



Using Machine Learning/Deep learning Algorithms for vehicle
recognition/detection



Course: Computer Science BSc

Name: Mohammed Abdullahi Guled

SID: 8504546

Supervisor: Nazaraf Shah

Contents

Abstract:.....	4
Introduction:	4
Importance of Machine Learning:.....	4
Applications of vehicle detection in the real world:	4
Aim/ objectives of this project:.....	5
Literature Review:.....	6
Machine learning:	6
Reinforcement Learning:	6
Unsupervised Learning:	7
Supervised Learning:.....	10
Neural Networks:	14
Faster R-CNN:.....	15
SSD (Single-Shot Detector):.....	17
Network Similarities:.....	18
Method:	19
TensorFlow + Setup:	19
Dataset Preparation:.....	19
LabelImg:.....	20
Training + TensorBoard:.....	21
Google Colab:	21
TensorBoard:.....	22
Evaluation:	23
Faster R-CNN Resnet 50:.....	27
Faster R-CNN Resnet 152:.....	29
SSD Resnet 50:	30
SSD Resnet 152:	31
Discussion:	32
Management / Reflection:.....	33
Kanban Board:.....	33
Gantt Chart:	33
Social, Legal and Ethical Issues:	34
Presentation Feedback:	34
Conclusion:.....	35
Bibliography:	36
Appendix:	39

Presentation Slides:	39
Supervisor Meetings:	47

6001CEM Declaration of originality

I Declare that This project is all my own work and has not been copied in part or in whole from any other source except where duly acknowledged. As such, all use of previously published work (from books, journals, magazines, internet etc.) has been acknowledged by citation within the main report to an item in the References or Bibliography lists. I also agree that an electronic copy of this project may be stored and used for the purposes of plagiarism prevention and detection.

Statement of copyright

I acknowledge that the copyright of this project report, and any product developed as part of the project, belong to Coventry University. Support, including funding, is available to commercialize products and services developed by staff and students. Any revenue that is generated is split with the inventor/s of the product or service. For further information, please see www.coventry.ac.uk/ipr or contact ipr@coventry.ac.uk.

Statement of ethical engagement

I declare that a proposal for this project has been submitted to the Coventry University ethics monitoring website (<https://ethics.coventry.ac.uk/>) and that the application number is listed below (Note: Projects without an ethical application number will be rejected for marking)

Signed: M. Guled

Date:15/03/2021

Please complete all fields.

First Name:	Mohammed
Last Name:	Guled
Student ID number	8504546
Ethics Application Number	P115742
1 st Supervisor Name	Nazaraf Shah
2 nd Supervisor Name	

This form must be completed, copied, or scanned and included with your project submission to Turnitin. Failure to append these declarations may result in your project being rejected for marking.

Abstract:

Vehicle Object detection is a major factor in traffic surveillance as it allows for accurate tracking of vehicles and information about those vehicles (e.g. make, colour and license numbers). This is done using deep learning machine learning techniques such as TensorFlow object detection models which can be trained for specialised use in vehicle detection.

The aim of this project is to create an Object Detection model capable of detecting vehicles of numerous classes at a reasonable accuracy. This project will also demonstrate a comparison of the performance (accuracy and speed) between different object detection models (Faster R-CNN, SSD-Resnet50, MobileNet) trained on a single image dataset and will explain the potential reasons for these differences.

Introduction:

Importance of Machine Learning:

Machine learning is an incredibly important part of Artificial Intelligence which is used in many parts of industry for the purpose of utilising data (typically in the form of large datasets being used to train machine learning models) to create predictions without the need for human interaction in a system. This is what gives AI the ability to independently learn and adapt to new data to fulfil its purpose at greater accuracy and efficiency. The importance of machine learning is even more apparent in the modern world due to the advent of Big Data which further proves that machine learning tools are necessary. It would not be feasible for humans to attempt to sort through and utilise such great amounts of data. In a previous semester, we covered artificial intelligence as part of our projects and I personally incorporated classification algorithms for the purpose of predicting cancer within patients utilising a pre-existing datasets. I found this project to be very interesting and decided that I wanted to use my dissertation as a way to expand my knowledge regarding machine learning and artificial intelligence.

Machine learning has been used for many purposes, one of which is in the medical industry where algorithms have been created capable of predicting potential diseases such as cancers based on large datasets of tumour measurements and other values. This allows doctors to pick up on problems much earlier than previously thought potentially saving lives. Another common use of machine learning that most people use frequently is voice recognition where a person is able to command a device utilising their voices as an input, these AIs are also a product of machine learning.

Applications of vehicle detection in the real world:

Vehicle Detection has many important applications in the real world, one of which is traffic management where machine learning techniques are utilised to accurately track different classes of vehicles on a live camera feed. The data gathered can then be used for analysis of traffic within specific areas and timeframes.

In large cities such as London, traffic congestion has been a major problem for decades and is a huge cause of frustration for drivers. There are also many negative environmental impacts due to the vast number of vehicles driving on roads that leave their engines running during traffic jams. Air pollution is already a large problem in many cities. In London alone, the average speed of vehicles on major roads have decreased over the last few years causing journey times to become longer which in turn causes more delays (*Franks 2016*). This is occurring because the roads have become overloaded with more traffic than they were designed to tolerate. Restructuring the cities road network would be an

incredibly expensive, resource heavy and time intensive task which is why effective traffic management would be very beneficial in situations like this as less time spent sitting in traffic is likely to reduce the amount of emissions from vehicles across the city.

There are multiple approaches to vehicle detection technology ranging from radar sensors, wireless ultrasonic sensors, magnetometers, and even optical sensors. These methods are typically used in places like car washes and car parks (both indoors and outdoors) and the range of their detection is small, and the data gathered is not enough for use in traffic systems or data analysis. None of these methods provide the wide scale coverage that an image processing system can, along with the vast amount of data that can be extracted from visual footage (e.g. color, make, type of vehicle, speed).

Aim/ objectives of this project:

The aim of this project is to demonstrate how it is possible to train a neural network for the purpose of vehicle detection and how we can use pre-existing image datasets to train these neural networks. This will be done utilising an open-source software called TensorFlow. Another objective of this project is proving the difference in performance between the different neural networks that will be trained using the same techniques and data. This will be done using the built-in evaluation tools in TensorFlow.

Literature Review:

Machine learning:

The term 'Machine Learning' was first used by Arthur Samuel in 1952 (Foote 2019). He created a program capable of playing checkers in the 1950s and had utilised some mechanics that allowed his program to improve which is a very primitive form of modern machine learning. Machine learning is the process of constructing algorithms that have the ability to continuously learn and create predictions based on input data (e.g. numerical values/variables, image datasets etc...). These algorithms then create models that are specialised for creating aforementioned predictions based on that specific data. For example, the models that are used in this project are specialised for use in detecting images of vehicles that are within the image dataset used for training. Machine Learning allows computer applications to independently act/improve without being driven by strict human-inputted programming. There are numerous forms of Machine learning; reinforcement learning, supervised learning, and unsupervised learning each with their own distinguishing features and applications.

Reinforcement Learning:

Reinforcement learning is a form of machine learning that revolves around training a model to make a sequence of decisions and learning from the consequences of those choices and also gaining "rewards" for the best choices made. Essentially, maximising the rewards in any given situation (Bajaj 2020). This form of machine learning heavily mimics the way our own brain works as we also evaluate the value of each choice made based on potential reward/consequence and prior experience (Lee et al. 2012). Reinforcement learning models rarely make the same errors more than once which is one of the stronger points compared to other deep learning techniques.

However, unlike supervised learning, there is no given answer for reinforcement learning whilst training which means the reinforcement model is on its own to figure out the optimal path given its starting point. An analogy for this model would trying to direct your way through a maze, attempting to make it to the exit, making the wrong turns would mean wasting time and not escaping the maze and vice versa the reward being escaping the maze upon making a set of good turns.

Models of Reinforcement Learning:

The most common form of a Reinforcement Learning model is the Markov decision process (MDPs). An MDP describes a framework/environment for reinforcement learning. MDPs are a tuple that typically consist of numerous elements (Sreenath14 2020):

- S , the set of states within the environment (e.g. every position within a grade).
- A , the set of potential actions (e.g. any number of directions from a specific point in a maze).
- R , the reward function
- T , the transition function (transitional probabilities between states)
- γ , the discount function.

The end goal of an MDP is to find an optimal “policy” which essentially the best choice the decision maker can make from any state within the environment and this is decided using the reward function MDPs are known for their flexibility and ability to adapt to changes such as anomalies within the environment. However, they can be time consuming.

Application of Reinforcement Learning:

The applications of this reinforcement learning are amongst the most well-known machine learning applications especially in regard to robotics and AI trained for specific games. One example is reinforcement learning being implemented into models trained for chess. Self-driving cars also implement some reinforcement learning algorithms.

Unsupervised Learning:

In this form of machine learning, there are no labels present to the algorithm in use, which means that the model has to independently create structure out of the inputted data with no guidelines in sight. Essentially, attempting to find hidden patterns from scratch without given labels is why this method is called unsupervised. Furthermore, this method of machine learning can be difficult to train as the model has no “error or reward” signal which would usually allow it to know how accurate its predictions/evaluations were. Unsupervised learning has flaws due to its inherent nature as non-directed algorithm. Yet, they are still necessary as large datasets are costly to evaluate so labelling all elements of the data would be difficult (Mishra 2017). Unsupervised learning techniques are usually applied for two purposes; Clustering and Association.

Clustering:

Clustering is the most prevalent use of unsupervised machine learning and it is the process of creating groups from input data based on potential similarities between those entities. These groups are known as clusters and each of group of clusters are distinguished by what it is dissimilar between them (Tzanis et al. 2006). Determining what makes a good cluster is up to the user as there are no pre-determined criteria for “good” clusters. However, the existence of clusters shows the model recognises differences between the data values.

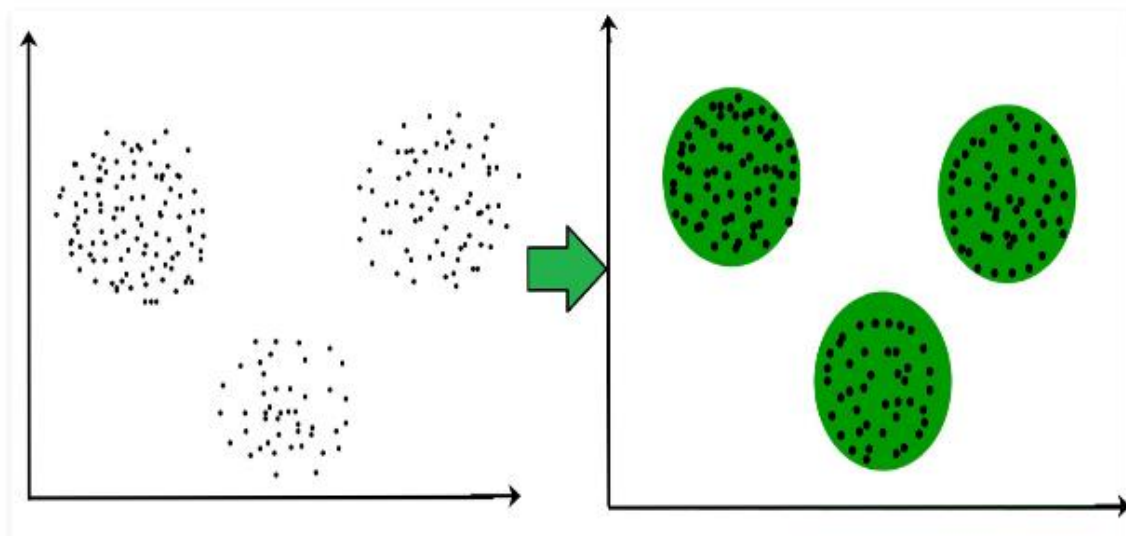


Figure 1 - Clustering in Machine Learning [6]

There are numerous algorithms that can be used for clustering (e.g. probabilistic, hierarchical, overlapping, and exclusive clustering...).

