Maynooth University
National University of Ireland Maynooth

EE297 Final Report

By

EE297, Group 2

Chizu Ulogwara - 18306543

Tinuola Laotan Brown - 18302983

Donovan Webb - 19330681

Declaration | Acknowledgements

We hereby declare that the project entitled “FINAL REPORT” that has been submitted by Team 2 (Chizu Ulogwara, Donovan Conor Weiss Webb and Tinuola Brown) to Maynooth University is our original work. Where others work HAS been used but with reference. We confirm that we have not used work previously produced by another student or any other person to hand in as our own. We, therefore, will not allow anyone to copy our work with the intention of passing it as his/her own.

From the outset and throughout this report, we would like to extend our sincere and yet heartfelt obligation towards all the different people who have helped us in one way or another. Without their active and continuous guidance, help, co-operation, and encouragement, we honestly would not have made headway in our project.

We are ineffably indebted to both our lecturer and Dr. John Dooley for continued guidance and encouragement to accomplish this project.

We extend our gratitude to MAYNOOTH UNIVERSITY for giving us this opportunity.

We also acknowledge with a profound sense of appreciation our gratitude towards our parents and members of our families, who have supported us morally.

Finally, we would like to extend the gratitude to our friends who have helped us both, directly and indirectly, to get to the point we are at in our project.

Thanking You,Team 2

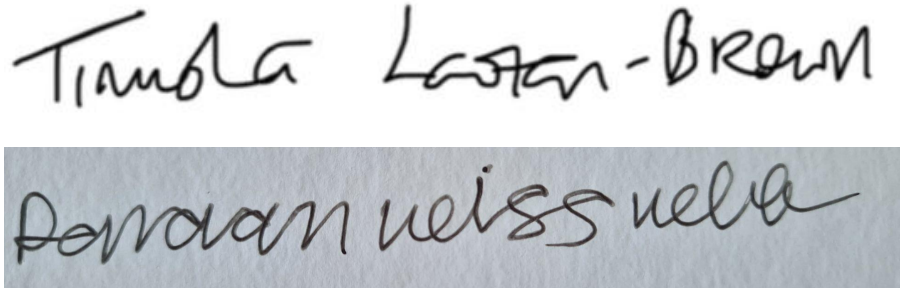


Table of Contents

List of Abbreviations

TF	Tensor Flow
CNN	Convolutional Neural Network
AI	Artificial Intelligence
GPU	Graphics Processing Unit
SSD	Single Shot MultiBox Detector
API	Application Programming Interface
MLP	Multi Layered Perceptron
IOU	Intersection over Union
ANN	Artificial Neural Network

1. Introduction

In this project, we as a team were required to implement a neural network to perform one of the following tasks:

1. Classification
2. Clustering
3. Association
4. Prediction

Not only were we tasked with the implementation of a neural network to perform the task of our choosing, but we were also tasked with understanding the underlying theory behind how what we implemented works.

1.1 Our Aim

Implement a neural network to perform classification, clustering, association or prediction.

1.2 Machine Learning

What is *Machine Learning?*

Machine learning is a branch of artificial intelligence aka (AI) that allows computers to learn and develop independently without being programmed. Machine learning is concerned with developing computer programmes that can access data and learn on their own. An algorithm is a set of statistical processing in steps that are used in data science. These algorithms are trained in machine learning to identify patterns and features in specific quantities successfully. This is so that they can further make predictions as well as decisions based on the new data. The more the data is processed, the faster and easier it is to train the algorithm, and the more accurate and reliable its predictions become.

In the world today, we have multiple examples of machine learning. Devices such as Alexa and google assistant surf the internet and play the music we request because of our voice commands. The websites and apps that we spend our time on will recommend things to us based on what we have watched, bought or listened to. The exciting thing is that this is only the beginning.

A Brief History into Machine Learning.

In 1943 the first mathematical model version of neural networks was shown in a scientific paper called “A logical calculus of the ideas immanent in nervous activity”. This scientific paper was by two men, Walter Pitts and Warren McCulloch.
Fast forward to the 1950s where Arthur Samuel made a program for playing championship-level computer checkers. Samuel was known as a pioneer in machine learning. Some even know him as the “Father of Machine Learning”, as he popularised the term “Machine Learning”. [1]

In 1958 a man called Frank Rosenblatt designed the first perceptron. The purpose of this was to recognise patterns and shapes. In 1965 the scientists Alexey Ivankhnenko and Valentin Lapa created a positional representation of a neural network. This was considered to be the first multi-layer perceptron (MLP). Oftentimes Alexey is referred to as the “Father of Deep Learning”.

In 1997 a monumental moment happened. Deep Blue, the IBM chess computer, beat the world champion (Garry Kasparov). This was taken and seen as proof that machines were indeed “catching up to human intelligence.” [1]

In 2012 the “Google Brain” developed by Google’s X Lab research team became outstandingly good at image processing. Two years later, the Facebook research team developed a deep learning facial recognition system. This was known as “Deep Face”. It is a nine-layer neural network trained on 4 million images of people who used Facebook (maybe even you). Deep Face was said to have recognised human faces in images with the same accuracy that even us humans do![2]

2014 seemed to be quite a big year as we also had the first chatbot to have passed the Turing test. Its name was Eugene Goostman. He was developed by three programmers that go by Vladimir Veselov, Eugene Demchenko and Sergey Ulasen. In a Turing test competition at the Royal Society, Eugene was victorious after successfully tricking 33% of the judges into thinking he was a human. [1]

How does Machine Learning *work?*

In general, the learning process necessitates massive quantities of data to provide an anticipated response, given specific inputs. Each input or response pair is an example, and the algorithm learns faster when there are more examples. This is because each input or response pair is contained within a problem domain defined by a line, cluster, or other statistical representation. The better the model has learned from the data inputs, the more accurately it can generate correct responses. [3]–[5]

Training Algorithm

There are a variety of ways that we could use to train our program more effectively and accurately. The main training program that was used to train our code was TensorFlow.

What is TensorFlow?

TensorFlow is a machine learning software library. It can be used for a variety of activities, but it focuses on deep neural network training. It uses Python, which is what we wrote our code in. Python provides a user-friendly API for developing applications within the framework.

Deep neural networks for handwritten digit detection, image recognition, object detection can be trained and run using TensorFlow. It also allows for large-scale output prediction using the same models that were used for testing. It enables developers to create data flow graphs. TensorFlow manages the architecture of the AI and mathematical operations.

The learning methods in machine learning also fall into categories. Mainly three, which are as follows.

1. Supervised Machine Learning
2. Unsupervised Machine Learning
3. Semi-Supervised Learning

Supervised Machine Learning is defined by the way it uses labelled datasets to train algorithms. It then puts the data into different classes that predict the outcomes as accurately as it can. The majority of practical machine learning uses supervised learning.

The main goal of supervised learning is to estimate the mapping of the function so well that when data is inputted, that it can predict the output variables for the data inputted. It is called supervised learning because the way the algorithms learn from the training dataset is similar to a teacher supervising a process. The algorithm makes predictions on the training data and is immediately corrected by the “teacher”. The learning is terminated when the algorithm gets to a reasonable standard of performance.[6][7][8]

Supervised learning problems can be further put into two groups, regression and classification.

