine if the worker is wearing a hardhat properly. Image processing was used in this study. The authors used the public datasets with hard hat-wearing images to measure the performance by using the mean average precision (mAp) where they obtained an average accuracy of 79.246. The acquired output shows that the algorithm successfully found the hard-hat with an average Training and Validation accuracy of 97.29 and 92.55, average evaluation accuracy of 79.24% with the highest model accuracy of 86.89%, and testing accuracy of 86.67%.

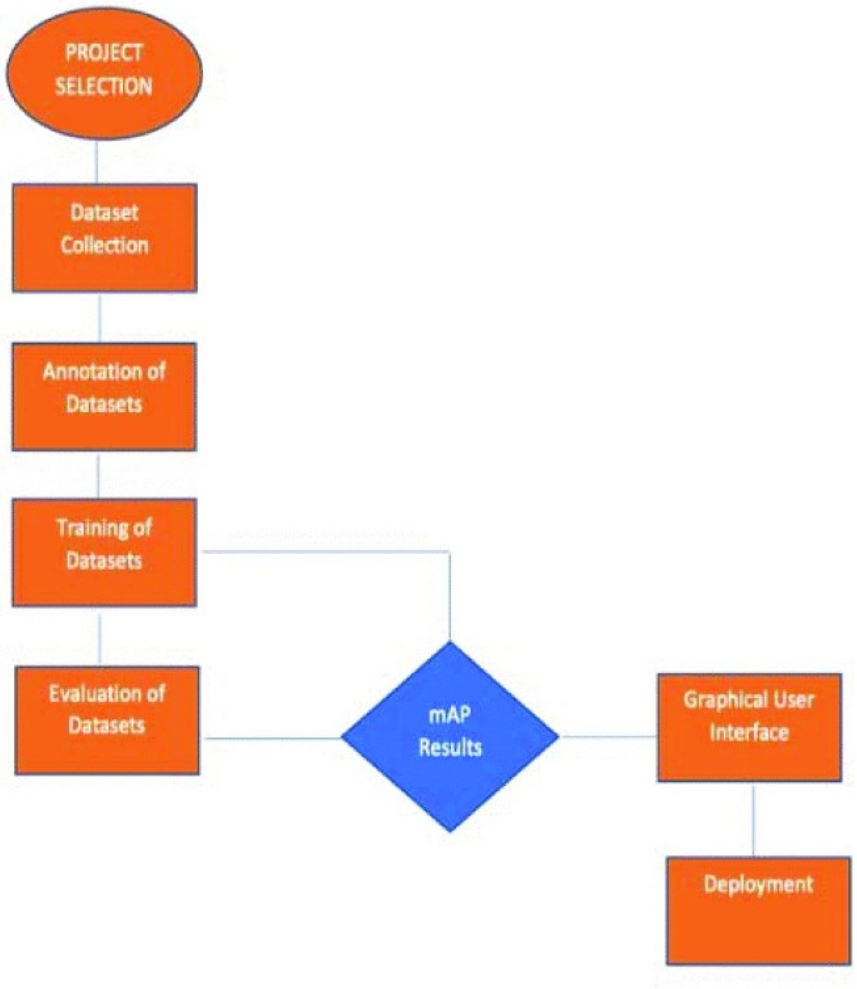


Figure-2.16 System workflow (Casuat *et al.*, 2020).

The strength of this study is the benchmarking of the models and results. It also collected data from on-site captures to enhance the object features. However, according to Figure-2.16, no proof of concept was outlined in this paper on how to handle the variation of the desired objects in real-time detection and how to handle the false positive detection as well.

In this paper, Zhang et al.(2021) proposed a one-stage object detection method based on convolutional neural networks. He presents a multi-scale strategy that selects the high-resolution feature maps of DarkNet-53 to effectively identify small-scale hardhats. In addition,

he presented an enhanced weighted bi-directional feature pyramid network (BiFPN), which could merge more semantic features from better scales. The offered method can not only detect hardhat but also recognize the color of the hardhat. The major strength of this detection method is to select high-resolution feature maps for object prediction to improve the accuracy of the model for small and medium objects. However, the use of data augmentation and on-site object features can help to improve the detection performance of small objects.

2.6 Conclusion

This chapter has outlined a potential review of the object detection, object detection algorithms and their performance comparison, recent studies on hardhat detection with the strengths and weaknesses. Many studies regarding the detection of worker hardhats had been conducted. Some researchers used different versions of YOLO models to detect hardhat-wearing in the construction sites; however, construction worker and hard-hat detection in real-time are very challenging because of the variation of the images captured from the on-sites. To detect smaller objects with different variations (object size, brightness, rotation, etc.) depends on the balanced dataset. Introducing the image augmentation features, active learning and on-site background images can improve the dataset to detect the diverse on-site objects and avoid detecting false positive objects.

3. Chapter Three: Proposed Approach

3.1 Fundamentals of YOLOv5

YOLO stands for 'You only look once', is one of the leading object detection algorithms in terms of speed and accuracy that splits images into a grid system. Each cell in the grid is responsible for detecting objects within itself (YOLOv5 Documentation, no date).

The history of the yolov5 is given below.

Version	**Author**	**Release**
YOLOv5	Glenn Jocher	18 May 2020
YOLOv4	Alexey Bochoknovskiy, Chien-Yao Wang, and Hong-Yuan Mark Liao	23 April 2020
YOLOv3	Joseph Redmon and Ali Farhadi	8 Apr 2018
YOLOv2	Joseph Redmon and Ali Farhadi	25 Dec 2016
YOLOv1	Joseph Redmon	8 Jun 2015

Table-3.1: The history of YOLO(YOLOv5 Documentation, no date)

Ultralytics released the first official version of YOLOv5 On June 25th (Solawetz, 2020). The performance of this new model is given below.

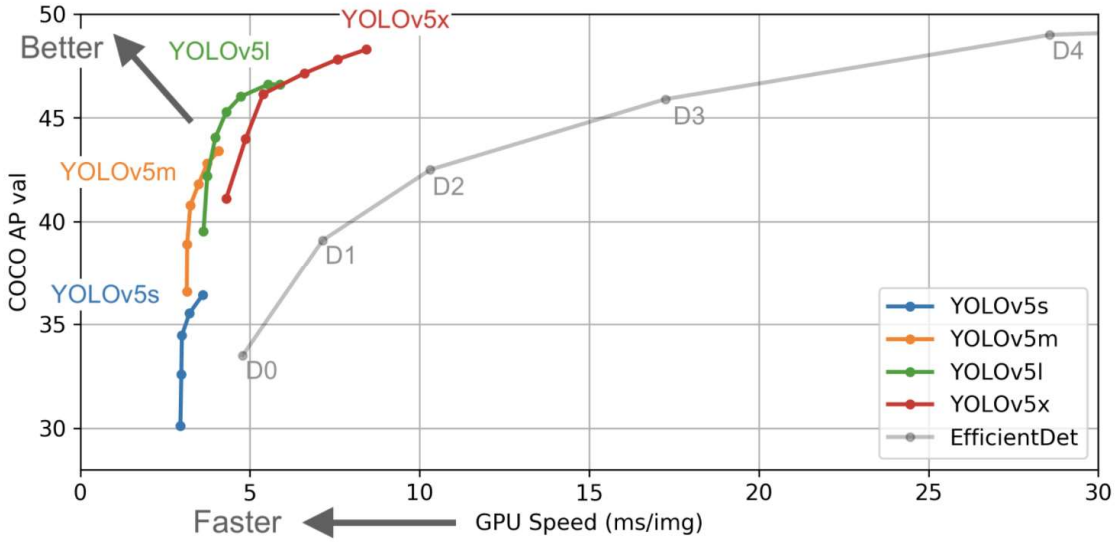


Figure-3.1: The first release of YOLOv5 shows the state of the art object detection (Solawetz, 2020)

In the above figure, the goal is to develop a high performed object detection model that is very efficient (Y-axis) relative to its inference time (X-axis). Initial results indicate that YOLOv5 exceeds well to this end relative to other state-of-the-art techniques. In summary, PyTorch training procedures improved YOLOv5 performance, while its architecture is almost close to YOLOv4.

YOLOv5 has four compound-scaled object detection models that were trained on the COCO dataset. Test Time Augmentation, model ensembling, hyperparameter evolution, and exporting to ONNX, CoreML and TFLite are the new additions in the latest YOLOv5.

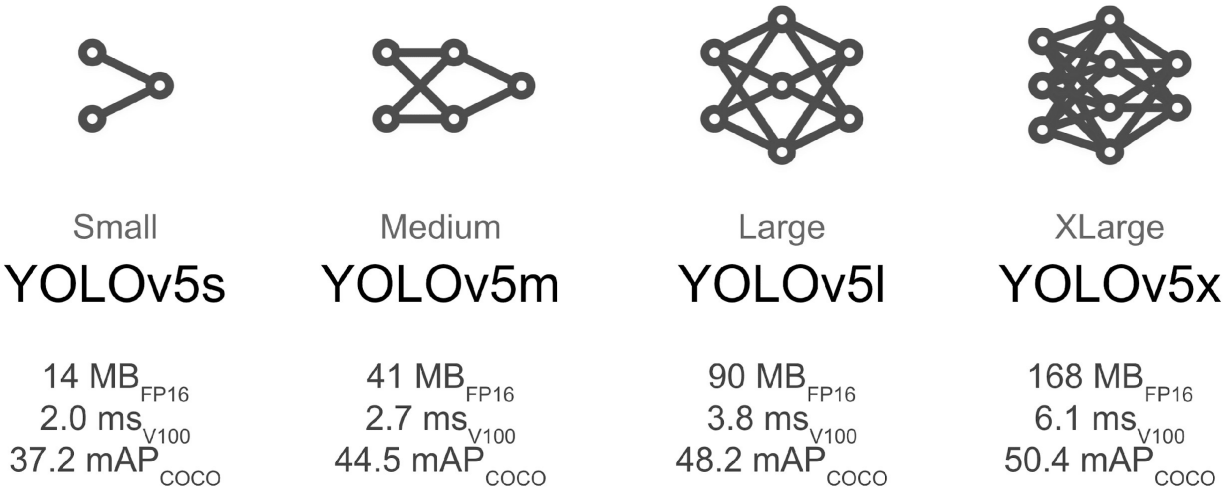


Figure-3.2: A family of YOLOv5 (PyTorch, no date)

The above figure depicts the summary of the compound-scaled YOLOv5 models including model size, interface time and mean average precision.

Figure-3.3 shows the network architecture of YOLOv5. The YOLOv5 network consists of three main parts.

1. Backbone (CSPDarknet): A convolutional neural network that receives the image input for feature extraction.
2. Neck (PANet): A series of layers to mix and combine image features to pass them forward to prediction.
3. Head (Yolo Layer): Receives features from the Neck and outputs the detection results within class name, object boundary box information and precision score.

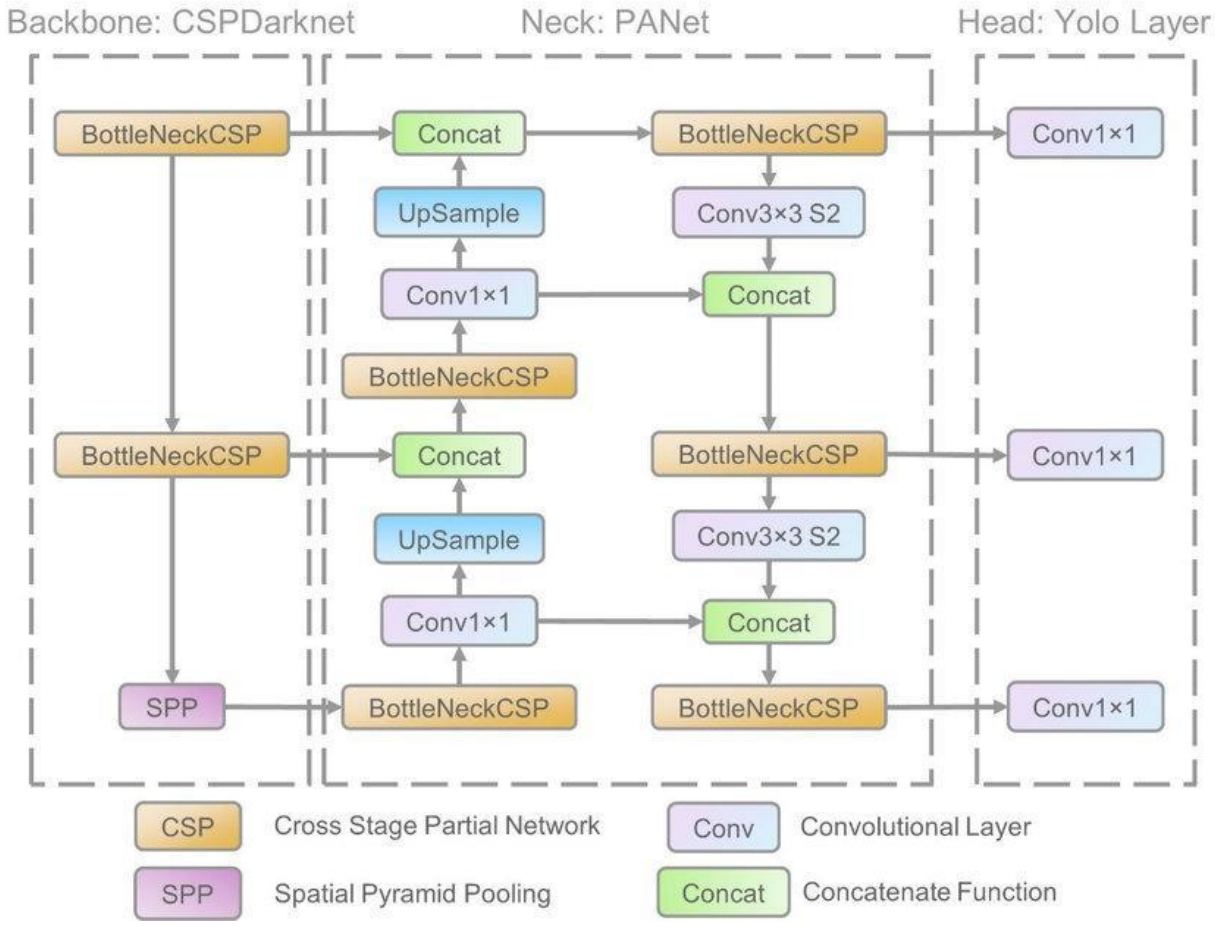


Figure-3.3: The network architecture of Yolov5 (Xu, Renjie & Lin, Haifeng & Lu, Kangjie & Cao, Lin & Liu, Yunfei, 2021)

There are three reasons for choosing YOLOv5 in this project. Firstly, cross stage partial network (CSPNet)(Wang *et al.*, 2020) is incorporated with Darknet that creates CSPDarknet as backbone in YOLOv5. CSPNet helps to resolve the repeated gradient information in large-scale backbones, and adds the gradient improvement into the feature map. Therefore, it decreases the

parameters and floating-point operations per second (FLOPS) of the model. It not only improves the inference speed and accuracy, but also reduces the model size. Secondly, the path aggregation network (PANet)(Wang *et al.*, 2019) was applied as the neck to increase information flow. PANet is adjusted with a new feature pyramid network (FPN) structure with the improved bottom-up path that enhances the propagation of low-level features. At the same time, adaptive feature pooling makes the useful information in each feature level propagate directly to the following subnetwork. PANet enhances the utilization of accurate localization signals in lower layers that can definitely improve the object location accuracy. Thirdly, the Head (YOLO layer) creates three different sizes (18 × 18, 36 × 36, 72 × 72) of feature maps to achieve multi-scale (Redmon and Farhadi, 2018) prediction and it allows the model to handle small, medium, and big objects.

3.2 System Design

This section represents the system design of detecting hardhat for the construction workers.

The system design consists of the following steps:

1. Dataset collection;
2. Data preprocessing;
3. Training setup;
4. Model testing;
5. API deployment;
6. User application development.

