Griffith School of Engineering Griffith University

7510ENG – Image Processing and Machine Vision

Real Time Traffic Sign Recognition

(GTRSB Dataset)

Waleed Umer : s5336317

Lab Instructor: Mr.Joel Dedini

Course Convenor: Dr Andrew Busch

TABLE OF CONTENTS

1 INTRODUCTION S

Real-Time Traffic Sign Recognition (RTSR) is an active area of research focusing on the problem of how to automatically detect traffic signs in real-time from the visual data [1]. Infotrafsignup is a subset of machine vision and pattern recognition designed for the automatic detection and recognition of traffic signs with a view of assisting ITS [2]. It is impossible to code the systems in order to replicate traffic signs; this is why the methods for deep learning, especially CNNs, can be used to compute the traffics rapidly and with high accuracy [3]. This does not only bring better security to self-driving cars but also optimizes the intelligential traffic control systems. Real-time decision making is central to the operation of self-driving cars, and a viable manner by which traffic signs are identified lies in their reliable detection [4].

This work is aimed at training the real-time traffic sign recognition model on the GTSRB: German Traffic Sign Recognition Benchmark dataset [5] and by utilizing YOLOv5 object detection algorithm [6]. Unified Real-Time Object Detection, YOLOv5 is one of the fastest and most accurate detectors designed specifically for vehicles' traffic sign detection in real-time. The GTSRB dataset gives an overall list of traffic sign images which is used to train machine learning algorithms. In this project, best practices of training and evaluating existing and novel neural network-based deep learning architectures such as YOLOv5 will be applied and they will be evaluated using standard benchmarking indices like accuracy and precision rates. To summarize, even for the model to be considered effective, it has to perform at a rate of at least 95% for all classes of the traffic signs which are safety standard in self-driving car.

1.1 Project scope

It is impossible to identify all the traffic signs possible and the variations of each one within the scope of this work. In fact, the German Traffic Sign Recognition Benchmark (GTSRB) dataset utilized for this project is comprised of more than 50,000 images and includes 43 distinct traffic sign classes [5]. This dataset mostly concentrates on the classification of signs for the regulation seen often in traffic areas; warning, and prohibitory signs. There are other large datasets available for traffic sign recognition, including some of greater size or with other region variations, however, for the sake of this project the focus must therefore remain

on the GTSRB due to its bench marking frequency in traffic sign detection systems.

To identify traffic signs in real-time, the YOLOv5 object detection model will be used in this research. YOLOv5's approach to predicting the bounding boxes and class probabilities at once aligns it perfectly to this task. This project will address two core aspects of the traffic sign recognition pipeline:

1. **Image preprocessing**: Ensuring the dataset images are in the correct format and resolution for efficient processing.

2. **Traffic sign detection and classification**: Using YOLOv5 to detect traffic signs and classify them into their respective categories.

Difficulty or impossibility to recognize all the types of traffic signs possible in the real world (and this can be not only regional variants but also various environmental conditions, for example, when a sign can be partially covered by something, or the situation is very bright or very dark) is beyond the framework of this work. Rather, this work proposes the design of an effective model for recognizing traffic signs that are often observed in the German roads.

This project will complement earlier studies by incorporating YOLOv5's high-performing detection features to the GTSRB. Past research has established the use of CNNs particularly Convolutional Neural Network traffic sign recognition work [3], and this work proves that YOLOv5 work yield higher accuracy and high speeds that are suitable for use in ITS applications.

1.2 Aims and objectives

The approach chosen for the current project is to assess how accurately YOLOv5 can recognize and sort traffic signs in real-time. The purpose is to prove that by utilizing YOLOv5 traffic signs can be successfully detected and all of them can be categorized with few rectangular frames without the necessity to further stimulate the algorithm. This project will also validate the performance of YOLOv5 in terms of its speed and accuracy for real time traffic analysis for autonomous vehicles and traffic management systems.

The objectives of this project are as follows:

1. Fine-tune the YOLOv5 model on GTSRB in order to identify regulatory, warning, and

prohibition signs.

2. Image preprocessing begins with size representation, followed by augmentation of the images for suitable use in the YOLOv5 dataset.

3. Assess the performance of the model and aim at competitiveness with high level of accuracy of 95% and above by evaluating Model's metrics i.e Precision, Recall and Mean Average Precision (mAP).

4. Real time detection system for classifying traffic signs on an ongoing basis; it is important for ITS applications as one of the demonstrations of model capabilities.

1.3 Report structure

The structure of this paper is as follows:
Section 2 provides an abrupt summary of the literature as well as some of the object detection fundamentals such as the CNN and metrics. It also discusses the work principle of the YOLO family and finally focuses on YOLOv5. Section 3 presents the method that was used in the study, in terms of data processing, model training and assessment. Section 4 provides the analysis of the experiments' results. Section 5 is devoted to the discussion of the results obtained, while Section 6 presents a summary of the study and advices on potentials for future work.

2 REVIEW OF RELEVANT LITERATURE

In this literature review on object detection, we start by laying out the basics about the field and concentrating on CNNs. It includes basic measurements entropy as a criterion for discovering evaluation metrics like precision, recall and the mean Average Precision (mAP) that define the performance of detection models. Included in the review are also key hyperparameters that influence both accuracy and time-efficiency of models. In Section 2.3 of this paper, the YOLO (You Only Look Once) algorithm is introduced, and the discussion aims to analyze the development of this technique and focus on the characteristics of YOLOv5 which is used in this project.

2.1 Object Detection

2.1.1 Introduction to Computer Vision

Computer vision is a branch of artificial intelligence whose main aim is to allow a computer to gain a perception of images in the world. One of the most important tasks in this field is object detection, which consists of two key components: object recognition and detection which are luminosity and specificity or luminosity and definition a term that refers to the ability of the model to distinguish one object from another. Classification is the action of recognizing the object of a given picture, while Localization is the action of determining the position of an object at the picture.

CNNs are better suited for image classification tasks as their structure is quite different from the structures seen in most traditional feed forward neural networks. CNNs have less parameters due to local connections and shared weights to enable them handle large amount of image data more efficiently. However, down sampling methods like pooling continues to shrink the size of the feature map, thereby reducing computational load though its ability to retain critical image features is compromised [7].

2.1.2 Convolutional Neural Networks

Convolutional Neural Networks (CNNs) have emerged as one of the of most successful and powerful architectures to Deep Learning, especially for image and signal applications. CNNs are particularly sufficient for process grid-like data like images and have been used in image and video analysis, natural language processing, brain-computer interfaces and traffic sign recognition most importantly.

Key Features of CNNs

1. **Local Connectivity and Weight Sharing**: CNNs do not have fully connected layers that are found in Multi-Layer Perceptron (MLP) where every neuron in one layer connects with every neuron of the neighbouring layer; instead its neurons are connected locally. This made the number of parameters significantly smaller pulling CNNs a long way in to being computationally efficient especially when handling large inputs as are the cases with images [8]. That is why, the utilization of shared weights (through filters) lets CNNs extract location-invariant features and, therefore,

minimizes the risk of overemphasizing on those features that needs a feature engineer's attention [9].

2. **Automatic Feature Learning**: CNNs also have the advantage of having a possibility to learn hierarchical features as inputs during learning without extracting them from the raw pixel data. In the previous approaches of image processing, the features needed to process the image were manually created with crafted filters. CNNs, however, permit the filters to be designed automatically by optimizing them over the training process [10]. The lower layers of convolutional networks search for such elements as edges or texture while, higher layers to look for a more complex geometry and shapes, or even for objects and their peculiarities [11].
3. **Convolutional Layers**: Convolutional layers take the filter weight learnt and convolve it with the input data to produce feature map that points to areas of importance in the data set. It retains the relative positional attitude of pixels while minimizing the number of parameters in comparison to fully connected networks (Reference 5). Feed forward CNNs can be fast but they are actually prone to choose large inputs like images because convolutions enable CNNs to intensely work with spatial correlations locally [12].
4. **Pooling Layers**: The pooling layers are beneficial in down sampling of the feature maps obtained from the convolutional layer as well as minimizing the computational load by further down sampling the dimensionality. There are different kinds of pooling methods: Actually, there are two types of max pooling – one that chooses important characteristics in small segments of an input and omits unimportant features [13]. This allows CNNs to be invariant to small shift or distortions of the input which improve the object recognition from any given orientation or scale [5].
5. **End-to-End Learning**: CNNs also work well for end-to-end learning, which means that from raw pixel values it is possible to get the required output directly without having to define special features on the data [14]. When trained properly, the filters and parameters of CNNs ascertain the features needed for application such as image categorisation, object recognition or segmenting [15].

Applications of CNNs

- **Medical Image Analysis**: CNNs have been extensively used in detecting and

diagnosing diseases from medical scans like MRI, CT, and X-rays [16]. They can learn to identify patterns such as tumors or other abnormalities directly from image data, making them invaluable in automated medical diagnosis.

- **Predictive Analysis and NLP**: CNNs are also used in natural language processing tasks to detect local features in text sequences. They have been applied in areas such as sentiment analysis, machine translation, and even language modeling [17].
- **Brain-Computer Interfaces (BCIs)**: In BCIs, CNNs are applied to recognize patterns in brain signals, enabling users to control devices like prosthetics using their neural activity [18].
- **Traffic Sign Recognition**: The CNNs are commonly utilized in traffic sign recognition in real driving conditions, that are necessary for automated driving. Traffic sign recognition entails characterizing of different signs and the network has to recognize the signs at different conditions such as lighting and weather conditions or even when parts of the sign is obscured. For instance, the application of a given data set such as the German Traffic Sign Recognition Benchmark (GTSRB), it is possible to train a CNN in order to detect/recognize traffic signs from raw imageacy [19]. This means that the model learns to recognize signs such as speed limits, warning or prohibitory signals for instance, by parsing hundreds and thousands of annotated cases. CNNs ability to learn features at multiple scales comes in handy when features like small but important features such as the figure ‘40’ on a ‘speed limit 40’ sign need to be detected [20].

Due to their inherent ability to learn hierarchical features of raw data and computational effectiveness, CNNs are best suited to image and signal processing applications. While the pooling operations help to minimize the data dimensions, feature maps help to reduce the necessity of the feature extraction that should be made by a human. For this reason, CNNs have been employed in various areas like, medical imaging, NLP, traffic sign recognition, and predictive analysis, and making it a core part of the current AI technology [21].

2.1.3 Object Localisation

Object detection algorithms enhance the basic design of the implementation by incorporating the ability to localise an object of interest within an input image [DOL 11, ORE 15, YAD 24].

Data preparation involves manual shot marking of object locations within a particular frame. Two major modes of labelling are used in most tasks which are the bounding boxes and segmentation [22]. In this case, a human operator fits a bounding box around the object of interest (or in the event of segmentation, a more intricate polygonal silhouette), and then assigns a class label [23]. Hence, ground truths are about categories as well as positions of objects of interests. BB and segmentation have been employed to a large extent of success even though there is a distinction between them. BBs on the other hand are cheap on computation and suffer from warping when augmented, polygons are more accurate, and while handling augmentation better they are relatively slower. For large datasets (containing hundreds of thousands of images) a purely supervised approach would be extremely slow, therefore the shift to focus on semi-supervised and transfer learning. Modern DL-based object detectors can be divided into two general categories : proposal-based and region regression-based [24, 25, 26, 27].

2.1.4 Dataset Considerations

In order for object detection algorithms to be as effective, this algorithm requires many large datasets with high quality so that it can identify the features correctly. That is why the dataset has to include a vast number of images, with enough images of the same class for better learning of the model. That is, in most of the cases the datasets include several hundred thousands of images, each of which needs to be labeled manually. Lacking sufficient and accurate data, the model cannot be generalized to recognize more aspects of feature-target correlation. Sadly, data collection and preparation can occupy a significant portion of the total effort, and for many practical problems datasets may be missing or not appropriate for the task at hand [11].

As mentioned before, obtaining and annotating vast datasets is a challenging task, which becomes virtually impossible to guarantee balanced representation of all classes. To be able to circumvent this problem, image augmentation is applied to artificially increase the size of the dataset. One of the ways in which this technique assists is in the ability to add variability into the data that may resemble the environment a model will experience out in the real world. Moreover, certain directives need to be followed if the data is divided into the training data, the validation data, and the test data. The test set for instance has to be balanced across all the classes in to offer the model a diverse and accurate set to perform its analysis on. While

working on images with multiple objects the model may need more training but this preprocessing makes the final model more powerful and efficient [6].

The purpose of training a neural network is to perform the iterations of back propagation adjusting weights in layers through the gradient descent method to match the prediction of the model with the actual results. It happens when a model is overfitting and optimized well for the training data and the data distributions on which it is trained. This issue can be easily spotted by finding out the difference between the training loss and the validation loss. Both should ideally reduce over time and ideally reduce together as proven that the model is learning correctly. Besides examination of the curve from the top-left corner of the plot, the two lines yield useful information; if validation loss is much lower than the training loss, it may suggest that underfitting has occurred, which means that the model is not picking up enough information from the training data. On the other hand side, if the validation loss starts to rise while the training loss is still continuously falling this means that the model has started to overfit [17].

Consequently, overfitting occurs; where the model is overly complex and optimized for the training dataset but performs poorly elsewhere. The primary method in guarding against overfitting is through the quality of data set used in the research. This is an indication of a network's architecture – it is more effective sometimes to have a model with fewer parameters to decrease the chances of over fitting [10].

2.1.5 Image Augmentation

Data augmentation is a method of increasing the volume of a dataset by creating similar images from previously existing ones. These transformations alter images in a way that makes them fresh to the model during commanding. The use of data augmentation has been removed to enhance applications more than raising the network size [6]. It is generally possible to identify two main categories of augmentation: the geometric one, possible operations in this case include: rotation, scaling, cropping, and translation, and the photometric one which influences the property of the image such as brightness, contrast, or saturation. Other existing techniques include adding noise, blurring, or erasing parts of the images to improve a dataset's variability [28].

Spatial modifications like scaling and cropping are among the inherent methods of data augmentation and are ordinarily used to normalize the dimensionality of images. There is also

an added ability of the models scaling to interpret objects in other aspects depending on their size. However, special effort should be paid to the fact that transformations should not affect the degree of object's spatial integrity defined by bounding boxes. For instance, too much rotation or shearing deforms the rebounding boxes causing them to be indifferent. With most spatial transformation done rightly, the model is capable of recognizing different versions of the same image enhancing its performance on the new data it has not seen before. These methods are particularly effective when the object may be subjected to different orientations. Realized photographic corrections of brightness and contrast are important when the model functioning will take place in conditions with different lighting or color is important for object classification [29].

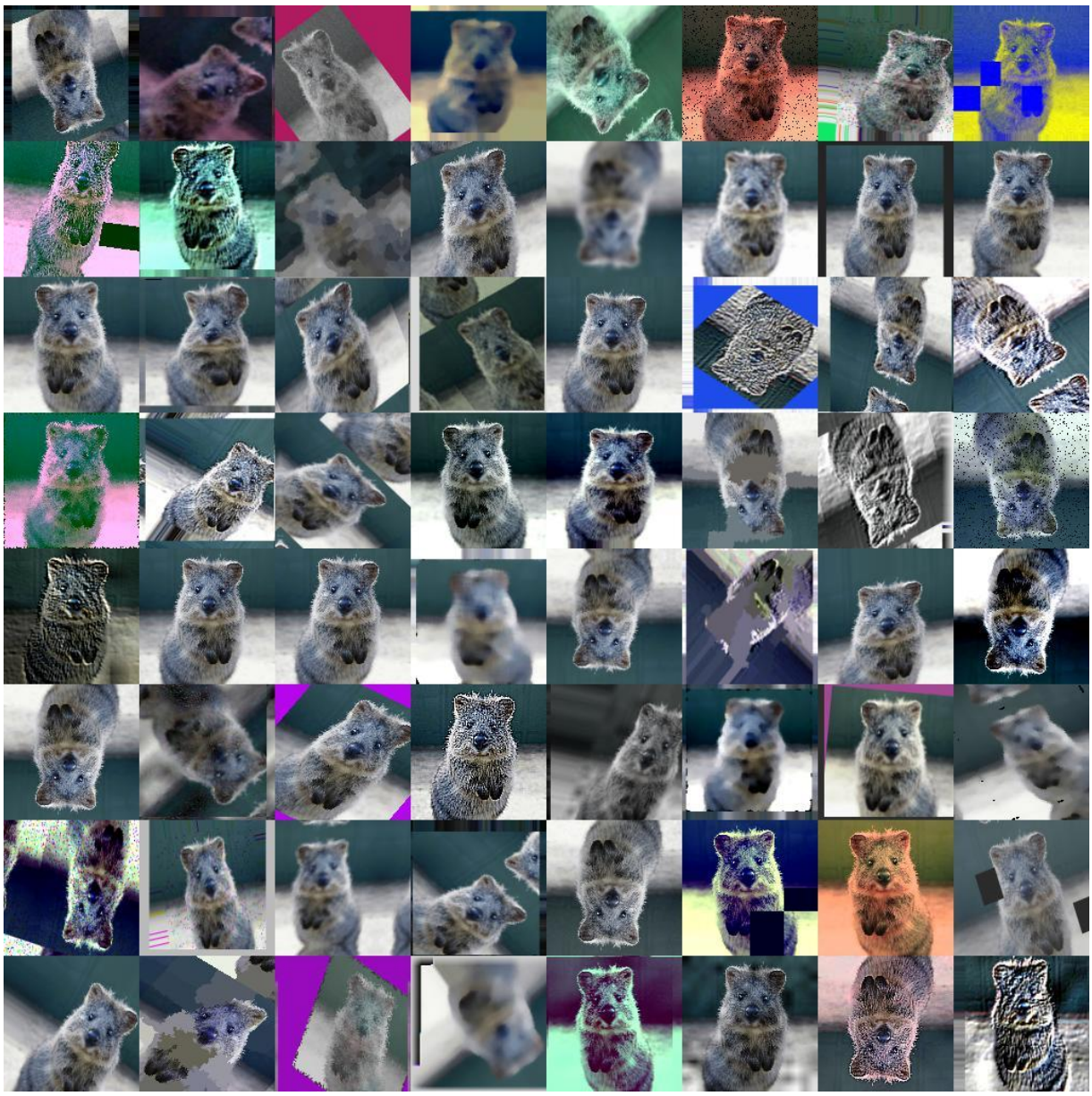

Figure 1: Various image augmentation techniques [30].