- Classification is when the output variable is a category, e.g. “dog” or “cat”.
- Regression is when the output variable is a real value, e.g. “dollars” or “weight”.

Unsupervised Machine Learning is defined as almost the opposite of supervised learning. Unsupervised Learning uses unlabeled data and from that data recognises patterns. This helps a lot for clustering and association type problems. In this technique, the users do not need to supervise the learning. Unsupervised learning allows users to achieve more complex tasks. However the outcome can be quite unpredictable.[9][10][11][12]

Unsupervised learning problems can be further put into two groups, clustering and association.

- Clustering is where it is wanted to discover the groupings in the data, like putting customers in groups by their purchase behaviour.
- Association is where it is wanted to discover rules that describe large portions of the data. For example, people that buy X also tend to buy Y.

Semi-Supervised Learning is a combination of both supervised and unsupervised learning. This is where only part of the data inputted is labelled, and the algorithm further builds on that. Semi-supervised learning is ubiquitous for many real-world applications as, in reality, not everyone has time to label every single image. Not only is it time-consuming, but it can be costly. having a partially labelled dataset allows the program to deduce patterns that have been recognised and then give an output based on those patterns.[13][14][15]

1.3 The Applications of Machine Learning

There are countless applications of Machine Learning today. In the modern era, we are constantly evolving. The following examples will be discussed in the context of machine learning:

1. Social Media
2. Self Driving Cars
3. Google Translate

1. Social Media.

Social Media is something that has crept in and now has a huge impact on our day to day life. Multiple Social Media platforms today use machine learning algorithms to tailor to our preferences. These algorithms consider how much time we spend on specific posts, what we like, what we save and even what we search and then give us further suggestions based on what the algorithms assume our preferences are.[16][17]

2. Self Driving Cars.

Self Driving Cars are a fascinating dream that is becoming a reality. The company Tesla, for example, is using the unsupervised learning method to train their car models to detect both people and objects. [18] [19]

3. Google Translate

Google Translate is one of the most used applications of machine learning in the modern-day today. Google Translate uses natural language processing to work on both languages and dictionaries. This is to ensure it maximises accuracy when translating sentences or words.[20][21]

2. Neural Networks Background

Artificial neural networks are highly fascinating machine learning models. These networks are inspired by the brain. The central nervous system as a whole, where smaller processing units which are called neurons, are linked together to make a then dynamic network that has the power to both learn and adapt.[22]

The intention of Artificial Neural Networks is based on the complex belief that by making specific connections, the human brain can indeed be imitated using silicon and wires as living neurons and dendrites.[23]

Artificial Neural Networks are made up of many nodes, which take after biological neurons. These neurons are connected by what we call links that can interact with each other. The nodes can take input data and perform operations on that data. The result of these operations after a calculation is sent on to other neurons. The output at a node is known as its activation or its node value.[24]

Each link is associated with weight. Artificial Neural Networks learn by altering weight and bias values. Here is an example of a simple Artificial Neural Network. [25]

2.1 Building Blocks

Neural Networks are made from smaller "Building Blocks". This is similar to logic gates in digital circuits. They are made up of three main "Building Blocks" namely:

1. Network Topology
2. The Adjustment of Weights and Learning
3. Activation Functions

This section describes and discusses these building blocks to gain more insight into them.

Neural Network Topology refers to how the neurons are connected. There are two types of Neural Network Topologies. These are called Feedforward and Feedback. [26]

Feedforward Neural Networks are ANNs in which the connections between their units do not form a cycle. They are called "Feedforward" as the information only travels forward in the network, beginning through the input nodes, then through the hidden nodes and finally through the output nodes.

How does it work?

A Feed Forward Neural Network is usually seen as a single layer perceptron. In this model, a amount of inputs enter the layer and are multiplied by the weights. A sum of the weighted input values is computed when each value is added together. The single-layer perceptron is the most simple model of feedforward neural networks, and it is used in classification tasks.

Single-layer perceptrons also tend to incorporate aspects of machine learning. Using the delta rule, the neural network can compare the outputs of its nodes with the intended outputs, which further allows the network to adjust its weights through the training process to produce more accurate output values. This training and learning process produces a form of a gradient descent that is discussed more in the next section.[27][28]

The Adjustment of Weights and Learning

Weights control the strength of the connection that is in between two neurons. In simpler terms, a weight decides how much influence the input will eventually have on the output.

Biases, which are constant and are an additional input into the next layer. These units are not influenced by the previous layer. However, they do have connections with all neurons in their layer, each connection with their own weights. The bias unit guarantees that even when all the inputs are zeros, there will still be some sort of activation in the neuron.[29][30]

2.2 Neural Network - Neurons

Designing a neural network can be quite difficult and confusing. This section is about Neural Network design and looking into the different layers associated with the structure/architecture of a neural network.

Neurons are fundamental units of the brain and nervous system. They are the cells responsible for receiving sensory input from the outside world, sending motor commands to our muscles, and transforming and relaying the electrical signals at every step in between. Neurons are messengers that send information. They use electrical impulses and chemical signals to transmit information between different areas of the brain, and between the brain and the rest of the nervous system. [31]

In deep learning, neurons are nodes that data and computations flow from. They receive one or more input signals from either a dataset or neurons positioned at a previous layer of the neural net. They then are multiplied by values that are known as "weights", added together, and finally,send some output signals to neurons deeper in the neural net.

The Perceptron

A perceptron is an artificial neural network that does a specific calculation on input data that was given to detect specific features. It is an algorithm designed for supervised learning with binary classifiers (deciding between two different classes). This design ensures that neurons are able to learn and process elements that are in the training dataset one by one. The perceptron was introduced by Frank Rosenblatt in 1957, he is seen as "The Godfather" of the perceptron. There are two types of perceptrons: single and multilayer. Single layered

perceptrons can only acquire information linearly. They also only have an input, output and one hidden layer. While multilayered perceptrons have a higher accuracy as a result of having multiple hidden layers. During the training process all the weights and biases are configured based on the training data. [32], [33]

Multiple signals from the previous node are accepted by the perceptron and combined with their biases, thus if the signal is above the threshold it will output 1 otherwise it will output zero. When it comes to calculating the loss function, the initial predicted output is compared to the known output, which generates the loss function, which we will cover more in gradient descent.[34]

MultiLayered Perceptrons

The Multilayer Perceptron is viewed as a good base for deep learning as it is seen as a good place to start when researching the topic of deep learning. It is a deep, artificial neural network that is a class of feedforward networks. They consist of three parts. An input layer which receives the signal, an arbitrary number of hidden layers in which calculations are made and finally, an output layer that decides or predicts the output. Each node is a node that utilises a non linear activation function. Oftentimes, these perceptrons are used in supervised learning problems where they train on a specific amount of labeled input data and further learn to find the correlation between the inputs and the outputs. To minimise error training must be done. This includes adjusting weights and biases of the model many times. [35], [36]

If you think about it, tennis and feedforward networks like MLPs are quite similar as they both involve various back and forth motions. In the first serve, the signal goes from the input layer to the hidden layer. In the end the output layer decides and is put up to measure to the ground truth labels.