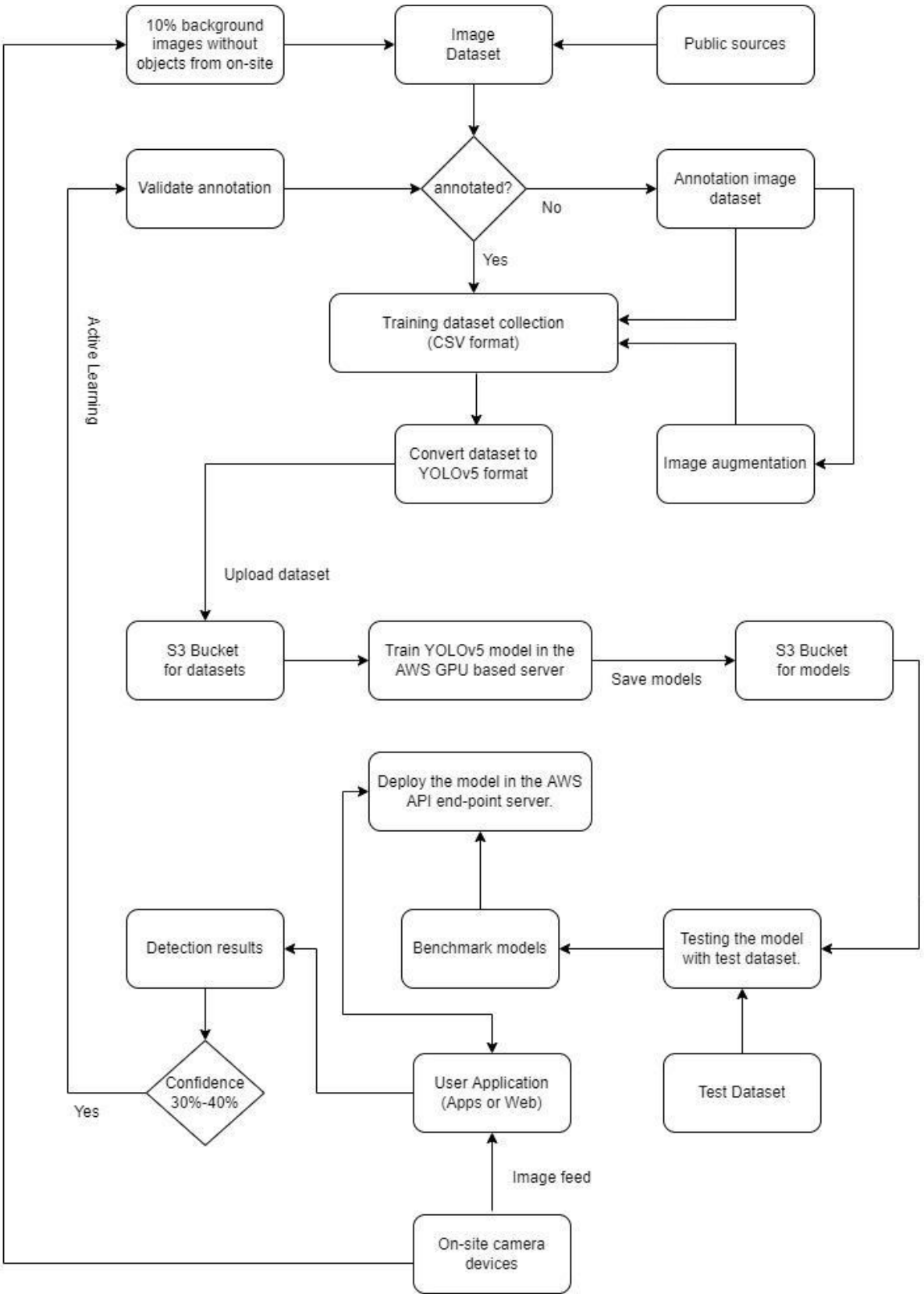


Figure-3.4: System diagram of detecting hardhat.

3.2.1 Dataset collection

The aim is to develop a balanced dataset for the hardhat detection that was collected from several sources like public sources, on-site as the part of active learning and the augmentation of the existing images.

The base dataset was collected from the Roboflow (Roboflow dataset, 2021) and Open images (Open Images Dataset V6, 2021). Table 3.2 shows the breakdown of the image sources by data source.

Source	**Number of Images**
Roboflow	7041
openimages	2500

Table-3.2: Base Dataset for hardhat detection

The total number of base images for this project is 9541. Images were annotated with the two main classes “Helmet” and “Head”. Figure-3.5 shows the annotated instances of helmet and head in the base dataset. This figure was taken from the training log.

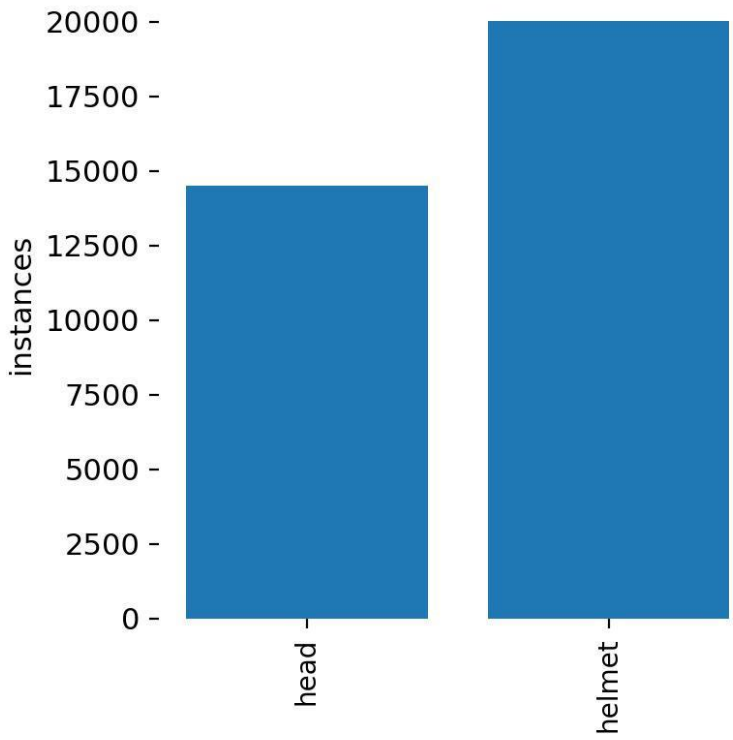


Figure-3.5: The number of Helmet and Head in the dataset.

3.2.1.1 Roboflow dataset

Roboflow hosts free public machine vision datasets in many popular formats (including CreateML JSON, COCO JSON, CSV, Pascal VOC XML, YOLO v3, and Tensorflow TFRecords). Some sample images of the hardhat dataset are given below.

Figure-3.6: Sample images of roboflow dataset (Roboflow dataset, 2021).

Roboflow hardhat dataset (Roboflow dataset, 2021) was downloaded in the form of CSV format annotation with the raw images. There are eight columns in the csv file such namely,

1. filename: The name of the image file.
2. width: Image width(px).
3. height: Image height(px).
4. class: Name of the object.
5. xmin, ymin: The coordinates of the top left corner of the bounding box of the object. Note: It starts from 1 and hence, both of them start from 1 and not 0.

6. xmax, ymax: The coordinates of the bottom right corner of the bounding box of the object Note: Maximum value of xmax is width of the image while maximum value of ymax is the height of the image.

filename	width	height	class	xmin	ymin	xmax	ymax
002141_jpg.rf.000c9596269b9771be6c4f1fea0b284a.jpg	416	416	helmet	326	127	367	175
002141_jpg.rf.000c9596269b9771be6c4f1fea0b284a.jpg	416	416	helmet	121	102	159	155
002141_jpg.rf.000c9596269b9771be6c4f1fea0b284a.jpg	416	416	helmet	284	167	308	195
002141_jpg.rf.000c9596269b9771be6c4f1fea0b284a.jpg	416	416	helmet	325	0	367	12
002141_jpg.rf.000c9596269b9771be6c4f1fea0b284a.jpg	416	416	helmet	121	0	160	37

Table-3.3: CSV format roboflow dataset.

3.2.1.2 Open image dataset

Open images dataset is one of the biggest dataset for machine vision activities. It has around 9 million varied images with rich annotations (Open Images Dataset V6, 2021). Only 2500 human face images with text annotation files were downloaded to balance the base dataset between head and helmet.

The following python package was installed to download the open image dataset.

```
pip install oidv6
```

```
pip install --upgrade oidv6
```

The terminal command for downloading classes (Human face) in one local directory from the train sets with labels in automatic mode and image limit = 2500

```
oidv6 downloader --dataset path_to_directory --type_data train --classes
Human face --limit 2500 --multi_classes --yes
```

Each image has a text annotation file including 5 values per line for each object, namely.

1. 1st value: Object name
2. 2nd and 3rd values: The coordinates of the top left corner of the bounding box of the object.
3. 4th and 5th values: The coordinates of the bottom right corner of the bounding box of the object.

Figure-3.7 provides a sample text file.

```
head 650.999808 71.00016000000001 766.999552 122.99976
head 280.000512 336.99984 300.99968 368.99996
head 305.000448 324.00028 337.000448 358.99988
head 332.99968 300.99996 355.999744 343.99976000000004
head 339.999744 323.0 389.999616 368.99996
head 437.000192 255.0 510.000128 314.9998
```

Figure-3.7: Sample text annotation

As the initial data format is CSV, each text file was converted to the CSV format and combined into a single csv file for the whole open image dataset. The following Python function was used for the data conversion from text format to csv format.

```
def openimage_to_csv():
    train_list = []
    test_list = []
    inc = 1
    for txt_file in glob.glob('train/head/labels/*.txt'):
        img_file = txt_file.replace("train/head/labels/", '')
        img_file = img_file.replace(".txt", ".jpg")
        df = pd.read_csv(txt_file, header=None, sep=' ' )
        df.columns =['label', 'xmin', 'ymin', 'xmax', 'ymax']
        for i, row in df.iterrows():
            xmin = row['xmin']
            ymin = row['ymin']
            xmax = row['xmax']
            ymax = row['ymax']
            label = row['label']

            value = (img_file,
                     label,
                     xmin,
                     ymin,
                     xmax,
                     ymax
                     )
            train_list.append(value)

        inc = inc + 1
```

```
column_name = ['filename', 'class', 'xmin', 'ymin', 'xmax', 'ymax']
train_df = pd.DataFrame(train_list, columns=column_name)
return train_df
```

3.2.1.3 Image annotation tool

The Roboflow annotation tool was used to annotate each object in a single image. It automatically created equivalent label files for each image. Most important feature of this platform is to export the dataset in any well-known format for computer vision. This tool has easy drag-drop and object selection facilities to label a specific object in an image. Figure-3.7 was taken from the Roboflow annotation tool. It shows how to annotate or label an object in an image.

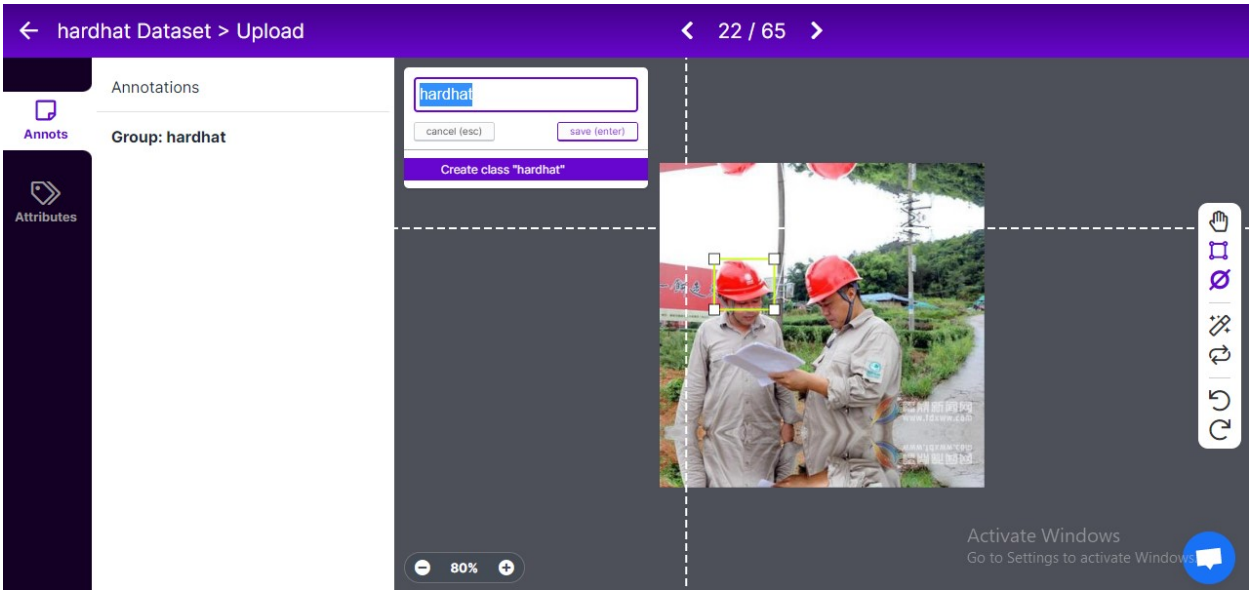


Figure-3.7: Roboflow annotation tool (Roboflow annotate, 2021).

3.2.1.4 Image augmentation

Image augmentation is one of the most essential steps to develop a balanced dataset that can increase the generalizability of the model performance by increasing the diversity of learning features from the dataset. Roboflow augmentation platform was used in this project. The following key augmentation features were considered to increase the diversity of the image dataset. There were two types of augmentation available in the Roboflow augmentation tool.

1. Image level augmentation: Figure-3.8 shows the image level augmentation features available in Roboflow.

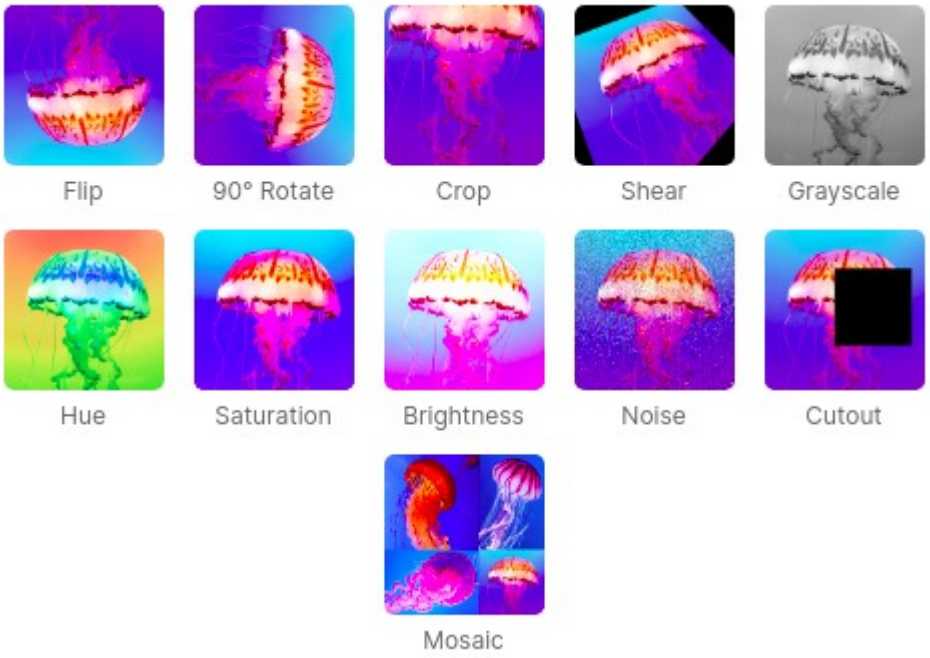


Figure-3.8: Image level augmentation.