The transformations described above operate on images without substantially altering the information: Advanced augmentations either increase or decrease the image. Of these operations, one is self-explanatory: blurring; adding noise to an image is taken to mean replacing some pixels with a different value [10]. These can help in identifying those targets that are blurred or those which are of low resolution. Cutout is extremely helpful in dealing with occluded objects: Some of the number of predetermined size boxes are taken away from the image, whereby black pixels replace them. As a number of these blank areas may occasionally encompass intricate features, the model can only infer about the object from the residual data [31].

Mosaic is an operation which is most frequently used in leveling out class ratios and carries new pictures created from segments of other pictures in the batch. Even though this process is not intelligent and usually makes random choices with reference to the original images, it produces an image with examples of many of the classes. Similar to spatial transformations, it destroys annotations which have to be checked to tackle the severe distortions off the ground truths[28].

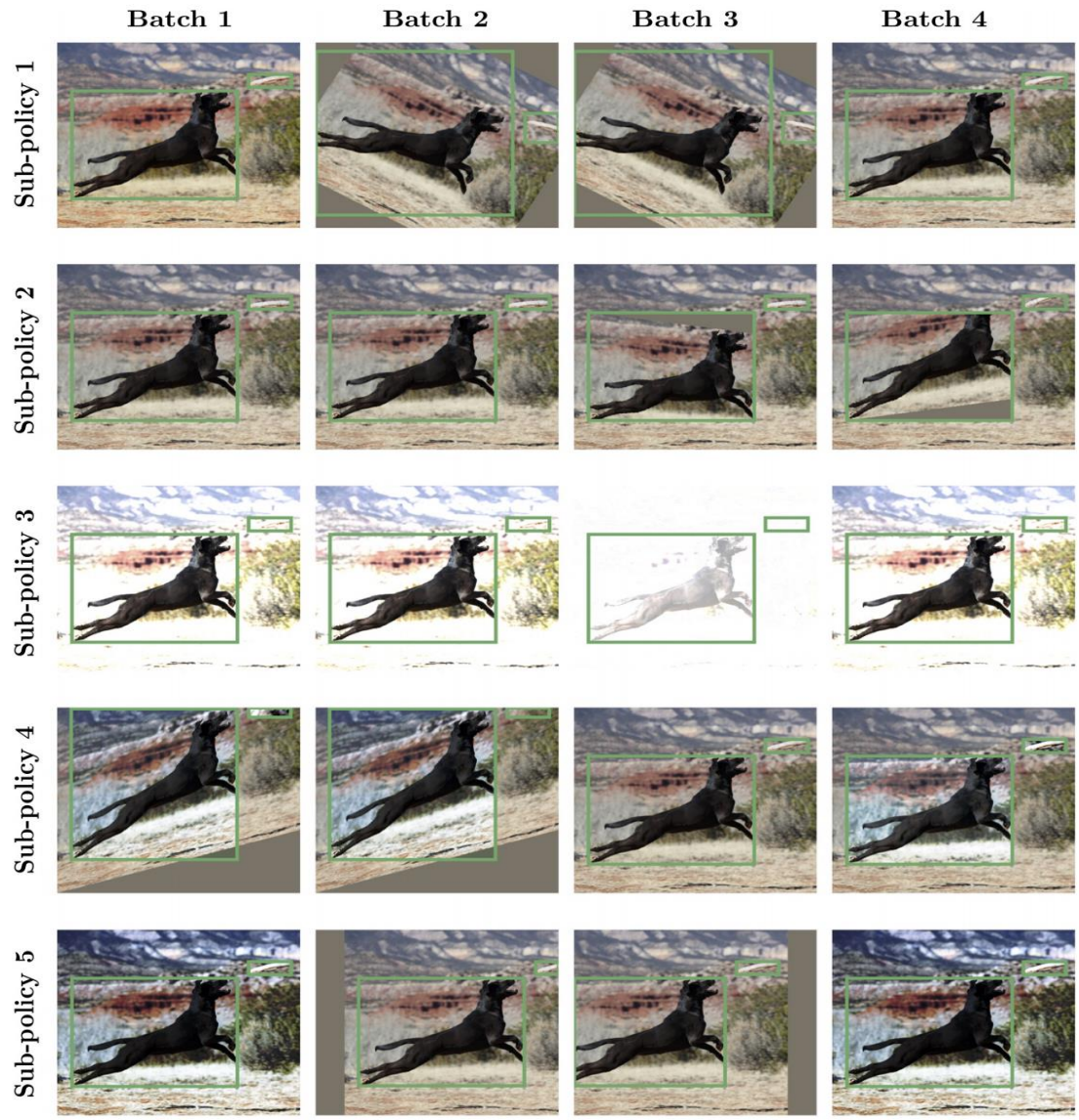

Figure 2: Various bounding box level augmentations [30].

Finally, "bounding box level" augmentations use standard augmentations at the inside of the bounding box. This leads to good results taking it over entire-image methods because it augments the subscription's context while at the same time minimizing background detections and false positives.

Augmentation, in general, is done before the actual training; thus, that is known as the preprocess stage. The aim is usually to add quantity to the training and validation sample sets,

and if possible, prepare for the sort of generalizations done by a finished model. Normally, certain transformations are prescribed as a range, such as scale up to 10%, as well as blur which can affect up to 2% of the pixels, while exact values are allowed to be a random. It is also worth though to note that some algorithms offer augmentation as a part of the training process (as in the case of YOLO algorithms), in this content raw images can be fed directly into the network in which case these transformations are adjusted as hyperparameters. Selecting right augmentations is again either a totally random process or it has some intuition regarding the task and the algorithm in practice.

2.2 Metrics and Hyperparameters

2.2.1 Evaluation Metrics

It was discovered that the percentage of correctly detected ground truths is not an ideal measure of object detection as it doesn't capture various types of error. When object detection problems have been better defined, improved measures of accuracy have been introduced and the most known measure is the mean Average Precision (mAP).

AP, Average Precision, is estimated with the help of IoU, Intersection over Union with take into account precision and Recall data for modelling results. IoU reflects the overlapping area between the predicted bounding box and the ground truth dividing it by the area of the predicted bounding box union with the ground truth. This offers a better assessment on the correlation between the predicted bounding and the actual object in the image [6].

$$\text{IoU} = \frac{\textbf{Area of Union}}{\textbf{Area of Overlap}} \tag{2}$$

If an IoU threshold is defined, it is possible to create a measure that defines the correctness of the predictions. There are four cases that must be considered [32, 33]:

1. **True Positive (TP)**
   The true positive is when the model detects an object and correctly assigns the class label to that object and the predicted bounding box entirely covers the ground truth

box – it means the prediction is good.

2. **False Positive (FP)**
   A miss identification is defined as the scenario in which the algorithm draws an object that is not in the picture or when it labels an object wrongly hence a wrong detection.

3. **False Negative (FN)**
   In the case where the model fails to end an existing object in the image, it is referred to as a false negative. In this case the model exclude the object from its consideration, deeming it to be part of the background.

4. **True Negative (TN)**
   The true negatives are also evaluated when the model is right when it says that there is not an object in such part of the image. Nevertheless, for the same precise reason that there are an infinite number of points in the image plane that could not contain objects to begin with, true negatives are not generally reported when evaluating object detection.

These terms are crucial for building confusion matrix which comes handy while calculating important parameters such as precision, recall and accuracy to assess the performance prediction model. It can be possible to help them create a confusion matrix, which is very useful statistic tool.

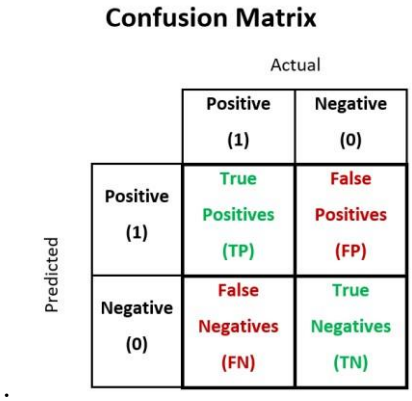

Figure 3: A confusion matrix layout.

The computation of confusion matrix is not a necessary step but it is effective way to convey the important data. The most important metrics are discussed below:

Accuracy (A) was used as the measure that explains the ability of the model to identify the

correct object from all the objects it was expected to predict. The mAP [5] quantifies the fraction of the ground truth and detected bounding boxes that are correctly classified on top of meeting a certain Intersection over Union (IoU) threshold. Accuracy is helpful in determining how credible the model predictions are to the percentage of total detections it makes [6, 32].

$$Precision = \frac{TP}{TP+FP}$$

Recall (R) is a performance measure that quantifies a model's capacity for identifying all the essential objects within a picture. It is arrived at by dividing total number of true positive detections by the total number of actual object (Ground truths) in the scene. Recall does this by indicating how many out of the actual objects was identified by the model without regard to whether all the predictions were accurate or specific in terms of location [34, 32].

$$Recall = \frac{TP}{TP+FN}$$

This is true due to the fact that the two measures are better than the one, accuracy since part of tuning a model is balancing between the precision and recall. An ideal detector will detect all ground truths and assign all the predictions correctly, therefore, FN = 0 and FP = 0. In practice, to achieve higher precision, the confidence level that defines the predictions is raised while in return the recall decreases. A model's robustness can be measured by the area of the precision-recall curve, by which we are able to judge the general performance when using different confidence levels. High requirements are given for precision and recall: a large area means that the model should still work well in case of varying confidence values.

The curve may be normal, logarithmic, or irregular, and so the area under the curve is initially interpolated (n-point or all). This technique averages the maximum precision at a set of equally spaced recall levels, as shown below:

$$\mathrm{AP_n} = \frac{1}{n}\Sigma_{R\ \{0,\frac{1}{,},\frac{2}{,},\ldots,\frac{n-1}{,}1\}} \in P_{interp}(R), \quad (5)$$

Average precision (AP) is the preferred metric, and this is easily extended to multiclass detectors using the mean average precision (mAP):

$$\text{mAP} = \frac{1}{N}\sum_{i=1}^{N} \text{AP}_i, \tag{6}$$

where AP_i is the average precision for the i^{th} class and N is the total number of classes.

Standard for object detection evaluations through the use of four main metrics:

1. **AP (Average Precision)**: This is done by measuring the Intersection over Union (IoU) of the car and local region. It can be a single value, like mAP50 or mAP75 which means that 50% or 75% of Intersection over Union was achieved. Alternatively, a range of IoU values from 50% to 95%, incremented by 5%, is used, commonly referred to as mAP50:5:95.
2. **AP across scales**: AP is categorized into three size classes based on the object's area: small (objects have an area less 32^2 pixels), medium (objects occupy a region bigger than 32^2 but less than 96^2 pixels) and large (objects occupy an area greater than 96^2 pixels).
3. **AR (Average Recall)**: This measures the maximum recall when a certain number of detections per image are obtained, taken over different IoU thresholds and over all object classes.
4. **AR across scales**: Similar to AP, AR is also reported for the same object size categories: small, medium, and large. These are represented by the AR-S: small; AR-M: medium; AR-L: large.

Alias	Meaning	Definition and Description	
FPS	Frame per second	The number of images processed per second.	
Ω	IoU threshold	The IoU threshold to evaluate localization.	
D_γ	All Predictions	Top γ predictions returned by the detectors with highest confidence score.	
TP_γ	True Positive	Correct predictions from sampled predictions	
FP_γ	False Positive	False predictions from sampled predictions.	
P_γ	Precision	The fraction of TP_γ out of D_γ.	
R_γ	Recall	The fraction of TP_γ out of all positive samples.	
AP	Average Precision	Computed over the different levels of recall by varying the γ.	
mAP	mean AP	Average score of AP across all classes.	
TPR	True Positive Rate	The fraction of positive rate over false positives.	
FPPI	FP Per Image	The fraction of false positive for each image.	
MR	log-average missing rate	Average miss rate over different FPPI rates evenly spaced in log-space	
Generic Object Detection			
mAP	mean Average Precision	VOC2007	mAP at 0.50 IoU threshold over all 20 classes.
		VOC2012	mAP at 0.50 IoU threshold over all 20 classes.
		OpenImages	mAP at 0.50 IoU threshold over 500 most frequent classes.
		MSCOCO	• AP_{coco}: mAP averaged over ten Ω: $\{0.5 : 0.05 : 0.95\}$; • AP_{50}: mAP at 0.50 IoU threshold; • AP_{75}: mAP at 0.75 IoU threshold; • AP_S: AP_{coco} for small objects of area smaller than 32^2; • AP_M: AP_{coco} for objects of area between 32^2 and 96^2; • AP_L: AP_{coco} for large objects of area bigger than 96^2;

Figure 4: Common evaluation metrics for generic object detection [32].

2.2.2 Hyperparameters

Hyperparameters are variables which are established prior to training a model and are very influential throughout the process. These can be structural, about the architecture of the network for learning or functional, about the learning rate, batch size, stride, and momentum. Actually, the selection of different hyperparameters can also influence the learning capacity of the model with regards to the data. Earlier approaches of choosing the hyperparameters of a model have involved guesswork, and the results depend on the experience and the knowledge of the model which often takes a lot of time. More recently, techniques including AutoML, have been devised to help in choosing both the number of algorithms and the values of hyperparameters most appropriate for a particular dataset [35,36].

No. of epochs:	The number of times training images pass through the algorithm.
Batch size:	The number of images processed per iteration.
Activations:	AFs fit complex functions to high-dimensional data.
Loss function:	Mean squared error, error rate, cross-entropy loss, etc.
Learning Rate:	The amount by which parameters are adjusted for a given gradient.
LR scheduling:	A programmed learning rate growth/decay sequence.
Momentum:	An SGD optimiser, creating a resistance to changes in the gradient.
Regularisation:	Techniques to minimise the likelihood of overfitting.
Filter sizes:	The dimensions of kernels in convolutional and pooling layers.
Stride:	The number of elements that a filter is translated at each step.
Pooling type:	Subsampling layers, typically average or max pooling.
Padding:	A method of dealing with mismatched stride and image size. If part of a kernel lies outside of the image, this extra area is populated with values (usually zero).
NMS threshold:	Non-maximal suppression limits the amount of bounding box overlap allowed in a prediction. When running inference, this can be tuned (along with confidence level) to improve precision.
Dataset Split:	The percentage of images set aside for each of the stages of training. The test set should be carefully curated to include examples of all classes in their natural contexts. Sets should not have any overlap of images to avoid data leakage and adapting to the training set.

Figure 5: Common model hyperparameters, with descriptions.

It is very fundamental, and its general designation reflects its strategic role in defining the size of the step that is taken for the adjustment of the model parameters. There are cases where its complexity is adjusted, beginning with the learning rate and moving to second-order derivatives, and with revisers like Adam and learning span schedulers; which can be changed during training to counter with problems that afflict learners early on, such as convergence or overfitting [37]. The monthly effect is another criterion since it makes the model proceed further in a more persistent manner, than getting stuck in local minimization. It also means that high momentum helps the model to go through the error surface with less jitters, although lower the momentum towards the final stages of the learning process can be helpful for convergence [38]. Batch size; the number of samples before adjusting the parameters of the model also has a relatively crucial role. Reduced batch size requires lesser memory, but extremely reduced sizes such as below fifteen may cause fluctuations in gradients and hence in training [39]. The learning rate itself and the parameter interacting with it, including batch size and momentum, have been in a trade-off and learning rate with some new studies focusing on various structures for optimum sizes of batch [40].