In the opposing serve, different weights, partial derivatives of the error function and biases are back propagated through the Multi Layered Perceptron. The differentiation involved gives us what is called a gradient. This is an area of error where parameters can be changed as they progress the MLP to the minimum amount of error.[36], [37]

Back Propagation

Back propagation is the practise of fine tuning the weights in a neural network that is based on the error that was previously computed. This “tuning” gives way for error rates to decrease to make the model more reliable. This is basically the messenger telling the neural network whether or not it made a mistake with the prediction. Back propagation is just a standard method of training artificial neural networks. [38]

The definition of propagate is to transmit something (for example, light or information) in a specific direction or through a specific way. The definition to backpropagate is to then send something in response, like information back except in this case, with fixing a specific error. When we talk about back propagation in deep learning, then we are discussing the transmission of information, and that information correlates to the error that is produced by the neural network when it makes a guess about data. [38], [39]

A Simple Explanation of Back Propagation

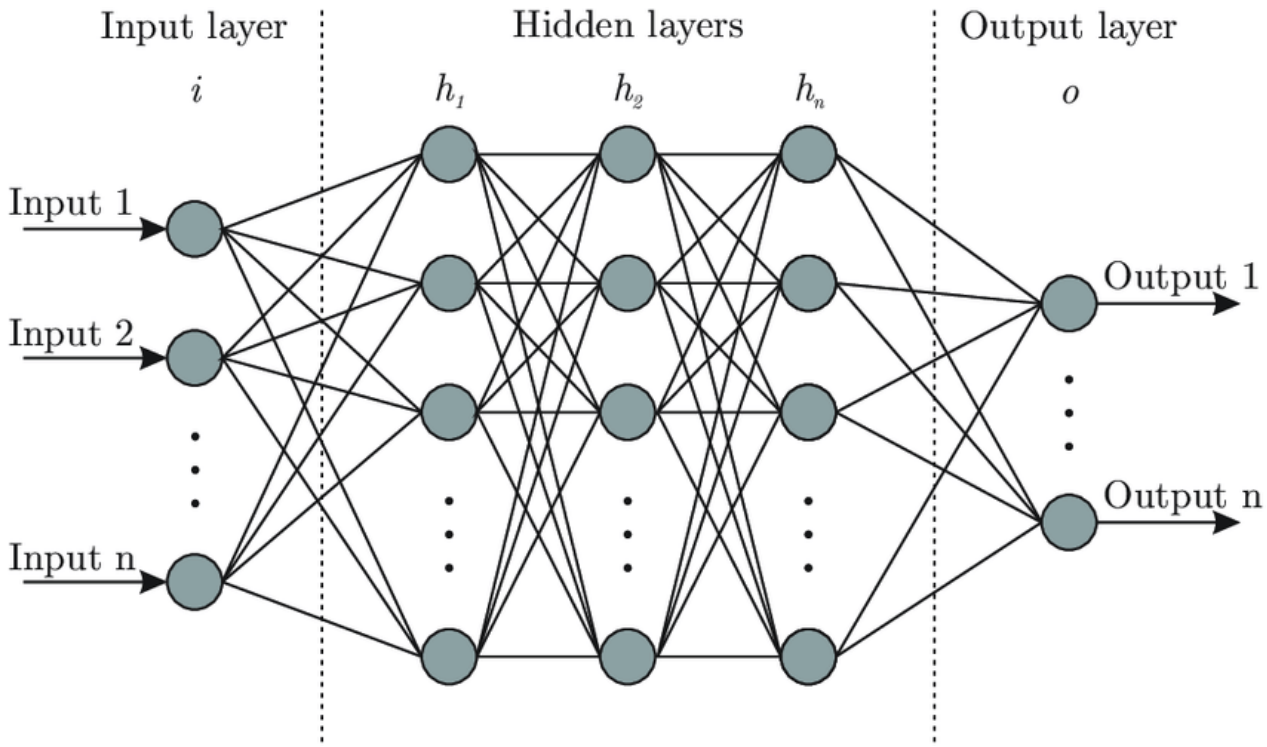


Figure 2.2.1 - Structure of a feedforward neural network [40]

See fig 2.2.1 above.

Step 1: The inputs aka “x” arrives from the dataset

Step 2: All 36 weights are randomly picked.

Step 3: The value of every neuron is calculated based on the previous weights*previous values + the bias of this layer

Step 5: The error is calculated by the following. Error = The actual output - The desired output

Step 6: all the weights are adjusted in order to decrease the error.

This is repeated max steps number of times in most of our models this was 500,000 times.[35]

Feedforward Neural Network

A feedforward neural network is the same as the multilayered persepticon except for it is not limited to binary classification as each neuron can have a value between 0 and 1 which can conveniently be displayed as percentage.

A fully connected layer just means that each neuron in the layer is connected to every neuron in the previous layer.

Gradient Descent

"A gradient measures how much the output of a function changes if you change the inputs a little bit." —Lex Fridman (MIT). [41]

Gradient descent is a mathematical method of calculating all the weights and biases for the network based on the loss function. The gradient descent measures, in simple terms, the change in the weights with attention also to the change in error. If you think of the gradient as the slope of a function. The higher the gradient is, then steeper the slope is and finally, the faster your model can learn. However, if the slope is zero, then the model simply stops learning. In maths, a gradient is known as a partial derivative with respect to its inputs. A Loss Functions reveals to us satisfying our model is at making predictions when given a set of parameters. The cost function has its own personal curve as well as its own specific gradients. The slope that is then given for this curve tells us how to update our parameters to make the model more accurate.[41], [42][43]

The Gradient Descent Procedure

To begin, there are initial values for the coefficient or coefficients for the function to be used. This could be any small random value.

coefficient = 0,The cost of the coefficients is calculated by inserting them into the function and calculating the cost.

cost = f(coefficient)
cost = evaluate(f(coefficient))

The derivative of the cost is premeditated. The derivative is the slope of the function at the given point. We need to calculate the slope so that we know the direction to move the coefficient values in order to get a lower cost.

delta = derivative(cost)

Now that we know from the derivative which direction is downhill, we can now update the coefficient values. A learning rate parameter which is alpha has to be specified because that controls how much the coefficients can change on each update.

coefficient = coefficient – (alpha * delta)

This process is repeated until the cost of the coefficients is 0 or even close enough to zero.

2.3 Activation functions

Activation functions are used to determine the output of a neural network, they are in fact a critical part of the design. Activation functions make their decisions on whether or not a neuron gets activated. This is based on calculating the weight and adding bias to it. They can also be divided into two types which are linear and non linear activation functions. Most activation functions used are in fact non linear.

Linear Activation Functions.

For linear activation functions this is defined as a straight line function where the activation is actually proportional to the input. By doing this it gives us a selection of activations so it is not in fact binary. This is a problem as the derivative remains constant which means that learning does not take place in back propagation. This means that the same linear result will always be concluded.[44][45]–[47]

Non Linear Activation Functions.