2. Bounding box level augmentation: Figure-3.9 shows the Boundary box level augmentation features available in Roboflow.

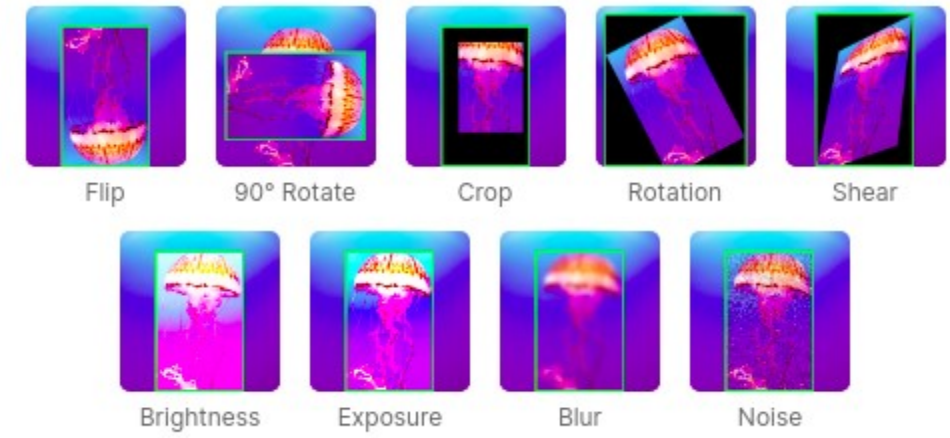


Figure-3.9: Boundary box level augmentation (Roboflow augmentation, 2021).

The following augmentation features were used at both levels to enhance the dataset.

1. Rotation: (+/-) 15 deg rotation was used to add variability to rotations to help the model be more resilient to camera roll.

2. Exposure: (+/-) 25 % exposure was used to add variability to image brightness to help the model be more resilient to lighting and camera setting changes.
3. Shear: (+/-) 10 deg was used to add variability to perspective to help the model be more resilient to camera and subject pitch and yaw.
4. Blur: 2px blur was used to add random Gaussian blur to help the model be more resilient to camera focus.
5. Noise: 5% noise was used to add noise to help the model be more resilient to camera artifacts.
6. Mosaic: It was used at the training stage to help the model perform better on small objects. Mosaic combines multiple images from the training set into a collage.

This augmentation tool helped a lot to increase the image variety in the dataset so that the model can detect the object from different times of day, different seasons, different weather, different lighting, different angles, and different sources.

Table-3.3 shows the different sample augmented images as an example.

Figure-3.10: Rotation

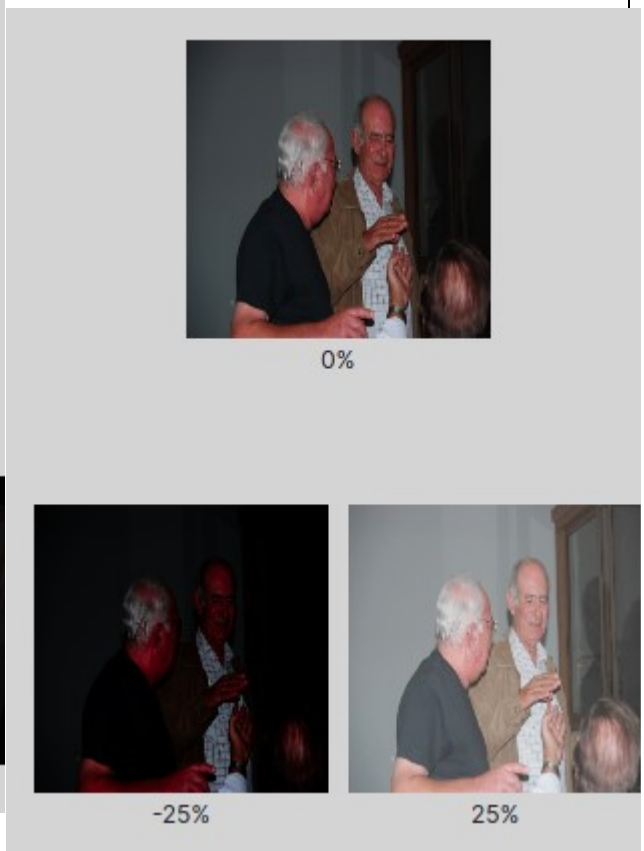


Figure-3.11: Exposure

Figure-3.12: Shear

Figure-3.13: Blur

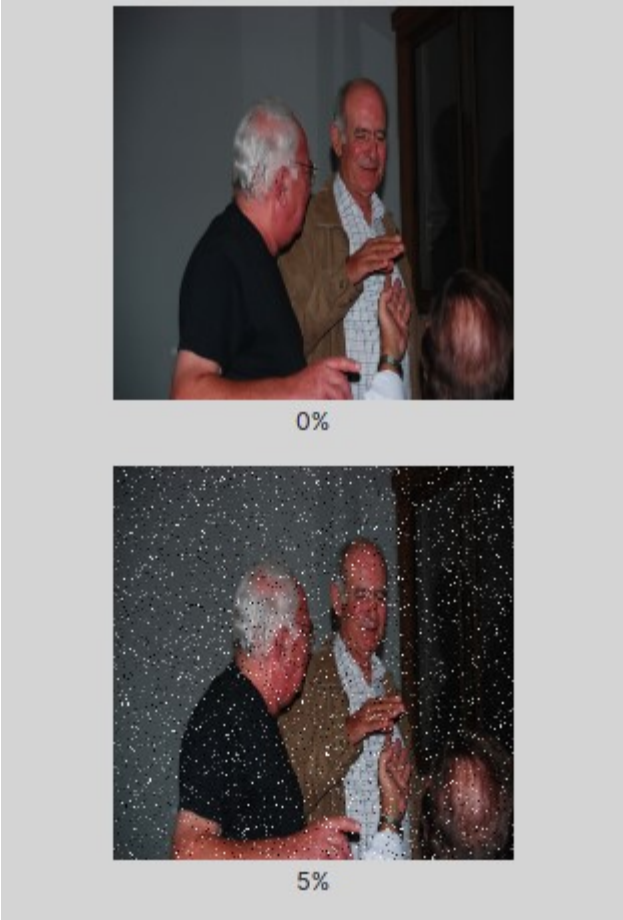


Figure-3.14: Noise

Figure-3.15: Mosaic

Table-3.4: Augmented image features.

3.2.1.5 Active learning

Active learning is a strategy for collecting or identifying the on-site data that can best improve the model performance. The system automatically collects the on-site images which have the 30-40% confidence of the desired objects. Then the collected images are manually validated for the correct annotation and added to the next dataset for further training. It increases the model performance to detect the desired objects with high confidence.

3.2.1.6 Background images

Background images with no objects were collected from on-site and added to the training dataset to reduce False Positives (FP) detection. The system collected 5% background images

for the training dataset. It helps the model to increase the performance over time for reducing the possible False Positives (FP) detection.

3.2.1.7 YOLOv5 dataset

YOLO v5 expects a .txt annotation file for each image where each line of the text file represents bounding box information of a single object in an image. Consider the following image. (Kathuria, 2021)

Figure-3.16: Sample images for YOLOv5 dataset. (Kathuria, 2021)

The annotation file of the above image looks like the following text file.

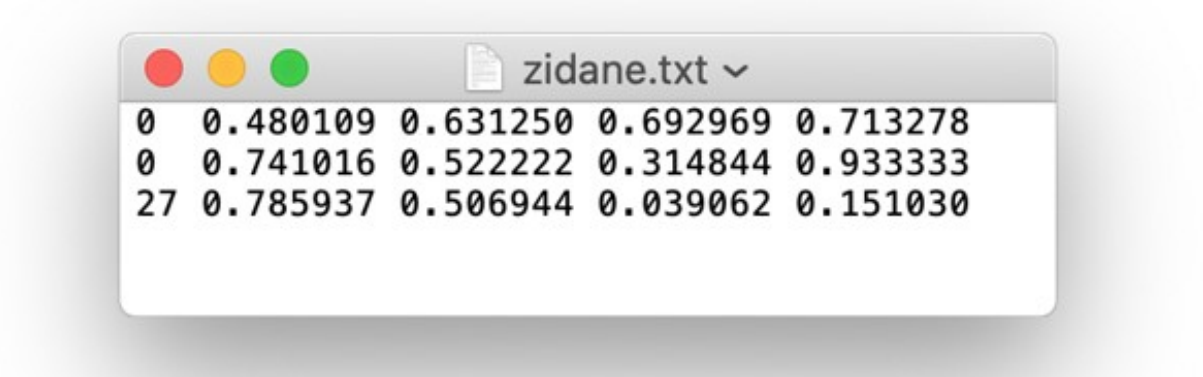


Figure-3.17: Sample .txt annotation file for YOLOv5 dataset. (Kathuria, 2021)

In the above image, there are three objects (2 persons and one tie). Each line of the .txt annotation file represents the following.

1. A single row per object
2. Each row represents class, x_center, y_center, width, height.
3. Annotation boxes must be normalized based on the image dimension(i.e. have values between 0 and 1)
4. Class: Index number of class list which starts from zero index.

Folder structure of the YOLOv5 dataset:

```
Dataset
    ├───test
    │    └───images
    │    └───labels
    ├───train
    │    └───images
    │    └───labels
    ├───data.yaml
```

The Dataset folder consists of an important file called data.yaml file. The specification of the file is given as follows.

```
train: ../train/images

val: ../test/images

nc: 2
names: ['head', 'helmet']
```

1. train: training image path
2. val: validation image path
3. nc: number of the class
4. names: list of the class names

Before training the dataset, the csv format dataset needs to convert into YOLOv5 format dataset. The following Python code will take the annotations in csv format and convert them to YOLOv5 format.

```
def convert(size, box):
    dw = 1./(size[0])
    dh = 1./(size[1])
    x = (box[0] + box[1])/2.0 - 1
    y = (box[2] + box[3])/2.0 - 1
    w = box[1] - box[0]
    h = box[3] - box[2]
    x = x*dw
```

```
    w = w*dw
    y = y*dh
    h = h*dh
    return (x,y,w,h)


input_path = 'data/input/'
output_path = 'data/output/'

classes = ['head', 'helmet']

df = pd.read_csv(input_path + 'labels.csv')

filename_group = df.groupby("filename")


for name, groups in filename_group:

    file_name = input_path + name

    basename = os.path.basename(file_name)
    basename_no_ext = os.path.splitext(basename)[0]

    image = imageio.imread(file_name)

    w = int(image.shape[1])
    h = int(image.shape[0])

    out_file = open(output_path + basename_no_ext + '.txt', 'w')


    for key, obj in groups.iterrows() :

        cls = obj['class']
        if cls not in classes:
            continue
        cls_id = classes.index(cls)

        b = (float(obj['xmin']), float(obj['xmax']), float(obj['ymin']),
float(obj['ymax']))
        bb = convert((w,h), b)
        out_file.write(str(cls_id) + " " + " ".join([str(a) for a in bb]) +
'\n')

print("Conversion done!!")
```

3.2.2 AWS environment setup

To work on the computer vision project needs high computational power to train the image dataset. AWS has pre-installed deep learning containers with deep learning frameworks to make it easy to deploy custom machine learning environments quickly.

The following server instances and storages were installed in the AWS platform.

1. S3 Buckets(Storage)
2. GPU Instance
3. Inference Instance

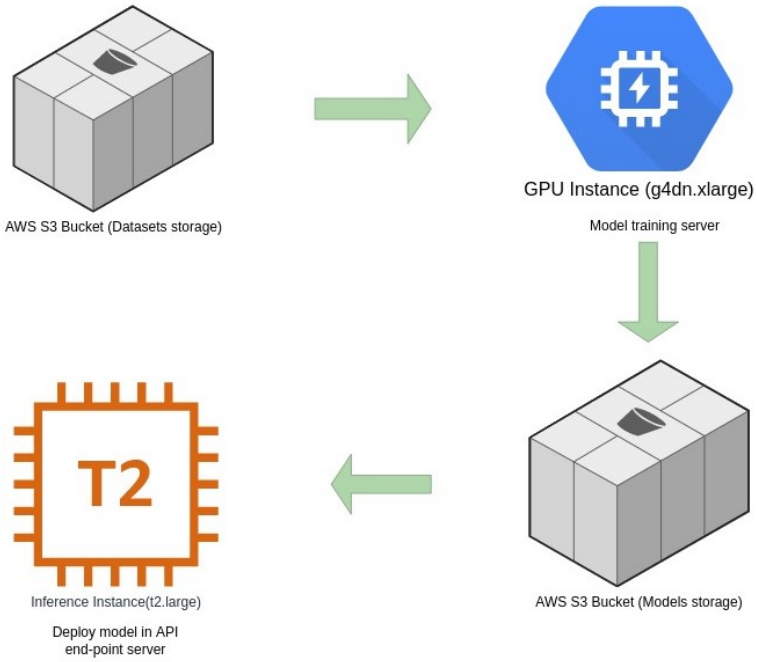


Figure-3.18: AWS System Diagram

3.2.2.1 AWS S3 bucket

Two S3 buckets were set up, one for datasets and one for trained models. Both S3 buckets were connected to my local machine. The new version of the datasets generated from the data collection pipeline are being stored in S3 bucket (s3://mv-datasets). Similarly, the new version of the models generating from the training pipeline are being stored in S3 bucket (s3://mv-models)

S3 Buckets are accessible with the AWS dashboard. However CLI is the great option to connect to the server. The following CLI commands are useful to upload or download the files and folders.

Sync from S3 to the local machine:

```
aws s3 sync s3://mv-models /.
```

Sync from local machine to S3:

```
aws s3 sync /. s3://mv-models
```

3.2.2.2 AWS GPU Instance

The server instance (g4dn.xlarge) was installed in the AWS cloud platform to set up the yolov5 training pipeline. EC2 G4 instance is a most cost-effective and versatile GPU instance for deploying machine learning models such as image classification, object detection. G4 instances are available with a choice of NVIDIA GPUs (G4dn) or AMD GPUs (G4ad). It's the lowest cost GPU-based instance in the cloud for machine learning inference and small scale training. It's a NVIDIA t4 GPU with 4 core CPUs and 16 GB of RAM. (Amazon, no date)

Connect to Linux instance using SSH:

```
ssh ubuntu@34.197.96.141 -i ~/mv/login.pem
```

After connecting to the GPU server, different environment variables were available. As YOLOv5 is pytorch based, pytorch_latest_p37 was selected as an environment variable. As YOLOv5 supports Python >= 3.6.0, Python 3.7.10 was installed on the server. The following terminal command was used to select the environment variable.

```
source activate putorch_latest_p37
```

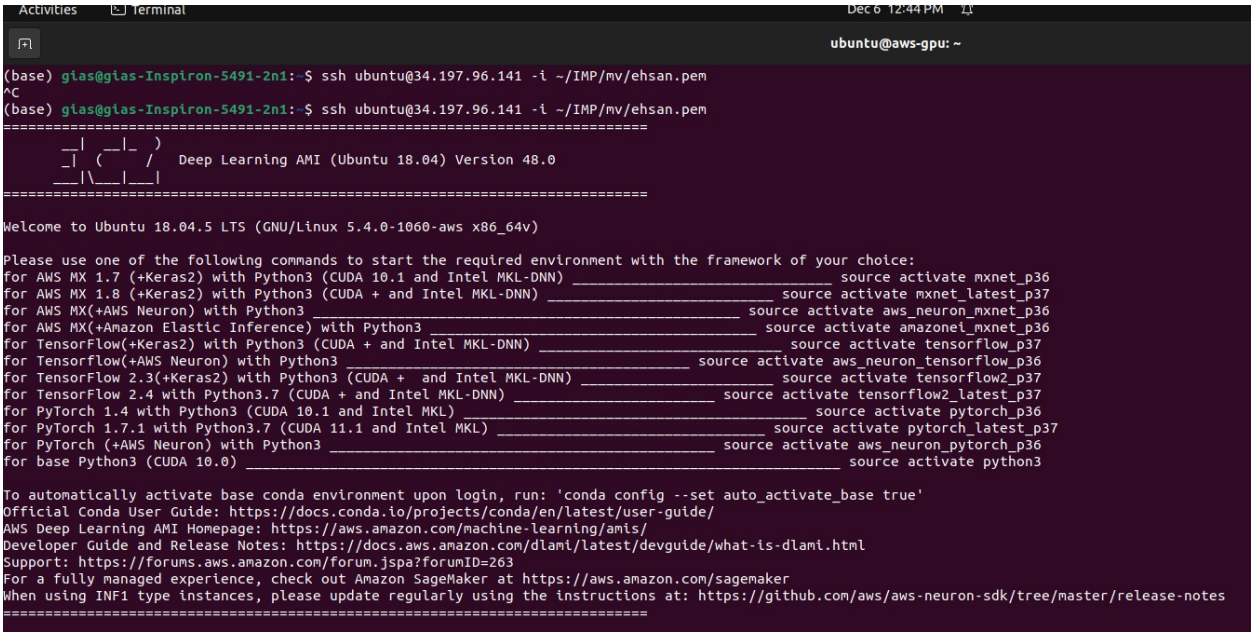


Figure-3.19: List of environment variables in g4dn.xlarge(GPU) server.

