"Implementation of a Traffic Signs Detection and Recognition System for Street View Images using the "You Only Look Once" (YOLO) Deep Learning Network"

Thesis

Submitted to the

Institute of Geodesy and Geoinformation Science
Technische Universität Berlin
Fakultät VI - Planen Bauen Umwelt

In Partial Fulfillment of the
Requirements for the Degree of

Master of Science

Submitted by

Prabesh Gupta
Matriculation No.: 405565

Examiners: Prof. Dr.-Ing. Martin Kada
Prof. Dr. Marc-Oliver Löwner

Berlin, April 05, 2022

Acknowledgements

For successful completion of any project there are always helping hands to whom we must acknowledge. I too have received a good amount of support and assistance, throughout the writing of this dissertation.

Firstly, I would like to express my deepest gratitude to supervisor, professor Dr.-Ing. Martin Kada, for continuous support and guidance. This project would not have been possible without your expertise and invaluable suggestions in every steps from formulating the research topic deciding methodology to writing of this thesis. Your insightful feedback pushed me to sharpen my thinking, thanks for being my mentor throughout my master study .

I want to thank to Technische Universität Berlin for awarding me an opportunity to pursue my master‘s degree. I am fortunate to have been part of one of the prestigious and top ranked university in the world.

I would like to acknowledge Mrs. Susi Döhle, hiring manager at Landesbetrieb Geoinformation und Vermessung (LGV), Hamburg, Germany for providing ma an opportunity to collaborate and do my thesis work. I would like to thank to Mrs.Dorothee Weniger, Annette Suthau, Hauke Lehmkuhl, Jörg Schulz, Tim Michailidis and all the colleagues from Traffic data team for wonderful collaboration. They have provided me with all the data along with various support needed for successful completion my dissertation.

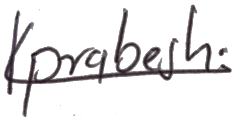
Most importantly, my parents and family deserve endless gratitude. I am grateful to their unconditional love and support throughout my journey.

Finally, many thanks to all of my friend who kept motivating and encouraging me every single time. Thank you for cheering and boosting me up through my hard times.

Declaration

I herewith formally declare that I, **Prabesh Gupta**, have written the submitted thesis independently pursuant to **Section 46 (8) AllgStuPO** of TU Berlin. I did not use any outside support except for the quoted literature and other sources mentioned in the paper. I clearly marked and separately listed all of the literature and all of the other sources which I employed when producing this academic work, either literally or in content. This thesis has not been handed in or published before in the same or similar form.

I am aware, that in case of an attempt at deception based on plagiarism, the thesis would be graded with 5,0 and counted as one failed examination attempt.



Prabesh Gupta
05 April 2022

Abstract

The field of deep learning and artificial intelligence are expanding and impacting almost every technological aspects of society. Transportation system and traffic management is one of the area where lots of research is going on to develop AI assisted automated system such as Autonomous driving (driver less car) , traffic assistance driving systems and many more can be thought of. In this context,Traffic sign detection and recognition play an important role. The main objective of this study is to explore YOLO architecture for training the model to detect and recognize traffic signs in street view images and videos of Hamburg, Germany. This paper briefly explain the advancements of YOLO family over the year for real time object detection and recognition and its suitability to address the problem of recognizing small targets in extended and complex image background.

In this paper, a complete implementation of YOLOv5 is explained from preparing data to creating model and inferencing with new data. Model was able to achieve 0.85 mAP @0.5 IoU. All other results are discussed in details and evaluation of YOLOv5 is carried based on key matrices that include mean average precision (mAP), Running time,Number of parameters of the model and the effect of size of traffic signs in image.

Abstrakt

Der Bereich „Deep Learning und der künstlichen Intelligenz“ breitet sich immer weiter aus und wirkt sich auf fast alle technologischen Aspekte der Gesellschaft aus. Verkehrssysteme und Verkehrsmanagement sind einer der Bereiche, in denen viel geforscht wird, um AI-gestützte automatisierte Systeme wie autonomes Fahren (fahrerloses Auto), Verkehrsassistenzsysteme und vieles mehr zu entwickeln. In diesem Rahmen spielt die Erkennung von Verkehrszeichen eine wichtige Rolle. Das Hauptziel dieser Studie ist die Erforschung der YOLO-Architektur für das Training des Modells zur Erkennung von Verkehrsschildern in Street-View-Bildern und -Videos von Hamburg, Deutschland. In diesem Beitrag werden kurz die Fortschritte der YOLO-Familie im Laufe des Jahres für die Echtzeit-Objekterkennung und ihre Eignung zur Lösung des Problems der Erkennung kleiner Ziele in ausgedehnten und komplexen Bildhintergründen erläutert.

In diesem Beitrag wird eine vollständige Implementierung von YOLOv5 von der Datenvorbereitung über die Modellerstellung bis hin zur Inferenz mit neuen Daten erläutert. Das Modell war in der Lage, 0,85 mAP @0,5 IoU zu erreichen. Alle anderen Ergebnisse werden im Detail besprochen und die Bewertung von YOLOv5 erfolgt anhand von Schlüsselmatrizen, zu denen die mittlere durchschnittliche Genauigkeit (mAP), die Laufzeit, die Anzahl der Parameter des Modells und die Auswirkung der Größe der Verkehrszeichen im Bild gehören.

Contents

List of Figures

List of Tables

Chapter 1. Introduction

1.1 Introduction

Traffic infrastructures (all objects in road traffic that influence and control road traffic functionality) are one of the central element of today‘s mobility. Road infrastructure is currently designed to for non autonomous vehicles. To be able to support new technologies and services related to autonomous driving, adaptation and enhancement of the capability of current traffic structures is necessary. An innovative solution is the digitization and visualization of conventional traffic infrastructures. The process of converting information into machine readable form is called as digitization. And the term visualization refers to the transformation of digital information in a such a way that it can be perceived by the users at an abstract level. From the combination of intelligent traffic systems, improved traffic information and intelligence traffic management new added values can arise which create sustainable mobility [[80], [74]].

Traffic signs are an integral part of all road infrastructure. They provide critical information like speed limit, sharp turn, road construction work or school crossing ahead which are very important for road safety and prevents accidents. These information provides road users a compelling recommendations to adjust their driving behaviour to make sure they hold fast with whatever road regulation currently enforced [62]. TSDR constitute a key component in trending real world applications, such as Autonomous vehicle (AV) technology, Intelligent speed Assistance (ISA), Blind spot detection and many more. Autonomous vehicle (AV) technology progresses from needing driver assistance to having full autonomy, driverless cars are looking more likely to become a reality. With this comes significant benefits, including increased personal safety, time saving for drivers, mobility for non-drivers, decreased environmental harm, and reduced transportation costs [83].

Intelligent Speed Assistance (ISA) is a safety feature that is embedded in vehicle to informs drivers of the current speed limit. Not only this it also acts as a speed limiter when needed, it automatically reducing a vehicle's speed by limiting engine power. The technology is estimated to lower the accidents rate by 30% and deaths by 20%. To ensure driving safer, EU legislation made ISA feature mandatory for all new vehicles starting in July 2022, and mandatory for all existing car lines per 2024. The legislation applies to all European cars, vans, trucks and buses. ISA can use cameras alone to identify speed limits through traffic sign recognition[85].

Automatic measurement and categorization of traffic signs improve traffic flow and mobility planning with automated updates visual inspection of damages, monitor their quality remotely to take timely measures and map regularly to keep information updated. The road sign information uses a camera with objection detection algorithms to detect and classify. The PASCAL VOC (visual object classes) is a standardized data format of object detection and object class recognition. In these datasets, target objects typically occupy a large proportion of each image.

The bounding box of each object of interest fills on average about 20% of the image. However, for some cases, objects of interest may only contain a small fraction of an image, such as traffic signs in images captured by camera while driving. A typical traffic sign might be say 80 × 80 pixels, in a 2000 × 2000 pixel image, or just 0.2% of the image. In fact, many tasks require detection and recognition of small but significant objects, so it is important to design and evaluate methods that perform well when the object of interest is not the main or major scene item [54].

Traffic signs were divided into different categories according to function (Prohibitory, Danger, Mandatory and others) and in each category they were further divided into sub-classes (speed limit, construction work, turning points etc) with similar generic shape and appearance but different details. This suggests traffic sign recognition should be carried out as a two phase task that is detection followed by classification. The detection step make use of shared information to suggest bounding boxes that may have traffic signs in a specific category, while the classification step uses differences to determine which particular kind of sign is present.

Since the launch of the German traffic sign detection and recognition benchmark data[[39], [40]], various research groups have made progress in both the detection benchmark (GTSDB) [29] task and classification benchmark (GTSRB) [36] task. While it may seems that these are thus solved problems, unfortunately, this benchmark data is not representative of the scene that come across in real tasks. In the GTSDB detection benchmark task, the methods must only detect traffic signs in one of four major categories. In the GTSRB classification benchmark, the traffic sign occupies most of the image, and the algorithms must only decide which subcategories the sign belongs to. Furthermore, there are no negative samples disrupting the classification. In real world scenario, the main difficulty when detecting and recognizing traffic signs in an ordinary image is, their small size often less than 1% of the image. The probable candidate regions are orders of magnitude smaller than in PASCAL VOC and ImageNet ILSVRC. Furthermore, the algorithm must filter out many potential negative cases while retaining true traffic signs. In this work, we will create a new, sample size of more realistic benchmark, and will also used it to create object detection model.

The main objective of this research is to develop an efficient TSDR system which can detect and classify traffic signs into different classes in real time environment. There are several methods and approaches that are being used for traffic sign detection and recognition. Road signs have specific properties and are analyzed in three main steps localization, detection and recognition. In localization and detection phase the image is preprocessed, enhanced and segmented according to the sign properties to separate the objects of interest from the background. The properties such as colour and shape are used in detection process. Some of the popular color and shape based method of detection are HSI/HSV Transformation [[18],[37]], Region Growing [4], Colour Indexing [2], YCbCr colour space transform [34], Hough Transformation [[28],[38],[35]], Similarity Detection [10], Distance Transform Matching [9], and Edges with Haar-like features [[21],[23]]. The final step is the recognition of the sign using a fixed database that consist of all possible traffic sign models. In the recognition stage, each of the candidates is tested against a certain set of a pattern or features .

For real time automatic detection and recognition the process need to be

fast and computationally efficient. From the above discussed process it is seen that all he stages need to be executed in sequence to come up with final recognition which is time consuming and cannot be operated in real time system. Hence, robust automatic detection and recognition system must be straightforward and ideally deep learning methods are straight forward that predicts the output directly looking into the image. There are several approaches that uses deep learning technique for the detection and recognition of traffic sign. The state-of-the-art two stage object detection algorithm Faster R-CNN [57], state-of-the-art single stage object detection algorithm Single Shot Detector (SSD) [66]. In this experiment a YOLO based approach, which is also a one stage object detection algorithm is used to develop TSDR system [75]. the latest YOLO architecture the version 5 is supposed to be very fast and accurate. At the end, the paper evaluates the performance of YOLO model based on results obtained from the experiments.

1.2 Structure of the work

The paper is organised as follows: Chapter 2 discusses some research on related works of traffic sign detection system in the past and in current scenario. Chapter 3 is all about the deep learning and neural networks. It describes the background history of YOLO family and its requirements to create model and also explain the theory related to matrices calculation, plots and graphs used for model evaluation. Chapter 4 elaborates overall methodology from preparing data to training model and inferencing in detailed. Then comes chapter 5 where all the implementation details are presented. While in chapter 6 details on the different experiments tried and the results from those experiments are described. In chapter 7 evaluation is done to the latest YOLOv5 algorithm based on the results of the experiment. And finally, chapter 8 concludes the paper and discuss future intentions for research and potential improvements.

Chapter 2. Related Work

In this chapter, the research work done prior to creating the model are introduced. The first section describes on the background history of traffic sign detection and recognition highlighting on when it begins and how it was done in past. In second section, some research work on deep learning methods are explained. At the end outcome of the research and motivation to this work is presented.

2.1 Background Of Traffic Sign Detection and Recognition

According to [25], the first work in this area can be traced back to the late 1960s, whereas according to [16] the first paper on automated traffic sign detection was appeared in Japan 1984. This concept was followed by several researchers and introduced different methods to develop an efficient TSDR system minimizing the issues that came across. The identification of the road signs was achieved by two main stages: detection and recognition. Most of the methods in detecting of traffic signs are based on color information extraction. However, the actual traffic conditions are complicated due to changes in illumination and adverse weather conditions making it difficult to choose the most proper color space to assure robust color analysis. With regards to this issue some authors comes up with shape based sign detection such as Hough Transform, ad-hoc models based on canny edges or convex hulls. Recognition of traffic signs has been addressed by a number of classification techniques from simple template matching to sophisticated Machine learning techniques like support vector machines, boosting, random forest, etc.

Color Based Sign Detection

Sign detection using color is based on the five typical colors (red, blue, yellow, white and black) defined in standard traffic signs. The first attempts to construct a real time system for automatic traffic sign recognition appears in Akatsuka et al. [1] where a look-up table in Nrgb color space is used in order to specifically design the segmentation of the speed limit signs. The changes in traffic sign colors according to the time of the day was studied by the author of [12] and deduced that the variation of the outdoor illumination does not remarkably affect the RGB component of the colors of Traffic signs. Zadeh et al. in [8] proposed a careful examination of the nature of changes of pixel values of the same color in the RGB space and designed sub-spaces corresponding to the highest rate of variation of traffic sign colors. The sub-spaces are described as canonical regions placed on straight lines from RGB (0,0,0) to the combination of primary colors corresponding to the traffic signs. The authors of [15] trained scale-specific road sign detectors using Haar wavelet features parameterized by color. The best features were selected by a cascaded AdaBoost framework from a large space of features defined over multiple color representations. Despite the great final results, one of the main cons of these methods is the high computational load that makes it more difficult to produce a

real time system. Recent work has recorded significant advances applying advanced machine learning techniques. Nguwi in [19] and author in [34] segment traffic signs pixels in YCbCr color space. Fang et al. in [13] applied a Spatio-Temporal Attentional neural network to extract information of interest related to the formation of focuses of attention. Sign detection is achieved by analyzing the combination of color and edge information.

Shape Based Sign Detection

Issues remain, such as the detection of traffic signs in cluttered scenes, in changing conditions of brightness and illumination, affine distortions according to the point of view, partial occlusions and other information attached to traffic signs. Provided the regular geometric information by the traffic signs, one of the first efforts to address the issue was to apply Hough Transform on the edge map of the region of interest [3]. In [7] potential traffic signs are identified by a template based correlation method using distance transforms, while the classification is based on radial basis function networks. machine learning approaches significantly improve the final results of traffic sign detection. In [6], the authors developed a simple method based on color thresholding and shape analysis to detect the signs, continued by a neural network to classify them.

Recognition of sign

Once the region of interest (ROI) was determined and a traffic sign was detected, the recognition was done using a predetermined database of all the traffic signs in the system. The most frequent similarity measures in traffic sign classification was normalized cross-correlation [5]. Another template matching approached was used by [11] and [14]. Inorder to differentiate between traffic signs with similar over all color ratio, model matching approach was presented in [8] which analyses the area ratio of the colors in the region of interest horizontally and vertically. This method was invariant to rotation and size or region [26]. Li et al [22] used Adabost learning containing five classical Harr wavelets and four HOG (Histogram of oriented gradient) features.

The recent approach is based on more sophisticated machine learning technique which are flexible, adaptive and robust. Support Vector Machine (SVM) is robust against illumination and rotation with a very high accuracy. Yang et al. [30] and García-Garrido et al. [27] used SVM with Gaussian Kernels for the recognition whereas Park and Kim [43] used an advanced SVM technique that improved the computational time and the accuracy rate for gray scale images. Hechri and Mtibaa [34] used a 3-layer MLP (multi layer perceptron) network whereas Sheng et al. [20] used a Probabilistic Neural Network for the classification process. Greenhalgh and Mirmehdi [[33],[32]] showed a comparison between SVM, MLP, HOG-based classifiers and Decision Trees, and found that a Decision Tree has the highest accuracy rate and the lowest computational time. Its accuracy is approximately 94.2%, whereas the accuracy of the SVM is 87.8% and that of MLP is 89.2%. [50].

2.2 Research on Deep learning methods for TSDR

As like other object detection such as face detection [[63] and [41]], iris detection [55] traffic sign detection has certain similarities to the general object detection. compared with other detection model traffic sign detection has its own particularities having the size of signs in the detected image vary differently. However, appropriate amendments and improvements makes it possible to directly apply other object detection methods to traffic sign detection.

There are so many papers and articles published describing their own way of detecting and recognizing traffic signs. Some of the authors adopted four stage architecture that include localization, detection, feature extraction and recognition [60]. Then comes methods with three stage algorithm. Three stages are defined differently in different papers. [44] defined 3 stage as segmentation, detection and recognition where they apply filters to normalized RGB channel map for segmentation, pattern recognition based on minimum rectangle for detection and matching with traffic sign database strategy for recognition. In [31] 3 stages are defined as color segmentation, shape classification and content recognition difference being the author here used SVM in the recognition step. With more advancements to deep learning and neural networks the methods stepdown to two stage approaches. The current popular method is the region based target detection. The first ever region based detection began with R-CNN [45] having three modules: region proposal, vector transformation and classification. SSP-net [47] modifies R-CNN and improve the detection accuracy. Fast R-CNN [46] introduces a multitask loss function and combines R-CNN and SSP-net that make training and testing more convenient. Faster R-CNN [49] uses the RPN (region proposal network) instead of selective search module in Fast R-CNN. This adaptation contribute to both, the speed and accuracy of detection and was greatly improved. In [68] the author slightly modifies the architecture of Faster R-CNN for detection of small traffic signs in real traffic situations. As an improvement, author combined the features from the third, fourth and fifth layer of the VGG16, unlike the Faster R-CNN that uses only the fifth layer feature of VGG16. These combination of features results in better detection of smaller targets.

And now in the recent time everything is done in single stage that does not require a region proposal process. The algorithm classify and locates object in one step making it more faster and robust. The most popular one stage architecture are YOLO (You Only Look Once) and SSD (Single Shot Detector). YOLO [52] directly regress the location of bounding box and do the categorization of the bounding box in output layer. Where as SSD [51] predict the class and coordinate offsets of a series of default bounding box using cnvolutional kernels on feature maps. In [66] the model has been simplified by author to have a better detection of smaller objects.

[87] Explore the YOLO architecture and propose a novel loss function for the YOLO model to perform better for traffic sign detection. They have used the Belgium traffic sign data set which are classified into 13 classes based on their shape and color. Likewise in 2020, there was another paper released „ Detection Small Chinese Traffic Signs Via Improved YOLOv3 Model“ [81]. They have replaced NMS (non-maximum suppression) with soft-NMS in order to reduce the occlusion effects of similar objects. However, they used data set that comes from CSUST, Chinese Traffic Sign Detection Benchmark (CCTSDB). CCTSDB originally contains

three categories but they further subdivided traffic sign into 10 new categories and relabeled them as prohibitory, mandatory, warning, speed limit, height limit, turn indicator, no sirens, no U-turns, lanes merge, and no parking and create model to detect only those 10 simplified classes.

In [54] authors firstly, created large traffic sign benchmark from 100,000 street views panorama that contains 30,000 traffic sign instances. Each traffic sign was annotated with a class label its bounding box and pixel mask. They have design their own CNN structure to perform traffic sign detection and classification and achieved betters performance in terms of recall precision and accuracy than that of Fast-RCNN. This approach does cover the individual traffic sign as a class but cannot achieve the speed that YOLO provides.

2.2.1 Outcome of Literature review and Motivation

As explained above, different approaches were explored where some uses limited categorization of traffic sign for model training. Which means similar signs are grouped together and named as one. This approach may not be useful to some application like autonomous driving where each sign need to be detected separately as single class. Where as in the paper [54] they classify each sign separately but their model is not as fast as to deploy in real time application. Considering both of scenario it was concluded that the latest YOLO model YOLOv5 might be the most appropriate for this work because of its performance and much greater development speed when moving into deployment. Also YOLOv5 is written in the PyTorch framework and is much more lightweight.

Chapter 3. Object Detection with Deep Learning

In this chapter, background knowledge and theory associated with matrices and its calculation are introduced. The first section broadly gives introduction to deep learning and neural networks to give overview about how CNN works. Second section 3.2 is all about YOLO and its advancement. It describes about the architectural changes that are made going from YOLOv1 to Latest YOLOv5. It also explains how YOLO was evolved and improved in terms of speed and performance. It is then followed by section that describes about the various requirements to develop YOLO model. And finally, this chapter ends with explaining the concept and theory about the outputs and the results generated as plots, graphs and matrices.

3.1 Introduction to Deep Learning and Neural Networks

Image classification is one of the well known computer vision task that predicts the object in image that belongs to. The class categories can be two or more depends on every individual task. Basically image classification answer what is the object in the image. Likewise, if we want to know where the object in the picture is, then localization method is used. In most real life scenarios, it is not only about one image and one object. We want to detect multiple objects in the same image and for this we have a method known as object detection that draws a bounding box around each individual object. An example could be a self driving cars that detect other vehicles, traffic lights, pedestrians, traffic signs and other entities that can influence driving behaviour.

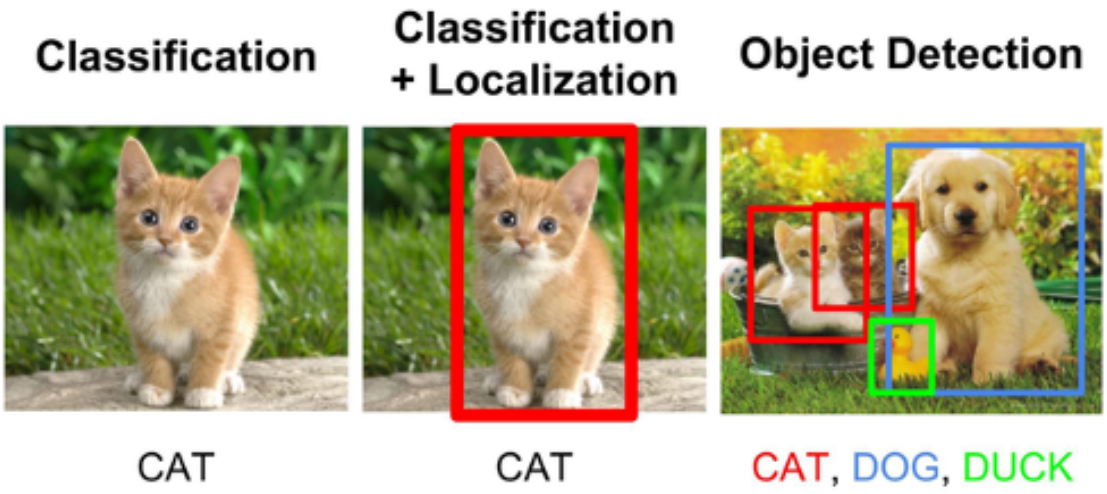
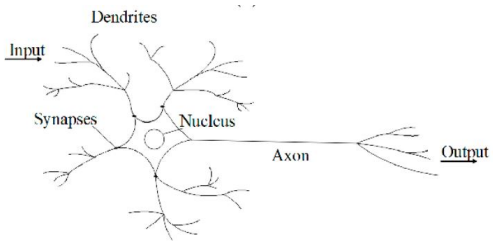


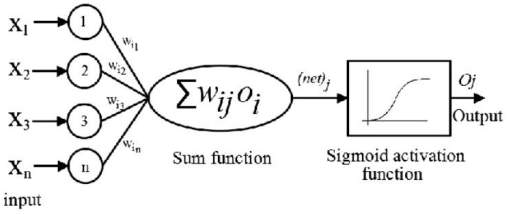
Figure 3.1: Depicting Classification, Localization and detection. source [70]

3.1.1 Convolutional Neural Network (CNN)

CNN is a deep learning architecture that is capable of doing all three aforementioned task (classification, localization and detection) and many more including segmentation. It is called as neural network because it resembles the features of the visual cortex, a region in the brain that extracts local reception fields, integrate and process visual information. It take name convolution from mathematical linear operation between matrices called convolution.



