Warsaw University of Technology

FACULTY OF
ELECTRONICS AND INFORMATION TECHNOLOGY

Institute of Computer Science

Bachelor's diploma thesis

in the field of study Computer Science
and specialisation Computer System Networks

Evaluation of Modern Object Detection Models for Deep Neural Networks

Keshav Dandeva
student record book number 302333

thesis supervisor
Radlak Krystian

WARSAW 2022

Evaluation of Modern Object Detection Models for Deep Neural Networks

Abstract. With the rapid evolution of deep convolutional neural networks (CNN), major breakthroughs have been achieved in object detection in the field of computer vision. However, the majority of state-of-the-art detectors, both in one-stage and two-stage methods, have limits and are inadequate for usage in a real-world setting where each step must be thoroughly checked. This thesis investigates advanced object detection models and frameworks. It offers an in-depth analysis of the most recent object detection models, their frameworks, and the performance criteria used to evaluate such models. The object detection models selected for this thesis are YOLOv5, Faster R-CNN using Detectron2, and SSD using TensorFlow 2, and the dataset selected is the Vehicles-OpenImages Dataset. The performance of the selected models in relation to many metrics is analyzed, and the findings are reported. In conclusion, the benefits and limits of the selected models, as well as their relative performance, are discussed.

Keywords: Object Detection, Computer Vision, Deep Convolutional Neural Network, CNN, YOLOv5, Faster R-CNN, SSD, Detectron2, TensorFlow 2

Ocena Współczesnych Modeli Detekcji Obiektów Wykorzystujących Głębokie Sieci Neuronowe

Streszczenie. Dzięki szybkiemu rozwojowi konwolucyjnych sieci neuronowych (CNN), dokonano znacznego przełomu w detekcji obiektów z wykorzytaniem wizji komputerowej. Jednak większość najnowocześniejszych modeli detektorów, zarówno jedno- jak i dwuetapowych, posiada szereg organiczeń i nie nadaje się do wykorzystania w warunkach rzeczywistych, gdzie każdy etap musi być dokładnie sprawdzony. Celem niniejszej pracy jest analiza zaawansowanych modeli oraz architektur sieci neuronowych dedykowanych do problemu detekcji obiektów. Oferuje ona dokładną analizę najnowszych modeli detekcji obiektów, ich ram oraz kryteriów wydajności używanych do oceny takich modeli. Modele detekcji obiektów wybrane do tej pracy to YOLOv5, Faster R-CNN wykorzystująca bibliotekę Detectron2 oraz SSD wykorzystująca bibliotekę TensorFlow 2, natomiast wybrany zbiór danych to Vehicles-OpenImages Dataset. W ramach pracy przedstawiona została analiza wydajności wybranych modeli z wykorzystaniem wielu metryk. W podsumowaniu omówiono korzyści i ograniczenia dla wybranych do badań modeli, jak również wpływ poszczególnych parametrów na ich skuteczność.

Słowa kluczowe: Detekcja obiektów, wizja komputerowa, Głębokie konwolucyjne sieci neuronowe, CNN, YOLOv5, Faster R-CNN, SSD, Detectron2, TensorFlow 2

załącznik nr 10 do zarządzenia
nr 46 /2016 Rektora PW

Politechnika Warszawska
Warsaw University of Technology

Warsaw, 09 September 2022
miejscowość i data
place and date

……Keshav Dandeva……..
imię i nazwisko studenta
name and surname of the student
……302333……..
numer albumu
student record book number
……Computer Science…….
kierunek studiów
field of study

OŚWIADCZENIE

DECLARATION

Świadomy/-a odpowiedzialności karnej za składanie fałszywych zeznań oświadczam, że niniejsza praca dyplomowa została napisana przeze mnie samodzielnie, pod opieką kierującego pracą dyplomową.
Under the penalty of perjury, I hereby certify that I wrote my diploma thesis on my own, under the guidance of the thesis supervisor.

Jednocześnie oświadczam, że:
I also declare that:

- niniejsza praca dyplomowa nie narusza praw autorskich w rozumieniu ustawy z dnia 4 lutego 1994 roku o prawie autorskim i prawach pokrewnych (Dz.U. z 2006 r. Nr 90, poz. 631 z późn. zm.) oraz dóbr osobistych chronionych prawem cywilnym,
- *this diploma thesis does not constitute infringement of copyright following the act of 4 February 1994 on copyright and related rights (Journal of Acts of 2006 no. 90, item 631 with further amendments) or personal rights protected under the civil law,*

- niniejsza praca dyplomowa nie zawiera danych i informacji, które uzyskałem/-am w sposób niedozwolony,
- *the diploma thesis does not contain data or information acquired in an illegal way,*

- niniejsza praca dyplomowa nie była wcześniej podstawą żadnej innej urzędowej procedury związanej z nadawaniem dyplomów lub tytułów zawodowych,
- *the diploma thesis has never been the basis of any other official proceedings leading to the award of diplomas or professional degrees,*

- wszystkie informacje umieszczone w niniejszej pracy, uzyskane ze źródeł pisanych i elektronicznych, zostały udokumentowane w wykazie literatury odpowiednimi odnośnikami,
- *all information included in the diploma thesis, derived from printed and electronic sources, has been documented with relevant references in the literature section,*

- znam regulacje prawne Politechniki Warszawskiej w sprawie zarządzania prawami autorskimi i prawami pokrewnymi, prawami własności przemysłowej oraz zasadami komercjalizacji.
- *I am aware of the regulations at Warsaw University of Technology on management of copyright and related rights, industrial property rights and commercialisation.*

Oświadczam, że treść pracy dyplomowej w wersji drukowanej, treść pracy dyplomowej zawartej na nośniku elektronicznym (płycie kompaktowej) oraz treść pracy dyplomowej w module APD systemu USOS są identyczne.

I certify that the content of the printed version of the diploma thesis, the content of the electronic version of the diploma thesis (on a CD) and the content of the diploma thesis in the Archive of Diploma Theses (APD module) of the USOS system are identical.

.....

czytelny podpis studenta
legible signature of the student

Contents

1. Introduction

Humans are capable of recognizing and locating objects in an image with the blink of an eye. The human visual system can carry out difficult tasks such as object identification and obstacle detection quickly and accurately and with minimal conscious effort. It would have been impossible to think that computers can do even half of that a couple of decades ago. Today however, due to the ease of access to enormous quantities of data, faster machines to process the data, and improved algorithms, computers can be trained with almost no effort for detecting objects and labeling them for their respective classes in an image with high accuracy [1]. Since approximately 20 years, object detection has been evolving in working on detecting occurrences of visual objects belonging to a specific class in digital images. As a very demanding issue in computer vision it is aiming to answer one question: "What objects are where?". This goal is trying to be accomplished by developing diverse computational models and techniques using state of the art technologies like machine learning, artificial intelligence and so on. Additionally, it constitutes the core foundation of numerous additional computer vision tasks including captioning of an image, instance segmentation, image captioning, tracking an object and-so-forth.
In addition to its importance in computer vision, object detection has a strong real-life importance nowadays. It is extensively applied in the real world in areas such as criminal investigation and autonomous driving (and robot vision). It plays an important, sometimes even crucial role in those domains as it reduces human involvement in certain very important processes therefore reducing errors. For instance, autonomous driving pursues to decrease the number of traffic accidents and optimizing traffic by reducing human involvement in the driving process. Going hand in hand with a crucial part of the autonomous driving technology is traffic sign and traffic light detection as well as pedestrian detection. It also improves life quality of people with different kinds of impairment, for example, text detection which allows visually impaired people to "read" street signs and currency.
Object detection may be less prone to error than humans but it is still far from perfect. In each of the above mentioned areas, object detection is facing some difficulties. The size of the pedestrians in the image or video can be too small, faces of people can be blocked partially or completely by other objects, texts can vary in fonts and languages and the image quality of traffic lights can be negatively impacted by bad weather conditions. These are just a few of many examples of challenges that can impair the accuracy of object detection. However, as mentioned, many types of object detection play crucial roles in real-world application and a wrong prediction can cause serious harm. Thus, in order to utilize the object detection model to its best capacity, it is vital to comprehend both its strengths and weaknesses.
With the advent of deep neural networks, a drastic improvement in the performance of object detection can be seen but still many of the methods are unable to measure how

certain they are in their predictions. This becomes a considerable problem when a model is encountering previously unseen data as it may not be able to encode this input. For instance, if a model was trained with decent weather data and then faced with bad weather.

1.1. Thesis Goal

The purpose of the present degree project is to research state-of-the-art object detection models in deep convolutional neural networks, examine prominent deep object detection frameworks, and evaluate their performance for a certain dataset. One-stage and two-stage detection frameworks can be used to classify state-of-the-art techniques to object identification that have produced promising results. R-CNN [2], Fast R-CNN [3], and Faster R-CNN [4] are well established examples of two-stage detection frameworks, whereas YOLO [5], [6], [7], and SSD [8] are instances of one-stage detection frameworks. After research and a literature review, three object detection models are chosen to be trained on the same dataset, fine tuned to get the best performance, and then the results are gathered. These results are then discussed, evaluated, and finally compared with each other in different regards.

1.2. Scope and Delimitation

This thesis is completed under certain time limitations and in a setting that comes with particular restrictions due to a limited amount of available resources. As the approaches that are explored in this thesis are of the deep learning nature and due to the number of object detection algorithms that are assessed, the amount of time required for executing all the training and testing is relatively long. Consequently, it is not plausible to do the training and testing with several datasets in order to determine statistical significance. In addition, we cannot test the null hypothesis statistically to support our findings without repeating experiments. Because of this limitation, it is stressed that the obtained results and evaluations from the thesis are only objective or observational as opposed to empirical, since they have been derived from a single training and inference run.
As a consequence of time limitations, for achieving the goal of the current thesis, we consider only a single object detection dataset, namely the Vehicles-OpenImages Dataset [9]. This dataset was selected as it is of moderate size in comparison to larger possibilities like Berkeley Deep Drive [10], thus allowing the experiments to be conducted in a reasonable amount of time. It is emphasized that for generalizations to be drawn, more study on additional datasets is required, which is outside the scope of this thesis.

1.3. Outline

The following is a brief general overview of components of the thesis report. The background component is covered in Chapter 2, beginning with a discussion of fundamental computer vision ideas in Chapter 2.1, followed by assessments of several object detection

models, including some understanding of performance measures, in Chapter 2.2. It can be pointed out that the experienced reader can skip Chapter 2's background information. The Chapter 3 addresses the methodology, detailing the architecture and mechanism of the chosen models' approach, the dataset, and their performance evaluation. The findings and evaluations of the chosen models are presented in Chapter 4. Chapter 5 concludes with the key findings of the thesis.

2. Theory and Literature Review

For many years, people working in the discipline of computer vision have focused on the problem of object detection using artificial intelligence. As lately, there have been quite a good amount of successes for advancing in this area with the use of deep neural networks. For example, the use of machine learning algorithms for classification of images. A number of such concepts have proven to be highly effective for the object detection challenge. In the following thesis, the main focus is on deep learning techniques for object detection models and latest frameworks for implementing them. Following a brief introduction to the fundamental ideas of computer vision and deep learning, this chapter examines the most common deep learning methods used to solve the object detection challenge. In addition, we examine the most important performance metrics used to evaluate object detection models, the frameworks used to implement the latest object detection models and end it with a discussion about the researched topics.

2.1. Computer Vision

It is a sub division of computer science aiming to replicate the intricacies of the human eye, or to be more precise vision, in order to enable computers to duplicate the way humans recognize different objects in a image or video and process them. The field wants to find strategies to obtain, process, analyze, and comprehend real-world data and images. The goal of this field is to process this data in order to generate numerical or symbolic information in the form of decisions. This data can appear in various forms, including views from numerous cameras or image sequences. Looking at it from a technological standpoint, the aim of computer vision is to apply its notions and algorithms to the development of computer vision systems. It has numerous sub-fields such as object detection, event detection, motion estimation, scene understanding, video tracking object pose estimation and learning [11]. While until not long ago, computer vision has been working in limited capacity, progress in deep learning, neural networks, and artificial intelligence has considerably increased its capabilities to the extent that it has exceeded the performance of humans in a few detecting and labeling tasks. It must be mentioned at this point that a crucial factor in this progress is the considerable quantity of data that is generated nowadays to train and improve computer vision.
In computer vision, a computer is trained to understand visual data (as input) on the principle of pattern recognition. The computer receives input from a huge amount of dataset of labeled images, which are then subjected to numerous algorithms. This enables the computer to recognize patterns in all the images that correspond to the provided class labels.
Another big step in computer vision was the introduction of machine learning, as it allowed developers to program "features" instead of manually coding each principle required for

object detection into their models. "Features" are applications of small scale that are able to detect specific patterns in the provided dataset of images. Next, statistical learning algorithms like linear regression, logistic regression, decision trees, or support vector machines (SVM) were utilized to recognize patterns, label images, and then to perform object detection within them.
Ultimately, a very different approach to machine learning or an efficient method to perform such tasks was introduced, and that is deep learning. It leans on a general-purpose function called a neural network, able to solve any problem that can be represented by an example. A neural network is able to take out common patterns of examples and translate them into a mathematical equation when presented with a number of labeled examples of a particular type of data. This equation then assists in classifying future information bits. Comparing deep learning to preceding kinds of machine learning, its advantage is that it is easier as well as quicker to develop and deploy.

2.1.1. Artificial Neural Network

Artificial neural networks (ANN) are a sub-division of the branch of machine learning. An ANN is a paradigm for information processing whose fundamental structure and nomenclature are derived from the human brain. It imitates the process of biological neurons signaling to each other, thereby how the human brain processes information. Compared to computers, the brain has extremely advanced capabilities in analyzing unclear information and recognizing a variety of objects in various, even new or changing environments. While humans can almost effortlessly recognize patterns, learn from, and make sense of a large amount of visual input from their environment, automating these tasks in computing systems is a challenging task. That is why it makes sense for those systems to draw inspiration from and try to imitate the way the human brain performs those tasks with so little difficulty. This requires the analysis of neural networks, which belong to the nervous system and consist of interconnected neurons communicating with each other.
A traditional ANN architecture is made up of three layers: an input layer, a hidden layer (or layers), and an output layer. The first layer, that is, the input layer gets the input values. There can be one or multiple concealed layers, which are between the initial layer (input) and the final layer (output) and comprise a set of neurons. Although the output layer consists of one neuron, there can be multiple outputs. An output can range between 0 and 1.
Every artificial neuron connects to another, and their processing ability is stored in so-called weights, which are inter-unit connection strengths. Every artificial neuron has an associated weight and threshold. An artificial neuron is only activated and passes data along to the next layer if the output of an individual neuron surpasses the designated threshold value. The strength of an input varies depending on the weight value. That value can be negative, positive, or zero. While a negative weight signifies that the signal is

inhibited or reduced, a zero weight is an indication that there is no connection between two neurons. In order to get the required output, these weights are adjusted with certain algorithms. The procedure of adjusting weights is referred to as training or learning.

