AI Scene Descriptor for the Visually Impaired

Randeev David

Supervisor : Sara Saravi

2022/2023

Abstract

People with visual impairments both severe and mild, face the challenge of not being able to fully process and understand their surroundings the same way those without impairments do. While those with mild impairments are able to use lenses in order to improve their ability to perceive their surroundings, this is not the case for those who suffer from complete blindness or severe impairments. This project aims to develop, or at least lay the groundwork for a solution that allows those with visual impairments to understand the contents of an image/scene and if applied in real time, can be extended to their surroundings as well. The goal is to use object detection to achieve this and thereafter convey the detected objects to the user in a form they can understand, which in the context of the visually impaired is ideally verbal.

Contents

1 Introduction **3**
- 1.1 Project Aims . . . 3
- 1.2 Motivation . . . 4
- 1.3 Project Objectives . . . 4

2 Literature Review **5**
- 2.1 Existing solutions for supporting the visually impaired . . . 5
  - 2.1.1 Seeing AI . . . 5
  - 2.1.2 Be My Eyes . . . 5
  - 2.1.3 Envision . . . 5
- 2.2 Computer Vision and Object Detection Algorithms . . . 6
  - 2.2.1 Region Based Convolutional Neural Networks . . . 6
  - 2.2.2 Single Shot Detector (SSD) . . . 7
  - 2.2.3 You Only Look Once (YOLO) . . . 8

3 Methodology **10**
- 3.1 Tools . . . 10
  - 3.1.1 Programming Language . . . 10
  - 3.1.2 TensorFlow . . . 10
  - 3.1.3 OpenCV . . . 10
  - 3.1.4 NumPy . . . 11
  - 3.1.5 Matplotlib . . . 11
- 3.2 Selecting an object detection algorithm . . . 11
  - 3.2.1 Faster-RCNN . . . 11
  - 3.2.2 Single Shot Detector (SSD) . . . 11
  - 3.2.3 YOLO . . . 12
- 3.3 Data Collection and Preparation . . . 12
  - 3.3.1 Mapillary Dataset . . . 12
  - 3.3.2 Udacity Dataset . . . 13
- 3.4 Performance Metrics . . . 14

4 Design and Implementation **15**
- 4.1 System Architecture . . . 15
  - 4.1.1 Input Layer . . . 16
  - 4.1.2 Convolutional Layers . . . 16
  - 4.1.3 Leaky ReLu Activation Function . . . 16
  - 4.1.4 Fully Connected Layers . . . 16
  - 4.1.5 Loss Function . . . 17

5 Experimental Results **19**
- 5.1 Setup and Limitations . . . 19
- 5.2 Experiments and changes tried . . . 19
  - 5.2.1 Results . . . 20

6 Conclusion **25**
- 6.1 Summary of project . . . 25
- 6.2 Summary of methodology . . . 25
- 6.3 Summary of experimental results . . . 25
- 6.4 Limitations and Challenges . . . 25
- 6.5 Future Work . . . 26

6.6 Final Thoughts 26

Chapter 1

Introduction

Visual impairments, which implies all impairments related to sight, ranging from complete blindness to a mild impairment affects an approximated 2.2 billion people worldwide [1]. In many cases, the impact these impairments have on the person can be mitigated by the use of prescription lenses. However, a large number out of those mentioned above, are considered to be completely blind i.e. a total loss of vision. These people who suffer from visual impairments whether they are completely blind or partly, face a multitude of challenges in their day to day lives as a result. The main challenge they face is their inability to fully process and understand their surroundings. This is where the idea of products such as scene descriptors come in, where technology is used to improve their quality of life and address the challenges they face as a result of their impairments.

Such technology is considered to fall under the general term of assistive technology. The World Health Organization (WHO) defines assistive technology as any product or service that "maintains or improves an individual's functioning and independence, thereby promoting their well-being" [2]. Some commonly known assistive technologies include hearing aids, prosthesis, spectacles and contact lenses. As of 2018, more than 1 billion people globally, required 1 or more assistive products with only 1 in 10 people actually having access to them [2]. This highlights not only the importance of assistive technology but also the importance of developing accessible and affordable ones.

This project uses computer vision and more specifically object detection to achieve the main task of identifying the contents of an image. Therefore, it is important to note the impact these fields have had on the development of not only assistive technology, but also in the development of various other cutting edge technologies. In recent years, computer vision which is essentially the act of getting computers to see, has made significant advances. A major factor in these advancements is the increased attention and research being conducted on artificial intelligence and more importantly artificial neural networks (ANN). Computer vision as a discipline, is at a point where it is able to identify and detect what it is trained on, at almost human level accuracy and in some cases even better. With this kind of precision and accuracy the applications of computer vision are limitless. Combined with the abilities of deep learning and convolutional neural networks (CNN), machines are being trained on large datasets of images, allowing them to learn how to identify different things. The ever increasing amount of digital data being generated, has contributed significantly to the ability of these machines to learn how to detect objects and observe scenes. Thus enabling computer vision to be applied to a multitude of tasks. Autonomous driving, facial recognition and detecting faults in manufacturing are just a few of the applications and industries being revolutionized by computer vision.

1.1 Project Aims

The main goal of this project is to use a selected object detection algorithm to implement an effective scene descriptor which can be used to get an understanding of the contents of an image or scene. The project will also focus on a suitable solution to convert the predictions generated by the model into a medium that is accessible by all regardless of whether they have a visual impairment or not. If time permits, some work may be done on a suitable user interface for the solution.

While the project aims to deliver a functioning scene descriptor that can be used by the visually impaired to improve their ability to understand their surroundings, the final result may not cover all aspects of it. The reason for this relates to limitations faced in terms of time, computational power and accessibility to high quality data. The process of implementing and training an object detection model is not an easy task as I've discovered through the work done on this project. It also requires more computational resources than my basic laptop and therefore was restricted to using the lab machines for all parts related to the object detection model.

1.2 Motivation

If successfully implemented, this project hopes to deliver a tool that will improve the quality of life for not only those who suffer with visual impairments but also those around them such as their loved ones and society in general. If the final product is not a complete solution, the project will aim to deliver a strong foundation on top of which a fully functional tool can be developed. By exploring the use of deep learning and object detection in such a manner, I hope it can inspire further research into similar use cases which will ultimately help improve the quality of life for so many people.

1.3 Project Objectives

This section lays out the main objectives of the goal which I hope to achieve.

- Review existing solutions and literature that is related or applicable to the concept of a scene descriptor.
- Based on the above, decide on a suitable method to implementing a scene descriptor such as a suitable object detection model.
- Identify and get familiar with any new techniques, libraries or concepts that I would need.
- Select and prepare a suitable dataset that can be used for training a model for detecting objects.
- Implement the selected object detection model along with all its components such as the loss function etc.
- Evaluate the model based on its accuracy and other important factors.
- Provide the implemented model in a suitable format according to the project.
- Write a report documenting the whole project.

Chapter 2

Literature Review

2.1 Existing solutions for supporting the visually impaired

This section will provide an overview of existing assistive technologies and other solutions that are aimed at improving the lives of the visually impaired.

2.1.1 Seeing AI

This product developed by Microsoft, is an IOS app that is able to give audible explanations based on what the device's camera is pointed at [3]. The app achieves exactly what this project hopes to achieve but on a much larger scale as it has a number of channels which are dedicated to identifying various things. Some of the channels included are for identifying people, text, currency, colours and other objects. The most impressive channel in my opinion, is the channel that exists for describing the scene around you. Having tried the app myself, I can safely say that it performs extremely well. By simply pointing the camera at a desired scene and tapping the screen, the app is able to provide extremely accurate descriptions. The app is so advanced that the people channel not only learns to recognize the user's friends but also predicts the ages of the people it identifies. The Seeing AI app has even evolved to the point where it can be used in other apps such as Twitter for example, thereby allowing those with visual impairments the chance to engage with apps in a way they would previously have been unable to. This app can therefore be considered as an impressively useful tool for the visually impaired and achieves exactly what this project aims to focus on and much more.

2.1.2 Be My Eyes

Be My Eyes is yet another innovative solution that has helped millions of people with visual impairments. The idea behind the app is to connect those with impairments, to a network of volunteers who can sign up from where ever they are to quite literally be someone else's eyes [4]. It enables those who would like to be of service to others, an easy solution to volunteering, as they can do their tasks as volunteers from wherever they are and whenever they are free, eliminating the need to set aside separate time to plan and volunteer. The app connects the user and the volunteer by video call allowing the user to ask the volunteer anything ranging from which way they need to turn to get to a train to whether a carton of milk has expired. By making this connection, it allows the visually impaired users to not feel like they are a burden to their friends and family and also get along with their lives a lot easier, as a result of now having an entire network of volunteers just a few taps away.

One problem that this product could face is the limited number of volunteers at any given time. This could potentially reduce the benefit of the app, if there are times when a user is unable to connect to a volunteer. The newest update to the app solves this issue as well. The Be My Eyes app has now introduced a virtual volunteer which is powered by OpenAI's GPT4 [5]. This addition ensures that the users will always have someone to assist them at any time. This product has been used by millions of visually impaired people and continues to help them improve their lives.

2.1.3 Envision

This section will cover more than one solution which has been developed for the ease of visually impaired users. The first product is an app called the Envision app [6] which is very similar to the Seeing AI app mentioned previously. The app uses the camera on your smartphone in order to read text or even entire books, describe scenes, detect colours, recognize and learn people, among other things.

The second product offered by Envision, are the Envision glasses [7]. This product is a pair of glasses with an in-built camera and speaker which allows for articulating the surroundings or reading text as desired.

These glasses allow for an even more seamless way for people with visual impairments to feel more included and involved in a group setting, while also enabling them to perform day to day tasks with increased ease. By incorporating all the features of the app into a pair of glasses, Envision has allowed its users to have a hassle free method of getting the information they need when they need it without even having to take their phone out.

Envision also has a feature called Ally, which incorporates a feature similar to the idea behind the Be My Eyes app right into its glasses [8]. This feature is an acknowledgement to the fact that there will be instances where some human input would be useful. It allows the user to video call trusted friends and family from the glasses itself, allowing the user to get help from a loved one.

All in all, Envision's products seem to be well tailored to the needs of the blind or visually impaired, by giving them useful tools that can help them navigate life with ease and independence.

2.2 Computer Vision and Object Detection Algorithms

Over the last decade or so, the amount of research into computer vision tasks such as image classification, object detection, semantic segmentation etc has increased greatly. This is in no small part due to the ground breaking research on deep learning that has been put forward by numerous experts. Similarly, the rapid rise in the quantity of digital data being produced throughout the world has also had an impact on this. This is because the larger the quantity of data that can be used to train a model, the more likely the model's accuracy will improve. A good example for this is the creation of the ImageNet database which consists of images in the millions and also a 1000 classes [9]. The ImageNet database is now used in almost all major computer vision tasks.