(a) resemblance to human brain



(b) structure

Neurons in the human brain are organized to process the image in layers, some having function to recognize features as edges, intersecting lines or shapes. In the same way computer only recognizes lower level features like dots, curves, bright spots and texture. So what CNN does is, it matches the part of the image instead of the entire picture. It reduces images into a form which is simpler to process, without loosing features that will later be used for object classification.

A CNN consists of an input layer, hidden layer and an output layer. Input layer an image, is a tensor with shape image height * image width * colour channels. These empty arrays are then assigned a value from 0 to 255, which describes the intensity of the pixel. This array of numbers then passes some hidden layers (convolutional layer, non-linearity layer, pooling layer) to eventually end up with a fully connected layer that results in an output as predicted class label. The convolutional and fully connected layers have parameters but pooling and non-linearity layers don't have parameters [65] [48]. Let's dive into the basics of each layer.

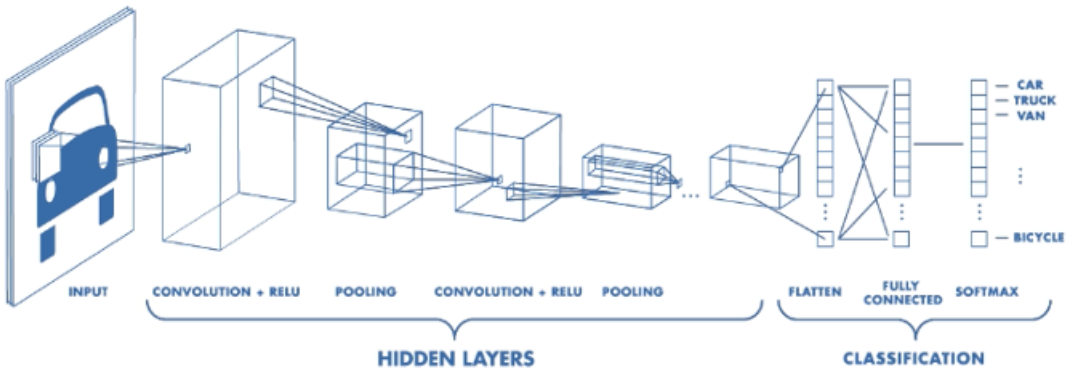


Figure 3.3: Convolutional Neural Network Architecture. source[70]

Convolutional Layer

Convolutional layers are the major building blocks in convolutional neural networks. In this layer filter is applied to the input. A filter is a two-dimensional array (usually 3x3 or 5x5) of weights. Each element of a filter is multiplied with corresponding element of filter sized patch of input, which is then summed to get a single value. This filter is slide across input repeatedly in a systematic way from left to right, top to bottom results in a two dimensional array of output values called as filter map.

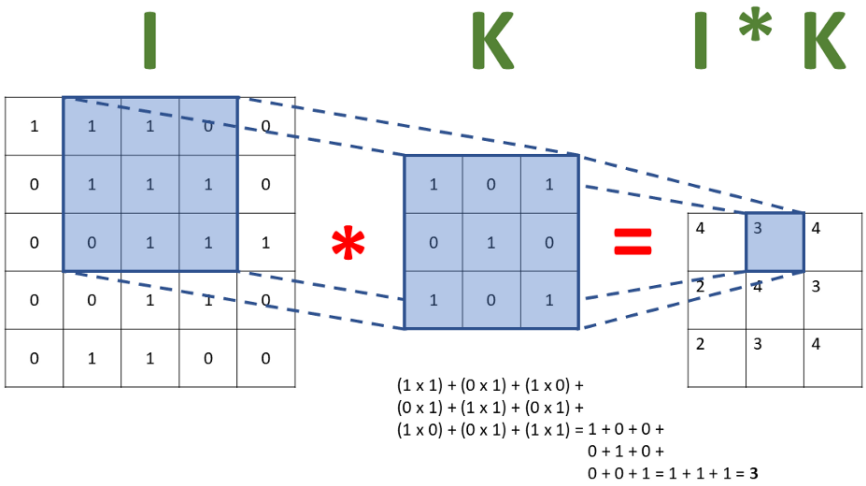


Figure 3.4: convolution Process Operation source[76]

A CNN can have one or more network of convolution layers depending on the task. To perform a simple task like edge detection, blur sharpening one layer with one filter is enough. But for an image recognition problem, need to recognize higher level of features like shapes and specific objects, where a whole network of convolution layer is needed.

Non-Linear Activation Function

The each value of resulting feature map from convolutional layer is pass through a non-linear activation function such as Rectified linear unit (ReLU). There are some other activation functions like leaky RELU, mish, swiss etc. among which few will be discussed later in the chapter. It is applied per pixel and all the negative pixel values is replaced in the feature map by 0. The function is defined as F(x) = max(0,x). The activation is basically a mathematical "gate" that determines the model's prediction. If each neuron's meets a certain threshold, it passes through the gate, if not, it is ignored. The function enables complex mappings between input and output layers in the network and brings the relevant features on the image into focus.

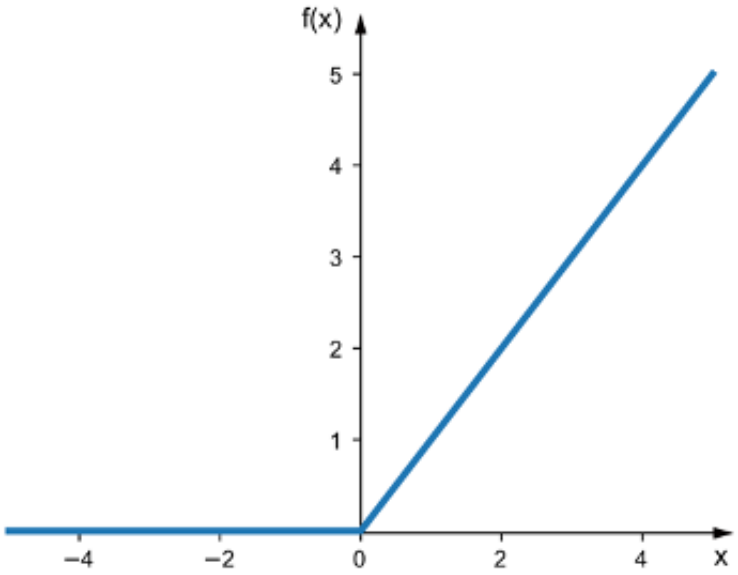


Figure 3.5: Graphical representation of ReLU source [70]

Pooling Layer

A pooling layer is a new layer added after the convolutional layer. Specifically, after a non-linearity (e.g. ReLU) has been put into the feature maps output by a convolutional layer. They are stacked in the order as, first input image then convolutional layer followed by non-linearity activation function and finally pooling. Pooling layers provide an approach to down sampling feature maps by reducing the spatial size of representation. Only the most influential features are kept. This finally decreases the amount of parameters and computation during training process enabling training of model to be fast. Also, decreased amount of parameters reduce the risk of over-fitting of the model.

Two most common methods of pooling are average pooling and max pooling. Average pooling calculate the average value for each patch on the feature map. And max pooling calculate the maximum value for each patch of the feature map. The size of the pooling filter is smaller than the size of the feature map. It is a 2*2 pixels that usually applied with a stride of 2 pixels. This means that the size of each feature map is reduced by a factor of 2.

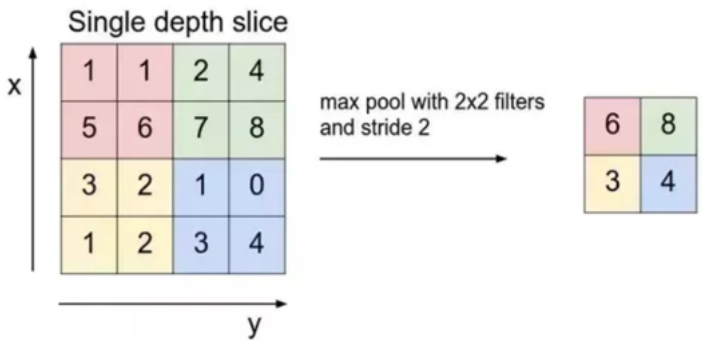


Figure 3.6: Maximum Pooling Layer. source [70]

Fully Connected Layer

The output from the pooling layer is input data to a fully connected layer. The task of fully connected layer is to flatten the value to one long vector which then combined with another fully connected layers. Each of the values in this vector represents a probability that a certain feature belongs to one of the classes. The final layer then uses the Softmax activation function or Sigmoid function to get the ultimate probabilities of the image fitting in a specific class.

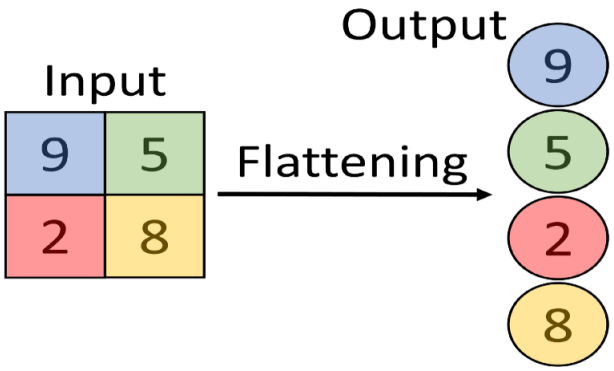


Figure 3.7: Fully connected layer.

3.2 You Only Look Once (YOLO)

YOLO is a state-of-the-art real time object detection framework that stands for You Only Look Once meaning the image is only passed once to FCNN (fully connected neural network). In terms of speed, YOLO is one of the great models in object detection and recognition. YOLO divides an image into grid system and each grid detects object within itself. YOLO was first introduced in 2016 and now it has its family. Let's get to know about each of YOLO how they evolve and developed progressively.

3.2.1 YOLOv1

The first model in YOLO family is YOLOv1. It was introduced as You Only Look Once: Unified, Real-Time Object Detection in the paper [52] released on 9 may 2016 by Redmon, Divvala, Girschick and Farhadi. The key features of the first ever YOLOv1 are:

- A single network of algorithms predicts bounding boxes and class probabilities directly from full image in one evaluation. Which means feature extraction and target localization along with classification head were unified into single monolithic block. Instead of cutting out areas with high probability for an object and feeding them to a algorithm (network) that finds boxes.
- The inference time for base model was at 45 FPS (frame per seconds) and smaller version of the network named Fast YOLO process images in real time at an 155 FPS, but fast YOLO is slightly less accurate.

- According to paper[52] as compared to other state-of-art detection system, YOLO makes more localization error, but its less likely to predict false positive on background.
- YOLOv1 has relatively low recall compared to region proposal based methods.

Yolov1 Architecture

It is basically convolutions all the way down with the occasional maxpool layer. The initial convolutional layers of the network extract features from the image while the fully connected layers guess (predict) the output probabilities and coordinates. The YOLOv1 is built on light-weight Google's MobileNet backbone. The base model network has 24 convolutional layers and then followed by 2 fully connected layers. It simply use 1×1 reduction layers followed by 3×3 convolutional layers, similar to Lin et al [42]. The final output of our network is the $7 \times 7 \times 30$ tensor of predictions. The full network structure is shown in Figure 3.8. Where as, Fast YOLO uses a neural network with lesser convolutional layers 9 instead of 24 and fewer filters in those layers.

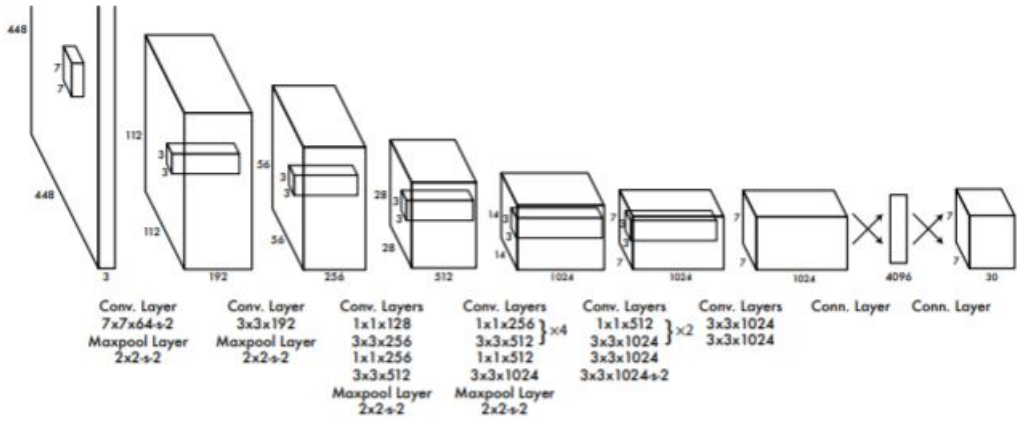


Figure 3.8: Architecture of YoloV1. Source [52]

3.2.2 YOLOv2

On 25th December 2016 YOLOv2 and YOLO9000 was introduced by J. Redmon and A. Farhadi in the paper titled YOLO9000: Better, Faster, Stronger [56]. YOLOv2 focus mainly on improving recall and localization error while maintaining classification accuracy and the speed of the architecture. YOLO9000 used YOLOv2 architecture, but was able to detect more than 9000 classes in real time. Both architecture can be run at a variety of image sizes to provide a smooth trade off between speed and accuracy. The authors have made the following Improvements over Basic YOLO:

- Added the batch normalization to the architecture that increases the convergence of the model leading to faster training. It was also observed that addition of batch normalization increase in mAP by 2% compared to basic YOLO.

- Uses of high resolution classifier so that network parameter can fit to any input resolution. This does not necessarily mean that the network will perform well on any resolution, a resolution augmentation routine was employed during training.
- The author removed the fully connected layer at the end responsible for predicting bounding boxes and replace it with anchor boxes prediction. A fully convolutional network with anchor boxes increase recall without changing mAP.
- Switching to the new network reduce computation by 33%.
- YOLOv2 is an improved model on standard task like PASCAL VOC and COOC detection data. At 67 FPS, YOLOv2 gets 78.6 mAP on VOC 2007 better than the models like Faster R-CNN and SSD.

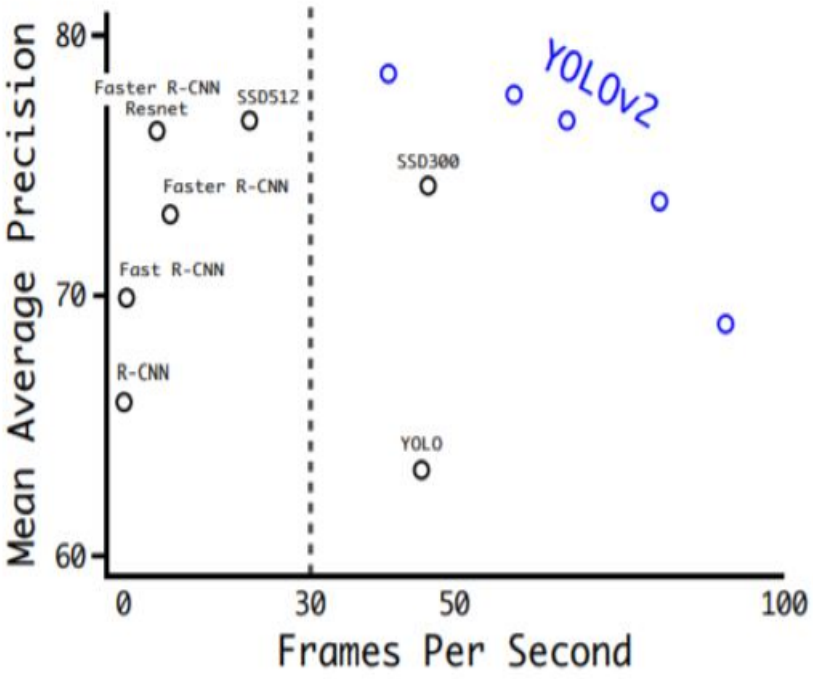


Figure 3.9: Accuracy and Speed on VOC 2007. Source[56]

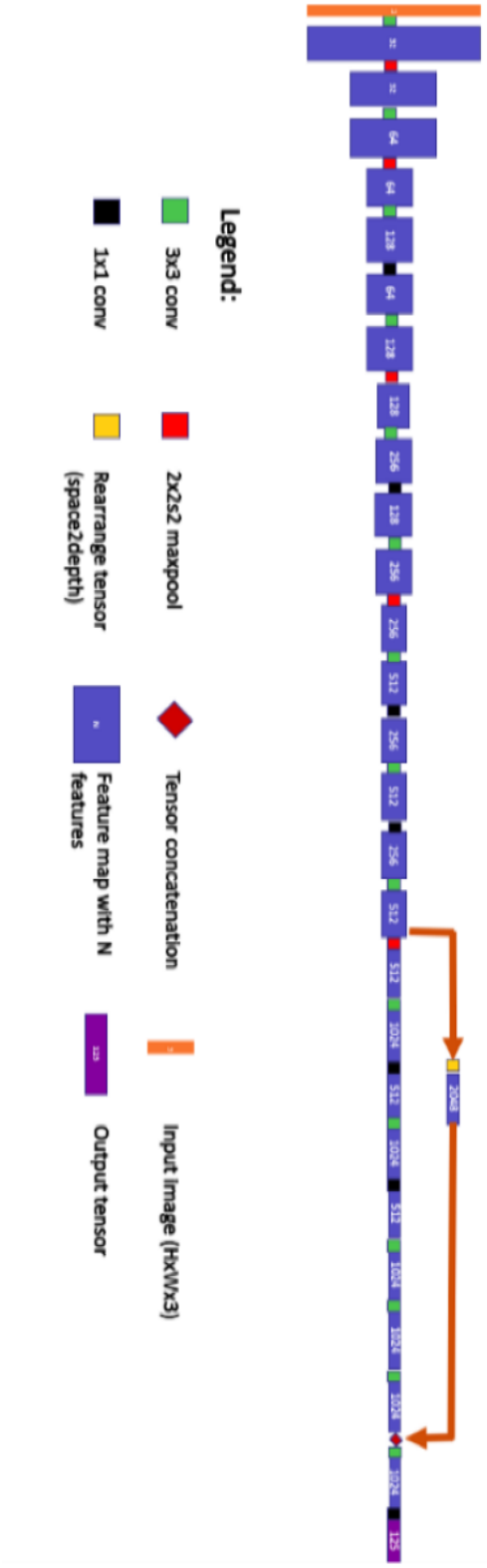
Yolov2 Architecture

YOLOv2 was trained on different architectures such as VGG-16 and GoogleNet and Darknet-19. The reason for choosing the Darknet architecture was its lower processing requirement than other architectures i:e 5.58 FLOPS (floating point operations per seconds) as compared to 30.69 FLOPS on VGG-16 for 224 * 224 image size and 8.52 FLOPS in customized GoogleNet. FLOPS are often used to describe how many operations are needed to run a single instance of a given model.

YOLOv2 Contains 22 convolutions and 5 maxpool operations. For VOC it predict 5 boxes with 5 coordinates [tx, ty, tw, th, to (objectness score)] each with 20 classes per box resulting to number of filters is 125.

Type	Filters	Size/Stride	Output
Convolutional	32	3 x 3	224 x 224
Maxpool		2 x 2/2	112 x 112
Convolutional	64	3 x 3	112 x 112
Maxpool		2 x 2/2	56 x 56
Convolutional	128	3 x 3	56 x 56
Convolutional	64	1 x 1	56 x 56
Convolutional	128	3 x 3	56 x 56
Maxpool		2 x 2/2	28 x 28
Convolutional	256	3 x 3	28 x 28
Convolutional	128	1 x 1	28 x 28
Convolutional	256	3 x 3	28 x 28
Maxpool		2 x 2/2	14 x 14
Convolutional	512	3 x 3	14 x 14
Convolutional	256	1 x 1	14 x 14
Convolutional	512	3 x 3	14 x 14
Convolutional	256	1 x 1	14 x 14
Convolutional	512	3 x 3	14 x 14
Maxpool		2 x 2/2	7 x 7
Convolutional	1024	3 x 3	7 x 7
Convolutional	512	1 x 1	7 x 7
Convolutional	1024	3 x 3	7 x 7
Convolutional	512	1 x 1	7 x 7
Convolutional	1024	3 x 3	7 x 7
Convolutional	1000	1 x 1	7 x 7
Avgpool		Global	1000
Softmax			

(a) colvolution layer numbers



(b) structure

Figure 3.10: Darknet-19 Architecture. Source [71]

3.2.3 YOLOv3

Two years after the introduction of YOLOv2, in the year 2018 third version of YOLO [61] was released with the further improvements in model architecture and training process. The author refined the design further to predict an objectness score for each bounding box using logistic regression. Earlier softmax activation was used to predict class for multi-label classification, but in YOLOv3 it uses independent logistic classifier and binary class entropy for the class prediction. Multi-scale prediction is another improvement made in YOLOv3.

Yolov3 Architecture

YOLOv3 uses a new network known as Darknet-53 for performing feature extraction. It was inspired by architecture like ResNet (Residual Network) and FPN (feature pyramid network). A feature pyramid is a topology in which a feature map gradually reduces in spatial dimension but increases again and is concatenated with previous feature maps with corresponding sizes [82]. It has 52 convolutional layer with skip connection like ResNet (making it more powerful than Darknet-19 and more efficient than competing backbones ResNet-101 or ResNet-152) and 3 prediction head like FPN each processing the image at different spatial compression.

	Type	Filters	Size	Output
	Convolutional	32	3 × 3	256 × 256
	Convolutional	64	3 × 3 / 2	128 × 128
1×	Convolutional	32	1 × 1	
	Convolutional	64	3 × 3	
	Residual			128 × 128
	Convolutional	128	3 × 3 / 2	64 × 64
2×	Convolutional	64	1 × 1	
	Convolutional	128	3 × 3	
	Residual			64 × 64
	Convolutional	256	3 × 3 / 2	32 × 32
8×	Convolutional	128	1 × 1	
	Convolutional	256	3 × 3	
	Residual			32 × 32
	Convolutional	512	3 × 3 / 2	16 × 16
8×	Convolutional	256	1 × 1	
	Convolutional	512	3 × 3	
	Residual			16 × 16
	Convolutional	1024	3 × 3 / 2	8 × 8
4×	Convolutional	512	1 × 1	
	Convolutional	1024	3 × 3	
	Residual			8 × 8
	Avgpool		Global	
	Connected		1000	
	Softmax			

Figure 3.11: Darknet-53 all layers. Source[61]

Backbone	Top-1	Top-5	Ops	BFLOP/s	FPS
Darknet-19	74.1	91.8	7.29	1246	**171**
ResNet-101	77.1	93.7	19.7	1039	53
ResNet-152	**77.6**	**93.8**	29.4	1090	37
Darknet-53	77.2	**93.8**	18.7	**1457**	78

Table 3.1: Comparison of backbones. Accuracy, billions of operations (Ops), billion floating-point operations per second (BFLOP/s), and frames per second (FPS) for various networks – Source[61]

This restructuring and little changes lead to an improvement in speed and accuracy. The comparison is shown in the figure below:

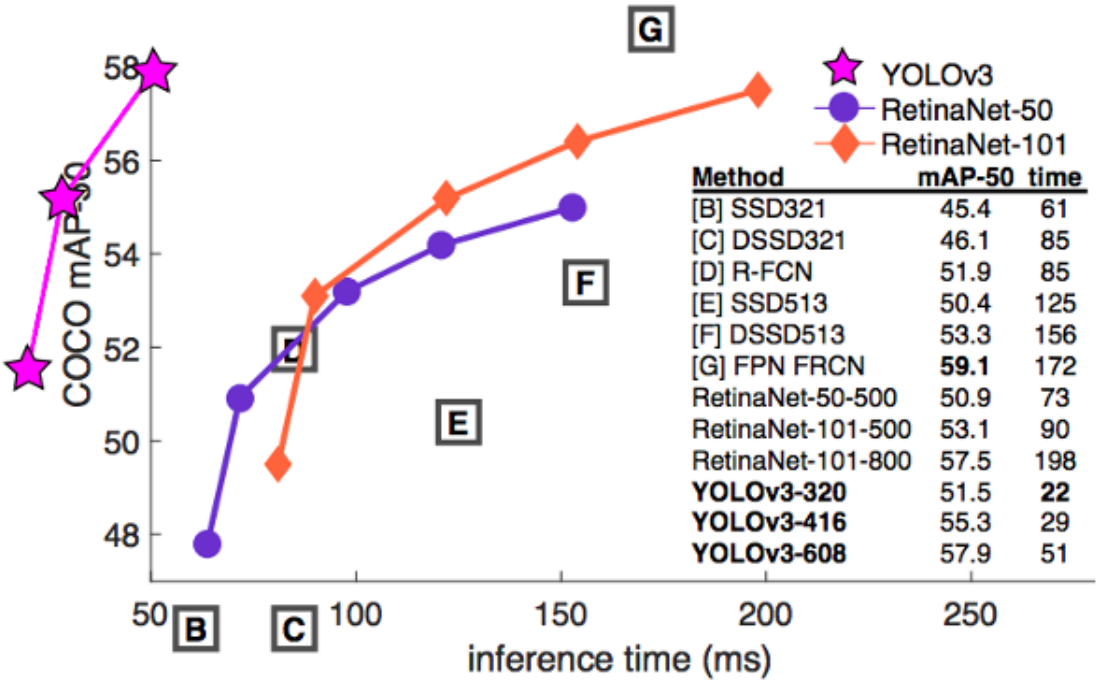


Figure 3.12: Displaying speed/accuracy trade-off on the mAP at .5 IOU metric. Source[61]

3.2.4 YOLOv4

Unlike the previous version, the fourth version of YOLO is developed by different group of author after the creator of YOLO, Joseph Redmond stopped his research in computer vision in February 2020. YOLOv4 [72] was released by three developers Alexey Bochkovskiy, Chien-Yao Wang, and Hong-Yuan Mark Liao in April 2020. This development was inspired from the state-of-art Bag of Freebies (BoF) and Bag of Specials (Bos) that improved the mAP by 10% and FPS by 12%. than YOLOv3.