2.3 YOLO Algorithm

One stage approaches jointly classify and localise in a forward pass through the network and the various tasks in OD are modelled as regression problems [6]. YOLO was the first of these to gain popularity notching through inference speeds of up to 155 fps in its small version – YOLO fast. Everything, from predicting bounding boxes, to feature extraction, NMS and reasoning about contents of an image, is done by just a single CNN.

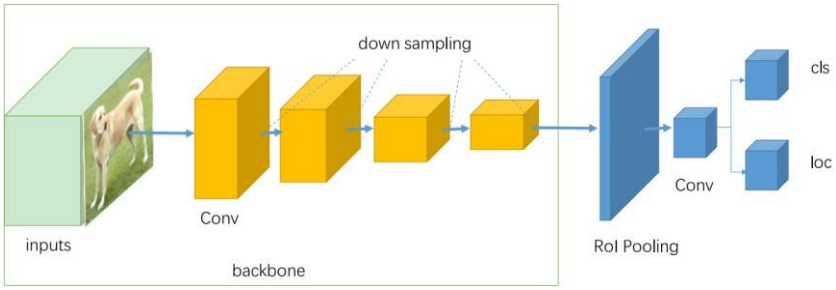

Figure 6: Block diagram of one-stage object detection algorithms

2.3.1 YOLOv5 Algorithm

YOLOv5 by Glenn Jocher from Ultralytics launched in 2020 is of interest because of its simplicity, Yet, there is no paper published about it officially. As a major advancement for YOLOv5 model, all the codes of YOLOv5 are written in Python, which is more friendly than those based on other programming languages of previous versions of YOLO. This readability has made this engine popular and well supported by their website users. One of its more unique characteristics is hyperparameter evolution, meaning that hyperparameters are trained in the framework of a model. A genetic algorithm is used to select the best configuration based on fitness, which is defined by a combination of mAP@0.50 and mAP@0.50:0.95. This kind of approach makes the model to converge much faster and with minimal adjustment of the error to obtain the best results. Thus, they can act as the basis for further modifications of the configurations.

As mentioned here this is the only image online for YOLO v5 integration, and there is no research paper available which was published for YOLO v5, the illustrations used bellow are all three unofficial and are being used for explanation purpose only.

It is also good to mention that YOLOv5 was released with five different sizes:

- **n** for extra small (nano) size model.
- **s** for small size model.
- **m** for medium size model.
- **l** for large size model
- **x** for extra large size model

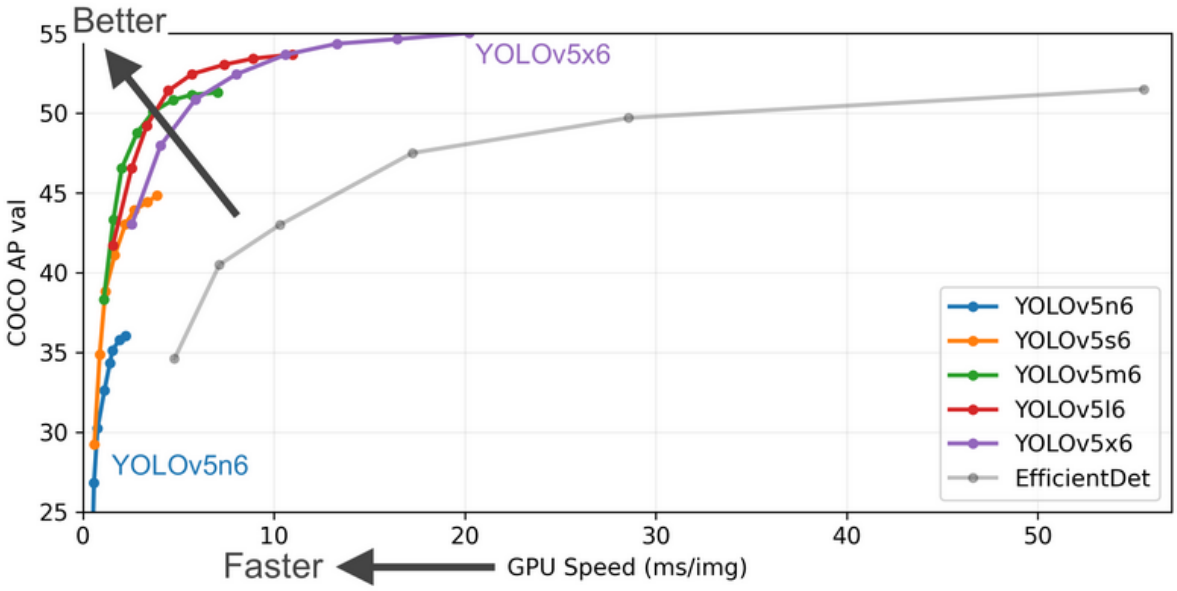

Figure 7 Different Size Yolov5 Models Speed

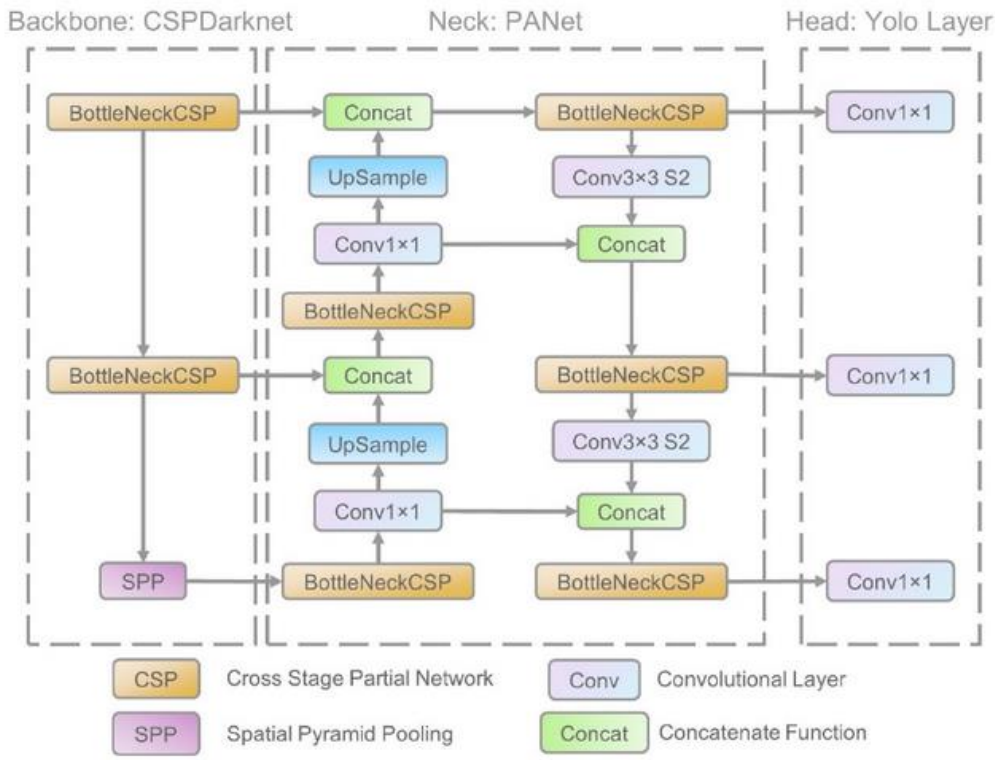

Figure 8 Yolov5 Architecture

Actually YOLO is a deep network and this is because its propagates information to the deepest layers and solves the vanishing gradient problem using what is referred to as residual and dense blocks. But one of the benefits of using dense and residual blocks is the issue of vanishing gradients. CSPNet is of assistance in addressing this issue because it reduces the gradient flow.

Selecting an activation function is essentail for any deep learning model, for the YOLOv5 the authors uses SiLU and Sigmoid activation function.

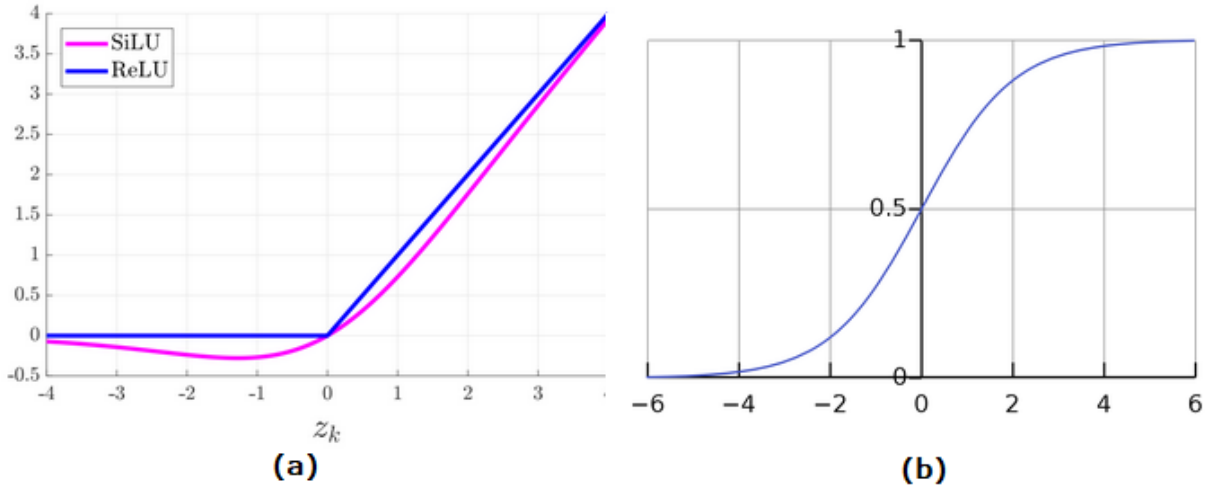

Figure 9 Activation functions used in Yolov5

Graph of the activation functions used in YOLOv5 Images of the activation functions used in YOLOv5 (a) SiLU function graph. (b) Sigmoid function graph.

The original name of the proposed function is SiLU or Sigmoid Linear Unit, and it is also known as swish activation function. Convolutions were applied with the parameter in the hidden layers. While, the Sigmoid activation function has been used with the convolution operations in the output layer. OLOv5 returns three outputs: the classes of the detected objects, their coordinates within in the frame and the general probability of the object's presence. Hence during the computation of the classes loss and the objectness loss, it employs the BCE – Binary Cross Entropy. While CIoU refers to computing the location loss, it is good to use GIoU (Generalized Intersection over Union). The final loss can be calculated by the loss thaw is given by the following equation

2.4 Traffic Sign Recognition

The aim of traffic sign recognition is to design algorithms which are able to successfully recognize the traffic signs from images; widely used in a wide variety of uses such as self-driving cars and traffic management systems. In this field a lot of progress has been achieved in the recent past when deep learning models were incorporated. However there still remain significant difficult in achieving human-like performance in realistic and dynamic road contexts. The German Traffic Sign Recognition Benchmark (GTSRB, Stallkamp et al., 2011 [5]) are widely used for this purpose. This dataset has been vital in making new findings due to the variety of images offered to evaluate models for traffic sign recognition. Also, Houben

et al. (2013) also highlight approaches for assessing IDS, it is known that we have to employ performance measures such as precision and recall for measuring the effectiveness of traffic sign detection in real time [41].

A main difficulty in traffic sign recognition is to determine what it takes for a system to recognize signs with high accuracy, due to the fact that the task involves several sub-tasks including detection, classification and handling occluded or overlapping signs. Studying this problem usually lacks holism and conceptually most of the existing research focuses only on some of its aspects [5]. Furthermore, traffic sign object provides technical aspects of variation in size, shape, and color because of horologic condition and lighting at night. Of course, such variations might cause problems with recognition and classification at the initial stage of the process. For example, in real-world scene, signs may be large or small, or may be placed close or far from the object they refer to; the models to be learned must have good generalization ability under different conditions. The issues such as the recognition of traffic signs which are complex when placed closely together or are partly obscured can be solved by using larger and multiple datasets coupled with the use of neural networks that can contextually interpret the picture [11].

Conventional methods of traffic sign recognition are procedural and are taken step by step after image preprocessing. In most cases, steps involved consists of noisy reduction, image segmentation or binarization and contrast/brightness stretching to improve readability. After the preprocessing of image, traffic signs are segmented or we can say traffic signs are segmented and isolated for the further analysis. Early systems employed supervised-learnt models for classification of the signs with K-nearest neighbors (KNN) and support vector machines (SVM). Nevertheless, with the emerging of the deep learning, CNNs are now widely used for traffic sign classification and are already demonstrated to outperform the compared approaches [41].

In traffic sign recognition, one of the major problems is the differences in form which can be influenced by the weather, light, and can be partially obscured. Moreover, traffic signs may vary in size and shape following its position in photograph or proximity to the camera. Some of these datasets are produced in order to offer a large number of examples in the field of recognition, for example the German Traffic Sign Recognition Benchmark (GTSRB). This dataset mainly containing more than thousands of images belonging to 43 classes of traffic signs has been a vital asset that helped in enhancing the precision and the resistant of traffic

sign recognition models [5].

This is especially important since data augmentation is used during training of these models by gradually enlarging the size of the dataset and adding variations. Rotation, scaling and brightness augmented images mimic the environment in which the traffic signs are exposed during real life situation. For instance, small scale transformations should be translated to tilted signs, where as scaling should be translated to signs viewed from different distances. Houben et al. (2013) demonstrated that applying these augmentations enable the model to avoid overfitting, and to improve its performance on unseen data [11].

In other cases, other forms of augmentations like adding noise or cutting some part of the image randomly is carried out to emulate circumstances where the sign might be partly screened by an object such as a tree or car in the real world. The processes of flipping through thePicture plane horizontally and vertically is a common practice in image augmentation for many tasks, but it is ineffective for the recognition of traffic signs, as these signs are aligned in a particular direction in the real world.

Traffic sign recognition has evolved hugely due to such deep learning models, and the high quality datasets such as GTSRB. However, the variation of data distribution from the source domain to real-world environment calls for ceaseless data augmentation and utilization of more complex model structures to enhance performance [11].

3 METHODOLOGY

3.1 Dataset Creation

3.1.1 Data Generation

In my case when working on the German Traffic Sign Recognition, the first thing I did to the dataset was to convert it to YOLO format. Following that, further augmentations were included in the images and the bounding box frames corresponding to them. To achieve this, these augmentations were created to simulate real life scenarios for instance, lights, orientation and scaling. However, these augment.

3.1.2 Annotation

Since traffic sign features should be selected based on vision, bounding boxes around the traffic signs should include not only the sign of interest but also a small part of the context in order to obtain better results. This takes a bow from the network inference model in an attempt

at correctly identifying the location and classification of signs as a driver does with road signs in their environment. Some of the labelled instances from my dataset can be illustrated as follows sample data instance as shown in the image above. This is illustrated in the following where bounding boxes have captured the actual region of interest of each sign, including areas where signs are partially masked, rotated or distorted to emulate real-world scenarios.

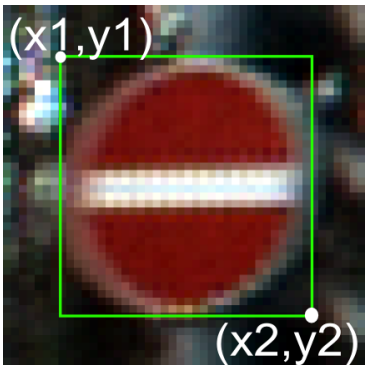

Figure 10: Example of bounding box annotation

The original dataset is 43,000 life like images. To increase the size of dataset and apply different augmentations In total, 67,020 images were created for training, all of these were labelled and included in the final da having separate train, validation and test folders each having their own images and labels folders separately. These 67020; which is the training image dataset images contains 43 classes. In figure 26 the class distributions of the data set is illustrated while figure 27 provides some insight of the image density.