The use of deep neural networks in tasks like object detection, has completely changed the way computer vision is approached. Prior to the use of deep neural networks, tasks such as feature extraction and the hierarchy of features were all specified and done manually using algorithms that would identify and extract the features that were thought to be important. Histogram of Oriented Gradients (HOG), Scale Invariant Feature Transform (SIFT) and Speeded Up Robust Features (SURF) were just a few methods that were used [10]. After the introduction of deep learning however, this was changed drastically. Feature extraction was now an automated process as part of the deep neural network, and it could also identify which features were most important to detect [11].

In the initial stages of this project, I had to look into the various object detection algorithms that were available for a task of this kind. The following sub sections will cover a few that I looked into.

2.2.1 Region Based Convolutional Neural Networks

RCNN

The first algorithm I looked into was the RCNN [12], which was followed by improved versions of it known as Fast RCNN and Faster RCNN. This section will give a brief overview of my findings.

The original RCNN algorithm was considered a breakthrough in the field of object detection due to it's significantly higher accuracy when compared to existing systems at the time. RCNN combined the idea of region proposals with the concept of CNNs to achieve this high accuracy.

The concept of region proposals is an important aspect of the RCNN algorithm. The way these regions are proposed is by using a selective search algorithm which divided the input image into different regions by taking into consideration similarities between pixels. These similarities that were considered included a combination of both high level and low level features such as intensity and colour for example.

These proposed regions were then resized and fed as input to a pre trained CNN. The purpose of this was to extract features from the proposed regions which could then be used to classify the objects in them, if any. These features that were output by the CNN were used by a Support Vector Machine (SVM) to predict a class score. A SVM is an algorithm that is used for solving classification and regression tasks. The way it works is that the SVM will take as input the features of a proposed region and then output a class score which is an indication of how likely it thinks that the region contains an object of that class. As implied by the previous sentence, a SVM is trained for each class for this purpose including one for the background class which is for when there is no object. The features output by the CNN are also used for the purpose of performing bounding box regression. This is where a linear regression model is used for predicting the required adjustments that need to be made to the coordinates of the proposed regions, so that it covers the entire object properly. For this purpose as well, a regression model is trained for each class.

The RCNN also uses Non-Maximum Suppression (NMS), which is a technique used to get rid of redundant proposals or bounding boxes. The idea behind NMS, is to select the region with the highest probability of having an object and then get rid of the other regions that have an IOU more than a specified value. This is then repeated until there are no regions that have an intersection over union (IOU) with any other region over

the specified threshold. IOU is basically a measure of how much overlap the two boxes/regions have. A higher IOU means there is more overlap between the two regions/boxes. Since RCNN deals with class scores and not objectness scores, NMS is applied for each class, thereby giving the same outcome of not having duplicate or redundant boxes/regions.

Fast-RCNN

The next algorithm that was introduced as part of the RCNN family was the Fast RCNN [13]. It was introduced to be an improved version of the RCNN algorithm. The changes made were mostly related to tackling the shortcomings of the previous version and as the name suggests, increasing the speed. The improved version also had notably higher accuracy and efficiency.

One improvement was the improved method of extracting features. Rather than extracting features from each region the way RCNN does, Fast RCNN extracts features for the whole image in one go. As a result, only one pass through the network is required, compared to running a forward pass for each region proposal in RCNN. A ROI (region of interest) pooling layer is then used to extract the associated feature vector for each region proposal. The ROI pooling layer is basically a max pooling layer that converts the feature vector for a proposal to a fixed size.

Another improvement seen in the Fast RCNN is how the model moves away from having different models for predicting the class score, and the offsets or adjustments to the region coordinates. Instead, Fast RCNN uses a single network with two output branches, one for predicting the class scores and one for predicting the adjustments to the coordinates.

Fast RCNN also uses a multi task loss which is made up of a classification loss for the class scores and a bounding box regression loss for the coordinate adjustments. By adding this multi task loss, the model is able to be trained end to end, which means it is able to learn to predict the output directly from the input data. This end to end training as a result of the model being a single network is what makes the Fast RCNN significantly faster than it's predecessor.

Faster-RCNN

The next version in the RCNN family is the Faster RCNN [14]. It is also the last one to be discussed in this section. The main improvement in the Faster RCNN, is related to the generation of region proposals. This algorithm gets rid of the selective search approach and introduces a Region Proposal Network (RPN). The Faster RCNN essentially integrates the RPN with the Fast RCNN to combine the proposing of regions and the object detection. The RPN is a CNN that takes the feature map output by the base CNN (feature extractor) as input, and outputs a set of region proposals that each have an objectness score, which represents the likelihood of the region containing an object. The way this is done is by the RPN applying a set of predefined anchor boxes in each location of the feature map. To explain what I mean by each location, consider the feature map as having a grid like structure where each cell in the grid represents a region of the image. The anchor boxes are predefined as I mentioned before and are all of different sizes and aspect ratios. The RPN then predicts for each of these anchor boxes an objectness score, as well as adjustments for the anchor box coordinates that will help it better capture the object it is detecting. The boxes with the highest objectness scores are then considered and adjusted according to the predicted adjustments. NMS is applied to these proposals/updated anchor boxes in order to get rid of duplicates or redundant proposals and then fed into the Fast RCNN for object detection. By combining the RPN for region proposals and the Fast RCNN for object detection, the Faster RCNN model as a whole is then able to be trained end to end. Since the speed of region proposals is significantly faster with the use of the RPN and all predictions are made with a single pass through the network, the Faster RCNN is not only able to give predictions with high accuracy but also do so with speed.

2.2.2 Single Shot Detector (SSD)

Another algorithm I looked into was the Single Shot MultiBox Detector [15] or SSD as it is more commonly known. As implied by the name, SSD carries out object detection in a single pass of the network by predicting both the class and the bounding box coordinates together. The SSD is essentially made up of a pre-trained CNN which can be called a feature extractor, with additional convolutional layers on top of that, that perform the detections/predictions. There are a few concepts that were used in the SSD that resulted in it being a breakthrough in object detection algorithms.

One such feature was the use of multi-scale feature maps. The feature extractor, which as mentioned before is a pre-trained CNN, will create feature maps from the input image with different scales. This allows the SSD to capture both high level as well as low level features. By doing so, it is able to detect and capture objects of various scales with high accuracy.

Another feature of the SSD, is the concept of default boxes which are similar to the idea of anchor boxes in Faster-RCNN. These default boxes are of a predefined aspect ratio which vary based on the dataset and type

of objects it is trying to detect. The number of default boxes is also something that is decided based on the task. In a SSD, each feature map produced is divided into a grid and the set of default boxes are applied to each cell in that grid. By doing this, the model is able to better predict objects of different sizes as the feature maps generated are of different scales as mentioned before, thereby enabling the model to actually pick these up regardless of their size.

The SSD algorithm also uses the additional convolutional layers I mentioned before, that are added on top of the feature extractor to predict the class score, as well as the bounding box offsets for each of the default boxes applied to each cell in the manner mentioned above. The class score is basically the likelihood or probability that the object in that default box belongs to a particular class. So if there are five classes, each default box will have five class scores associated with it i.e. one for each class. The bounding box offsets are basically the adjustments that need to be made to the default box to fully enclose or fit the object it is detecting, if any. These offsets are needed because, while the default boxes are defined based on the dataset and the objects in it, they won't always perfectly fit the objects. Therefore by learning to predict these offsets accurately the model is able to effectively localize the detected object. This is similar to the concept of bounding box regression described in the RCNN section.

The loss function used by the SSD algorithm is a combined loss function, that focuses on the localization loss which is related to the prediction of the offsets and the confidence loss which deals with the class predictions/scores.

Based on the approach taken by the SSD, it is clear that there will be many predicted boxes which are likely to be duplicates. To deal with this, the SSD uses NMS. In SSD, NMS is applied in exactly the same way it was used in RCNN, which was by performing it for each class.

This gives a brief summary of the SSD object detection algorithm and the idea behind it.

2.2.3 You Only Look Once (YOLO)

The next set of algorithms I looked into was the YOLO family. Since it's first release in 2016, which was YOLOv1, there have been a number of versions each improving on some aspect of the previous. The different versions of the algorithm have been developed by various groups and people, with the original authors also being involved in a few of the versions that came after. The name YOLO which stands for You Only Look Once, comes from the fact that a YOLO model will only need to look at the image once in order to accomplish the necessary tasks such as predicting bounding boxes and their classes.

YOLOv1

The idea behind YOLO, which was put forward in the original paper which covered YOLOv1, is to have a single stage object detection algorithm i.e. unify the different components of object detection [16]. YOLOv1 works by splitting the input image into a specified number of grid cells (S), and each of these cells will be tasked with predicting any object that has its center within that cell. Each cell then predicts a predefined number of boxes (B) and confidence scores for each of the predicted boxes, which represents how confident the model is that the predicted box contains an object. Each prediction will be made up of the x and y coordinates of the center of the bounding box, the width and height of the bounding box, confidence score and the class prediction. The confidence is represented by the IOU between the predicted and ground truth boxes. A higher IOU means the predicted box is more likely to be an accurate prediction of the ground truth box. The class prediction is a one hot encoded vector with a length of as many classes the dataset has (C). At the point of it's release, YOLO was a significant breakthrough in the field of object detection as it was the fastest yet, with it being able to detect objects in real time. However, it did have a significant drawback which was due the fact that each cell could only predict one class. So if two objects belonging to different classes were located in the same cell, one would be ignored or not detected. It was also found to have trouble with localizing objects, while also struggling to detect small objects and objects at different scales.

YOLOv2

YOLOv2 or first introduced as YOLO9000 [17], was introduced by the same authors as YOLOv1, with the goal of solving some of the problems faced by YOLOv1. Some of the main changes included the use of anchor boxes, which in simple terms is using predefined or reference shapes for bounding boxes, which helps the model identify and detect objects of different sizes better. Another addition was the use of batch normalization, which is the process of normalizing the inputs to each layer, thereby stabilizing the training and also speeding it up.

YOLOv3

YOLOv3 [18] which was considered as an incremental improvement to the YOLO algorithm by the authors, was also an important milestone for YOLO. It introduced a few additions to the way the algorithm worked with the

hopes of improving its performance. One such addition or improvement was the use of a new architecture, the new architecture was made up of more layers and also included residual connections, which allowed for better feature extraction. Another important improvement was the use of a concept similar to a feature pyramid network [19], this enabled the model to learn to detect objects at various scales and sizes. The way in which the classes are predicted was also changed. Instead of the softmax classifier that was used previously, the algorithm now uses independent logistic classifiers which are used to decide whether an object is of that class or not. There would ideally be as many independent logistic classifiers as there are classes. The paper also specifies that another motivation behind this switch is to ensure that it can also be used for multi-label classifications i.e. where an object can fall under more than one class.