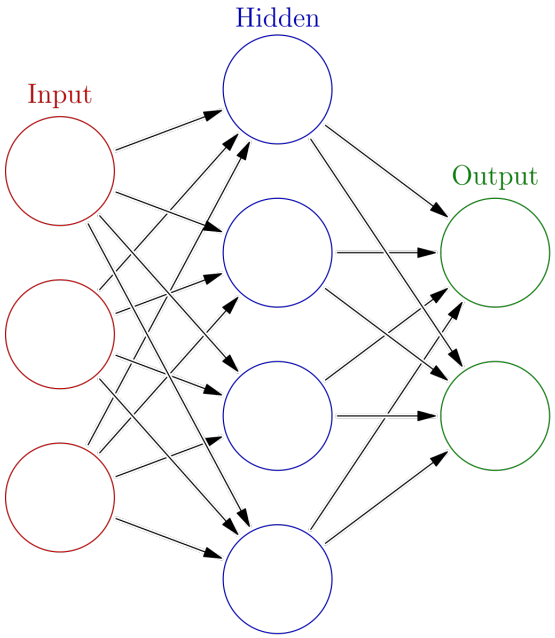

Figure 2.1. Artificial Neural Network Structure [12]

In order to comprehend how neural networks function, they may be represented as the linear regression model of each artificial neuron, which includes the weights, data as input, a threshold value, and finally the output. To put this into a mathematical formula, it would be as follows:

$$\sum_{i=1}^{m} w_i x_i + \text{ bias } = w_1 x_1 + w_2 x_2 + w_3 x_3 + \text{ bias} \tag{1}$$

$$\text{Output} = f(x) = \begin{cases} 1 \text{ if } \sum w_1 x_1 + b \geq 0 \\ 0 \text{ if } \sum w_1 x_1 + b < 0 \end{cases} \tag{2}$$

After determining a layer as input, the weights are allocated. The weights will assist in establishing the importance of a variable, as larger ones have a more important contribution to the output than other inputs. Next, each input is multiplied by its weight, and those results are then summed. Then, the results are processed with the aid of an activation function. Finally, the output is determined by the activation function, which if it surpasses a designated threshold, "fires" the artificial neuron. Only if that happens, is data sent

to the following layer in the model. As a result, obtained output from a artificial neuron becomes the input of the following one. The described procedure of sending data from one layer to the following one, flowing only from input to output, makes this neural network a feed-forward network.
Neural networks serve an important function in practical uses such as image recognition and/or classification. For that, the algorithm needs to be trained by efficiently utilizing supervised learning models or annotated datasets. During the process of training the model, there is a need to assess its accuracy. That's where the cost function comes to play. It is sometimes also known as the mean squared error (MSE). The following is the equation of a cost function:

$$\text{Cost Function} = MSE = \frac{1}{2m}\sum_{i=1}^{m}(\hat{y} - y)^2 \tag{3}$$

where, i = index of the sample

$\hat{y}$ = predicted outcome

y = actual value

m = number of samples

The end goal is to have the cost function minimized in order to guarantee the correctness of fit for any observation. While adjusting its weights and threshold, the model utilizes the calculated cost function and reinforcement type learning to approach the point of convergence, also known as the local minimum. The procedure of adjusting the weights is done by gradient descent. That enables the algorithm to establish the optimal route for minimizing the cost function, thereby reducing errors. Every training sample drives the parameters of the algorithm to progressively converge on the minimum value.
As mentioned, feed-forward networks move only in one direction (input to output) and the majority of neural networks work that way. Another possible way a model can be trained is through back-propagation, where it moves from output to input. A benefit that comes with back-propagation is the ability to calculate and attribute the error rate associated with each neuron. Consequently, the parameters of the model can be adjusted appropriately.

2.1.2. Deep Neural Network

At its most basic, a deep neural network (DNN) is a artificial neural network with an amount of degree of complexity, often no less than two layers. DNN improves the performance of a model's accuracy. It enables a model to generate an output value on the basis of supply of a bunch of inputs. A Deep Neural Network enables a model to develop

its own generalizations, which are subsequently stored in the black box, which is a hidden layer. The black box is difficult to investigate. Even if the values in the black box are known, they don't exist within a framework for understanding [13].

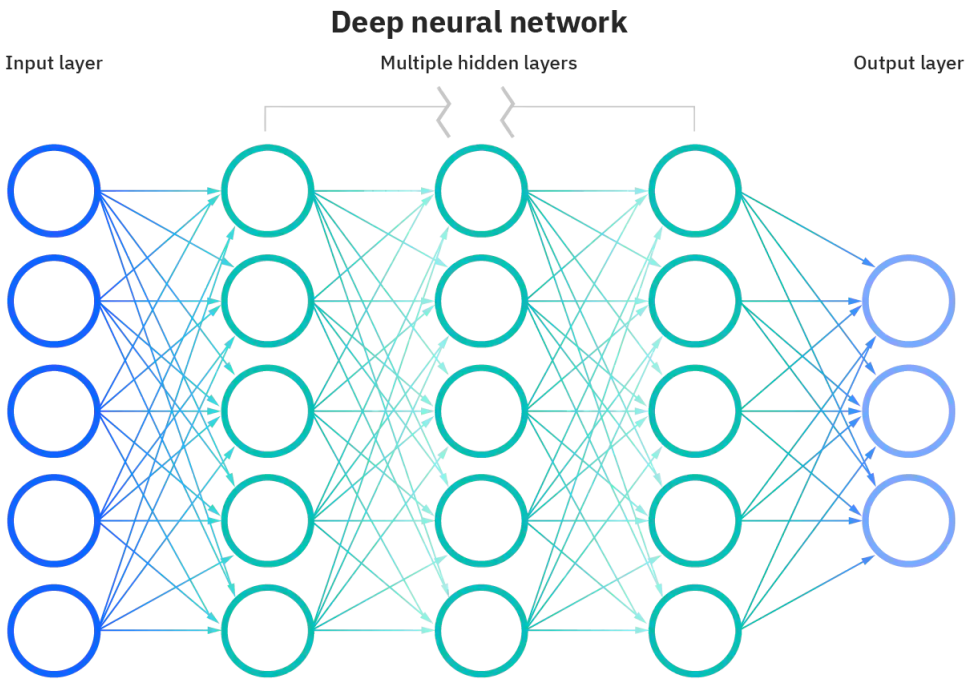

Figure 2.2. Deep Neural Network Structure [14]

Deep learning consists essentially of an exceptionally big neural network, aptly termed "deep neural network." It is referred to as "deep learning" because they include numerous hidden layers and are substantially bigger than regular neural networks, allowing them to store and process more data. Deep learning and deep neural networks are subcategories of machine learning which rely on artificial neural networks, as opposed to machine learning, which relies simply on algorithms.

The development of artificial neural networks was motivated by the learning phase of model creation. ANNs employ the hidden layer to store and assess the relative importance of each input to the output. The hidden layer saves information on the input's significance and creates links between the significance of input pairs.

Therefore, deep neural networks exploit the ANN component. The logic of ANN is particularly effective at enhancing a model because every node in the hidden layer establishes both associations and weighs the input's significance in determining the output. The core idea is to develop subsequently more of these on top of one another and get even greater benefits from the hidden layer.

Summing up, the deep net has numerous hidden layers. "Deep" refers to the presence

of numerous layers inside a model. Deep neural networks enable the performance of a model to improve as its accuracy increases. They enable a model to receive a collection of inputs and generate an output. In layman's terms, copying and pasting a single line of code for each layer is all that is required to implement a deep net.

2.1.3. Convolutional Neural Network

It is possible to increase the complexity of a neural network and model more complicated functions by increasing the number of layers in the network so that it gets deeper. However, with that comes a significant issue, which is that the number of weights and thresholds also increases exponentially. Consequently, the neural network may not be able to learn a problem with such strong difficulty anymore. This is where convolutional neural networks come in.
Convolutional neural networks (CNNs) are a class of deep neural networks that can detect and classify specific features in images and are commonly utilized to analyze visual images. Similarly to traditional artificial neural networks, CNNs consist of neurons that self-optimize through learning. The foundation is the same as in ANNs in that a neuron receives a certain input and performs an operation. In addition, the entire network will represent an uniform perceptual scoring function, or weight.
Unlike classic ANNs, CNNs are primarily employed for pattern identification inside pictures. This permits the encoding of image-specific characteristics into the architecture, making the network better suited for image-centric tasks and reducing the number of parameters necessary to build up the model. In terms of CNN architecture, there are typically three types of layers that, when layered, comprise a CNN architecture. These layers are convolutional, pooling, and fully connected. Following is a full description of the before mentioned layers.

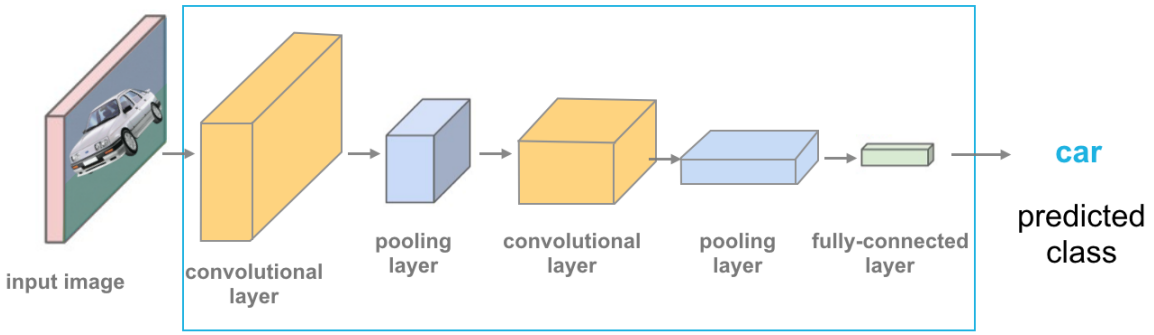

Figure 2.3. Convolutional Neural Network Structure [15]

Convolutional Layer. This initial layer extracts several characteristics from the provided input data. This layer focuses on the usage of learnable kernels, which are distributed throughout the net input depth. After receiving data, a convolutional layer generates a

2D activation map by convolving each filter over the spatial dimension of the input. For each value in a kernel, the scalar product is calculated so that kernels can learn to "fire" in response to seeing a certain characteristic at a given spatial point in the input. The term for this is "activations."
Consequently, each kernel has a corresponding activation map, which is layered along the depth dimension to generate the whole output volume from the convolutional layer. Each neuron in a convolutional layer is only connected to a small fraction of the input volume. This region's size is referred to as a neuron's receptive field size.
A further advantage of a convolutional layer is that it significantly improves its output via the depth, stride, and zero-padding hyper-parameters. Utilizing this, the spatial dimension of the output volumes may be modified.

Pooling Layer. The pooling layer commonly acts as a bridge between the convolutional layer and the fully connected layer. It aims to decrease the dimensionality of the convolved feature map in order to decrease the number of parameters and computational costs (i.e., the complexity of the model). The input of the pooling layer consists of the output of the convolutional layer, as pooling layers are commonly introduced between convolutional layers. Every pooling layer is formed by one or multiple two-dimensional filters, with the size of the height multiplied by the filter's breadth. They work on depth-wise slices of the input by multiplying the receptive field of the input feature map with the adaptive weights and adding the adaptive biases followed by the pooling operation [16]. There can be various kinds of pooling operations dependent on the method used. In the most frequently used one, Max-Pooling, the highest value is taken from the feature map to determine the output.

Fully Connected Layer. It connects the neurons between the two layers. It consists of neurons that are directly connected to neurons in the two adjoining layers but have no connection to any other layers within them. Consequently, the result obtained from each unit of every layer transforms into the input for every unit in the following layer. In particular, the input images from the preceding layers are flattened and sent to the FC layer. Next, the flattened vector goes through some more FC layers where the mathematical function operations usually are performed. At that point, the classification process is starting to operate. By linearly combining the weight and the input, the activations are computed.

2.2. Object Detection

Object recognition is an umbrella term for a set of interrelated computer vision tasks involving the identification of objects in digital pictures. Classification of images includes predicting the category of an object inside a picture. Object localization is the process of identifying the position of one or multiple items in an image and constructing a bounding

box around their perimeter. Object detection combines both of these operations and localizes and categorizes one or multiple items inside an image [17].
Thus, we can differentiate the following three computer vision tasks:

- Image Classification: Identify the category or class of an object in the given input.
  - Input: An image of a dog
  - Output: A class label or annotation of the image
- Object Localization: Identify the existence of an object in the image and predict its location with a bounding box
  - Input: An image of a garden with dogs, cats and bird
  - Output: Bounding boxes for all the objects present in the image
- Object Detection: Identify the existence of objects in the image and predict their location with a bounding box and label them with respective labels
  - Input: An image of a park with dogs, cats and bird
  - Output: Bounding boxes for all the objects present in the image with their respective labels

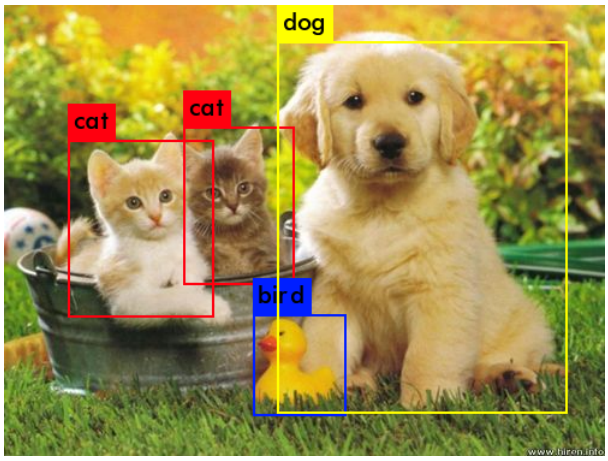

Figure 2.4. Object Detection in action for a sample image

There were various methods explored and implemented for achieving excellence in object detection, but there is one that revolutionised the field with the remarkable triumph of deep learning on the image classification task, that is, AlexNet [18] in 2012. This switched the study field's emphasis from object detection to utilizing concepts and techniques that were already implemented for deep learning object classifiers. Thus, resulting in the creation of two primary types of object detectors using deep learning: two-stage object detection, which contains a phase for region proposal, and one-stage object detection , which does not need region proposal.
This section begins with a discussion of the main performance measures used to evaluate object detection models and is followed by a discussion of the most prevalent frameworks for object detection models.

2.2.1. Performance Metrics

Metrics for evaluating an object detection model's performance are referred to as object detection performance metrics. It also permits the objective comparison of numerous detection systems or compares them to a standard. In this section, the primary object detection metrics and the meaning behind their abstract concepts and percentages are explained.