- Hierarchical Clustering – This form of clustering involves finding new and consecutive clusters based on a top-bottom hierarchy of clusters (Prasad 2020). Hierarchical clustering are known for being one of the easiest forms of clustering to implement utilising code. The output of this clustering is called a dendrogram and is simple to understand and provides a useful insight into the clustered data which adds to its appeal. However, this algorithm cannot undo any actions it has previously taken (e.g. merging/separating clusters or assigning clusters). Furthermore, this algorithm is known to be less efficient and slower than other forms of clustering due to its time complexity causing it to have long computation times for each step.

There are also two main approaches to this form of clustering. One of which is called the Divisive approach which works by taking the whole set as a single large cluster and dividing into successively smaller clusters as the process constantly iterates. At the first iteration, the cluster is split into two smaller clusters and the same process occurs until the desired number of clusters is achieved.

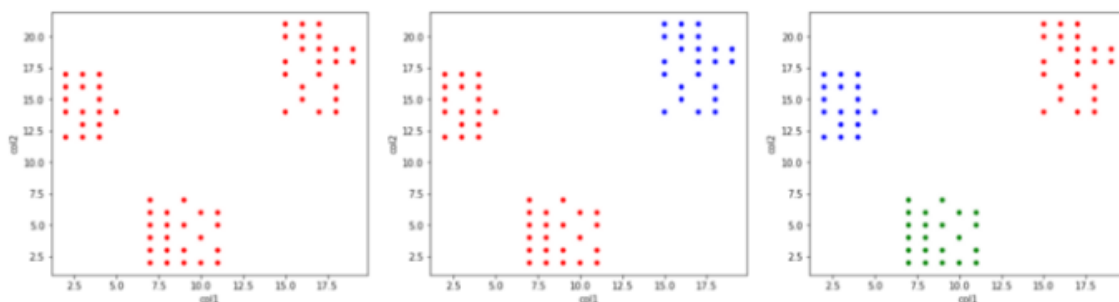


Figure 2 - Divisive Clustering [12]

In the image above, we can see that from the 1st image to the 2nd image, the three red “sets” become separate into two separate sets of data as we can see the new blue cluster. The same thing happens again in the 3rd image, finally achieving 3 separate clusters. The algorithm works its way down the hierarchy.

The second approach is called the Agglomerative approach which works by each element of the data being considered its own cluster and then these clusters are combined based on their similarity. Essentially working the opposite direction in the hierarchy of the divisive approach. The steps of this form of clustering are; 1. assigning each element of the data as its own individual cluster, then finding and merging the most similar pair of clusters from the data, the distances/similarity between the new and older clusters is calculated and the final step is just repeating the 2nd step until all the elements of the data are compiled into a single cluster (Marinova–Boncheva 2008).

- K-means Clustering Algorithm:
K-means clustering works by attempting to cluster “n” number of objects based into “k” number of clusters (typically $k < n$) so essentially partitioning an unlabelled dataset. This algorithm works differently to hierarchical clustering as it has a pre-determined number of clusters (k) that it has

to search for within the data. Each cluster will also have a central point called a 'centroid'. Centroids are vectors that contains the mean of the objects within each of these clusters. Euclidian distance is a measure used in K-means clustering to find the distance between data points and centroids.

The basic steps:

1. Choose the K number of clusters that are to be identified.
2. Random points within the data are to be selected as the initial centroids for the clusters so the number of these points is to be equal to K.
3. All points within the data are then assigned to the closest relative cluster centroid based on Euclidian distance.
4. Recompute the centroid of all the newly formed clusters within the dataset.
5. Steps 3+4 are to be repeated until the centroids of new clusters no longer change.

K-means clustering unlike hierarchical clustering can be used to be for large datasets as it scales much better due to its superior computational costs, but it can still be sensitive to scale. K-means also performs quickly and is also simple to implement. It produces tighter clusters than other forms of clusters. However, this algorithm can struggle when the dataset is not linear and also lacks consistency due to the random placement of the initial centroids.

Association:

Association on the other hand, is about associating a description to elements within a dataset to discover relationships between them. Typically used for Market Basket Analysis to find patterns between which items a customer usually puts in their basket before checking out, allowing an online marketplace to know which items to recommend to customers based on their current selection (Wagle 2020). This logic is also applied to what kind of stores the same customers frequent and other assorted behaviours and choices.

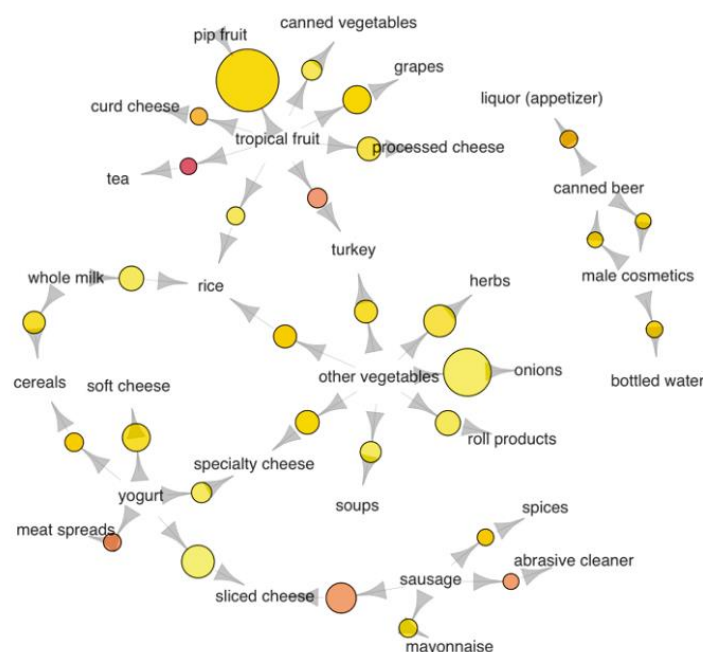


Figure 3 - Association Products [8]

Supervised Learning:

Unlike unsupervised learning, supervised learning contains the labels required for all of the training data. The premise of this form of machine learning is that the system (e.g. a TensorFlow object detection model), is provided with two sets of data; a testing set and a pre-prepared training set which is labelled and classified before inputting into as a dataset. The model will then use the training dataset to learn how to better predict the unlabelled objects within the testing dataset. For example, if we wanted to create a model that was capable of detecting fruits, we would input a large training dataset full images of various classes of fruits which are also labelled and a smaller testing image dataset of the same classes of fruits (common ratios are 80/20 or 90/10). The algorithm will begin using that training data and we can track its prediction performance over time utilising its loss function.

Supervised learning models have many advantages over the forms of machine learning especially in regard to classified data and image recognition. Supervised learning is easy to understand as we can accurately track the algorithms' improvement unlike unsupervised learning. Supervised learning also often falls victim to overfitting to the training dataset, making the model less suitable for any new data that it might be exposed to after training (e.g. testing a model on completely new images of specific classes). On the other hand, since supervised learning generally has superior accuracy to unsupervised due to its guided learning making its training more straightforward.

Forms of Supervised Learning:

There are numerous forms, however the most common forms of supervised learning problems typically involve classification, regression, and neural network algorithms.

Classification:

Classification algorithm problems usually revolve around categorising a dataset into either multiple suitable classes or into binary classification (Sathya and Abraham 2013). This can be used for face detection, handwriting recognition and even speech recognition (Waseem 2020). There are numerous forms for classification algorithms and here is a brief overview of some:

- *Logistic Regression:*

This is a binary classification algorithm and uses true labels for its input making it a supervised learning algorithm. It is used to predict a categorical dependent variable based on a set of independent variables (Point 2021). This algorithm utilises the logistic function to predict the output of a categorical dependent variable and outputs a binary probability (e.g. 0 or 1, true or false). I previously incorporated this algorithm when creating a model capable of predicting breast cancer based on variables such as tumour size.

- *Naïve Bayes:*

This algorithm is based upon the Bayes' theorem that gives the assumption of independence between the features in the data. Naïve Bayes theorem allows us to predict the probability of an event based on previous knowledge of other events related to the former event (Gupta 2017).

$$P(B|A) = \frac{p(A|B)p(B)}{p(A)}$$

Figure 4 - Bayes Formula [19]

1. $P(B|A)$ = Probability of B given Event A has already occurred and vice versa for $P(A|B)$.
2. $P(A)$ = Probability of Event A.
3. $P(B)$ = Probability of Event B.

This algorithm works well with weather prediction or emails spam detection which makes sense as both of those involve utilising information about previous events for predictions. It also outperforms most other classification in terms of computation is known for only needing a small dataset to acquire the necessary parameters.

- *Decision Tree:*

This can be a classification model that is also used for prediction purposes. There are two elements of decision trees; nodes and branches which is how it emulates a tree and utilises rules to continually break down the structure into smaller partitions and move down the tree in a hierarchical sense.

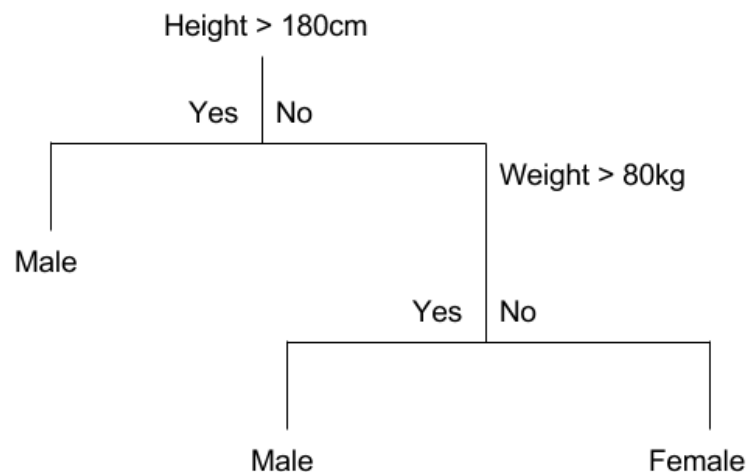


Figure 5 - A decision tree [21]

Decision tree makes it incredibly easy to visualise data and also is useful for numerous forms of data (both discrete and categorical). Furthermore, it has low computational costs compared to other forms of classification algorithms which means it performs quickly but with more class labels, the computational complexity of the algorithm also increases which leads into the next point. Decision trees work best with smaller datasets

as the larger the data, the more complex the decisions trees become which gradually become harder for the user to interpret. Furthermore, decision trees are prone to overfitting to the data and can be unstable compared to other algorithms due to small changes in the input data can completely change the overall structure of the decision tree.