Non linear activation functions are used for most modern neural networks. It makes it quite simple for the model to adapt with data and differentiate between the output. Non linear activation functions make the whole network more accurate as the derivative is not fixed which enables better accuracy of an output.[45][48]

Relu

Relu is non linear in nature but can give an output of A if A is positive and 0 if not. It is a popularly used activation function that involves simple maths problems that are implemented in the hidden layer. It is seen to be quite efficient computationally.[45], [46]

2.4 Convolutional Neural Networks (CNN)

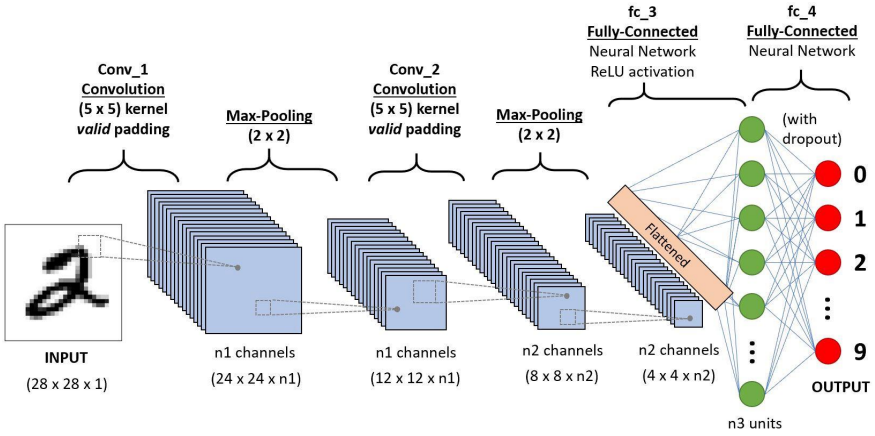


Fig 2.4.1 - Example CNN [49]

An example CNN for recognising handwritten digits is shown in Fig 2.4.1 above. As can be seen from the diagram, there are three different types of CNN layers: convolution, pooling, and fully connected layers.[49]

Convolutional layer

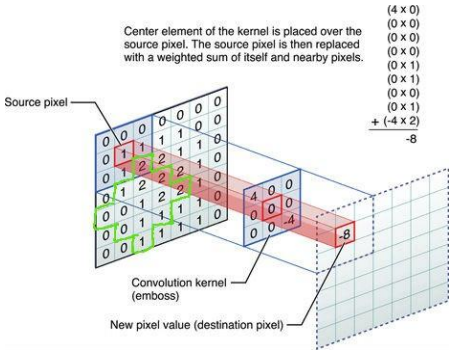


Fig 2.4.2 - Calculating a single Convolutional layer output. [50]

The purpose of a convolutional layer is to extract the features from the image. It accomplishes this by comparing the image to a filter(kernel). The input image is represented as a 3D matrix(red, green, blue). The kernel is another 3D matrix. Each value in the kernel corresponds to a pattern it is trying to identify. At the moment, let us discuss only the red channel. In Fig 2.4.2, above the 7x7 matrix on the left is the input image. The output table on the right is calculated by first considering the 3x3 grid in the top left of the input. The dot product of this and the filter is calculated. This value becomes the value in the corresponding output table. This box then moves over one, and the new dot product is found. This is repeated until the entire input image is analysed. In this case, the output table will be 5x5. This is because the edges are not being considered. This can be fixed by adding padding to the image, adding zeros to all sides of the matrix. This output matrix is then combined with the matrix of Green and blue. This means the output matrix is the same

dimensions as the input. Then an activation function is applied to the values to increase their nonlinearity. [51] [52][53]

There can be multiple filters applied in the same convolution layer. This means it outputs a multidimensional matrix.

In summary
A Convolutional layer takes in an RGB matrix, applies various filters to it, then outputs a matrix of the same x-y matrix but with the z dimension equal to the number of filters applied. The output matrix contains information about the features or patterns in the image. For example, it could show where the outlines of an object are.

Pooling Layer

The purpose of the pooling layer is to downsample the imputed matrix. This means that it reduces the dimension of the inputted matrix while preserving their relevant features.
To accomplish this, it starts in the top left of the input, finds the min, average or max of the values in the 2x2 section, then it saves this in the output matrix. Then it moves stride length(2) over and repeats this until it has gone through the entire matrix. With a stride length of two, it means that it halves the size of the output matrix. The mathematical function used can adjust what features it preserves. The most common function is maximum, as it will show if a feature is present or not. Fig 2.4.4 below demonstrates this.

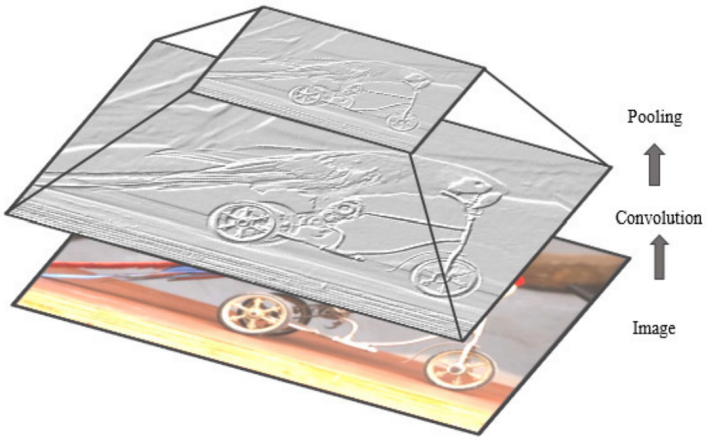


Fig 2.4.3 - Visual representation of a CNN [54]

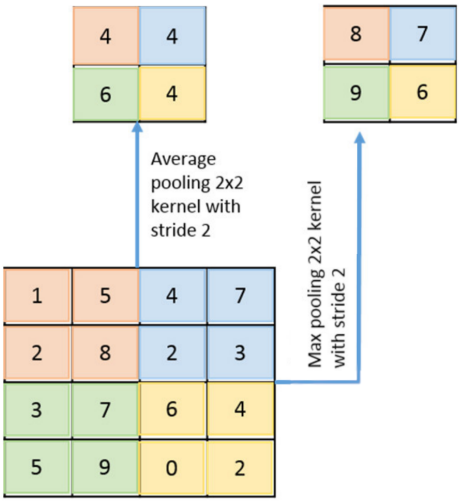


Fig 2.4.4 - Max/Mean Pooling layer [54]

Fig 2.4.3 above shows an overview of the effect of a convolution layer and poling layer on an input image.[55]

Dense Layer

A dense layer(s) is the same as the fully connected layers of a feedforward network, as discussed in section 2.1 and 2.2. The purpose of this is to connect the features to the different output classifications.

Overview

These layers are fed one into the next for example, in Fig 2.4.1 above shows a convolution -> max-pooling -> convolution -> max-pooling. This means that the first convolution layer outputs the lines in the image, then the pooling layer reduces the dimensions of the image to reduce processing time. Then the next convolution layer will combine the lines to form shapes, e.g. a circle is composed of top left line straight horizontal line, top right line, straight down the line... Then another pooling layer. Then the matrix is flattened to a column vector so that it can be fed into the Dense layer. This dense layer(s) connect these features to the different classes. In this case, there are ten output classes meaning the dense layers connect the features to these ten output neurons. For example, the zero node should be connected to the circle feature connected to the eight different lines that make up a circle. [56]

Very complicated CNNs can be built using these basic layers and adjusting all the parameters mentioned above.