YOLOv4

YOLOv4 [20], which was the first version that wasn't released by the original authors, focused on improving the speed and accuracy of the YOLO algorithm. It added a more efficient and accurate backbone which was built based on the one in YOLOv3. This version also employed and integrated multiple feature extraction techniques, which improved the feature extraction further. There were other changes and improvements that were made as well in addition to the above.

YOLOv5

The YOLOv5 model, was developed by a company called Ultralytics and was not released as a paper. The company also released multiple versions of it based on the size of the network. Similar to the other versions, it builds on top of the strengths of its predecessor and makes changes that improve it overall. A main difference is the fact that it uses anchor free object detection, and is still more accurate. Overall, the algorithm seems to have improved in all aspects compared to its predecessors and is considered as a top choice for object detection in multiple fields.

Chapter 3

Methodology

3.1 Tools

In this section, I will specify and briefly explain the tools and libraries I used. While it may not be an exhaustive list, it will aim to cover the important ones at minimum.

3.1.1 Programming Language

The programming language I chose for the project was Python and the version I used was 3.9 [21]. The main reason I chose this language is because it was what I was most comfortable with. However, it is important to note that it is considered a good choice for a vast number of applications, including machine learning and deep learning tasks. This can be attributed to a number of things, the first and probably most common is the ease and simplicity associated with programming in Python. The large community of users that share knowledge, answer questions and clarify doubts of programmers of all skill levels is another reason why it is a popular choice. Python is also known for having an extensive collection of libraries, with libraries for almost any specific case you can think of. For example libraries like NumPy are useful for a range of mathematical operations; TensorFlow and PyTorch are both libraries that are used for deep learning along with possibly thousands more.

3.1.2 TensorFlow

TensorFlow is a deep learning and artificial intelligence library created by Google [22]. It is the library I chose to use for the purpose of this project. While there are other choices such as PyTorch, I chose to go with TensorFlow as I found it to be easier to understand.

TensorFlow is a good choice for an object detection task as it has a wide range of APIs for building as well as training models for various machine learning and deep learning tasks. TensorFlow's ability to be run on CPUs, GPUs and even TPUs (Tensor Processing Unit) is another reason I chose it (other options have these as well). The ability to utilize the GPU for computation is an especially useful feature, this is because a GPU is designed to perform many parallel computations and at very high speeds. This can significantly speed up the process of training a model, which involves multiple large computations. Similar to Python, TensorFlow has a large active community of developers and enthusiasts. This makes learning or even figuring out issues a lot easier due to the freely available support and resources users have access to.

For this project, I utilised many aspects and techniques provided by TensorFlow. The main use being, to develop the YOLOv1 model. For this purpose, I made use of the Model class from the TensorFlow Keras API. This provides an interface which enables the user to easily define the architecture of the model. To specify the different layers in the architecture, I utilised the layers class which has different layers such as convolutional, dense, max pooling etc. already defined and ready to use. To implement the loss functions, I used certain parts of the losses API offered as well. I also used a separate library which is built on TensorFlow known as TensorFlow probability [23]. The main purpose of this library is to perform probabilistic and statistical computations when working with deep learning models and otherwise.

3.1.3 OpenCV

OpenCV is an open source computer vision library that contains a number of functions and algorithms that are built with the goal of solving various computer vision problems and tasks [24]. It is favoured by both academics and in industry, which contributes to its development and improvement as a result of its large user base. The main reason for using OpenCV in this project was for reading the images from the dataset for training. I also used OpenCV for certain aspects of preprocessing one of the datasets I attempted to use, but didn't end up

using it in the end. While OpenCV offers many other functions that are useful, I found that it wasn't required as much in this case. However, as I will cover later on in this report, I did use it for preprocessing purposes when needed.

3.1.4 NumPy

NumPy is a library built for simplifying many forms of scientific computation when using Python. NumPy offers a multi dimensional array known as a N-dimensional array which makes representing and manipulating such objects easier. The library consists of multiple functions that simplify the process of working with multi dimensional arrays of numerical data.

The main use of NumPy in my project is due to its similarity to the tensors used in TensorFlow. TensorFlow tensors, unlike those in PyTorch does not allow for updating certain parts of a tensor. Therefore, I have converted tensors to NumPy arrays where it is required to update these tensors. Once I have completed the operations I needed, the NumPy arrays are converted back to tensors.

3.1.5 Matplotlib

Matplotlib is a python library with the sole purpose of creating various visualizations in Python. I use this library to visualize my results as well as to monitor various metrics during training.

3.2 Selecting an object detection algorithm

As I had never implemented or even attempted to implement an object detection model prior to this project, I spent some time trying to understand the differences between the object detection algorithms available. This also involved looking into the different ways I could get a working object detection model with decent accuracy as a final product. From what I found, I established that I could do so in one of two ways. The first is to develop and implement the model and all its accompanying features and functions from scratch. While the second is to download one of the many models available online and to fine tune it to suit my problem by training it on my dataset.

I decided to brave the challenge and went with developing a model from scratch. There were a few reasons which led me to choosing this approach even though I knew that fine tuning an existing pre trained model would likely yield better results and also take significantly less time to arrive at a working solution. The main reason was the fact that I had very little to no experience with such a task and concluded that by implementing a model from scratch I would be able to better understand the workings of object detection thereby giving me a better understanding of the project as a whole. In hindsight, this was probably not the best decision and I will get into more detail on why I think so, later on in this report. Another reason which led me to this decision, was the numerous tutorials and resources I came across during my research, which covered implementing these models from scratch. While there were many disclaimers that pointed out that this was not an easy task, it also gave the idea that they were not impossible to do so, given a bit of effort and time. In addition to all this, I also believed that as a Computer Science and AI student, the task of implementing such a model from scratch would be a valuable and worthwhile experience which would also broaden my knowledge. This section will cover the different models I attempted to implement, with the reason for there being more than one model is that I was unable to get anywhere in getting the models to work.

3.2.1 Faster-RCNN

The first model I decided to have a go at was the Faster-RCNN. Based on the reading I had done at this point, I decided that Faster-RCNN would be a good choice given that RCNN's were considered to be models that could achieve high accuracies with the Faster-RCNN being the most recent and best performing version of RCNN. After watching a few tutorials and getting an understanding of the way the algorithm works, I decided to begin implementing it. This experience was far from what I had imagined. I was not able to fully understand what was being done in the code or why. I attempted it a couple more times after going through a number of tutorials and reading more about the algorithm and was just not able to get it working or even figure out what was going wrong. After carefully analysing my situation at this point, I decided that this model wasn't working out for me and that I had sufficient time to attempt a different model/algorithm and abandoned the Faster-RCNN idea.

3.2.2 Single Shot Detector (SSD)

The next model I decided to implement was the SSD. In this case as well, I spent some time reading about the algorithm and tried to understand the concept behind it. After understanding the basics of the algorithm, I

watched a few tutorials and following these, I attempted to implement my own version. The implementation however was not very successful. As there are many different concepts involved with implementing a SSD, I had to go step by step for each concept to ensure I was doing it correctly. During the process I kept running into errors and ended up spending a lot of time trying to understand and fix them. After attempting it a couple of times, I had made very little progress and no where close to actually completing the implementation. At this point, as I was spending a lot of time trying to understand what was going wrong so while working on it I also started looking into the YOLO algorithms as it was the only one out of what I had researched and found suitable that I hadn't attempted. Shortly after this, I made the decision to stop attempting the SSD, as I was spending a lot of time trying to get it working and wasn't making any progress.

3.2.3 YOLO

When my attempt at implementing a SSD proved to be futile, I looked into the process of implementing a YOLO model instead. At this point, I think the attempts at both the faster-RCNN as well as the SSD had significantly improved my understanding and confidence in the use of TensorFlow. After reading a bit more on the workings of YOLO and getting a solid understanding of the original YOLO algorithm, I planned to first implement the YOLOv1 and thereafter implement one of the improved versions most probably YOLOv3 or YOLOv5.

While I did find it easier to follow the code, and also understand what was being done and for what reason, the final model I had seemed to be doing something incorrectly. The reason I was certain something was not working as expected was because the tutorial explained that testing the model by overfitting it on a very small dataset, was a good way of checking that the model was able to in fact learn. However, my model was showing no sign of this. Since this was the furthest I had gotten in implementing any model, I spent some time going over the code repeatedly and checking that my code was doing what I was expecting it to, as I did not have the time to implement another model from scratch. One main reason I faced a few issues is because the tutorial I followed, while extremely descriptive and easy to follow, was implemented in PyTorch and I had to therefore convert it to TensorFlow. The debugging process went on for some time and uncovered a few mistakes with the main one being a small mistake made in the loss function with regards to how the total loss was calculated.

As I was unable to get the model to a somewhat decent accuracy, I ended up spending too much time trying to fix it, leaving me with some serious time constraints. I had to therefore decide how I would proceed because the project required a model with an acceptable accuracy. It was due to this, that I decided I would aim at trying to get a decent result out of this model rather than trying to implement another one in case I would end up not having any working models.

3.3 Data Collection and Preparation

Over the course of the project, I went through approximately three datasets before finally settling on one, which is what I used for the YOLOv1 model. Selecting a dataset and preparing it was a more tedious task than I expected. As creating a general scene descriptor would require an extremely large dataset with a large number of classes, and training a model on this would require significant resources; the project will focus on implementing a descriptor specific to one area such as street view scenes. The idea is that by implementing a descriptor that works on specific type of scene, it can be used as proof of concept that it can be applied to a broader and more general scene.

3.3.1 Mapillary Dataset

The first dataset I decided to use was the Mapillary Vistas Dataset [25] [26], which was made up of street view images labelled with 60 classes. The dataset was originally created for semantic segmentation but also comes with labelled bounding boxes, hence the reason I chose it. One issue I had to deal with when using this dataset is that all the images were of varying sizes. This was an issue because object detection models such as Faster-RCNN, SSD and YOLO require the images to be of a uniform size. While this can be easily done by just resizing the images, there is also the problem of updating the bounding box coordinates as well.

Resizing the images

In addition to this there are a few things I had to consider when resizing the images. The first consideration I had to make was whether to simply just resize the images to one size using the resize function offered by OpenCV. The issue with this was, when increasing or decreasing the size to get it to the target size, the images would get stretched or distorted based on whether the image was smaller or larger than the target size. This could lead to a loss of detail in the images making it harder for the model to learn. The second consideration I had to make was what size I should resize the images to. The images in this dataset were of varying sizes,

some extremely large and some small. I needed to find a size that would not stretch the small images or shrink the large ones too much. As this was something that I figured I would need to experiment with, I selected a few sizes and created different datasets which were of different sizes, allowing me to experiment at the point of training. I did not consider however the limitations I would have in terms of computation power. I learned later on that I couldn't have images that were too large as it would take a long time to process when training. In the end, as a result of this a few of the sizes I chose had to be discarded.