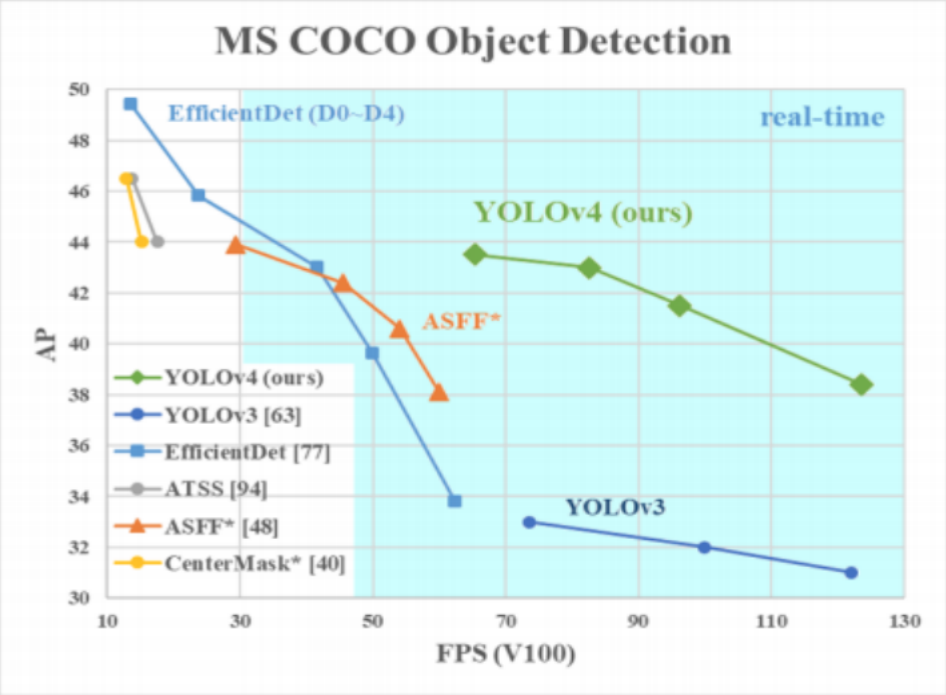


Figure 3.13: Comparison of the proposed YOLOv4 and other state-of-the-art object detectors. Source[72]

Bag of Freebies

Bag of freebies (BoF) are the set of methods that only increase the cost of training or change the training strategy without increasing the cost of inference. The methods adopted in YOLOv4 that meets the definition of bag of freebies are Data Augmentation, DropBlock Regularization, Label Smoothing, complete IOU loss (CIOU) and Self-Adversarial Training (SAT).

Data Augmentation: Data augmentation is a method to increase the number of data bringing in some variation to existing images in order to improve the generalization of the training model. The details about its type and how it works is discussed later in this chapter in section YOLOv5.

DropBlock Regularization: Deep neural networks have huge amount of parameters. Dropping out some of the features (deactivating certain neurons) during training is known as dropout. The main drawbacks of dropout was that, it drops out feature randomly which is less effective to to convolutional layers where features are correlated spatially. When the features are correlated, even with dropout, information about the input can still be sent to the next layer. Therefore, a concept of dropblock was introduced as solution. In dropblock structured form of dropout, that is features in block (a contiguous region of a feature map) are dropped together as shown in 3.14. As dropblock rejects features in a correlated area, the networks must look elsewhere for evidence to fit the data [see 58].

Label Smoothing: Confidence value of 1 that is 100% sure in a prediction may revel that the model is overfitting. If the model learns to allocate full probability to the ground truth label for each training instances, it is not guaranteed to generalize [see 53, page 6]. Therefore, adjusting the target upper bound of the prediction to a

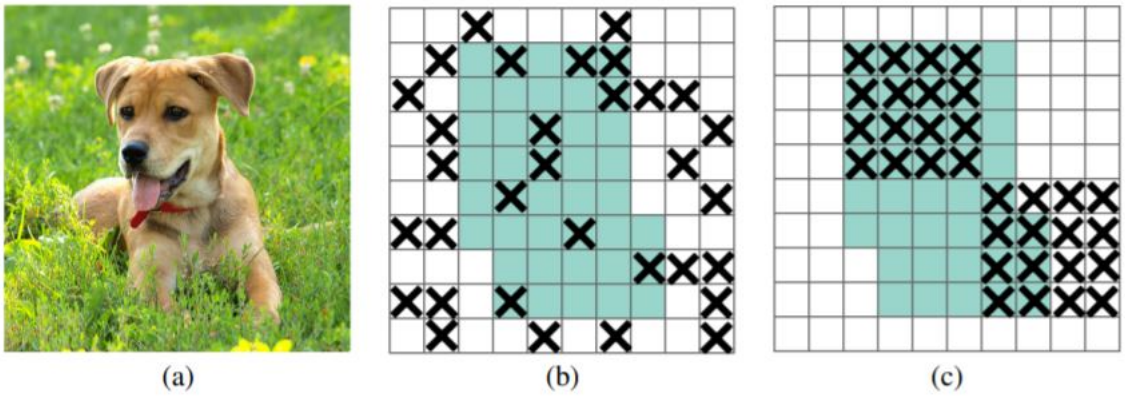


Figure 3.14: (a) input image to a convolutional neural network. The green regions in (b) and (c) describe the activation units that have semantic information in the input image. Dropping out those activation's at random is not helpful in removing semantic information because nearby activation's contain closely related information. Instead, dropping continuous regions can remove certain semantic information e.g. head or feet and consequently enabling remaining units to learn features for classifying input image. Source:[58]

slightly lower (say 0.9) to mitigate over fitting is known as label smoothing.

Complete IOU Loss (CIOU): To evaluate the quality of a model ln-norm loss function are usually adopted in bounding box regression. But they are sensitive to scales variation. To addressed this problem IOU loss was introduced that consider the bounding box as unit and are invariant to scale. Then the generalized IOU (GIOU) loss have been proposed to benefit the IOU metric, but still bear from the problem of low convergence and inaccurate regression. Recently complete IOU (CIOU) loss is proposed incorporating the overlapping area, distance between center e points and the aspect ratio leading to faster convergence and better performance [see 79].

Self-Adversarial Training (SAT): Adversarial Training is a new data augmentation technique. The objective of SAT is to maximize the mutual information (MI) between the representations of clean images and their corresponding adversarial examples, so the learned feature can mitigate the effect of adversarial perturbation. This technique operates in two stages. First generating adversarial examples, maximizing the MI. In second part, noise contrast estimator is utilized to estimate MI. Finally the model is updated by minimizing the opposite value of estimated MI [73].

Bag of Specials

Bag of Specials (BoS) are the set of post processing methods which increase inference cost by a small amount but can greatly improve the performance of object detection. The methods used are mish activation, multi-input weighted residual connections (MiWRC), Spatial pyramid pooling (SPP), Spatial attention Module (SAM) and Path aggregation Network (PAN).

Mish Activation: The activation function is key to introducing non-linearity in the network. The reason of introducing non linearity in the network was already

explained above in the subsection Non-linearity activation function. Over so many activation functions, only a few are widely used such as sigmoid, RELU, Leaky RELU, Tanh etc. Among them mish is a novel smooth and non-monotonic neural activation function which can be defined as:

$$f(x) = x.Tanh[\zeta(x)]$$

where,$\zeta(x) = ln(1 + e^x)$ is the softplus activation function. The properties of mish being unbounded above, bounded below, smooth and non monotonic, all play a significant role in the improvement of results [67].

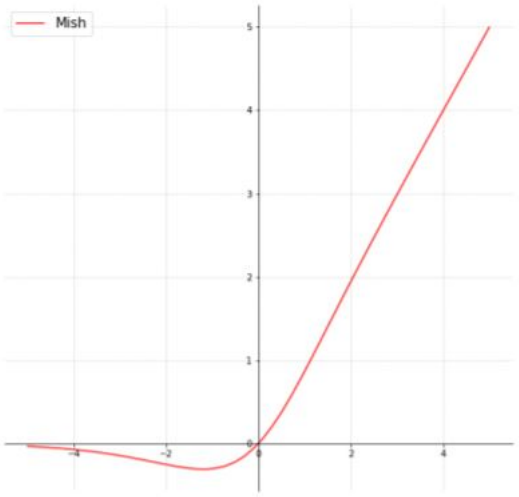


Figure 3.15: Mish Activation Function. Source[67]

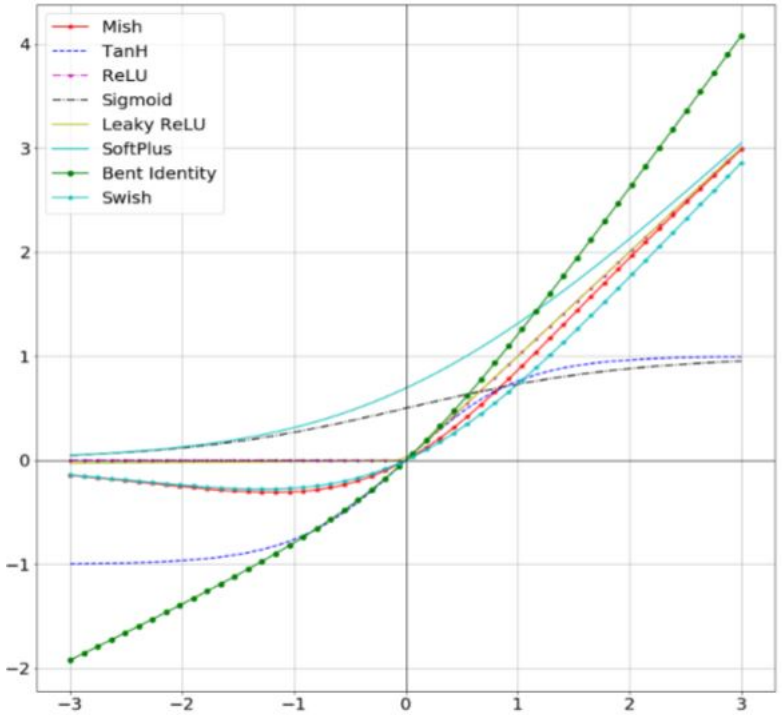


Figure 3.16: Comparison of activation functions. Source[67]

Spatial Pyramid Pooling (SPP): The fully connected layer at the end of the network require a fixed image size meaning a fixed length vector as input. On contrary, convolutional layers accepts input of arbitrary size. Therefore, spatial pyramid pooling emerges as a technique to generate fixed length vector from variable input sizes maintaining the spatial information by pooling in local spatial bins. This also enables feature extraction at multi scale [47].

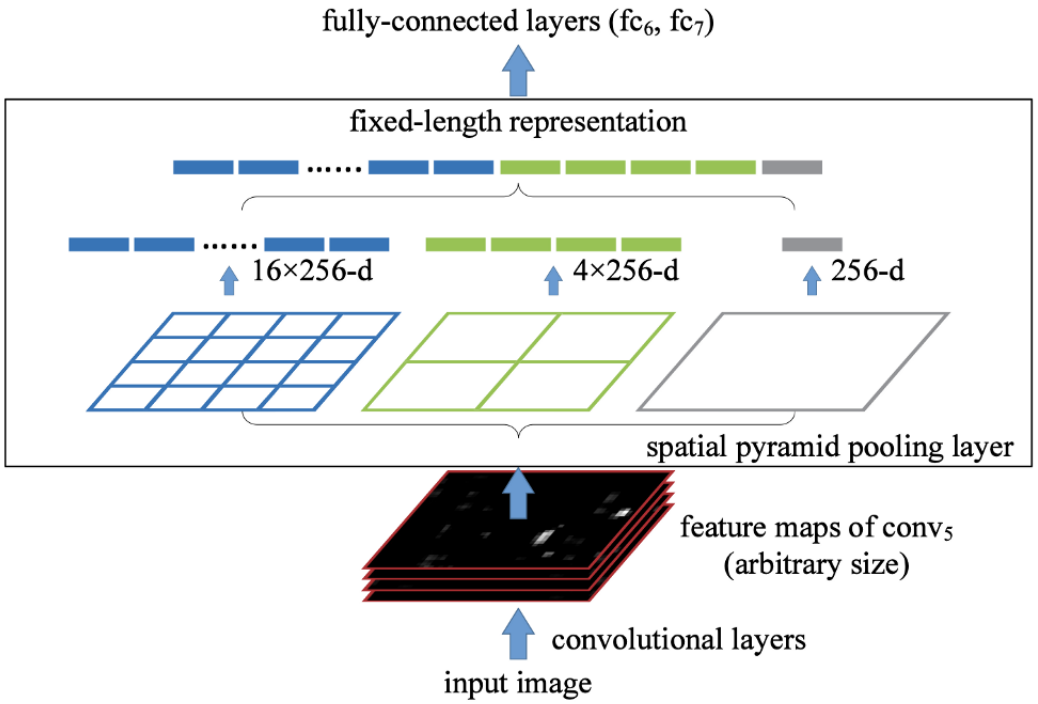
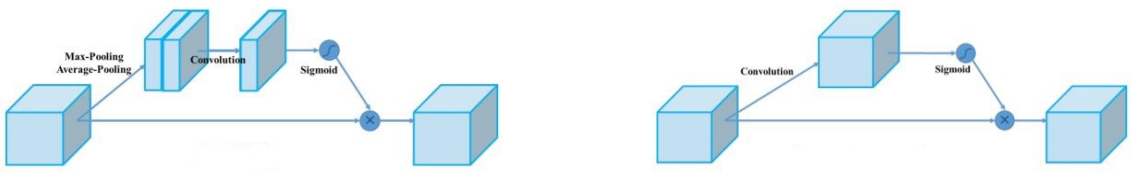


Figure 3.17: A network structure with a spatial pyramid pooling layer. Source[47]

Spatial Attention Module (SAM): Spatial attention module uses average pooling and max pooling method to generate spatial attention map by utilizing spatial relationship of features[64]. The reason behind choosing SAM over other attention module was that it increase the performance of of network without affecting the speed of inference on the GPU at all. In YOLOv4 [72] the author modify SAM from spatial-wise attention to point-wise attention shown in 3.18



(a) SAM as [64]

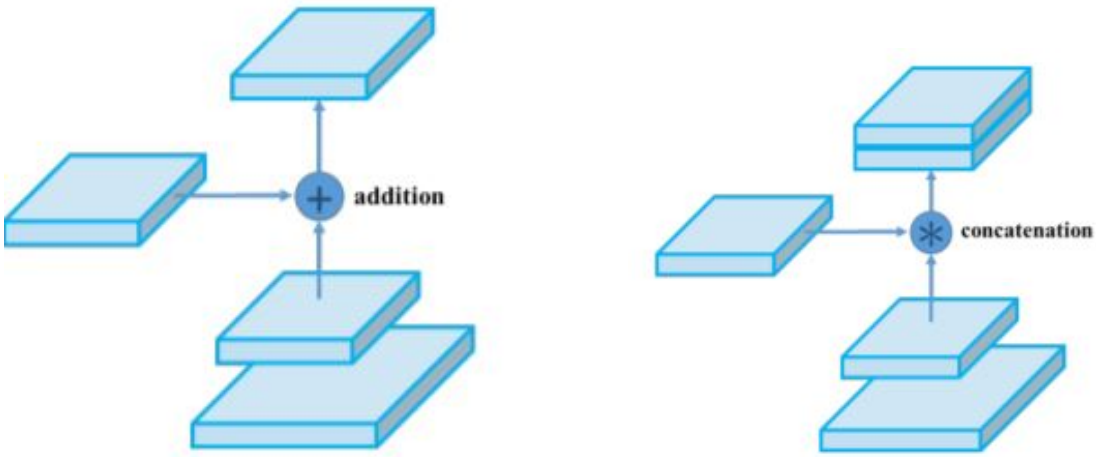
(b) Modified SAM as [72]

Figure 3.18: Spatial Attention Module

Multi-input Weighted Residual Connections (MiWRC): Many lightweight modules that integrate different feature pyramid have been proposed for multi-scale prediction. Bi-directional feature pyramid network (BiFPN) is one introduced by

[77] where, the multi-input weighted residual connections is proposed to execute scale-wise level re-weighting, and then add feature maps of different scales[72].

Path Aggregation Network (PAN): The flow of information in neural networks is of great importance. Path aggregation network does this by better propagation of layers information from bottom to top. This path augmentation shortens the information path between lower layers and topmost features [59]. But in YOLOv4 [72] author modified PAN by replacing shortcut connection to concatenation shown in figure 3.19.



(a) PAN BY [59] (b) Modified PAN by [72]

Figure 3.19: Path aggregation Networks

YOLOv4 Architecture

YOLOv4 is composed of CSPDarknet53 as backbone, Spatial Pyramid Pooling (SPP) and Path Aggregation Network (PAN) as Neck of the architecture and YOLOv3 for the head.

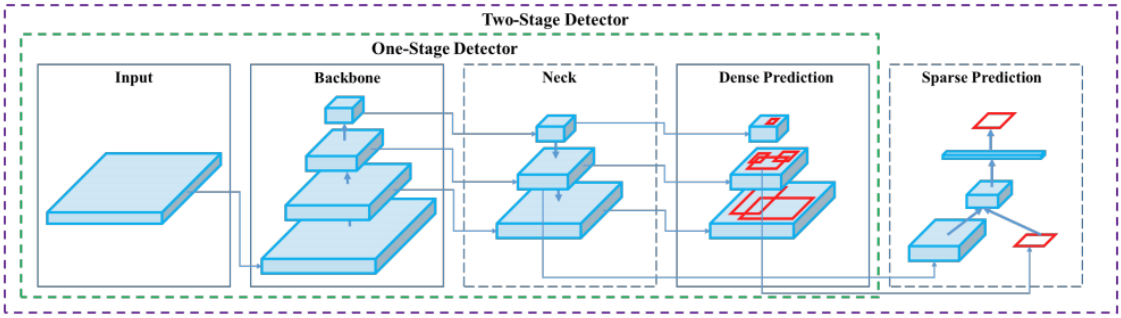


Figure 3.20: Object Detector. Source[72]

Backbone

CSPDarknet53 is used as the backbone. It is a new backbone derived from the DenseNet, which was designed to connect layers in covolutional neural network. Cross stage partial network (CSPNet) separates feature map of the base layer into two part, one part will go through a dense block and a transition layer and the other part is then merged with transmitted feature map in the next stage through cross stage hierarchy. This strategy greatly reduce the amount of computation and improve inference speed along with accuracy [78].

Neck

The next step is to collect the features formed in the earlier stage and prepare for the detection step. A neck is composed of several bottom-up and several top-down paths. YOLOv4 consider PANet (path aggregation network) for the feature aggregation and adds SPP (spatial pyramid pooling) block after CSPDarknet53 to increase the receptive field and split out the most important features from the backbone.

Head

The role of the head in the case of one stage detector is to consume features from the Neck and perform dense prediction. The dense prediction is the final prediction which is composed of a vector with the coordinates (center, height, width) of the predicted bounding box, the confidence value of the prediction and the label. YOLOv4 deploys the same head as YOLOv3 for detection.

3.2.5 YOLOv5

YOLOv5 was released two months after YOLOv4 on 9 June 2020 by Glenn Jocher of Ultralytics company. YOLOv5 is written in python programming language and released as PyTorch based version which is different from the previous release that were from Darknet written in C. It was released as GitHub repository [75] and has no any article till date by the author. But there are many articles and blogs written on YOLOv5 by different authors on different platform like Medium, Roboflow.

There are four different variant of YOLOv5 models, namely small (s), medium (m), large (l) and extra-large (x). All models run on PyTorch and having its own detection accuracy and performance shown in figure below 3.21.

Yolov5 Architecture

This version is similar to YOLOv4 with same focus structure and CSP network as backbone. The neck is composed of SPP block and PA-NET . It has

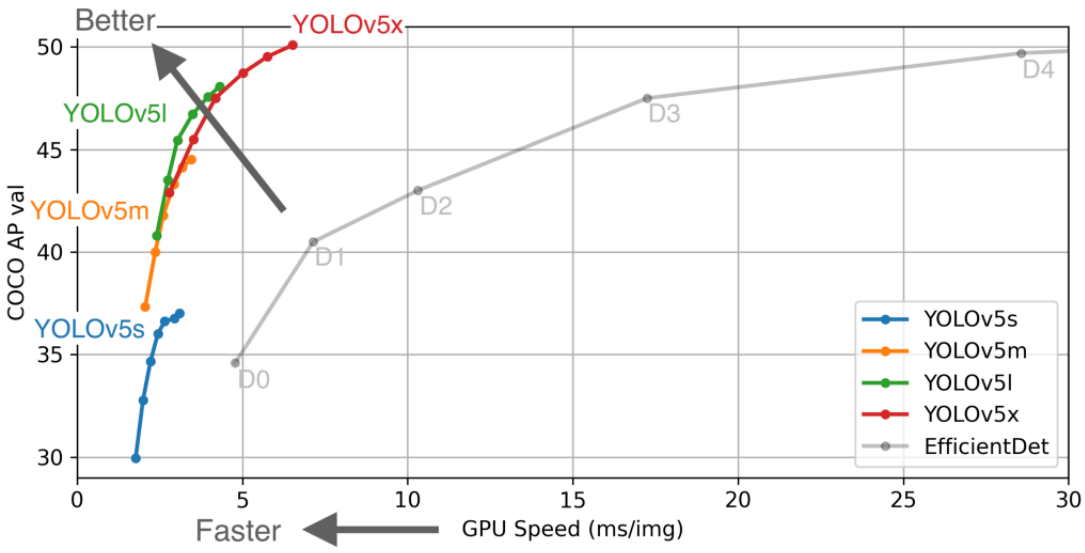


Figure 3.21: speed and accuracy comparison based on COCO AP. Source[75]

a YOLOv3 head using GIoU-loss. The major difference includes the introduction of mosaic data augmentation and auto learning bounding box anchors. The explanation to CSP and PA-NET is done above in the section YOLOv4 therefore, here paper discuss about data augmentation and auto learning bounding box that makes it difference with YOLOv4.