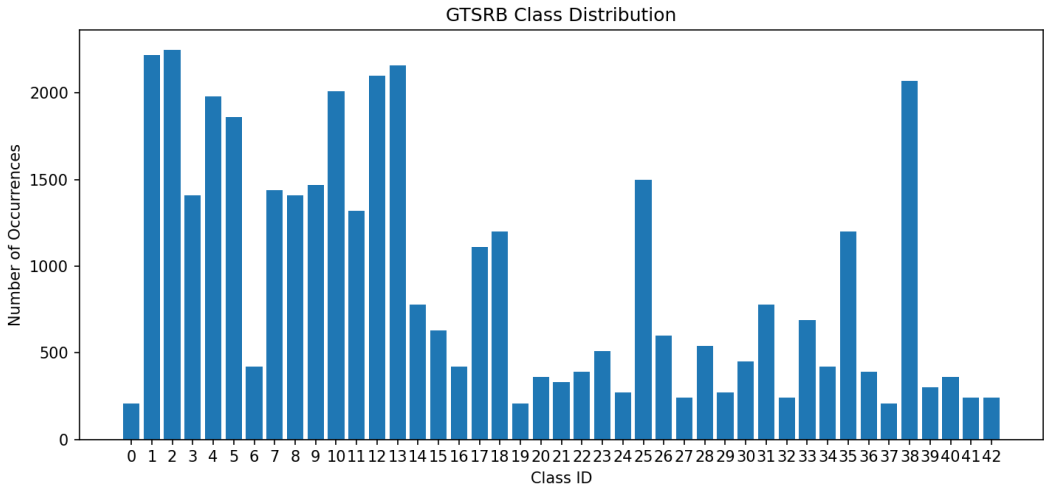

Figure 11: Distribution of instances across classes

. Typically, the classes will be overrepresented and the algorithm will assign high values of both precision and recall scores to them .After augmentation steps, the instances across dataset's each class is increased across training , validation and test folders shown respectively in Figures below:

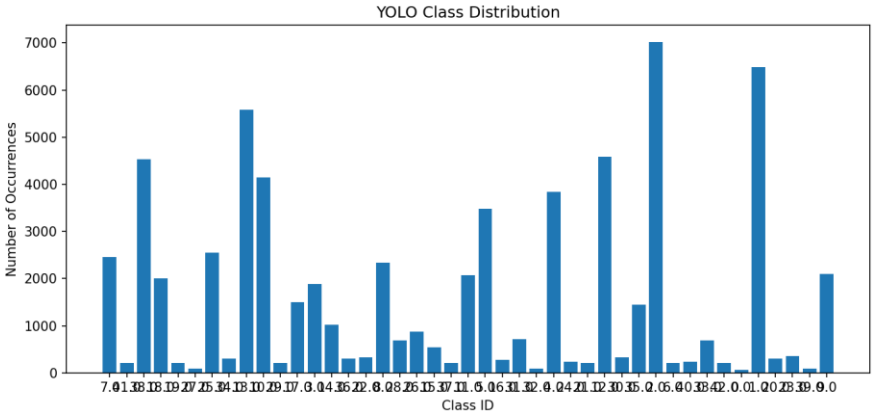

Figure 12: Distribution of instances across classes

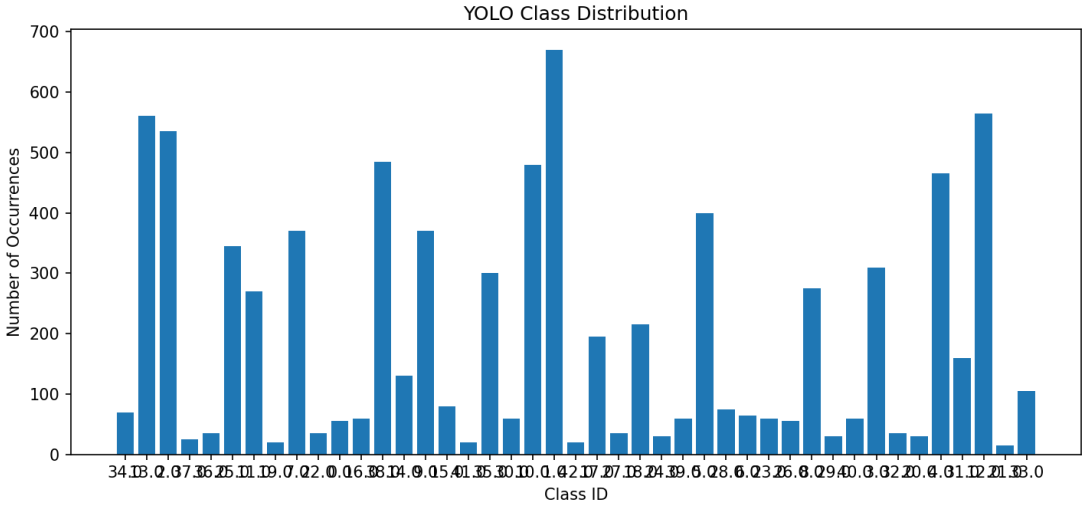

Figure 13: Distribution of instances across classes

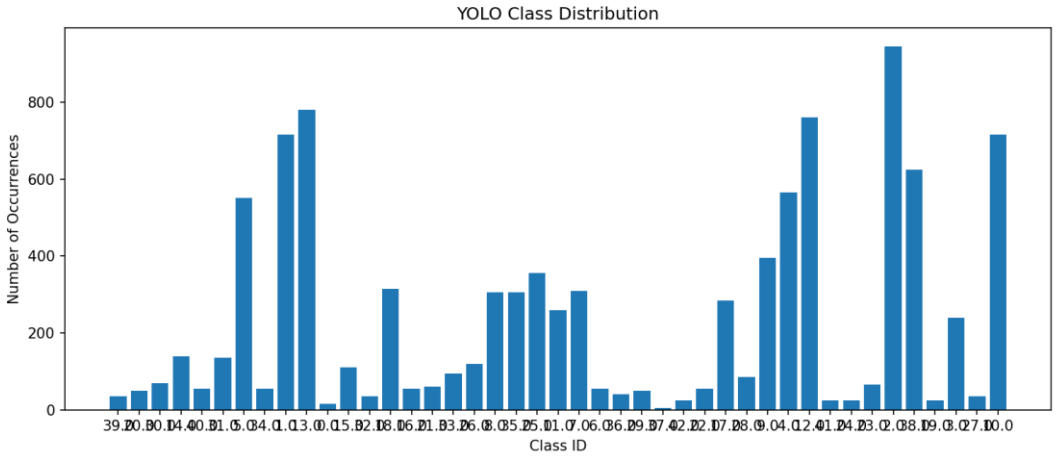

Figure 30: Distribution of instances across classes

3.1.3 Preprocessing and Augmentation

To enhance traffic sign recognition performance of the trained model, I employed extensive data augmentation methods involving spatial and photometric data distortions. The augmentations were used progressively across the training data to diversify the set of training images and improve the model stability. The specific augmentations included:

1. **Image-wide transformations (spatial)**:
    - **Random 90-degree rotations** (with a probability of 0.5) to allow the model to recognize signs from different orientations.
    - **Horizontal and vertical flips** (at a probability of 0.5) to help the model understand signs that might appear from various viewpoints.
2. **Image-wide transformations (photometric)**:
    - **Brightness and contrast adjustments** (p=0.3) to simulate different lighting conditions and improve the overall visibility of the signs.
    - **Gaussian blur** (p=0.3) to mimic the effect of out-of-focus images and ensure the model can handle blurry inputs.
    - **Alterations in hue, saturation, and value** (p=0.3) to ensure adaptability to varying color perceptions due to different lighting conditions.
    - **Contrast Limited Adaptive Histogram Equalization (CLAHE)** (p=0.2) to enhance the details in images that may be poorly lit.
    - **Gamma adjustments** (p=0.2) to replicate exposure variations.

- **Channel shuffling** (p=0.2) to diversify color representation within the images.
- **Motion blur** (p=0.2) to account for potential blurriness from movement.

3. **Bounding-box level transformations (spatial)**:
   - **Shift, scale, and rotation transformations** with controlled limits (shift limit: 0.0625, scale limit: 0.1, rotation limit: 45 degrees) at a probability of 0.5. This ensures that the bounding boxes accurately represent any modifications applied to the images, maintaining alignment with the actual traffic signs.

These augmentation techniques made the rich augmentation of the dataset handy which helped the model learn and classify frequently appearing traffic signs in various different challenging circumstances. Such an approach will make the improvement of the model to be strong when used to predict in the real-world setting.

3.1.4 Final Dataset

The dataset used for training the final models of this research has 67,020 images belonging to 43 classes. Figure below shows few images from the training batch.

Figure 15: Example training images with different augmentations

3.2 Model Training and Optimisation

3.2.1 YOLOv5s (Preliminary Model)

The model was trained in the original German Traffic Sign dataset with no augmentation. Owing to this constraint inherent in my hardware, the training was done on the CPU instead of the GPU, thus taking approximately two days to undertake 49 epochs of training. Below are the key training metrics:

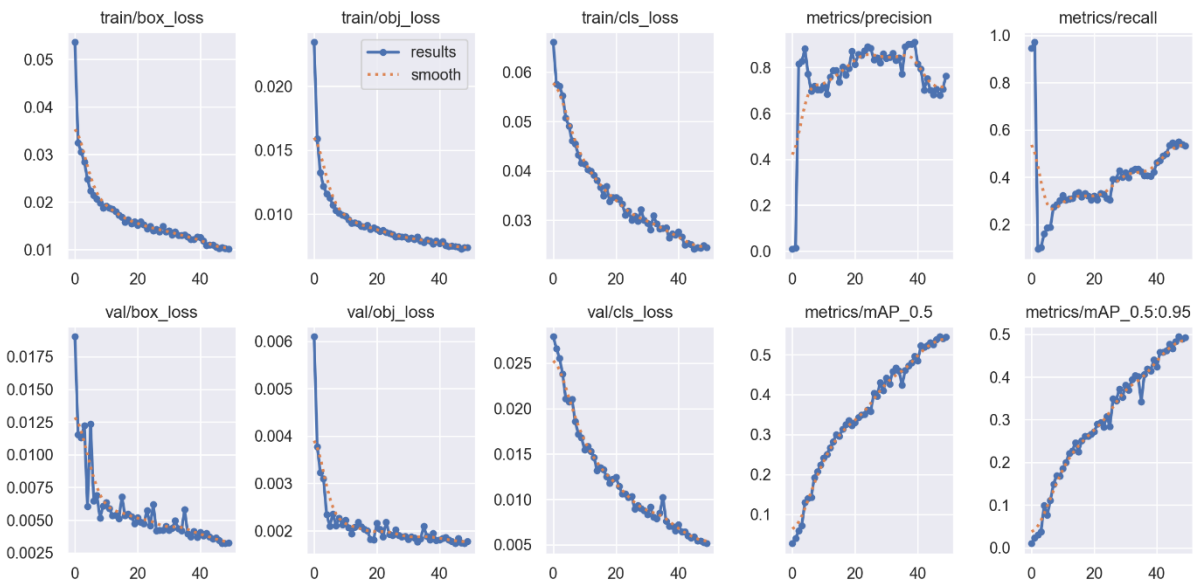

Figure 14 Training Metrics for Preliminary Model

3.2.2 YOLOv5s (SGD)

As the optimizer during training this traffic sign recognition model, I employed Stochastic Gradient Descent (SGD). Here's a breakdown of the key training and validation metrics, focusing on the most significant aspects:

Training Loss Graphs:

1. **train/box_loss**:

   This metric quantifies the deviation of the predicted bounding boxes from the true bounding boxes or ground truth. The gradual reduction in this loss also shows changes in the model with more accurate bounding box estimation, which has plateaued a little in the present phase. The final value converge to 0.03 indicating high box prediction accuracy.

2. **train/obj_loss**:

   This reflects the ability of the model to classify objects from the background depending on the loss difference produced. Early falling with a slower decline later on is a good prognosis. Object Doom Objectless rapidly trained with the objectness to a loss that was stable at approximately 0.02 indicating low false positives.

3. **train/cls_loss**:

Accuracy loss calculates to how accurate the model is in categorization of objects or items into their respective groups. A slow decrement with almost stagnation at 0.02 points to the fact that the model was perfectly able to identify traffic sign images with greater success.

Validation Metrics:

1. **metrics/precision**:

   The accuracy means how many out of the predicted positive cases are actually positive. The line becomes almost closer to 0.99 and therefore, the model is good at its job of predicting actual traffic signs without classifying imaginary signs in images.

2. **metrics/recall**:

   Recall measures the model's ability to identify all necessary objects. Recall with a value of around 0.9 means that most objects are captured and the model may not be bad at all at capturing a few that are left behind.

3. **metrics/mAP_0.5**:

   Overall object detection performance is measured with Average Precision (AP) while the mean value is obtained with Prefix mAP denoting the IoU threshold at 0.5. A sharp increase in the mAP value and its further stabilization at a level close to 0.95 convincingly proves the effectiveness of the model for accurate object detection.

4. **metrics/mAP_0.5:0.95**:

   The higher mAP score with more stringent IoU thresholds and the saturation near 0.8 of this metric illustrates that our model successfully identifies traffic signs with less or more overlap between the predicted and ground truth box locations.

Validation Loss Graphs:

1. **val/box_loss**:

   The decreasing trend in validation box loss, stabilizing around **0.04**, confirms the model's capability to accurately predict bounding boxes on unseen data.

2. **val/obj_loss**:

   Similar to the training objectness loss, this metric shows improvement in identifying objects in the validation set, with a final value of **0.02**, indicating minimal false positives on validation data.

3. **val/cls_loss**:

   Validation classification loss closely follows the trend of the training classification loss, leveling off around **0.02**, signaling good generalization in classifying traffic signs on both training and unseen data.

Learning Rate Schedulers:

The learning rate schedulers show an initial sharp increase (due to warm-up) followed by a steady decrease. This pattern helps the model start learning quickly and then stabilizes learning as the training progresses, which leads to effective optimization without abrupt changes.

Decreasing loss curves for both training and validation data suggest the model is learning efficiently and avoiding overfitting. **High precision (~0.99) and recall (~0.9)** indicate the model detects traffic signs with minimal false positives and negatives. **Strong mAP scores (~0.95 for IoU 0.5, ~0.8 for IoU 0.5:0.95)** confirm accurate object detection and bounding box placement. These metrics demonstrate that the model is well-optimized and exhibits excellent detection performance, though minor tweaks could further improve its accuracy in detecting and classifying traffic signs under varied conditions.

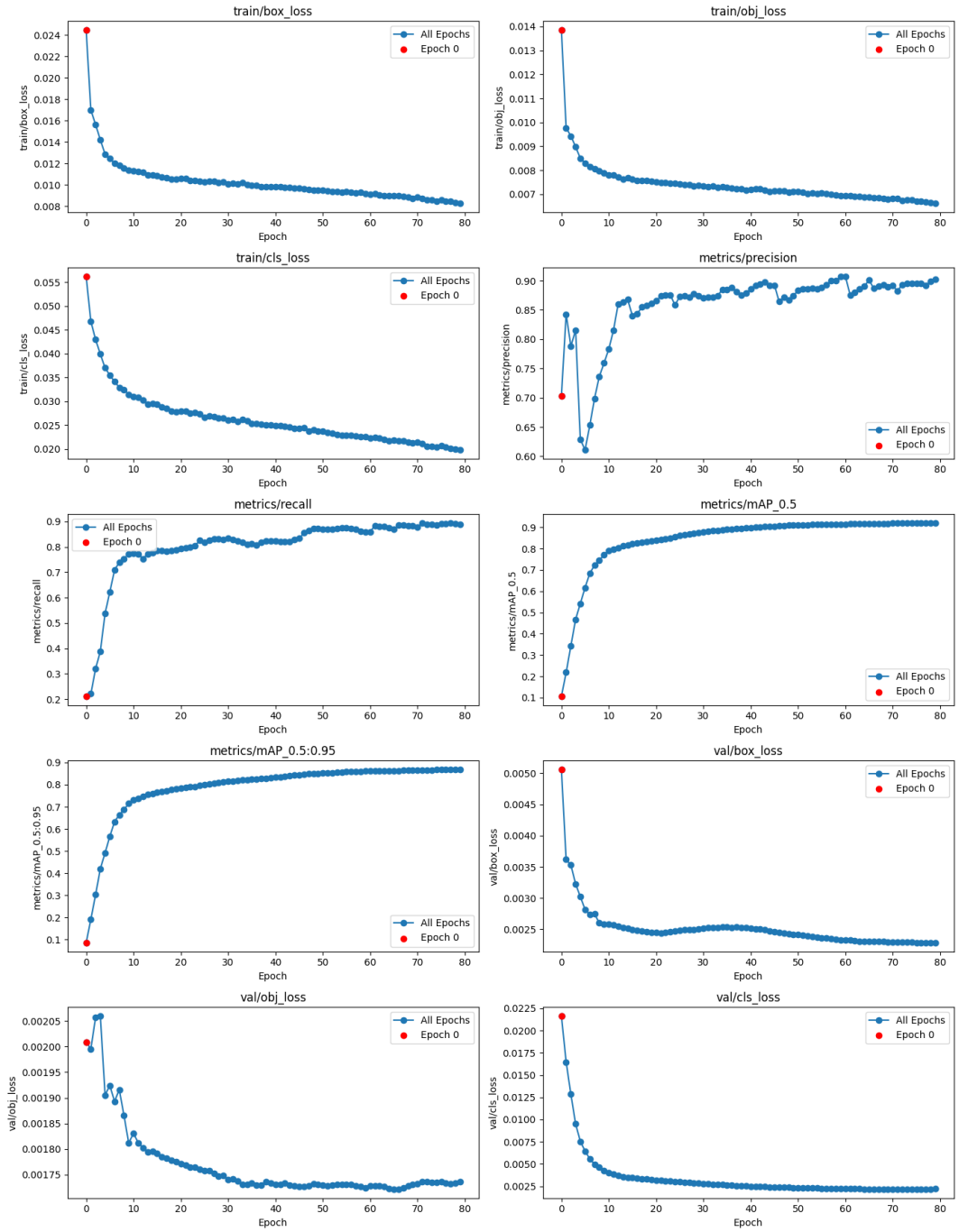

Figure 16: Training Metrics of yolov5s (SGD) model Training

.