Precision and Recall. Precision is a model's ability to detect only relevant objects. Also known as the positive predictive value, this is the likelihood that the predicted bounding boxes will correspond with the actual ground truth boxes. Accuracy ratings vary from 0 to 1, and a high precision value indicates that the majority of identified items correspond to the ground truth. For instance, if precision is 0.8, then when an object is detected, the detector is accurate 80 percent of the time.

$$\text{Precision} = \frac{\text{TP}}{\text{TP} + \text{FP}} = \frac{\text{TP}}{\text{all detections}} \tag{4}$$

where, TP = True Positive and FP = False Positive

Recall is a model's capacity to identify all relevant cases (all ground truth bounding boxes), also known as sensitivity. Recall scores vary from 0 to 1, with a high recall score indicating that the majority of ground truth objects were discovered. For instance, recall = 0.6 indicates that the model accurately recognizes 60 percent of the items.

$$\text{Recall} = \frac{\text{TP}}{\text{TP} + \text{FN}} = \frac{\text{TP}}{\text{all ground truths}} \tag{5}$$

where, FN = False Negative

Intersection over Union (IoU). Intersection over Union, commonly known as the Jaccard Index, is an assessment metric that measures the quality of the predicted bounding box by quantifying the similarity between the ground truth bounding box (targets labeled with bounding boxes in the test dataset) and the predicted bounding box. The range of IoU scores is from 0 to 1.
IoU is computed by dividing the area of overlap between the anticipated and actual bounding boxes by the area of union between them. Below is an illustration of the IOU between a ground truth bounding box (green) and a detected bounding box (red).

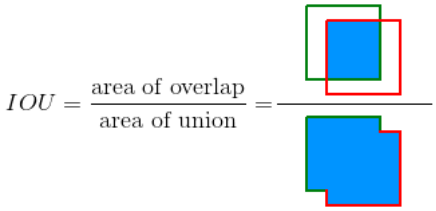

Figure 2.5. Intersection over Union [19]

Some fundamental principles utilized by metrics:

- TP (True Positive): A correct detection, i.e., with the value of IoU greater than or equal to the threshold value
- FP (False Positive): A wrong detection, i.e., with the value of IoU less than the threshold value
- FN (False Negative): A ground truth that is not detected.

Threshold is the value set depending on the metric. It is customarily set to 50%, 75% or 95%.

Mean Average Precision (mAP). Average precision (AP) is a single-number statistic that encompasses both accuracy and recall and is utilized to assess the performance of object detectors. In practice, AP represents the average accuracy of all recall values between 0 and 1. The average accuracy evaluation metric relates mostly to the PASCAL Visual Object Classes (VOC) competitive dataset [20]. For a given IoU threshold, the predictions may be sorted according to the objectness score, followed by the calculation of accuracy and recall for various objectness scores in an interval of N values that encompass the entire range. This yields N values for accuracy and recall, allowing the precision to be shown as a function of recall. AP equals to the average value of precision for each recall value and may be expressed using the formula below [16].

$$\text{AP} = \frac{1}{N} \sum_{r} p(r) \tag{6}$$

According to the PASCAL VOC Challenge publication [20], both classification and detection were evaluated using interpolated average precision. Interpolated AP is the AP that is defined across a collection of recall values from 0.0 to 1.0 that are uniformly spaced. We are aware that the accuracy and recall numbers fall between 0.0 and 1.0. Therefore, we must compute AP for eleven recall values equally spaced out.
AP is determined separately for each class, and when these values are averaged, the statistic is referred to as mean average precision (mAP). If the dataset has N class categories, the mAP is computed by averaging the AP over all N classes. Furthermore, PASCAL VOC challenge [20] use mAP as a metric with an IoU threshold of 0.5, whereas MS COCO [21] averages mAP over several IoU thresholds (0.5, 0.55, 0.6, 0.65, 0.7, 0.75, 0.8, 0.85, 0.9,

0.95) with a step of 0.05; this measure is designated in studies as mAP@[.5,.95]. Hence, in addition to calculating the average AP across all classes, COCO also does so on the IoU criteria.

2.2.2. Frameworks

A framework is a structure that acts as a foundation, allowing one to avoid beginning from scratch. Typically, frameworks are connected with a particular programming language and are suited to certain types of tasks. The literal definition of framework is "fundamental structure underlying a system". Whether the system being constructed is a home, an automobile, a theory, or a mobile application, the principle of the framework is the same: it provides support and a fundamental "guide" for the construction.
Various frameworks are being created to facilitate and facilitate the process of deep learning. Researchers and data scientists utilize a deep learning framework to create and train deep learning models. The purpose of these frameworks is to enable anyone to train their models without requiring a comprehensive understanding of the methods behind deep learning, neural networks, and machine learning. Through a high level programming interface, these frameworks provide components for constructing, training, and verifying models [22].
Each framework has its own pros and cons. Let's examine some of the most popular and effective deep learning frameworks in further detail.

OpenCV. OpenCV (Open Source Computer Vision Library) [23] is an open source software library for computer vision and machine learning. OpenCV was developed to provide a standardized infrastructure for computer vision applications and to accelerate the implementation of machine perception into commercial products. In the library one can find more than 2500 optimized algorithms, including a comprehensive collection of both traditional and cutting-edge computer vision and machine learning methods. These algorithms can be utilized for a variety of tasks, including detection and recognition of faces, labeling actions performed by humans in videos, camera movement tracking, extracting 3D models of objects, producing 3D point clouds from stereo cameras, stitching images in order to produce a high-resolution image of an entire scene, locating similar images in an image database, removing red eyes from flash-captured images, following eye movements, recognizing scenery, and establishing markers to overlay [23].

TensorFlow. TensorFlow [24] is an open-source library of software for numerical calculation utilizing data flow graphs. The edges of the graph represent the multidimensional data arrays (tensors) that flow between the nodes. This adaptable architecture allows the deployment of computing to one or multiple CPUs or GPUs on a server, desktop or mobile device without requiring a modification of the code. TensorFlow offers TensorBoard, a selection of visualization tools, for displaying TensorFlow results. TensorFlow is a Python

library, while efforts are underway to convert it to other popular programming languages like Java, JavaScript, C++, and others. However, creating a deep learning model with TensorFlow requires a significant investment of time and resources due to its reliance on a large amount of code to describe the network's topology. TensorFlow operates with a static computation graph, meaning that the algorithm must be executed each time to detect changes. However, the platform is very flexible and robust, which contributes to its widespread usage.

Detectron2. The Detectron2 [25] framework built by Facebook's Artificial Intelligence Research (FAIR) team is a library of the next generation that covers the majority of cutting-edge detection techniques, object identification methods, and segmentation algorithms. The Detectron2 library is a PyTorch-based framework for object detection. The library is very versatile and expandable, offering users several high-quality implementation methodologies and algorithms. Additionally, it supports several Facebook apps and production projects. Facebook's Detectron2 library, built on PyTorch, has several uses and can be trained on a single or multiple GPUs for rapid and accurate results. Using this library, you may construct many high-quality object detection algorithms for optimal results. The Detectron2 package also facilitates the training of bespoke models and data sets. The installation technique is fairly straightforward. The only required dependencies for the Detectron2 are PyTorch and the COCO API. After meeting the prerequisites, one may install the Detectron2 model and train a myriad of models with relative simplicity.

Keras. Keras [26] is a deep learning framework which is developed on top of TensorFlow, Theano, and the Microsoft Cognitive Toolkit (CNTK). It is one of the most prominent high-level APIs for neural networks. It is developed in Python and supports several neural network computation engines on the back end. Even though it lacks the programmability of PyTorch and TensorFlow, it is the best starting place for beginners to understand neural networks. Keras enables users to construct big, sophisticated models using simple instructions. This implies that it is less customizable than its rivals, but it makes producing prototypes and proofs of concept much simpler. It is also available as an application programming interface (API), making it usable in any circumstance. Low-level computations like tensor products and convolutions are performed by a back-end engine and not by Keras itself. Although Keras supports many back-end engines, TensorFlow is its default and major back-end, and Google is its principal backer. The Keras API is included in TensorFlow as tf.keras, which will be the primary TensorFlow API as of TensorFlow 2.0.

2.2.3. R-CNN Models

R-CNN, which is short for Region-based Convolutional Neural Network, model family uses the two-stage object detection model. These 2 stage object detection models, otherwise called region-based frameworks, suggest a collection of regions of interest which may

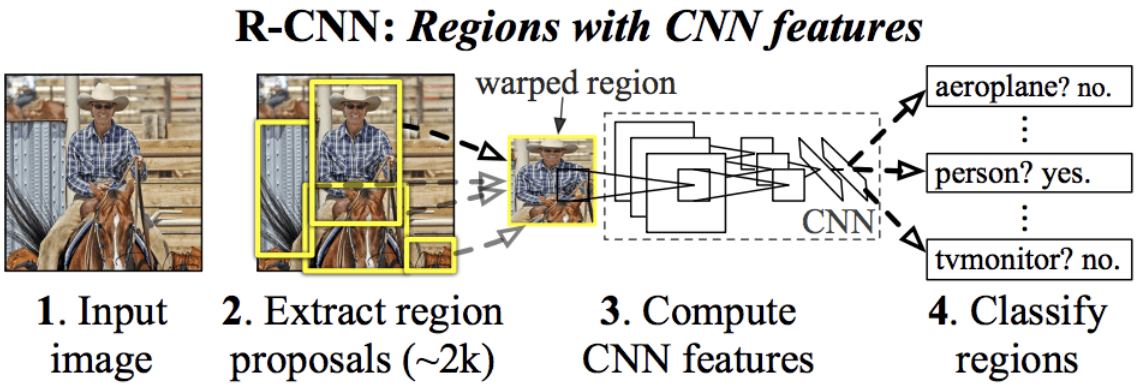

Figure 2.6. Summary of the R-CNN Model Architecture [2]

include an object using an algorithm. Then, a feature extractor is applied to each region, and the output is delivered to a classifier.
These models comprise the object localization and object recognition techniques R-CNN, Fast R-CNN, and Faster-RCNN that were devised and demonstrated. Let's examine the key features of each of these strategies individually.

R-CNN. Girshick, Donahue, Darrell, et al. [2] advocated utilizing deep CNNs for the job of feature extraction, motivated by the triumph of deep learning in the object identification challenge. The recommended technique comprises of three modules and is commonly known as R-CNN, an abbreviation for Region-CNN. The 1^{st} module generates region ideas for each picture and extracts category-independent region proposals. For this assignment, the authors conduct a targeted search which returns 2000 suggested regions. The 2^{nd} module is the convolutional feature extractor, for which the ImageNet Dataset [27] pre-trained AlexNet [18] is utilized. The 3^{rd} module is comprised of a series of linear class-specific SVM classifiers which accept the feature maps as input and output a correctness score for a particular class label, along with a bounding-box regression specific for individual classes.
"Selective search" is a computer vision approach utilized to offer candidate areas or bounding boxes of prospective items in an image. Other region proposal algorithms are possible due to the design's adaptability. The application of CNNs to the challenge of object localisation and recognition is quite basic and straightforward. A disadvantage of the technique is its slowness, since it requires a CNN-based feature extraction step for each prospective region proposed by the region proposal algorithm. This is a concern because the study mentions the model functioning on about 2,000 suggested areas for each image during testing.

Fast R-CNN. Despite the fact that the R-CNN achieves outstanding performance in comparison to object detectors that rely on cherry-picked features or simple neural networks,

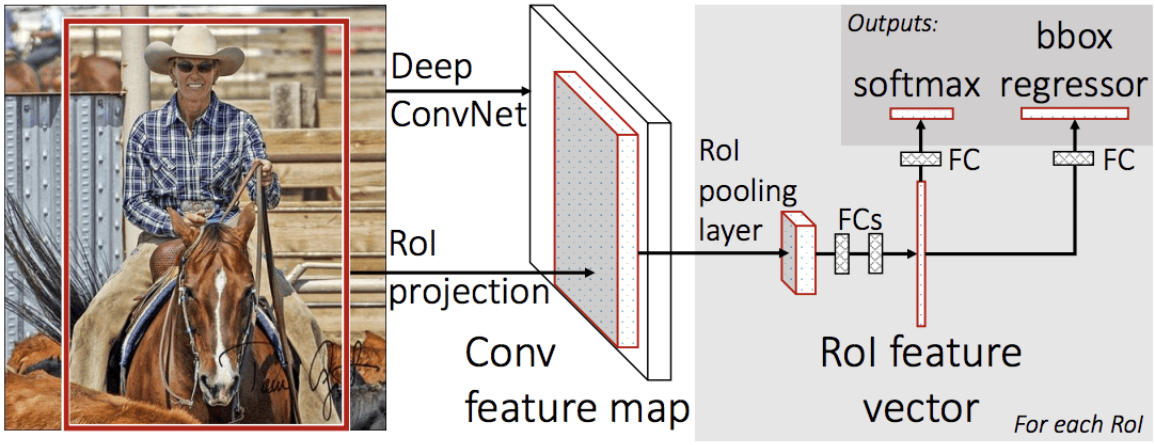

Figure 2.7. Summary of the Fast R-CNN Model Architecture [3]

it has a few flaws. Initially, the process to train comprises of of three steps, which makes optimization challenging. In addition, convolutional deep feature extractors process each and every region of all the input images making it a computationally intensive operation. Girshick presented an upgraded model of R-CNN with the moniker Fast R-CNN [3] in response to these shortcomings. In lieu of a pipeline, Fast R-CNN is offered as a single model that can immediately train and output regions and classifications.
The model's architecture accepts as input a series of region suggestions that are handled by a deep convolutional neural network. For feature extraction, a CNN like VGG-16 that has been pre-trained is utilized. Region of Interest Pooling Layer (RoI Pooling) is the last layer of a deep CNN that recovers properties specific to a particular input candidate region.
The CNN output is then processed by a fully connected layer, at which point the model bifurcates into two outputs: one for class prediction via a softmax layer and one with a linear output for the bounding box. That procedure then is performed several times for each region of interest within a picture. The model is considerably quicker to train and produce predictions, but each input image nevertheless needs a collection of candidate areas to be presented.

Faster-RCNN. Fast R-CNN provides an evident enhancement to the R-CNN architecture; however, the selective search technique employed for region suggestion remains a bottleneck. It is approximated that the majority of time taken for processing in the Fast R-CNN model is devoted to the development of region proposals by the selective search as it takes around two seconds per picture [4].
Consequently, Ren, He, Girshick, et al. [4] enhanced the model architecture in terms of both training and detection time. The improvements, the authors introduced in the paper, consists of two main parts. A fully convolutional network that generates suggestions with different sizes and aspect ratios is the region proposal network (RPN), which is the first phase. In order to instruct the object detection (Fast R-CNN) where to look, the RPN

utilizes neural network terminology. Second, this study proposed the idea of anchor boxes rather than pyramids of pictures (i.e., numerous instances of the image but at various scales). A reference box with a certain scale and aspect ratio is called an anchor box. If there are numerous reference anchor boxes, then the same region might have different sizes and aspect ratios. This can be compared to a pyramid of anchors.
The RPN and the Fast R-CNN share the convolutional calculations, which speeds up processing. Figure 2.8 depicts the Faster R-CNN architecture. There are two modules in it. The first, RPN, is used to generate suggestions for regions, while the second, Fast R-CNN, is used to find objects in the regions RPN has suggested.