3.2.2.3 AWS Inference Instance

The server instance (t2.large) was installed in the AWS cloud platform to deploy models at the API end-point. EC2 t2 instance is a cost-effective and small scale CPU instance for deploying models for object detection. It has 2 core CPUs and 8 GB of RAM. As YOLOv5 supports Python >= 3.6.0, Python 3.7.10 and Flask library were installed in the server to deploy Python based API. User application calls the API function for detecting the objects from a given image.

3.2.3 YOLOv5 training

The training pipeline on custom dataset in AWS GPU machine is structured as follows.

1. Task overview
2. Configuration details
3. Train Model
4. Inference.

3.2.3.1 Task overview

To start training on the custom dataset needs to set up YOLOv5 code in the training server, define training parameters, configure dataset, hyperparameters and network architecture.

Then it's ready to start training the YOLOv5 model on the custom dataset. Finally, the trained model is ready to test on the real-time detection.

First of all, begin by cloning the YOLO v5 repository and setting up the dependencies required to run YOLO v5.

Command in terminal:

Clone the yolov5 source code from the repository.

```
git clone https://github.com/ultralytics/yolov5
```

Install all required dependencies mentioned in following txt file.

```
pip install -r yolov5/requirements.txt
```

3.2.3.2 Configuration details

There are several training options or parameters. They were configured as follows.

img: The size of the image. The image is a square one. The original image is resized while maintaining the aspect ratio. The dimension of the image is 416px.

batch: A numeric number to define the batch size.

epochs: A numeric number of epochs to train the model.

data: A YAML file that contains the information about the dataset (path of images, labels)

cfg: It's the yaml file to define model architecture. There are four available model architecture options: yolo5s.yaml, yolov5m.yaml, yolov5l.yaml, yolov5x.yaml. The size and complexity of these models increases in the ascending order and It's essential to choose a model which suits the complexity of the object detection task. In case of working with a custom architecture, it's important to define a YAML file in the models folder specifying the network architecture.

weights: It defines the pre-trained weights to start training from. If there is no value of this parameter, that means to start training from scratch, e.g. --weights ' '. There are four available pre-trained weights: yolo5s.pt, yolov5m.pt, yolov5l.pt, yolov5x.pt. The size and complexity of these models increases in the ascending order and need to choose a weight which suits the complexity of the object detection task.

name: The name of the folder to save various things about training such as train logs. Training weights would be stored in a folder named runs/train/name

hyp: It's a YAML file that describes hyperparameter choices. If unspecified, the default file data/hyp.scratch.yaml is used.

Data Config File:

The detailed configurations of the dataset to train the model on are defined by the data config YAML file. The following parameters have to be defined in a data config file:

train, test, and val: The locations of train, test, and validation images in the dataset.

nc: A numeric number to define the classes in the dataset.(nc=2; hardhat, head)

names: The list of the classes in the dataset. The index of the classes in this list would be used as an identifier for the class names in the code.

data.yaml was created inside the dataset folder and placed it in the yolov5/dataset. Then it was populated with the following.

```
train: ../dataset/images/train/
val:  ../dataset/images/val/
test: ../dataset/images/test/

# number of classes
nc: 2

# class names
names: ["head","hardhat"]
```

YOLO v5 looks for an equivalent label file for each image in the folder whose name can be derived by replacing images with labels in the path to dataset images. For example, in the example above, YOLO v5 looks for train labels in ../dataset/train/labels. Similarly test and validation labels are in ../dataset/test/labels, ../dataset/val/labels.

Hyperparameter Config File:

Hyperparameters were defined in the hyperparameter config file for the neural network of the YOLOv5 model. The default config file name and location is data/hyp/hyp.scratch.yaml. The file is given as follows.

```
# Hyperparameters for COCO training from scratch
# python train.py --batch 40 --cfg yolov5m.yaml --weights '' --data coco.yaml
--img 640 --epochs 300
# See tutorials for hyperparameter evolution
https://github.com/ultralytics/yolov5#tutorials
```

```
lr0: 0.01  # initial learning rate (SGD=1E-2, Adam=1E-3)
lrf: 0.2  # final OneCycleLR learning rate (lr0 * lrf)
momentum: 0.937  # SGD momentum/Adam beta1
weight_decay: 0.0005  # optimizer weight decay 5e-4
warmup_epochs: 3.0  # warmup epochs (fractions ok)
warmup_momentum: 0.8  # warmup initial momentum
warmup_bias_lr: 0.1  # warmup initial bias lr
box: 0.05  # box loss gain
cls: 0.5  # cls loss gain
cls_pw: 1.0  # cls BCELoss positive_weight
obj: 1.0  # obj loss gain (scale with pixels)
obj_pw: 1.0  # obj BCELoss positive_weight
iou_t: 0.20  # IoU training threshold
anchor_t: 4.0  # anchor-multiple threshold
# anchors: 3  # anchors per output layer (0 to ignore)
fl_gamma: 0.0  # focal loss gamma (efficientDet default gamma=1.5)
hsv_h: 0.015  # image HSV-Hue augmentation (fraction)
hsv_s: 0.7  # image HSV-Saturation augmentation (fraction)
hsv_v: 0.4  # image HSV-Value augmentation (fraction)
degrees: 0.0  # image rotation (+/- deg)
translate: 0.1  # image translation (+/- fraction)
scale: 0.5  # image scale (+/- gain)
shear: 0.0  # image shear (+/- deg)
perspective: 0.0  # image perspective (+/- fraction), range 0-0.001
flipud: 0.0  # image flip up-down (probability)
fliplr: 0.5  # image flip left-right (probability)
mosaic: 1.0  # image mosaic (probability)
mixup: 0.0  # image mixup (probability)
```

Most of the default setting was used to train the model except some augmentation features at the training stage. Image rotation was used (+/-) 15 deg and Image shear was used (+/-)10 deg. YOLOv5 uses mosaic augmentation techniques at the training stage. This is what it looks like.

Figure-3.20: YOLOv5 mosaic augmentation.

Custom Network Architecture:

YOLO v5 also allows configuring the custom architecture and anchors if the pre-defined networks don't fit the bill for a specific object detection project. For this reason, it's important to define a custom weights config file. The default yolov5s.yaml was used in this project. The config file looks like this.

```
# parameters
nc: 80  # number of classes
depth_multiple: 0.33  # model depth multiple
width_multiple: 0.50  # layer channel multiple

# anchors
```

```
anchors:
  - [10,13, 16,30, 33,23]  # P3/8
  - [30,61, 62,45, 59,119]  # P4/16
  - [116,90, 156,198, 373,326]  # P5/32

# YOLOv5 backbone
backbone:
  # [from, number, module, args]
  [[-1, 1, Focus, [64, 3]],  # 0-P1/2
   [-1, 1, Conv, [128, 3, 2]],  # 1-P2/4
   [-1, 3, C3, [128]],
   [-1, 1, Conv, [256, 3, 2]],  # 3-P3/8
   [-1, 9, C3, [256]],
   [-1, 1, Conv, [512, 3, 2]],  # 5-P4/16
   [-1, 9, C3, [512]],
   [-1, 1, Conv, [1024, 3, 2]],  # 7-P5/32
   [-1, 1, SPP, [1024, [5, 9, 13]]],
   [-1, 3, C3, [1024, False]],  # 9
  ]

# YOLOv5 head
head:
  [[-1, 1, Conv, [512, 1, 1]],
   [-1, 1, nn.Upsample, [None, 2, 'nearest']],
   [[-1, 6], 1, Concat, [1]],  # cat backbone P4
   [-1, 3, C3, [512, False]],  # 13

   [-1, 1, Conv, [256, 1, 1]],
   [-1, 1, nn.Upsample, [None, 2, 'nearest']],
   [[-1, 4], 1, Concat, [1]],  # cat backbone P3
   [-1, 3, C3, [256, False]],  # 17 (P3/8-small)

   [-1, 1, Conv, [256, 3, 2]],
   [[-1, 14], 1, Concat, [1]],  # cat head P4
   [-1, 3, C3, [512, False]],  # 20 (P4/16-medium)

   [-1, 1, Conv, [512, 3, 2]],
   [[-1, 10], 1, Concat, [1]],  # cat head P5
   [-1, 3, C3, [1024, False]],  # 23 (P5/32-large)

   [[17, 20, 23], 1, Detect, [nc, anchors]],  # Detect(P3, P4, P5)
  ]
```

In the training script, yolov5 uses the default configuration. For custom architecture, need to define the config file path. For example, --cfg ../models/custom_cfg.yaml.

3.2.3.3 Train Model

The location of train, val and test, the number of classes (nc) and the names of the classes were defined in the data.yaml of the dataset. The model was started training from the pre-trained

model yolo5s.pt to keep things simple and avoid overfitting. A batch size of 32, image size of 416, and train for 100 epochs were used in the training pipeline to utilize the smaller GPU server. If any issues were raised fitting the model into the memory, the following things were considered.

1. Smaller batch size
2. Smaller network
3. Smaller image size

Of course, all of the above things might impact the performance. It depends on the design decision and the computational power of the GPU machine.

The prefix of the name was used “hardhat-v06-” for the training log. The tensorboard training logs can be found at runs/train/ hardhat-v06-01 or next is hardhat-v06-02 and so on.

Final training script:

```
python train.py --img 416 --cfg ./models/yolov5s.yaml --hyp
./data/hyp/hyp.scratch.yaml --batch 32 --epochs 100 --data
./datasets/data.yaml --weights yolov5s.pt --name hardhat-v06-
```

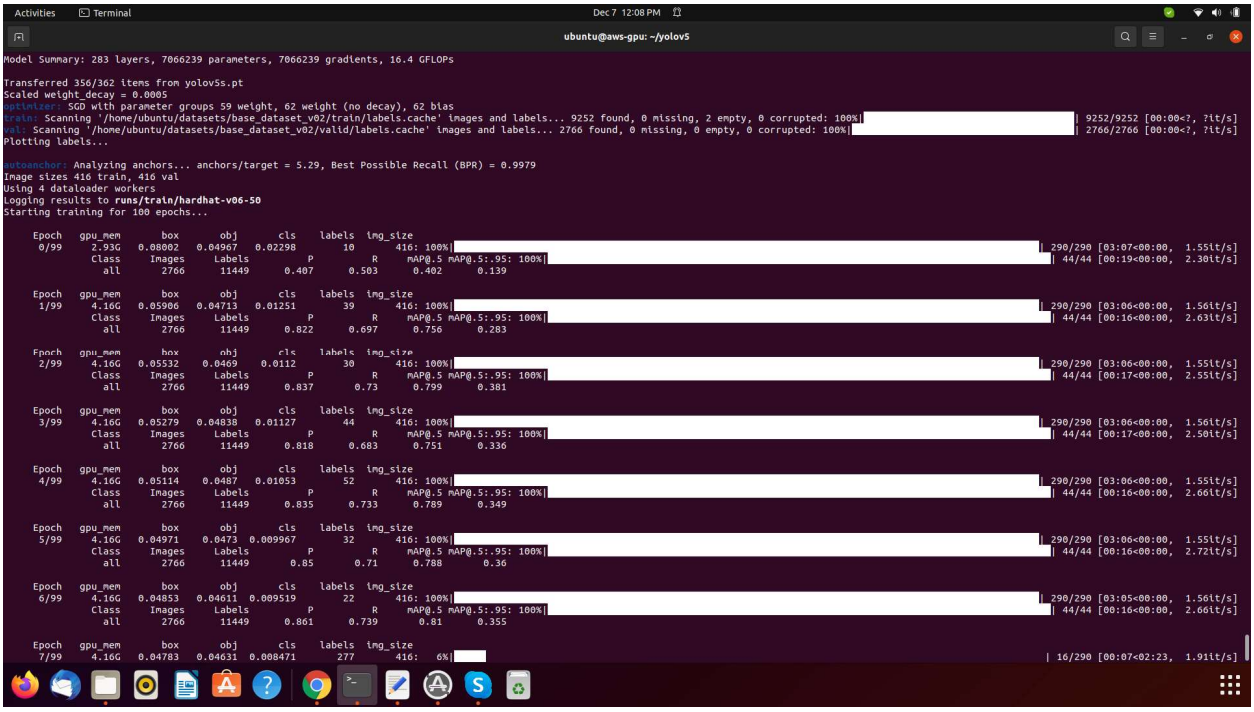


Figure-3.21: Training YOLOv5 in GPU server.

Each epoch took around 3.5 minutes for training and validation in the g4dn.xlarge GPU server. That means 350 minutes total for 100 epochs.

The following script was used in the local notebook to monitor the training log in the tensorboard.

```
# Tensorboard
%load_ext tensorboard
%tensorboard --logdir runs/train
```

The training log looks like a tensorboard. It's representing the mean average precision (mAP) of each model against the number of epochs.

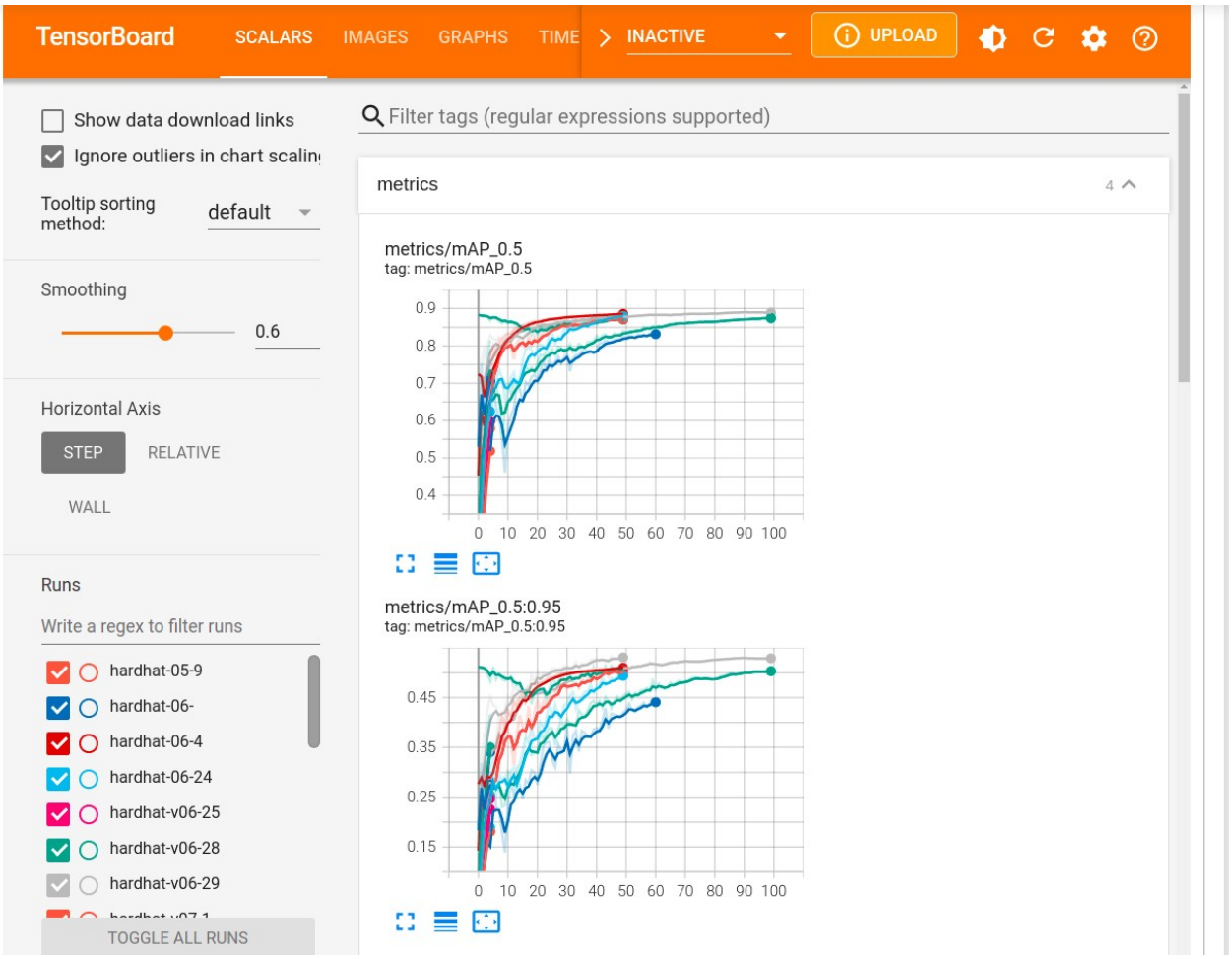


Figure-3.22: Training log tensorboard.