3.2.3 YOLOv5s (Adam Optimizer)

For my project, I trained YOLOv5s on a paid Google Colab instance, utilizing a 16GB Tesla V100 GPU. The optimizer used for this training was Adam, selected due to its adaptive

learning rate updates at each epoch, which provides faster convergence than SGD in some cases.

```
lr0: 0.001
lrf: 0.1
momentum: 0.937
weight_decay: 0.0005
warmup_epochs: 3.0
warmup_momentum: 0.8
warmup_bias_lr: 0.1
box: 0.05
cls: 0.3
cls_pw: 1.0
obj: 0.7
obj_pw: 1.0
iou_t: 0.2
anchor_t: 4.0
fl_gamma: 0.0
hsv_h: 0.015
hsv_s: 0.7
hsv_v: 0.4
degrees: 0.0
translate: 0.1
scale: 0.9
shear: 0.0
perspective: 0.0
flipud: 0.0
fliplr: 0.5
mosaic: 1.0
mixup: 0.1
copy_paste: 0.1
```

Figure 17: Custom hyperparameter settings

The Adam optimiser was used instead of stochastic gradient descent, with an initial learning rate of 0.001.

Training Metrics:

train/box_loss: The consistent downward trend, flattening around 0.015, shows that the Adam optimizer helps in quickly converging towards accurate bounding box predictions, efficiently fine-tuning from the start due to adaptive learning rates.

train/obj_loss: A rapid decline followed by a gradual reduction towards 0.007 suggests that Adam's momentum allows for stable object detection learning while avoiding sharp oscillations.

train/cls_loss: The steady drop in classification loss, stabilizing at 0.015, indicates that Adam's ability to adjust learning rates has helped the model effectively learn

object categories while minimizing misclassifications.

Validation Metrics:

metrics/precision: Some fluctuations are seen, which might result from Adam's adaptive nature. While precision stabilizes around 0.75, fine-tuning learning rates or increasing the batch size could help reduce variability and increase precision consistency.

metrics/recall: The upward trend in recall (around 0.6) indicates that the model is learning to detect more true positives but still misses some objects. Adam may benefit from a lower learning rate or more epochs to further improve recall.

metrics/mAP_0.5: The steady increase in mAP, nearing 0.65, shows good object detection performance. This is expected with Adam's strong convergence properties.

metrics/mAP_0.5:0.95: Similar mAP progress under stricter IoU conditions, reaching about 0.65, further highlights the robustness of object localization, aided by Adam's adaptive optimization.

Validation Loss:

val/box_loss, val/obj_loss, val/cls_loss: All validation losses consistently decrease, showing that the model is generalizing well to unseen data. However, they are slightly higher than the training losses, suggesting slight overfitting that could be addressed with regularization or data augmentation.

Improvements:

Precision Stability: Adam's sensitivity to learning rate changes could be contributing to precision fluctuations. You may want to lower the learning rate or adjust the beta parameters to stabilize precision.

Improving Recall: To improve recall, consider increasing the number of epochs, experimenting with higher data variety, or adjusting the loss function to better emphasize missed detections.

The Adam optimizer has helped the model achieve **strong performance**, with rapid convergence and good generalization. However, **stabilizing precision** and further **improving recall** through fine-tuning could lead to even better results.

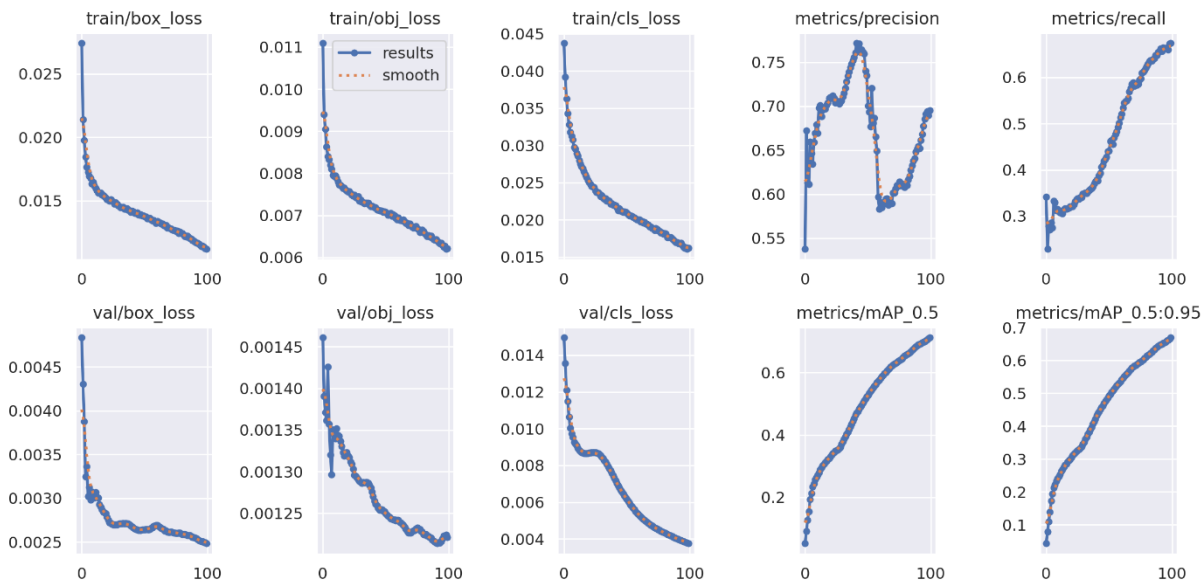

Figure 18 :Training Metrics Result of YOLOv5s (Adam_Optimizer)

4 EXPERIMENTAL RESULTS

This section presents results of the models by YOLOv5s

4.1 Model Evaluation

The model evaluations is done below:

4.1.1 YOLOv5s (Preliminary Model)

The evaluation of the model trained on the GTSRB dataset with YOLOv5s shows it to be highly effective in recognizing traffic signs. Analysis of the confusion matrix points to strong accuracy in classifying most sign types, though some classes with similar appearances occasionally get mixed up. Key performance indicators, such as the steadily decreasing loss metrics across box, abjectness, and classification losses during both training and validation phases, indicate the model's improving accuracy over time.

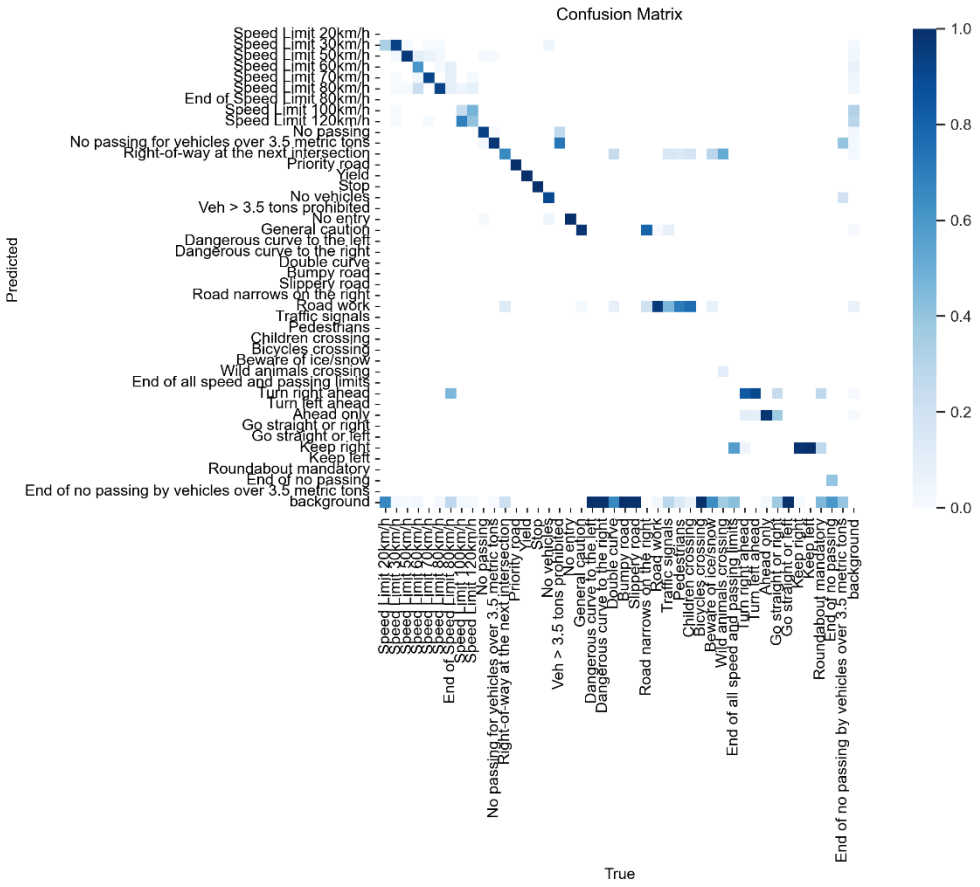

Figure 19: Confusion Matrix of Preliminary Model

Precision metrics close to perfection illustrate the model's reliability in making correct predictions, while a recall rate near 0.9 confirms its effectiveness in identifying relevant traffic signs. Mean Average Precision (mAP) scores, particularly mAP@0.5 nearing 0.95 and mAP@0.5:0.95 around 0.8, further demonstrate the model’s excellent detection capabilities at various IoU thresholds.

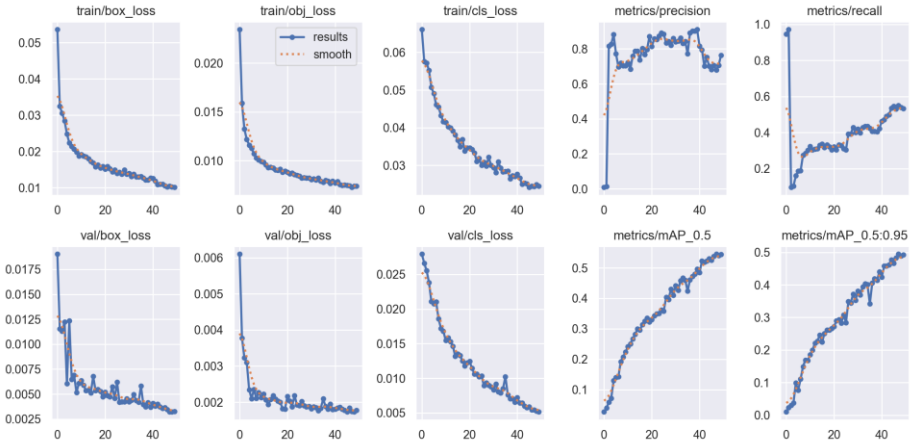

Figure 20 Training Metrics of Preliminary Model

The Recall-Confidence Curve displays changes in recall against various confidence thresholds, useful for determining the ideal confidence level for the model's predictions; in your model, recall might peak at 0.94 with a confidence setting of about 0.1. These curves collectively provide essential insights into how the model performs under varying thresholds, helping to refine the settings for best deployment outcomes.

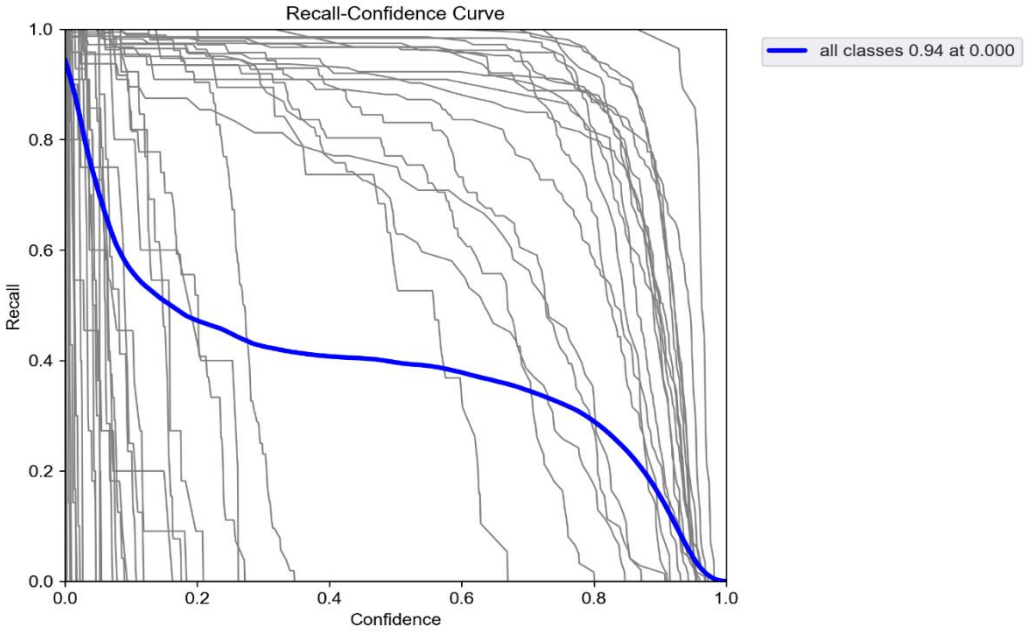

Figure 21 Recall Confidence Curve of Preliminary Model

The Precision-Recall Curve illustrates how precision juxtaposes with recall across different thresholds, highlighting the model's ability to accurately predict positive instances versus its capacity to capture all relevant instances. For example, this curve may stabilize showing a precision nearing 0.95 and recall around 0.9, demonstrating high accuracy while still identifying some positive cases.

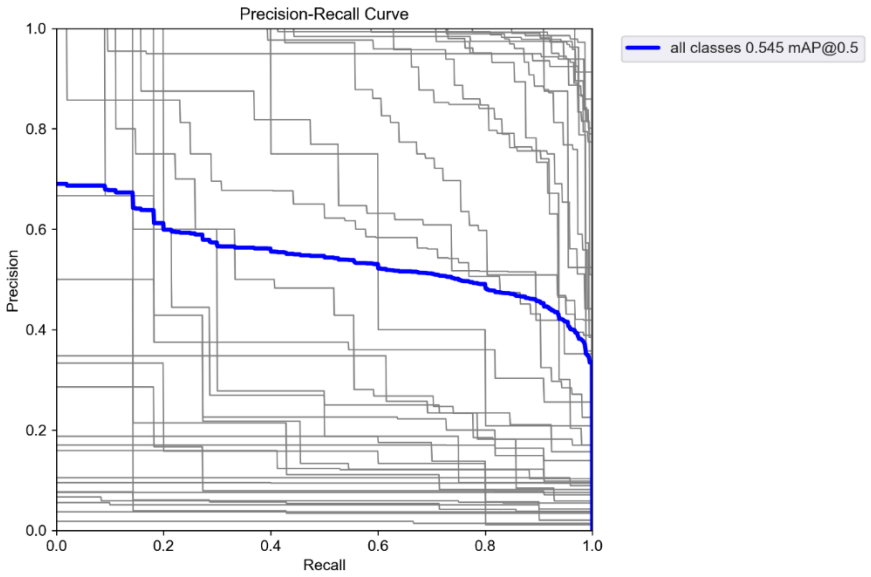

Figure 22 Precision Recall Curve of Preliminary Model

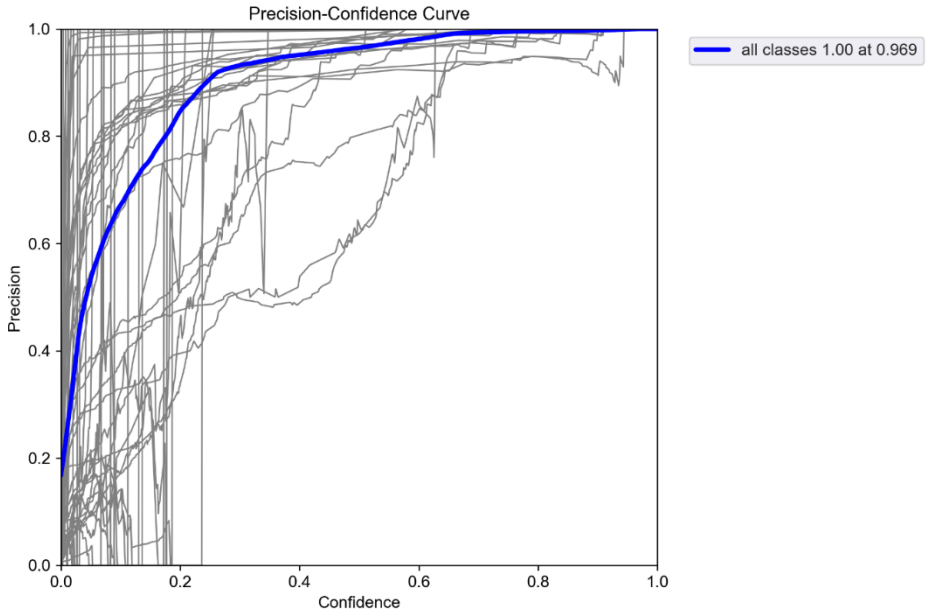

Figure 23: Precision Confidence Curve of Preliminary Model

The F1-Confidence Curve, which plots the F1 score against confidence levels, identifies where precision and recall are optimally balanced, possibly reaching an F1 score of 0.43 at a confidence of 0.133.