2.5 Data Pre-Processing

Data fed into a CNN has to be a specific size as in an image classification, CNN could be trained to accept a 500x500 image. This means that if the input image is 800x600 pixels, it must be disproportionately scaled down to 500x500. The pixel values in the data often have to be reduced from 0-255 to 0-1 per pixel. This is to speed up the training as large values would make the gradient descent take longer.

3. Object Detection

3.1 - Base Models

See figure 3.1.1 in appendix for a comparison of the object detection models.

Our Base Model - ResNet 50

The base model chosen is SSD ResNet50 V1 FPN 640x640 (RetinaNet50) as it has a good balance of speed, accuracy and output resolution. This is a fully convolutional neural network which means that it can only perform convolution operations. It is a Convolutional Neural Network, CNN, without fully connected layers. ResNets operate on the principle of building deeper networks compared to other plain networks while simultaneously determining an optimal number of layers to avoid the vanishing gradient problem. [57]

The Vanishing gradient problem occurs on deep neural networks as a CNN is trained with backpropagation. Each time the activation signal moves backwards, it reduces in magnitude, which means that it is practically zero when it reaches the start of the network. This means that the initial layer is not learning effectively and leads to a network which does not perform well.[58]

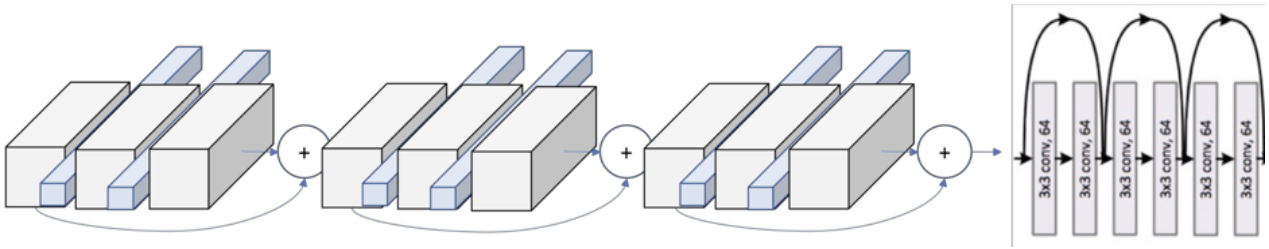


Fig 3.1.1 - 6 layers of Resnet [58], [59]

ResNet solves this by skipping every two layers, as shown in Fig 3.1.1 above. Each white square represents convolution and batch normalisation, and the blue square represents the ReLU activation function. Each of the arrows skipping means that the result from the previous layer is added to the next layer.

This allows it to skip layers when going through backpropagation, meaning that the Vanishing gradient problem is significantly reduced.

See Fig 3.3.1 to see the full ResNet model.

3.2 - Object Detection algorithm

This method is used to adapt ResNet 50 from an image classification CNN into an object detection network. This means that instead of just labelling an image "cat", a box is drawn around the object, and there can be multiple instances of the same or different objects in an image, See Fig 3.2.1.

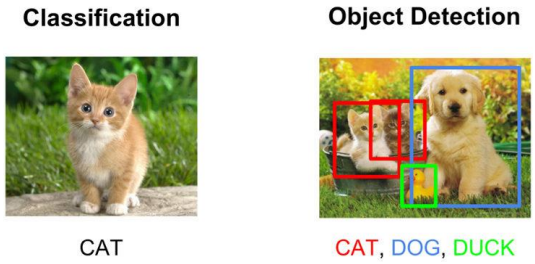


Fig 3.2.1 - object classification vs object detection [60]

In order to draw this, our model must output the following information for each detected object:

Accuracy of detection.
Height of the box.
Width of the box.
X coordinate of the centre of the box.
Y coordinate of the centre of the box.
Using this data, a box can be overlayed on the image. [61]

The most straightforward way of achieving this is with a brute force method. This would mean that the image is cropped to a box and slid across the image. Each time it moves, the region is run through an image classification CNN. As well as moving the windows across the image, the scale and aspect ratio needs to be changed. This is because the number of objects, the location and their scales is unknown. This is not very practical as it would require way too much processing power. [62]

The following algorithm to consider is R-CNN. This improves the brute force method by testing 2,000 regions instead of all possible regions. These regions are found by a traditional image processing technique that will output the 2000 most likely boxes for an object to be present. This then runs these regions through CNN to predict what, if anything, is in the regions. [7]

The model that was chosen is Single Shot MultiBox Detector (SSD).
This model was chosen as it is much faster than R-CNN, which allows it to be deployed on embedded systems, which still retain a fair amount of accuracy. [63]

The first part of the network is a standard image classification CNN with the last layer removed, so instead of outputting object classes, it outputs the results from the feature maps. This uses ResNet as the base model, which produces a 2048x1x1 feature map. The base model in Fig 3.2.2 below is shown in white. The base model has had training turned off as its only purpose is to identify features in the input image. The important part is the SSD Layers which is another CNN that outputs the object detection results. This is trained to figure out what connections are needed in order to obtain the desired boxes.

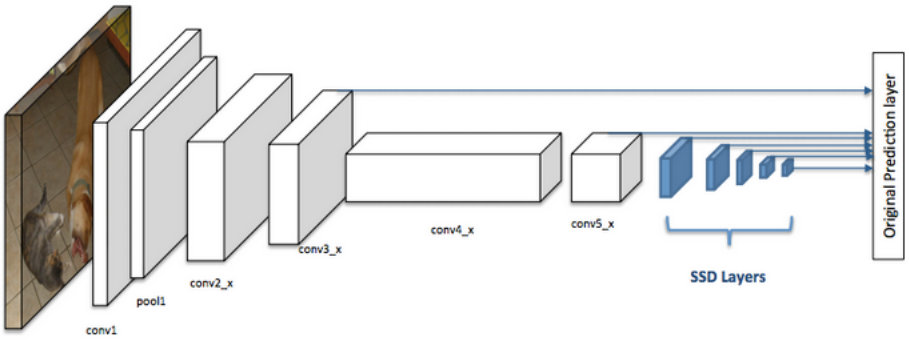


Fig 3.2.2 - Example SSD Object Detection Network [61]