3.2.3.4 Inference

Several sources of the detector were used to run inference using the detect.py file.The --source parameter defines the source of the detector, which can be:

1. The location of the single image
2. The location of the image folder.
3. The location of the video
4. Webcam

For example: --source ../hardhat_test_images/batch1

--weights: The parameter defines the path of the model which was trained on the custom dataset to run the detector. The best.pt weight was used for the detection results. The best.pt contains the best-performing weight that was saved during the training.

--conf: The parameter is the minimum threshold value of the precision to consider the detection objects in an image. 50% confidence was used for the detection objects.

--name: The parameter defines the location where the detection results are stored. It was hardhat_det; therefore, the detection results would be stored in runs/detect/hardhat_det/.

--IoU: It defines the amount of overlap between the predicted and ground truth bounding box. It is a number from 0 to 1. 30% iou was used when calculating mAP.

The full inference script is given as follows.

```
!python detect.py --source ../ hardhat_test_images /batch1/ --weights
runs/train/hardhat-v06-45/weights/best.pt --conf 0.50 –iou 0.30 --name
yolo_road_det
```

Apart from the source of the image folder, other sources can be used for the detector as well. The sample command is given as follows.

```
python detect.py
--source 0  # webcam
         file.jpg  # single image file.
         file.mp4  # single video file
         path/  # the location of image directory
         path/*.jpg  # The only jpg files from a directory

rtsp://170.93.143.139/rtplive/470011e600ef003a004ee33696235daa  # rtsp stream
rtmp://192.168.1.105/live/test  # rtmp stream
http://112.50.243.8/PLTV/88888888/224/3221225900/1.m3u8  # http stream
```

Some real time detection results are given as follows.

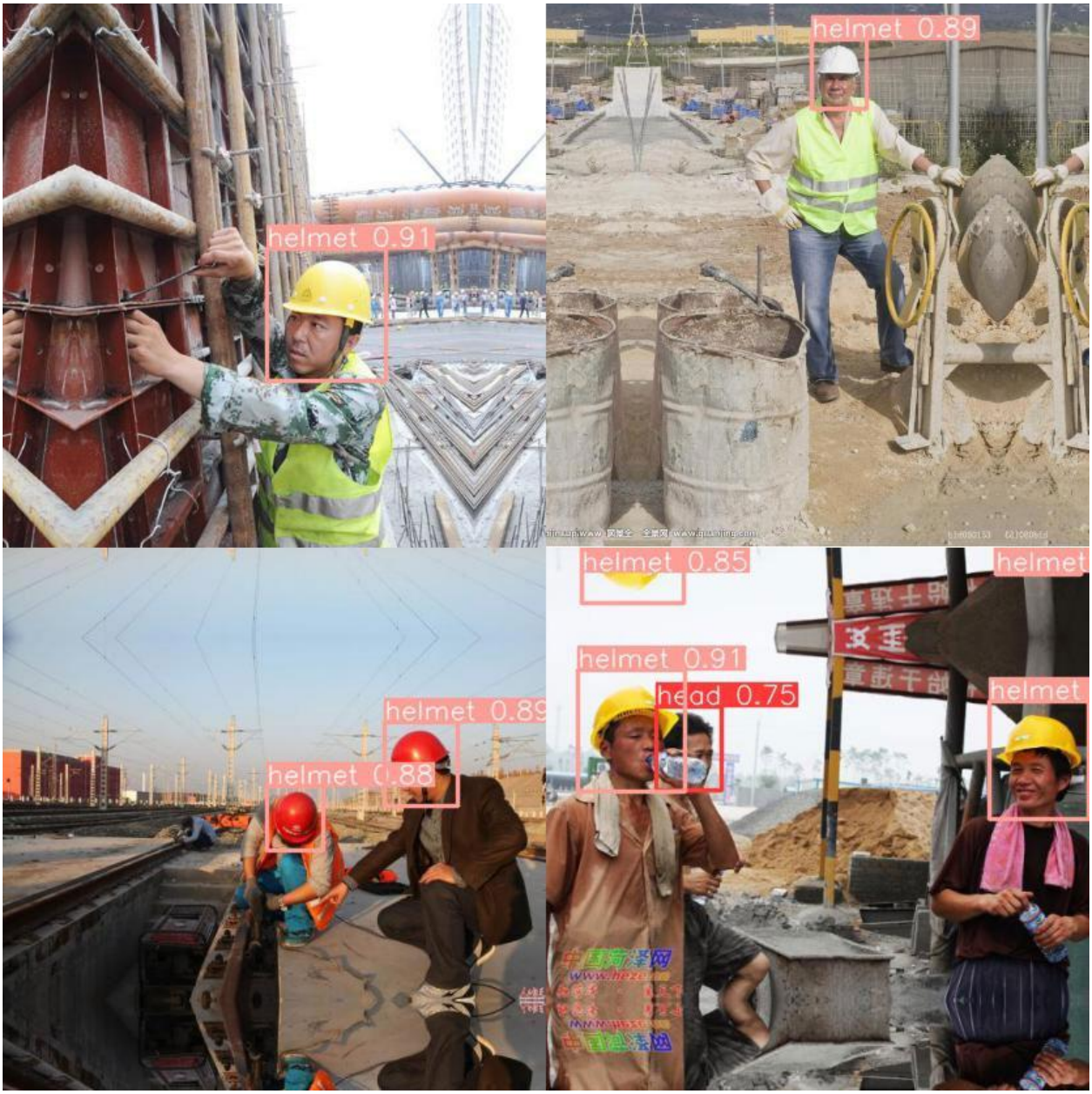


Figure-3.23: Sample test results

3.2.4 Test Dataset & Benchmark models

Test dataset is the standard dataset that was collected from the on-site environment for testing the models that were generated from the training pipeline. When it was collected, image variety was considered. Test images were captured from different times of the day, different weather, different lighting, different angles and different sources (scraped online, collected

locally, different cameras) etc. The models generated from the training pipeline were trained on the different versions of custom training datasets. To benchmark the models, the test dataset was used to evaluate the performance of the overall models. The model and dataset history were recorded during the execution of the whole process.

3.2.5 Model deployment and API

API end-point was developed and deployed in the AWS server. Always the best model generated from the training pipeline, was deployed in the API end-point server. Python library Flask was used to develop the python based API end-point. Any web/apps/backend portal can consume the api for detecting the hardhat/head from the given image. The sample api function is given as follows.

```
@app.route("/detect", methods=['POST'])
def detect():
    if not request.method == "POST":
        return
    image = request.files['image']
    confidence = request.form['confidence']
    overlap = request.form['overlap']
    model = request.form['model']
    output = request.form['output']

    image.save(os.path.join(uploads_dir, secure_filename(image.filename)))
    subprocess.run(['python3', 'detect.py', '--source',
os.path.join(uploads_dir, secure_filename(image.filename)), '--conf-thres',
conf, '--iou-thres', ovl, '--weights', model, '--augment'])
    obj = secure_filename(image.filename)

    if(output == 'json')
        obj = get_json(image.filename)

    return obj
```

Function name: detect

Method: POST

Input parameters: @image, @confidence, @overlap, @model and @output

Output: Detected image with boundary boxes or json format.

API test interface is located at http://ec2-44-198-243-194.compute-1.amazonaws.com:8080/

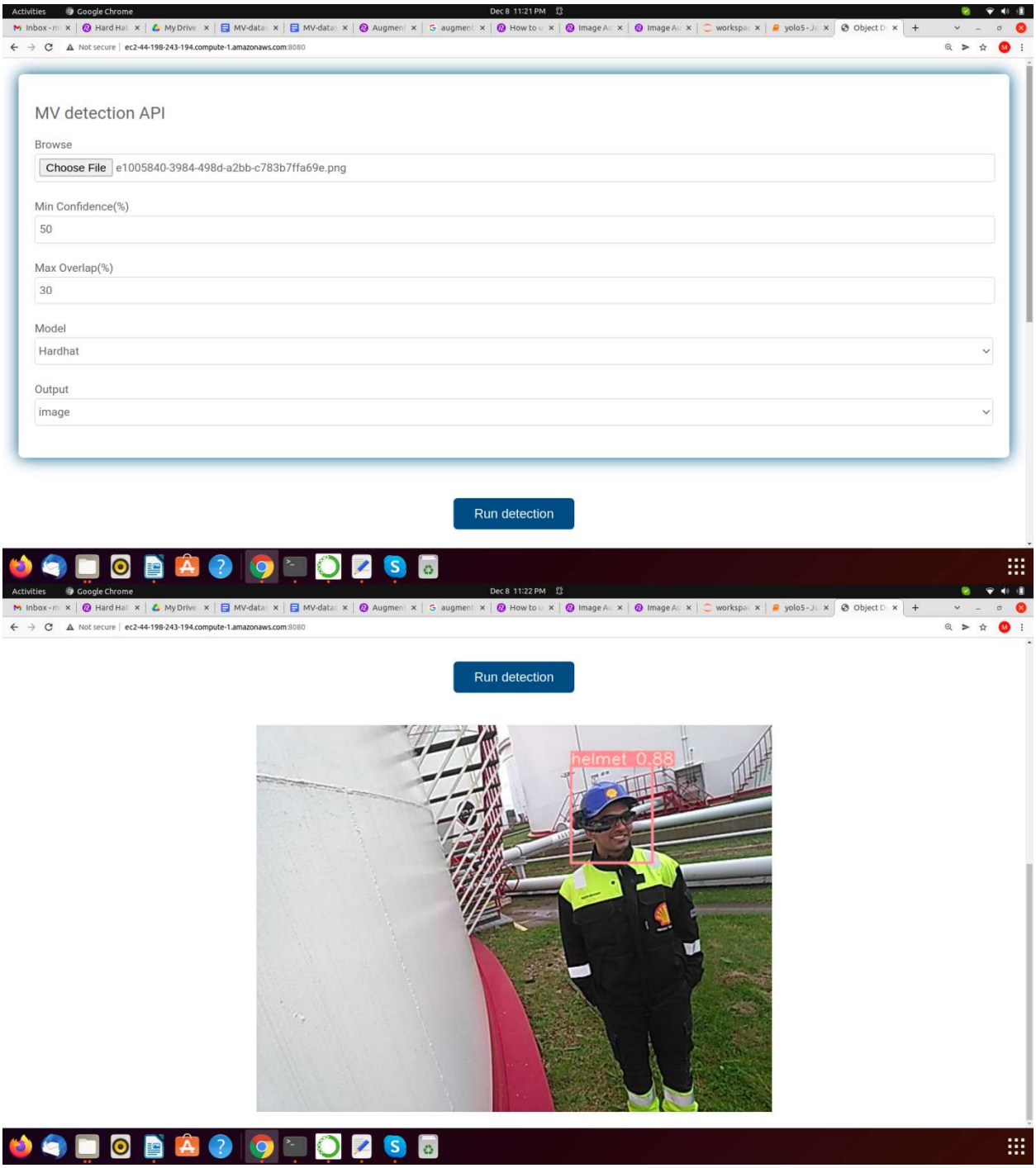


Figure-3.24: Hardhat detection API interface.

3.2.6 User Application

User application server receives the image feed from the different on-site cameras by using api end-point. Then this back-end application calls the API end-point for the detector. It's an automated process. Camera devices and application servers are automatically synchronized after specific time duration that can be configured in the back-end application. Back-end application records the site name, location and camera device information. Images can be filtered by the device name, site location, and object name. User dashboard is given as follows.

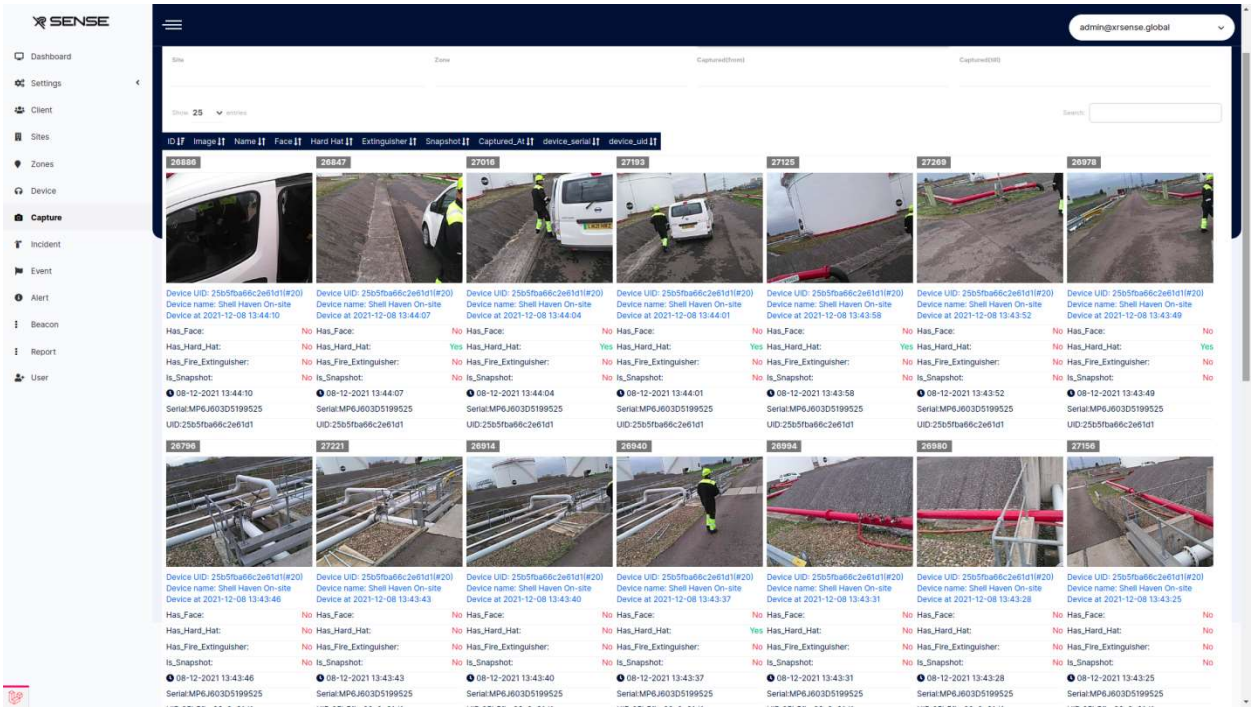


Figure-3.25: User application dashboard.