- *K-Nearest Neighbour (KNN):*

KMN can be used for solving both classification and regression problems. This algorithm is known as a simple/lazy algorithm as it does not build a model but rather it stores the instances of training data and then performs actions when classifying new data. It works by assuming that elements that are similar must be close to one another and it uses this assumption for classifying new data into categories. K is the number of the neighbours that are closest to the new data point hence the algorithms name being “nearest neighbour”. KNN has a major disadvantage in regard to its performance, as the number of variables in our data increases, the algorithms’ computational cost increases on top of it already being higher than other algorithms (Harrison 2018).

Regression:

Regression problems on the other hand revolve around a mode capable of predicting a continuous numerical value for a specific variable as the output. Also, regression models can only be used with continuous data unlike discrete data used for classification models. For example, predicting age, shoe size or even predicting changes in stock prices. Regression models are able to do this as they can find the correlation between different variables. These variables are the independent and dependent variables; independent variables are the contributing factors we know affects the dependent variables (UYSAL and GÜVENİR 1999). Dependent variables is the variable we are trying to gain insight to and predict values for. Which is why regression models can also be used for just finding a casual-effect relationship between variables and not always predicting values.

Regression models typically work by plotting a graph with the two variables plotted as the axis. We then plot the data and use that to plot a “line of best fit” or a curve that passes through most of the data points and remains in close proximity to all. There can be values that are not close to this curve and these are known as outliers/anomalies. Just like classification models, regression models can also overfit and underfit to training data. Regression models can be classified in two way; linear and non-linear regression.

- *Linear Regression:*

LR is a very straightforward approach to regression and works essentially the same way I explained above. A graph that utilises data points and a line of best fit between the two axis (independent and dependent variable) for prediction purposes. However, there are two types of linear regression; simple and multiple. Simple linear regression only consists of one input (x) variable and multiple is self-explanatory. This models work best when the user is already aware of a linear relationship between the two variables and that keeps its complexity low. It also works well regardless of dataset size.

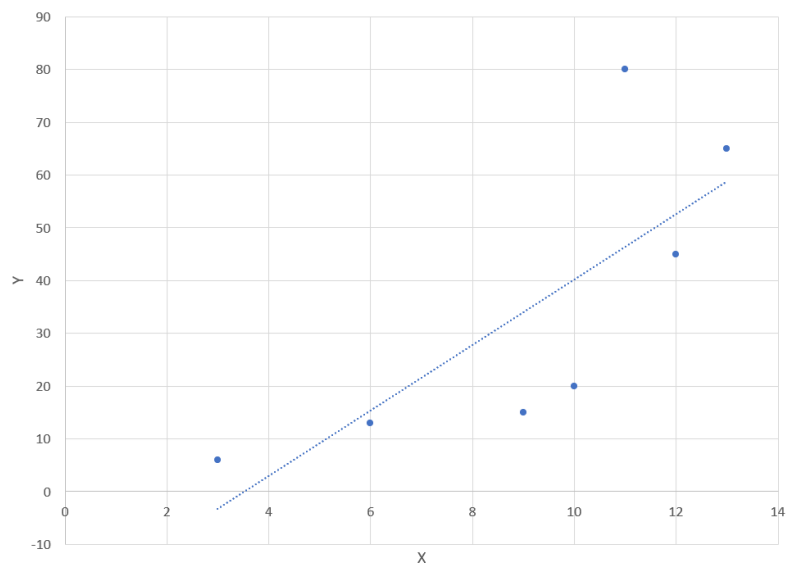


Figure 6 - Regression Graph + Trendline Example

- **Polynomial Regression:**

PR works in a very similar manner to linear regression also utilising a graph to demonstrate correlation between the independent and dependent variables. However, instead the data used is non-linear which means a linear curve is not suitable for producing predictions which is why a polynomial curve is used to best fit the data.

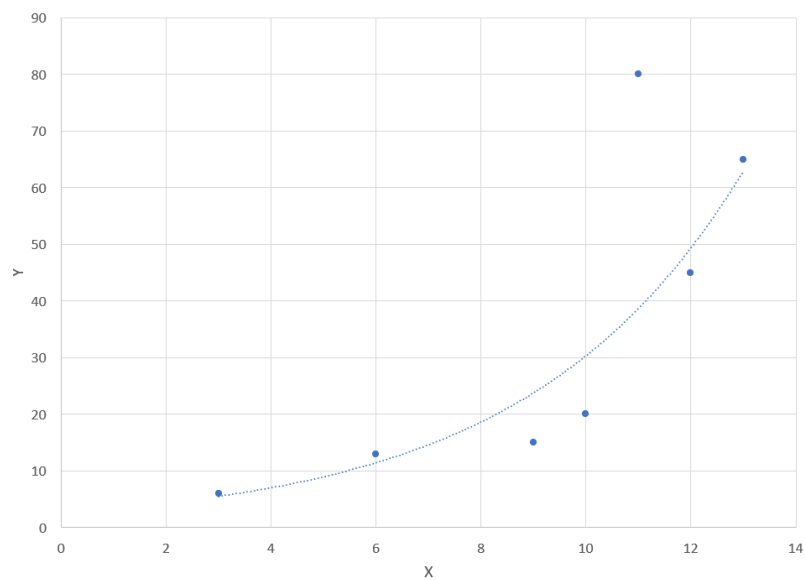


Figure 7 - Example of Polynomial Curve

Neural Networks:

The method of machine learning that is typically used for any type of image processing/object detection is neural networks. Neural Networks are typically used to discover complex relationships between input and output data and also patterns within data (e.g. object detection, speech recognition and image processing). Neural networks are a form of machine learning that is heavily inspired by how our own biological neural networks function. It simulates the network of neurons that are present in our brains which is what allows an artificial neural network model to learn and make decisions independently. These “neurons” can also be referred to as nodes and they are capable of communicating with one another within the network.

The network functions when presented with input data and this data is processed utilising multiple layers to produce the output result. Most neural networks contain a structure of fully connected layers and these layers typically come in three forms; input, hidden and output layers. Input layers are what contain the users’ raw input data and there is always one. Hidden layers are located between the input and output layers and this is where the model does all of the “learning”. These layers are referred to as hidden as they are not visible to external systems and only to the network itself (Malik 2019).

Hidden layers are capable of transforming the input data and each layer can carry a different function. For example, in an object detection task, each hidden layer can be used to recognise different elements of an object which the overall network can then use to make predictions and then the output layer then outputs a final image that uses the predictions to classify the image. The more hidden layers a network contains, the longer the model takes to produce an output which was a problem that was also present in this project.

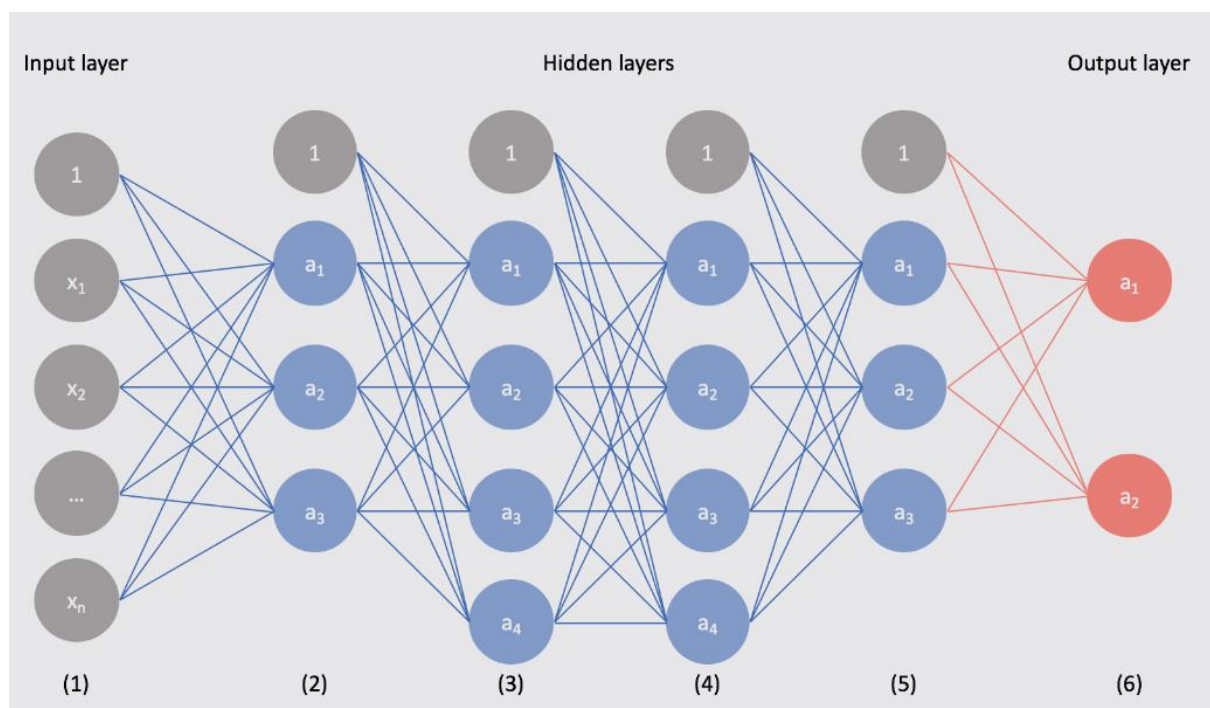


Figure 8 - Typical Neural Networks [26]

In this project, the form of neural networks that will be used are Convolutional Neural Networks (CNN). As the name implies, CNN have a structure of one or more convolutional layers, pooling layers, and fully connected layers.

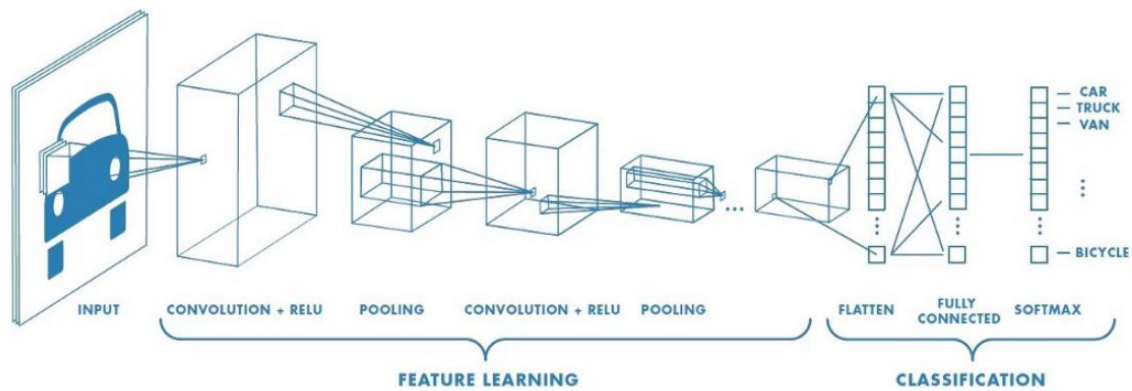


Figure 9 - CNN Architecture [25]

Faster R-CNN:

This is one of the two forms of object detection models that will be evaluated during this project. Faster R-CNN is the latest and most impressive of R-CNN model family, the first of which was called just R-CNN and was created by Ross Girshick (Girshick 2014). His goal was to develop a CNN that was capable of differentiating parts of an image by utilising selective search to find regions of the image and those are called region proposals. There are 2000 region proposals for each images and these were then warped and inputted into the CNN element of the model where the feature extraction occurred, and the presence of objects was then predicted within each of the region proposals. However, this method caused the network to take an extremely long time to train as each of the 2000 region proposals would have to be classified which also meant that using this model for video detection or live feeds was infeasible.