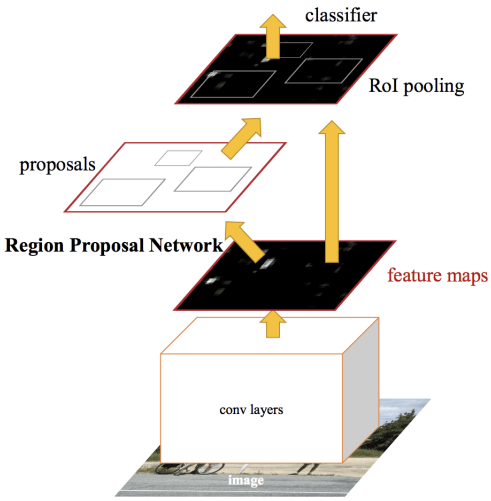

Figure 2.8. Summary of the Faster R-CNN Model Architecture [4]

2.2.4. YOLO Models

In 2015, researcher Joseph Redmon and colleagues [5] created the YOLO (You Only Look Once) family of object detection models. The YOLO method is a first-of-its-kind object detection system that utilizes a single neural network to conduct all the necessary phases for detecting an object. It recasts object detection as a single regression issue, converting picture pixels directly into bounding box coordinates and class probabilities. This unified model predicts numerous bounding boxes and class probabilities simultaneously for objects covered by bounding boxes. When it was published, the YOLO algorithm's criteria for speed and accuracy in detecting and calculating object coordinates exceeded those of the leading algorithms. The YOLO algorithm was improved to five versions during the subsequent years, which are explained below.

YOLOv1 The YOLO model was at first presented in the 2015 publication "You Only Look Once: Unified, Real-Time Object Detection" [5] by Joseph Redmon et al. The method employs a single, end-to-end trained neural network which if it receives an image as input, directly predicts bounding boxes and class labels for each bounding box. Despite operating

at 45 frames per second and up to 155 frames per second for a speed-optimized version of the model, the approach provides less accurate predictions (e.g., more localization mistakes).

The core concept of this model is to impose an S-by-S grid cell on an image. If the center of an object lies within a grid cell, that grid cell is responsible for detecting the object. Consequently, even the emergence of an object shown in many cells is dismissed by the remaining cells. Each grid cell forecasts B bounding boxes, together with their parameters and confidence ratings, in order to execute object detection. These confidence scores reveal if an object is present or absent within a bounding box. The confidence score is calculated as follows:

$$\text{confidence score} = \text{p(Object)} \times \text{IoU} \tag{7}$$

where p(Object) is the probability that the cell contains an object and IoU is the intersection over the union of the prediction box and the ground truth box. p(Object) is between 0 and 1, hence the confidence score approaches 0 if the cell contains no object. If not, the score is equivalent to IoU.

An image can be split into a 7×7 grid, where each cell predicts two bounding boxes, resulting in 94 recommended bounding box predictions. The class probability map and the confidence-adjusted bounding boxes are merged to produce the ultimate set of bounding boxes and class labels. The accompanying picture from the paper [5] sums up the model's two outputs.

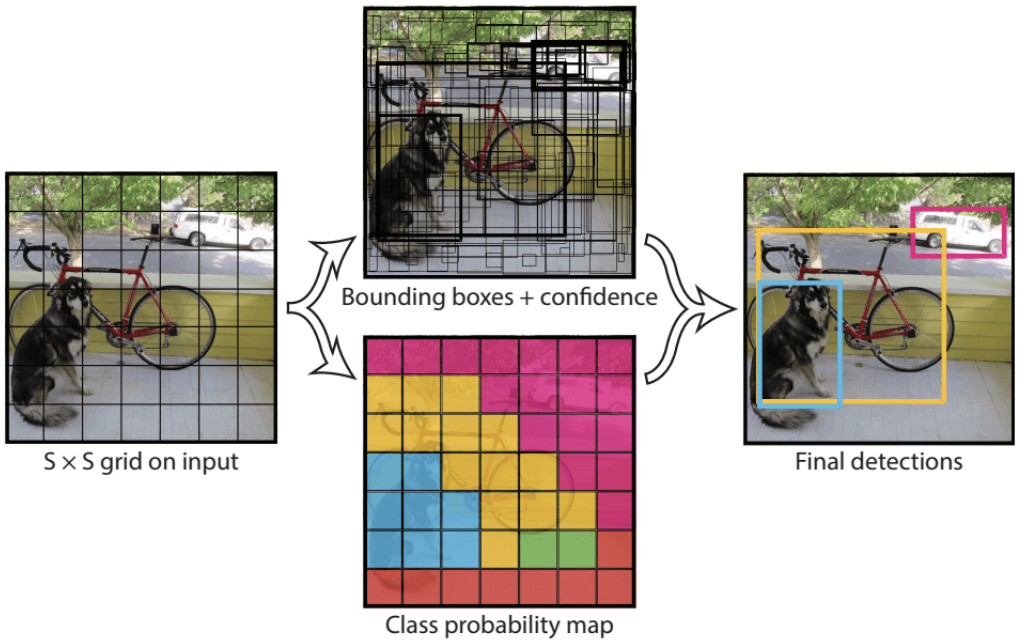

Figure 2.9. Summary of Predictions made by YOLO Model [5]

YOLOv2 In their 2016 work named "YOLO9000: Better, Faster, Stronger" ,[6] Joseph Redmon and Ali Farhadi revised the model in an effort to increase model performance further. Although this variation of the model is referred to as YOLOv2, a sample of the model is reported which was trained in parallel on two object detection datasets and is able to predict 9,000 object classes, thus the term "YOLO9000." Several adjustments were made to the model's training and architecture, including the use of batch normalization and high-resolution input photos.
Initially, the regularization approach is modified to incorporate simply batch normalization, thereby eliminating the requirement for further regularizing. In addition, instead of a preset $S \times S$ grid, the writers suggest utilizing anchor boxes for which the ratio and size are learned from the processed data obtained after applying k-mean clustering method to the bounding boxes depicting ground truth. YOLOv2 employs a modified neural architecture called Darknet-19 which consists of 19 convolutional layers. Furthermore, this architecture accomplishes equivalent accuracy to state-of-the-art classifiers on the ImageNet classification task [27]. In addition to these modifications, YOLOv2 employs a multi-scale training in which the network evolves every 10 iterations by arbitrarily selecting a new picture input size where the lowest choice measures 320×320 and the biggest measures 608×608 [6].

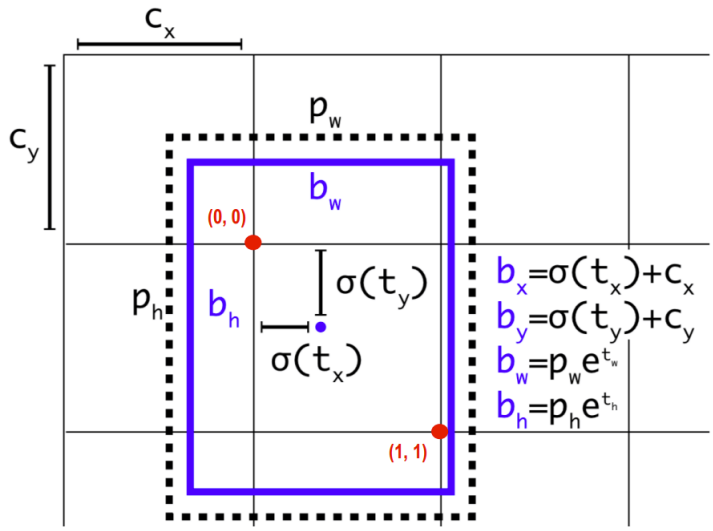

Figure 2.10. Exemplification of the Selected Representation for Predicting Bounding Box Position and Shape [6]

YOLOv3 The authors of YOLO object detection model published a technical paper entitled YOLOv3 [7] that introduces an incremental upgrade to the YOLO framework. A new backbone network called Darknet-53 that uses residual connections, or as the author put it, "those newfangled residual network stuff," has been used as an improvement. YOLOv3 forecasts the amount of bounding boxes as an offset to preset anchor boxes, coupled with objectness scores and classes.

For the objectness scores, logistic regression is utilized, and during training a confidence score of 1 is provided to an anchor box if it has the greatest overlap with a ground truth box with a score greater than 0.5. In addition, single logistic regressions are employed for classification in place of a softmax layer. The primary enhancement is founded on the concept of feature pyramid networks [28], in which predictions are produced at many sizes. Numerous convolutional layers are first added to the underlying feature extractor, followed by the selection of the final feature map and the prediction of a three-dimensional tensor representing the probabilities of class, objectness score and bounding box offsets. After adjoining it with an output layer taken from previous layers of the model, the following step is to up-sample the feature map from two layers earlier so that its size is twice that of the preceding, and then to up-sample it again. Again employing the same concept, YOLOv3 is able to make predictions at three distinct scales.
In addition, three possible anchor ratios are estimated for each scale, with their sizes learned from the dataset using K-means clustering, similar to YOLOv2 [6]. YOLOv3 employs a bigger feature extractor known as Darknet53, which expands Darknet19 via residual connections. According to the given data, YOLOv3 is equal to other single-shot detectors such as SSD [8], but three times quicker. Nonetheless, the writers show that YOLOv3's performance degrades as the IoU threshold for identifying good samples grows.

YOLOv4 The YOLOv4 [29] was published in April 2020, however it was not written by the original YOLO author. YOLOv4 promises to offer advanced accuracy while keeping a high processing frame rate, as seen in the image. As depicted in Figure 2.11, it obtains an accuracy of 43.5 % AP for the MS COCO with an inference speed of roughly 65 FPS on the Tesla V100. In object detection, great precision is no longer the only major requirement.The model needs to operate efficiently on edge devices. Also vital is the processing of visual input in real-time using low-cost hardware. Numerous enhancements have been made to YOLOv4. It is an effective as well as potent object detection model that allows anybody possessing a 1080 Ti or 2080 Ti GPU to train an incredibly quick and accurate object detector. One improvement is the concept of "bag of freebies," i.e., unique training process enhancements that maximize accuracy without affecting inference speed (like data augmentation, soft labeling, cost function, etc.). Improving the network, which has a negligible influence on the inference time but yields a high performance return, is another strategy. These enhancements include expanding the receptive field, using attention, integrating features such as skip-connections, and utilizing post-processing techniques such as non-maximum suppression.
The improved algorithms, such as CBN (cross-iteration batch normalization), PAN (path aggregation network), etc., are currently more efficient and appropriate for training on a single GPU. In addition, Yolov4 employs a genetic approach to pick appropriate hyperparameters during network training during the initial 10% of time periods. Cross mini-batch

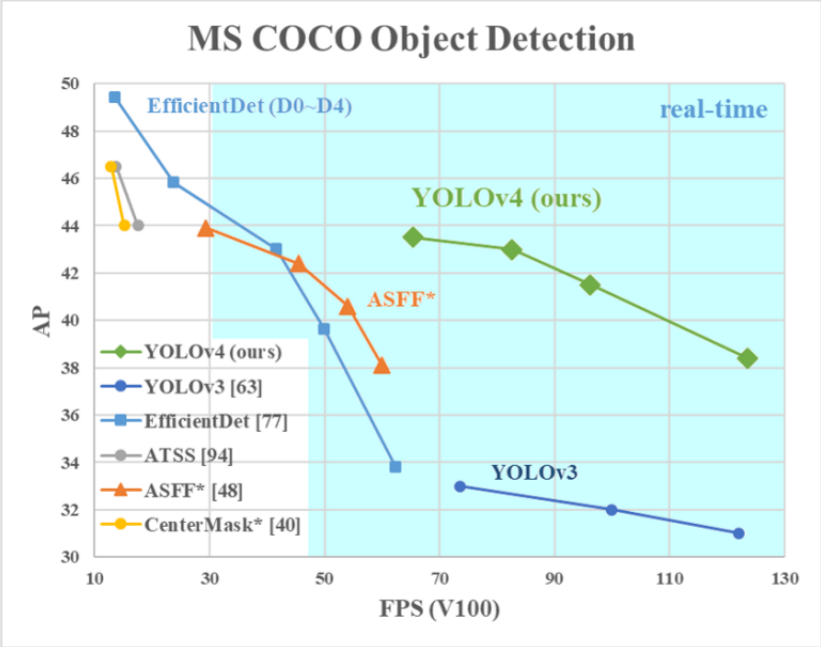

Figure 2.11. Comparison between the YOLOv4 and other object detectors [29]

normalization takes information from the full batch, rather than from a single mini-batch, essentially aggregating statistics over many training rounds.

YOLOv5 Shortly following the publication of YOLOv4, researcher Glenn and his colleagues at Ultralytics LLC published YOLOv5 [30]. Ultralytics is the company that transforms previous versions of YOLO to one of the most popular frameworks in the discipline of deep learning, the Python-based PyTorch.
Although YOLOv5 was launched one month after YOLOv4, YOLOv4 and YOLOv5 research began almost simultaneously (March-April 2020). Glenn chose to label his version of YOLO "YOLOv5" to avoid collision. Thus, both researchers essentially utilized the revolutionary developments in computer vision at the time. Consequently, the architectures of YOLOv4 and YOLOv5 are very much alike, and numerous people feel displeased with the alias "YOLOv5" as it does not present numerous significant enhancements compared to the previous version. In addition, Glenn did not issue any YOLOv5-related publications, which increased doubts about the program.
Nevertheless, YOLOv5 held the engineering benefits. YOLOv5 is written in Python instead of C, as was the case with prior versions. This facilitates the setup and integration of IoT devices. Additionally, the PyTorch community is bigger than the Darknet community, indicating that PyTorch may get more contributions and have greater future development potential. The YOLOv5 model can be summed up in the following way by examining its structural code in the .yaml file. The Backbone, which is, Focus structure and CSP network.The Neck, which is, SPP block and PANet. The Head, which is, YOLOv3 head using GIoU-loss

2.2.5. SSD Models

Initially suggested by Liu, Anguelov, Erhan, et al. [8], the single shot multibox detector (SSD) is a one-stage object detector which merges speed and accuracy equivalent to two-stage frameworks. It blends the concepts of region-based object identification techniques with YOLO's fully CNN and multiscale convolutional features.
SSD consists of a backbone model and an SSD head. Generally, the backbone model is an image classification network that has been pre-trained and is utilized to extract features.. Typically, this is a ResNet-like network trained on ImageNet whose last fully linked classification layer has been deleted. Hence, what remains is a deep neural network which can extract semantic meaning from an input image while maintaining its spatial structure, albeit at a lesser resolution. The backbone of ResNet34 generates 256 7x7 feature maps for an input image. Later on, we shall define a feature and a feature map. The SSD head consists of one or multiple convolutional layers added to this backbone, and the outputs are interpreted as bounding boxes and classifications of objects in the spatial position of the final layer activations. The first few layers (white boxes) in the Figure 2.12 constitute the backbone, whereas the final few levels (blue boxes) represent the SSD head.