3.3 Technologies used

The following tools and technologies were used in the several parts of the project.

1. YOLOv5: Python>=3.6.0, PyTorch>=1.7
2. API end-point: Python>=3.6.0 and Flask python library
3. User Dashboard: PHP, Laravel, MySQL
4. Cloud Platform: AWS, Roboflow
5. Annotation tool: Roboflow
6. Augmentation tool: Roboflow
7. Python notebook: Jupyter notebook.
8. Repository: Github

3.4 Conclusion

The proposed system was discussed in detail in the above sections. Datasets were collected from internal and external sources in this project. During the project development time, datasets were updated from different angles and trained the models. The training logs were recorded and the QA process also recorded to benchmark the models. It's a continuous process to improve the dataset as well as the model performance. Moreover, the system diagram represents the way to develop computer vision models to detect any objects. However, it depends on the dataset. Improving the dataset means improving the model performance.

4. Chapter Four: Research findings

4.1 Training Datasets

Many versions of the dataset were generated from the data preparation pipeline. However, two stable datasets were elaborated in this section. They are given as follows.

1. Base dataset (dataset-v01): The base dataset was collected from public sources like roboflow and open images. Total number of training images is 9541. No augmented or extra background images were added into this dataset.
2. Augmented dataset (dataset-v02): The base dataset was enhanced by using the augmentation features, active learning and background images. 10% training images were added from the on-site to improve the detection performance. 5% background images without the objects were added to this dataset to handle the false positive (FP) detections. The total number of images is 31578.

4.2 Evaluation metrics

The performance of an object detection model can be evaluated from many measurement criteria. Intersection over Union (IoU) Precision, Recall, Average Precision (AP), and Mean Average Precision (mAP) are the most used criteria that are given as examples (CENGİL and ÇINAR, 2020).

Intersection over Union (IoU) is a measurement that computes the difference between the bounding box for the ground truth and the predicted bounding boxes. This metric is used in the state of the art object detection algorithms. In object detection, the model detects multiple annotated boxes for each object and removes unnecessary boxes depending on the threshold value based on the confidence scores of each annotated box. A threshold value is determined according to requirements of the use cases. If the IoU value is greater than the threshold value, it is taken as an object, otherwise, the box is removed (Rezatofighi et al., 2019). IoU is calculated as given in equation (1).

$$\text{IoU} = \frac{Area\ of\ union}{Area\ of\ intersection} \qquad (1)$$

$mAP_{0.5}$ is the mean value of $AP_{0.5}$ (average precision) of the detection results for all classes. $AP_{0.5}$ value is calculated from the enclosed area of the accuracy and recall curves for an IoU (Intersection over Union) threshold of 0.5. Equations (2) and (3) represent the calculation of precision and recall values. Equations (4) and (5) represent the calculation of $AP_{0.5}$ and $mAP_{0.5}$.

$$precision = \frac{TP}{TP+FP} \qquad (2)$$

$$recall = \frac{TP}{TP+FN} \quad (3)$$

True Positives (TP) shows the number of objects actually detected in the dataset. False Positives (FP) shows the number of objects detected by the detection model in error. False Negatives (FN) shows the number of objects missed by the detection model. The general definition for the $AP_{0.5}$ (Average Precision) is finding the area under the precision-recall curve. Precision and recall are always between 0 and 1.

$$AP_{0.5} = \int_0^1 Pr\, dr, IoU \geq 0.5 \quad (4)$$

$$mAP_{0.5} = \frac{\sum_{\iota=1}^{n} AP0.5\iota}{N} \quad (5)$$

4.3 Test results and Analysis

Many models were trained on the both custom datasets (dataset-v01 and dataset-v02). However two stable models trained on the both dataset were elaborated in this section. Two ways of the model evaluation process were considered. They are given as follows.

1. Training logs analysis
2. Manual QA

4.3.1 Training logs analysis

The summary of both models are given as follows.

Model	Dataset	Dataset features	Number of Training images	epoch	$mAP_{0.5}$
hardhat-v06-28	dataset-v01	Base dataset: Collected from the public sources	9541	100	0.875

hardhat-v06-50	dataset-v02	- Base dataset - Augmented images - On-site images(Active learning) - Background images without objects	31578	100	0.890

Table-4.1: The summary of both models

Both models (hardhat-v06-28, hardhat-v06-50) were trained on the custom datasets (dataset-v01, dataset-v02) by using YOLOv5s weight. In order to compare the magnitude of the performance difference between both models, the metric changes during the training process are printed out together for comparison. The experimental results are shown in Figure 4.1. The gray curve is the improved hardhat-v06-50 and the green curve is hardhat-v06-28. As can be seen in Figure 4.1, the model (hardhat-v06-50) trained on the improved dataset (dataset-v02) achieves better performance in $mAP_{0.5}$ value than the model (hardhat-v06-28) trained on the base dataset (dataset-v01).

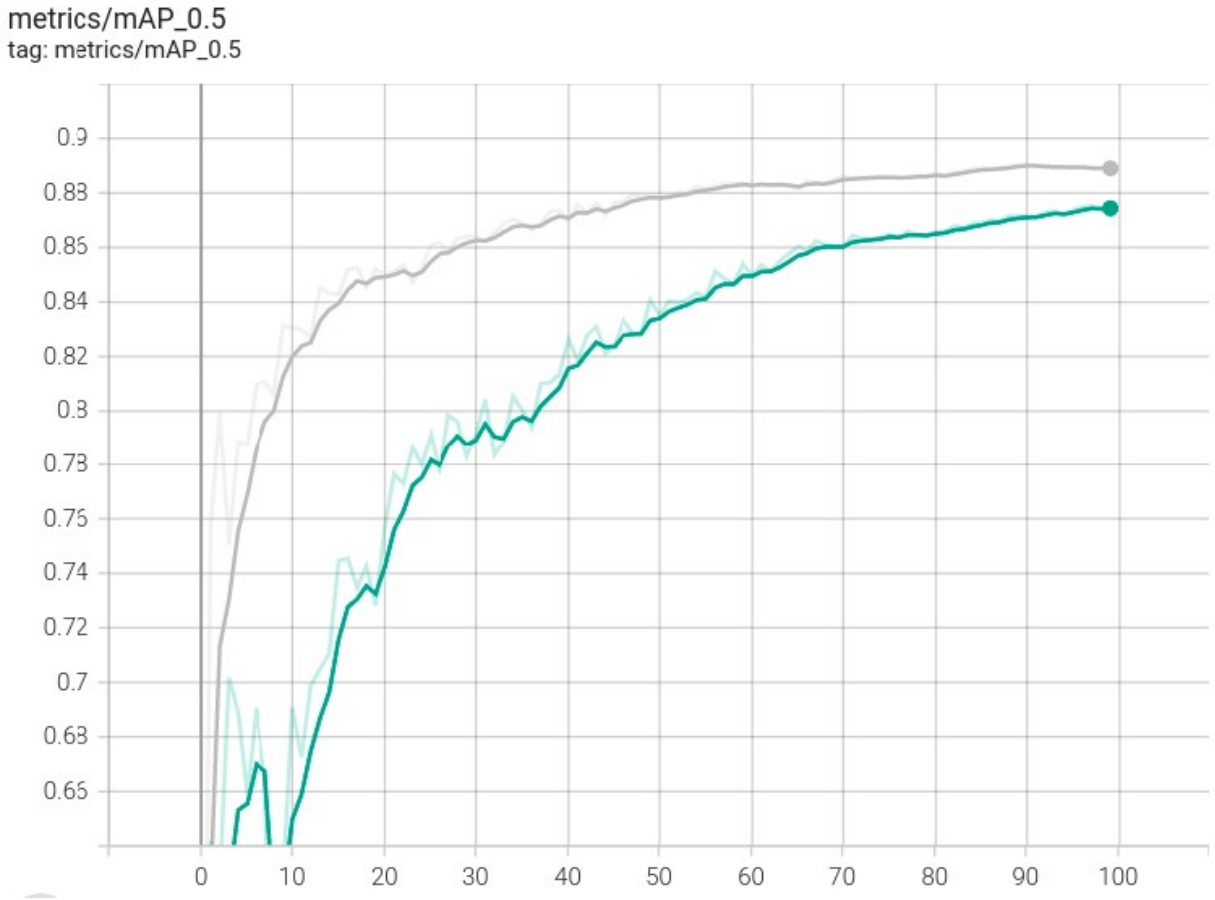


Figure-4.1: $mAP_{0.5}$ curves of both models.

The precision-recall curves obtained from the training of both models are given in figure 4.2. According to the figure, the mean average Precision value for all classes of hardhat-v06-28 model was found to be 0.875. In addition, AP values of head and helmet are given as 0.820 and 0.929. Similarly, the mean average Precision value for all classes of hardhat-v06-50 model was found to be 0.890 and AP values of head and helmet are given as 0.837 and 0.944. According to the figure, hardhat-v06-50 achieves better performance not only in $mAP_{0.5}$ value but also $AP_{0.5}$ values of each class.

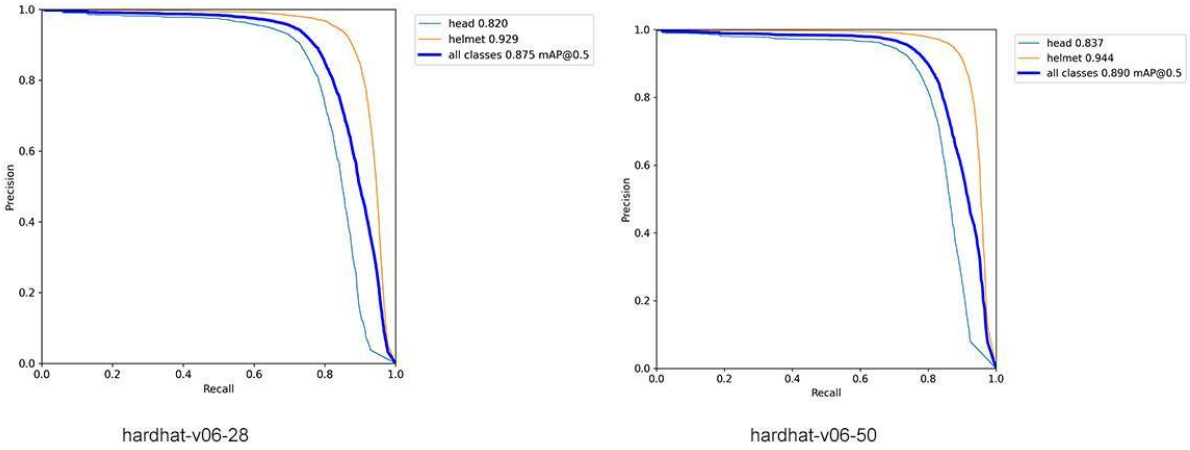


Figure-4.2: Precision-recall curve of both models.

More training data is not always good. Because It may cause overfitting of the model. However quality data matters a lot to improve the model. In real-time detection, the performance depends on the object representation in the training dataset. According to the training logs, the major improvement was not found between both models (hardhat-v06-28, hardhat-v06-50). However, major differences in the performance between both models were found on-site testing (Manual QA). The augmentation technique added new training examples out of existing training data. It's really hard to truly capture an image that handles for every real world scenario a model may encompass. Augmenting the existing training data to generalize to other situations allows the model to learn from a wider array of situations.

4.3.2 Manual QA

As hardhat-v06-50 was trained on the improved dataset (dataset-v02), it achieves the best result rather than hardhat-v06-28 (trained on dataset-v01). The augmentation feature, active learning and background images from on-site increased the model performance by increasing the diversity of learning examples for the model.

Images were collected on-site for the test dataset. 171 images were used to test the model. The crucial criteria considered for the manual testing are small object detection, object detection from different lighting conditions, object detection from a different camera angle, and the reduction of false-positive detection. Table-4.2 shows the success rate of the both models. The success rate was counted based on the minimum confidence 50% and maximum overlap 30% and image size was 640px.

As can be seen in Table-4.2, the model (hardhat-v06-50) trained on the improved dataset (dataset-v02) achieves better performance than the model (hardhat-v06-28) trained on the base dataset (dataset-v01). The improved model (hardhat-v06-50) works around 100% on the test dataset to detect the normal representation of the desired objects. It also works well to

handle or detect the small, blurry or multiple objects. On the other hand, hardhat-v06-28 is underperforming due to the poor object representation in the base dataset.

Number of images and type	**hardhat-v06-28**	**hardhat-v06-50**	**Success Rate**
Image: 50 Type: Blurry, small objects, multiple hardhat and head	Success = 32 Fail = 18	Success = 47 Fail = 3	hardhat-v06-28 = 64% hardhat-v06-50 = 94%
Image: 50 Type: Normal objects, multiple hardhat and head	Success = 44 Fail = 6	Success = 50 Fail = 0	hardhat-v06-28 = 88% hardhat-v06-50 = 100%
Image: 71 Type: Mixture of blurry and normal objects, single/multiple hardhat and head, background images	Success = 57 Fail = 14	Success = 69 Fail = 2	hardhat-v06-28 = 80% hardhat-v06-50 = 97%

Table-4.2: The success rate of both models on test dataset.

4.3.2.1 Test case-1

In the following example, hardhat-v06-28 failed to detect the other two hardhats in dark lighting conditions. On the other hand, hardhat-v06-50 could detect the all desired hardhats.

Figure-4.3: Test case-1 results.

4.3.2.2 Test case-2

In this test result, hardhat-v06-28 was unable to detect the hardhat from the different camera positions. On the other hand, hardhat-v06-50 was able to detect both hardhats. The model(hardhat-v06-50) also shows the high precision of the detection results rather than hardhat-v06-28.

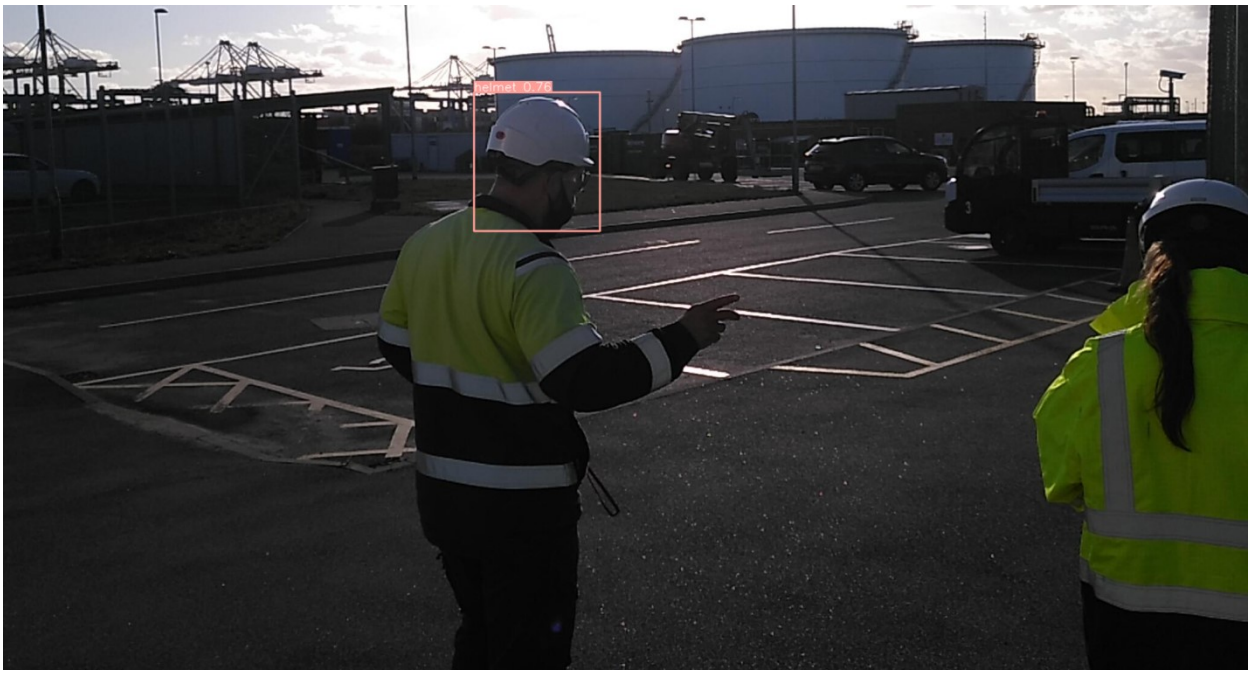


Figure-4.4: Test case-2 results.