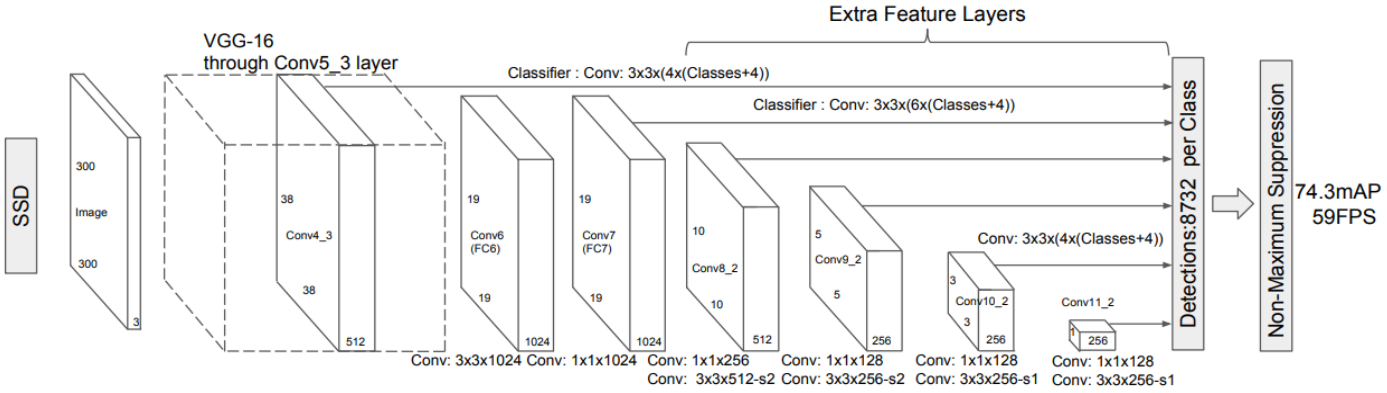


Fig 3.2.3 - Accurate representation of SSD [63]

The VGG16 CCN seen in Fig 3.2.3 above, in our case, is the resnet50 base model. These SSD Layers will produce 8732 predictions per class then uses non-maximum suppression to pick the best 200.[63]

3.3 - Putting it All Together

To summarise what happens when you run the network. First, an image is inputted then is scaled to 640 x 640. This is then fed into the ResNet 50 base model, which identifies the features in the image. This is then fed through the SSD CNN that will output 52,392 (8,732*6) predictions (8,732 predictions/class). This then selects the best 200 boxes according to non-Maximum suppression. Then it will discard any boxes with less than 50% certainty. Finally, it draws the remaining boxes based on their centre coordinates and their width and height.

The diagram of the exact full network which was used is shown in Fig 3.3.1.

4. Implementation

4.1 Coco Dataset

The Coco Dataset is a large scale object detection, Segmentation and captioning dataset. The term COCO stands for Common Objects in Context. This is one of the most common datasets at this current time. The dataset is used to detect objects, segment masks and separate things on continuous backgrounds. See Fig 4.1.1 below.

The COCO dataset is used to test and train different models with different model parameters. It makes it easy to evaluate the performance of the network and compare it with others.

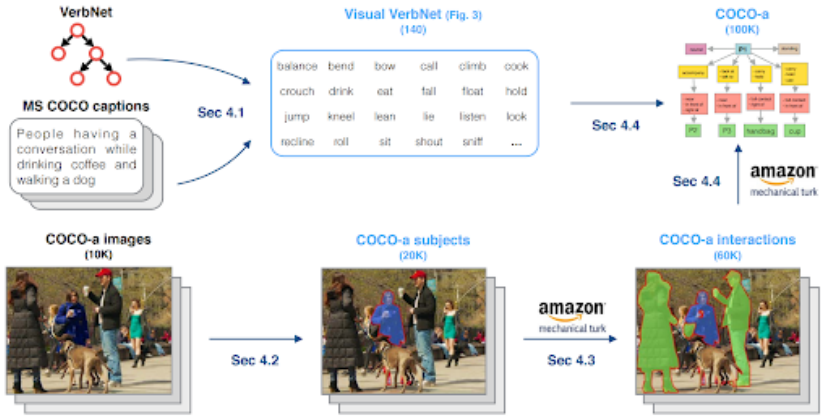


Fig 4.1.1 - Visual Representation of COCO dataset [64]

COCO Dataset Contains 330k images, but we decided to use 70k images instead.
Six different classes were used as a subset for the COCO Dataset as using the entire dataset would take too much time to train. This would be highly problematic for the code as it would use up more time each time it was trained to get accurate results. [65]

The classes we used were:

- Dog
- Person
- Bicycle
- Bus
- Car
- Truck

4.2 Data Preprocessing

The most challenging part of developing a Neural Network is the Preprocessing of the data. The goal for preprocessing is to have 70,000 numbered images and a record that contains the description for that image. That record should not contain anything other than the chosen classes: Car, Bus, Bicycle, Truck, Dog and Person, as these are the classes a CCTV camera is likely to see.

A

The first problem to solve is how to separate the 65 classes from the six chosen.
The coco dataset format is originally stored in PASCAL Visual Object Classes (VOC) data format, which means that All of the labels are stored in a single JSON file.
This had to be converted using anno_coco2voc.py [66], which changed the VOC data format so it could be stored in a folder for each class with a .xml file for each image. That is about 75,000 XML files.

B

The problem now is that multiple items can be in an image and, therefore, in the XML file, see Fig 4.2.1 in appendix. As can be seen, that image has a cat and a bottle which needs removing. Since there are so many XML files, it does not make sense to modify them by hand, so we wrote a python program to do it see Fig 4.2.2 in the appendix.
The way the program accomplishes this task is it moves all the XML files into a tmp directory, then it reads through each XML object tag for any which has a name, not in the above six classes. If it finds this, then it deletes the object tag and saves the file.

C

The next problem is that the XML files are separate from the images. At this point in one directory are all of the images from all classes and in another it has all the XML files which have been processed. The program solves this by moving the XML file into a new directory and copies an image file with the same name to the same directory.

D

The last preprocessing step is to run the generate_tfrecord.py, see for the code [67].
This converts the many XML files to a single train.record and test.record

E

Tensorflow has also been configured to crop the images randomly within the following limits.
The aspect ratio of a cropped image to 0.75-3:1.
Area ratio between cropped and before crop 0.75-1:1.

In the future we could experiment with mirrors, rotating, combining multiple images, adjust contrast and add noise. These methods could improve the accuracy of the CNN as it will greatly increase the number of training images.

4.3 Training

Training requires an abundance of processing power and time which is why we chose to run it locally.

All six convolutional neural networks were trained on a local installation of TensorFlow with GPU acceleration. This allowed us to use Donovan's computer which has a GTX 1070 graphics card with 8GB of GDDR6 VRAM. [68]

This is very useful as CNN training is much faster on a GPU when compared to a CPU. [69] This is because GPU is optimised for doing lots of parallel processing and has onboard high-speed memory. For example, the GTX 1070 has 1920 Cuda cores meaning it can potentially do 1,920 operations simultaneously, whereas a high-end CPU might have eight cores 16 threads, meaning it can do 16 things simultaneously. As you can imagine this makes a big difference to the training times.

Even with a fairly high-end GPU, the 8GB VRAM is a limiting factor meaning a batch size of over six could not be trained. See Fig 4.3.1 in appendix for all the training settings used to train the AI with Batch Size 6 and 500,000 steps. The parameters we varied were Batch Size, Images trained on and Number of Steps.

We trained the following models:-

Parameters	**Time training took (HH:MM)**	
4K-1Batch-25KSteps	01:36	
4K-6Batch-500KSteps	213:21 (8.9 days)	See note about inaccurate time
70K-4Batch-25KSteps	04:09	
70K-5Batch-60KSteps	11:45	
70K-6Batch-150KSteps	34:22 (1.4 days)	
70K-6Batch-400KSteps	143:10 (5.9 days)	
70K-6Batch-500KSteps	175:24 (7 days)	

Note: training times vary because the computer was used while it was training.