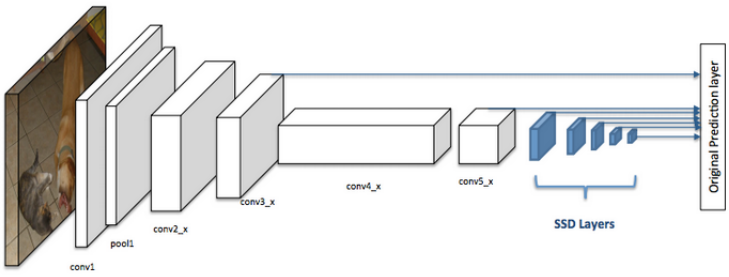

Figure 2.12. Architecture of a convolutional neural network with a SSD detector [8]

SSD utilizes varied sizes, shapes, and aspect ratios for default boxes on different output levels. It employs 8732 boxes to provide more comprehensive coverage of location, size, and aspect ratios. The majority of forecasts will not include any items. SSD eliminates predictions with a confidence level of less than 0.01. Then, 0.45 Non Max Suppression (NMS) overlap is applied per class, and the top 200 detections per picture are retained. While YOLO has a fixed aspect ratio for grid cells. SSD employs numerous aspect ratios and many boxes for more precision. At the conclusion of VGG-16, SSD adds extra convolutional layers for object detection. A convolutional layer has numerous characteristics with varying scales; hence, it can recognize things at multiple scales more effectively.

3. Methods

3.1. Dataset

In this section, we analyze the class distribution and training/validation splits to define the dataset utilized for all experiments in this thesis. The Vehicles-OpenImages Dataset [9] supplied by Jacob Solawetz in Roboflow was used as the data set for the thesis implementation. The primary applications for this dataset are:

- Train an object detector to differentiate among a vehicle, bus, motorbike, ambulance, and truck
- Detector for objects at checkpoints for autonomous vehicle
- Examine the density of ambulances in automobiles with an object detector
- Train ambulance detector
- Explore the variety and precision of the Open Image dataset

The dataset contains a total of 627 photos of cars, with each image measuring 416 × 416 pixels. These photos contain numerous kinds of vehicles for object detection. These pictures are extracted from the Open Images [31] open-source computer vision datasets. This collection represents only a fraction of the Open Images dataset for vehicles.
There are five designated categories, namely, car, bus, motorcycle, truck, and ambulance. The dataset used is not proportionally balanced for all the classes, as is evident from the bar chart in Figure 3.1. We can see that the class containing the label "car" is the one with the largest amount of instances in dataset, with nearly 651 labeled objects, followed by "bus" with 141, "motorcycle" with 140, "truck" with 136, and "ambulance" with 126. All other classes, excluding car, fall within the same range, which is considerably lower than the car class.

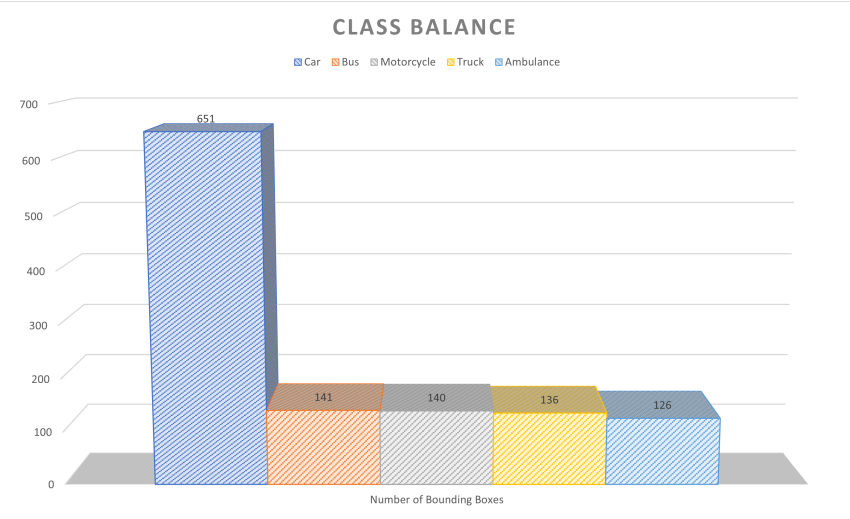

Figure 3.1. Class distribution of the Vehicles-OpenImages dataset

In this dataset, there are no missing annotations, meaning images that do not have an accompanying annotation file, or null examples, meaning images that deliberately do not contain any objects. In total, there are 1194 objects annotated, which roughly translates to 1.9 per image (average). The average image size, i.e., the size of images in megapixels, is 0.72 mp with images ranging from 0.29 mp to 1.05 mp.
For using the dataset in experiments conducted in the scope of this thesis, the dataset was split in the ratio of 7:2:1 for the training dataset, validation dataset, and testing dataset, respectively. Resulting in 439 images being used as a training dataset, 125 images as a validation dataset, and 63 images as a testing dataset.

3.2. Object Detection

In this section, we will go through the chosen object detection models for the thesis. For each of them, we will go through the research strategy used, all the components of the model, the implementation of the model with the respective framework, the use of the dataset with respect to the model and the limitations faced.
All the object detection models are trained and tested using the online platform of Google Colab which provides free GPU support for running such tasks requiring great deal of computer resources.

3.2.1. YOLOv5

The first chosen object detection model for this thesis is YOLOv5. The reason to go with YOLOv5 is fairly simple as it is the latest version of the YOLO object detection models with the advantage of being implemented in Python and having great accessibility for integration and use in most of the fields.
First, the repository of YOLOv5 provided by Ultryalytics on GitHub is cloned. As this is the original source of the YOLOv5 implementation, any experiment based on YOLOv5 starts with the base of the code provided by Ultralytics for YOLOv5. Then, the requirements for running the model are installed. One of the main requirements, being, PyTorch as this is the framework on which YOLOv5 is built. Then, we also install the library for Roboflow as the dataset we are using is provided by Roboflow.
We must put up a dataset of sample images with bounding box annotations surrounding the things we wish to identify in order to train our own model. Additionally, we require a dataset in YOLOv5 format. As we are using Roboflow, we can convert our dataset for the YOLOv5 model using the Roboflow library.
Then comes the main part, where we train our custom YOLOv5 model. To do this step, first we need to define some parameters necessary for the best training. Let's go through each parameter:

- Img: It is used to define the size of the images present in given dataset. In our case, the images are 416 × 416. Hence, the value used for this parameter is 416

- Batch: it is used to determine the batch size in which the input images will be provided to the model. Batch size should be used as large as possible with respect to the hardware resources available. Considering all these factors, the value used for this parameter is 16
- Epochs: It is used to define the number of training epochs. One epoch is when an entire dataset is passed forward and backward through the model exactly one time. So, for our case, as there are 627 images divided into the batches of 16 images, it will take 40 iterations to complete a single epoch. The value used for this parameter is 500
- Data: It is used to provide the location of the dataset. For this parameter, we provide the location of the .yaml file which contains the configuration of the dataset.
- Weights: It is used to specify a path to weights to start transfer learning from. For this parameter, as the dataset we are using is small sized, we use the pretrained weight file i.e., yolov5s.pt
- Cache: It is used for caching the images for faster training process

The training process stopped after 28 minutes and completing 264 epochs. It stopped the training early as no improvement was observed in last 100 epochs. The best results were observed at epoch 193 and saved as the best model.

3.2.2. Faster R-CNN

The second chosen model for the experiments under the scope of the thesis is Faster-RCNN using the framework Detectron2. The reason for using this model is as it is considered one of the best two stage object detection model and the reason for choosing the Detectron2 framework is also similar to this. It is the Facebook AI Research's next-generation software system that implements state-of-the-art object detection algorithms [32].
For implementing the Faster R-CNN model, first we need to set up detectron2 framework. As Detectron2 is based on PyTorch as well, we start by installing the relevant torch libraries. After the installation of the torch libraries, we install the detectron2 framework provided by Facebook Research team on Github [32].
The detectron2 comes with a bundle of helper functions and is fully stacked with utilities. For our task, we use the logger functionality of detectron2 to keep track of all the steps and information being processed and the time taken for doing so. We use the model_zoo to import the pre-trained models that we will use in the experiments, Visualizer library for creating graphs and evaluating the results in the end in the form of charts and DefaultPredictor utility for simple demo purposes as it creates a straightforward predictor with respect to the given configurations that runs for a sole input image.
After setting up all the required libraries, we will start by importing the dataset. In Detectron2, we need to register the dataset that will be used for training and testing the model. As our dataset is taken from Roboflow, we can easily download it in any required format. For this model, we downloaded the dataset in COCO format. Thus, enabling us

to use the already existing function for registering the data and the metadata associated with it. After providing the location to the dataset and .json file, which contains the annotations for the images, to this function, we can use the MetadataCatalog.get() function and DatasetCatalog.get() function to access the metadata and the dataset respectively. A key-value mapping known as metadata comprises information that is shared by the whole dataset and is typically needed to interpret the contents of the dataset, such as class names, class colors, file paths, etc. For augmentation, assessment, visualization, and logging, this information will be helpful.
Before proceeding further, with the training and testing of the model, we will confirm if the imported dataset is correct and has been loaded as required. For this, we import the random library and randomly select some images from the dataset and use the visualizer function to show the images annotated with their respective classes in the output. After confirming, the correctness of the data loaded visually, we proceed to the next step i.e., training the model on this dataset.
From the imported library of detectron2 engine, we use the default trainer to train our model on our loaded custom dataset. First, we obtain the detectron2's default config. Then we load the values in this config using the provided config file for Faster RCNN as per the COCO detection dataset. This is one of the files from the collection of files that documents a large collection of baselines trained with detectron2 in Sep-Oct, 2019 by the developers of Detectron2 [32]. The chosen model for this task is the Faster RCNN R50 FPN which is the Resnet 50 model trained with the 3x schedule (37 COCO epochs) and operating on a ResNet+FPN backbone with standard convolutional layer and fully connected heads for mask and box prediction, respectively [32]. Then the prepared dataset, already split between training and testing, in the previous step is provided as well to the function. Then we set the amount of workers for our data loader, which provides the number of sub processes to be used for data loading to the data loader, as 2. Then the weight file is supplied to the model and used as the initializer for the training. The file for the weights is taken the same as for the model selected.
Then we set some parameters for the training process. First, we set a base learning rate, which is basically the amount of the time that the weights are updated during training and is also known as the step size. For this task, after some trials, we finally set the base learning rate value to be 0.0025. Then, we set the value for the maximum iterations. It is the stopping condition for the training process after the set number of iterations are completed. For this task, we set the maximum number of iterations to 2000, which is already a high value for a small dataset. Then, we set the images per batch value to 2 and the RoI (Regions of Interest) minibatch size per image during training to 128. Thus, the total number of RoIs per training minibatch will be equal to, 128×2, 256. This value is set as it will be faster and good enough for our small dataset. Then we set the value of the RoI number of classes which, in our case, is 5. After setting all the parameters, we start the

training process.
After the training process is completed, we proceed to the testing and evaluation step. For this purpose, we use the detectron2 evaluation library and data library. We use the COCOEvaluator function to evaluate AR (Average Recall) for object proposals, AP (Average Precision) for instance detection/segmentation, AP (Average Precision) for key point detection outputs using COCO's metrics. We provide the test dataset and the config file to this function with location for the output. We also use the inbuilt function to build detection test dataset loader and finally provide all the values to the inference on dataset function to see the final evaluations and results.

3.2.3. SSD

The third and the last chosen model for the experiments performed under the scope the thesis is the SSD (Single-Shot Multi-box Detection) object detection model using TensorFlow 2 framework. The reason for choosing the SSD object detection model is that it detects objects with high precision in a single forward pass computing feature map. It is also a single stage object detector like YOLO but has proved to be more precise than YOLO in earlier studies. The feature that sets it apart from YOLO is its approach to bounding-box regression. The reason for using TensorFlow 2 framework is that it is the state of the art deep learning framework at the time of writing this thesis and has solved many long lasting issues of its predecessor.
For implementing the SSD model, we start by installing the required libraries. The main and the foremost library being the TensorFlow2. We also install the library tf-slim, which is a lightweight library for defining, training and evaluating complex models in TensorFlow [33]. Then we traverse to the directory with the tensorflow2 model and install all the requirements. Next we will, proceed with the importing of dataset and loading it correctly. For importing the dataset, first we need to convert the downloaded config file from xml to csv as TensorFlow requires the annotations file to be in the csv format. This is done by implementing some basic logic in python to convert xml to csv. Then, we need to generate config files for training data and testing data in the format of .record which is required by tensorflow. To do this, we use the generate_tfrecord.py file, which comes as an utility in the tensorflow2 package and provide it the location of the labels file in csv format, that we converted in the previous step and also the location of the images of the training dataset and the same for testing dataset separately. Finally, the function returns the train.record and test.record files which will be used in the further process.
Then, we download the selected SSD model, that is pre trained on COCO Dataset, from the tensorflow 2 models archive. It is also useful for initializing the models when training on novel datasets like the one we are using for this thesis. For this task, we selected SSD ResNet50 V1 FPN 640x640 (RetinaNet50) model. After successful downloading of the model, we extract the contents of it and place it in the appropriate directory. We take the pipeline config file from the downloaded model and move it to our directory as we need to

make changes to the parameters before we start with the training process.
In the pipeline config file, we start by setting the value of the number of classes that are present in our dataset. So, the value for the parameter of num_of_classes is set to 5. Then, we set the path to checkpoint of pre-trained model in the parameter of fine_tune_checkpoint. Then, we set the number of steps to be equal to 5000 as the dataset we are using is small and after a few experiments, it was concluded that there was no increase in the performance beyond 5000 steps. Then, we set the batch size value to be equal to 4. Then, we set the fine_tune_checkpoint_type parameter to 'detection' as we want to be training the full detection model. Finally, we provide the config file with location of the .record file generated for training and testing dataset in the earlier step and also the location of the label mapping file for both the datasets. The rest of the parameter values are left as default.
After all the configurations are in place, we proceed to the main step of training the model. To train the model, we run the file model_main_tf2.py and provide it with the arguments specifying the location of the training dataset and the pipeline.config file where set all the parameters. The training process completed in almost 49 minutes. Then finally, the model underwent testing and the results are stored for further evaluation.

4. Results

In this chapter, we will go through the results obtained from all the experiments performed under the scope of this thesis. As mentioned before in the thesis, all the object detection models are used to train and test the same dataset, which is the Vehicles-OpenImages Dataset [9]. The results are subdivided into the three subsections corresponding to the three object detection models chosen to perform the experiments. First, we will see the results from the individual object detection models, i.e., YOLOv5, Faster RCNN, and SSD, and then we will compare them in the final subsection of the chapter titled "Evaluation."