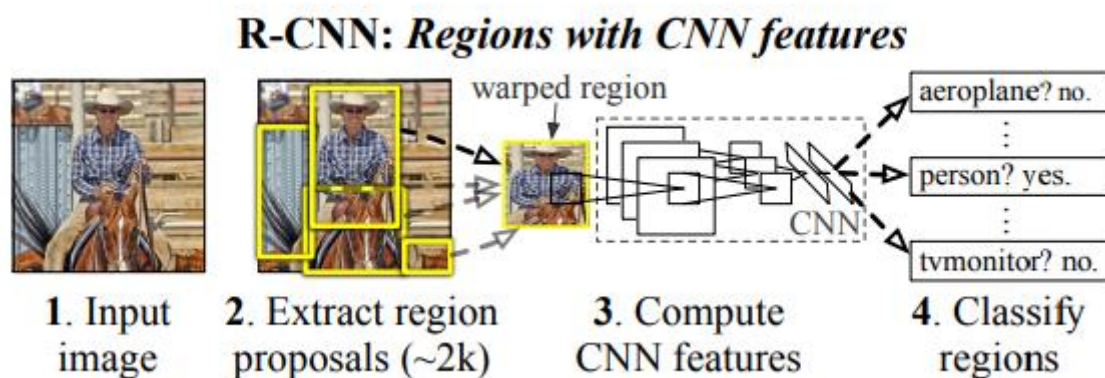


Figure 10 - R-CNN [27]

To overcome this performance issue, Girshick developed another model called "Fast R-CNN" which instead of extracting features for each individual region proposal, the CNN takes the whole image and the region proposals as an input whilst combining numerous parts of the R-CNN architecture to make the whole process more streamlined. Furthermore, this model uses a "softmax" layer for the

purpose of classifying the region proposals instead of SVM (support vector machine). This change allowed for greater accuracy and faster classification performance. Overall, this model allowed for greater mAP and vastly improved training speed by up to 213x faster than R-CNN (Girshick 2015). However, even Fast R-CNN had a problem with its performance due to a speed bottleneck caused by the selective search region proposal generation (pawangfg, 2020).

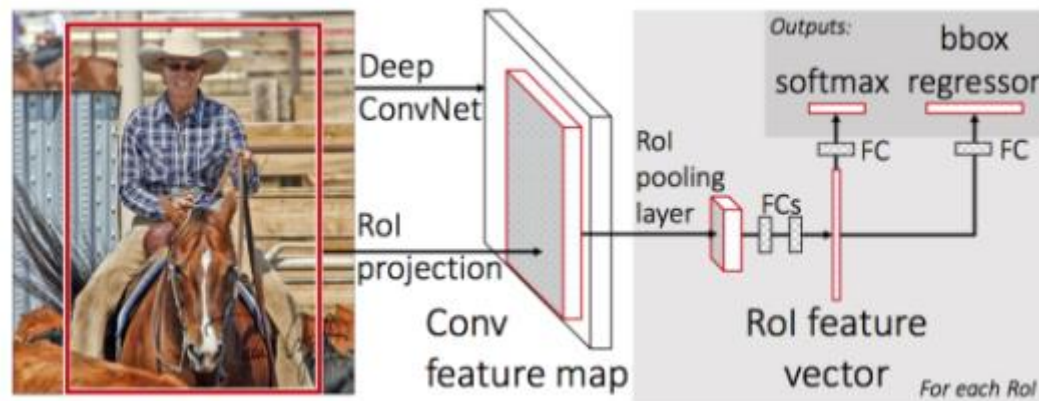


Figure 11 - Fast R-CNN Architecture [28]

There were further potential improvements that could've been made to the Fast R-CNN approach which is what led to the creation of Faster R-CNN. Faster R-CNN is very similar in architecture to Fast R-CNN except for one major difference being their approach to region proposal selection. Seeing as how selective search slows down both the previous models and the main cause of bottlenecks on Fast R-CNN, Shaoqing Ren (Ren et al. 2017) created a new method that removed that issue by integrating the region proposals via a separate network called the RPN (region proposal network) which vastly improved the speed of Faster R-CNN. The RPN takes the image as an input and outputs rectangular object proposals each with their own objectness score (Ren et al. 2017).

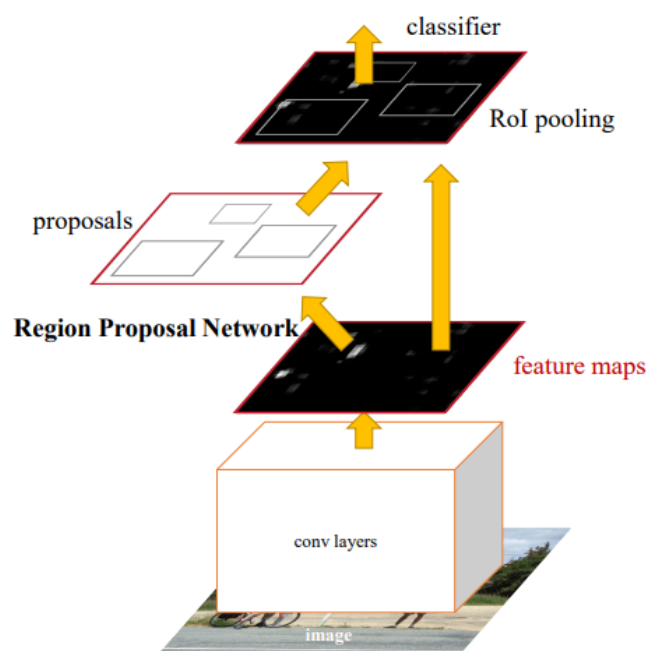


Figure 12 - Faster R-CNN Architecture [30]

A study done by Nikhil Yadav revolves around comparing the performance of numerous object detection models found that Faster R-CNN (Resnet) consistently outperformed other models such as multiple variations of SSD and R-FCN in terms of mAP (mean average precision) which is the standard metric for measuring the performance of models (Yadav and Binay 2017). However, as Yadav states in the article, Faster R-CNN is still much slower than its counterparts. As this project, will utilise both SSD and Faster R-CNN, it is most likely that the Faster R-CNN models will outperform SSD in accuracy.

Model Combination	mAP score	GPU time
SSD MobileNet	19	40
SSD VGG-16	20.5	130
SSD Resnet-101	26.2	175
SSD Inception Resnet	20.3	80
Faster R-CNN MobileNet	19	118
Faster R-CNN VGG-16	24.9	250
Faster R-CNN Resnet-101	33	396
Faster R-CNN Inception Resnet	34.2	860
R-FCN MobileNet	13.4	75
R-FCN Resnet-101	30.5	386
R-FCN Inception Resnet	30.7	388

Figure 13 - Comparison of different models and their mAP scores [31].

SSD (Single-Shot Detector):

SSD is the second type of object detection models that will be used during this project and its architecture is quite different to Faster R-CNN. SSD models are known for their performance both in terms of their accuracy scores and also in terms of speed as they can be used for real-time video feeds. The SSD architecture only consists of a single CNN that is capable of predicting the bounding box positions and also classifying them in one pass unlike Faster R-CNN in which the process takes two stages for the final result (Kang 2020). SSD also lacks any initial region proposal or object proposal generation.

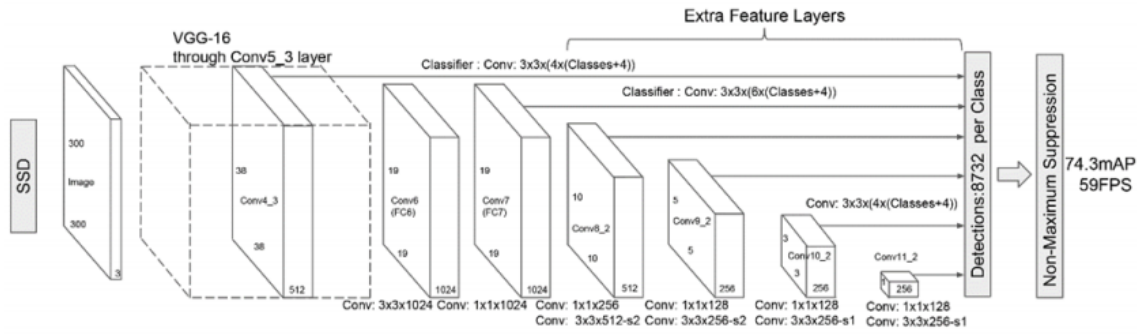


Figure 14 - SSD Architecture [32]

Network Similarities:

The Faster R-CNN and SSD models that will be used in this project will utilise the ResNet architecture which is a deep residual network (a form of CNN) and the two models will use ResNet 50 and 152 respectively, the numbers referring to the number of layers each network contains. This keeps the comparison fair between the models. The computational complexity of ResNet152's is 3.1 times that of ResNet50 which explains why those models will take longer to train. Furthermore, all models are pre-trained on larger image datasets such as COCO (Common Objects in Context) as without this pre-training, the models would take far longer to train.

Method:

TensorFlow + Setup:

The TensorFlow API by Google forms the backbone of this whole project. It is an open-source software which specialises in a wide range of machine learning tasks and the relevant one being the ability to utilise neural networks to train object detection models. Using the TensorFlow documentation (Vladimirov 2020) as a tutorial, I was able to setup the API along with the libraries that were required for the object detection tasks. First, I had to download the Tensorflow API from the official GitHub repo (<https://github.com/tensorflow/tensorflow>) and then setup the working folder. I also utilised Jupyter Anaconda to create a virtual environment (this helps avoid any incompatibility problems with pre-installed packages on my PC) that would contain all the necessary libraries and packages such as; tensorflow-gpu (allowed the use of my graphics card), pycocotools/TensorBoard (this allowed for the evaluation of the models during and after training). I also had to download the models that would be used for object detection from the Tensorflow 2.0 Model Zoo (https://github.com/tensorflow/models/blob/master/research/object_detection/g3doc/tf2_detection_zoo.md). One issue I came across with this was the restrictive model selection available for TensorFlow. Majority of the models were either Faster R-CNN or SSD variations and I came across an error when trying to train the other models, so I decided to only use Faster R-CNN and SSD models for this project as they were more reliable.

Hardware used for this project:

Current PC:

- AMD Ryzen 5 3600: 6 core/ 12 thread processor at 4.2 GHz Boost
- 16GB DDR4 RAM
- GeForce RTX 2070 Super: 8GB VRAM

This is the setup I used for the initial training of the models. My graphics card is plenty powerful for this task.

Dataset Preparation:

I had to prepare a large image dataset that fulfilled certain conditions:

- Not too high resolution (close to 1000x1000). Otherwise it would cause very large folder/image sizes and also
- Had to mostly revolve around inner city environments as stated in the project proposal.
- Within rights to use image without paying/requesting explicit permission from the author.
- Large number of images with a wide variety of settings, angle, and vehicle types.

Finding a single image dataset that fulfilled all of these requirements and also contained all 4 classes proved to be difficult to find so I utilised images from multiply sources; the primary source was from an MIT StreetScenes dataset (Bileschi 2007) and the secondary source from Flickr Albums (Richardson 2021). My final images dataset contained 451 images with a 9:1 split for the training (375) and testing (41) dataset partitions.