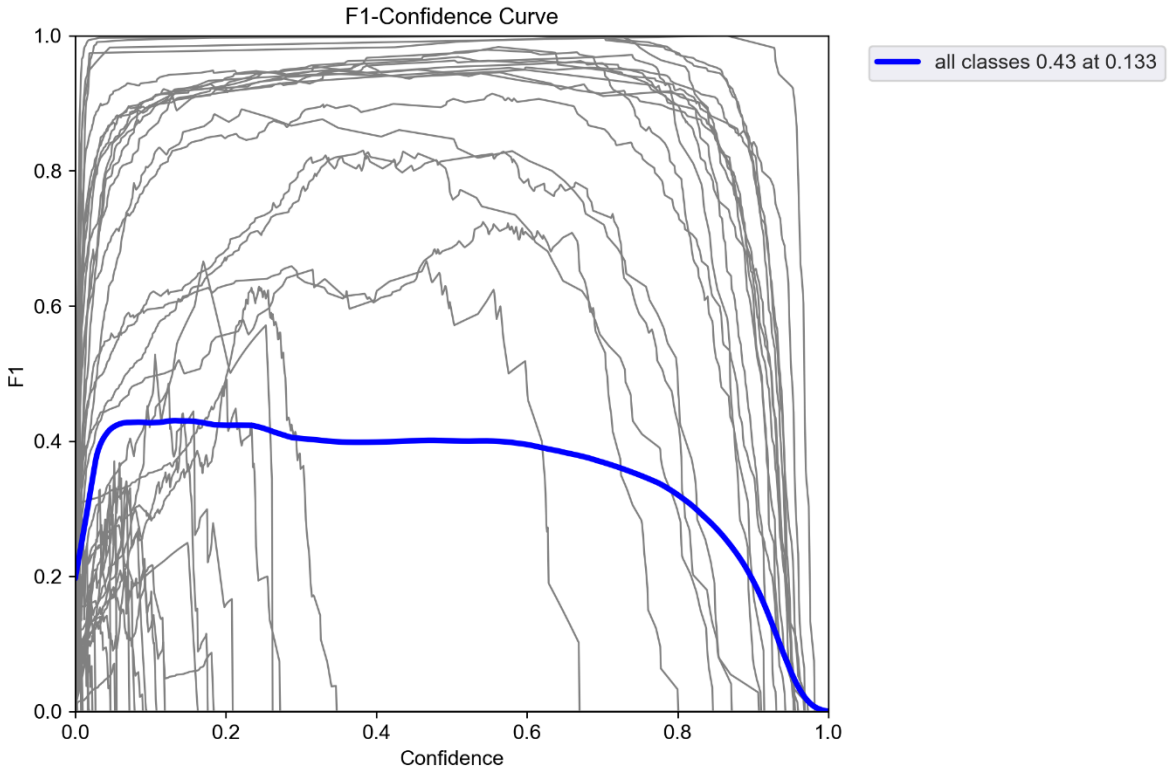

Figure 24 F1 Confidence Curve of Preliminary Model

The Figure below display a collection of traffic signs with their bounding boxes, which are likely outputs from a traffic sign recognition system. Each bounding box is color-coded to distinguish different classes of signs, such as speed limits, prohibitory, or warning signs. The clarity and correctness of these bounding boxes are crucial for accurate real-world performance of the recognition system.

Figure 25 Training batch1

4.1.2 YOLOv5s (SGD)

After 100 epochs of training, YOLO achieved 90.7% precision , 89.5% recall, 92.0% mAP@50 , 86.9% mAP@50:95. The confusion matrix for this model was not saved as colab has a runtime policy after which any data on the colab is deleted. The training metrics and training batch result is shown below in Figure 34

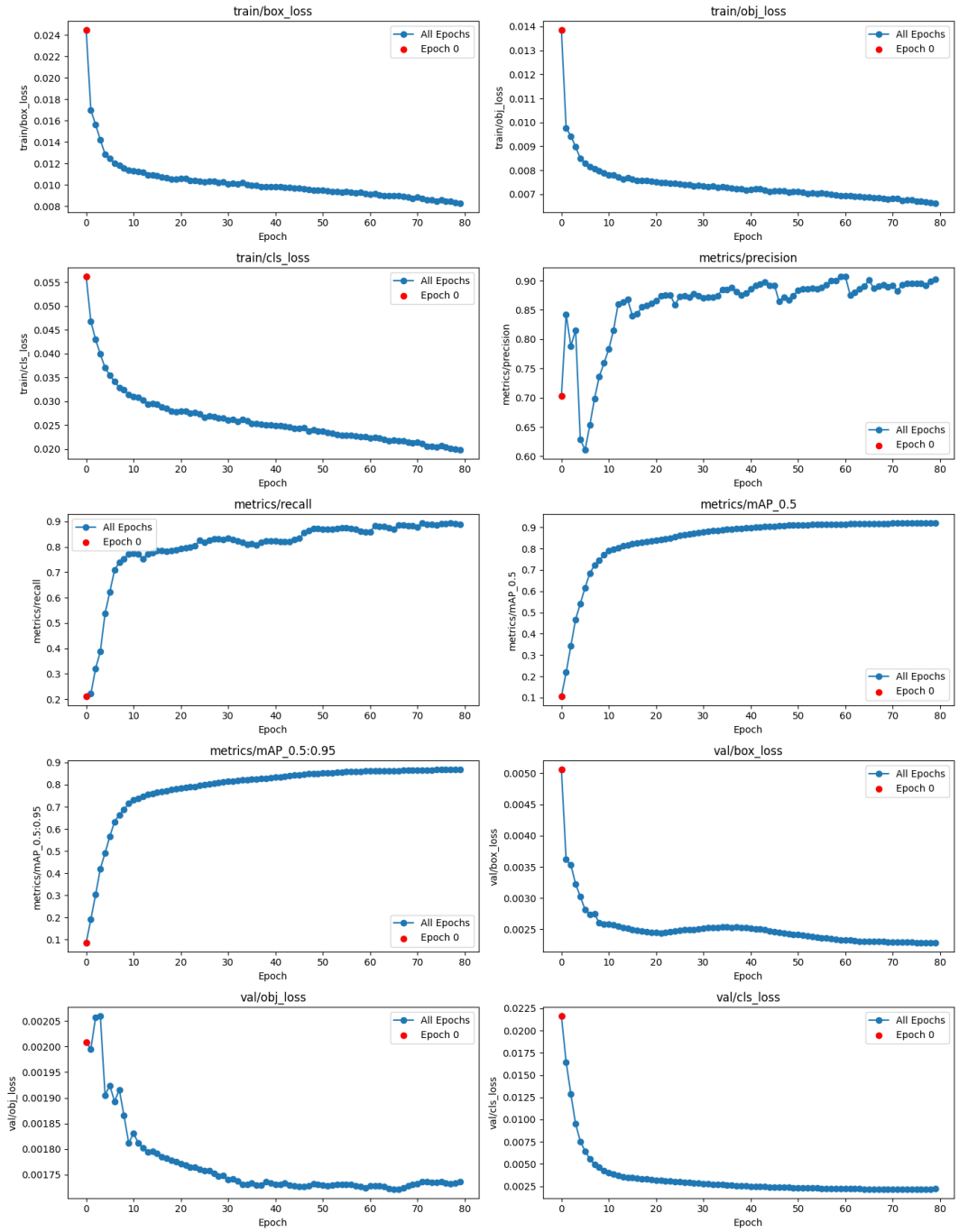

Figure 26: Training metrics results for YOLOv5s (SGD)

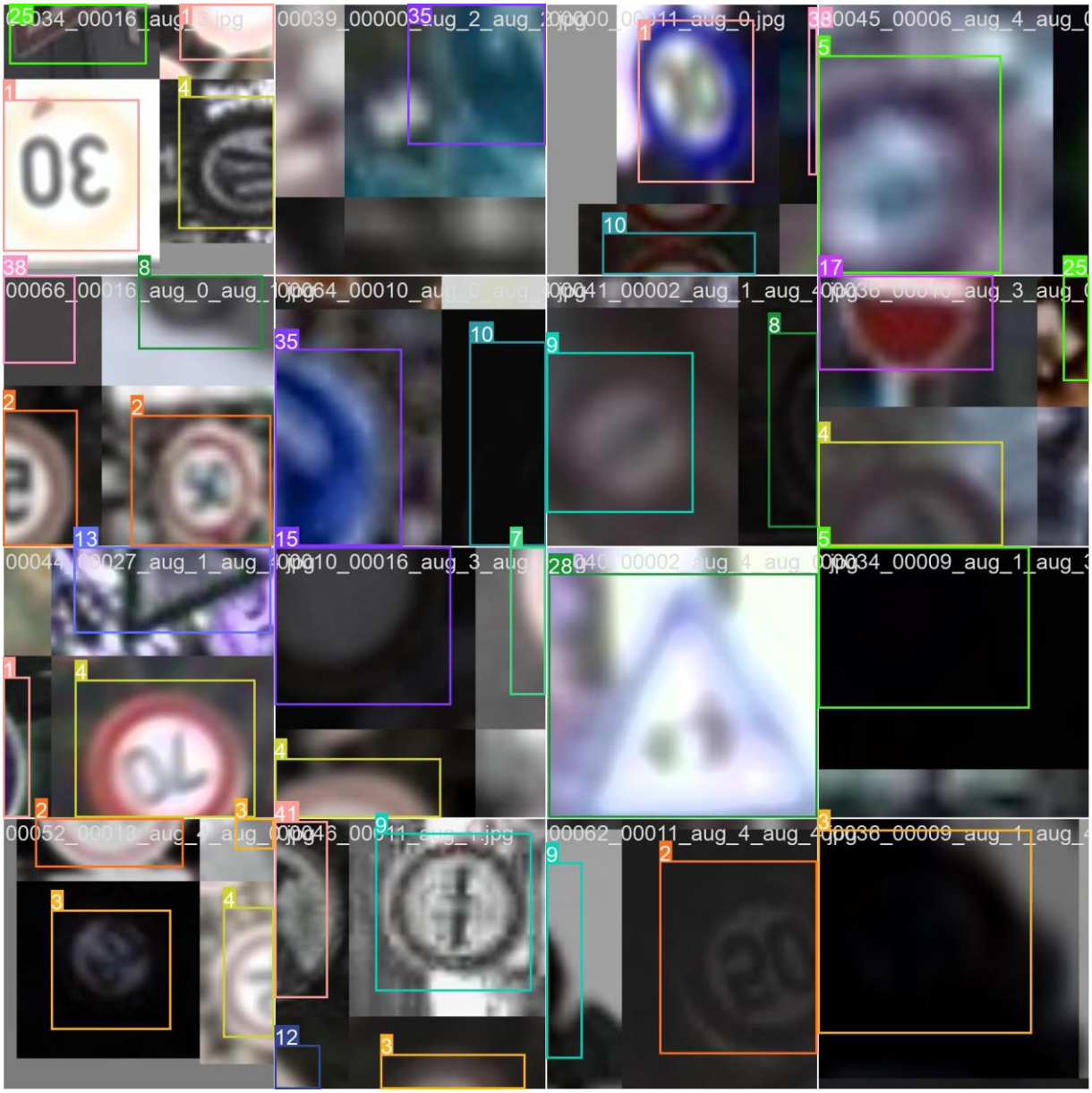

Figure Training Batch 1 Resukt of Yolov5s (SGD) Model

4.1.3 YOLOv5s (Adam Optimzer)

In addition to training and validation losses, overall precision/recall and mAP, YOLOv5 also generates confidence curves for precision, recall, and F1 scores, along with a precision-recall curve and confusion matrix. These additional metrics are useful for determining appropriate confidence thresholds at inference time, as well as isolating dataset issues (e.g., problematic or underrepresented classes). Figure 36 shows training/validation losses, along with overall precision, recall, and mAP. Results show that the model had not begun overfitting: mAP is still rising rapidly and could be improved by training for additional epochs. Overall, in just 100 epochs, YOLOv5s (Adam_Optimzer) achieved 77.2% precision

,67.4% recall,71.2% mAP@50, and 67.0% mAP@50:95 across 43 classes.