Data Augmentation

Data augmentation is the process of creating new training examples out of existing training data. It is not possible to truly capture an image for every real world scenario therefore, data augmentation does the trick of adjusting existing data to generalize to other situation and increase the variability of an image that allows models to train for wider scenario. The various strategies that are deployed as data augmentation in YOLOv5 are listed below. Not all the augmentation suits in all case of training the model. It depends on the individual task and purpose of model we are training for. These can be tuned and adjusted during model training process.

Distortion (photo-metric and geometric): Photometric distortion creates new images by adjusting brightness, saturation, contrast, hue and noise. Where as in geometric distortion images are rotated, flipped, scaled, cropped to generate new image samples.

Figure 3.22: Example of photo-metric distortion as hue

Figure 3.23: Example of Geometric distortion as rotation

Mix-Up: It is a convex combinations of pairs of examples and their labels. In this approach a new image is formed through weighted linear interpolation of two existing images. Mix-up regularizes the neural network to favor simple linear behavior in-between training examples and reduce the memorization of corrupt labels.

Figure 3.24: Example of mix-up

Cut-Mix: It combine the image by cutting out the patches from one image and pasting onto another training images. The ground truth labels are also mixed proportionally to the area of the patches. Cutouts of the image force the model to learn to make predictions based on a good amount of features.

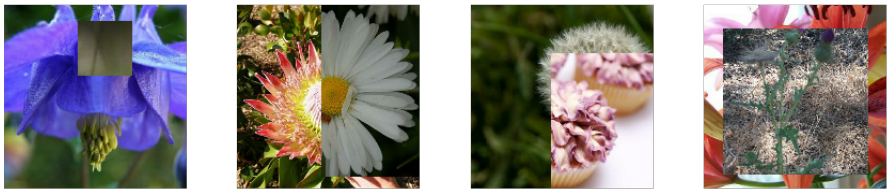


Figure 3.25: Example of cut-mix

Mosaic Data Augmentation: Mosaic data augmentation combines 4 training images into one in certain ratios (instead of only two in Cut-mix). This allows for the model to learn how to identify targets at a smaller scale than normal. It also motivates the model to localize different types of images in different portions of the frame.

Figure 3.26: Examples of Mosaic data augmentation

Auto Learning Bounding Box Anchors

In order to make box predictions the the latest YOLO network predicts bounding boxes as deviations from a list of anchor box dimensions around the image. As the YOLO was trained for coco data set, it is very important to know that when we train for custom data set , the distribution of bounding box sizes and locations may be significantly different than the preset bounding box anchors in the COCO data set. The highest difference in anchor boxes may occur if we are trying to detect tall and skinny objects or very wide and flat objects. Therefore, there is need to customize the set of anchor boxes which is done automatically on YOLOv5. It auto learns anchor box distribution based on training set.

```
AutoAnchor: 5.86 anchors/target, 1.000 Best Possible Recall (BPR). Current anchors are a good fit to dataset ✅
Image sizes 1280 train, 1280 val
Using 2 dataloader workers
Logging results to runs/train/exp41
Starting training for 50 epochs...

     Epoch   gpu_mem       box       obj       cls    labels  img_size
train: Scanning '/home/jovyan/yolov5/datasett/labels/train.cache' images and lab
train: Scanning '/home/jovyan/yolov5/datasett/labels/train.cache' images and lab
      0/49     13.8G   0.01385  0.004265  0.007797         8      1280: 100%|
               Class     Images     Labels          P          R     mAP@.5 mAP@
                 all         67        101      0.816      0.743      0.815       0.64
```

Figure 3.27: Auto anchor generated. Source: Screenshot of training process.

3.3 Requirements for Training YOLO Model

3.3.1 Data Requirements

According to the author of the YOLOv5, to get most out of the model following data set requirements are recommended.

- Images per class should be greater than equal to 1500.
- Instances per class should be grater than equal to ten thousand (labeled objects).
- Image variety must be representative of deployed environment. To match up with real-world use cases images from different times of day, different weather, different seasons, different angles, different lighting, different sources (collected locally, scraped online, different cameras) etc. are recommended.

3.3.2 Machine Requirements

There are few options to train the model without needing the high end device of own. That automatically means either using virtual machine or Docker which fulfill all the requirements and all dependencies preinstalled. Such as amazon web service and Google cloud platform. How to work with these platform is explained stepwise at:[https://docs.ultralytics.com/environments/GCP-Quickstart/.] However, to run these model in local resources (pc) there are some must need requirements. The first and most important is local machine must have GPU (Graphical Process unit) with good amount of memory. The exact need of memory depends on the volume of the training data. To make use of GPU, NVIDIA GPU drivers must be installed beforehand. The detailed process and required files for installing GPU drivers can be found at https://catalog.ngc.nvidia.com/orgs/nvidia/containers/pytorch. Along with GPU the machine should also have sufficient CPU work space (at least 8GB) to run all the application smoothly.

Machine used by the author was built with V100 GPU: V100 GPU is the most advanced and powerful data center GPU ever built to accelerate AI, high performance computing (HPC), data science and graphics. Whereas in this experiment firstly the machine with NVIDIA Geo-force GTX 1070 WITH 8GB was used and later with access to university server Quadro RTX 6000, 22696MiB GPU was used.

3.4 Background Theory on Plots and Graphs

The model performance is evaluated in terms of matrices generated as value, graphs and plots. To have clear understanding of those results generated it is very much important to understand the theory behind each of them. Here, in this section brief introduction to background theory and their calculation is explained. With these knowledge one can easily judged weather the model has performed well or need to be improved further.

3.4.1 Confusion Matrix

A confusion matrix also known as error matrix is a simple way to lay out how many predicted classes were correctly predicted and how many were not. It is used to evaluate the performance of the model. This matrix tell about what number or percentage of total instance were correctly predicted as their true classes. Not only this it also gives the clear picture about how the model confuses with other classes. To understand what‘s going inside the confusion matrix of correct classes versus incorrect classes we first need to understand few terms like **True positive, True negative, False positive and False negative**. Let‘s take an example of one traffic sign from this work, a parking traffic sign.

True Positive (TP): If an image has a parking sign on it and it is classified as parking, then this is called true positive. (in the figure 3.28 1 is TP)

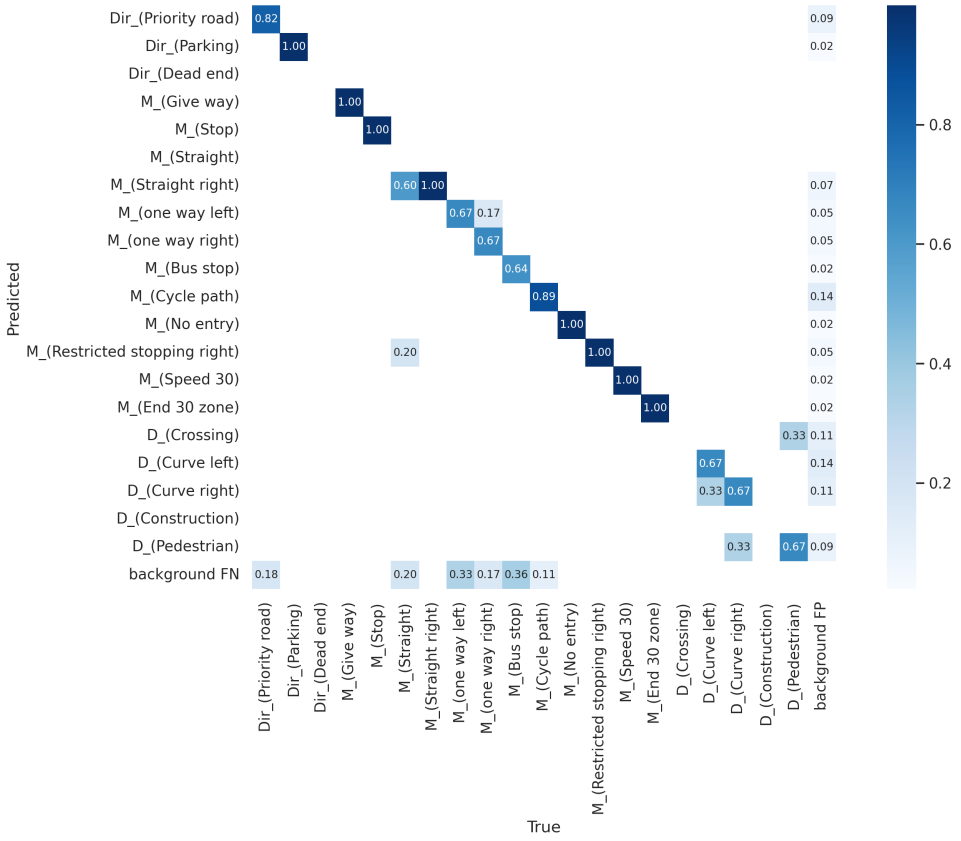


Figure 3.28: Confusion matrix. Source: Model Output

True Negative(TN): If an image has some other sign on it and it is not classified as parking, then this is called true negative. (sum of all diagonal element except for parking)

False Positive(FP): If an image has some other sign on it and it is classified as parking, then this is called false positive. (sum of all horizontal element except for parking)

False Negative (FN): If an image has parking sign on it and it is not classified as parking, then this is called false negative. (sum of all vertical element except for parking)

For model to perform best it is desired that all the value falls in True positive and True negative and these values lie on the diagonal of confusion matrix.

3.4.2 Calculation of Matrices

Accuracy is defined as the ratio of the number of correctly classified data instances over the total number of data instances. Mathematically it can be expressed as:

$$Accuracy = \frac{sumofdiagonalofconfusionmatrix}{totalcount}$$

For the imbalance data set, accuracy measure does not prove to be good.

Precision also known as positive predictive value can be calculated as:

$$Precision = \frac{Truepositive(TP)}{TruePositive(TP) + FalsePositive(FP)}$$

For good classifier, precision value should be high ideally 1. This can only happen if the false positive (FP) is zero.

Recall is also known as sensitivity or true positive rate and mathematically defined as:

$$Recall = \frac{Truepositive(TP)}{TruePositive(TP) + FalseNegative(FN)}$$

For good classifier, also recall value should be high ideally 1. This can only happen if the false negative (FN) is zero.

Fβ Score is the metric that take into account both precision and recall and is defined as:

$$F\beta = (1 + \beta^2) * \frac{Precision * Recall}{\beta * Precision + Recall}$$

F1 score is the simplified form of Fβ score and is defined as harmonic mean of precision and recall. That is when beta = 1 above expression is modified to:

$$F1Score = 2 * \frac{Precision * Recall}{Precision + Recall}$$

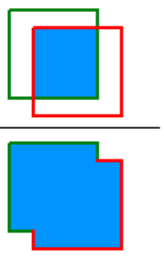
β is defined depending on the problem statement. When problem needs to give equal importance to both precision and recall alternatively to FP and FN beta is equal to 1, if more focused to False negative then beta should be increased and if more concern is for false positive then β should be decreased.

PR Curve: A precision-recall curve is a plot of the recall on the X-axis and the precision on the Y-axis for different probability thresholds (Confidence value). Confidence value is the confidence of the model on the detection and it ranges between 0 and 1. For a good model, precision and recall stays high even when confidence score is varied.

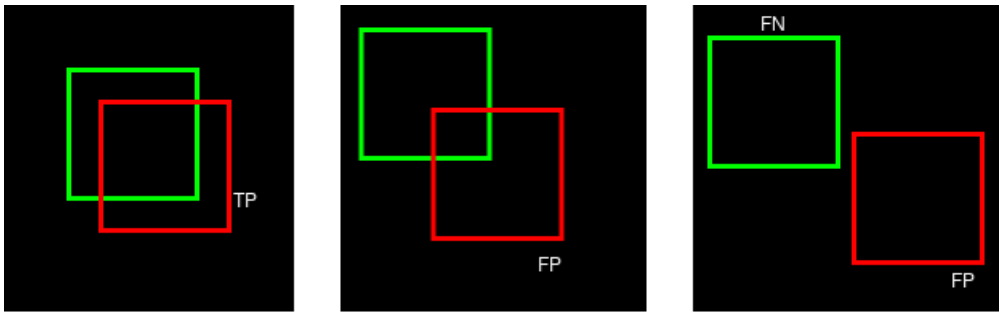
3.4.3 Intersection Over Union (IoU)

Intersection Over Union (IoU) is the degree of overlap between the ground truth and prediction. It is expressed as the ratio of area of intersection to the area of union of the predicted bounding box over ground truth bounding box.

$$IOU = \frac{\text{area of overlap}}{\text{area of union}} =$$



IoU ranges between 0 and 1 where 0 represents no overlap and 1 shows complete overlap between ground truth and prediction. IoU require threshold value let say (α), to decide if a detection is correct or not. For example take IoU threshold, $\alpha = 0.5$, then TP, FP and FNs can be identified as shown below:



IoU = 0.86 IoU = 0.24 IoU = 0

If we raise IoU threshold above 0.86, the first instance will change from TP to FP and if we lower IoU threshold below 0.24, the second instance will becomes TP from FP.

Average Precision (AP): AP at α is area under the Precision-Recall Curve (AUC-PR) evaluated at α IoU threshold. A high area under PR curve means high recall and high precision. Various ways of calculating AP are:

Based on IoU values

- AP@IoU=0.5 (traditional way)
- AP@ IoU= 0.50: 0.05: 0.95 (primary challenge metric)
- AP@IoU=0.75 (IoU of bounding boxes need to be > 0.75)

0.5:0.05:0.95 usually means IoU starting from 0.5, with steps of 0.05, and increase to an IoU equal to 0.95. These would result in computations of AP threshold at ten different IoUs. An average is calculated to get a single number which rewards detectors that are better at localization.

Based on Scales

- APŝmall: AP for small objects: area $< 32^2$ px
- APm̂edium: AP for medium objects: $32^2 <$ area $< 96^2$ px
- APl̂arge: AP for large objects: area $> 96^2$ px

Mean Average Precision (mAP): AP is calculated individually for each class. The mAP for object detection is the average of the AP calculated over all the classes. If threshold for IoU is chosen 0.5, it is called as mAP@0.5 and if Iou is (0.5:0.05:0.95), it is called as mAP@(0.5:0.95). The plot 3.29 shown is the output from this experiment for first 40 epochs. At the end of 40 epochs model gain an mAP of 0.55. And later after the model was fully trained it reach an mAP of 0.84 which is very good result for any model and can be deployed to many application.

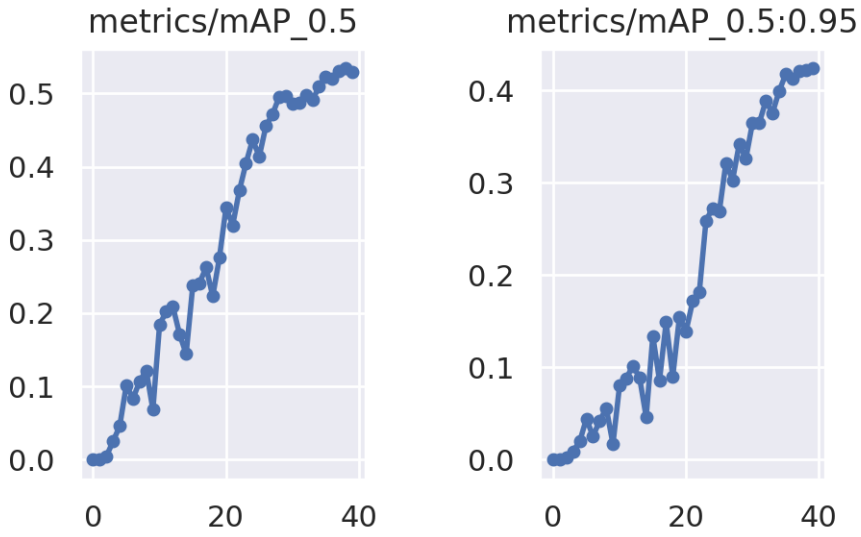


Figure 3.29: Plots for mAP @ 0:5 and mAP@(0.5:0.95). Source: Model Output

Train and Validation Loss

Train and validation loss gives an idea, if the model is overfitting or underfitting. Both the plots are compared and judged the results based on following are the scenario:

- If validation loss » training loss overfitting.
- If validation loss > training loss some overfitting.
- If validation loss < training loss some underfitting.
- If validation loss « training loss underfitting.

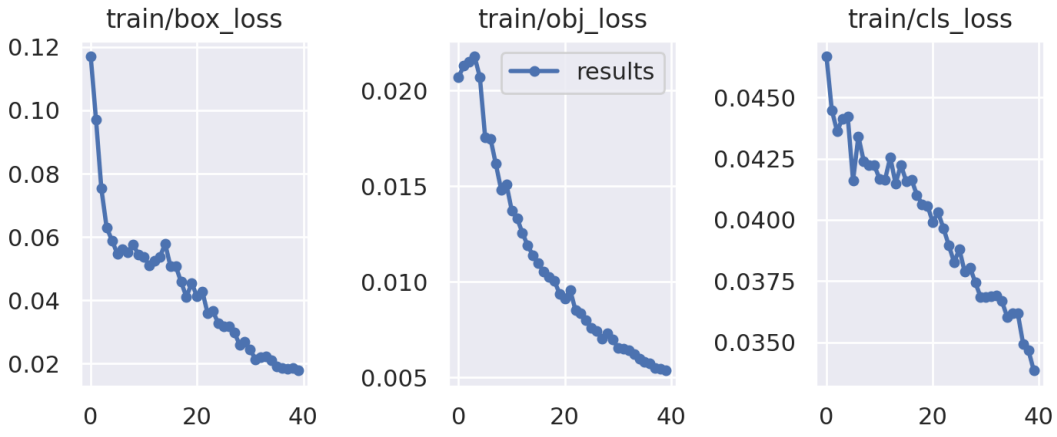


Figure 3.30: Training Losses. Source: Model Output

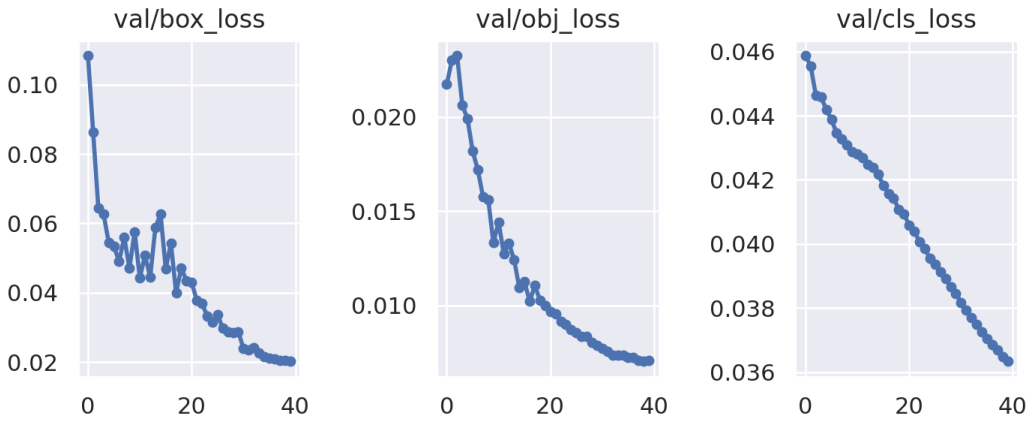


Figure 3.31: Validation Losses. Source: Model Output

These losses are further categorized into three parts. they are:

Box Loss: A loss that measures how "close" or "tight" the predicted bounding boxes are to the ground truth object.

Object Loss: A loss due to wrong box-object prediction.

Class Loss: A loss that measures the correctness of the classification of each predicted bounding box. Each box may hold an object class, or a "background". This loss is usually called cross entropy loss.

Chapter 4. Methodology

Here this chapter, it is focused on explaining all the processes and methods need to be carried out, from preparing data to training model and inferencing. It begins with data preparation that include methods of collection, annotation and splitting. In next step model selection and its configuration is explained. In 4.3 training methods are elaborated and in the last section it describes about how to inference with different data source and then saving the model for future use.

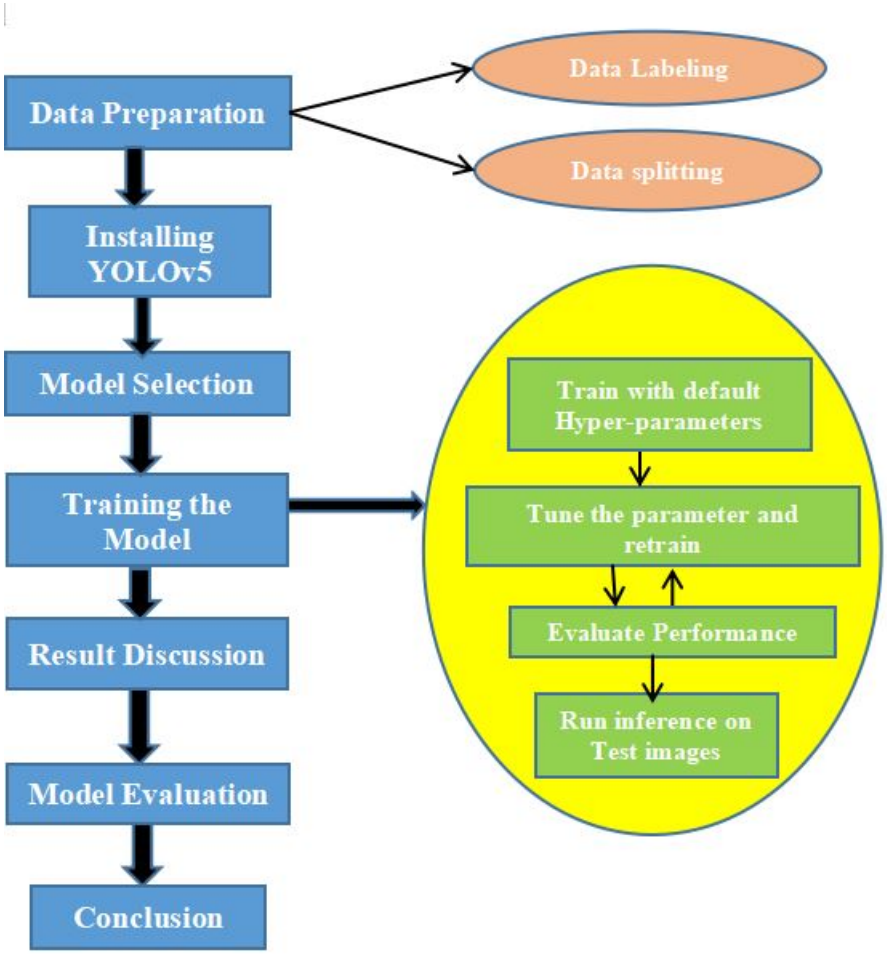


Figure 4.1: Flow chart

4.1 Data Preparation

Data is the main ingredient for any machine learning and deep learning task. The quality and quantity of data depicts the performance of the model. Therefore, data mining prior to training the model is very important steps. Information about the data is needed to build the base knowledge for any machine learning or deep learning task.

4.1.1 Data Collection

Data collection work has already been done by the traffic department of State Office for Geoinformation and Surveying Hamnurg, Germany (Landesbetrieb für Geoinformation und Vermessung, LGV Hamburg, Germany) in the year 2016. The data (images) was captured by the four camera installed on the car. Each of the camera was faced in different direction to capture all four (front, rare, left and right) view at a location. The images were collected driving against and in the direction of road section from the federal roads of Hamburg. In this work, most of the images from three road section from federal road number five (B5) were used for creating the model. The images were captured at the interval of every 5 meter. For the first section of road car was drove to a total of 878 meter against the direction of road and 875 meter in the direction of road. Likewise, for second section car drove to total of 370 meter against and in the direction of road. Similarly, for the third section car was drove to total of 766 meter in both direction. From all three section, decent amount of images were selected along with some more images from another road section were added to create a data set required for model development.

The resolution of the camera used was 1920 pixel horizontal and 1080 pixel vertical. All the images were saved as .jpg image format.