4.1. YOLOv5 Result

As mentioned in Section 3.2.1, the training process for the YOLOv5 model was completed in 27 minutes and 46 seconds. The model was supplied with the parameter for epochs set to 500, but it stopped the training process after 294 epochs as the model did not observe any improvement in the results in the last 100 epochs.
The brief summary of the YOLOv5 model that we used for this task can be expressed as 213 layers, 7023610 parameters, 0 gradients, and 15.8 GFLOPs. FLOP, or floating point operations per second, is a measure of performance, meaning how fast the computer can perform calculations. GFLOP is simply a Giga FLOP. We can see the metrics calculated for the validation dataset by the YOLOv5 model in the Table 4.1

Class	Images	Labels	Precision	Recall	mAP_0.5	mAP_0.5:0.95
all	125	227	0.685	0.548	0.562	0.413
Ambulance	125	32	0.565	0.812	0.761	0.616
Bus	125	23	0.941	0.652	0.7	0.551
Car	125	119	0.653	0.403	0.474	0.321
Motorcycle	125	23	0.602	0.609	0.563	0.37
Truck	125	30	0.666	0.266	0.313	0.209

Table 4.1. YOLOv5 Evaluation Results

Let us define the columns present in the table and their meaning.

- Class: This column enlists all the classes present in the dataset.
- Images: This column gives the number of images present in the validation dataset for which the respective classes were tested.
- Labels: This column depicts the total value of the respective class labels present in the validation dataset.
- Precision: This column is the measure of the precision, which is the number of correct bounding box predictions.

- Recall: This column is the measure of the recall, which is the number of true bounding boxes that were correctly predicted.
- mAP_0.5: This column is the measure of mean Average Precision (mAP) at an IoU (Intersection over Union) threshold of 0.5.
- mAP_0.5:0.95: This column is the measure of the average mAP over different IoU thresholds, ranging from 0.5 to 0.95.

From Table 4.1, we can see that for all the classes in the validation dataset, the precision is 68.5% and the recall is 54.8%. The value of mAP_0.5 is 56.2% and mAP_0.5:0.95 is 41.3%. The highest value of mAP_0.5 is obtained for the 'Ambulance' class, which is 76.1%, whereas the lowest is for the 'Truck' class, which is, 31.3%. The trend stays the same for the mAP_0.5:0.95 values, but with an overall decrease in the measures.

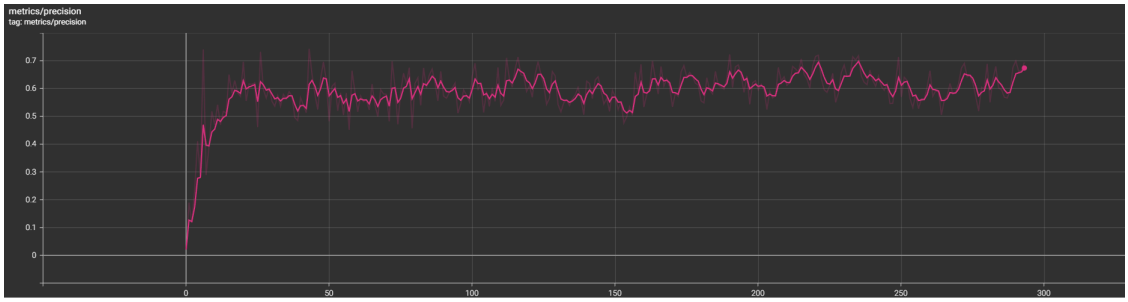

Figure 4.1. Precision vs Epochs

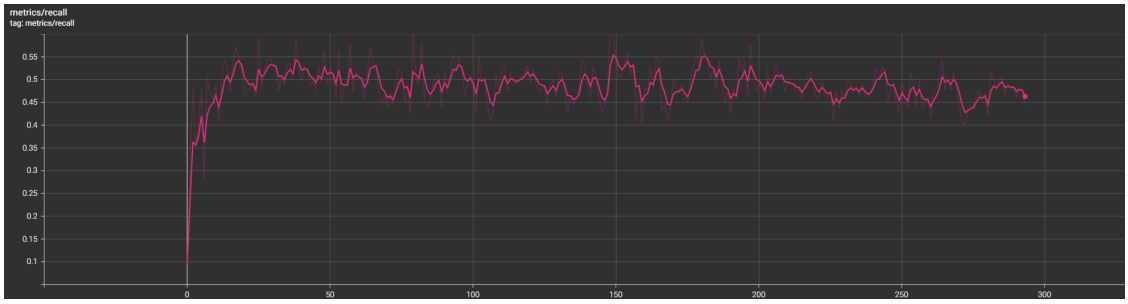

Figure 4.2. Recall vs Epochs

We can see the trend of the above mentioned metrics (precision, recall, mAP_0.5, and mAP_0.5:0.95) in the graphs presented in Figure 4.1, 4.2, 4.3, 4.4. The graphs are generated with a smoothing value of 0.6 for the clarity of the depiction of trends.

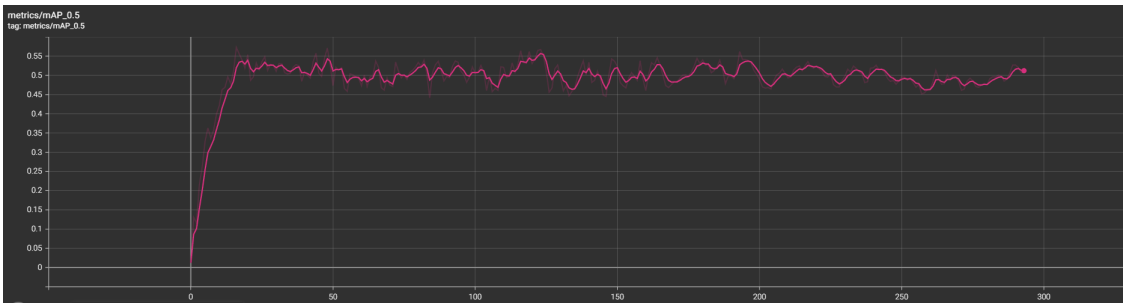

Figure 4.3. mAP_0.5 vs Epochs

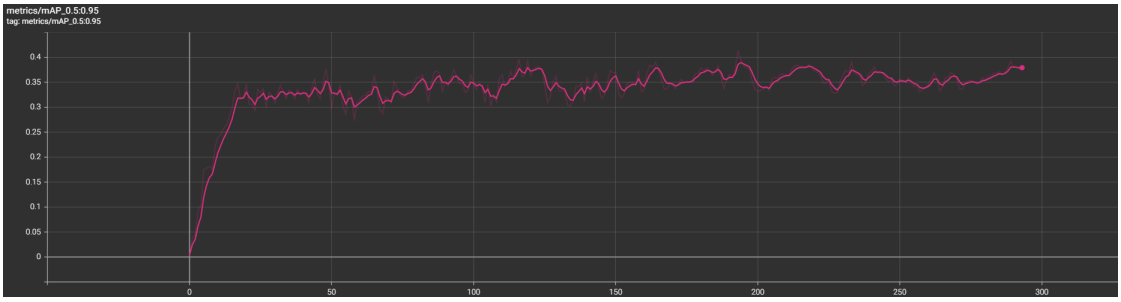

Figure 4.4. map_0.5-0.95 vs Epochs

For better understanding of the results, we should also look at the losses in the YOLOv5 model. Yolo's loss function is composed of three parts. One, box_loss which is the bounding box regression loss (also called Mean Squared Error). Second, object loss which is the confidence score of presence of object. It is the objectness loss (also called Binary Cross Entropy). Third, class loss which is the classification loss. As we can see from the Figures 4.7, 4.5, 4.6 the box_loss is reduced to 0.02 by the last step of training, in contrast to it being started from a value of 0.11 in the first step. The same trend is observed for obj_loss and cls_loss. The cls_loss value started at 0.05 in the first step and was reduced to 1.31e-3 in the last step, and the obj_loss value started at 0.04, and by the last step it was reduced to 0.02.

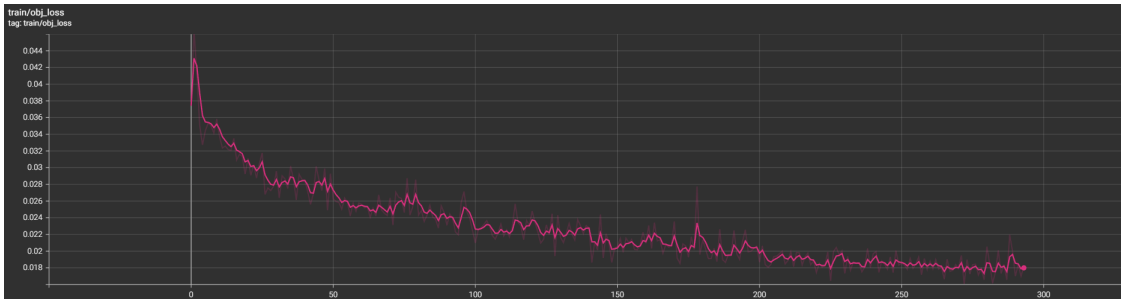

Figure 4.5. obj_loss value vs Epochs

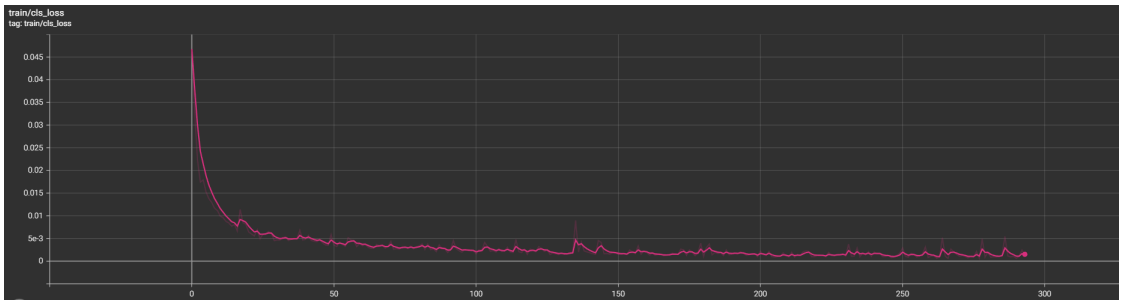

Figure 4.6. cls_loss value vs Epochs

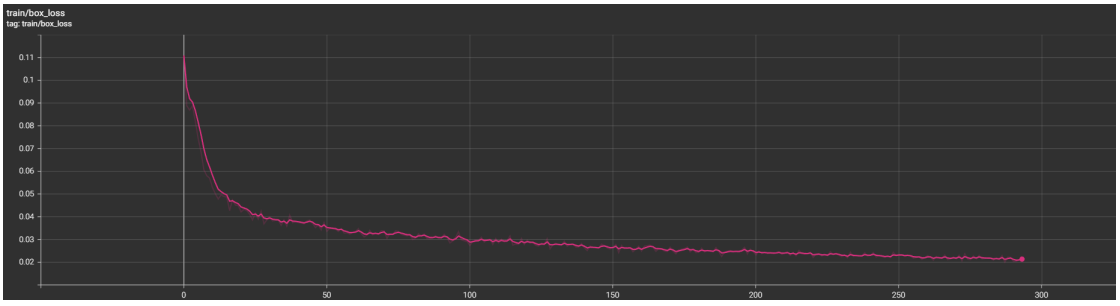

Figure 4.7. box_loss value vs Epochs

Finally, after training the model and obtaining satisfying performance results, we run inference with the trained weights on the test dataset. The final outcomes after running the inferences are saved and the images with the bounding boxes and predictions are displayed. The speed of the model for running inference on the test dataset is 0.5 ms pre-process, 13.7 ms inference, and 1.7 ms NMS (Non Maximum Suppression) per image at shape (1, 3, 416, 416).

4.2. Faster R-CNN Result

As mentioned in Section 3.2.2, after setting the parameters and starting the training process of the Faster RCNN model, it completed the process in 10 minutes and 29 seconds for 2000 iterations. As per the overall training speed for the 2000 iterations, it took approximately 0.31 seconds per single iteration. For training, the model loaded 439 images in the COCO format from the given location of the training dataset. There were no images found with any usable annotations. The distribution of the instances among all the categories is presented in the Table 4.2

Category	Instances
Ambulance	85
Bus	99
Car	457
Motorcycle	101
Truck	96
Total	838

Table 4.2. Distribution of instances for training Faster RCNN model

During the training process, we can observe various performance metrics and loss function evaluations for the model. In the Figure 4.8 , we can see that during the early steps of the process, the false negative value peaked at 0.98, but over the course of the process and by the last iteration, i.e., at step 2000, the value is reduced to approximately 0.05. In the Figure 4.9, we can observe that the class accuracy during the training process started from a value of 0.3 in the beginning steps but soon climbed up to 0.93 in 300 steps and then maintained an almost steady value with the peak value being 0.98 near the final iterations.