My first approach to resizing the images was to do so while preserving its aspect ratio and also keeping it within the target dimensions. I did this by calculating the scaling factor and then resizing the image based on that. Since many images resized in this manner ended up being smaller than the target dimension, I decided to use padding to get the images up to the required size. To do this, I used the resize and copyMakeBorder functions offered by openCV to resize and add padding respectively. When resizing the image, I had to choose a suitable interpolation method that would be used to calculate the new pixel values for the resized image. Out of the options offered by OpenCV, I chose to go with the cubic interpolation as it seemed like the best option and the one that would result in the least loss of detail.

The resulting dataset was what I used when attempting to train the faster R-CNN, but as mentioned before I was never able to fully implement it. My attempt at a SSD never reached the point of using the data so there were no changes made for it. When I implemented my YOLOv1 model, at first I used this and as it was the first time I was able to actually get the model to train I could see that something was not right. Since I wasn't sure what the issue was at the time, I decided to experiment further with the dataset as there were some images that were shrunk quite a bit in order to maintain the aspect ratio. In order to deal with this issue, I decided to try another method of resizing the images with the hope that it would improve the performance of the model and be able to learn better. The approach I took was center cropping. The reason I chose center cropping and not some other form of cropping, is due to the fact that the images were all different in terms of where the objects were. Therefore by cropping from the center, I would be able to get at least a few of the objects from the original image in the event that objects get cropped out. To actually implement the center cropping, I first calculated the scaling factor and resized the image to the size closest to the target size that maintained the aspect ratio. Once I had this resized image, based on the new size I calculated how much needed to be cropped to get it to the target size. In this instance as well, I used the resize function from OpenCV to resize the image and then used slicing to get the required part of the image. The reason I use slicing is because OpenCV stores the image as a NumPy array and by slicing it I am able to get just the part of the image that I want.

Preprocessing the annotations

Now that I've covered the preprocessing of the images in the Mapillary dataset, I will now explain how I preprocessed the bounding box coordinates to reflect the changes I made to the images. Before I get into that I will explain how I got only the information I needed by getting rid of the other information provided that I wouldn't be using. The dataset provided all bounding box and other information in the form of a JSON file. Using the JSON library I was able to extract the information and store it in a dictionary with each image file name as the key, and it's corresponding data as the value. Once I had the dictionary, I simply created another dictionary and looped through the original dictionary, adding only what I needed from each key value pair to the new dictionary. To update the bounding box coordinates, I included a section that did so when resizing the images. So it would first resize the image, and then update the relevant entry in the dictionary based on the changes made to the image. The Mapillary dataset has bounding box coordinates in the form [xmin, ymin, width, height] where as Faster-RCNN, SSD and YOLOv1 require the coordinates to be in the form [xCenter, yCenter, width, height] therefore, I had to calculate the center coordinates as well.

3.3.2 Udacity Dataset

While most of the work I did with relation to preprocessing data was on the Mapillary dataset, I had to switch to a different one when I got the YOLOv1 model working. The main reason for this was because of the large number of classes in the Mapillary dataset. Since I was still trying to get the model to a decent point, I made the decision to switch to a dataset with fewer classes. After a bit of searching, I decided to go with the version of the Udacity Self Driving Car Dataset found on Roboflow [27] [28]. The reason I used the Roboflow version and not the one on the Udacity GitHub is because, the Roboflow version has a resized version ready to download and also because this version has added labels for many objects that were missed in the original dataset. Since I was able to download a dataset that was ready to use, with the annotations available to be downloaded in many formats including one for YOLO, there was very little I had to do in order to use this dataset. This included resizing as the site offered already resized images which I downloaded. The dataset has the different variations of traffic lights as different classes, but since I didn't want to confuse the model when loading the data, I combined all traffic light related classes into one class. This reduced the number of classes from 11 to 5, making it easier for the model to learn as well.

Another aspect related to the preparation of the data is the creation of the label matrix. This concept is related only to the YOLOv1 model. The label matrix is basically how YOLO stores the ground truth annotations. As mentioned previously, the YOLO model works by dividing the input image into a grid like structure. In addition, since each cell is able to only predict one class (limitation of YOLOv1), the label matrix for the ground truths will have only one ground truth box associated to each cell. This is regardless of whether there are other boxes that fall within that cell or not. Basically, if the label matrix has already added a box to the cell and then comes across another ground truth box that falls within that cell, only the first box is considered and any boxes that come after that are disregarded. While I did use this approach, I also tried another approach. The approach I took was, if there is a box that falls within a cell that has already been assigned a box, the area of the 2 boxes (the one already in the matrix and the one that is being considered at this point) are calculated and if the new box has a larger area than that of the box already in the matrix, then this box is replaced with the new box and it's associated values.

3.4 Performance Metrics

This section will cover the metrics I used to evaluate the performance of my object detection model. Since I was unable to implement a text generator that would be used to convey the results of the model to the user, I will not cover any performance metrics for that aspect of the project.

The main metric I used throughout this project to assess the performance of my object detection model is, the mean average precision metric or more commonly referred to as mAP. mAP is a widely used metric as it gives an overview of the model's accuracy as a whole when used on a labelled set of data. The reason it needs a labelled dataset to calculate the mAP is because the calculation involves comparing the predictions to the ground truths. In order to fully understand the mAP metric, there are a few concepts that need to be understood first.

The first is the idea of true positives, false positives and false negatives. True positives (TP) refer to predictions that correctly identify and detect an object in an image. False positives (FP) are predictions that incorrectly detect an object in an image when it doesn't actually exist. Finally, false negatives (FN) refer to when the model completely fails to detect an object that is present in the image.

The reason we needed to understand those concepts is because in order to calculate the 2 metrics that lead to the mAP are calculated using true positives, false positives and false negatives. Precision, which is the proportion of detections that were true positives out of all positive detections and is calculated by $\frac{TP}{TP+FP}$ Similarly, the recall which measures the proportion of true positives out of all actual positives is calculated by $\frac{TP}{TP+FN}$.

A detection is classified as either a TP or FP, based on the IOU between the predictions and the ground truths. In order to achieve this, there needs to be a threshold that specifies how high of an IOU is considered a true positive and thereby how low it would need to be for it to be considered a false positive.

Another important concept involved with calculating the mAP is the precision recall curve. This is a curve which, as the name suggests plots the precision against the recall. For multi class problems such as this project, the precision recall curve is calculated for each class. In addition, this is done for different confidence levels and not just one. However, as the calculation takes some time, I calculate it for only one confidence level.

Once the precision recall curve has been plotted, we can then calculate the average precision (AP) which is done by calculating the area under the curve. For this project, in order to compute the area under the curve I used the math.trapz function from the TensorFlow probability library, which does exactly that. Once the average precision has been calculated for each class, the mean is taken, giving the mean average precision or mAP of the model. This then gives us a single value that represents the model's ability to predict objects across all the classes.

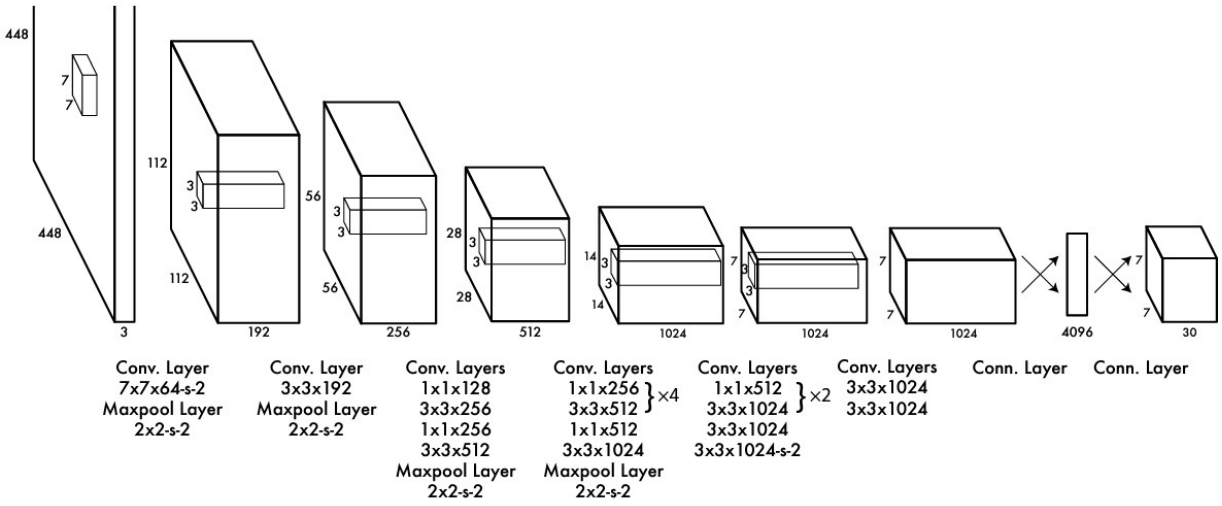
Chapter 4

Design and Implementation

This section will cover the process of designing and implementing the object detection model. As mentioned previously in the report, as a result of the challenges faced while attempting to implement a working object detection model, the project will not deliver all aspects related to implementing a scene descriptor for the visually impaired. The components that the system was meant to have include, a working object detection model with satisfactory accuracy, a system that could convert the predictions of the object detection model into verbal descriptions, and maybe a user interface. However, as a result of struggling to implement a working object detection model and thereby spending a lot of time trying to first implement it and then get it to have an acceptable accuracy which I still haven't been able to achieve, this project will only deliver a object detection model. As I explained the difficulties I faced in implementing the first two models I chose to implement, I will not discuss them here any further as I didn't get any working solutions out of them. After my first two attempts failed, I chose to go for the simplest out of the remaining algorithms I had found to be suitable, which was the YOLOv1 model. The reasoning behind this was that I would first get some sort of model working and thereafter focus on implementing a better one. However, as I struggled with that as well for quite some time, I was unable to tackle another model as well. Therefore, this section will mainly cover the implementation of the YOLOv1 model I used. Before getting into it, I would like to mention that my code is based off a tutorial I watched which implemented a YOLOv1 model from scratch in PyTorch [29] [30]. Since I had used TensorFlow for my attempts at the other two models, I chose to convert this to TensorFlow as I had got a foundation in TensorFlow felt more comfortable with it than PyTorch which I hadn't used before.

4.1 System Architecture

The architecture I used for my implementation closely follows the original YOLOv1 paper [16] (seen below) with some adjustments made to account for computational power required.