4.1.2 Data Annotation

Data annotation task is laborious at the same time sensitive. Therefore, before begin to label the image the number of classes and their respective labels must be defined first. This is very important as later in the middle of the labeling process most of the changes cannot be made. For example, it is possible to increase the number of instances by adding more images, it is also possible to increase the number of classes but decreasing or removing the classes that were already defined is not allowed. If it is done so, the class number get altered and all the previous work done be of no use and have to restart from very beginning. Therefore, it becomes very necessary to define the numbers of classes and their names (labels) beforehand with all the considerations. This work was dedicated to build the traffic sign detection model for street view images and videos of Hamburg. Traffic signs were classified into categories and sub categories based on the division made by the traffic department of LGV. They have their own categorization of traffic signs[86] [84] which are little bit different. Below is the image 4.2 showing how the classes are named for this work and same are attached as label to each object in image.

Training and testing a Deep Convolutional Neural Network requires a large amount of data annotated with higher accuracy. Accurately labeled data is essential to successful machine learning, and computer vision work. Therefore, data annotating becomes a critical task and to be carried out with higher consistency. To achieve best annotation all the images were annotated manually using LabelImg tool available as free of source for labeling images. All instances of all classes in all images were labeled without missing any of the objects present in image. Each objects were tightly enclosed inside bounding box making sure that it did not miss any part of object and also not enclose more background. In this way data annotation tasks was performed with great consistency and accuracy in the format required to train

YOLO models.

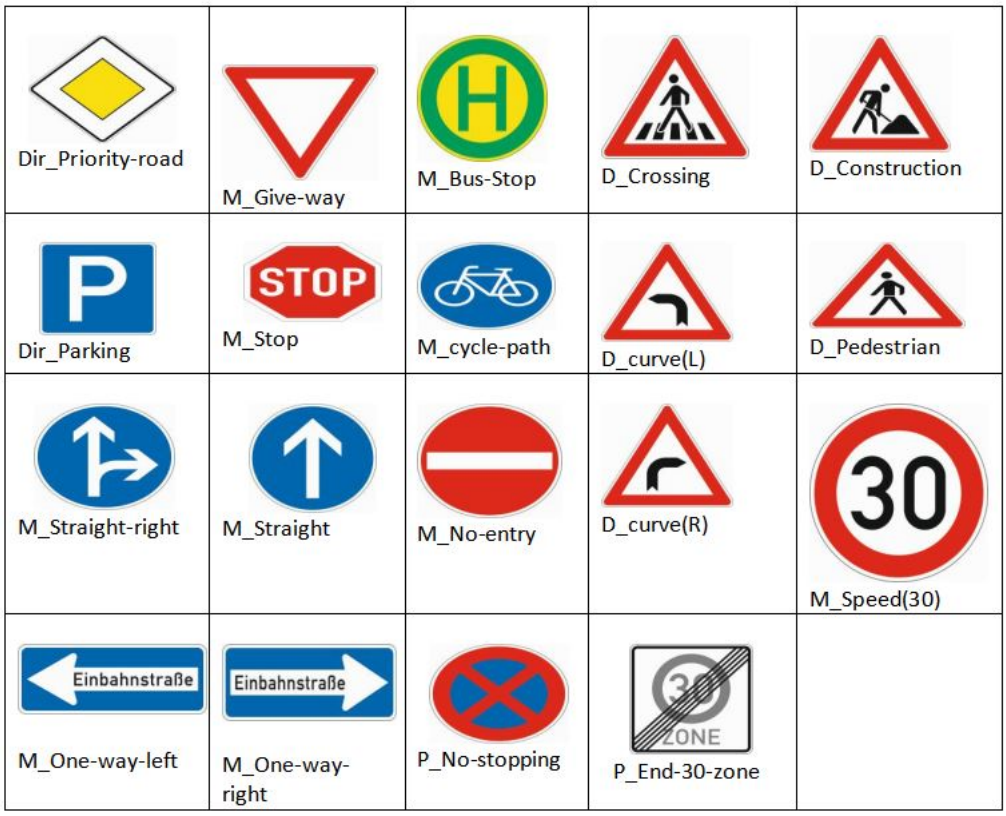


Figure 4.2: Traffic Sign classification. Source: LGV, Hamburg

4.1.3 Data Statistics

The quantity of data has direct relation to model performance. If the number of data is big enough model learns sufficiently enough to generalize the real world scenario and detection and prediction become more accurate.

In the figure 4.3 the first one is the bar plot for counting the number of instances that each class has. The x-axis represent the name of the classes and y-axis represents the number of instances that each class has. This plot gives an idea about how balanced the data set is. It is not always true that only balanced data set gives the best accuracy but one should be careful that each class consist of enough numbers to learn the pattern than to memorise. There is no any science behind choosing the exact numbers of data. It can be configured from the experience and the initial results of trained model. [Note: Not be confused that number of instance is not equal to the number of images, as one image may contain more than one instance of one or many classes]. Looking at the plot 4.3 it is clear that data set is not that balanced. The class Dir_(priority road) and M_(Bus stop) has maximum number of instances where as M_(straight) and D_(Crossing) has least. The bottom two plots in 4.3 shows the image size distribution. It was obvious that the traffic signs are smaller and occupy less space in image.

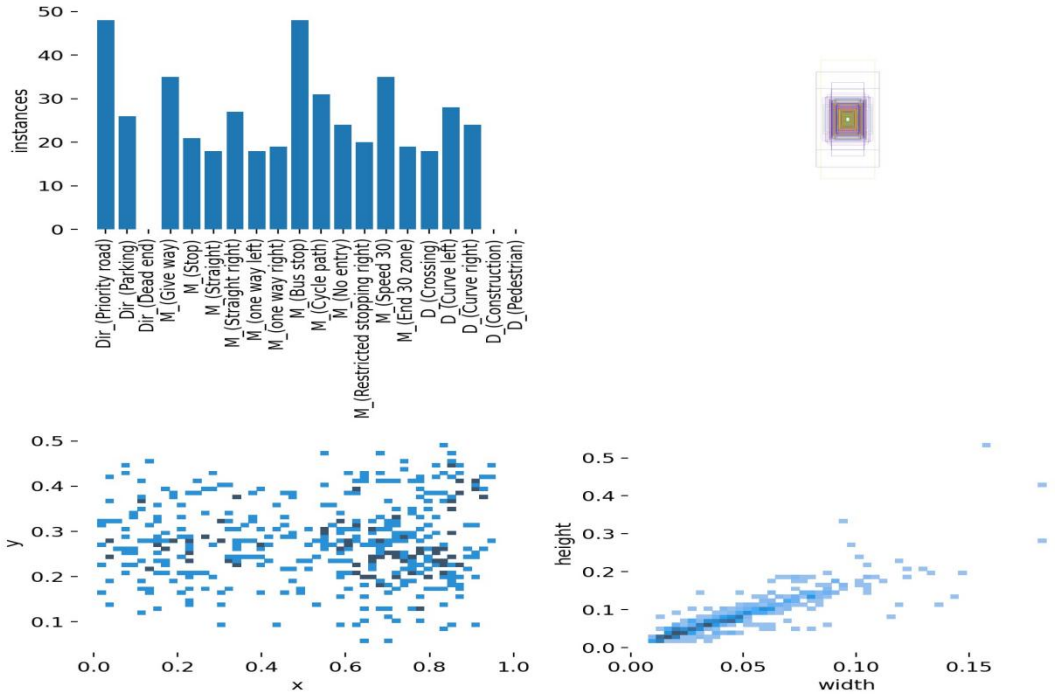


Figure 4.3: Plot showing data statistics. Source: Model Output

The total number of images annotated for this experiment was 400. of those 400 images, contain 700 instances of traffic sign.

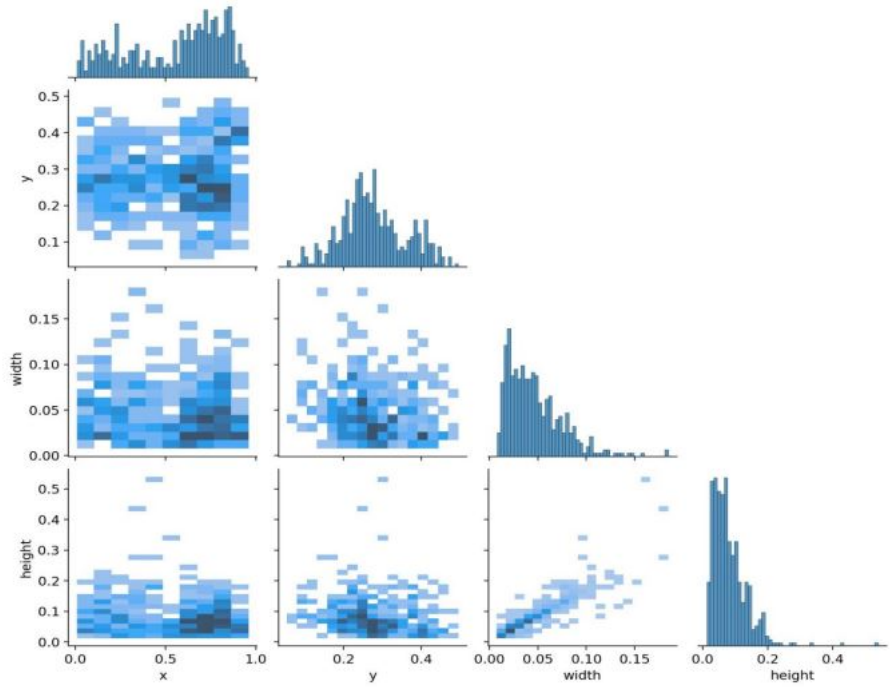


Figure 4.4: Label Co-relation Graph. Source: Model Output

The plot 4.4 shows the co-relation between the x and y coordinate and height and width of the label box. From the plot it can be concluded that most of the traffic signs were small in the size, whose height and width were ranges from 0.0

to 0.2. Also height and width showed linear co-relation i:e height of the instances increases with the increase in width size.

4.1.4 Data Split

Data splitting is most important, when it comes to machine learning or deep learning as it allow to cross verify the model performance and change the parameters accordingly to get best fit for the model. Data split is the act of dividing original data into train set and validation set.

At the end of the training process, the final model should predict correct outputs for validation data set as well as be able to generalize well to previously unseen data (test data). Poor generalization can be characterized by over-training where the model just memorizes the training examples. This could happen if the number of training dataset is less or in insufficient amount. The two terms good prediction and good generalization is often known as Bias and Variance dilemma.[see 24] [17]. To balance between Bias and Dilemma and makes sure model does not just memorize training samples, exploratory data set was divided in the ratio 7:3 (70% training and 30%validation).

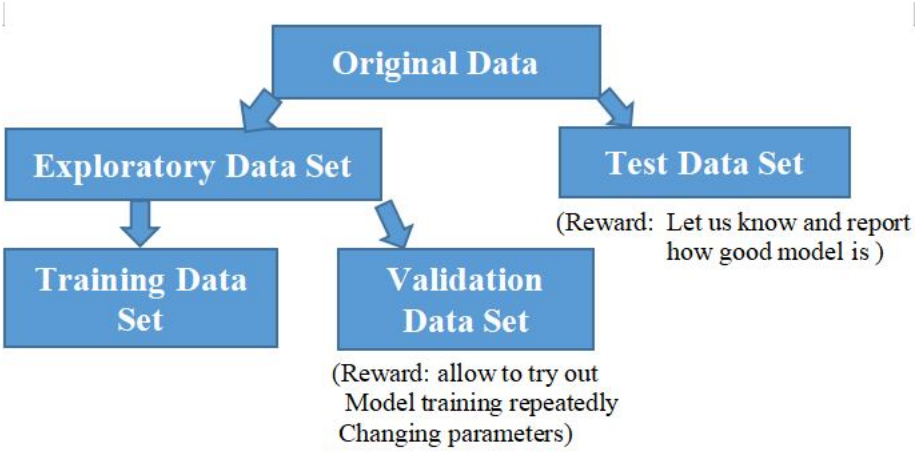


Figure 4.5: Showing Data split

The figure 4.5 show the data split concept. First the whole dataset was separated into two parts exploratory dataset and test dataset. This task was done manually keeping in mind to include all the examples from classification set on test data set so that a compete performance of model could be judged. In next step Exploratory data set were divided into training and validation using function from PyTorch library i:e torch.utils. The function is: Data.random_split(dataset, length). More details on function and for documentation it can be looked at:
https://pytorch.org/docs/stable/_modules/torch/utils/data/dataset.html#random_split
https://pytorch.org/docs/stable/data.html#torch.utils.data.random_split

4.2 Model Selection

The YOLOv5 repository provides 5 pre-defined version of models namely Yolov5-nano, Yolov5-small, Yolov5-medium, Yolov5-large, Yolov5-extraLarge. Each of them differing in size as well as performance. It gives flexibility to select any one of the model depending on task problem, or one can try with all the version one by one to see which one better perform for given task and data set. According to the author of YOLOv5 the larger models like YOLOv5X would produce better result in nearly all cases, but have more parameters and require more CUDA memory to train the model.

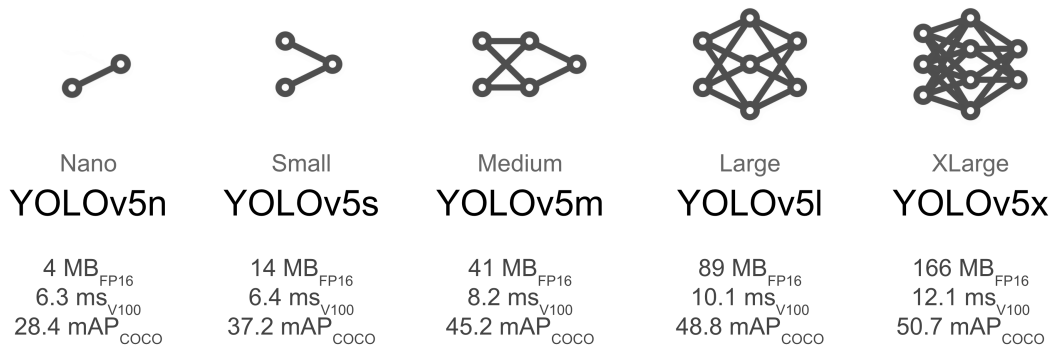


Figure 4.6: Model comparison. Source[75]

The figure 4.6 briefly shows the various features such as size of the model in mb, speed, and mAP value based on cocoval2017 data set. These models can be found as .yaml files inside the models' folder from the cloned repository.

4.2.1 Configuring the Model

In this experiment, custom dataset is used with the goal to perform traffic sign detection and recognition. The data set consists of images with 20 different signs representing 20 classes. To process training of YOLOv5 on custom data, coco128.yaml file inside the data folder from the cloned repository need to be edited or new yaml file can be crated and replaced it with coco128.yaml. For editing, the first step is to provide root directory to the dataset folder, where all the images are organised as train and validation. Then provide the relative path to the folder with train, validation and test images. Next step is to replace the value for 'nc' that is number of classes and finally replace the names of classes with the one that was defined during annotation process.

Contents inside yaml file

path:`C:\Users\MSI\Desktop\Traffic_sign_detection\dataset\images`
(dataset root directory)
train: train (relative path to train images folder)

val: val (relative path to val images folder)
test: test (relative path to test images folder. This is optional)

Classes
nc: 20 (number of classes)

names: ['Dir_(Priority road)', 'Dir_(Parking)', 'Dir_(Dead end)', 'M_(Give way)', 'M_(Stop)', 'M_(Straight)', 'M_(Straight right)', 'M_(one way left)', 'M_(one way right)', 'M_(Bus stop)', 'M_(Cycle path)', 'M_(No entry)', 'M_(Restricted stopping right)', 'M_(Speed 30)', 'M_(End 30 zone)', 'D_(Crossing)', 'D_(Curve left)', 'D_(Curve right)','D_(Construction)', 'D_(Pedestrian)'] # class names

For this experiment, model was trained for 20 classes therefore, nc =20 and names were replaced with the classes name that were defined at the time of annotation . For this work the names were of traffic sign according to their categorization. Dir_(name) for Directional, M_(name) for Mandatory and D_(name) for Danger.

After finishing up these all configuration and settings we are ready to train the model.

4.3 Training of the Model

The YOLOv5 repository comes with a model pre-trained on the coco data set, which is very popular data set for bench-marking different object detector. The coco data set contains 80 classes including traffic light, stop sign, parking meter, bird, cat, dog, cars, bike and many more. Therefore, model can take the advantage of pretrained model in the context of transfer learning. Transfer learning reliably speeds up training because the model already knows the low level features and can focus on adjusting higher level one and improves accuracy.

Everything is ready by now to train YOLOv5 on custom data set. The last step is to set the training parameters and execute the command to begin the training. The command is:

!python train.py –weights <yolov5s.pt> –img <'width of image'> –batch <'batch size'> –epochs <'no of epochs'> –data <'location of the .yaml file'> –cfg <'Which yolo configuration you want'> –device <Gpu or cpu > – nosave –cache

- **Img:** Stands for image, we need to define the input size of the image in pixels.
- **Batch:** During training the dataset is split into mini batches defined by the programmer. The batch size is in the order of 2, 4 8 16, 32 and continue. To choose between these again depends on GPU memory.
- **Epochs:** When the model trains on all the mini batches once it completes one epoch. Any number can be set to begin training.
- **Data:** This is the path to data yaml file that has been created during configuring model step.

- **Weights:** Training on custom dataset can take the advantage of the pre-trained model. Here, path to weights file of pretrained model is passed.
- **Device:** This is to choose between CPU and GPU. If nothing is defined the default is CPU.
- **Nosave:** Nosave options allows only to save the final checkpoint.
- **Cache:** Save the images cache for faster training.

4.3.1 Optimization and Tuning of the Model

Hyper-parameters control various aspects of training, and finding optimal values for them is a challenge. It is very rarely seen that model get trained perfectly with initial settings of parameters. Even with the transfer learning everything does not fit accordingly. Therefore, once the model get trained as baseline with initial settings the results need to be carefully analyzed to perform optimization of the parameters. YOLOv5 has about 30 hyper-parameters used for various training settings. These are defined in yaml files in the data directory inside cloned repository. The default parameters are shown below:

- **lr0:** 0.01 [initial learning rate (SGD=1E-2, Adam=1E-3)]
- **lrf:** 0.1 [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]
- **textbox:** 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)]
- **objß_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)]
- **copy_paste:** 0.0 [segment copy-paste (probability)]

4.4 Inference and Save the Model

The final step to this experiment is inferencing. Inferencing means testing the model performance on new and unseen data. The good thing with the YOLOv5 is, it support testing for all kind of data types (images, videos, webcams, YouTube streaming, directories). It does not require separate training of model with videos to inference on videos. The only thing we have to do is define the source while running inference. The command for running inference is:

!python detect.py –source test data –weights runs/train/exp18/weights/best.pt –img 640 –conf 0.50 –save-txt –save-conf

- **img:** To inference on images, the same image size should be passed at which model was trained.
- **source:** YOLOv5 accepts a directory of images folder, video files, YouTube videos and also webcam port.
- **weights:** The model weights were saved in weight folder of cloned repository. Here, need to pass the path to weight folder that contain file 'best.pt'.
- **device:** This is to choose between CPU and GPU. The inference time is very fast with GPU.
- **save-text:** This allow to save the result as text document
- **save-conf:** This save the confidence value for all detected object.

4.4.1 Save the Model

This is the final step and carried out when satisfied with the testing results. The weights file of custom YOLOv5 object detector are saved and exported for future inference, for use on a live computer vision task and for deployment to many application.

Chapter 5. Implementation

To create the custom object detector several setups and installations of software were carried out. Firstly, annotation tool LabelImg was installed for labeling the images in the format required for training the YOLO model. The process of installing to using the tool is explained in the first section of the chapter. YOLOv5 is PyTorch based and written in python. Therefore, python need to be installed along with all the dependencies. 5.2 and later section describes all the process. Basically, in this chapter all implementation details like software, libraries and frameworks that has been used are briefly explained.

5.1 Data Annotation Tool LabelImg

LabelImg is a free, open source and powerful tool that allows to label the class in the image graphically. It's written in Python language and uses QT for its graphical interface. Qt is a cross-platform application development framework for desktop. LabelImg annotate the image in two different format PASCAL VOC format and YOLO format. The advantage is it can be installed and run on all the operating system Linux, mac OS and windows.

5.1.1 Installing LabelImg on Windows with Anaconda.

Installing LabelImg tool in windows and anaconda distribution environment is very easy and required followings steps.

1. First we need to download the zip file from the GitHub repository at https://github.com/tzutalin/labelImg. and save with the name labelImg.
2. Open the anaconda prompt command, create virtual environment and install the required libraries as follows:
   - (base) C:/Users/MSI>conda create -n (give_name) python==3.7.0
   - conda activate (give_name)
   - (give_name) C:/Users/MSI>pip install lxml
   - (give_name) C:/Users/MSI>pip install PyQt5
3. Change the file directory to the folder labelImg that we saved in first step. And run the command as follow to initiate tool.
   - (give_name) C:/Users/MSI>E:
   - (give_name) E: cd E:/labeling/labelImg
   - (give_name) E:/labeling/labelImg>pyrcc5 -o libs/resources.py resources.qrc

- (give_name) E:/labeling/labelImg>python labelImg.py

After successfully running all the steps it will open up the graphical interface as shown below, where we can import the image and label the class as per the model requirements. Before starting to label the image it is important to modify predefined_classes.txt inside data folder to define the list of classes that will be used for training. More detail and stepwise explanation can be found at https://github.com/tzutalin/labelImg.

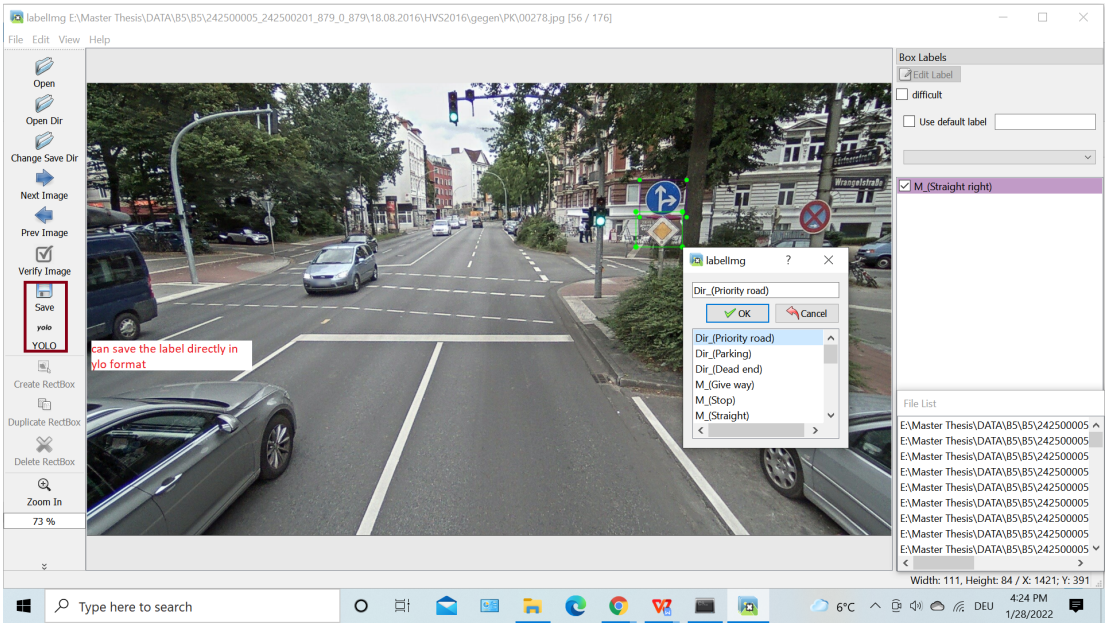


Figure 5.1: Graphical interface of labelimg tool image (1).