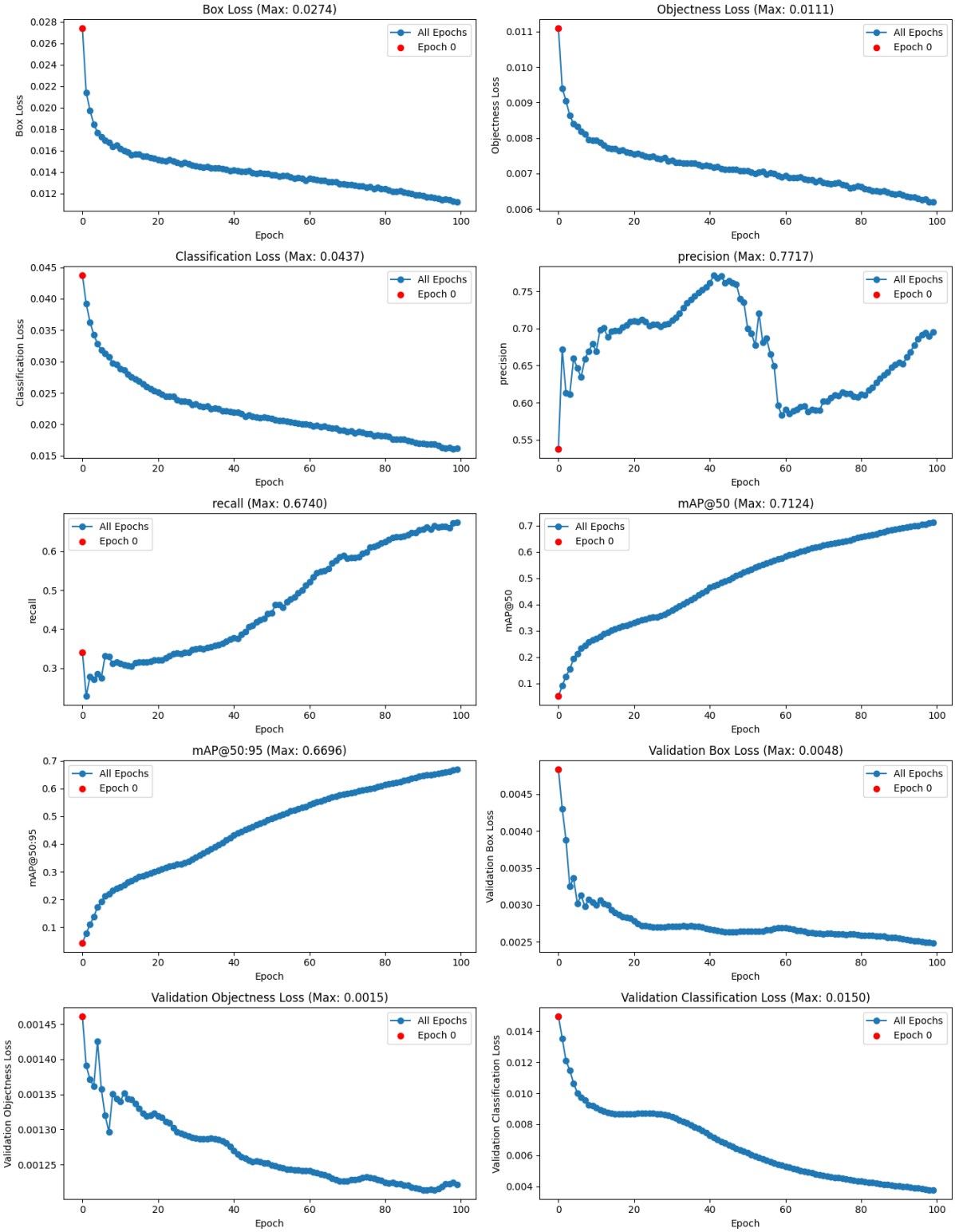

Figure 27: Training/Validation Metrics

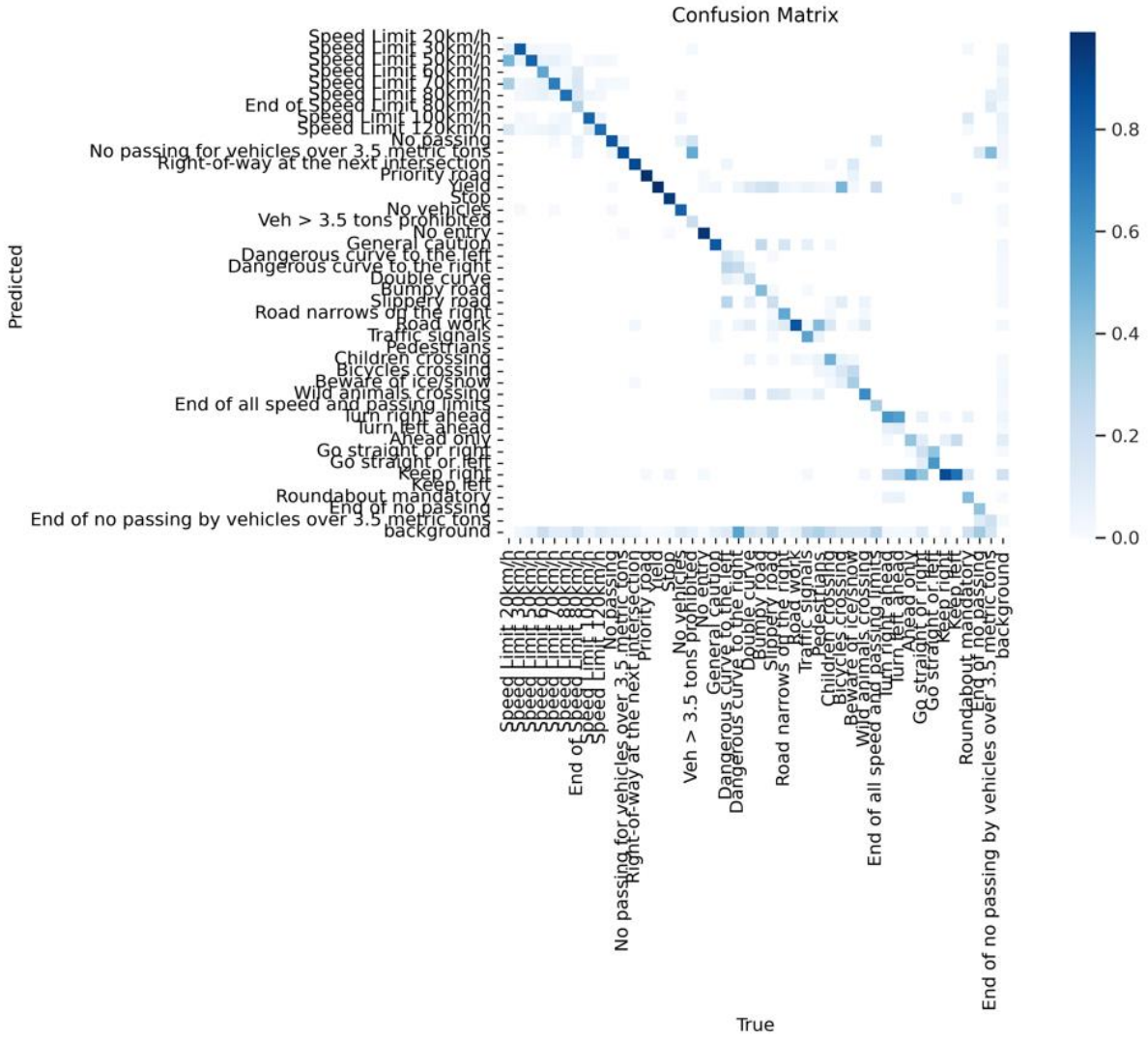

Figure 28: Confusion matrix for YOLOv5s (Adam_Optimzer)

The confusion matrix demonstrates YOLO's ability to reason about background data vs foreground objects: There are also some off-diagonal markings that suggest misclassifications, likely due to similarities in the appearance of certain signs or insufficient examples of some classes in the training dataset.

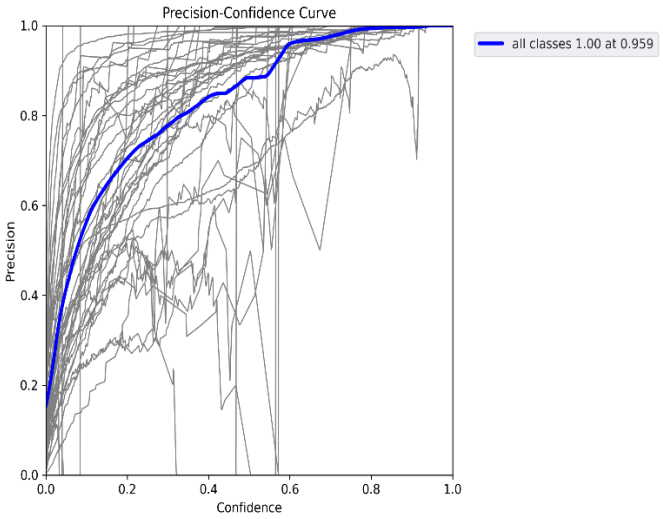

Figure 39: Precision curve

The Precision-Confidence Curve highlights how the model's precision improves significantly as the confidence threshold increases, achieving perfect precision at a confidence level of 0.959. This indicates that the model is exceptionally accurate when highly confident in its predictions. The graph shows that lower confidence levels result in more variable precision, which stabilizes and becomes consistently high as the confidence threshold approaches its maximum.

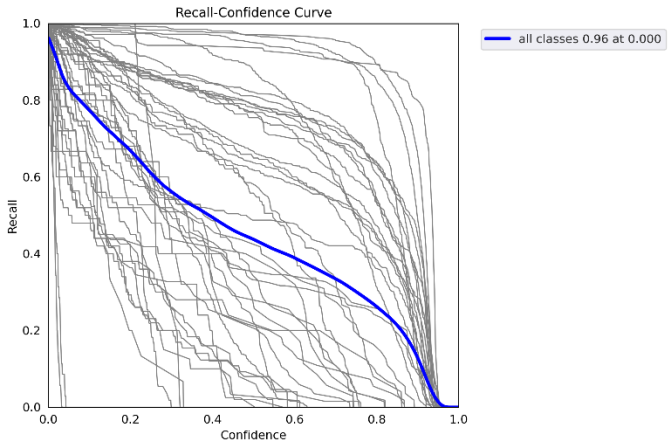

Figure 29: Recall curve

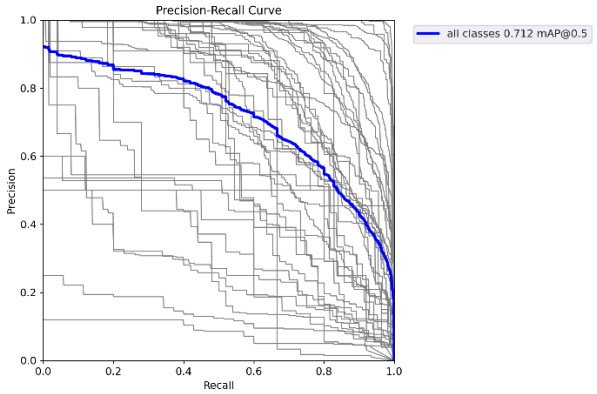

Figure 30: Precision-recall curve

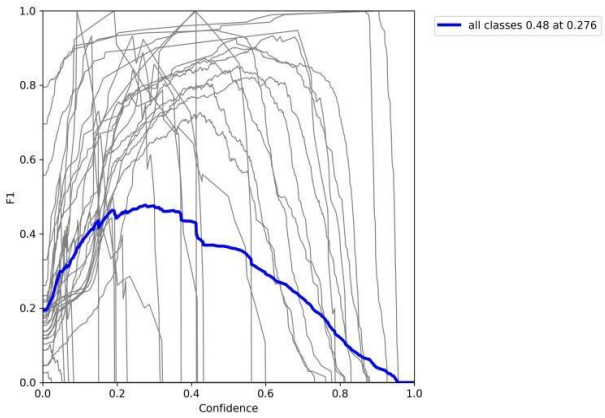

Figure 31: F1 curve

4.2 Inference Results

I only did inference on the Yolov5s (SGD) & Yolov5s(Adam_Optimzer) as both models were trained for the 100 epochs.

4.2.1 Yolov5s (SGD)

The image shows several traffic signs detected by your YOLOv5s model, which was trained using the SGD optimizer. Each sign has a confidence score displayed above it, indicating how certain the model is about its prediction. The highest confidence scores are 0.86 and 0.84, meaning the model is quite sure about these specific signs. However, some detections have lower confidence, like 0.58 and 0.44, suggesting the model is less certain about them. The green boxes likely represent correct detections, while the red boxes could indicate false positives or detections below a certain threshold. This result highlights that while the model is

performing reasonably well, there is room for improvement in recognizing certain signs.

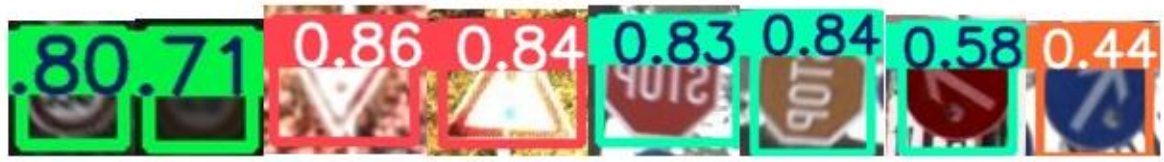

Figure 32: Varying confidence levels; all detections are accurately classified

4.2.2 YOLOv5s (Adam_Optimzer)

The results from your YOLOv5s model, which was trained with the Adam optimizer, show varying confidence levels for different traffic sign detections. In the first two images, the model has relatively high confidence in recognizing the triangular signs, with scores of 0.75 and 0.81.

Figure 33: Inference Results with high confidence levels

In the figure below, several signs are detected, but the confidence scores range more widely, from as low as 0.26 to higher scores around 0.71. The pink bounding boxes seem to indicate lower confidence predictions, while the green one could suggest a more confident detection. This suggests that while the model is performing well on certain signs, it might need more fine-tuning for cases where the confidence is lower.

Figure 34: Lower confidence levels results

5 DISCUSSION

There are few major issues with the project as conducted in this paper:

In the project, few issues were encountered with the GTSRB dataset. First, the unbalanced nature of the dataset impacted overall performance. Some classes were under-represented, with only a few images, which limited their inclusion in the train, validation, and test sets. This was evident in the confusion matrix of each model.

Secondly, we faced challenges with augmentation. While augmentations were applied to mimic real-world scenarios, they disproportionately increased the instances of already over-represented classes, reaching several thousand. Although under-represented classes also saw an increase, it was not substantial enough, leaving a prominent void in the augmentation process.

The issue of data imbalance negatively influenced both model performance and generalization. This imbalance likely led to a bias toward classes with more instances, as the model became more attuned to recognizing them compared to the under-represented classes.

Furthermore, the augmentations, though beneficial for simulating real-world conditions, failed to address the core issue of imbalance adequately. As a result, model predictions skewed toward classes with higher representation, leading to less reliable detection and classification of signs from minority classes.

To mitigate these issues, one possible solution could be to enhance the augmentation strategy. By carefully selecting augmentation techniques for the under-represented classes or augmenting only those classes that are underrepresented, we can artificially increase the sample size of these categories. This would create a more balanced training set and allow the model to generalize better across all traffic sign classes. Access to a sufficiently powerful computer with discrete hardware rather than a cloud GPU and VM instance would allow training to continue until peak performance is achieved .

The unbalanced dataset and augmentation challenges highlight the need for better data handling techniques to improve model training and evaluation. By addressing these issues, it is possible to achieve more accurate and reliable traffic sign recognition across all categories, leading to better real-time performance.

Both models were stopped early, either due to a lack of control over the training settings, or in the case of Google Colab instances, usage/runtime restrictions. Overfitting was not observed in either model using the final dataset, therefore the network complexity and number of instances were appropriate for the number of classes. The max number of epochs obtained were 100 for each model with exception of preliminary model that was run on CPU.

Exploring the performance dynamics between different YOLO models and optimizers, this project initially utilized YOLOv5s with Stochastic Gradient Descent (SGD) due to time constraints. SGD, known for not adjusting learning rates based on the gradients' second moments, potentially offers better exploration of the solution space and generalizes more effectively to unseen data. However, its slower convergence and higher demand for epochs can be less efficient, particularly with extensive datasets, necessitating precise adjustments in learning rate and other parameters like momentum.

Subsequently, the YOLOv5s model was trained using the Adam optimizer, renowned for its Adaptive Moment Estimation that tailors learning rates from the gradients' first and second moments. Adam's capacity for faster convergence proved advantageous in the initial training phase, especially where gradients show considerable variance. However, this rapid convergence requires careful management to avoid overfitting, as Adam also necessitates additional memory for the gradients' running averages, posing challenges for larger models. The Adam-optimized model reached its training completion in a day and a half, contrasting with the one-day duration for the SGD-optimized model to complete 100 epochs.

In summary, while Adam excels in achieving quick initial results with high precision, recall, and mAP, its tendency to plateau may inhibit extended learning and generalization. Conversely, SGD, with its gradual and steady performance improvement, is more suited for applications demanding long-term stability and broad generalization. This distinction makes Adam optimal for projects with immediate performance goals, whereas SGD is preferable for endeavors requiring lasting robustness and adaptability across diverse operational conditions.

6 CONCLUSION

Both models developed in this project are capable of localising and classifying traffic signs with high precision and recall. Using both networks, it was Adam is ideal for scenarios requiring swift results, while SGD is better suited for projects where enduring robustness and

broader generalization are crucial. As expected, more work is needed to expand and balance the dataset if peak performance is to be achieved. Future research should extend this line of research, by tuning hyperparameters to improve the overall convergence .The results emphasize the need for further improvements, particularly concerning dataset management. Achieving peak performance will require expanding and balancing the dataset. Future research should continue along this path, with a focus on refining hyperparameter settings to enhance model convergence and correct imbalances in the dataset.

7 REFERENCES

[1] A. Møgelmose, M. M. Trivedi, and T. B. Moeslund, "Vision-based traffic sign detection and analysis for intelligent driver assistance systems: Perspectives and survey," *IEEE Transactions on Intelligent Transportation Systems*, vol. 13, no. 4, pp. 1484-1497, 2012.

[2] C. Bahlmann, Y. Zhu, V. Ramesh, M. Pellkofer, and T. Koehler, "A system for traffic sign detection, tracking, and recognition using color, shape, and motion information," in *Proceedings of the IEEE Intelligent Vehicles Symposium*, 2005, pp. 255-260.

[3] *D. C. Cireşan, U. Meier, and J. Schmidhuber, "Multi-column deep neural networks for traffic sign classification," Neural Networks, vol. 32, pp. 333-338, 2012.*

[4] F. Pizzati, G. Allodi, A. Broggi, and S. Debattisti, "Domain-agnostic real-time perception adaptation for autonomous vehicles," *IEEE Transactions on Intelligent Transportation Systems*, 2020.

[5] J. Stallkamp, M. Schlipsing, J. Salmen, and C. Igel, "Man vs. computer: Benchmarking machine learning algorithms for traffic sign recognition," *Neural Networks*, vol. 32, pp. 323-332, 2012.

[6] J. Redmon, S. Divvala, R. Girshick, and A. Farhadi, "You only look once: Unified, real-time object detection," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2016, pp. 779-788.

[7] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "ImageNet classification with deep convolutional neural networks," *Communications of the ACM*, vol. 60, no. 6, pp. 84-90, 2017.

[8] F. Chollet, "Deep learning with convolutional neural networks," in *Deep Learning with Python*, Manning Publications, 2017, ch. 5, pp. 137-160.

[9] Y. LeCun, Y. Bengio, and G. Hinton, "Deep learning," *Nature*, vol. 521, no. 7553, pp. 436-444, May 2015.

[10] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "ImageNet classification with deep convolutional neural networks," in *Advances in Neural Information Processing Systems 25 (NIPS 2012)*, Lake Tahoe, NV, USA, 2012, pp. 1097-1105.

[11] K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition," in *Proc. of the International Conference on Learning Representations (ICLR)*, 2015.

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

[13] M. D. Zeiler and R. Fergus, "Visualizing and Understanding Convolutional Networks," D. F. e. al. Ed. Switzerland: Springer International Publishing, 2014, pp. 818-833.

[14] P. Sermanet, D. Eigen, X. Zhang, M. Mathieu, R. Fergus, and Y. LeCun, "OverFeat: Integrated recognition, localization and detection using convolutional networks," in *Proc. of the International Conference on Learning Representations (ICLR)*, 2014.

[15] S. Ji, W. Xu, M. Yang, and K. Yu, "3D convolutional neural networks for human action recognition," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 35, no. 1, pp. 221-231, Jan. 2013.

[16] A. Graves, A. Mohamed, and G. Hinton, "Speech recognition with deep recurrent neural networks," in *Proc. of the IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, 2013, pp. 6645-6649.

[17] I. Goodfellow, Y. Bengio, and A. Courville, *Deep Learning*, MIT Press, 2016, ch. 9.

[18] G. Litjens et al., "A survey on deep learning in medical image analysis," *Medical Image Analysis*, vol. 42, pp. 60-88, Dec. 2017.