4.1.1 Input Layer

As I mentioned previously, when I first implemented the YOLOv1 algorithm I used the same dataset I used for the other two models (even though I never actually used them). These images were very large before I resized them so when I used it for the YOLOv1 model, I initially resized them to 704x704 and then later on to 640x640. The reason I changed sizes was because I was having trouble running the code and kept getting memory issues, so I decided to reduce the size to see if that would help. This was all in vain though, as I decided to use a smaller dataset with fewer classes to try and reduce the load my program would have, which was when I started using the Udacity dataset. The images in the dataset I downloaded were already resized for convenience, leaving me with little work on that end. The images were of the size 512x512 and therefore the input layer accepts a shape of (512, 512, 3) where the first two correspond to the size of the image and the third value depicts the number of channels which is red, green and blue.

4.1.2 Convolutional Layers

The YOLOv1 architecture, on which my implementation is based has 24 convolutional layers. A convolutional layer works by sliding a filter or kernel over the input image and calculating the dot product between the kernel and the region of the image it is on [31]. The amount by which the kernel moves is specified by the stride. The output of the layer would then be a feature map made up of the values output by the dot product of the kernel and each region it passes over. Based on the size of the kernel and also the stride, the feature map would usually have lower spatial dimensions which is the width and height, than the input image. Similarly, each layer can have a different number of filters, which will learn to detect different features in the input image. Therefore, the feature map will have as many channels as there are filters, which is essentially the depth of the feature map. The convolutional layer can also specify padding to be applied to the input image before the kernel is slid over the image. This adds pixel values to the edges of the image which ensures that the feature map has the same spatial dimensions as the input, and if no padding is specified the feature map that is output will have smaller spatial dimensions.

The architecture I used for my model, specifies different numbers of filters, kernel sizes, strides and padding for the different layers in the model. This is done in a way that reduces the spatial dimensions of the feature map as it moves through the network while also increasing its depth. By doing so, the model is able to learn more low level features early on in the network and more high level features as it moves through the layers. By doing so the model learns to perform object detection based on the features from the images it is trained on.

The architecture also includes a number of max pooling layers placed at certain points. The purpose of these layers are to reduce the spatial dimensions of the feature maps while retaining the most important information. It does this in a similar manner to the convolutional layer, which is by sliding a filter or window over the input, but rather than performing element wise multiplication, it takes the maximum value within the region on which the filter is. By doing so, it also helps the model to not overfit to the data it is trained on, because by performing max pooling the model retains the more important features which can help it generalize to future examples and unseen data.

4.1.3 Leaky ReLu Activation Function

Another important component of the convolutional layers described above is the activation function. The activation function serves the purpose of introducing non-linearity to the model, which enables it to learn complex pattern. The YOLOv1 architecture uses a Leaky ReLU (Rectified Linear Unit) activation function [32] in all it's convolutional layers. Leaky ReLU is a modified version of the ReLU activation function with main difference being its attempt to solve the dying gradient problem. The dying gradient problem is introduced when the output of the ReLU function is zero as a result of the input being negative. This results in the gradient also being zero, leading to the neuron becoming inactive and therefore not learning. The leaky ReLU function tries to solve this by introducing a small non-zero constant when the input is negative rather than setting it to zero as was the case in ReLU. This allows the neuron to remain active and keep learning even when the input is negative.

4.1.4 Fully Connected Layers

Both the original YOLOv1 architecture as well as my implementation, use two fully connected layers (dense layers) after the convolutional layers. Every neuron in a fully connected layer as per the name, is connected to every neuron in the previous layer. So the output of the neurons in the layer before it, will be used as input for each neuron in the fully connected layer. The layer then applies the weights it learns during training along with the activation function (if any) to these inputs, which is the output of the layer. These layers basically take the features learned by the convolutional layers and use them to make predictions regarding the class and bounding box coordinates.

The first fully connected layer in the original architecture had 4096 neurons in it. However, as this increases the computation required, I experimented with it and in most cases reduced it to 496 (which was recommended in the tutorial I followed). Since this layer would be connected to the last convolutional layer, it takes the feature map output by the convolutional layer and learns what features are most important. The activation function used in the first fully connected layer is also the Leaky ReLU function.

Additionally, after the first fully connected layer, the original architecture uses a dropout layer with a value of 0.5. My implementation uses a dropout of 0.5 as well. This means that the network randomly switches off or drops out 50 percent of the neurons. By doing this, it ensures that the model does not get dependent on any one neuron, which increases the models ability to generalize and also prevent overfitting.

The next fully connected layer which is also the last, has as many neurons as the output size which is calculated by S * S * (B * 5 + C). These values correspond to the values I mentioned previously in this report when explaining YOLO. For ease, S refers to the grid size i.e. how many cells the image is divided into, B is the number of bounding boxes the model will predict for each cell and C is the number of classes. The reason for the output shape to be S * S * (B * 5 + C) is because it outputs the values it predicts for each cell hence the S * S, and predicts B bounding boxes which each have 5 values (4 coordinates and score) and the C is for the one hot encoded class representation. My implementation like the original uses S=7 and B=2, which is to say the image is divided into a 7x7 grid and each cell will predict 2 boxes. The value I use for C however, differs from the original as a result of my dataset having only 5 classes compared to the 20 classes that made up the dataset used in the original paper. This gives me an output size of 735 and thereby 735 neurons in the final fully connected layer. This layer is is what makes the final prediction and does so by using the values from the previous fully connected layer after the dropout has been applied. This layer also uses a linear activation function which means it does not manipulate the input it gets but rather uses it to make the final predictions.

The final output of the model is the output of the last fully connected layer, reshaped to be (S, S, (B*5+C)). This is so that working with the predictions is easier, as this gives a three dimensional tensor rather than a one dimensional tensor, making it easier to access the relevant information when required.

4.1.5 Loss Function

$$
\begin{aligned}
&\lambda_{\textbf{coord}} \sum_{i=0}^{S^2} \sum_{j=0}^{B} \mathbb{1}_{ij}^{\text{obj}} \left[(x_i - \hat{x}_i)^2 + (y_i - \hat{y}_i)^2 \right] \\
&+ \lambda_{\textbf{coord}} \sum_{i=0}^{S^2} \sum_{j=0}^{B} \mathbb{1}_{ij}^{\text{obj}} \left[\left(\sqrt{w_i} - \sqrt{\hat{w}_i} \right)^2 + \left(\sqrt{h_i} - \sqrt{\hat{h}_i} \right)^2 \right] \\
&+ \sum_{i=0}^{S^2} \sum_{j=0}^{B} \mathbb{1}_{ij}^{\text{obj}} \left(C_i - \hat{C}_i \right)^2 \\
&+ \lambda_{\text{noobj}} \sum_{i=0}^{S^2} \sum_{j=0}^{B} \mathbb{1}_{ij}^{\text{noobj}} \left(C_i - \hat{C}_i \right)^2 \\
&+ \sum_{i=0}^{S^2} \mathbb{1}_{i}^{\text{obj}} \sum_{c \in \text{classes}} \left(p_i(c) - \hat{p}_i(c) \right)^2
\end{aligned}
$$

This section will cover the loss function I used, which is based on the loss function specified in the original YOLOv1 paper [16] (seen above). The loss function in any neural network is used to evaluate the accuracy of the predictions made by the model. This is done by comparing the prediction with the ground truths. The aim of the neural network will be to minimize the loss function, which means the lower the loss value the better the predictions. However,it is important to note that there can be cases where the loss becomes too low. This generally indicates that the model is overfitting to the data it is being trained and losing it's ability to generalize to new and unseen data. Due to this, is important to also calculate the loss on another set of data, generally

the validation data which the model will not learn from. By monitoring the validation loss during training, we can generally get an idea if the model is overfitting as the loss on the validation data will start to increase. The purpose of the loss function is vital to the process of the model learning. This is because, the loss calculated by the loss function, is what is used to decide by how much to update the weights and biases in the network using gradient descent.

The YOLOv1 loss function is made up of multiple parts, which is related to the different predictions it makes. It consists of the coordinate loss, the confidence losses and the class loss. All parts of the loss function use Mean Squared Error (MSE) to calculate the loss, which is basically taking the square root of the difference between the predictions and the ground truth. In my implementation I use the MSE function offered by TensorFlow for calculating the errors.

The coordinate loss in the function measures the error between the true bounding box coordinates and those that were predicted by the model. Since the YOLOv1 model predicts two bounding boxes for each cell, we first need to take whichever of the boxes that has the higher IOU with the ground truth box. This is the box that will then be considered. Once this box has been selected, we then calculate the MSE between the chosen box and the ground truth box. The YOLOv1 paper does however make a few additions to this. The first is that when calculating the MSE between the coordinates, it takes the square root of the width and height for both the predicted and the ground truth boxes. This is so that a larger box which will have larger width and height values doesn't have a larger impact than one which was smaller. So by doing this, it reduces the impact the size of the box has on the prediction. Similarly, the way the YOLOv1 loss function is calculated in a way that all components have the same weight in the total loss. Since this is not desirable, the authors introduced a constant λ_{coord} (5 in the paper), by which the coordinate loss was multiplied, giving it a higher weight in the total loss.

The confidence loss measures the error between the predicted confidence scores and that of the ground truth. The confidence score relates to the how certain the model is that there exists an object. Therefore, the scores for the ground truth will be one or zero, based on whether or not there is an object in that cell. The confidence loss is made up of two parts, object loss and no object loss. The object loss relates to when there is an object in that cell, and is calculated by taking the MSE between the score of the predicted box which has the higher IOU with the ground truth box and the score of the ground truth box which is generally one. Similarly, when there isn't an object in the cell, the loss is calculated for both boxes, between the scores predicted for the two boxes and the ground truth box's score which is generally zero as a result of there being no object present. Since there are likely to be more cells that do not contain objects than ones that do, the authors introduce another constant λ_{noobj} (0.5 in the paper), which reduces the impact the no object loss has on the total loss.

The final part of the loss is the class loss, which measures the error between the predicted class probability and that of the ground truth which will have a probability of 1 for the true class and zero for all the others. The total loss is then the sum of all of these parts with the λ_{coord} and λ_{noobj} being applied to the relevant parts.

Chapter 5

Experimental Results

This chapter will go over the different experiments I conducted in my search to find the ideal configuration for my YOLOv1 model. As I was able to get my model working only very late in the project, the experiments conducted while informative were possibly insufficient as a result of the time constraint I was facing.

5.1 Setup and Limitations

For the purpose of conducting these experiments and training and evaluating the model, I used the N109 labs mostly by remote connection but sometimes in person as well. The machines I used run a Linux OS with 32 GB RAM and a NVIDIA GeForce RTX 2080 Ti graphics card. For majority of the training and experimentation I conducted, I used TensorFlow with GPU support. However, for the initial steps I was using the CPU only as a result of some challenges I faced with installing the required drivers etc. for using TensorFlow with GPU support. While the addition of the GPU did speed up the training process a fair bit, it was still fairly time consuming.