I found that the composition of my data could cause some potential problems with the predictions that models would make. This is due to the ratio of the classes being incredibly unbalanced with a heavy bias towards cars and vans over trucks and buses. This could cause the model to make much better predictions for the former classes due to greater sample size which did turn out to be the case across all of my models.

Labellmg:

After gathering the images and creating the partitions for the test/training dataset, the images would have to be labelled for use in training. There are 4 classes in my dataset; cars, van, trucks, and buses. There are tools that are capable of automatically labelling images but with varying levels of accuracy so I decided to manually label using a python program called Labellmg (<https://github.com/tzutalin/labellmg>). This program works by granting the user a graphical interface that allows them to draw boxes around the desired objects in the images and assign a label to each box which forms the classes. These boxes and their labels are then outputted as an XML file with the same name as the image. The output folder for these XML files can be changed but for simplicity I kept them in the same folder.

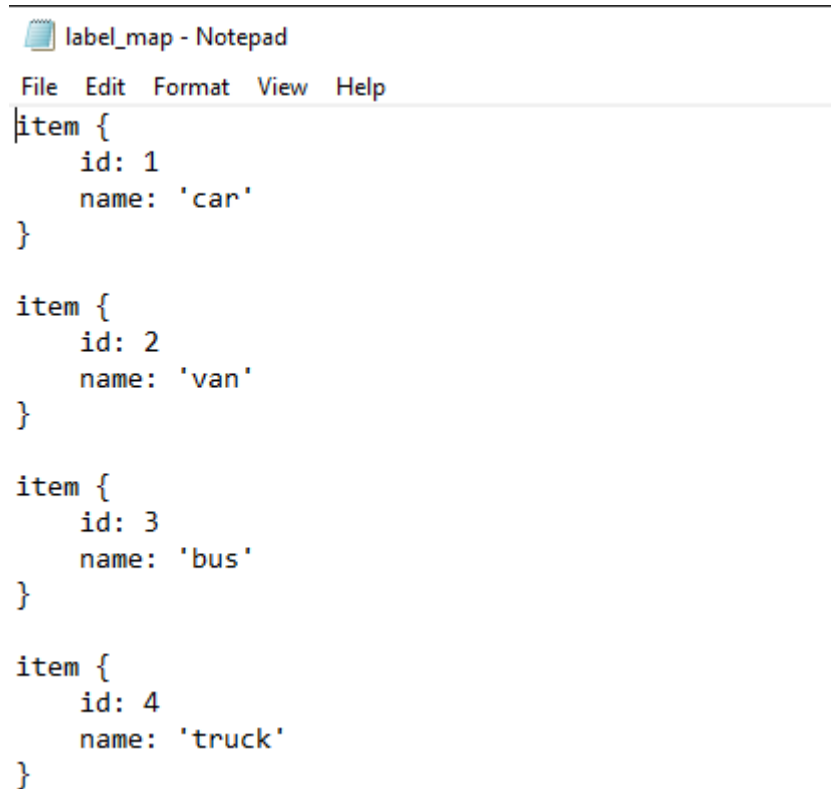


Figure 15 - Labellmg

Furthermore, the process of labelling was incredibly long as there were 100s of images with multiple objects in these images as well. This is the part of object detection which makes it supervised learning as the model already knows what the objects are prior to testing. This also introduces the possibility for human error as it is easy to mislabel the boxes within the program and it is also possible to draw a poor bounding box around the object which could negatively impact the model as well.

After labelling all of the images in both datasets, I utilised a built in TensorFlow python script that allowed me to convert the images and their corresponding XML files into a TF Record file which is

what the TensorFlow models takes as input for training and testing. I also created a label map which is what allows the model to know which classes correlate to the annotations of the images in the dataset.



```
label_map - Notepad
File Edit Format View Help
{
  "item": {
    "id": 1,
    "name": "car"
  },
  "item": {
    "id": 2,
    "name": "van"
  },
  "item": {
    "id": 3,
    "name": "bus"
  },
  "item": {
    "id": 4,
    "name": "truck"
  }
}
```

Figure 16 - Label Map

Training + TensorBoard:

Now that my dataset was complete and all the input files were ready, it was time to start training. Before training, I had to create new folders for each model and configure a file called 'pipeline.config', these files are used to let the model know where to find the training and testing dataset and the other input files. I was also able to edit the batch size number using this file. Batch size is the number of samples that goes through the network in each pass, the greater the batch size the more memory is required. This hyperparameter caused problems throughout my training as many of my models would not even start training due to memory problems caused by the batch size. To solve this issue, I had to decrease the batch size significantly and eventually the models began to train but now I came across another issue, the training process took an immense amount of time and this caused my computer to get very hot and slow down over time.

Google Colab:

To combat this issue, I decided to use external hardware in the form of Google Colab. This is a service from Google that allows users to use notebooks (Jupyter Notebook style scripts) on their hardware. These systems had superior hardware that was better suited for this type of work due to difference in GPU ram sizes. I was able to gain access to Tesla T4s equipped with 16GB VRAM which turned out to be very beneficial as it significantly improved my training speeds and reduced the load on my own system. Furthermore, I had to move all my working folders onto Google Drive for easy mounting onto the Colab system. I wrote a script that would allow me to install the same libraries onto the Colab system. Whilst the move to Colab was very valuable, it did not come without problems. For example, every time I reset the Colab session or started a new one I would have to reinstall all of the libraries again. This took 5 minutes every time and it was quite inconvenient. Another problem was that with the free tier of Google Colab, access to powerful GPUs was not always guaranteed which meant I would have to wait hours and sometimes even a whole day to gain access to those configurations again.

TensorBoard:

I used a library called TensorBoard to monitor the progress of the training in real time. I was able to track certain metrics such as Total Loss. Loss is a value used in TensorFlow models to describe bad predictions that the model makes. The higher the loss number, the poorer the predictions and vice versa. That makes Loss useful to tell if the model's accuracy is improving. However, it is possible that even with low loss that the model can begin to overfit to the training dataset, so its accuracy on the testing dataset begins to decrease which is why I stopped training at regular interventions to run testing evaluation.

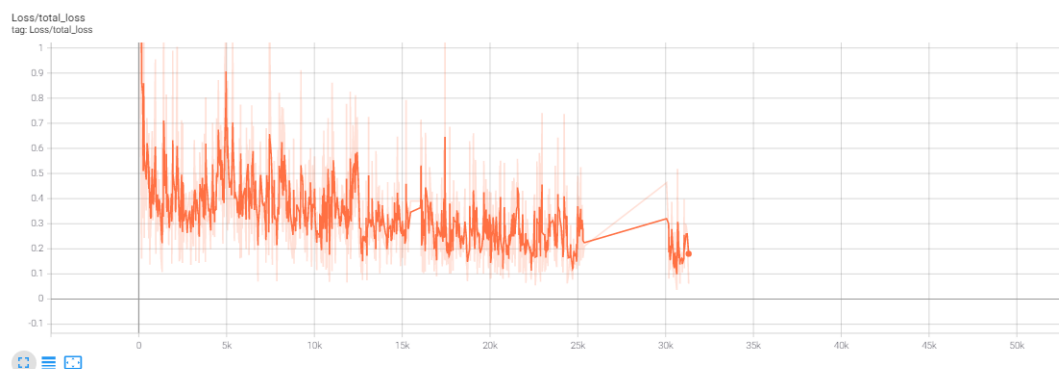


Figure 17 - TensorBoard Total Loss

After completing the training, I was able to export the completed models into a separate folder where they would be used later.

Evaluation:

To evaluate the models, I utilised two methods; automated and manual evaluation of the completed models. The first method I used was the included evaluation file within the TensorFlow library which used the testing image dataset. It ran the model on the test images and compared its predictions against the ground truth (hand-labelled) bounding boxes I created using Labellmg and stored in the TF record. The output of this evaluation was mainly a mAP score amongst other metrics. mAP (mean average precision) is the main metric used to determine the performance of an object detection model on a specific dataset and it is commonly used in competitions such as the Pascal Visual Object Challenge. It is the average of the average precision score over multiple classes.

Here are the results I found:

Model	mAP
Faster R-CNN Resnet 50	0.643126
Faster R-CNN Resnet 152	0.586296
SSD Resnet 50	0.287755
SSD Resnet 152	0.435566

It was clear using this method that the Faster R-CNN models performed significantly better than their counterpart SSD models. One abnormality was that Faster R-CNN Resnet 152 model could not achieve a higher mAP score than its Resnet 50 counterpart which was the case for the SSD models. Furthermore, I had predicted the Resnet 152 models to have superior performance due to them having a greater number of layers within their convolutional networks.

After gaining these results, I wanted to verify them utilising a manual method of evaluation, so I decided to create a new testing dataset which was completely unlabelled unlike the previous datasets I had used. They are from the same two sources as the previous data which is why these images were of a similar nature, but the model had never seen these specific samples. This new dataset consisted of 25 images with a variety of the 4 different classes to properly gauge the model's performance across the board. As this was a process that was prone to human error as I would gauge incorrect and correct predictions, I choose images in which the objects were straightforward to detect and not in awkward positions.

For the purpose of this evaluation, I wrote code that was capable of using the final exported models for inference on these images which would allow me to manually count the detections the models make.

I utilised 3 classifications for the predictions that the models made (El Aidouni 2019);

- *True Positive (TP):*

Correct prediction of a positive sample. This is when the model makes a prediction on the object that is correct. Both the bounding box and the classification of the object are correct.



Figure 18 - Detection Example 1

The image above shows two objects being correctly detected so this would count as two true positive values.

- *False Positive (FP):*

Incorrect prediction of a positive sample. The models makes a prediction on an object with a bounding box, but the classification of the object is incorrect. It can also be when the model makes multiple bounding boxes for a single object, the bounding box with the highest confidence is counted as TP if correct but the rest are considered FP.



Figure 19 - Detection Example 2

We can see in this images that both the truck and the vans are correctly detected which counts a 3 TP values, however there is second bounding box around the truck. This second bounding box has a confidence score of 85% which is the lower of the two, so it is counted as a false positive value.

- *False Negative (FN):*

False/non prediction of a positive sample. This is when the model doesn't make a correct prediction on an object. For example, a car that has no bounding box around it at all would be a FN or car that is predicted as a van would count as an FN as the car was not detected correctly and a FP due to incorrect prediction.



Figure 20 - Detection Example 3

In the image above, we can see that the only correct prediction made is the silver car on the left, whilst model fails to detect the bus in the foreground and the car in the background. This would count as two FN values and one TP.

True negatives are not used in this project as they indicate every part of the image where there is no object and no box. Essentially, no box for a non-object which could end up with 100s or 1000s of TN values as there is no way to pin down how large these “empty” boxes would be. This information is useless and adds no value to calculating the performance on these models, so it was removed as a classification.

I was able to calculate these values for each of the models utilising the code I previously mentioned. The code ran the models on each of the images only presenting a bounding box when its confidence threshold was above 0.6 (60% confident in its predictions). This also allowed me to create a confusion matrix.

In addition to the confusion matrix, I also calculated precision, recall and accuracy values and here are the corresponding formulas:

$$\text{Accuracy} = \frac{(TP + TN)}{(TP + TN + FP + FN)}$$

$$\text{Precision} = \frac{TP}{(TP + FP)}$$

$$\text{Recall} = \frac{TP}{(TP + FN)}$$

Accuracy is a measure that demonstrates how often the model made correct predictions out of all of the predictions it made. However, this is typically not a good metric for model performance as it can heavily be swayed by class imbalances such as the ones present in my own datasets which contain more cars than any other classes.