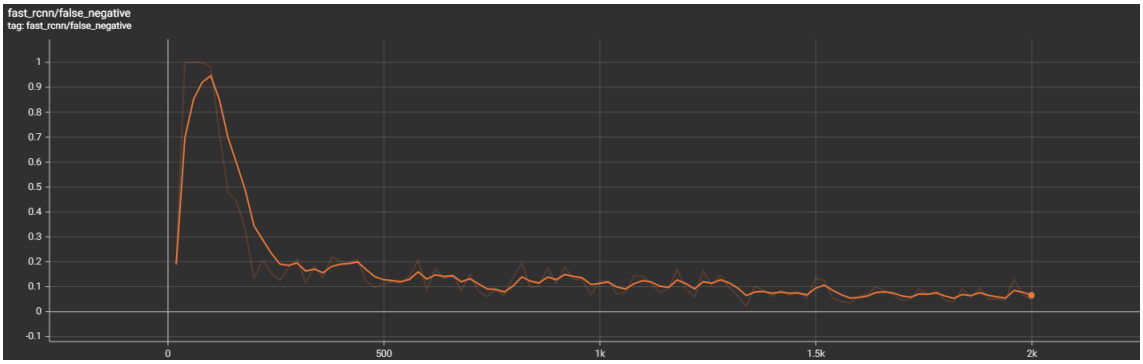

Figure 4.8. false_negative value vs Steps

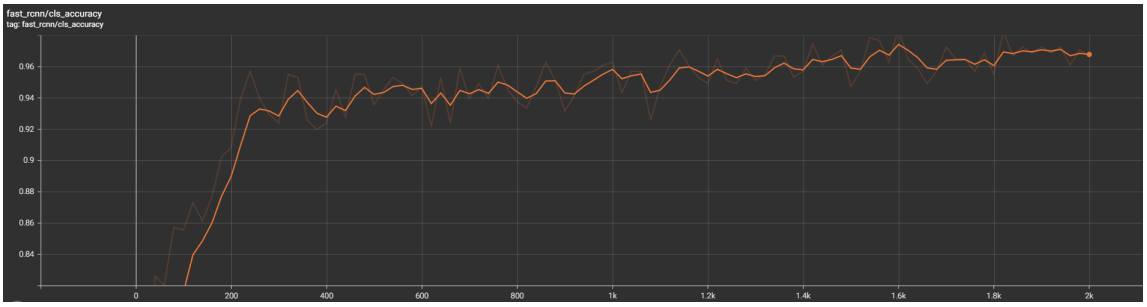

Figure 4.9. cls_accuracy value vs Steps

One of the important metrics to note during the training process are the loss functions. As we can see in Figure 4.10, the value of the loss_box_reg starts at more than 0.6 in the first 80 steps but then reduces to less than 0.3 after 200 steps, and finally is the lowest towards the final steps with the value of 0.17. The "loss_box_reg" is localization loss present in the Region of Interest head which calculates the loss for localisation of box. From the Figure 4.11, we can see the value of loss_cls being reduced from 1.73 in the first iteration to 0.08 in the last iteration. 'loss_cls' is the classification loss in the Region of Interest head which calculates the box classification loss, i.e., the value of the performance of model in labeling a correct class. In the Figure 4.12, we can see that the value of the 'loss_rpn_cls' starts from 0.022 in the beginning and then fluctuates between 9.81e-3 and 2.75e-3 for iterations after 400 and before 1400. Finally, at the end of the training process, the value reduces to 2.99e-3. This value corresponds to classification loss in RPN which calculates the "objectness" loss, i.e., the value of the performance of model in labeling the foreground anchor boxes and background anchor boxes correctly. Finally, we take a look at the total loss which is the weighted sum of all the individual losses calculated during the iterations, in the Figure 4.13. We can see that total loss follows the same trend of starting at a high value of 2.34 in the first steps and then finally reducing to value of 0.25 by the last step.

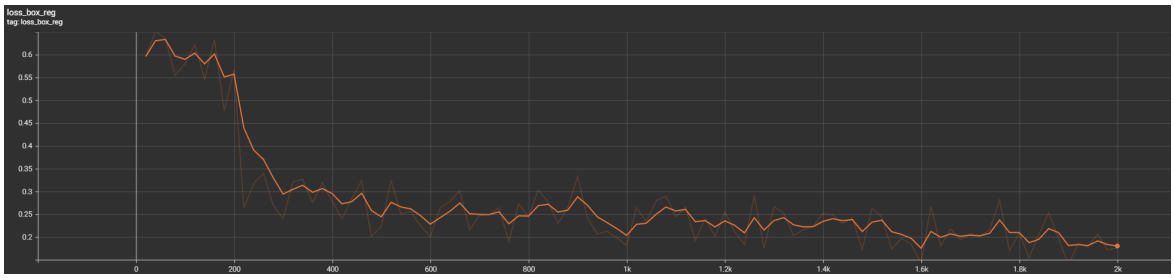

Figure 4.10. loss_box_reg value vs Steps

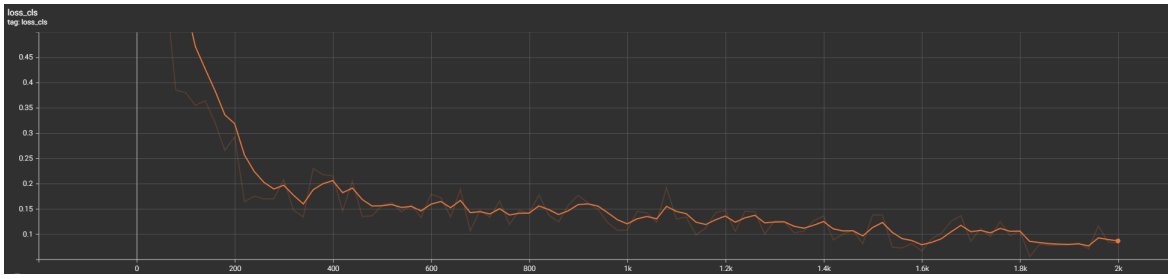

Figure 4.11. loss_cls value vs Steps

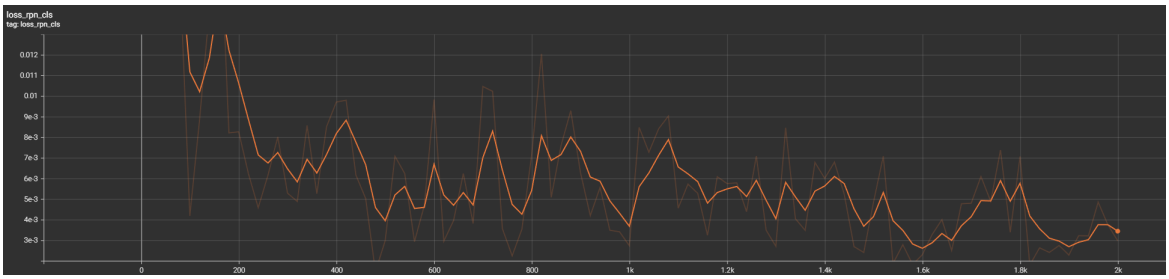

Figure 4.12. loss_rpn_class value vs Steps

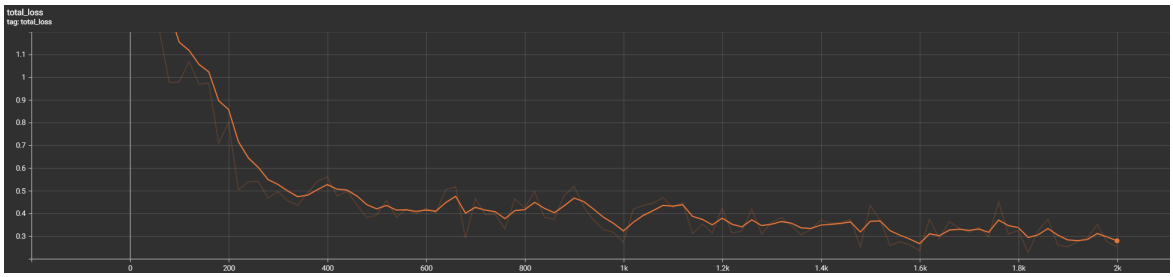

Figure 4.13. total_loss value vs Steps

After the training process is complete with satisfying metrics, we run the inference on the model using the test dataset. The time taken by the model for testing the dataset was 9 minutes and 57 seconds, with an average of 0.165 seconds per iteration. For testing, the model loaded 63 images in the COCO format from the given location of the testing dataset. The distribution of the instances among all the categories is presented in the Table 4.3

Category	Instances
Ambulance	9
Bus	19
Car	75
Motorcycle	16
Truck	10
Total	129

Table 4.3. Distribution of instances for testing Faster RCNN model

The evaluations from the inference on the testing dataset are presented in the Table 4.4 and Table 4.5

AP for IoU=0.50:0.95	area = all	maximum detections = 100	= 0.400
AP for IoU=0.50	area = all	maximum detections = 100	= 0.613
AP for IoU=0.75	area = all	maximum detections = 100	= 0.440
AP for IoU=0.50:0.95	area = small	maximum detections = 100	= 0.126
AP for IoU=0.50:0.95	area = medium	maximum detections = 100	= 0.104
AP for IoU=0.50:0.95	area = large	maximum detections = 100	= 0.505
AR for IoU=0.50:0.95	area = all	maximum detections = 1	= 0.363
AR for IoU=0.50:0.95	area = all	maximum detections = 10	= 0.477
AR for IoU=0.50:0.95	area = all	maximum detections = 100	= 0.478
AR for IoU=0.50:0.95	area = small	maximum detections = 100	= 0.277
AR for IoU=0.50:0.95	area = medium	maximum detections = 100	= 0.149
AR for IoU=0.50:0.95	area = large	maximum detections = 100	= 0.580

Table 4.4. Evaluation of the Faster RCNN Model

Category	AP
Car	38.262
Ambulance	78.344
Motorcycle	33.365
Bus	49.999
Truck	0.000

Table 4.5. Evaluation results per cateogry for bbox of the Faster RCNN Model

4.3. SSD Result

As mentioned in Section 3.2.3, we configured the pipeline.config file with parameters for the training process. Before starting the actual model training, it took a few minutes to set up everything and read the provided pipeline file and training.record file generated for the dataset. The actual training process took 46 minutes and 31.56 seconds for 5000 steps. During the training process, we can observe various performance metrics and loss function evaluations for the model. For every step, we can observe the values for these functions. One of the main metrics for the training process is the learning rate of the model. We used the default values for the pre-trained model for setting the base learning rate and its related values. From Figure 4.14, we can see that the learning rate during the training process started with a steady increase from 0 to up until 0.4 in the first 2500 steps, then it maintained the learning rate around that value for the next 1500 steps, but after that had a gradual dip and ended up at 0.38 by the final step of the training.

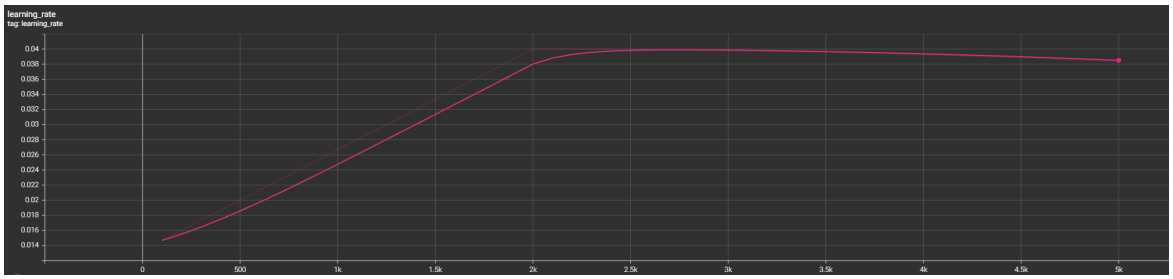

Figure 4.14. Learning Rate vs Steps

Another important metric to keep note of during the training process is the loss function. The loss function, in general, is the measure of how successfully the model can assess a certain amount of information in a given algorithm based on the quantity of data provided. For this model, during training there were logs of four different loss functions. First one being, classification loss which loosely means the classes or the labels correctly identified are more than the incorrectly identified ones. From the Figure 4.15, we can see that the classification loss value started at a high point of 0.89 in the beginning of the training process but then steadily decreased throughout with exception of a bump around 1500 steps and finally reducing to the lowest value of 0.32 after 5000 steps.

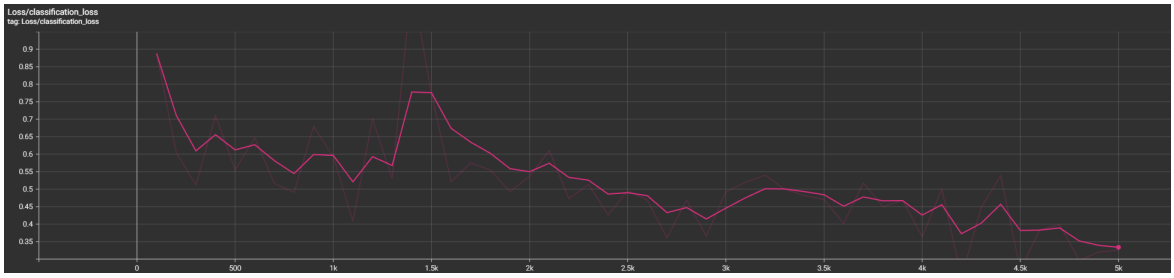

Figure 4.15. Classification Loss vs Steps

A second loss function that we have for this training process is the localization loss, which is a smooth L1 loss between the projected bounding box adjustment and the actual values. From Figure 4.16, we can see that the value for this loss started at 0.64 in the initial steps of training and then had a steady decline with a bit of fluctuation in the values but overall reducing to 0.2 by the end of the training process.

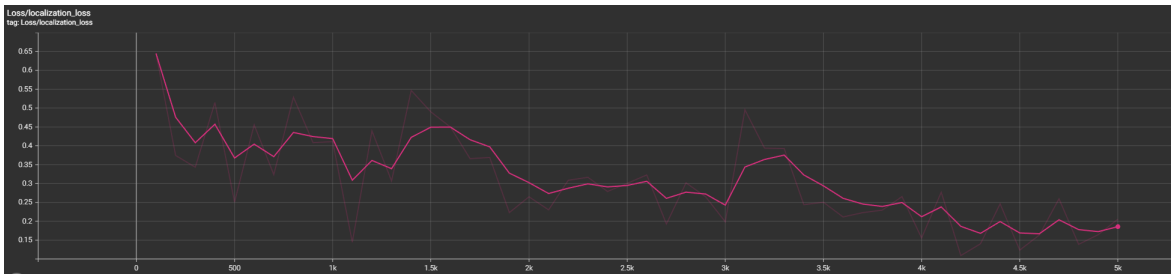

Figure 4.16. Localization Loss vs Steps

A third loss function that is noted during this training process is regularization loss. Regularization process helps the model generalize better. The regularization loss is the loss generated by the regularization functions. For our training process, as we can see in Figure 4.17, the value of the regularization loss started at a low of 0.26 in the initial steps, then had a steep increase after 1400 steps and had its peak value of 0.32 around 2000 steps, but then had a steady decline and finally reached the value of 0.27 by the end of the training process, i.e., at 5000 steps.

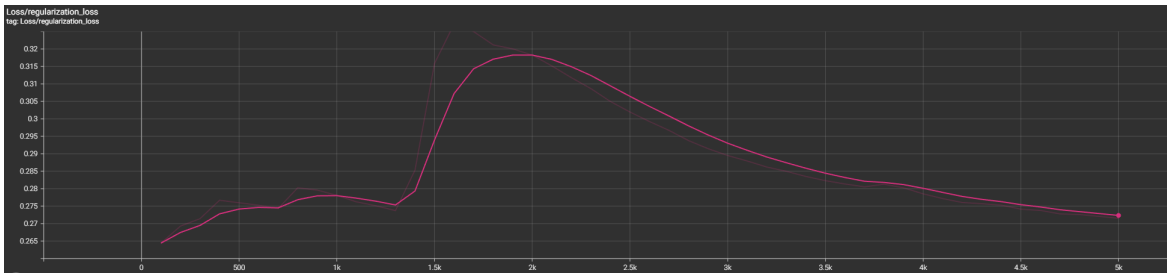

Figure 4.17. Regularization Loss vs Steps

Finally, we have the total loss function, which is the weighted sum of all the individual losses calculated during the steps of the training process. From Figure 4.18, we can observe that the value of the total loss function started at 1.8 in the initial steps of training but then had a great decrease in the value and reached around 1.3 by step 300. After that, the value had a steady decline but with various fluctuations along the way, with a peak value of 1.58 around 1500 steps and then the lowest value at the very last step of the training, 0.8.