I also did have a few issues which hindered the process slightly. These were mainly related to the use of the lab machines which did cause a few problems from time to time. This included a few times where the machine I was using froze mid training and crashed and in one instance was unable to get it back up. There were also a few issues where the machines would randomly disconnect while I was working on them as well. While these issues are to be expected, I thought I might just briefly mention them as well.

There were also other limitations that I was exposed to during training and configuring the model. The main one being issues related to memory. There were a few things I was unable to try due to the limited resources the machine has, and would result in a resource exhausted error. The first being that I was unable to try any batch size over 32 without getting such an error. This limited my ability to fully experiment with the impact a larger batch size would have, especially considering the fact that the original paper used a batch size of 64. Similarly, I was also limited to the number of neurons I could have in my first fully connected layer. Whenever I tried to have more than 1024 neurons in the fully connected layer, I would get the resource exhausted error. Once again this proved to be unfortunate since the fully connected layer in the original paper used 4096 neurons in it. Since the more neurons there are, the more it is able to learn, this limitation too proved to be unfortunate as it limited my ability to observe the impact it would have.

5.2 Experiments and changes tried

Once I was able to confirm that my model was working i.e. learning from the training data, I conducted a number of different training loops in order to try and find the best configuration for the model that would produce a satisfactory accuracy. Throughout my experiments I used mAP as the main evaluation metric, with an IOU threshold of 0.5 and a probability threshold of 0.4. An important experiment I should cover, is the way I checked and confirmed that my model was in fact learning. To do this, I tried to purposely overfit the model on some training data. After training the model on 200 images for 100 epochs, the mAP for the training data was 0.5 while the mAP on the validation data was 0.2. This difference in mAP scores for the training and validation data showed that the model had in fact overfit to the training data and was thereby learning for sure. However, when I tested the model on the images I used for training the predictions were far from satisfactory.

After this, I ran through as many training loops as time permitted making changes to the different hyper parameters, number of epochs, learning rates etc. The approach I took was to try a configuration and then based on it's performance either continue training the model or start training a new model. Since I was stuck for time, I was unable to test on a large dataset as this would not allow me to try very many different things.

5.2.1 Results

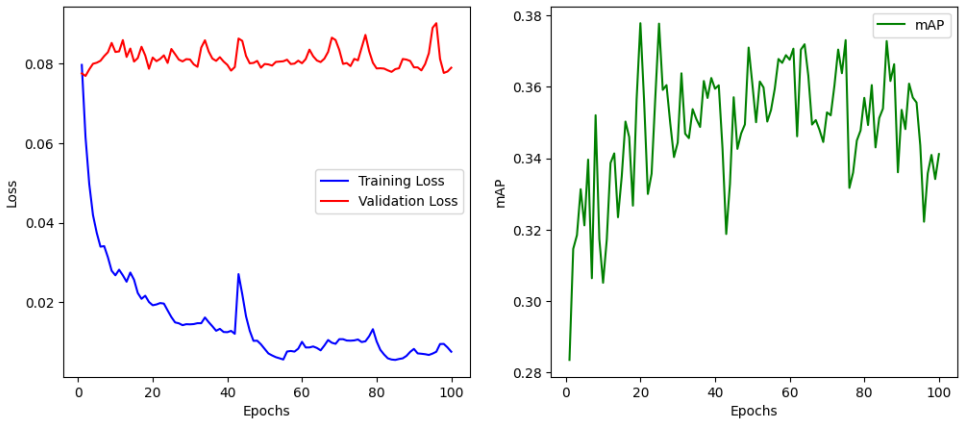
The next experiment I conducted was to use a dataset of 200 images again and increased the batch size to 32. In addition, I used a dropout of 0.5 after the first fully connected layer which had 496 neurons in it. I used the Adam optimizer with a learning rate of $2e-4$ and ran the training loop for a 100 epochs. After the 100 epochs, the mAP was at approximately 0.12 on the validation set. Over the 100 epochs, the mAP peaked at about 0.2 in the first few epochs and then dropped to 0.1 where it was pretty much stuck the rest of the time. The training loss also reduced and flattened out at about 0.2 after the first few epochs. What I observed from the predictions the model made on images in the validation set was that it was predicting 3 bounding boxes it thought to be cars in pretty much the same locations in all images. It did this on the train dataset images as well.

As I was unsure what was causing the behaviour and failure of the previous model, in the next experiment I decided to try a different optimizer to see if it would have an impact. This experiment closely followed what was done in the original paper. I used the Stochastic Gradient Descent (SGD) optimizer with a learning rate of $1e-2$, weight decay of $5e-4$ and a momentum of 0.9 which is what the authors used in their paper. The results this produced were very similar to that of the previous one, where the mAP got stuck at one point and remained there throughout the training. The loss showed similar behaviour, with it flattening out after the first few epochs and not changing thereafter. This too resulted in very poor predictions and was only predicting cars again. This allowed me to ascertain that the issue was very likely to be as a result of insufficient data for it to train on and learn from.

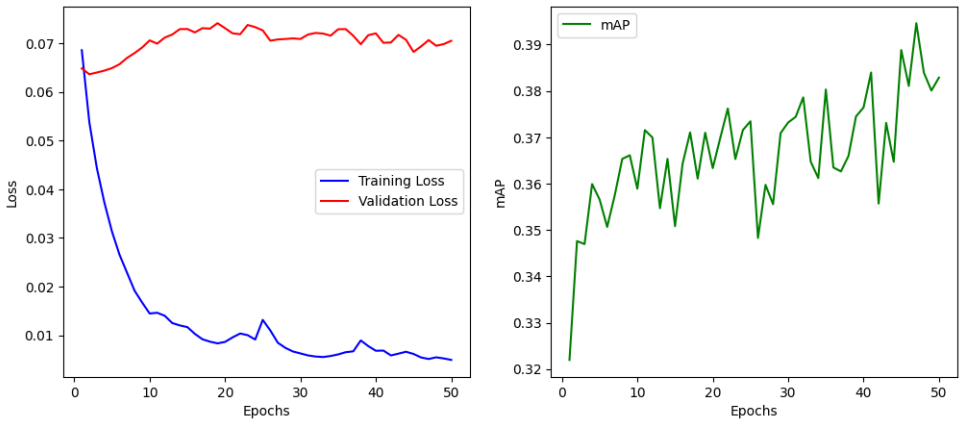
To deal with this, I then increased the size of the dataset and ran a few more experiments. I won't explain them all as they were mainly to test the impact the size of the dataset had on the models accuracy. I also came to the conclusion that the Adam optimizer seemed to produce better results and therefore switched back to Adam and continued to use this thereafter.

I increased the size of the dataset to 2000 and carried out more attempts at trying to find the optimal configuration of the different parameters. After training the model for a 100 epochs with the batch size set to 16 and the learning rate at $2e-5$, the model finally showed signs of learning. I also used a dropout of 0.5 after the first fully connected layer and had a weight decay of $5e-4$ for the optimizer. The training loss was decreasing as expected throughout the 100 epochs, while the mAP was increasing steadily during the same time. After the training the mAP was at 0.3 and by observing the mAP throughout the training it was clear that the model was still learning.

Since the model that resulted from the experiment mentioned above showed good signs, I continued training it for another 100 epochs but this time I used a different set of 2000 images. This approach of taking only small batches of the training data at a time was my way of dealing with the inability to train on large amounts of data due to both the time constraints as well as the resource constraints. This experiment resulted in a final mAP of around 0.34. However, it was fluctuating a bit with it even touching about 0.37 initially as seen in the graph below. The mAP on the test set was 0.37, indicating that the model wasn't overfitting. The predictions however were still very unsatisfactory.

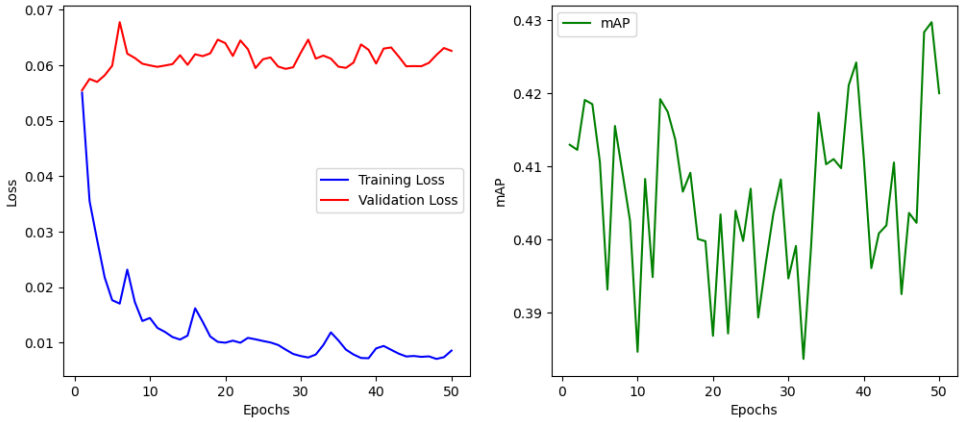


Since this model was showing promise and seemed to be improving, I trained it on a further 2000 images for 50 epochs. Since the model seemed to have learnt something, I reduced the learning rate slightly to $1e-5$ so that it wouldn't lose what it had already learned. The mAP on the validation set was 0.38 after the 50 epochs and 0.32 on the test set. While the mAP has gone down, it is important to factor in that this was on a new set of images. This can be seen by the way the training loss decreased over the 50 epochs and even from the fact that the mAP did in fact climb throughout the 50 epochs as seen in the graph below.