4.3.2.3 Test case-3

The following example is a bit complex. hardhat-v06-28 failed to detect all desired hardhats and heads from the image. On the other hand, hardhat-v06-50 could detect 2 hardhats and 3 heads with comparatively high precision.

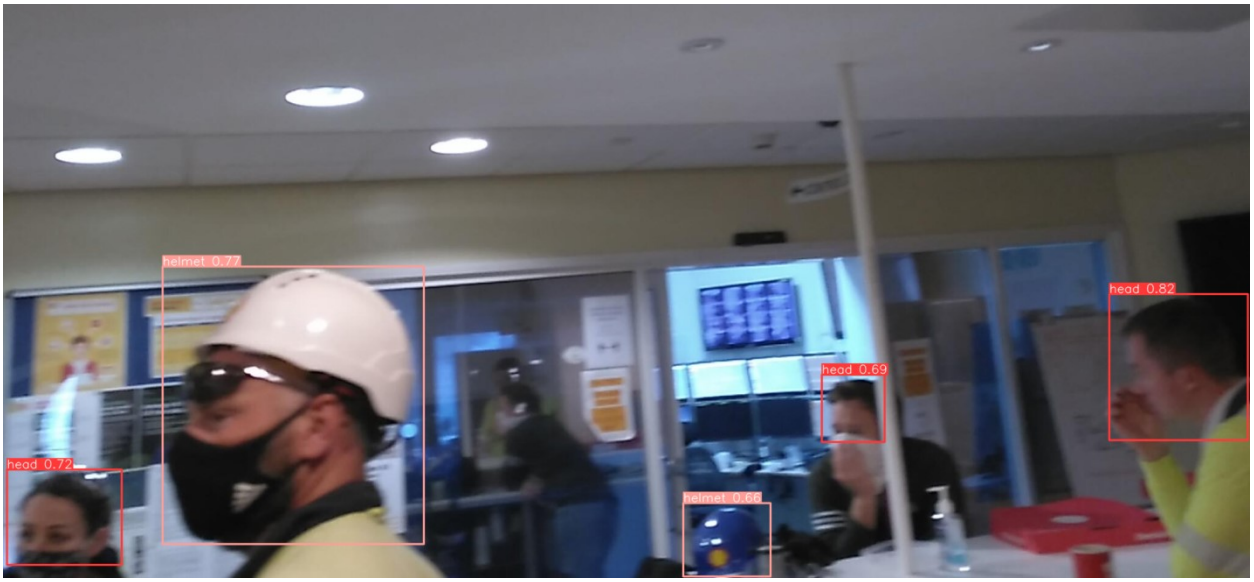


Figure-4.5: Test case-3 results.

4.3.2.4 Test case-4

In the following example, hardhat-v06-28 detected false positives due to the poor representation of hardhats in the base dataset. . On the other hand, hardhat-v06-50 avoided false positive results because of the strong features available for the hardhats in the improved dataset.

Figure-4.6: Test case-4 results.

4.3.2.5 Test case-5

In the following group image, hardhat-v06-28 failed to detect the maximum desired heads available in the image and even it detected false heads as well. On the other hand, hardhat-v06-50 worked better to detect the maximum desired heads available in the image.

Figure-4.7: Test case-5 results.

4.4 Conclusion

This section has clearly outlined and evidenced the performance between base model and improved model from the context of training log analysis and practical QA process. The next chapter will go on to discuss and analyse these results in the context of each research question.

5. Chapter Five: Discussion & Analysis

5.1 Addressing the research questions

5.1.1 How to prepare a well balanced training dataset for hardhat detection?

A well-balanced dataset is a key to success in the object detection project. 80% of efforts (Press, 2016) are involved in the data preparation. For this reason, the major focus of this research was to prepare an improved training dataset with a diverse representation of the desired objects. The base dataset was collected from public sources. Figure-3.4 shows how the base dataset is being improved according to the data collection pipeline. Active learning is continuously enriching the features of the desired objects. Image augmentation features help to improve the variation of the object representation.

Although the dataset improvement is a continuous process, Table-4.1 represents the experimental results of the improved model trained on the enhanced dataset. The model achieved better mAP (Mean Average Precision) rather than the base model.

As can be seen in Table-4.2, according to the manual QA process, the improved model achieved very good results rather than the base model. Because the improved model was trained on the dataset which had rich features and the diverse representation of the hardhat and human head.

According to the evidence discussed in section 2.5, no researchers provided a way of continuous training dataset improvement that can enhance the model performance over time. Table-5.1 shows the dataset improvement of the proposed model and other related methods discussed in section 2.5.

Method	Data sources	Improvement Type
Bhadeshiya, Brahmbhatt and Pitroda(2021)	Public dataset	No improvement found
Casuat et al.(2020)	Public dataset, On-site images	Static improvement
Zhang et al.(2021)	Public dataset.	No improvement found
Proposed	Public dataset, On-site images and augmented images.	Continuous improvement

Table-5.1: Training dataset Improvement.

This research provided a way of continuous improvement of the training dataset so that the model can perform in real-time hardhat detection for the safety of construction workers.

5.1.2 How image augmentation features can enhance the dataset as well as the model performance to detect the diverse instances of the hardhats?

To increase the generalizability of the model performance by increasing the diversity of learning features from the dataset, this study outlined the image augmentation features in section 3.2.1.4. The success to detect the objects in real-time construction use cases depends on the variety of the desired objects representation in the dataset.

In this research, various augmentation features such as rotation, exposure, shear, Blur, noise, and mosaic were used to enhance the learning features in the dataset. As image augmentation is part of the data collection pipeline, other augmentation features can be used to check the model performance in the real-time hardhat detection use cases.

As can be seen in section 4.3.2.1 to 4.3.2.5 show the performance of the improved model trained on the augmented dataset. The model was able to detect the hardhats and heads from the diverse instances available in an image.

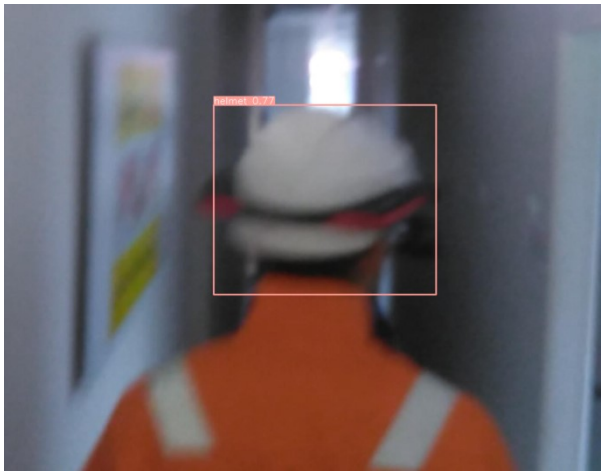
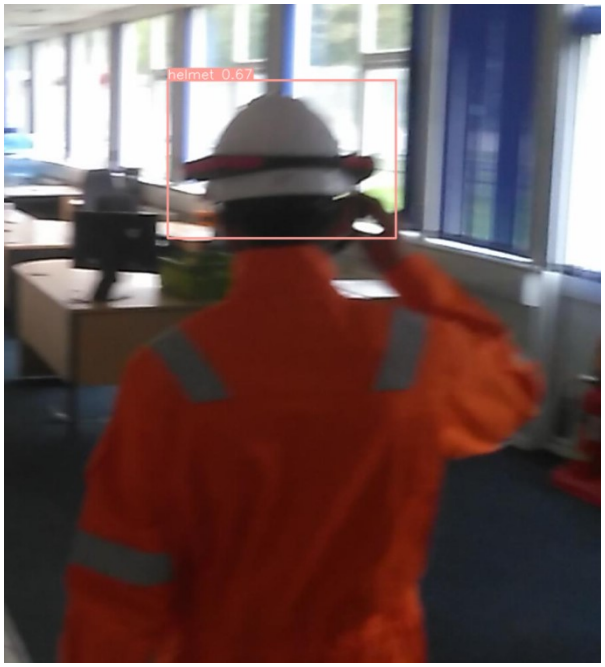
No related methods outlined in section 2.5 used the augmented images. All those methods can struggle to detect the diverse hardhats in real-time construction production.

This study outlined how augmentation can continue to boost the model performance to achieve the goal of a computer vision project.

5.1.3 How on-site learning examples (Active Learning) can enhance the model performance?

The strategy of using active learning is to improve the training dataset from the on-site learning examples. It continuously enriches the features of the on-site desired objects. As can be seen in Figure-3.4, hardhat or heads detected in the images from the on-site captures with the 30-40% confidence are automatically collected for the further training dataset. It not only enhances the dataset but also detects the desired objects with high confidence. The improved dataset helps the model learn from the available on-site object features. Some testing examples are given as follows.

The model trained on base dataset (hardhat-v06-28)	The model trained on improved dataset (hardhat-v06-50)

 Confidence = 55%	 Confidence = 77%
 Confidence = 54%	 Confidence = 67%

	
Confidence = 63%	Confidence = 85%

Table-5.2: Testing results after adding on-site images.

As can be seen in the above Table-5.1, the improved model can detect hardhat with high confidence because of the rich features gained from the enhanced dataset.

Except for Casuat et al.(2020) method, other related methods didn't use the on-site learning examples to enhance the model performance. In Table-5.2, evidence was provided on how on-site images improve detection accuracy.

5.1.4 How to reduce the False Positive (FP) detection on-site?

In object detection, except for the area of the objects, the other portion of the images work as negative examples. In reality, some background features might be similar to the desired object features. In that cases, background image features can help the model enhance the negative learning examples. As stated in section 3.2.1.6, 5% of background images from the on-site captures are being collected for the further training dataset. It improves the dataset with negative examples. According to this experiment, in-general background images don't improve significantly in avoiding false-positive detection. However, the combination of on-site learning(Active learning) and on-site background images improve the model performance potentially reducing the false positive detection. Because on-site learning examples enrich positive features of the desired objects, while on-site background images enhance the negative features. Some pieces of evidence of the test results are given in section 4.3.2.4 and 4.3.2.5.

No related methods discussed in section 2.5 addressed how to reduce the false positive detection. On the other hand, the proposed method added 5% negative examples to increase the negative features of the hardhat and head.

5.2 Conclusion

This chapter has discussed the test results, analyzed the impact of those results on this study, and outlined the potential summary which can be drawn from this study. The analysis has been accomplished in the context of each research question and the drawbacks of the related studies discussed in section 2.5. This chapter also discussed the considered answers to the questions this research aimed to address.

6. Chapter Six: Conclusion

The computer vision-based reform of construction site surveillance is an upcoming wave. The machine vision-based surveillance management of construction workers is one of the most important parts of this reform. The hardhat is also a crucial protective gear to save the workers from the chance of brain damage in production work.

Sought to detect hardhat-wearing for the safety of construction workers, the research objectives set out in section 1.1.4 have been achieved.

This research has discussed how to continuously improve the training dataset by including the combination of image augmentation features and on-site images. The proposed system delivered not only to detect hardhat wearing but also provided a proof of concept to be successful in any machine vision-based object detection.

This study also analyzed the testing results in terms of addressing the research questions stated in section 1.1.2. and the drawbacks of the related studies discussed in section 2.5.

Model YOLOv5 was used to train on the enhanced custom dataset. The improved resultant model achieved 0.890 mAP according to the training logs and achieved a 97% success rate in the manual QA process stated in Table-4.2.

6.1 Research limitations

This study has some limitations for several reasons.

6.1.1 Training server capacity

Deep learning-based object detection needs high computational power for training image datasets. However, a smaller and cost-effective AWS GPU instance g4dn.xlarge was used for training the models. Many models were trained for a maximum of 100 epochs. Training model for many epochs is expensive and time-consuming. In that case, needed a high-power GPU instance. To optimize the model performance, the model needs to train for many epochs.

6.1.2 Time constraint

Most time of this research was spent on the data preparation, training, and testing of the models. The basic user dashboard and API end-point were developed for inferencing the detection results. However, many functionality and features can be added to extend the user application for commercial use in construction production. The development of the whole system stated in Figure-3.4 needs time and the cross-functional team to finish the project for commercial use in the computer vision-based surveillance management.

6.1.3 Limited testing in construction site

Due to limited permission to get access to the actual construction site, most of the testing models were performed on the collected test images. A laptop webcam was used to test the

model as well. The improved model still needs to test in actual construction production for optimizing the model performance for commercial use.

6.1.4 Model overfitting

If the proposed system continuously collects the images from on-site cameras to enrich the training dataset, it might overfit the model due to the continuous collection of similar images with the same objects from the same location. Manual validation was applied to collect the unique images to enhance the training dataset. However, an image filtering algorithm can be used to filter unique images for making it an automated process.

6.2 Future research

This study identified several potential extensions of this project for further research.

The proposed system can detect a person without hardhat or with hardhat. The administrator can get an in-general notification if someone is not wearing a hardhat from a specific camera location. However, identifying specific individuals who are not wearing a hardhat on-site will allow the admin to get the specific notification on exactly who is not wearing a hardhat on-site. Adding face recognition with object detection can help to extend this project to a new height.

This system can be extended to detect other protective gear such as safety goggles, safety vests, safety gloves, and safety boots, etc.

Finally, an advanced extension could be, the device will speak with the workers before entering the restricted area (e.g. hardhat area) based on the detection intelligence of the camera vision. If a worker doesn't wear any of the protective gears, the device will ask to wear the missing protective gears. Once a worker wears all required protective equipment, then the device will automatically open the gate for the worker's entrance. On-site cameras will supervise the workers inside the restricted areas as well.

6.3 Contribution and Implications of this Research

Research findings should help future researchers to develop more intelligent object detection solutions. The system design in Figure-3.4 provided a complete architecture of the computer vision-based project.

The success of the machine vision project depends on the object's representation in the dataset. This study should help future researchers how to prepare a potential dataset with a diverse representation of the desired objects.

This research also discussed how to enhance the positive and negative learning features for object detection with high precision.

Ultimately, this system was proposed to detect hardhat for the safety of the construction workers. It should help the worker surveillance management protect workers from fatal injuries or brain damage on-site.

7. References

Amazon (no date) *Amazon EC2 G4 Instances — Amazon Web Services (AWS)*. Available from: https://aws.amazon.com/ec2/instance-types/g4/ [Accessed 6 December 2021].

Bhadeshiya, R.N., Brahmbhatt, K.N. and Pitroda, J.R. (2021) Hard-hat Detection using YOLOv4. *2021 Second International Conference on Electronics and Sustainable Communication Systems (ICESC)*. doi:10.1109/icesc51422.2021.9532896 [Accessed 8 November 2021].

Bharath K (2021) *Object Detection Algorithms and Libraries*. Available from: https://neptune.ai/blog/object-detection-algorithms-and-libraries [Accessed 2 January 2022].

Boesch, G. (2021) *Object Detection in 2021: The Definitive Guide*. Available from: https://viso.ai/deep-learning/object-detection/ [Accessed 2 January 2022].

Casuat, C.D., Merencilla, N.E., Reyes, R.C., Sevilla, R.V. and Pascion, C.G. (2020) Deep-Hart: An Inference Deep Learning Approach of Hard Hat Detection for Work Safety and Surveillance. *2020 IEEE 7th International Conference on Engineering Technologies and Applied Sciences (ICETAS)*. doi:10.1109/icetas51660.2020.9484208 [Accessed 8 November 2021].