Once everything is configured in the pipeline.config, all you have to do is call model_main_tf2 and wait a long time [70].
It shows helpful progress reports in the console as it trains. See Fig 4.3.2 in appendix.
While it is training, Tensorboard can be used to graph lots of data about the network.

4.4 Validation

We can validate the performance of the AI using the following methods

1. Mathematical
2. Comparing images
3. Comparing videos
4. Real-time analysis

1. Mathematical

We can use the model_main_tf2 TensorFlow script which uses the cocoa API to produce the following metrics.

IOU (Intersection Over Union) is a value used to calculate if a particular prediction is a false positive or true positive. It is calculated by calculating the ratio of the area of overlap between the prediction and the ground truth, See Fig 4.4.1 in appendix. This means if you had a prediction which only covered a small part of the ground truth, it would not be marked as a success. [71] [72]

mAP(Mean Average Precision) score is how many false positives there are compared to true positives. Therefore a higher mAP score is beneficial, See Fig 4.4.2 in appendix

mAP@.50@IOU or mAP@.75@IOU is the mean average precision at a particular IOU value. mAP (small, medium, large) these are for different sizes of ground truth objects. For example, you could tune your network only to detect large objects.

AR(Average Recall) score is how many false negatives vs true positives it has. Therefore a higher AR score is beneficial.

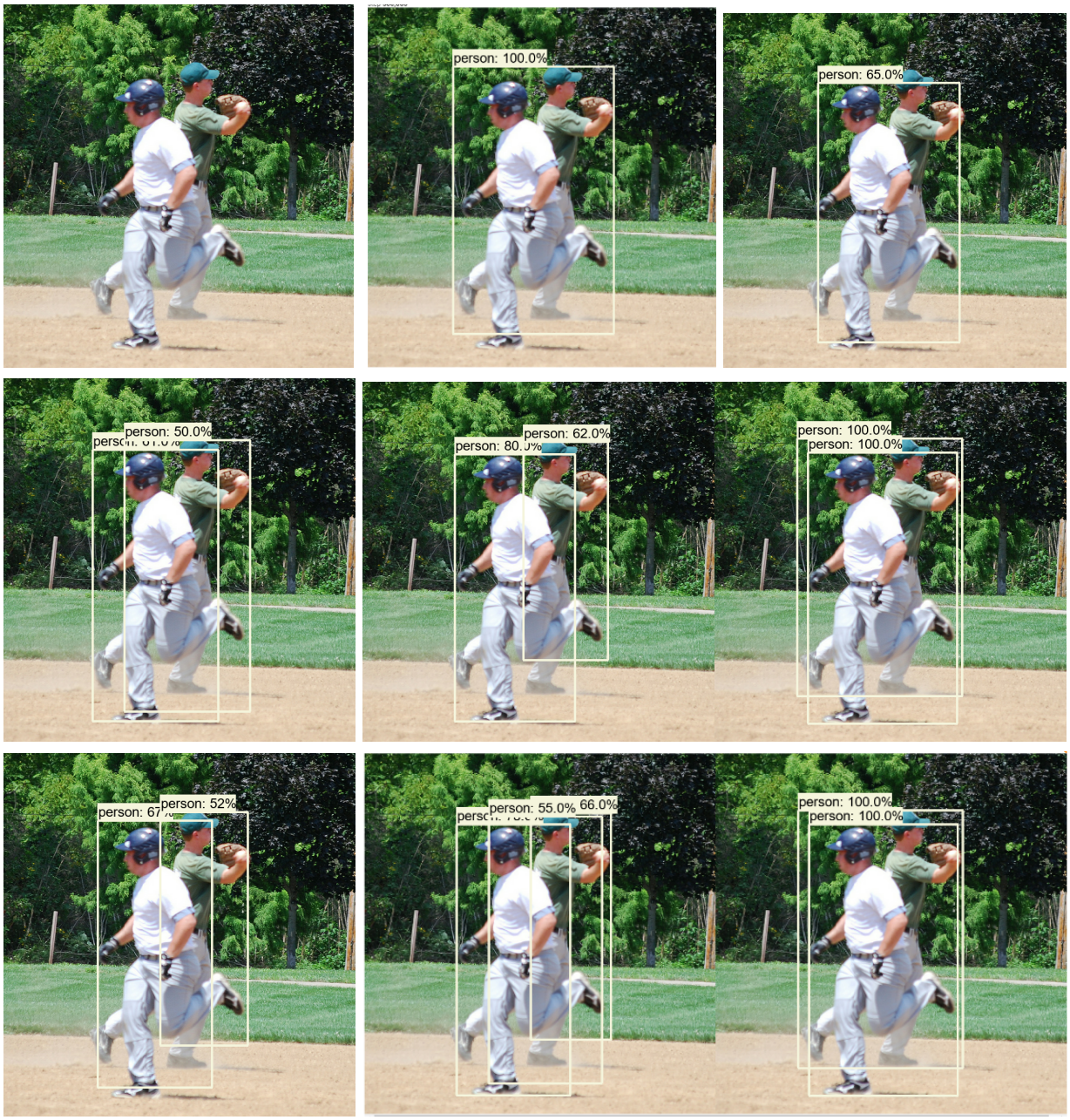
AR@1 means that it is only considering images with at most 1 object for Average Recall.
AR@100 only considered images with at most 100 objects for Average Recall.
AR@100 includes all the images in our dataset as all images have less than 100 objects.

Classification Loss and localisation Loss is a measure of how inaccurate the AI is at identifying the class and position of the boxes, respectively. Therefore a lower Loss score is beneficial. The way it gathers these metrics is that it uses the 3,000 images set aside for evaluation. It then feeds these images into the network and records the predictions the network makes. It then can use the IOU value to decide on whether each predicted box is covering a ground truth box. It then records how many boxes predicted the correct class, wrong class and missed completely, see Fig 4.4.2 in appendix for the diagram. Then it feeds this information into formulas for the above metrics and averages this overall 3,000 images.

2. Comparing images

Comparing images method also uses the 3,000 evaluation images, but instead of evaluating them mathematically, images can be picked at random and examined by hand see Fig 4.4.3 below.

This method is good for demonstrating the capabilities of the CNN(convolutional neural network) and useful to visualise its performance.



4K-1Batch-25KSteps	4K-6Batch-500KSteps	70K-4Batch-25KSteps
70K-5Batch-60KSteps	70K-6Batch-150KSteps	Ground Truth
70K-6Batch-400KSteps	70K-6Batch-500KSteps	Ground Truth

Fig 4.4.3 - example of comparing images method

3. Comparing Videos

Comparing Videos is another visual way of accessing the performance of CNN. The way this one works is we wrote a custom Python script. See Fig 4.4.4 in appendix.
This loads the video a frame and then runs it through the detection model, overlays the detection boxes on top of the image and then writes this to a new .avi video file.
This then can be transcoded to a different format using FFmpeg [73]

4. Real-time analysis

The real-Time analysis involves feeding the computer's webcam into the CNN and displaying the output results in real-time. This is what we demonstrated in the interim video and presentation.