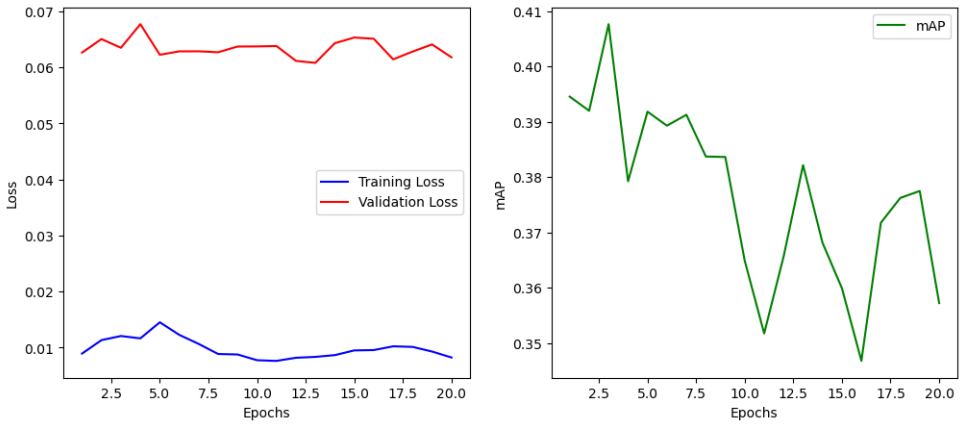


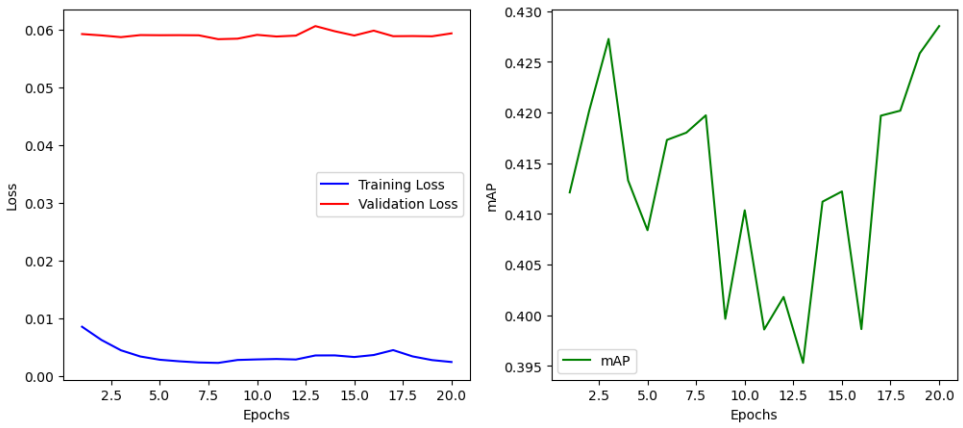
The somewhat satisfactory results, pushed me to continue training this model even further with the hope that I could improve the accuracy even if it was slow. After training it for a further 41 epochs (training cut short due to machines being switched off in lab), the mAP on the validation set was 0.367 and 0.33 on the test set.

Since that didn't seem to have too much of an impact, I resumed training the model from the previous experiment as it had the best mAP at this point. I trained it for 50 epochs on a new batch of 2000 images with the learning rate at $2e-5$. The results were promising as the mAP had risen to 0.42 on the validation set and even got a mAP of 0.47 on the test set. The increase can be seen in the graph below. It is important to note, that at this point even though the model is being trained on batches of 2000 images, it has been trained on around 6000 images at this point, which could be the reason for the better mAP results. Despite the higher mAP the predictions were still very poor.

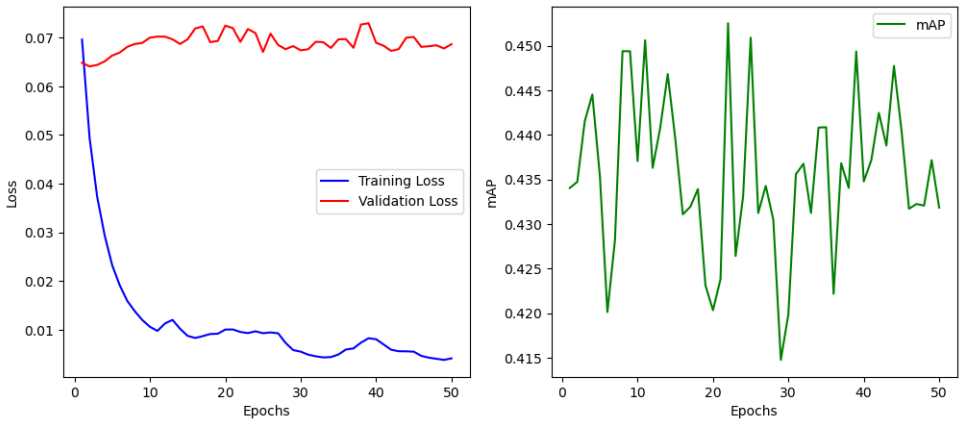


Since the mAP in the previous experiment was fluctuating quite a bit, I decided to see if I could address this by playing with the learning rate. I trained the model for 20 epochs first with a higher learning rate of $3e-5$, and then a lower one of $1e-5$. As seen in the graph below, the higher learning rate resulted in the mAP on both the validation and test sets reducing. I gathered from this that the model was now overshooting the optimal weights and thereby causing the model to become worse. The lower learning rate however, had a mAP of about 0.43 on the validation set and a significant improvement to 0.5 on the test set. While this was the highest mAP achieved on any set at this point, the predictions were still poor with the localization of the objects proving to be the main issue. The graphs below show the behaviour of both the losses and the mAP for both these cases.



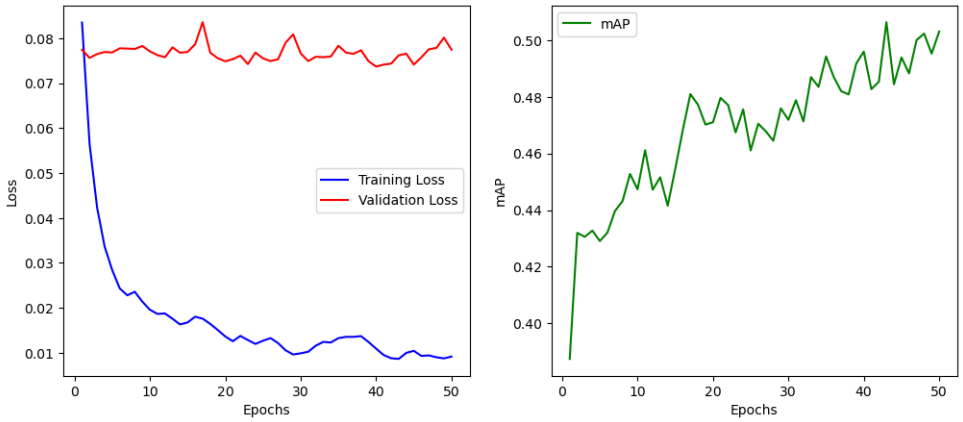


I then carried on training the better model from above, as it seemed to still be improving. I increased the number of epochs to 50 but left the learning rate as is at $1e-5$. Since the model seems to be struggling with the localization I increased the value of λ_{coord} from 5 to 6, to see if it would have an impact. The results were still not that great even though some of the predictions seemed to be accurate, but only a very few. The mAP on the validation set was 0.43 after the 50 epochs and the mAP on the test set was 0.36. The graph can be seen below.



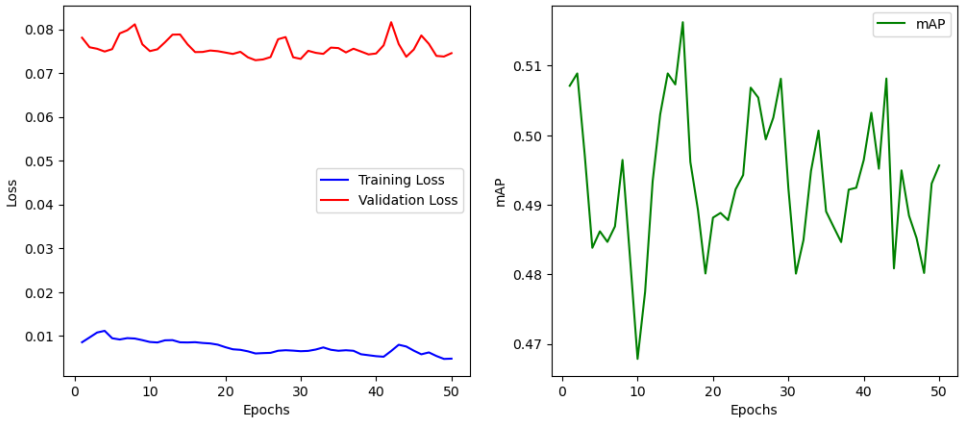
I did a few more similar experiments which I won't cover as they were mainly to try and improve the localization by playing around with λ_{coord}.

The next experiment I will discuss is again where I took the model with the best mAP I had got so far and trained it on a larger set of 5000 images. I used a batch size of 32, a learning rate of $2e-5$ and increased both λ_{coord} and λ_{noobj} to 7 and 0.7 respectively. This experiment proved to be fruitful with the mAP on the validation set ending at 0.5 with the best checkpoint being 0.51. The mAP on the test set was not too bad either at 0.45. While it is visible from the graph below that the model was learning and improving, the predictions were still quite poor and not at the standard I was looking for.

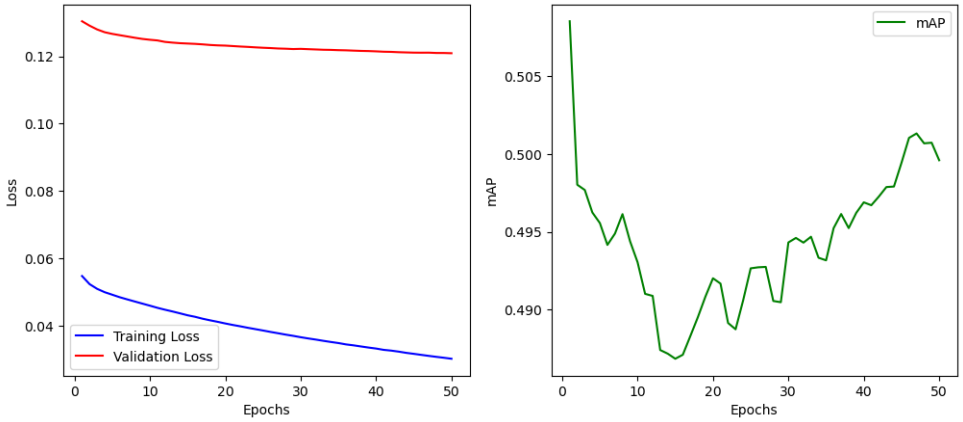


Since the mAP seemed like it was still increasing and the loss still decreasing, I trained the model for a further 50 epochs as is. The results were not too bad, with the mAP on the validation set being 0.49 and 0.49 on the test set as well. I also wanted to see exactly how good the model was at this point so I also calculated the mAP with a IOU threshold of 0.5 and a probability threshold of 0.8. The result was a mAP of 0.32, which was a fairly big drop, meaning that the model wasn't very confident about what it was predicting. The predictions were still very poor and were mainly due to incorrect localization. The graph related to this experiment is seen

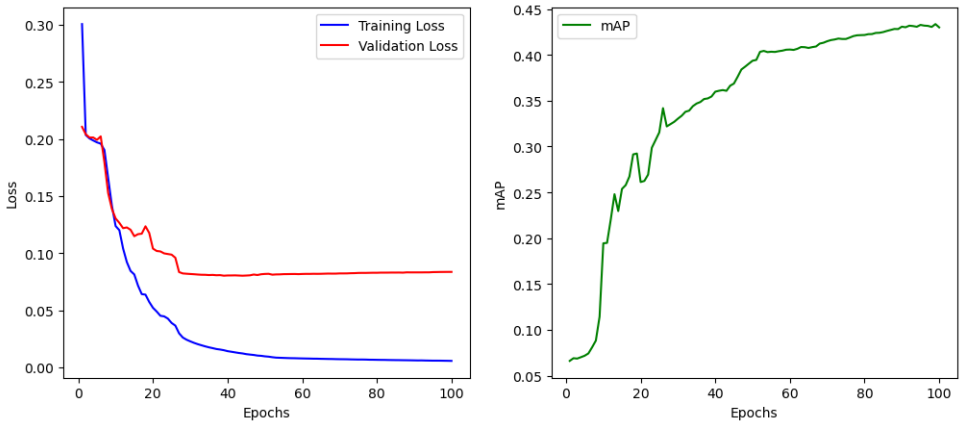
below.