CENGİL, E. and ÇINAR, A. (2020) Göğüs Verileri Metrikleri Üzerinden Kanser Sınıflandırılması. *DÜMF Mühendislik Dergisi*. 11 (2), pp. 513–519. doi:10.24012/dumf.578606.

Chahal, K.S. and Dey, K. (2018) A Survey of Modern Object Detection Literature using Deep Learning. *arXiv:1808.07256 [cs]* [online]. Available from: https://arxiv.org/abs/1808.07256 [Accessed 3 January 2022].

Choudhury, A. (2020) *Top 8 Algorithms For Object Detection One Must Know*. Available from: https://analyticsindiamag.com/top-8-algorithms-for-object-detection/ [Accessed 2 January 2022].

Enriquez, K. (2018) *Faster face detection using Convolutional Neural Networks & the Viola-Jones algorithm* [online]. Available from: https://www.csustan.edu/sites/default/files/groups/University%20Honors%20Program/Journals/01_enriquez.pdf [Accessed 15 April 2021].

Girshick, R. (2015) *Fast R-CNN*. Available from: https://arxiv.org/abs/1504.08083 [Accessed 3 January 2022].

GOV.UK (2018) *Construction Sector Deal*. Available from: https://www.gov.uk/government/publications/construction-sector-deal/construction-sector-deal.

Horak, K. and Sablatnig, R. (2019) Deep learning concepts and datasets for image recognition: overview 2019 Xudong Jiang and Jenq-Neng Hwang (eds.). *Eleventh International Conference on Digital Image Processing (ICDIP 2019)*. doi:10.1117/12.2539806 [Accessed 2 January 2022].

HSE (2020) *Health and Safety Executive* [online]. Available from: https://www.hse.gov.uk/statistics/pdf/fatalinjuries.pdf.

Hui, J. (2019) *Object detection: speed and accuracy comparison (Faster R-CNN, R-FCN, SSD, FPN, RetinaNet and…*. Available from: https://jonathan-hui.medium.com/object-detection-speed-and-accuracy-comparison-faster-r-cnn-r-fcn-ssd-and-yolo-5425656ae359 [Accessed 3 January 2022].

IOGP (no date) *Life-Saving Rules*. Available from: https://www.iogp.org/life-savingrules/ [Accessed 24 December 2021].

Kadhm, M.S. (2017) *(PDF) Arabic Handwritten Text Recognition and Writer Identification*. Available from: https://www.researchgate.net/publication/329151860_Arabic_Handwritten_Text_Recognition_and_Writer_Identification [Accessed 3 January 2022].

Kathuria, A. (2021) *How to Train YOLO v5 on a Custom Dataset*. Available from: https://blog.paperspace.com/train-yolov5-custom-data/.

Khandelwal, R. (2019) *SSD : Single Shot Detector for object detection using MultiBox*. Available from: https://towardsdatascience.com/ssd-single-shot-detector-for-object-detection-using-multibox-1818603644ca [Accessed 2 January 2022].

Moore, S.K., Schneider, D. and Strickland, E. (2021) *How Deep Learning Works*. Available from: https://spectrum.ieee.org/what-is-deep-learning [Accessed 2 January 2022].

Nagalakshmi, G. and Jyothi, S. (2015) *Image Acquisition, Noise removal, Edge Detection Methods in Image Processing Using Matlab for Prawn Species Identification*. Available from: https://www.semanticscholar.org/paper/Image-Acquisition%2C-Noise-removal%2C-Edge-Detection-in-Nagalakshmi-Jyothi/9d3f0f0a59d202f78aeb6cd51b45d08ba297540f [Accessed 2 January 2022].

Open Images Dataset V6 (2021) *Open Images V6 - Download*. Available from: https://storage.googleapis.com/openimages/web/download.html [Accessed 9 December 2021].

Press, G. (2016) *Cleaning Big Data: Most Time-Consuming, Least Enjoyable Data Science Task, Survey Says*. Available from: https://www.forbes.com/sites/gilpress/2016/03/23/data-preparation-most-time-consuming-least-enjoyable-data-science-task-survey-says/?sh=603d220f6f63 [Accessed 7 November 2021].

Pujara, A. (2020) *Image Classification With MobileNet*. Available from: https://medium.com/analytics-vidhya/image-classification-with-mobilenet-cc6fbb2cd470 [Accessed 3 January 2022].

PyTorch (no date) *PyTorch*. Available from: https://pytorch.org/hub/ultralytics_yolov5/ [Accessed 3 December 2021].

Redmon, J., Divvala, S., Girshick, R. and Farhadi, A. (2016) You Only Look Once: Unified, Real-Time Object Detection. *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)* [online]. Available from: https://arxiv.org/pdf/1506.02640.pdfdoi:10.1109/cvpr.2016.91 [Accessed 2 January 2022].

Redmon, J. and Farhadi, A. (2018) *YOLOv3: An Incremental Improvement*. Available from: https://arxiv.org/abs/1804.02767.

Rezatofighi, H., Tsoi, N., Gwak, J., Sadeghian, A., Reid, I. and Savarese, S. (2019) Generalized Intersection Over Union: A Metric and a Loss for Bounding Box Regression. *2019 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*. doi:10.1109/cvpr.2019.00075.

Roboflow annotate (2021) *Annotate - Roboflow*. Available from: https://docs.roboflow.com/annotate [Accessed 9 December 2021].

Roboflow augmentation (2021) *Image Augmentation - Roboflow*. Available from: https://docs.roboflow.com/image-transformations/image-augmentation [Accessed 9 December 2021].

Roboflow dataset (2021) *Hard Hat Workers Object Detection Dataset*. Available from: https://public.roboflow.com/object-detection/hard-hat-workers [Accessed 8 November 2021].

Solawetz, J. (2020) *YOLOv5 New Version - Improvements And Evaluation*. Available from: https://blog.roboflow.com/yolov5-improvements-and-evaluation/ [Accessed 5 December 2021].

U.S. Bureau of Labor Statistics (2021) *Industries at a Glance: Construction: NAICS 23*. Available from: https://www.bls.gov/iag/tgs/iag23.htm.

U.S. Bureau of Labor Statistics (2020) *Number and rate of fatal work injuries, by industry sector*. Available from: https://www.bls.gov/charts/census-of-fatal-occupational-injuries/number-and-rate-of-fatal-work-injuries-by-industry.htm.

US Census Bureau Construction Expenditures (2021) *US Census Bureau Construction Spending Survey*. Available from: https://www.census.gov/construction/c30/prpdf.html.

Villanueva, A., Benemerito, R.L.L., Cabug-Os, M.J.M., Chua, R.B., Rebeca, C.K.D.C. and Miranda, M. (2019) Somnolence Detection System Utilizing Deep Neural Network. *2019 International Conference on Information and Communications Technology (ICOIACT)*. doi:10.1109/icoiact46704.2019.8938460.

Wang, C.-Y., Mark Liao, H.-Y., Wu, Y.-H., Chen, P.-Y., Hsieh, J.-W. and Yeh, I-Hau. (2020) CSPNet: A New Backbone that can Enhance Learning Capability of CNN. *2020 IEEE/CVF Conference on Computer Vision and Pattern Recognition Workshops (CVPRW)*. doi:10.1109/cvprw50498.2020.00203.

Wang, K., Liew, J.H., Zou, Y., Zhou, D. and Feng, J. (2019) PANet: Few-Shot Image Semantic Segmentation With Prototype Alignment. *2019 IEEE/CVF International Conference on Computer Vision (ICCV)*. doi:10.1109/iccv.2019.00929.

Xu, R., Lin, H., Lu, K., Cao, L. and Liu, Y. (2021) A Forest Fire Detection System Based on Ensemble Learning. *Forests*. 12 (2), pp. 217. doi:10.3390/f12020217 [Accessed 17 May 2021].

YOLOv5 Documentation (no date) *YOLOv5 Documentation*. Available from: https://docs.ultralytics.com/ [Accessed 2 December 2021].

Zhang, C., Tian, Z., Song, J., Zheng, Y. and Xu, B. (2021) Construction worker hardhat-wearing detection based on an improved BiFPN. *2020 25th International Conference on Pattern Recognition (ICPR)*. doi:10.1109/icpr48806.2021.9412103 [Accessed 5 November 2021].

APPENDIX A: List of Abbreviations

Abbreviation	Meaning
AP	Average Precision
CNN	Convolutional Neural Network
COCO	Common Objects In Context
CSPNet	Cross stage partial network
FLOPS	Floating-point operations per second
FN	False Negative
FP	False Positive
HOG	Histogram of Oriented Gradients
IoU	Intersection over Union
mAP	Mean Average Precision
PPE	Personal Protective Equipment
R-CNN	Region-based Convolutional Neural Network
ROI	Region of Interest
SSD	Single Shot Detector
SSP	Spatial Pyramid Pooling
TP	True Positive
YOLO	You Only Look Once

APPENDIX B: Necessary project source code

YOLOv5 model open-source repository:

https://github.com/ultralytics/yolov5

Model training script:

Start training on the custom dataset.

```
python train.py --img 416 --batch 150 --epochs 500 --save-period 20 --data
            /home/gias/workspace/dataset/base_dataset_v02/data.yaml --weights yolov5s.pt --
name hardhat-local-v0
```

Inference script:

Run the detection results on the test dataset.

```
python detect.py --weights ./md/hardhat-v06-50/weights/best.pt --img 640 --conf 0.50 --iou
0.30 --source ./data/images/batch3 --line-thickness 2 --augment
```

Data augmentation:

```
import os
import glob
import pandas as pd


import imageio
import re
import cv2
import numpy as np
import imgaug as ia
import imgaug.augmenters as iaa
from imgaug.augmentables.bbs import BoundingBox, BoundingBoxesOnImage
```

```
%matplotlib inline
ia.seed(1)

input_path = 'roboflow//test/images/'
output_path = 'roboflow/test/aug-images/'

df = pd.read_csv('roboflow_test_annotations.csv')

filename_group = df.groupby("filename")

i = 0
# create data frame which we're going to populate with augmented image info
aug_bbs_xy = pd.DataFrame(columns=['filename','width','height','class',
                                   'xmin', 'ymin', 'xmax', 'ymax'])
for name, groups in filename_group:

    file_name = input_path + name
    image = imageio.imread(file_name)

    boxes = []

    for key, obj in groups.iterrows() :
        # find the bounding box coordinates
        xmin = int (obj['xmin'])
        ymin = int (obj['ymin'])
        xmax = int (obj['xmax'])
        ymax = int (obj['ymax'])
        label = obj['class']

        boxes.append(ia.BoundingBox(x1=xmin, y1=ymin, x2=xmax, y2=ymax,
label=label))
```

```
bbs = BoundingBoxesOnImage(boxes, shape=image.shape)



for x in range(4):

    seq = iaa.Sequential([

        iaa.Fliplr(0.5), # horizontal flips
        iaa.Crop(percent=(0, 0.1)), # random crops
        iaa.MultiplyBrightness((0.5, 1.5)),
        iaa.Resize((0.5, 1.0)),
        iaa.Affine(
            scale={"x": (0.8, 1.2), "y": (0.8, 1.2)},
            translate_percent={"x": (-0.2, 0.2), "y": (-0.2, 0.2)},
            rotate=(-15, 15)
            #shear=(-8, 8)
        )


    ], random_order=True)


    image_aug, bbs_aug = seq(image=image, bounding_boxes=bbs)

    # disregard bounding boxes which have fallen out of image pane
    bbs_aug = bbs_aug.remove_out_of_image()
    # clip bounding boxes which are partially outside of image pane
```

```
        bbs_aug = bbs_aug.clip_out_of_image()



        #don't perform any actions with the image if there are no bounding
boxes left in it
        if re.findall('Image...', str(bbs_aug)) == ['Image([]']:
            pass


        else:
            # write augmented image to a file
            imageio.imwrite(output_path + "aug_" + str(x) + "_" + name,
image_aug)


            for bbs_box in bbs_aug:


                rows = []
                rows.append(["aug_" + str(x) + "_" + name,
image_aug.shape[1], image_aug.shape[0],
                                    bbs_box.label, bbs_box.x1, bbs_box.y1,
bbs_box.x2, bbs_box.y2])


                aug_bbs_xy_new = pd.DataFrame(rows,
columns=['filename','width','height','class', 'xmin',
                                                                'ymin', 'xmax',
'ymax'])
                aug_bbs_xy = pd.concat([aug_bbs_xy, aug_bbs_xy_new])




aug_bbs_xy.to_csv(output_path + 'aug-labels.csv', index=False)
```

```
print("************** Augmentation is Done ***************")
```

Convert CSV to YOLOv5 format:

```
import os
import glob
import pandas as pd
import imageio
import re
import cv2
import numpy as np
import imgaug as ia
import imgaug.augmenters as iaa
from imgaug.augmentables.bbs import BoundingBox, BoundingBoxesOnImage
input_path = 'data/input/openimage-helmet/helmet/'
output_path = 'data/input/openimage-helmet/helmet/'

classes = ['head', 'helmet']

df = pd.read_csv(input_path + 'labels_helmet_train.csv')

filename_group = df.groupby("filename")


for name, groups in filename_group:

    file_name = input_path + name

    basename = os.path.basename(file_name)
```

```
    basename_no_ext = os.path.splitext(basename)[0]


    image = imageio.imread(file_name)


    w = int(image.shape[1])

    h = int(image.shape[0])


    out_file = open(output_path + basename_no_ext + '.txt', 'w')



    for key, obj in groups.iterrows() :


        cls = obj['class']

        if cls not in classes:

            continue

        cls_id = classes.index(cls)


        b = (float(obj['xmin']), float(obj['xmax']), float(obj['ymin']),
float(obj['ymax']))

        bb = convert((w,h), b)

        out_file.write(str(cls_id) + " " + " ".join([str(a) for a in bb]) +
'\n')


print("Done.........Done.....")
```

Flask API function:

```
from re import DEBUG, sub

from flask import Flask, render_template, request, redirect, send_file,
url_for

from werkzeug.utils import secure_filename, send_from_directory

import os
```

```
import subprocess

app = Flask(__name__)

uploads_dir = os.path.join(app.instance_path, 'uploads')

os.makedirs(uploads_dir, exist_ok=True)

@app.route("/")
def hello_world():
    return render_template('mv-api.html')


@app.route("/detect", methods=['POST'])
def detect():
    if not request.method == "POST":
        return
    video = request.files['video']
    conf = request.form['conf']
    ovl = request.form['ovl']
    model = request.form['model']
    output = request.form['output']

    if model == 'HH':
        model = 'HH.pt'
    else:
        model = 'FE.pt'

    video.save(os.path.join(uploads_dir, secure_filename(video.filename)))
    print(video)
```

```
    subprocess.run("ls")

    subprocess.run(['python3', 'detect.py', '--source',
os.path.join(uploads_dir, secure_filename(video.filename)), '--conf-thres',
conf, '--iou-thres', ovl, '--weights', model, '--augment'])



    # return os.path.join(uploads_dir, secure_filename(video.filename))

    obj = secure_filename(video.filename)

    return obj


@app.route('/return-files', methods=['GET'])

def return_file():

    obj = request.args.get('obj')

    loc = os.path.join("runs/detect", obj)

    print(loc)

    try:

        return send_file(os.path.join("runs/detect", obj),
attachment_filename=obj)

        # return send_from_directory(loc, obj)

    except Exception as e:

        return str(e)


# @app.route('/display/<filename>')

# def display_video(filename):

#   #print('display_video filename: ' + filename)

#   return redirect(url_for('static/video_1.mp4', code=200))
```

APPENDIX C: Test Datasets and results

Test dataset	https://drive.google.com/drive/folders/1gUdVXa86jJdMByuBu-NEuCsswougvnbU?usp=sharing
Base model results on the test dataset.	https://drive.google.com/drive/folders/1QrGRfsDSllJdFIYqwJBFrfB1W1E4IGTO?usp=sharing
Improved model results on the test dataset.	https://drive.google.com/drive/folders/1MYgFGoEZfSvHL1V1J8YNfReK0IABx5na?usp=sharing