Precision on the other hand, revolves around the bounding boxes that model creates. It is a measure of how accurate the bounding boxes are relative to the objects in the image.

Recall is also known as the true positive rate, so it is a measure of how many objects are being correctly detected in the image.

[Faster R-CNN Resnet 50:](#)

<i>Image</i>	<i>TP</i>	<i>FP</i>	<i>FN</i>
1	2	1	0
2	4	0	0
3	2	2	0
4	1	4	1
5	3	0	0
6	6	0	0
7	2	0	0
8	1	2	0
9	2	1	0
10	2	1	0
11	3	1	0
12	2	0	0
13	2	0	1
14	4	0	0
15	3	1	0
16	4	0	0
17	4	0	0
18	3	0	0
19	2	0	1
20	2	0	0
21	2	0	0
22	2	0	0
23	4	0	1
24	2	0	1
25	6	1	3
	70	14	8

Faster R-CNN Resnet 50:

Precision = $70 / (70 + 14) = 0.833$

Recall = $70 / (70 + 8) = 0.897$

Accuracy = $(70) / (70 + 14 + 8) = 0.761$

TP = 70	FP = 14
FN = 8	

Faster R-CNN Resnet 152:

Image	TP	FP	FN
1	2	0	0
2	4	0	0
3	2	1	0
4	1	2	2
5	3	0	0
6	5	1	0
7	2	0	0
8	1	0	0
9	2	2	0
10	2	1	0
11	3	0	0
12	2	0	0
13	2	0	1
14	4	0	0
15	3	1	0
16	4	0	0
17	4	0	0
18	3	0	0
19	2	0	1
20	2	0	0
21	2	0	0
22	2	1	0
23	5	0	0
24	2	0	1
25	6	1	3
	70	10	8

Faster R-CNN Resnet 152:

Precision = $70 / (70 + 10) = 0.875$

Recall = $70 / (70 + 8) = 0.897$

Accuracy = $(70) / (70 + 10 + 8) = 0.795$

TP = 70	FP = 10
FN = 8	

SSD Resnet 50:

<i>Image</i>	<i>TP</i>	<i>FP</i>	<i>FN</i>
1	2	0	0
2	2	0	2
3	1	0	1
4	0	1	3
5	1	0	2
6	4	1	2
7	2	0	0
8	1	0	0
9	2	0	0
10	2	0	0
11	1	0	2
12	2	0	0
13	0	0	3
14	1	0	3
15	2	0	1
16	3	0	1
17	3	0	1
18	1	0	2
19	2	0	1
20	1	0	1
21	1	1	1
22	2	0	0
23	3	0	2
24	2	0	1
25	1	0	7
	42	3	36

SSD Resnet 50:

Precision = $42 / (42 + 3) = 0.933$

Recall = $42 / (42 + 36) = 0.538$

Accuracy = $(42) / (42 + 3 + 36) = 0.518$

TP = 42	FP = 3
FN = 36	

SSD Resnet 152:

<i>Image</i>	<i>TP</i>	<i>FP</i>	<i>FN</i>
1	1	0	1
2	2	0	2
3	1	0	1
4	0	1	3
5	2	0	1
6	4	0	2
7	1	0	1
8	1	0	0
9	1	0	1
10	2	0	0
11	1	0	2
12	2	0	0
13	0	0	3
14	1	1	3
15	2	0	1
16	2	0	2
17	3	0	1
18	3	0	0
19	2	0	1
20	2	0	0
21	2	0	0
22	2	0	0
23	2	0	3
24	1	0	2
25	1	0	7
	41	2	37

SSD Resnet 152:

Precision = $41 / (41 + 2) = 0.953$

Recall = $41 / (41 + 37) = 0.526$

Accuracy = $(41) / (41 + 2 + 37) = 0.512$

TP = 41	FP = 2
FN = 37	

	Faster R-CNN Resnet 50	Faster R-CNN Resnet 152	SSD Resnet 50	SSD Resnet 152
mAP	0.643126	0.586296	0.287755	0.435566
Accuracy	0.761	0.795	0.518	0.512
Precision	0.833	0.875	0.933	0.953
Recall	0.897	0.897	0.538	0.526

We can see that the results from my manual evaluation of the models mirrors that of the automated evaluation as the Faster R-CNN models vastly outperform their SSD counterparts. The best performing model in automated evaluations was Faster R-CNN Resnet 50 but in the manual evaluation it turned out to be Faster R-CNN Resnet 152 which was my original prediction as the best model. It achieved the highest accuracy and also gained high precision and recall scores.

High precision and high recall mean that the model performed very well and correctly predicted most objects in the images which was the case for both the Faster R-CNN models. On the other hand, the two SSD models did not achieve this result, instead gaining high precision scores but low recall scores. This means that these models were unable to detect a large number of objects within the images but the ones that were detected were mostly correct predictions. This heavily impacted the accuracy of the SSD models as well. One reason for the relatively poor performance of the SSD models could be due to the confidence threshold I set in my code being 60% (`min_score_thresh = 0.6`). This means that the model might've made more predictions than what was shown in the images but those were not presented as the confidence the model had in its prediction was less than 60%. It is likely with a lower threshold; the SSD models would perform slightly better but then it would not be a fair comparison between the models.

Discussion:

During this project, I have accomplished the majority of the goals I set out to complete. I was able to create multiple models capable of vehicle detection at varying levels of accuracy. I was also able to thoroughly evaluate the performance of these models. I came across numerous problems during this project. One of those problems was the time spent setting up the Tensorflow environment, making the GPU element of the Tensorflow library work was very inconvenient due to many incompatibility issues between the packages.

After a long time, I was able to get it working only to come across GPU memory issues due to the relatively high resolution of the images. This caused many errors with the models training that could only be fixed by making the batch size smaller which also made the training a lot slower whilst my systems temperature and power consumption increased.

Evaluation went well and the models performed as expected, I utilised two methods; one automated and one manual with both showing the same pattern. However, I felt that because of the class imbalances in the dataset, the performance metrics (precision, accuracy, and recall) might not have been representative of the model's overall performance across all classes. Another testing dataset consisting of more balanced classes might have been more suitable.

Management / Reflection:

Kanban Board:

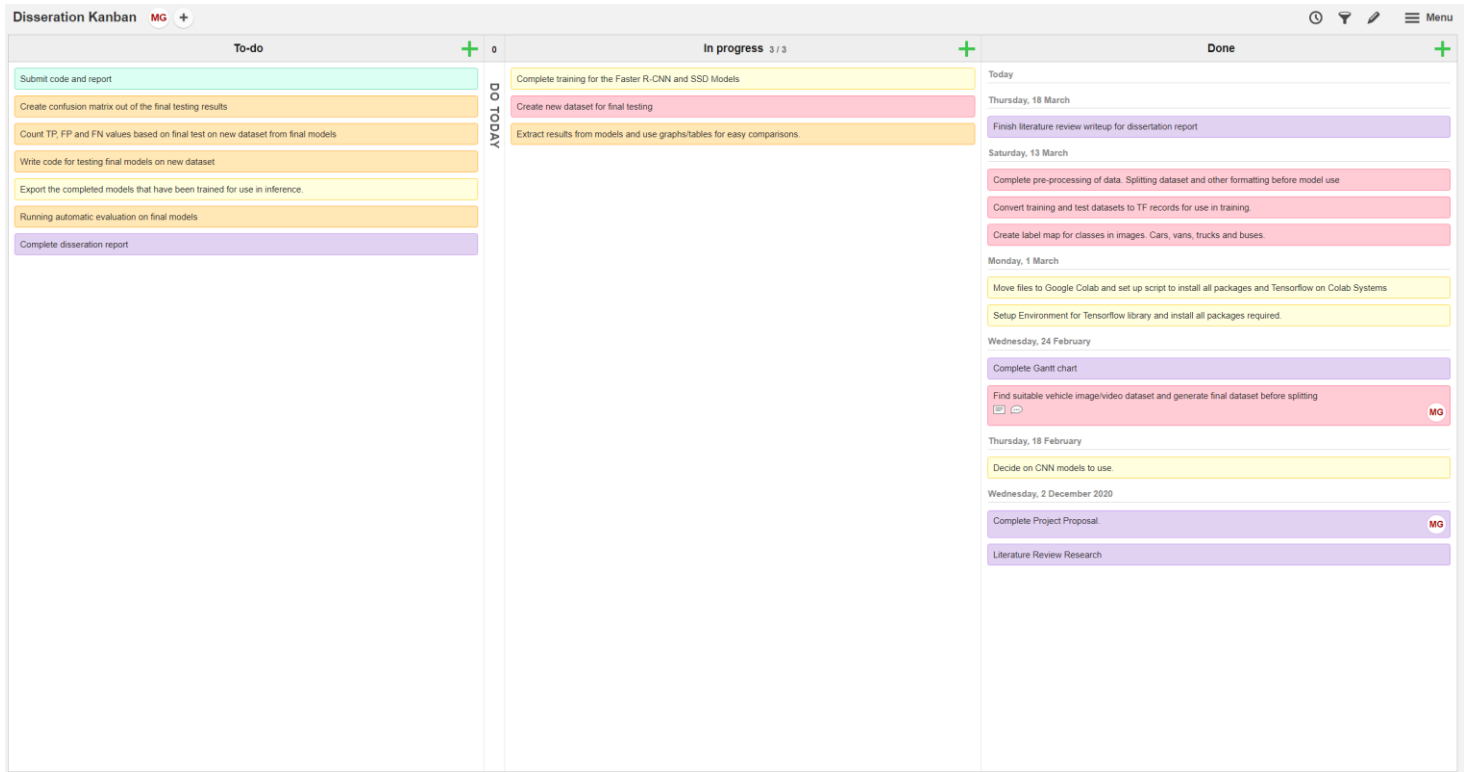


Figure 21 - Kanban Board

Throughout this project, I utilised an agile project management tool called a Kanban board for the purpose of easily tracking the many tasks I had to complete. It allowed me to focus on specific tasks and create a simple roadmap/plan for the future. I color coded each type of task, for example the purple tasks revolved around the external parts of my project (not training/code) such as the report, proposal, and the literature review. The red tasks involved the preparation of data that would be used in the project such as collecting images for the dataset and pre-processing. The yellow tasks were about the actual project tasks such as the training and setting up the environments used. Orange tasks were for generating results and the evaluation of the completed models. This Kanban board allowed for great flexibility and easy visualisation of the project's progress.

Gantt Chart:

I also utilised a Gantt chart throughout this project so that I could plan out the amount of time I would use on each stage of the project (e.g. preparing data, setting up environment, training models etc...). This allowed for greater time management and easy separation of the tasks that I had to complete very much like the Kanban board I had also used.