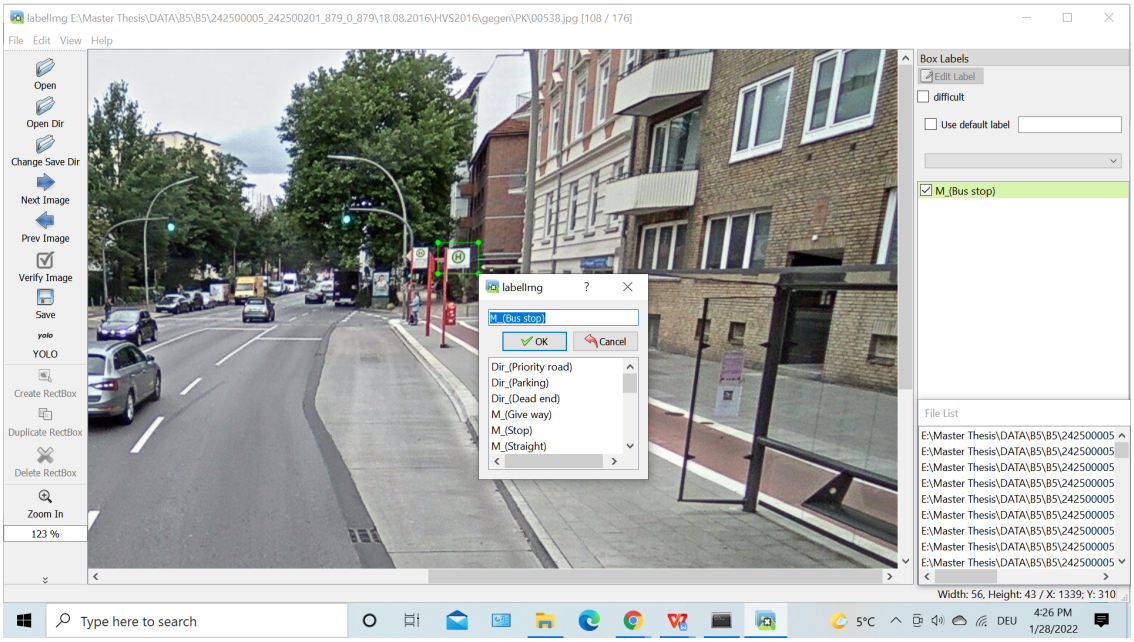


Figure 5.2: Graphical interface of labelImg tool image (2)

The output of the annotation is .txt file in the format shown in table 5.1. The format is called as YOLO format. It consist of single or multiple rows. Each

row represents for one object in the image. Each row has five column namely Class, X_centre, Y_centre, width and height. Class numbers are zero indexed and start from 0. Box coordinates X and Y are in normalised form (from 0-1). If the values of coordinates are in pixels then x_coordinate and width of object in image should be divided by image width and y_coordinate and height of object in image should be divided by image height to get to normalized form.

Class	X-centre	Y-centre	width	height
0	0.718	0.86	0.158	0.196
5	0.423	0.561	0.145	0.188

Table 5.1: Yolo format txt file

5.2 Python and PyTorch Installation

After the completion of data preparation task , next was to setup and create the environment to train YOLOv5 on custom data set. The YOLOv5 was downloaded from its repository at https://github.com/ultralytics/yolov5. It can also be cloned directly from the same repository. YOLOv5 requires Python>=3.6.0 environment, including PyTorch>=1.7. Therefore, python and PyTorch were installed along with all the dependencies. The installation procedure are described in following subsections.

5.2.1 Installation of Python

There are several methods of installation of python. It can be downloaded from official python distributions from https://www.python.org/ , install from a package manager and even one can install specialized distributions for scientific computing. Downloading from official distributions is the most popular installation method as they are the best and easy option for getting started with python and provide full-featured python development environment. The another way is to install 'Anaconda', which is a distribution of python and R programming language for scientific computing. The distribution includes data-science packages suitable for Windows, Linux, and macOS. This work has been done in Jupyter notebook, a web based application that comes with anaconda. The details on system requirements and installation of anaconda can be found at https://docs.anaconda.com/anaconda/install/ . And how to use Jupyter notebook with anaconda can be looked at https://docs.anaconda.com/ae-notebooks/4.3.1/user-guide/basic-tasks/apps/jupyter/ [Note: Download the latest version,Python>=3.6.0]

5.2.2 Installation of PyTorch

YOLOv5 uses PyTorch, one of the most popular machine learning framework to define models, run inferences, and perform training. Machine learning

frameworks provide an easy and efficient way to run inferences and training calculation on GPU. PyTorch is a GPU accelerated tensor computational framework for deep learning. In PyTorch, the torch.cuda package has additional support from CUDA (Compute Unified Device Architecture) tensor type that implement the same function as CPU tensors but they utilizes GPU for computation. This functionality brings a high level of flexibility and speed. Therefore, it is necessary to install the correct version of NVIDIA's parallel computing platform CUDA that enables GPU acceleration for training and inferencing. The steps shown below can be followed to install CUDA enabled PyTorch:

- Go to https://pytorch.org/
- Select the operating system (available FOR Linux, Mac and Window)
- Select package (Here choose option Conda)
- Select CUDA version 10
- Download the installer file and execute the installation.

After the installation of CUDA toolkit, PyTorch need to be installed. For this open the command prompt cell and run the command shown below:

conda install pytorch torchvision torchaudio cudatoolkit=10.2 -c pytorch

To check for if installations are successful, easiest way is to call torch.cuda.is_available(). If it returns true, this means the nvidia drivers correctly installed.

5.3 Installation of Python Modules and Libraries

There are some other python modules and libraries that YOLOv5 need. These modules doesn't install with default installation and need to be installed manually. Below show the list of these modules. This list can be found in the requirements.txt file inside YOLOv5 folder downloaded from repository.

- matplotlib>=3.2.2
- numpy>=1.18.5
- opencv-python>=4.1.2
- Pillow>=7.1.2
- PyYAML>=5.3.1
- requests>=2.23.0
- scipy>=1.4.1
- torch>=1.7.0

- torchvision>=0.8.1
- tqdm>=4.41.0
- tensorboard>=2.4.1
- pandas>=1.1.4
- seaborn>=0.11.0
- coremltools>=4.1 (CoreML export)
- onnx>=1.9.0 (ONNX export)
- onnx-simplifier>=0.3.6 (ONNX simplifier)
- scikit-learn==0.19.2 (CoreML quantization)
- tensorflow>=2.4.1 (TFLite export)
- tensorflowjs>=3.9.0 (TF.js export)
- albumentations>=1.0.3
- Cython (for pycocotools)
- https://github.com/cocodataset/cocoapi/issues/172
- pycocotools>=2.0 (COCO mAP)
- thop (FLOPs computation)

To install all theses modules was easy. We just need to change the directory of working jupyter notebook to Yolov5 folder and execute the shell with command pip install -r requirements.txt.

[Note: The process off installation might end showing the error message that ask to install Microsoft C++ build tools. The message also has link that guide to installation. This is usually asking to install visual studio. Its simple just follow the error link and install it. After the successful installation run the pip install -r requirements.txt again.]

Successful installation of all above setup the programming environment to be ready to execute YOLOv5.

Chapter 6. Experiment

After having all set, this chapter explained how all the methods explained in before chapters were implemented and performed to train custom YOLOv5 model. Early sections, describes how base line training was performed and visualized. Which is then followed by explaining parameter optimization process and its impacts on output. Finally, the chapter closed with testing trained model on various data source such as images, videos and webcam feeds.

6.1 Train Custom YOLOv5 Model

Firstly the pretrained model was decided to serve as baseline training of custom model. The weight file 'yolov5 large' was selected because of its better performance than the smaller model. The pretrained model was trained with coco dataset for 80 classes. Where as the custom model need to be trained for 20 classes, nc value inside models yaml file need to be changed from 80 to 20. Even one doesn't replace nc value, architecture automatically overrides it. During training process the message "Overriding./yolov5/models/yolov5s.yaml nc=80 with nc=20" confirms that, this step in not mandatory.

To begin with base line training, there are also some other settings that must be defined. They are as follows:

- **Img** = 640
- **Batch** = 4
- **Epochs** = 40
- **Data** = coco128.yaml (This is the path to edited coco128.yaml file.)
- **Weights** = yolovls.pt (This is the weight file for large model. It was selected because of its better performance than smaller one.)
- **Device** = 0 (for single GPU option)

Once the model get trained for first 40 epoch the looses were monitored. from the plot 6.1 it was clearly seen that looses were still decreasing and precision and recall were increasing. Therefore, the training process was continued until the model was fully trained and no further improvements was seen.

The horizontal axis represent numbers of epoch for model was trained and in vertical axis is the precision value in 6.2a and recall value in 6.2b. With the increase in epoch number the precision and recall both are gradually increasing. Since, the model was trained step by step with 40 to 50 epochs at a time, the above plots shows the result for 40 epoch. Model was able to reach 80% precision with just of 40 epochs training

and recall was 55%. At the end after complete training, model was able to gain mAP of 0.77.

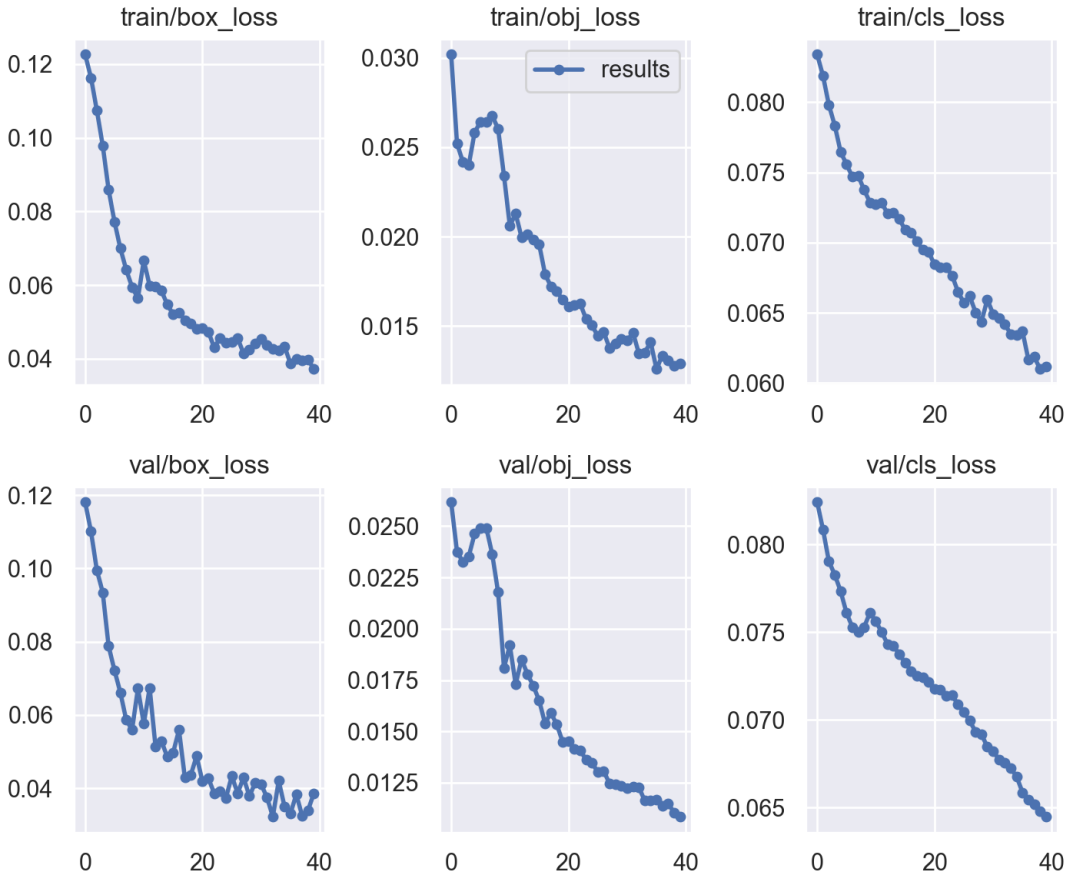
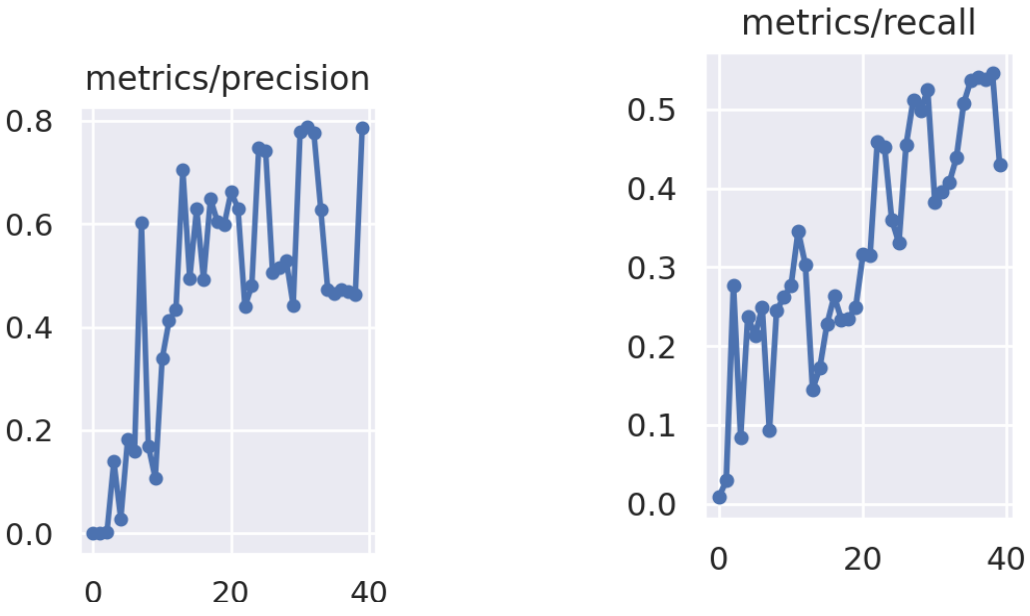


Figure 6.1: Loss plot. Source:model output



(a) Precision Curve.t

(b) Recall Curve.

Figure 6.2: Precision and Recall curve. Source: Model Output

Visualize the Performance of Trained Model

The model was visualized and its performance was judged also on the basis of prediction on validation image set. Here, the paper discusses the prediction result on validation images and present the analysis and reason for positive and negative results obtained. With this in later section it was tried to solve the issue through hyper-parameter optimization.

Lets begin with the figure 6.3. This is the first batch of validation image with true labels and figure 6.4 is the prediction done by the model in those images. Let us compare these two first.

Figure 6.3: First batch validation image with true labels. Source:Model Output

Figure 6.4: First batch validation image with prediction by model. Source:Model Output

The first and third image of 1st batch are correctly predicted with 90% of confidence. where as if we look into image 2nd, 4th,5th and 7th model fails to

correctly predict the correct class. True label was 'curve left' but model predict it as 'curve right'. Model gets confused between left and right curve that can be confirmed by seeing the confidence value. The model predicts it with low confidence below 50%. One simple reason could be the number of training instances. Both the signs are similar looking therefore, needed more number of samples to make model learn the pattern of these two signs more distinctly. Another reason that can be thought of is that the model was trained with default hyper-parameters. Among the so many hyper-parameters one of them is augmentation. During training images are flipped left and right to increase the number of images in context of generalizing the model. Flipping the image results better when we have all unique patterns. But in this case the sign 'curve left' and 'curve right' both are complementary to each other. If one is flipped horizontally looks similar to another. Therefore, it creates confusion and model trained with this augmentation on will not be able to correctly predict between these two. To correct this and not allow to flip the image during training we can change the hyper-parameter setting that is shown in sub section Optimization and tunning of the model.

Similarly, for the 6th image classes are predicted correctly but with low confidence and with two extra prediction that are not labeled. In the original image it contains the sign for 'speed limit 50'. This model was not trained for predicting the 'speed limit 50' sign instead it was trained for predicting the 'speed limit 30' sign. Due to similarity in pattern for both the signs model predict speed limit 50 as 30. If we have close look and analyse the both sign in terms of color and pattern, we can see that both sign are almost same except for the number 3 and 5. Both sign matched each other more than 90% which is why the sign speed 50 is predicted as speed 30. This can be solved by training the model for all the speed limits signs with sufficient number of training samples and correct hyper-parameters.

Finally, the last image of 1st batch validation images. Here, it is found missing of class prediction along with few extra prediction for sign that are located far behind in image. In the image there is a sign that is labeled as 'straight' but model fails to detect the sign. If we see to figure 4.3 the number of instances for training the sign 'straight' is very few (less than 10). As consequence model fails to learn the patter. With so very less samples model can't be trained to generalize any sign correctly for all conditions.

Figure 6.5: Second batch validation image with true labels. Source:Model Output

Figure 6.6: Second batch validation image with prediction by model. Source:Model Output

Likewise, in the second batch of validation images 6.5 and 6.6 the same issue of curve left and right can be found again. Even if the curve left and right are predicted correct but predicted with low confidence. The other thing that was seen in this batch was for 2nd image, where the sign is labeled as crossing but model do not predict anything. The model misses out this sign, this is because the sign in image is not clearly visible. The sign is distorted in some angle also the background is dark lies in between the shadows of trees. Therefore, if the samples with such conditions are not feed during training, model could not be able to trained properly to generalize all those real world situations. Increasing the numbers of samples from variable scenario would solve these issue.

6.2 Hyper-Parameter Optimization

There are too many variations of hyper-parameters to look into. The list can be seen at 4.3.1. Those parameters can be grouped based on their characteristics and can be played around to see the differences in performance. There are bunch of augmentation hyper-parameters (from hsv_h to copy_paste in 4.3.1) that are set to some value which were fit for the coco dataset. As discussed earlier to minimise the confusion between the signs that are complementary to each other flipping of image in any direction need to be discarded. Therefore, the value for fliplr and flipud was set to 0 that means no flipping left-right and up-down. Similarly, rotation can also create such confusion, it was also set to zero (degrees: 0.0 in 4.3.1). Loss gain component are used to scale and balance losses. Reduction in the component like (class loss gain, object loss gain) help to reduce over-fitting in those specific components. Therefore, class loss gain was reduced to 0.3 (cls:0.3 in 4.3.1) and object loss gain was set to 0.7 (obj:0.7 in 4.3.1). For this experiment rest of other parameters were left as unchanged. The model was retrained in the same way as it was done earlier with the above mentioned updates in parameters. This time model was able to reach an mAP of 0.84 which is very good result. Hence, optimization has improved the performance and was accepted.

Visualize the Performance after Tuning

Once again the prediction on validation image set were visualized to check if the issues that appeared before were solved or not. Visualizing the first set of validation and prediction image after optimization, figure 6.8 and 6.9 it was noticed that most of the predictions are true to the original labels. While saying this, there were also few issues seen that cannot be optimized in this case. To address those kind of issues whole process need to be repeated with changes in data number and quality, annotation method and many more. For example, in the second image the sign was labeled as curve right but models predict it as pedestrian from nowhere. We can only speculate some reason like, could be the shadow (dark background) where the model only sees the outer boundary of sign (red triangle here) and made the prediction on the basis of that. Likewise, for the object in the 4th image true class was 'cycle path' but it was predicted as straight right. When zoomed and look into the real image it was seen that cycle path sign was damaged (part of the paint was pilled off) see figure 6.7. That is why model found 'straight' sign as close match after damage of original sign. This also suggest these kind of damage signs need to be repaired and replaced.

Figure 6.7: Zoomed image to show damaged sign. Source:Model Output

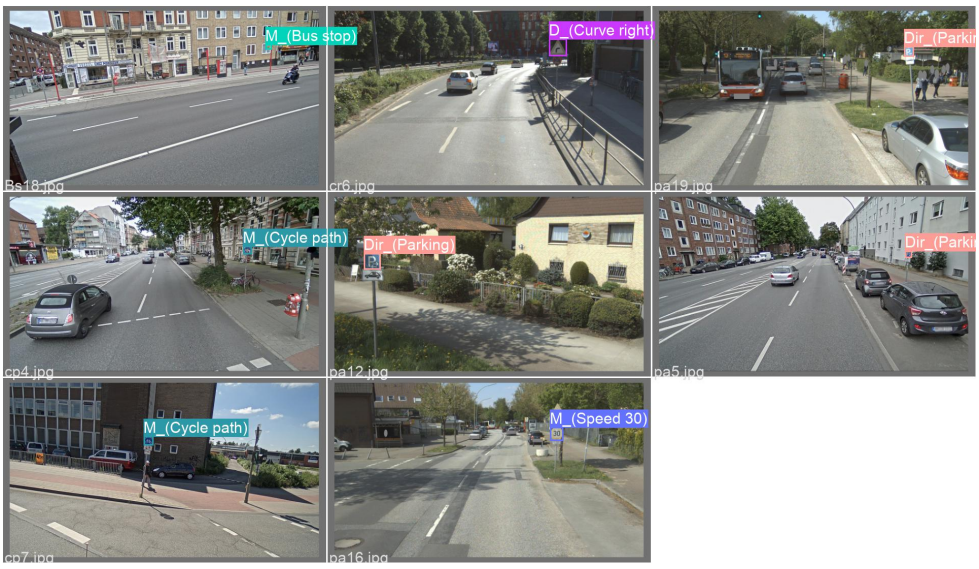


Figure 6.8: First validation batch with labels. Source:Model Output

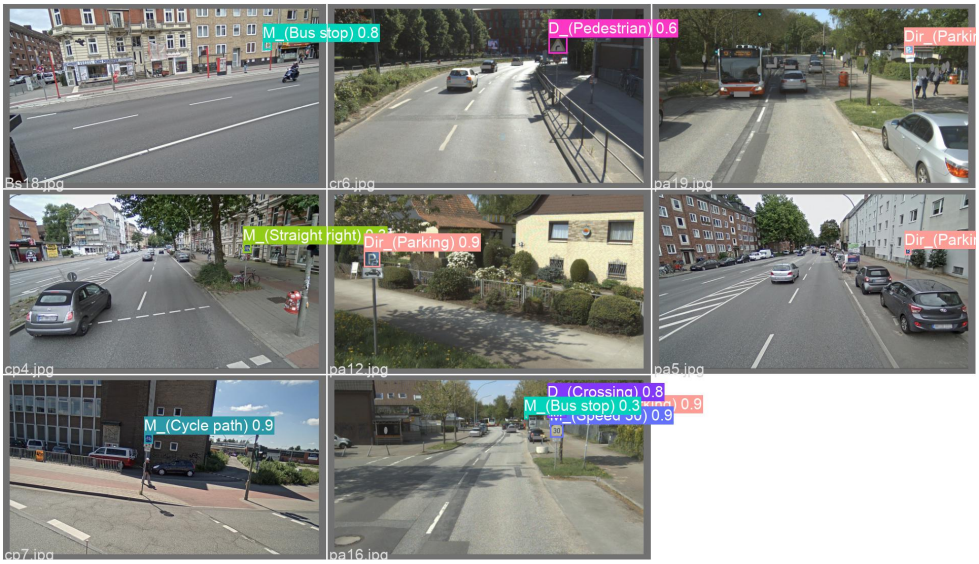


Figure 6.9: Prediction for first validation batch. Source:Model Output

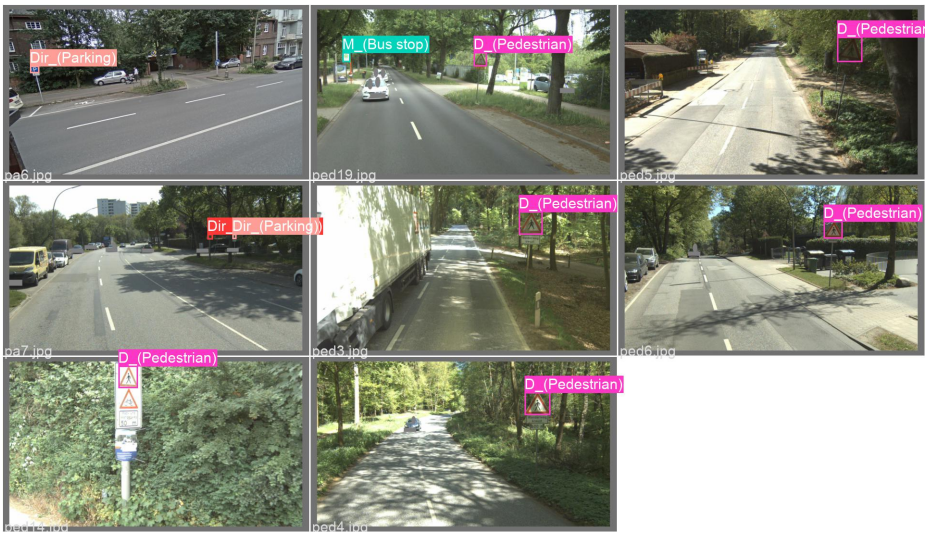


Figure 6.10: Second validation batch with labels.Source:Model Output

Figure 6.11: Prediction for second validation batch.Source:Model Output

For second pair of figure 6.10 and 6.11 class prediction done by model were all correct and matched the true labels. At the same time, model also predict the result for the sign that are not labeled and trained for. Based on the closest match model has predicted wrong class for the sign which were not included in training. This can only be optimized by training simultaneously for all other remaining signs that are left out and not included in training.