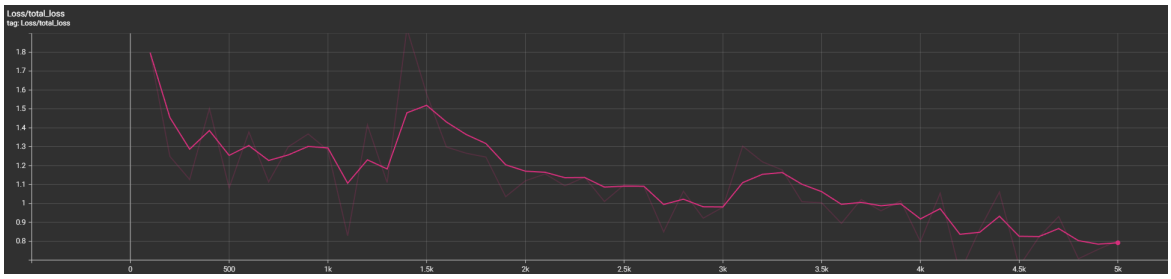

Figure 4.18. Total Loss vs Steps

After the successful completion of the training process, now we start with the testing process. For testing the model, 63 images were provided as per the dataset split, and the model performed the testing and evaluation in less than 1 minute. The results from the evaluation are present in Table 4.6 and Table 4.7 for the final step.

AP for IoU=0.50:0.95	area = all	maximum detections = 100	= 0.222
AP for IoU=0.50	area = all	maximum detections = 100	= 0.357
AP for IoU=0.75	area = all	maximum detections = 100	= 0.264
AP for IoU=0.50:0.95	area = small	maximum detections = 100	= 0.087
AP for IoU=0.50:0.95	area = medium	maximum detections = 100	= 0.048
AP for IoU=0.50:0.95	area = large	maximum detections = 100	= 0.276
AR for IoU=0.50:0.95	area = all	maximum detections = 1	= 0.283
AR for IoU=0.50:0.95	area = all	maximum detections = 10	= 0.395
AR for IoU=0.50:0.95	area = all	maximum detections = 100	= 0.409
AR for IoU=0.50:0.95	area = small	maximum detections = 100	= 0.171
AR for IoU=0.50:0.95	area = medium	maximum detections = 100	= 0.254
AR for IoU=0.50:0.95	area = large	maximum detections = 100	= 0.483

Table 4.6. Evaluation Results for the SSD Model

Localization Loss	0.324438
Classification Loss	0.566739
Regularization Loss	0.271516
Total Loss	1.162693

Table 4.7. Evaluation of Loss Functions for the SSD Model

4.4. Evaluation

In this section, we will compare the results obtained from the chosen object detection models for this thesis, i.e., YOLOv5, Faster RCNN using Detectron2, and SSD using TensorFlow 2. The models are trained and tested on the same dataset in order to achieve

results that can be compared quantitatively. We will evaluate the performance of the before mentioned models on some common performance metrics and also take a look at the time taken by each model to train on the dataset to acquire the presented results.

4.4.1. Training Time

The time the models required for training on the dataset is presented in the Table 4.8. As we can see, the YOLOv5 model completed the training in 27 minutes and 46 seconds for 294 epochs, which results in 11760 iterations, whereas the Faster RCNN model took 10 minutes and 29 seconds for 2000 iterations and the SSD model completed the training process in 46 minutes and 31.56 seconds for 5000 iterations. This gives us the average time taken by the models per step, as presented in the Table 4.8. Although Faster RCNN took the least amount of time for training overall, YOLOv5 has significantly less time per step. The time taken by YOLOv5 model per iteration is almost half of what the Faster RCNN takes. The SSD model has the longest time per iteration for the training process.

Model	Total Training Time	Total Iterations	Average Time/Step
YOLOv5	27 min 46 sec	11760	0.142 sec
Faster RCNN	10 min 29 sec	2000	0.315 sec
SSD	46 min 31.56 sec	5000	0.558 sec

Table 4.8. Evaluation of Loss Functions for the SSD Model

4.4.2. Performance Metric

One of the most important metrics to measure the performance of any object detection model is mAP (mean Average Precision). The value of mAP is the score generated by comparing the bounding box of ground truth with the boundix box detected by the model. The higher the value obtained for this metric mAP, the more precise are the findings of the model. mAP@0.5 corresponds to the mAP for IoU (Intersection over Union) thresholds 0.5, and mAP@0.5:0.95 corresponds to the average mAP for IoU thresholds from 0.5 to 0.95 with a step size of 0.05 (0.5, 0.55, 0.6, 0.65, 0.7, 0.75, 0.8, 0.85, 0.9, 0.95). We can see the evaluation results from the three models for the mAP@0.5 values and mAP@0.5:0.95 values for the test dataset in Table 4.9.

As is evident from the Table 4.9, for mAP with IoU threshold 0.5, the best results are obtained from the model Faster RCNN with the precision percentage being equal to 61.3%. YOLOv5 comes in as close second with 56.2% for the same metric and SSD comes in last with a low value of 35.7%. The map@0.5 value for Faster RCNN is almost double than of SSD with YOLOv5 being only around 8% less than Faster RCNN's value.

Model	mAP@0.5	mAP@0.5:0.95
YOLOv5	0.562	0.413
Faster RCNN	0.613	0.400
SSD	0.357	0.222

Table 4.9. Comparsion of mAP@0.5 and mAP@0.5:0.95

For the metric of average mAP calculated over IoU thresholds ranging from 0.5 to 0.95 with a 0.05 step value, we can see that YOLOv5 takes the first stop with the highest value among all the three models of 41.3%. Faster RCNN comes a close second with a value of 40%, which is only slightly less than YOLOv5. The SSD model scores the lowest value again in this metric as well, with a percentage of 22.2%. The mAP@0.5:0.95 value for SSD is almost half of the values obtained by the other two models.

5. Summary

The work done under the scope of this thesis constitutes state-of-the-art object detection models and deep learning neural network frameworks. The purpose of the present degree project is to research the latest object detection models in deep convolutional neural networks, examine prominent deep object detection frameworks, and evaluate their performance for a certain dataset.

After researching about the object detection models and a brief literature review of the object detection algorithms and frameworks, three object detection models, namely, YOLOv5 (You Only Look Once), Faster R-CNN (Regional Convolutional Neural Network) using Detectron2 framework, and SSD (Single Shot Detector) using TensorFlow 2 framework, were selected to perform experiments and then performance evaluation. The dataset was chosen with images related to autonomous vehicle driving, and the size of the dataset chosen was relatively small due to some delimitation and to allow the experiments to be conducted in a reasonable amount of time.

All the models were trained and tested using Google Colab online environment. The YOLOv5 model was implemented using the online available source code from the authors at Ultralytics GitHub, the Faster RCNN model was implemented using the Detectron2 framework source code available for various pre-trained model in the official GitHub repository of Facebook AI Research Team and the SSD model was implemented using the TensorFlow 2 framework's official GitHub repository with pre trained models. All the models were trained on the selected custom dataset and optimized as required.

The models YOLOv5 and Faster RCNN outperformed the SSD model by a great margin in terms of time taken for training and the final mean Average Precision (mAP) value for the test dataset. The YOLOv5 model proved to be the fastest, taking less than half the time taken by the Faster RCNN model in training. Faster RCNN model proved to be slightly more accurate than YOLOv5 for mAP@0.5, but for mAP@0.5:0.95 metric, YOLOv5 outperformed Faster RCNN model by a slight margin. Concluding, the findings of the thesis are in clear favor of YOLOv5 in terms of speed and accuracy, but with a slight advantage to Faster RCNN in terms of accuracy for a trade-off of speed.

Bibliography

[1] J. Miya and M. A. Ansari, "Medical images performance analysis and observations with spiht and wavelet techniques", *Journal of Information and Optimization Sciences*, vol. 41, pp. 273–282, Jan. 2020. DOI: 10.1080/02522667.2020.1721616.

[2] R. Girshick, J. Donahue, T. Darrell, and J. Malik, "Rich feature hierarchies for accurate object detection and semantic segmentation", *2014 IEEE Conference on Computer Vision and Pattern Recognition*, Jun. 2014. DOI: 10.1109/cvpr.2014.81. [Online]. Available: https://dl.acm.org/citation.cfm?id=2679851.

[3] R. Girshick, "Fast r-cnn", *2015 IEEE International Conference on Computer Vision (ICCV)*, pp. 1440–1448, Dec. 2015. DOI: 10.1109/iccv.2015.169.

[4] S. Ren, K. He, R. Girshick, and J. Sun, "Faster r-cnn: Towards real-time object detection with region proposal networks", *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 39, pp. 1137–1149, Jun. 2017. DOI: 10.1109/tpami.2016.2577031.

[5] J. Redmon, S. Divvala, R. Girshick, and A. Farhadi, "You only look once: Unified, real-time object detection", *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2016. DOI: 10.1109/cvpr.2016.91. [Online]. Available: https://arxiv.org/pdf/1506.02640.pdf.

[6] J. Redmon and A. Farhadi, "Yolo9000: Better, faster, stronger", *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, Jul. 2017. DOI: 10.1109/cvpr.2017.690.

[7] J. Redmon and A. Farhadi, "Yolov3: An incremental improvement", *arXiv preprint arXiv:1804.02767*, 2018. [Online]. Available: https://arxiv.org/abs/1804.02767.

[8] W. Liu, D. Anguelov, D. Erhan, *et al.*, "Ssd: Single shot multibox detector", *Computer Vision – ECCV 2016*, pp. 21–37, Sep. 2016. DOI: 10.1007/978-3-319-46448-0_2. [Online]. Available: https://arxiv.org/abs/1512.02325.

[9] J. Solawetz, *Vehicles-openimages object detection dataset*, Roboflow, Jun. 2020. [Online]. Available: https://public.roboflow.com/object-detection/vehicles-openimages.

[10] F. Yu, H. Chen, X. Wang, *et al.*, *Bdd100k: A diverse driving dataset for heterogeneous multitask learning*, 2020. [Online]. Available: https://openaccess.thecvf.com/content_CVPR_2020/papers/Yu_BDD100K_A_Diverse_Driving_Dataset_for_Heterogeneous_Multitask_Learning_CVPR_2020_paper.pdf.

[11] D. Lomonaco and Maltoni, *Deep learning for computer vision: A comparison between convolutional neural networks and hierarchical temporal memories on object recognition tasks*, Sep. 2015. [Online]. Available: https://amslaurea.unibo.it/9095/1/Vincenzo_Lomonaco_tesi.pdf.

[12] W. Contributors, *Artificial neural network*, Wikipedia, Dec. 2018. [Online]. Available: https://en.wikipedia.org/wiki/Artificial_neural_network.

[13] J. Emmons, S. Fouladi, G. Ananthanarayanan, S. Venkataraman, S. Savarese, and K. Winstein, "Cracking open the dnn black-box", *Proceedings of the 2019 Workshop on Hot Topics in Video Analytics and Intelligent Edges - HotEdgeVideo'19*, 2019. DOI: 10.1145/3349614.3356023.

[14] *What are neural networks?*, IBM, Aug. 2020. [Online]. Available: https://www.ibm.com/cloud/learn/neural-networks.

[15] C. Camacho, *Convolutional neural networks*, Jun. 2018. [Online]. Available: https://cezannec.github.io/Convolutional_Neural_Networks/.

[16] M. Stoycheva, *Uncertainty estimation in deep neural object detectors for autonomous driving*, 2021.

[17] J. Brownlee, *A gentle introduction to object recognition with deep learning*, Machine Learning Mastery, May 2019. [Online]. Available: https://machinelearningmastery.com/object-recognition-with-deep-learning.

[18] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks", *Communications of the ACM*, vol. 60, pp. 84–90, May 2012. DOI: 10.1145/3065386.

[19] R. Padilla, W. L. Passos, T. L. B. Dias, S. L. Netto, and E. A. B. da Silva, "A comparative analysis of object detection metrics with a companion open-source toolkit", *Electronics*, vol. 10, 2021. DOI: 10.3390/electronics10030279. [Online]. Available: https://www.mdpi.com/2079-9292/10/3/279.

[20] M. Everingham, L. Van Gool, C. K. I. Williams, J. Winn, and A. Zisserman, "The pascal visual object classes (voc) challenge", *International Journal of Computer Vision*, vol. 88, pp. 303–338, Sep. 2009. DOI: 10.1007/s11263-009-0275-4.

[21] T.-Y. Lin, M. Maire, S. Belongie, *et al.*, "Microsoft coco: Common objects in context", *Computer Vision – ECCV 2014*, pp. 740–755, 2014. DOI: 10.1007/978-3-319-10602-1_48. [Online]. Available: https://link.springer.com/chapter/10.1007/978-3-319-10602-1_48.

[22] *Deep learning frameworks*, NVIDIA Developer, Apr. 2016. [Online]. Available: https://developer.nvidia.com/deep-learning-frameworks.

[23] G. Bradski, "The OpenCV Library", *Dr. Dobb's Journal of Software Tools*, 2000.

[24] TensorFlow, *Why tensorflow*, TensorFlow. [Online]. Available: https://www.tensorflow.org/about.

[25] *Detectron2*, Facebook. [Online]. Available: https://ai.facebook.com/tools/detectron2/.

[26] Keras, *Keras documentation*, Keras, 2019. [Online]. Available: https://keras.io/.

[27] O. Russakovsky, J. Deng, H. Su, *et al.*, "Imagenet large scale visual recognition challenge", *International Journal of Computer Vision*, vol. 115, pp. 211–252, Apr. 2015. DOI: 10.1007/s11263-015-0816-y. [Online]. Available: https://hci.stanford.edu/publications/2015/scenegraphs/imagenet-challenge.pdf.

[28] T.-Y. Lin, P. Dollar, R. Girshick, K. He, B. Hariharan, and S. Belongie, "Feature pyramid networks for object detection", *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, Jul. 2017. DOI: 10.1109/cvpr.2017.106.

[29] A. Bochkovskiy, C.-Y. Wang, and H.-Y. Liao, *Yolov4: Optimal speed and accuracy of object detection*, Apr. 2020. [Online]. Available: https://arxiv.org/pdf/2004.10934.pdf.

[30] G. Jocher, *Ultralytics/yolov5*, Ultralytics, Aug. 2020. [Online]. Available: https://github.com/ultralytics/yolov5.

[31] A. Kuznetsova, H. Rom, N. Alldrin, *et al.*, "The open images dataset v4", *International Journal of Computer Vision*, Mar. 2020. DOI: 10.1007/s11263-020-01316-z.

[32] *Detectron2 source code*, Facebook, May 2020. [Online]. Available: https://github.com/facebookresearch/detectron2.

[33] Y. Hongkun, C. Chen, D. Xianzhi, *et al.*, *Tensorflow model garden*, 2020. [Online]. Available: https://github.com/tensorflow/models.

List of Figures

List of Tables