Dissertation Gantt Chart

* - an automatically calculated cell

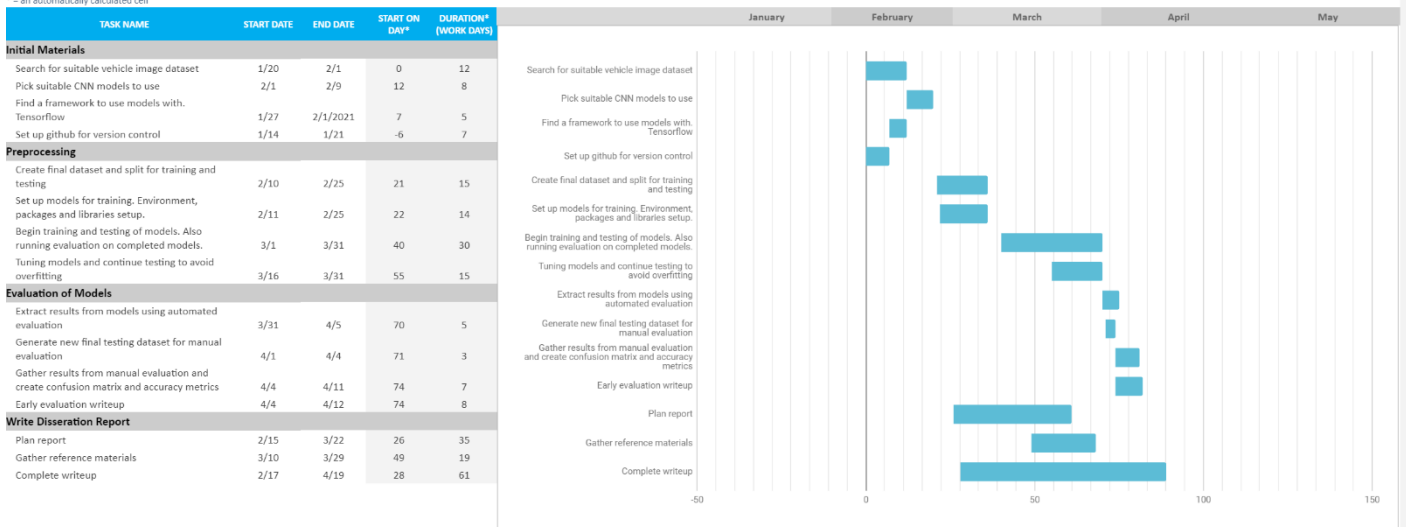


Figure 22 - Gantt Chart

I had wanted to utilise my GitHub repository for this project, but a lot of the files used were larger than 250mb (e.g. the models used in training were multiple GBs large and the images used were also above the 200mb file limit size). So, the files uploaded to my GitHub were very limited and the code I did write was in the Jupyter/Colab notebook format. Instead, I decided to host the final working files on my university OneDrive account.

Social, Legal and Ethical Issues:

As stated in the ethics proposal, this project does not involve any personal information of any third parties that is not already public domain. The two sources I had utilised for my datasets were both available publicly for use (e.g. the Flickr source required credit for external use which was given in the bibliography). The images were taken in a public environment so there are no legality issues in that regard either. None of the assets in this project are to be used in commercial sense. Majority of the picture do not include people as the focus and only vehicles.

Presentation Feedback:

The overall feedback I gained for my project was that it was good with the one criticism of it being that I did not include any introduction and went straight into the technical details of the project. I was also recommended to include the details of my project management and agile methodology into the presentation.

Conclusion:

In conclusion, I was able to accomplish the goal I set out to during this project. I was able to create and train object detection models capable of detecting vehicle in a wide variety of inner-city environments. I was also able to evaluate and compare the performance between these models. One goal that I was not able to accomplish was the counting of vehicle within a specific location. This was due to the lack of inner-city video footage in similar environments which would've been required on top of all the images.

Another reason for this was that I would have to tinker with the TensorFlow library which I already knew was very unstable as small changes to it had caused many errors early on in the project so I left this idea behind and decided to focus on the performance aspect of the project.

Bibliography:

1. Hadi, R., Sulong, G. and George, L. (2014) "Vehicle Detection And Tracking Techniques: A Concise Review". *Signal & Image Processing: An International Journal* 5 (1), 1-12
2. Franks, N. (2016) [online] available from <<https://www.london.gov.uk/about-us/londonassembly/meetings/documents/s61576/Traffic%20congestion%20report.pdf>> [12 November 2020]
3. Foote, K. (2019) *A Brief History Of Machine Learning - DATAVERSITY* [online] available from <<https://www.dataversity.net/a-brief-history-of-machine-learning/#:~:text=Arthur%20Samuel%20first%20came%20up,%E2%80%9CMachine%20Learning%E2%80%9D%20in%201952.&text=In%201957%2C%20Frank%20Rosenblatt%20%E2%80%93%20at,efforts%20and%20created%20the%20perceptron.>>> [2 April 2021]
4. Tzanis, G., Katakis, I., Partalas, I. and Vlahavas, I. (2006) *Modern Applications Of Machine Learning* [online] available from <https://www.researchgate.net/publication/228340464_Modern_Applications_of_Machine_Learning> [3 April 2021]
5. Mishra, S. (2017) *Unsupervised Learning And Data Clustering* [online] available from <<https://towardsdatascience.com/unsupervised-learning-and-data-clustering-eeecb78b422a>> [3 April 2021]
6. Priy, S. (2020) *Clustering In Machine Learning - Geeksforgeeks* [online] available from <<https://www.geeksforgeeks.org/clustering-in-machine-learning/>> [3 April 2021]
7. Wagle, M. (2020) *Association Rules: Unsupervised Learning In Retail* [online] available from <<https://medium.com/@manilwagle/association-rules-unsupervised-learning-in-retail-69791aef99a>> [3 April 2021]
8. Ng, A. (2016) *Association Rules And The Apriori Algorithm: A Tutorial - Kdnuggets* [online] available from <<https://www.kdnuggets.com/2016/04/association-rules-apriori-algorithm-tutorial.html>> [3 April 2021]
9. Bajaj, P. (2020) *Reinforcement Learning - Geeksforgeeks* [online] available from <<https://www.geeksforgeeks.org/what-is-reinforcement-learning/>> [4 April 2021]
10. Lee, D., Seo, H. and Jung, M. (2012) "Neural Basis Of Reinforcement Learning And Decision Making". *Annual Review Of Neuroscience* 35 (1), 287-308
11. Prasad, S. (2020) *Types Of Clustering Algorithms In Machine Learning With Examples* [online] available from <<https://www.analytixlabs.co.in/blog/types-of-clustering-algorithms/>> [6 April 2021]
12. Kumar, S. (2020) *Hierarchical Clustering: Agglomerative And Divisive — Explained* [online] available from <<https://towardsdatascience.com/hierarchical-clustering-agglomerative-and-divisive-explained-342e6b20d710>> [6 April 2021]
13. Marinova-Boncheva, V. (2008) *USING THE AGGLOMERATIVE METHOD OF HIERARCHICAL CLUSTERING AS A DATA MINING TOOL IN CAPITAL MARKET* [online] available from <<https://core.ac.uk/download/pdf/62657622.pdf>> [12 April 2021]
14. Sreenath14 (2020) *Reinforcement Learning Via Markov Decision Process* [online] available from <<https://www.analyticsvidhya.com/blog/2020/11/reinforcement-learning-markov-decision-process/>> [12 April 2021]
15. Waseem, M. (2020) *Classification In Machine Learning | Classification Algorithms | Edureka* [online] available from <<https://www.edureka.co/blog/classification-in-machine-learning/#:~:text=In%20machine%20learning%2C%20classification%20is,recognition%2C%20document%20classification%2C%20etc.>>> [13 April 2021]

16. Sathya, R. and Abraham, A. (2013) "Comparison Of Supervised And Unsupervised Learning Algorithms For Pattern Classification". *International Journal Of Advanced Research In Artificial Intelligence* 2 (2)
17. Point, J. (2021) *Logistic Regression In Machine Learning - Javatpoint* [online] available from <<https://www.javatpoint.com/logistic-regression-in-machine-learning>> [13 April 2021]
18. Gupta, P. (2017) *Naive Bayes In Machine Learning* [online] available from <<https://towardsdatascience.com/naive-bayes-in-machine-learning-f49cc8f831b4>> [13 April 2021]
19. Schott, M. (2019) *Naives Bayes Classifiers For Machine Learning* [online] available from <<https://medium.com/capital-one-tech/naives-bayes-classifiers-for-machine-learning-2e548bfbd4a1>> [13 April 2021]
20. Raina, H. and Shafi, O. (2015) *Analysis Of Supervised Classification Algorithms* [online] available from <<https://www.ijstr.org/final-print/sep2015/Analysis-Of-Supervised-Classification-Algorithms.pdf>> [14 April 2021]
21. Chakure, A. (2019) *Decision Tree Classification* [online] available from <<https://medium.com/swlh/decision-tree-classification-de64fc4d5aac>> [14 April 2021]
22. Harrison, O. (2018) *Machine Learning Basics With The K-Nearest Neighbours Algorithm* [online] available from <<https://towardsdatascience.com/machine-learning-basics-with-the-k-nearest-neighbors-algorithm-6a6e71d01761>> [14 April 2021]
23. UYSAL, İ. and GÜVENİR, H. (1999) "An Overview Of Regression Techniques For Knowledge Discovery". *The Knowledge Engineering Review* 14 (4), 319-340
24. Malik, F. (2019) *What Are Hidden Layers?* [online] available from <<https://medium.com/fintechexplained/what-are-hidden-layers-4f54f7328263>> [16 April 2021]
25. Saha, S. (2018) *A Comprehensive Guide To Convolutional Neural Networks — The ELI5 Way* [online] available from <<https://towardsdatascience.com/a-comprehensive-guide-to-convolutional-neural-networks-the-eli5-way-3bd2b1164a53>> [16 April 2021]
26. Jordan, J. (2017) *Convolutional Neural Networks*. [online] available from <<https://www.jeremyjordan.me/convolutional-neural-networks/>> [16 April 2021]
27. R. Girshick, J. Donahue, T. Darrell, and J. Malik (2014) "Rich Feature Hierarchies for Accurate Object Detection and Semantic Segmentation," IEEE Conference on Computer Vision and Pattern Recognition [16 April 2021]
28. R. Girshick, "Fast R-CNN," 2015 IEEE International Conference on Computer Vision (ICCV), [16 April 2021]
29. pawangfg (2020) R-CNN Vs Fast R-CNN Vs Faster R-CNN | ML - Geeksforgeeks [online] available from <<https://www.geeksforgeeks.org/r-cnn-vs-fast-r-cnn-vs-faster-r-cnn-ml/>> [18 April 2021]
30. Ren, S., He, K., Girshick, R. and Sun, J. (2017) "Faster R-CNN: Towards Real-Time Object Detection With Region Proposal Networks". *IEEE Transactions On Pattern Analysis And Machine Intelligence* 39 (6), 1137-1149 [18th April 2021]
31. Yadav, N. and Binay, U. (2017) "Comparative Study Of Object Detection Algorithms". *International Research Journal Of Engineering And Technology (IRJET)* 04 (11)
32. Kang, Y. (2020) "Research On SSD Base Network". *IOP Conference Series: Materials Science And Engineering* 768, 072031 [18th April 2021]
33. Vladimirov, L. (2020) *Training Custom Object Detector — Tensorflow 2 Object Detection API Tutorial Documentation* [online] available from <<https://tensorflow-object-detection-api-tutorial.readthedocs.io/en/latest/training.html>> [19 April 2021]

34. Bileschi, S. (2007) *CBCL Streetscenes Database* [online] available from <<http://cbcl.mit.edu/software-datasets/streetscenes/>> [22 April 2021]
35. Richardson, G. (2021) *Graham Richardson Albums* [online] available from <<https://www.flickr.com/photos/didbygraham/>> [22 April 2021]
36. El Aidouni, M. (2019) *Evaluating Object Detection Models: Guide To Performance Metrics* [online] available from <<https://manalelaidouni.github.io/Evaluating-Object-Detection-Models-Guide-to-Performance-Metrics.html#predictions-tp---fp---fn>> [24 April 2021]


Appendix:

Presentation Slides:



TensorFlow

+ The API I used for this project and also the backbone, it allowed for the training of object detection models on the image dataset I had created. Also the use of external libraries that allowed for evaluation.