Figure 6.12: Third validation batch with labels. Source:Model Output

Figure 6.13: Prediction for third validation batch. Source:Model Output

Similarly, for last pair of validation and prediction image 6.12 and 6.13 most of the prediction are correct. Overall to conclude the visualization, the model performs very well to the signs that are very unique in pattern and doesn't match to other signs. Whereas, for similar looking signs model confuses. For instance 'straight' and 'straight right', 'pedestrian' and 'crossing', and many other. They have similar pattern and color that match each other more than 50%. As from experience through this work, this kind of issues can be addressed by increasing

training samples that cover all the real world situation. One another approach that can be adopted in 'data annotation' process is labeling the image as bounding box and segmentation mask together [see 54, sec 3.2]. Segmentation differentiate between objects with the highest degree of accuracy. It is labor intensive, as it requires pixel level accuracy may take more than 20 minutes to just annotate a single image. Here, each annotated pixel in image belongs to single class. This type of annotation is often used when required higher accuracy.

6.3 Inferencing with Images

Below were the prediction result on test images. In the source field at 4.4, the path to the folder containing the test images were passed. The model took no time to process all those images and generate the following results.


D_(Crossing) 0.87
50 m


M_(End 30 zone) 0.90


M_(End 30 zone) 0.97
M_(Give way) 0.94


M_(Give way) 0.92


Dir_(Parking) 0.87

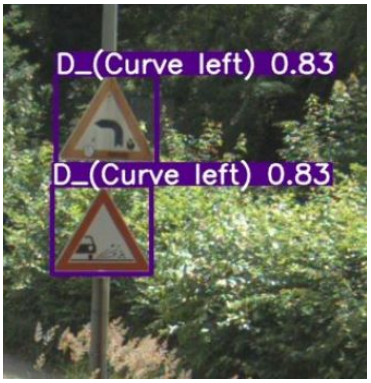

M_(Straight right) 0.74


D_(Crossing) 0.81
M_(Restricted stopping right) 0.95


M_(Cycle path) 0.87
M_(No entry) 0.93
Ende


Dir_(Priority road) 0.92

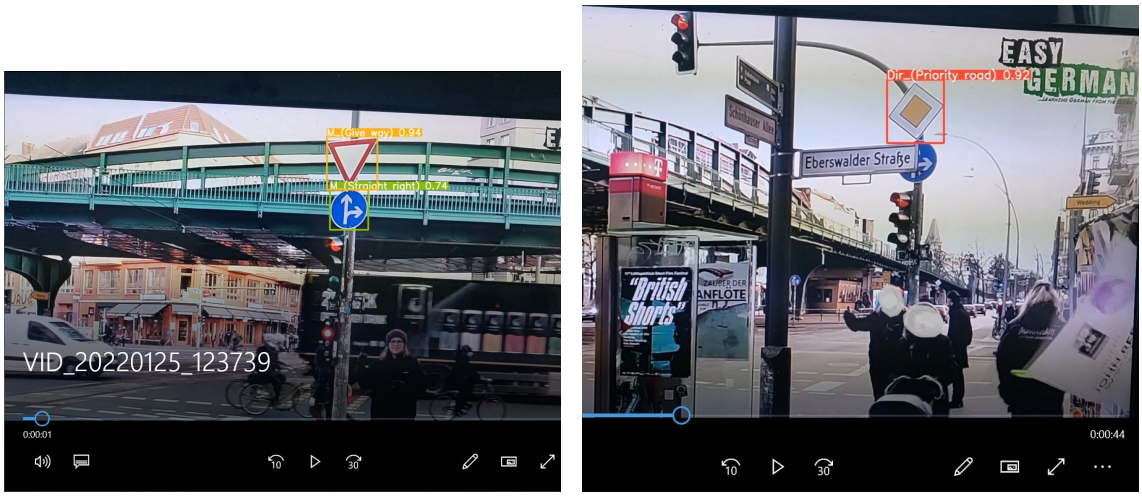

M_(Bus stop) 0.91


D_(Curve left) 0.83
D_(Curve left) 0.83


M_(Stop) 0.91

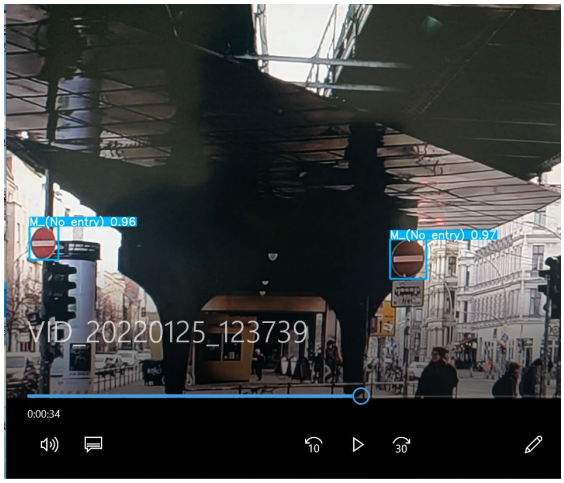
6.4 Inferenccing with Videos and Webcam Feeds

Initially, it was planned to train and inference the model only for images. Later, when come to know that YOLOv5 accepts the videos and also webcam port without needing of training again with videos, it was decide to inference with videos and webcam feeds. Since, there are no videos available, a video was downloaded from YouTube. In the video, one of the tutor was introducing about Germany traffic signs live on the road with real traffic sign standing in the roads. The link to the video is: https://youtu.be/iU2aMi4bLNs. After running inference to this video, model was able to predict the sign that were trained for. The screenshot from the inferenced video is shown in figure 6.22a, 6.22b, 6.23a and 6.23b. In the same way webcam port number (here 0) was passed as source and it display the window showing webcam feed. As the camera was in the room it cannot capture any traffic sign to show detection. To check, the webcam was turned towards the computer screen displaying traffic signs. With no surprise model was able to detect and classify those signs displayed in screen in real time.

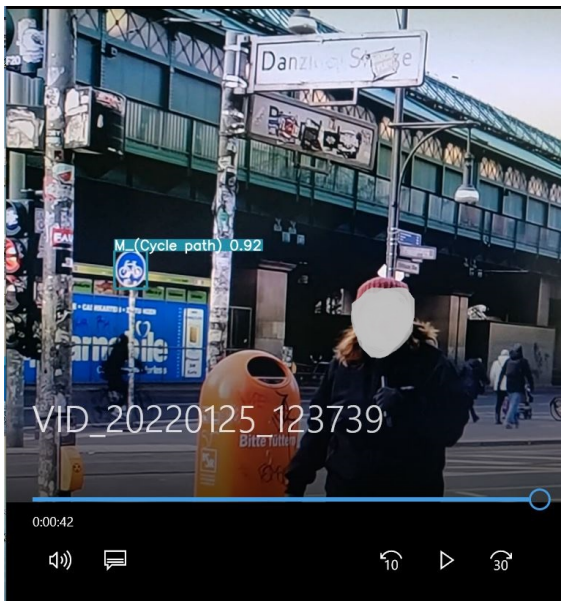


(a) Screenshot from video 1

(b) Screenshot from video 2



(a) Screenshot from video 3



(b) Screenshot from video 4

Chapter 7. Results & Evaluation

After training and testing 'traffic sign detection and recognition model', it was necessary to assess the overall accuracy and performance of of the model. In this chapter, the model evaluation is done based on experimental results. The performance was judged in terms of data requirement, speed, mean average precision (mAP) and number of parameters (size of the model).

7.1 Evaluation interms of Data Requirements

To compare with 3.3.1, this work has been performed with very small amount of data that automatically means instances per class is also very low. Despite of training instances being very low model trained was performing good and was able to recognize signs in unseen data. The model was able to generalize for the given certain variation very well with small set. It is obvious that to generalize for all the variation training samples from all the scenario is needed. Therefore, it can be evaluated that YOLOv5 is good at recognizing pattern even if it is trained with small data set. This could be very useful to the cases where there are only limited data available and need to have detection model.

In the far side, to develop model that are very sensitive and cannot afford to miss object or have wrong prediction, both quality and quantity of data play critical role. To detect the traffic signs where most of the signs looks similar in pattern, are small in sizes and are expose to the open environment that leads to deterioration in some way it require large data set to get close to reality.

7.2 Evaluation interms of Speed, mAP and Number of Parameters

The confusion matrix is shown in the plot 7.1. Class labels in horizontal direction are true labels and the class labels in vertical direction are predicted labels by the model. All the value in diagonal represents true prediction meaning predicted class label matched with true label. Apart from the diagonal all other values represents confusion between the classes. Model was 100 % successful to predict the most of the classes like parking, give way, stop, straight right, No entry, restricted stopping, speed30 and end zone 30. Priority road and cycle path were predicted with more than 80 % and remaining of the signs between (60 to 70) %. As prototype model with limited data, this shows very good performance by the model.

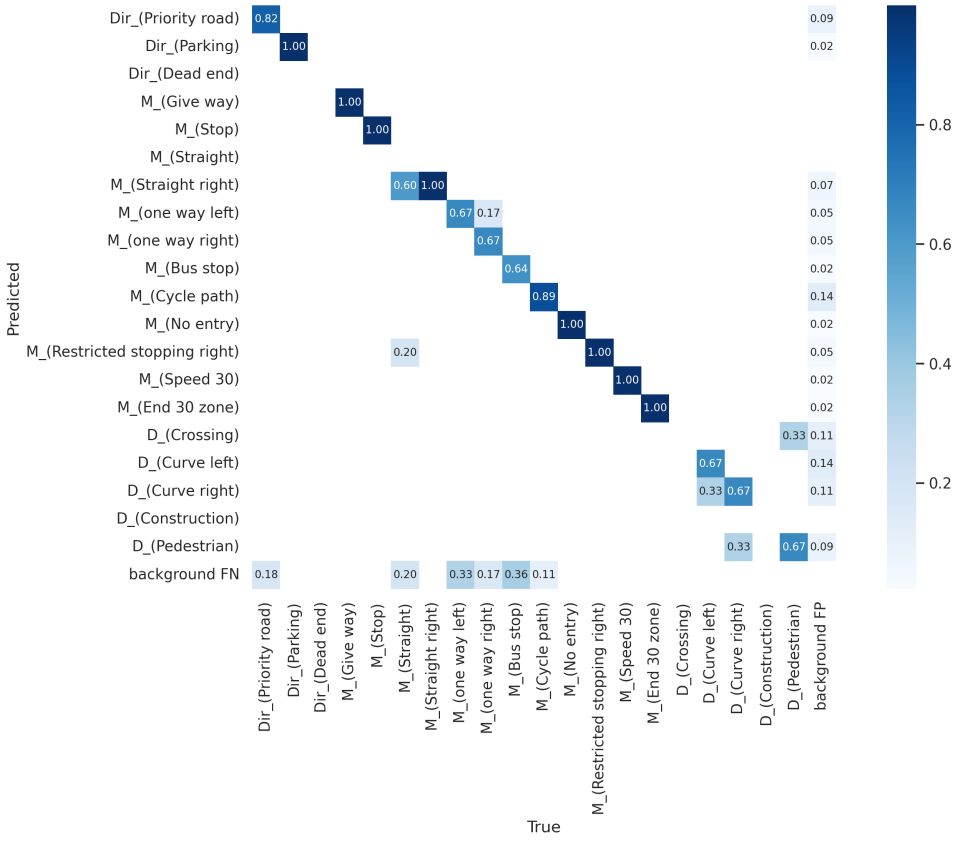


Figure 7.1: Confusion matrix. Source:Model Output

The values shown in the table 7.1 are generated for single model on coco val 2017 data set after the model has been trained for 300 epochs with default setting and hyper-parameters. In this work, having small data set model was able to trained completely with 200 to 300 epochs. The final model was trained with some changes in hyper-parameter i:e flip left right probability is set to 0 default being is 0.5. The overall speed and mAP was produced as shown below. The output generated by the command is presented in tabular form similar to 7.1 to evaluate the model.

mAP value was reproduced by:

!python val.py –data coco128.yaml –img 640 –conf 0.001 –iou 0.65 –device 0 –workers 2 –weights runs/train/exp18/weights/best.pt

Speed value was reproduced by:

!python val.py –data coco128.yaml –img 640 –device 0 –workers 2 –weights runs/train/exp18/weights/best.pt –task speed –batch 32

Comparing the values in both tables 7.1 and 7.2 clearly there is not much different. The inference time of 3.0 ms for large model and 5.2 ms for extra large model verify that YOLOv5 models are very fast with inferencing and can be deployed to real time application. Similarly, mAP of 0.84 and 0.82 verifies how efficient

Model	size (pixels)	mAP^{val} 0.5:0.95	mAP^{val} 0.5	Speed CPU b1 (ms)	Speed V100 b1 (ms)	Speed V100 b32 (ms)	params (M)	FLOPs @640 (B)
YOLOv5n	640	28.4	46.0	**45**	**6.3**	**0.6**	**1.9**	**4.5**
YOLOv5s	640	37.2	56.0	98	6.4	0.9	7.2	16.5
YOLOv5m	640	45.2	63.9	224	8.2	1.7	21.2	49.0
YOLOv5l	640	48.8	67.2	430	10.1	2.7	46.5	109.1
YOLOv5x	640	50.7	68.9	766	12.1	4.8	86.7	205.7

Table 7.1: Pretrained checkpoints for all available models. Source[75]

Model	**size (pixels)**	**mAPval @ 0.5:0.95**	**mAPval @ 0.5**	**Speed b1 (ms)**	**Speed b32 (ms)**	**params (M)**	**FLOPS @ 640 (B)**
YOLOv5l	640	0.66	0.84	14.6	3.0	46.2	108.1
YOLOv5x	640	0.64	0.82	17.9	5.2	86.3	204.4

Table 7.2: Pretrained checkpoints for two selected models in this experiment.

the model was. The individual mean average precision of each class can also be monitored. For this precision vs recall (pr curve) can be looked into. Below is the plot of precision vs recall. For most of the classes individual average precision are excellent and are above 0.9, which means the overall performance for this model, and its prediction was robust.

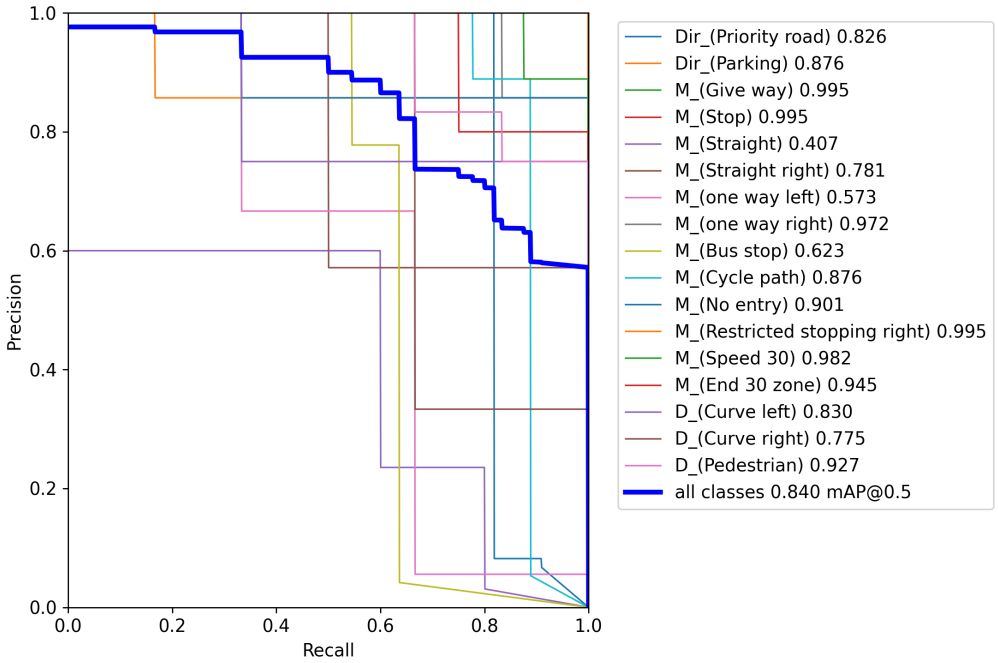


Figure 7.2: PR Curve. Source:Model Output

In next step, similar training setting were setup to evaluate for training time

and model size. The two big model of YOLOv5, YOLOv5l (large) and YOLOv5x (extra large) were trained for 50 epochs to compare their training time and model size. The results are shown in plots 7.3 and 7.4.

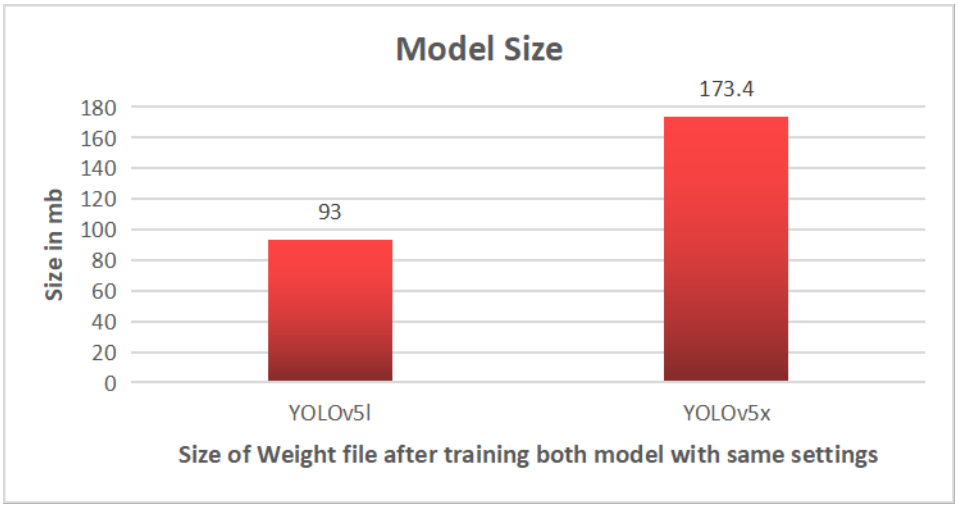


Figure 7.3: weight file size. Source:Output plotted on excel

The size of weight file for YOLOv5l is 93 mb where as for YOLOv5x is 173.4 mb which is very less compare to other state of the art detection model like Fast-RCNN and Faster-RCNN.

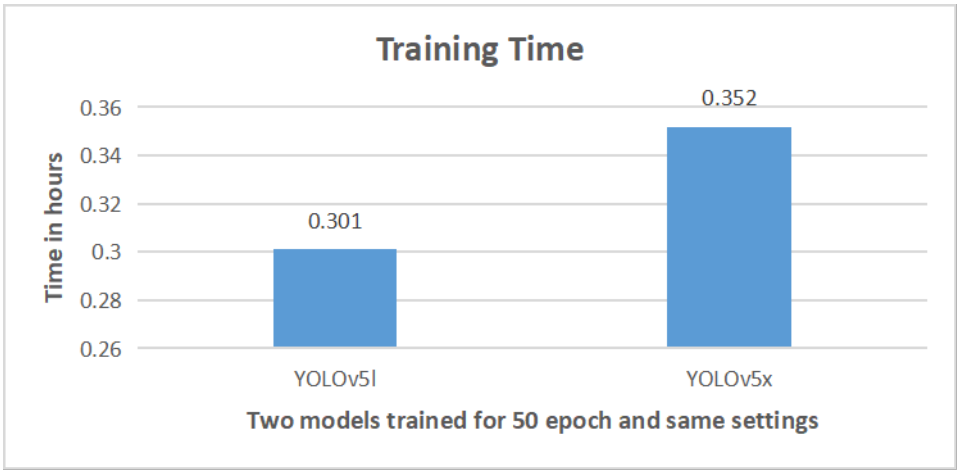


Figure 7.4: Model Training time. Source:Output plotted on excel

Also, YOLOv5l took 18.06 minutes and YOLOv5x took 21.12 minutes to train the model for 50 epochs with Quadro RTX 6000, 22696MiB using 2 data-loader worker. This suggest the training time was fast enough and can be more faster with more high end device with powerful GPU.

Based on this experiment and analysing at facts and results obtained, YOLOv5 definitely is one of the state of art detection model for real time detection and recognition.

Chapter 8. Conclusion

8.1 Conclusion

In this paper, We have build a custom YOLOv5 model for detecting and recognizing small traffic signs in street view images and videos in real time. Overall experience with the Yolov5 was great. Before this work, have only theoretical knowledge about YOLO and its family. With this experiment got an opportunity to explore the structure of YOLO model and use it to create a model for custom and real data set. The result achieved was impressive and proved to be true as explained by the author. It is no doubt the model get trained faster and inference time is real quick to real time. Choosing Yolov5 to create object detection model prove to be fruitful and this work succeeded to satisfaction. There is still no official paper on Yolov5 comparing other benchmark model. It is believed that later on at some point of time paper will be released claiming its outstanding performance. Till then it is still an open area where one can continue the research and release the paper with more detailed analysis and comparisons.

Particularly, speaking to this experiment the model was created just with the 20 different signs and in real field scenario there are hundreds and thousands of different traffic sign having their own significance. It is therefore, important to develop a model which include all the traffic sign that can represent the whole traffic system completely. The aim of this work was to build concept model, looking at the performance (84% mAP) it can be concluded that the work was successful to achieve its objective.

8.2 Area for Further Development

This work lighten ups the possibilities to take this work forward and convert this concept model to real model that can be implemented in many useful application like updating maps with traffic sign and road markings, creating alert system for repair and maintenance of damaged sign in roads and many more. To achieve all these, one most important information about traffic sign is its positioning. Therefore, positioning needs to be defined along with detection and recognition.

The classic automatic traffic sign detection and extraction system includes components for detection, recognition, and positioning. To manage and maintain the regular operation in road system, departments of transportation need reliable and up-to-date information about the location and condition of road traffic signs. Most of the methods developed so far including this work only detect and recognize the traffic signs from the images. Actually, traffic sign images come with additional useful information such as location. Researchers from different fields have developed several methods to realize traffic sign extraction. However, it remains a challenging task to extract coordinates (accurate location information) from a vast amount of traffic sign images.

According to [69] a better solution is to acquire geo-tagged traffic assets and to engage image recognition with GPS information. To achieve this, author proposed two methods to collect the geo-tagged images. One is to download images from Google street view images. However, In some locations Google Street View is not available and for some street views they are currently out of date. Also, a personal user is only allowed to download only 25,000 images per day. To overcome this, GoPro video was chosen as a second source. GoPro is a versatile action camera with a GPS sensor and an Inertial Measurement Unit (IMU). This specific camera can be used to record videos and take images with coordinate information. Where as IMU measures speed (both 2D and 3D speed) and accelerator of camera motion.

The model that we developed in this work can also be used for positioning the traffic sign accurately, provided the images should be geo-tagged with correct coordinates. The first step is to detect and recognize the traffic sign from the geo-tagged image and in second step plot the recognized signs on map with GIS software.

Creating the traffic sign detection, recognition and positioning model with the benchmark data set that represent the whole traffic system covering the all real field scenario, opens up the door for more advance technological development.

Bibliography

[1] Hidehiko Akatsuka and Shinichiro Imai. *Road signposts recognition system.* Tech. rep. SAE Technical Paper, 1987.

[2] Michael J Swain and Dana H Ballard. "Color indexing". In: *International journal of computer vision* 7.1 (1991), pp. 11–32.