In an effort to try and reduce the localization error, I trained the model further after increasing λ_{coord} to 10. I also reduced the learning rate to $2e-7$ so that the model wouldn't lose what it had already learned. The dataset now had 6000 images as a result of adding another 1000 images to the dataset that was being used. At the end of the 50 epochs, the loss seemed to still be decreasing and the mAP increasing. The mAP was 0.49 on the validation set and 0.48 on the test set. When calculated for the test set with a higher probability threshold of 0.8, the mAP was 0.3, so the model still wasn't very confident with its predictions. The graph is seen below. As the accuracy seemed to be on the rise, I trained it further for 20 more epochs but there was little change.



The final experiment I was able to run, involved using a much larger dataset which consisted of 10,000 images. For this experiment, I used a batch size of 32 and set λ_{coord} and λ_{noobj} to 8 and 0.5 respectively. For the optimizer (Adam), I used a learning rate scheduler where I decreased the learning rate every 25 epochs. While this is the main experiment that I'm covering where a scheduler was used, I did try a few others initially with a scheduler as well. The other addition I made to the experiment was that I increased the number of neurons to 1024 in the fully connected layer. The results were quite similar to that of previous experiments with mAP of 0.43 on the validation set and 0.39 on the test set. When calculated with a higher probability threshold of 0.8, the mAP was 0.28. The graphs for this are seen below.



I trained this model further, for 20 epochs more to see if I could get the accuracy up but had no luck and was unable to conduct any further experiments as I had run out of time.

In summary, all the experiments I conducted in trying to find the optimal configuration for this model,

resulted in poor results. Based on my observations, the main issue seemed to be that the model was struggling to accurately predict the bounding box coordinates. The reason I say this is because in most images, the model seems to be predicting the correct classes but the boxes are in the wrong places. It is also important to note, that while I was able to get the model to learn something from the data, I achieved this fairly late in the project and therefore had very little time to experiment, which from what I've gathered is an important part of building neural networks.

Chapter 6

Conclusion

In this chapter, I will very briefly cover the entirety of the project and any important points that I think need to be pointed out.

6.1 Summary of project

This project involves the development and implementation of a scene descriptor for the visually impaired or at least a starting point for it. The goal was to improve the ability of those with visual impairments to get an understanding about the things around them without having to rely on other people. My approach to this was to implement an object detection model which could be used to detect the objects in an image or scene and then convey this to the user in a medium they could understand. However, as I struggled greatly with implementing the object detection model, I ended up spending majority of the time I had for this project trying to get a working model. This involved 2 failed attempts at a RCNN model as well as a SSD model. I was finally able to get a YOLOv1 model working to an extent, but no where close to the accuracy required for a project of this nature.

6.2 Summary of methodology

The process of collecting and preparing the data for the purpose of training a model was also a fairly large component of the project. This is owed to the importance the data used, plays in implementing a successful model. The implementation of the actual models, in the case of the three different object detection algorithms I attempted was also a huge part and as mentioned previously took up most of my allocated time. There were many other components that were instrumental, such as the loss function and the many helper functions used throughout.

6.3 Summary of experimental results

The experimentation I did was an important part as well. It helped me to fine tune the model as best I could which in the end was insufficient. However, it provided useful insights which can be used to develop this idea further beyond it being just a project. The main finding was that the YOLOv1 model struggles with localizing objects it detects. While this particular model is known to have trouble with the part of object detection I believe it is still able to get better results than what I was able to achieve. With more training and time, the model implemented here is likely to be able to achieve higher accuracy and better results.

6.4 Limitations and Challenges

As the whole concept of implementing an object detection model was something I started from scratch for this project, there were significant challenges I faced. While I knew the theory behind what I was meant to do, implementing it was a whole different story as I found out. Some of the challenges I faced included the use of TensorFlow at a level I had not been exposed to previously. This required a significant amount of time on my part spent reading about and following tutorials with the aim of mastering its use. Similarly, the implementation of the models for which the concepts can be grasped without too much effort proved to be a lot more challenging than expected. I also struggled a fair bit with the process of identifying and using suitable datasets, which led me to starting over with new datasets more than once.

In terms of limitations, the main one would be the lack of sufficient resources on my own machine. This required me to rely on the machines in the lab which at times were a bit unreliable and also unavailable.

6.5 Future Work

While I would ideally have preferred to have a working solution at the end of the project, this was unfortunately not something I was able to achieve. Additionally, the work I have done is also far from sufficient to be a building block on which an entire system can be built. However, the project as a whole has had a tremendous impact on the knowledge I have gained throughout my degree. Every aspect of the project has in someway improved my knowledge or ability in some area or the other.

Even though I struggled throughout the project, it is an experience I am truly grateful for as it piqued my interest in the fields of artificial intelligence and computer vision even further. I do plan on using what I have done over the course of this project as a foundation on which I will continue to develop and improve this object detection model as a side project of mine. Part of the reason I know I will continue it is because, I have learnt that the process while challenging is something doable and being able to implement models such as object detection models is something I hope to master and fully understand.

6.6 Final Thoughts

Overall, I believe this project has given me a great deal of knowledge and I hope to look into the concept of object detection further and get a better understanding of it.

Bibliography

[1] *Vision Impairment and blindness.* URL: https://www.who.int/news-room/fact-sheets/detail/blindness-and-visual-impairment.

[2] *Assistive technology.* URL: https://www.who.int/news-room/fact-sheets/detail/assistive-technology.

[3] *Seeing ai.* Oct. 2018. URL: https://www.microsoft.com/en-us/garage/wall-of-fame/seeing-ai/.

[4] *The story about be my eyes.* URL: https://www.bemyeyes.com/about.

[5] *Introducing our virtual volunteer tool for people who are blind or have low vision, powered by OpenAI's GPT-4.* URL: https://www.bemyeyes.com/blog/introducing-be-my-eyes-virtual-volunteer.

[6] *Envision app.* URL: https://www.letsenvision.com/app.

[7] *Envision glasses.* URL: https://www.letsenvision.com/glasses.

[8] *Envision ally.* URL: https://www.letsenvision.com/ally.

[9] Jia Deng et al. "ImageNet: A large-scale hierarchical image database". In: *2009 IEEE Conference on Computer Vision and Pattern Recognition* (2009). DOI: 10.1109/cvpr.2009.5206848.

[10] Sidheswar Routray, Arun Kumar Ray, and Chandrabhanu Mishra. "Analysis of various image feature extraction methods against noisy image: SIFT, surf and Hog". In: *2017 Second International Conference on Electrical, Computer and Communication Technologies (ICECCT)* (2017). DOI: 10.1109/icecct.2017.8117846.

[11] *What is deep learning?* URL: https://www.ibm.com/topics/deep-learning.

[12] Ross Girshick et al. "Rich feature hierarchies for accurate object detection and semantic segmentation". In: *2014 IEEE Conference on Computer Vision and Pattern Recognition* (2014). DOI: 10.1109/cvpr.2014.81.

[13] Ross Girshick. "Fast R-CNN". In: *2015 IEEE International Conference on Computer Vision (ICCV)* (2015). DOI: 10.1109/iccv.2015.169.

[14] Shaoqing Ren et al. "Faster R-CNN: Towards real-time object detection with region proposal networks". In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* 39.6 (2017), pp. 1137–1149. DOI: 10.1109/tpami.2016.2577031.

[15] Wei Liu et al. "SSD: Single shot multibox detector". In: *Computer Vision – ECCV 2016* (2016). DOI: 10.1007/978-3-319-46448-0_2.

[16] Joseph Redmon et al. "You only look once: Unified, real-time object detection". In: *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)* (2016). DOI: 10.1109/cvpr.2016.91.

[17] Joseph Redmon and Ali Farhadi. "Yolo9000: Better, faster, stronger". In: *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)* (2017). DOI: 10.1109/cvpr.2017.690.

[18] Joseph Redmon and Ali Farhadi. *YOLOv3: An Incremental Improvement.* 2018. arXiv: 1804.02767 [cs.CV].

[19] Tsung-Yi Lin et al. "Feature Pyramid Networks for Object Detection". In: *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)* (2017). DOI: 10.1109/cvpr.2017.106.

[20] Alexey Bochkovskiy, Chien-Yao Wang, and Hong-Yuan Mark Liao. *YOLOv4: Optimal Speed and Accuracy of Object Detection.* 2020. arXiv: 2004.10934 [cs.CV].

[21] *Python 3.9.16 documentation.* URL: https://docs.python.org/3.9/.

[22] *Tensorflow.* URL: https://www.tensorflow.org/.

[23] *Tensorflow probability.* URL: https://www.tensorflow.org/probability.

[24] URL: https://opencv.org/.

[25] Gerhard Neuhold et al. "The mapillary vistas dataset for semantic understanding of street scenes". In: *2017 IEEE International Conference on Computer Vision (ICCV)* (2017). DOI: 10.1109/iccv.2017.534.

[26] URL: https://www.mapillary.com/dataset/vistas.

[27] Roboflow. *Udacity Self driving car object detection dataset - fixed-small*. Aug. 2022. URL: https://public.roboflow.com/object-detection/self-driving-car/3.

[28] Udacity. *Self-driving-car/annotations at master · udacity/self-driving-car*. URL: https://github.com/udacity/self-driving-car/tree/master/annotations.

[29] Aladdin Persson. *Yolov1 from scratch*. Oct. 2020. URL: https://www.youtube.com/watch?v=n9_XyCGr-MI.

[30] Aladdin Persson. *Machine-learning-collection/ml/pytorch/object$_d$etection/yoloatmasterAladdinpersson/machine−learning − collection*. URL: https://github.com/aladdinpersson/Machine-Learning-Collection/tree/master/ML/Pytorch/object_detection/YOLO.

[31] *What are convolutional neural networks?* URL: https://www.ibm.com/topics/convolutional-neural-networks.

[32] Saurabh Singh. *Leaky relu as an activation function in neural networks*. Aug. 2021. URL: https://deeplearninguniversity.com/leaky-relu-as-an-activation-function-in-neural-networks/.