[19] Y. Kim, "Convolutional neural networks for sentence classification," in *Proc. of the Conference on Empirical Methods in Natural Language Processing (EMNLP)*, 2014, pp. 1746-1751.

[20] W. Wei, M. Wahab, T. Reda, and X. Liao, "A novel CNN-based method for speech emotion recognition in human-computer interaction," *International Journal of Computers and Applications*, vol. 41, no. 2, pp. 116-123, 2019.

[21] P. Serre, A. Oliva, and T. Poggio, "A feedforward architecture accounts for rapid categorization," *Proceedings of the National Academy of Sciences*, vol. 104, no. 15,

pp. 6424-6429, 2007.

[22] J. Long, E. Shelhamer, and T. Darrell, "Fully convolutional networks for semantic segmentation," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2015, pp. 3431-3440.

[23] R. Girshick, "Fast R-CNN," in *Proceedings of the IEEE International Conference on Computer Vision*, 2015, pp. 1440-1448.

[24] T. Lin et al., "Feature pyramid networks for object detection," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2017, pp. 2117-2.

[25] J. Redmon, S. Divvala, R. Girshick, and A. Farhadi, "You only look once: Unified, real-time object detection," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2016, pp. 779-788.

[26] Z. Cai and N. Vasconcelos, "Cascade R-CNN: Delving into high-quality object detection," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2018, pp. 6154-6162.

[27] T. Lin, P. Goyal, R. Girshick, K. He, and P. Dollár, "Focal loss for dense object detection," in *Proceedings of the IEEE International Conference on Computer Vision*, 2017, pp. 2980-2988.

[28] T. DeVries and G. W. Taylor, "Improved regularization of convolutional neural networks with cutout," *arXiv preprint arXiv:1708.04552*, 2017.

[29] A. Shorten and T. M. Khoshgoftaar, "A survey on image data augmentation for deep learning," *Journal of Big Data*, vol. 6, no. 1, pp. 1-48, 2019.

[30] G. Boesch, "Image Data Augmentation for Computer Vision (2024 Guide)," viso.ai, Dec. 2, 2023. [Online]. Available: https://viso.ai/computer-vision/image-data-augmentation-for-computer-vision/.

[31] Z. Zhong, L. Zheng, G. Kang, S. Li, and Y. Yang, "Random erasing data augmentation," in *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 34, no. 07, pp. 13001-13008, Apr. 2020.

[32] M. Everingham, L. Van Gool, C. K. I. Williams, J. Winn, and A. Zisserman, "The PASCAL Visual Object Classes (VOC) challenge," *International Journal of Computer Vision*, vol. 88, no. 2, pp. 303-338, 2010.

[33] S. Ren, K. He, R. Girshick, and J. Sun, "Faster R-CNN: Towards real-time object detection with region proposal networks," in *Advances in Neural Information Processing Systems 28 (NIPS 2015)*, Montreal, Canada, 2015.

[34] A. Bochkovskiy, C. Y. Wang, and H. Y. M. Liao, "YOLOv4: Optimal speed and

accuracy of object detection," *arXiv preprint arXiv:2004.10934*, 2020.

[35] J. Bergstra and Y. Bengio, "Random search for hyper-parameter optimization," *Journal of Machine Learning Research*, vol. 13, no. 1, pp. 281-305, Feb. 2012.

[36] F. Hutter, L. Kotthoff, and J. Vanschoren, *Automated Machine Learning: Methods, Systems, Challenges*, Springer, 2019.

[37] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," in *Proceedings of the 3rd International Conference on Learning Representations (ICLR)*, 2015.

[38] S. Ruder, "An overview of gradient descent optimization algorithms," *arXiv preprint arXiv:1609.04747*, 2016.

[39] L. Bottou, "Large-scale machine learning with stochastic gradient descent," in *Proceedings of the 19th International Conference on Computational Statistics (COMPSTAT)*, 2010, pp. 177-186.

[40] S. Goyal et al., "Accurate, large minibatch SGD: Training ImageNet in 1 hour," *arXiv preprint arXiv:1706.02677*, 2017.

[41] F. Houben, J. Stallkamp, J. Salmen, M. Schlipsing, and C. Igel, "Detection of traffic signs in real-world images: The German Traffic Sign Detection Benchmark," in *Proceedings of the International Joint Conference on Neural Networks (IJCNN)*, 2013, pp. 1-8.

8 APPENDICES

Figure 35: Examples of heavy augmented images

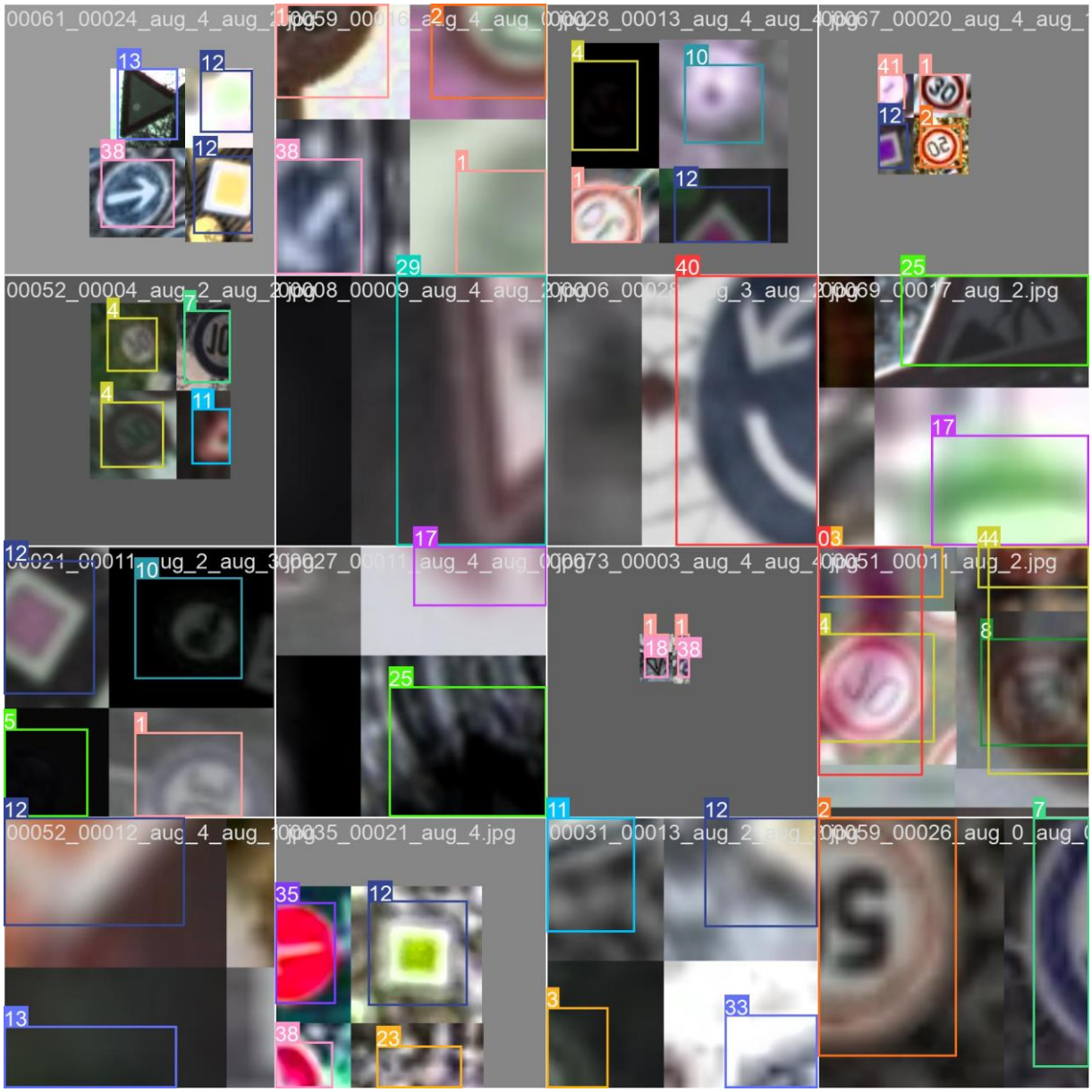

Figure 36: Batch of training images (YOLOv5s)

Figure 37: YOLOv5s (Adam) inference results Varying Confidence Levels;

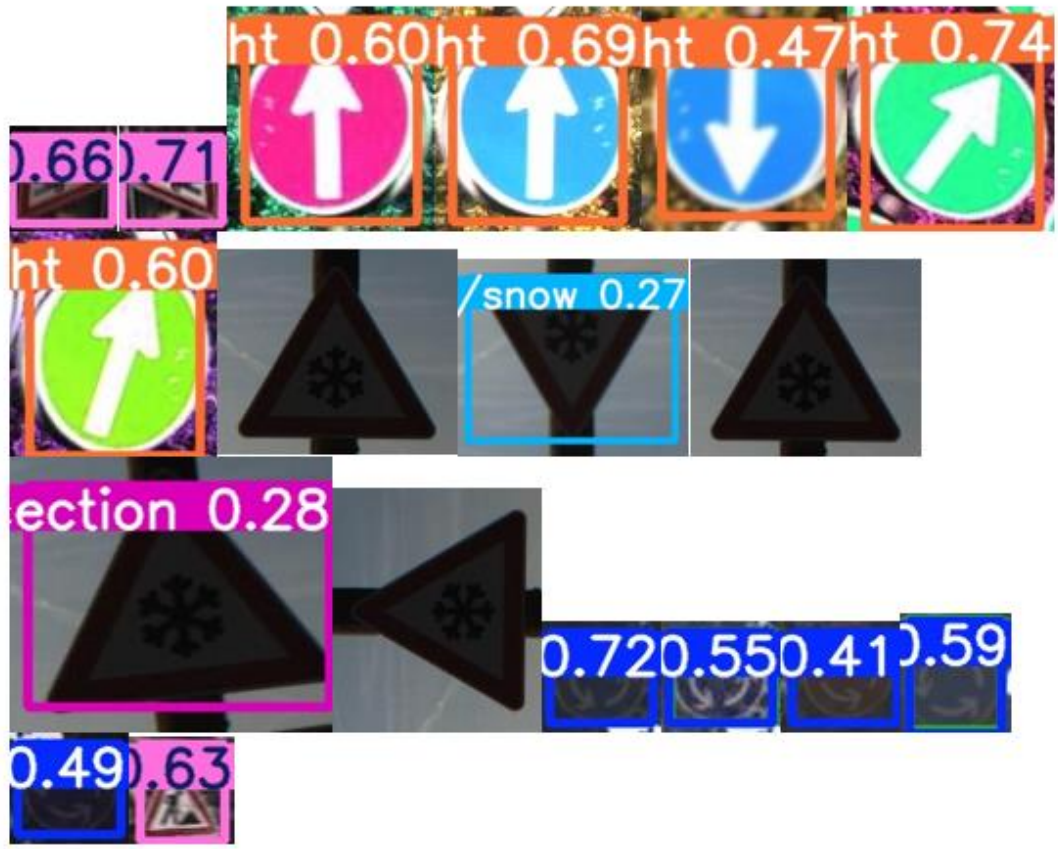

Figure 38: Yolov5s (SGD) Inference Results ; Varying Confidence Levels

Colab notebook training script:

```
 Augmentation Script
import os
import pandas as pd
import cv2
import albumentations as A
import random
from tqdm import tqdm

# Directory paths
base_dir = 'GTSRB/Images'  # Root directory containing the class directories
output_dir = 'Ablo_3'  # Directory where augmented images and updated CSVs will be
saved
max_images = 2250  # The target number of images for each class

# Augmentation pipeline with additional techniques
augmentation_pipeline = A.Compose([
    A.RandomRotate90(p=0.5),
    A.HorizontalFlip(p=0.5),
    A.VerticalFlip(p=0.5),
    A.RandomBrightnessContrast(p=0.3),
    A.GaussianBlur(p=0.3),
    A.HueSaturationValue(p=0.3),
    A.CLAHE(p=0.2),
    A.RandomGamma(p=0.2),
    A.ChannelShuffle(p=0.2),
    A.MotionBlur(p=0.2),
    A.ShiftScaleRotate(shift_limit=0.0625, scale_limit=0.1, rotate_limit=45, p=0.5)
])


def augment_and_save_image(image, img_file, class_dir, label_df, bbox, output_dir,
image_count):
    augmented = augmentation_pipeline(image=image)
    aug_img = augmented['image']

    # Keep the original format for filenames
    track_number, running_number = img_file.split('_')
    new_running_number = f'{int(running_number.split(".")[0]) + image_count:05d}'
    aug_img_name = f'{track_number}_{new_running_number}.ppm'
    aug_img_path = os.path.join(output_dir, class_dir, aug_img_name)
```

```
    # Save the augmented image
    cv2.imwrite(aug_img_path, aug_img)

    # Add the label for the augmented image to the DataFrame, keeping the format
unchanged
    new_row = pd.DataFrame({
       'Filename': [aug_img_name],
       'Width': [image.shape[1]],
       'Height': [image.shape[0]],
       'Roi.X1': [bbox[0]],
       'Roi.Y1': [bbox[1]],
       'Roi.X2': [bbox[2]],
       'Roi.Y2': [bbox[3]],
       'ClassId': [label_df['ClassId'].iloc[0]]
    })

    return new_row


def load_and_augment_images():
    # Get list of all class directories
    class_dirs = [d for d in os.listdir(base_dir) if
os.path.isdir(os.path.join(base_dir, d))]

    # Process each class directory
    for class_dir in tqdm(class_dirs):
       class_path = os.path.join(base_dir, class_dir)
       image_files = [f for f in os.listdir(class_path) if f.endswith('.ppm')]

       # Display the number of images in the current class
       print(f"Class {class_dir}: {len(image_files)} images")

       # Calculate the number of augmentations needed
       augmentations_needed = max_images - len(image_files)
       print(f"Class {class_dir} needs {augmentations_needed} more images to reach
{max_images}.")

       if len(image_files) == 0:
          print(f"Skipping class directory {class_dir} as no images were found.")
          continue

       # Load the corresponding CSV file
       csv_file_path = os.path.join(class_path, f'GT-{class_dir}.csv')
       label_df = pd.read_csv(csv_file_path, sep=';')
```

```
        # Augment images if the number is less than the maximum (2,250)
        num_images = len(image_files)
        img_id = 0
        image_count = 0

        while num_images < max_images:
            # Randomly pick an image for augmentation
            img_file = random.choice(image_files)
            img_path = os.path.join(class_path, img_file)

            # Load the image
            image = cv2.imread(img_path)

            # Extract the corresponding bounding box and class label from the CSV
            image_label = label_df[label_df['Filename'] == img_file].iloc[0]
            bbox = [image_label['Roi.X1'], image_label['Roi.Y1'],
image_label['Roi.X2'], image_label['Roi.Y2']]

            # Perform augmentation and save augmented image and label
            new_row = augment_and_save_image(image, img_file, class_dir, label_df,
bbox, output_dir, image_count)
            label_df = pd.concat([label_df, new_row], ignore_index=True)

            num_images += 1
            img_id += 1
            image_count += 1

        # Save the updated label CSV with augmented labels
        output_csv_path = os.path.join(output_dir, class_dir, f'GT-
{class_dir}_augmented.csv')
        label_df.to_csv(output_csv_path, sep=';', index=False)

        # Display completion message after finishing augmentation for the current
class
        print(f"Finished augmenting class {class_dir}.")


if __name__ == "__main__":
    # Ensure output directories exist
    for class_dir in os.listdir(base_dir):
        if os.path.isdir(os.path.join(base_dir, class_dir)):
            os.makedirs(os.path.join(output_dir, class_dir), exist_ok=True)

    load_and_augment_images()
```

```
Training Scripts
!git clone https://github.com/ultralytics/yolov5  # clone
%cd yolov5
%pip install -qr requirements.txt comet_ml  # install

import torch
import utils
display = utils.notebook_init()  # checks
from google.colab import drive
drive.mount('/content/drive')
!unzip -q Albu_Split.zip -d ../data  # unzip
#@title Select YOLOv5 🚀 logger {run: 'auto'}
logger = 'Comet' #@param ['Comet', 'ClearML', 'TensorBoard']

if logger == 'Comet':
  %pip install -q comet_ml
  import comet_ml; comet_ml.init()
elif logger == 'ClearML':
  %pip install -q clearml
  import clearml; clearml.browser_login()
elif logger == 'TensorBoard':
  %load_ext tensorboard
  %tensorboard --logdir runs/train
# Train YOLOv5s on COCO128 for 3 epochs
!python train.py --img 640 --batch 16 --epochs 100 --data gtsrb.yaml --weights
yolov5s.pt --cache  --hyp hyp_custom.yaml --optimizer Adam --project
'/content/drive/MyDrive/Adam_Optimizer' --name 'experiment_Adam'
!python train.py --resume
/content/drive/MyDrive/Adam_Optimizer/experiment_Adam3/weights/last.pt # Change it
Waleed
```