TensorFlow

Setup

- + I used Jupyter Anaconda to create a virtual environment.
- + This environment was used to install all the necessary packages and libraries for training the models whilst also keeping it separate from the rest of my system. This helps avoid any incompatibility problems with pre-installed packages on my PC.

Hardware Specification:

- + CPU: Ryzen 5 3600
- + GPU: GeForce RTX 2070 Super
- + RAM: 16GB DDR4

Dataset Collection

Had to collect 100s of images from the internet that fulfilled certain conditions:

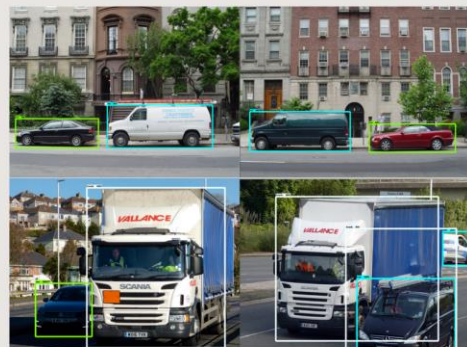
- + Not too high resolution (close to 1000x1000)
- + Had to revolve around inner city environments
- + Within rights to use image without paying/explicit permission
- + Large amount of images with a wide variety of settings, angle and vehicle types.

A single dataset that contained all the classes and fulfilled all the above requirements was difficult to find so I had to utilise a mixture of two online datasets to create my own.

- + <http://cbcl.mit.edu/software-datasets/streetscenes/> - Primary source used. Cars and vans mainly.
- + <https://www.flickr.com/photos/didbygraham/> - Secondary source used. Trucks and Buses.

Final Dataset:

- + 415 images in total.
- + Ratio = 9:1 - Train:Testing.
- + ~ 375 training images, ~41 testing images.



Final Dataset Issues.

- + 415 images in total
- + Ratio = 9:1 - Train:Testing
- + Majority of images contained cars and so the dataset overall has a heavy bias towards cars over the other classes.
- + Van and buses are far rarer in the dataset which might cause accuracy issues during training due to insufficient samples.
- + This turned out to be the case as my final models were much better at predicting cars and vans over the bus/truck objects in the images.

Labelling and Preparing Images.

- + Each and every image in the dataset had to be labelled utilising an software called Labellmg,
- + 4 separate classes that had to be labelled accurately across all 415 images. Very time consuming process to do manually.
- + Upon completing labelling, I used a script from the TensorFlow repo I downloaded to create a Label Map which is what shows the model which label correlates to each class.
- + Furthermore, I used another script to convert the training and testing datasets into TF record which is the necessary format for TensorFlow.



Training

- + After preparing the dataset, i had to download the models that would be used for the project. I decided on using:
 - + Faster R-CNN ResNet 50
 - + Faster R-CNN ResNet 152
 - + SSD Resnet 50
 - + SSD Resnet 152
- + I then configured the configuration files for each model linking them to the image datasets and label map that would be used.
- + I trained one model on my own system before realising that it took an incredibly long time to train these models. Not only because of batch size errors but also because of the complexity of the images within the dataset. A variety of different shaped cars and environments which most likely cause the models to struggle while training.

Google Colab

I came across batch size errors during my training which caused very slow training that would cause my PC to get too hot over long hours and using a lot of power. I decided to utilise external hardware in the form of Google Colab. This is a service that allows users to use notebooks (code/scripts similar to Jupyter anaconda layout) on their own systems that included more powerful hardware than my own. For example, I gained access to a Tesla T4 with 16GB VRAM (twice my own) and this proved to be incredibly beneficial as it significantly improved training speeds and reduced the load on my own system.

I moved all my working folders and contents over to my personal Google Drive and wrote a script that allowed me to install all the same libraries and packages I had on my personal system. One problem was that I would have to do this every single time I used Google Colab which proved to be inconvenient as it took 5 minutes to setup every time. Any errors during the training would require me to restart the session and setup again. Furthermore, access to powerful GPUs was not always guaranteed which meant sometimes I would have to wait hours or even a whole day before getting the configuration I wanted otherwise I could no longer train.



Evaluation Part 2

I created a new testing dataset that was completely unlabelled unlike the previous datasets I used. I wrote some code that was capable of using the final exported models for inference on the new image dataset. This dataset included 25 images from the two sources that I used earlier but didn't use in the prior datasets. I created 4 tables for each model to count the number of TP, FP and FN.

TP = Box with correct prediction.

FP = Box with incorrect prediction or object with more than one box on it.

FN = Object that isn't detected at all.

True negatives are not used in this project as they indicate every part of the image where there is no object and no box. This information is useless and doesn't contribute to the accuracy of the model. The boxes for each detection are only shown if a specific confidence threshold is surpassed (threshold is how confident the model is in its prediction of the object). This threshold is 0.6 in my code.



Confusion Matrix

- + Using these values I was able to create 4 confusion matrices for each model.

Confusion Matrices:
Faster RCNN ResNet 50

Image	TP	FP	FN
1	2	2	0
2	4	0	0
3	2	2	0
4	1	4	1
5	3	3	0
6	6	0	0
7	2	0	0
8	1	2	0
9	2	0	0
10	2	1	0
11	3	1	0
12	2	0	0
13	2	0	1
14	4	0	0
15	3	1	0
16	4	0	0
17	4	0	0
18	3	0	0
19	2	0	1
20	2	0	0
21	2	0	0
22	2	0	0
23	4	0	1
24	2	0	1
25	6	1	3



This is how I counted the confusion matrix values for each image in the dataset.

TP = 70	FP = 14
FN = 8	

Faster R-CNN Resnet 50

TP = 42	FP = 3
FN = 36	

SSD Resnet 50

TP = 70	FP = 10
FN = 8	

Faster R-CNN Resnet 152

TP = 41	FP = 2
FN = 37	

SSD Resnet 152

Confusion Matrix Values

- + Recall is also known as the true positive rate, so a measure of how many objects are being correctly detected in the image by the model. Precision on the other hand is simply how accurately the model detects the objects that exist, not the class of the object but simply detecting that the object is there and box that it generates. Accuracy is another metric that calculates the percentage of correct predictions out of all the predictions, however this can be skewed by class imbalances in the data. For example, all my datasets having a heavy bias towards cars over the other 3 classes.

Faster R-CNN Results

- + Accuracy = $(TP+TN) / (TP+FP+TN+FN)$
- + Precision = $TP / (TP + FP)$
- + Recall = $TP / (TP + FN)$

TP = 70	FP = 14
FN = 8	

Faster R-CNN Resnet 50:

$$\text{Precision} = 70 / (70 + 14) = 0.833$$

$$\text{Recall} = 70 / (70 + 8) = 0.897$$

$$\text{Accuracy} = (70) / (70 + 14 + 8) = 0.761$$

High precision and high recall meaning that this model performed very well and correctly predicted most objects within the image.

TP = 70	FP = 10
FN = 8	

Faster R-CNN Resnet 152:

$$\text{Precision} = 70 / (70 + 10) = 0.875$$

$$\text{Recall} = 70 / (70 + 8) = 0.897$$

$$\text{Accuracy} = (70) / (70 + 10 + 8) = 0.795$$

High precision and high recall meaning that this model performed very well and correctly predicted most objects within the image.

SSD Results

- + Accuracy = $(TP+TN) / (TP+FP+TN+FN)$
- + Precision = $TP / (TP +FP)$
- + Recall = $TP / (TP + FN)$

TP = 42	FP = 3
FN = 36	

SSD Resnet 50:

Precision = $42 / (42 + 3) = 0.933$

Recall = $42 / (42 + 36) = 0.538$

Accuracy = $(42) / (42 + 3 + 36) = 0.518$

High precision but low recall meaning that the model missed a large number of objects in the model but the ones that were detected were mostly correct predictions.

TP = 41	FP = 2
FN = 37	

SSD Resnet 152:

Precision = $41 / (41 + 2) = 0.953$

Recall = $41 / (41 + 37) = 0.526$

Accuracy = $(41) / (41 + 2 + 37) = 0.512$

High precision but low recall meaning that the model missed a large number of objects in the model but the ones that were detected were mostly correct predictions.

Supervisor Meetings:

Supervisor Meetings Logs:

Supervisor: Nazaraf Shah

Meeting 1 (22/01/2021):

Discussed:

- <https://boxy-dataset.com/boxy/>
- This will be the dataset I will attempt to use for my project.
- First use Faster R-CNN and complete vehicle detection on this model before moving on to the next model.
- Finish 2 weeks before deadline.

Next time:

- Complete the dataset gathering and start labelling for next meeting.

Notes for this week:

- <http://cbcl.mit.edu/software-datasets/streetscenes/>
- This is the dataset of images I ended up using. Lower res images so easier for training the model. Also, far smaller file size so easier to use/transport.
- 300 images used overall.

Meeting 2 (08/02/2021):

Discussed:

- Dataset used has been changed.
- Have 300 images so far in training but only cars and vans are included.
- Must have minimum of 3 classes before training.
- Will add motorbikes and lorries to the dataset.

Next Time:

- Will finish dataset gathering and my first faster R-CNN model training complete by next meeting.

Meeting 3 (19/02/21):

Discussed:

- Faster RCNN model not as accurate as expected. Reasons being training didn't finish it seems. Should run for longer and re-evaluate progress.

Next Time:

- Start lit review for dissertation report from now on.

Meeting 4 (07/04/21):

Discussed:

- Completed training of all models and ran built in evaluation of all trained models.
- Switched to Google Colab by writing up a script and transferring all files to Google Drive.
- This was due to hardware limitations caused by RTX 2070 Super VRAM. Could not train numerous models at a decent speed due to small batch size. Gained access to far more powerful graphics card which improved the training process immensely. Also wrote code for active inference of the final models on any image.
- Creating final eval dataset of new images for confusion matrix and other metrics from the final models.
- Mid literature review for report.

Next Meeting:

- Next meeting will be presentation for this project.
- Make haste with dissertation report.

Meeting 5 (21/04/21):

Discussed:

- Final meeting with supervisor
- Presented my project.
- Received feedback on the presentation:
 - Did not give a proper introduction into the project and instead went straight into technical details.
 - Other than that, presentation was good and thorough.
- Finishing up report for submission.

GitHub Link:

- <https://github.coventry.ac.uk/guledm2/vehicledetectionproject>

OneDrive Link:

- https://livecoventryac-my.sharepoint.com/:f/g/personal/guledm2_uni_coventry_ac_uk/Ekjp5XibDQNK1NPWTD4bzAB4jhx0hBjg4TjK9hytU9pw?e=djrBdi
- -This link contains the files for the final exported models after training. It also includes the image datasets used throughout the project and their corresponding TF Records.