[3] Nasser Kehtarnavaz, Norman C Griswold, and DS Kang. "Stop-sign recognition based on color/shape processing". In: *Machine Vision and Applications* 6.4 (1993), pp. 206–208.

[4] Lutz Priese and Volker Rehrmann. "On hierarchical color segmentation and applications". In: *Proceedings of IEEE Conference on Computer Vision and Pattern Recognition.* IEEE. 1993, pp. 633–634.

[5] Giulia Piccioli et al. "Robust method for road sign detection and recognition". In: *Image and Vision Computing* 14.3 (1996), pp. 209–223.

[6] Arturo De La Escalera et al. "Road traffic sign detection and classification". In: *IEEE transactions on industrial electronics* 44.6 (1997), pp. 848–859.

[7] Dariu M Gavrila. "Multi-feature hierarchical template matching using distance transforms". In: *Proceedings. Fourteenth international conference on pattern recognition (Cat. No. 98EX170).* Vol. 1. IEEE. 1998, pp. 439–444.

[8] Mahmoud M Zadeh, T Kasvand, and Ching Y Suen. "Localization and recognition of traffic signs for automated vehicle control systems". In: *Intelligent transportation systems.* Vol. 3207. SPIE. 1998, pp. 272–282.

[9] Dariu M Gavrila. "Traffic sign recognition revisited". In: *Mustererkennung 1999.* Springer, 1999, pp. 86–93.

[10] Salvatore Vitabile et al. "Road signs recognition using a dynamic pixel aggregation technique in the HSV color space". In: *Proceedings 11th International Conference on Image Analysis and Processing.* IEEE. 2001, pp. 572–577.

[11] Hirofumi Ohara et al. "Detection and recognition of road signs using simple layered neural networks". In: *Proceedings of the 9th International Conference on Neural Information Processing, 2002. ICONIP'02.* Vol. 2. IEEE. 2002, pp. 626–630.

[12] Mohamed Benallal and Jean Meunier. "Real-time color segmentation of road signs". In: *CCECE 2003-Canadian Conference on Electrical and Computer Engineering. Toward a Caring and Humane Technology (Cat. No. 03CH37436).* Vol. 3. IEEE. 2003, pp. 1823–1826.

[13] Chiung-Yao Fang et al. "An automatic road sign recognition system based on a computational model of human recognition processing". In: *Computer vision and Image understanding* 96.2 (2004), pp. 237–268.

[14] Jim Torresen, Jorgen W Bakke, and Lukas Sekanina. "Efficient recognition of speed limit signs". In: *Proceedings. The 7th International IEEE Conference on Intelligent Transportation Systems (IEEE Cat. No. 04TH8749).* IEEE. 2004, pp. 652–656.

[15] Claus Bahlmann et al. "A system for traffic sign detection, tracking, and recognition using color, shape, and motion information". In: *IEEE Proceedings. Intelligent Vehicles Symposium, 2005.* IEEE. 2005, pp. 255–260.

[16] Pavel Paclík, Jana Novovicová, and Robert PW Duin. "Building road-sign classifiers using a trainable similarity measure". In: *IEEE Transactions on Intelligent Transportation Systems* 7.3 (2006), pp. 309–321.

[17] Igor Kononenko and Matjaz Kukar. *Machine learning and data mining.* Horwood Publishing, 2007.

[18] Saturnino Maldonado-Bascón et al. "Road-sign detection and recognition based on support vector machines". In: *IEEE transactions on intelligent transportation systems* 8.2 (2007), pp. 264–278.

[19] Yok-Yen Nguwi and Abbas Z Kouzani. "Detection and classification of road signs in natural environments". In: *Neural computing and applications* 17.3 (2008), pp. 265–289.

[20] Yehua Sheng et al. "Automatic detection and recognition of traffic signs in stereo images based on features and probabilistic neural networks". In: *Optical and Digital Image Processing.* Vol. 7000. SPIE. 2008, pp. 459–470.

[21] Benjamin Hoferlin and Klaus Zimmermann. "Towards reliable traffic sign recognition". In: *2009 IEEE Intelligent Vehicles Symposium.* IEEE. 2009, pp. 324–329.

[22] Ying Li, Sharath Pankanti, and Weiguang Guan. "Real-time traffic sign detection: an evaluation study". In: *2010 20th International Conference on Pattern Recognition.* IEEE. 2010, pp. 3033–3036.

[23] Victor Adrian Prisacariu et al. "Integrating object detection with 3D tracking towards a better driver assistance system". In: *2010 20th International Conference on Pattern Recognition.* IEEE. 2010, pp. 3344–3347.

[24] Zuzana Reitermanova. "Data splitting". In: *WDS.* Vol. 10. 2010, pp. 31–36.

[25] Andrzej Ruta, Yongmin Li, and Xiaohui Liu. "Real-time traffic sign recognition from video by class-specific discriminative features". In: *Pattern Recognition* 43.1 (2010), pp. 416–430.

[26] Sergio Escalera et al. "Background on traffic sign detection and recognition". In: *Traffic-sign recognition systems.* Springer, 2011, pp. 5–13.

[27] MA Garcia-Garrido et al. "Robust traffic signs detection by means of vision and V2I communications". In: *2011 14th International IEEE Conference on Intelligent Transportation Systems (ITSC).* IEEE. 2011, pp. 1003–1008.

[28] Gary Overett and Lars Petersson. "Large scale sign detection using HOG feature variants". In: *2011 IEEE intelligent vehicles symposium (IV).* IEEE. 2011, pp. 326–331.

[29] J. Stallkamp M. Schlipsing J. Salmen and C. Igel. "The German Traffic Sign Recognition Benchmark: A multi-class classification competition". In: *In Neural Networks (IJCNN) The 2011 International Joint Conference* (2011), pp. 145 3–1460.

[30] Yang Siyan, Wu Xiaoying, and Miao Qiguang. "Road-sign segmentation and recognition in natural scenes". In: *2011 IEEE International Conference on Signal Processing, Communications and Computing (ICSPCC).* IEEE. 2011, pp. 1–4.

[31] Usman Zakir. "Automatic road sign detection and recognition". PhD thesis. Loughborough University, 2011.

[32] Jack Greenhalgh and Majid Mirmehdi. "Real-time detection and recognition of road traffic signs". In: *IEEE transactions on intelligent transportation systems* 13.4 (2012), pp. 1498–1506.

[33] Jack Greenhalgh and Majid Mirmehdi. "Traffic sign recognition using MSER and random forests". In: *2012 Proceedings of the 20th European Signal Processing Conference (EUSIPCO)*. IEEE. 2012, pp. 1935–1939.

[34] Ahmed Hechri and Abdellatif Mtibaa. "Automatic detection and recognition of road sign for driver assistance system". In: *2012 16th IEEE Mediterranean Electrotechnical Conference*. IEEE. 2012, pp. 888–891.

[35] Andreas Mogelmose, Mohan Manubhai Trivedi, and Thomas B Moeslund. "Vision-based traffic sign detection and analysis for intelligent driver assistance systems: Perspectives and survey". In: *IEEE Transactions on Intelligent Transportation Systems* 13.4 (2012), pp. 1484–1497.

[36] J. Stallkamp M. Schlipsing J. Salmen and C. Igel. "Man vs. computer: Benchmarking machine learning algorithms for traffic sign recognition." In: *Neural Networks* (2012).

[37] Gauri A Tagunde, NJ Uke, and C Banchhor. "Detection, classification and recognition of road traffic signs using color and shape features". In: *Int. J. Adv. Technol. Eng. Res* 2.4 (2012), pp. 202–206.

[38] Fatin Zaklouta and Bogdan Stanciulescu. "Real-time traffic-sign recognition using tree classifiers". In: *IEEE Transactions on Intelligent Transportation Systems* 13.4 (2012), pp. 1507–1514.

[39] *Detection of Traffic Signs in Real-World Images: The German Traffic Sign Detection Benchmark*. 2013. URL: http://benchmark.ini.rub.de/?section=gtsdb&subsection=news.

[40] *Detection of Traffic Signs in Real-World Images: The German Traffic Sign RECOGNITION Benchmark*. 2013. URL: http://benchmark.ini.rub.de/?section=gtsrb&subsection=news.

[41] Chun-Fu Lin and Sheng-Fuu Lin. "Accuracy enhanced thermal face recognition". In: *Infrared Physics & Technology* 61 (2013), pp. 200–207.

[42] Min Lin, Qiang Chen, and Shuicheng Yan. "Network in network. CoRR abs/1312.4400 (2013)". In: *arXiv preprint arXiv:1312.4400* (2013).

[43] Jung-Guk Park and Kyung-Joong Kim. "Design of a visual perception model with edge-adaptive Gabor filter and support vector machine for traffic sign detection". In: *Expert Systems with Applications* 40.9 (2013), pp. 3679–3687.

[44] Ali Behloul and Yassmina Saadna. "A fast and robust traffic sign recognition". In: *International Journal of Innovation and Applied Studies* 5.2 (2014), p. 139.

[45] Ross Girshick et al. "Rich feature hierarchies for accurate object detection and semantic segmentation". In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2014, pp. 580–587.

[46] Ross Girshick. "Fast r-cnn". In: *Proceedings of the IEEE international conference on computer vision*. 2015, pp. 1440–1448.

[47] Kaiming He et al. “Spatial pyramid pooling in deep convolutional networks for visual recognition”. In: *IEEE transactions on pattern analysis and machine intelligence* 37.9 (2015), pp. 1904–1916.

[48] Karel Lenc and Andrea Vedaldi. “R-cnn minus r”. In: *arXiv preprint arXiv:1506.06981* (2015).

[49] Shaoqing Ren et al. “Faster r-cnn: Towards real-time object detection with region proposal networks”. In: *Advances in neural information processing systems* 28 (2015), pp. 91–99.

[50] Safat B Wali et al. “An automatic traffic sign detection and recognition system based on colour segmentation, shape matching, and svm”. In: *Mathematical Problems in Engineering* 2015 (2015).

[51] Wei Liu et al. “Ssd: Single shot multibox detector”. In: *European conference on computer vision*. Springer. 2016, pp. 21–37.

[52] Joseph Redmon et al. “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.

[53] Christian Szegedy et al. “Rethinking the inception architecture for computer vision”. In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2016, pp. 2818–2826.

[54] Zhe Zhu et al. “Traffic-sign detection and classification in the wild”. In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2016, pp. 2110–2118.

[55] Christo Ananth. “Iris recognition using active contours”. In: *International Journal of Advanced Research in Innovative Discoveries in Engineering and Applications [IJARIDEA]* 2.1 (2017), pp. 27–32.

[56] Joseph Redmon and Ali Farhadi. “YOLO9000: better, faster, stronger”. In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2017, pp. 7263–7271.

[57] Zhuo Zhang et al. “Faster R-CNN for small traffic sign detection”. In: *CCF Chinese Conference on Computer Vision*. Springer. 2017, pp. 155–165.

[58] Golnaz Ghiasi, Tsung-Yi Lin, and Quoc V Le. “Dropblock: A regularization method for convolutional networks”. In: *arXiv preprint arXiv:1810.12890* (2018).

[59] Shu Liu et al. “Path aggregation network for instance segmentation”. In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2018, pp. 8759–8768.

[60] Kay Thinzar Phu and Lwin Lwin Oo. “RGB Color based Myanmar Traffic Sign Recognition System from Real-time Video”. In: *International Journal of Information Technology (IJIT)* 4.4 (2018).

[61] Joseph Redmon and Ali Farhadi. “Yolov3: An incremental improvement”. In: *arXiv preprint arXiv:1804.02767* (2018).

[62] Faming Shao et al. “Real-time traffic sign detection and recognition method based on simplified Gabor wavelets and CNNs”. In: *Sensors* 18.10 (2018), p. 3192.

[63] Xudong Sun, Pengcheng Wu, and Steven CH Hoi. "Face detection using deep learning: An improved faster RCNN approach". In: *Neurocomputing* 299 (2018), pp. 42–50.

[64] Sanghyun Woo et al. "Cbam: Convolutional block attention module". In: *Proceedings of the European conference on computer vision (ECCV)*. 2018, pp. 3–19.

[65] Jason Brownlee. *How Do Convolutional Layers Work in Deep Learning Neural Networks?* 2019. URL: https://machinelearningmastery.com/convolutional-layers-for-deep-learning-neural-networks/.

[66] Benhe Gao, Zhongjun Jiang, and Jiaman zhang. "Traffic sign detection based on ssd". In: *Proceedings of the 2019 4th International Conference on Automation, Control and Robotics Engineering*. 2019, pp. 1–6.

[67] Diganta Misra. "Mish: A self regularized non-monotonic neural activation function". In: *arXiv preprint arXiv:1908.08681* 4 (2019), p. 2.

[68] Faming Shao et al. "Improved faster R-CNN traffic sign detection based on a second region of interest and highly possible regions proposal network". In: *Sensors* 19.10 (2019), p. 2288.

[69] Zihao Wu. "Computer Vision-Based Traffic Sign Detection and Extraction: A Hybrid Approach Using GIS And Machine Learning". In: (2019).

[70] Karlijn Alderliesten. *YOLOv3 — Real-time object detection*. 2020. URL: https://medium.com/analytics-vidhya/yolov3-real-time-object-detection-54e69037b6d0.

[71] Uri Almog. *YOLO V3 Explained*. 2020. URL: https://towardsdatascience.com/yolo-v3-explained-ff5b850390f.

[72] Alexey Bochkovskiy, Chien-Yao Wang, and Hong-Yuan Mark Liao. "Yolov4: Optimal speed and accuracy of object detection". In: *arXiv preprint arXiv:2004.10934* (2020).

[73] Kejiang Chen et al. "Self-supervised adversarial training". In: *ICASSP 2020-2020 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*. IEEE. 2020, pp. 2218–2222.

[74] DLR. *ViVre Project*. 2020-2022. URL: https://verkehrsforschung.dlr.de/en/projects/vivre-virtual-stops-automated-traffic-future.

[75] Glenn-jocher. *ultralytics yolov5*. 2020. URL: https://github.com/ultralytics/yolov5.

[76] Bartosz Szabłowski. *How Convolutional Neural Network works*. 2020. URL: https://towardsdatascience.com/how-convolutional-neural-network-works-cdb58d992363.

[77] Mingxing Tan, Ruoming Pang, and Quoc V Le. "Efficientdet: Scalable and efficient object detection". In: *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*. 2020, pp. 10781–10790.

[78] Chien-Yao Wang et al. "CSPNet: A new backbone that can enhance learning capability of CNN". In: *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition workshops*. 2020, pp. 390–391.

[79] Zhaohui Zheng et al. "Distance-IoU loss: Faster and better learning for bounding box regression". In: *Proceedings of the AAAI Conference on Artificial Intelligence*. Vol. 34. 07. 2020, pp. 12993–13000.

[80] Louis Calvin Touko Tcheumadjeu et al. "New Concepts to improve Mobility by Digitization and Virtualization: An Analysis and Evaluation of the Technical Feasibility". In: *EAI INTSYS 2021 Proceedings* (2021).

[81] Baojun Zhang et al. "Detecting small chinese traffic signs via improved yolov3 method". In: *Mathematical Problems in Engineering* 2021 (2021).

[82] Akshay Atam. *Architecture of YOLOv3*. URL: https://iq.opengenus.org/architecture-of-yolov3/.

[83] McKinsey & Company. *Autonomus Driving*. URL: https://www.mckinsey.com/features/mckinsey-center-for-future-mobility/overview/autonomous-driving.

[84] *Name of traffic signs according to LGV - Hamburg*. URL: https://www.bast.de/BASt_2017/DE/Verkehrstechnik/Fachthemen/v1-verkehrszeichen/unterseiten/bezeichnung-der-vz.pdf?__blob=publicationFile&v=17.

[85] TomTom. *Intelligent Speed Assistance*. URL: https://www.tomtom.com/use-cases/intelligent-speed-assistance/.

[86] *Traffic sign and symbol classification according to LGV - Hamburg*. URL: https://www.bast.de/BASt_2017/DE/Verkehrstechnik/Fachthemen/v1-verkehrszeichen/vz-download.html?nn=1837978.

[87] Mandikal Vikram. "A YOLO Based Approach for Traffic Sign Detection". In: *National Institute of Technology Karnataka [April 2, 2018][Online] Available: https://Vikram-mm. github. io/yolo_ report. pdf* ().

Appendix A. Python Code

1 Traffic sign recognition in road images

Pip need to be upgraded and updated. Therefore ddepending on the machine and system try to update it. following is the few methods. YOLOv5 runs on top of PyTorch

[1]:
```
!python -m pip install --upgrade pip
!pip install tensorflow==2.3.1
!pip install tensorboard==2.4.1
!pip install torch
```

[ ]:
```
#!python -m ensurepip
```

[3]:
```
#!python -m pip install -U pip
```

[11]:
```
#pip install torch
```

Import of required libraries YOLOv5 implemented using pytorch so we need to import it to the notebook

[1]:
```
import torch
```

[2]:
```
from IPython.display import Image #this is to render predictions
```

We need to clone the YOLOv5 repo inorder to implement YOLOv5 on custom data set. After cloning inter to YOLOv5 folder directory and install all the libraries from requirements.txt

[4]:
```
!git clone https://github.com/ultralytics/yolov5
```

[5]:
```
%cd yolov5
```

[6]:
```
!pip install -r requirements.txt
```

2 Divide the dataset into train and validation folder.

[30]:
```
import os
from random import choice
import shutil
#arrays to store file names
imgs =[]
xmls =[]
#setup dir names
trainPath = 'C:
 ↪\\Users\\MSI\\Desktop\\Traffic_sign_detection\\dataset\\images␣
 ↪\\train'
valPath = 'C:
 ↪\\Users\\MSI\\Desktop\\Traffic_sign_detection\\dataset\\images␣
 ↪\\val'
crsPath = 'C:
 ↪\\Users\\MSI\\Desktop\\Traffic_sign_detection\\data\\'
#dir where images and annotations stored
#setup ratio (val ratio = rest of the files in origin dir after␣
 ↪splitting into train and test)
train_ratio = 0.8
val_ratio = 0.2
#total count of imgs
totalImgCount = len(os.listdir(crsPath))/2
#soring files to corresponding arrays
for (dirname, dirs, files) in os.walk(crsPath):
    for filename in files:
        if filename.endswith('.txt'):
            xmls.append(filename)
        else:
            imgs.append(filename)
#counting range for cycles
countForTrain = int(len(imgs)*train_ratio)
countForVal = int(len(imgs)*val_ratio)
print("training images are : ",countForTrain)
print("Validation images are : ",countForVal)
```

```
training images are :  269
```

```
Validation images are :  67
```

[31]:
```
trainimagePath = 'C:
↪\\Users\\MSI\Desktop\\Traffic_sign_detection\\dataset\\images␣
↪\\train'
trainlabelPath = 'C:
↪\\Users\\MSI\\Desktop\\Traffic_sign_detection\\dataset\\labels␣
↪\\train'
valimagePath = 'C:
↪\\Users\\MSI\Desktop\\Traffic_sign_detection\\dataset\\images␣
↪\\val'
vallabelPath = 'C:
↪\\Users\\MSI\Desktop\\Traffic_sign_detection\\dataset\\labels␣
↪\\val'
#cycle for train dir
for x in range(countForTrain):
    fileJpg = choice(imgs) # get name of random image from␣
↪origin dir
    fileXml = fileJpg[:-4] +'.txt' # get name of corresponding␣
↪annotation file
    #move both files into train dir
    #shutil.move(os.path.join(crsPath, fileJpg), os.path.
↪join(trainimagePath, fileJpg))
    #shutil.move(os.path.join(crsPath, fileXml), os.path.
↪join(trainlabelPath, fileXml))
    shutil.copy(os.path.join(crsPath, fileJpg), os.path.
↪join(trainimagePath, fileJpg))
    shutil.copy(os.path.join(crsPath, fileXml), os.path.
↪join(trainlabelPath, fileXml))
    #remove files from arrays
    imgs.remove(fileJpg)
    xmls.remove(fileXml)
```

[32]:
```
```

```
#cycle for test dir
for x in range(countForVal):
    fileJpg = choice(imgs) # get name of random image from origin dir
    fileXml = fileJpg[:-4] +'.txt' # get name of corresponding annotation file
    #move both files into train dir
    #shutil.move(os.path.join(crsPath, fileJpg), os.path.join(valimagePath, fileJpg))
    #shutil.move(os.path.join(crsPath, fileXml), os.path.join(vallabelPath, fileXml))
    shutil.copy(os.path.join(crsPath, fileJpg), os.path.join(valimagePath, fileJpg))
    shutil.copy(os.path.join(crsPath, fileXml), os.path.join(vallabelPath, fileXml))
    #remove files from arrays
    imgs.remove(fileJpg)
    xmls.remove(fileXml)
#rest of files will be validation files, so rename origin dir to val dir
#os.rename(crsPath, valPath)
shutil.move(crsPath, valPath)
```

[32]:
```
'C:\\Users\\MSI\\Desktop\\Traffic_sign_detection\\dataset\\images\\val\\data'
```

3 Create dataset.yaml

3.0.1 Edit the coco128.yaml file inside yolov5/data directory as shown below to train on custom data set.

path: C:/Users/MSI/Desktop/Traffic_sign_detection (dataset root dir)
train: dataset/images/train (provide relative path to training images folder)
images val: dataset/images/val (provide relative path to validation images folder)
images test: dataset/test (provide relative path to test images folder. This is optional.)

4 Classes

nc: 20 (number of classes)
(Classes names): [‘Dir_(Priority road)’, ‘Dir_(Parking)’, ‘Dir_(Dead end)’, ‘M_(Give

way)', 'M_(Stop)', 'M_(Straight)', 'M_(Straight right)', 'M_(one way left)', 'M_(one way right)', 'M_(Bus stop)', 'M_(Cycle path)', 'M_(No entry)', 'M_(Restricted stopping right)', 'M_(Speed 30)', 'M_(End 30 zone)', 'D_(Crossing)', 'D_(Curve left)', 'D_(Curve right)','D_(Construction)', 'D_(Pedestrian)']

[7]:
```
# Train YOLOv5s on COCO128 for 3 epochs
!python train.py --img 640 --batch 4 --epochs 50 --data coco128.
 ↪yaml --weights yolov5l.pt
 --cfg models/yolov5l.yaml --hyp data/hyps/hyp.scratch.yaml␣
 ↪--device 0
```

4.1 At the end of the training, two files should are saved in yolov5/runs/train/exp/weights:

last.pt and best.pt. We'll use best.pt for inferencing and last.pt to further train the model from the same point where we left. we just need to replace –weights yolov5l.pt to last.pt

4.2 Explore the metrics recorded during training.

we can use TensorBoard, a very interactive exploration tool we can also use wandb which is also recomnded by YOLOv5 author.

[8]:
```
%load_ext tensorboard
%tensorboard --logdir runs/train
```

4.3 Let's explore now how confident our model is. We can plot a validation batch obtained during training and inspect the confidence score of each label

[9]:
```
Image(filename='runs/train/exp2/val_batch1_labels.jpg',␣
 ↪width=1000)
```

[10]:
```
Image(filename='runs/train/exp2/val_batch1_pred.jpg',␣
 ↪width=1000)
```

You'll be implementing the detect.py script with the best.pt weights and image dimensions of 640x640 pixels (it's really important to comply with that). The results will be saved to runs/detect/exp. To display the results, run the following code:

[11]:
```
#!python detect.py --source runs/train/exp/testimg.jpg( can also be vidoe file directore can be web cam port)
#--weights runs/train/exp/weights/best.pt --conf 0.25 ( this we can change according to your wish and task requirements)
!python detect.py --source VID_20220125_123739.mp4 --weights runs/train/exp/weights/best.pt
--conf 0.50 --save-txt --save-conf --device 0
```

[12]:
```
# visualize the inference result
Image(filename='runs/detect/exp3/cl31.jpg', width=640)
```

[45]:
```
from utils.plots import plot_results
plot_results('runs\\train\\exp10\\results.csv')
#from utils.plots import plot_results
#plot_results(save_dir='runs/train/exp8')  # plot results.txt as results.png
```

[ ]:
```
```