We also wrote a script for real-time analysis of the CNN, see Fig 4.4.5 in appendix [74]
The disadvantage of this is that it runs at a low framerate as the computer has to do all the processing in real-time.

We have not validated the network with Adversarial Attacks, which could trick the network into not detecting someone. [75]

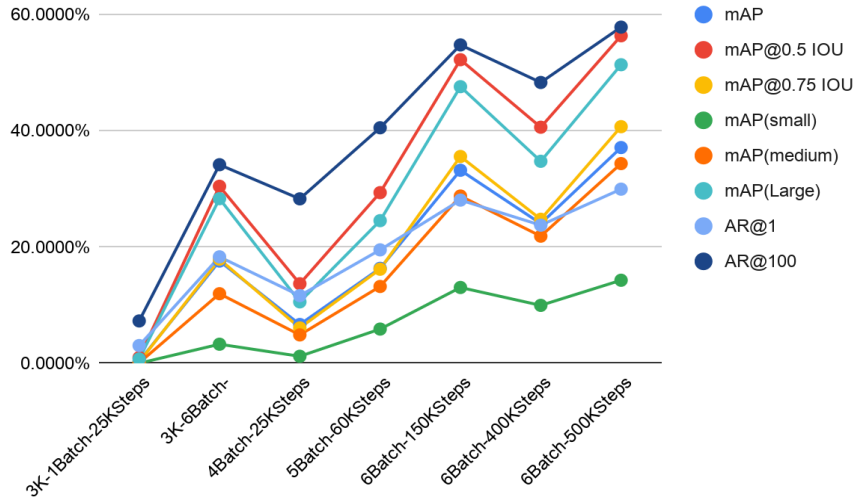
4.5 Problems encountered

- Tensorflow with GPU acceleration was very difficult to install and required fiddling with Nvidia drivers a lot, including reinstalling my graphics drivers three times.
- There was much fiddling around with the versions of the Python packages so that all packages were compatible. In the end, two Anaconda virtual environments were required, each with different versions of NumPy so that validation and training would both succeed.
- The Model, which was trained on 400,000 steps, was trained up to 500,000 steps until windows decided to update almost six days into training.
- Not enough GPU VRAM to increase the batch size.
- Not enough GPU power to increase the number of steps without increasing the training time too much.

4.6 Results

Mathematical

Validation Data										
Models	mAP	mAP@ 0.5 IOU	mAP@ 0.75 IOU	mAP (small)	mAP (medium)	mAP (Large)	AR@1	AR@100	localization loss	classification loss
3K Images Batch Size 1 25K Steps	0.3666%	0.9915%	0.2219%	0.0204%	0.0811%	0.7456%	3.0613%	7.2957%	56.6553%	69.6365%
3K Images Batch Size 6 500K Steps	**17.5975%**	30.4661%	17.9251%	3.2931%	11.9812%	**28.3300%**	18.3232%	**34.1724%**	33.2116%	64.9525%
70K Images Batch Size 4 25K Steps	6.6865%	13.6947%	6.0943%	1.2158%	4.8910%	10.5776%	11.6488%	28.2994%	37.0971%	48.1443%
70K Images Batch Size 5 60K Steps	16.3550%	29.3702%	16.1959%	5.8962%	13.2250%	24.5504%	19.5251%	40.5296%	28.2252%	37.8204%
70K Images Batch Size 6 150K Steps	**33.2172%**	**52.2373%**	35.5855%	13.0523%	28.7876%	**47.5977%**	28.0879%	**54.7798%**	18.9524%	27.4378%
70K Images Batch Size 6 400K Steps	24.0427%	40.6225%	24.7876%	9.9756%	21.8802%	34.7629%	23.7685%	48.3192%	23.4299%	33.0798%
70K Images Batch Size 6 500K Steps	**37.1248%**	**56.3542%**	40.7161%	14.2981%	34.3656%	**51.3718%**	29.9675%	**57.8401%**	17.1319%	25.7544%
	green = low false positives, higher numbers better						green = low false negatives		The measure of how inaccurate the AI is	
	red = predictions are wrong						red = missed many objects		The lower number the better	



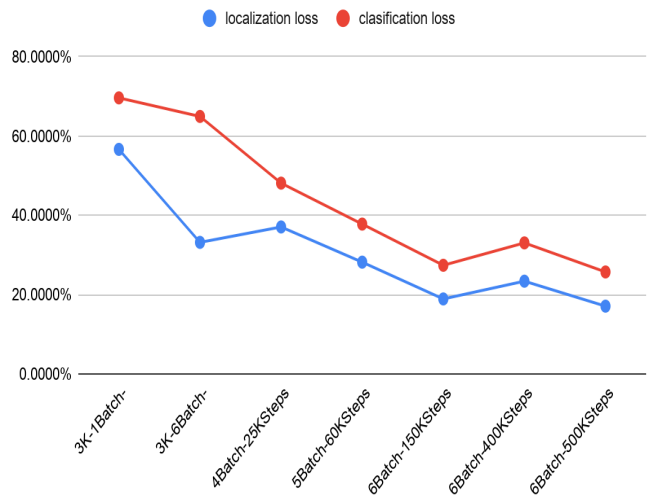


Fig 4.6.1 - Validation Data Mathematical

Training Data										
Models	mAP	mAP@ 0.5 IOU	mAP@ 0.75 IOU	mAP (small)	mAP (medium)	mAP (Large)	AR@1	AR@100	localization loss	classification loss
3K Images Batch Size 1 25K Steps	0.3602%	0.9963%	0.1697%	0.0092%	0.0799%	0.7794%	3.7758%	6.5835%	56.0097%	69.4560%
3K Images Batch Size 6 500K Steps	83.6378%	96.2759%	90.7599%	64.0740%	87.2254%	91.6516%	47.6362%	87.1156%	2.0938%	6.4966%
70K Images Batch Size 4 25K Steps	6.8192%	13.7485%	5.8971%	1.3748%	4.1721%	11.1881%	11.9462%	26.7177%	37.0334%	48.4817%
70K Images Batch Size 5 60K Steps	15.7539%	28.0559%	15.8546%	3.3516%	11.8453%	24.2199%	18.5797%	39.5762%	27.8188%	38.2817%
70K Images Batch Size 6 150K Steps	34.0906%	52.5451%	36.7451%	9.7080%	27.2981%	49.3257%	28.9555%	54.5936%	17.6640%	26.2613%
70K Images Batch Size 6 400K Steps	24.0700%	40.0923%	25.3540%	7.4604%	19.9290%	35.3482%	23.3659%	47.1934%	22.6722%	32.7392%
70K Images Batch Size 6 500K Steps	39.7349%	59.8009%	43.4217%	14.5958%	34.6318%	54.7510%	31.7656%	59.2214%	14.9288%	23.4055%
	green = low false positives						green = low false negatives		The measure of how inaccurate the AI is	
	red = predictions are wrong						red = missed many objects		The lower number the better	

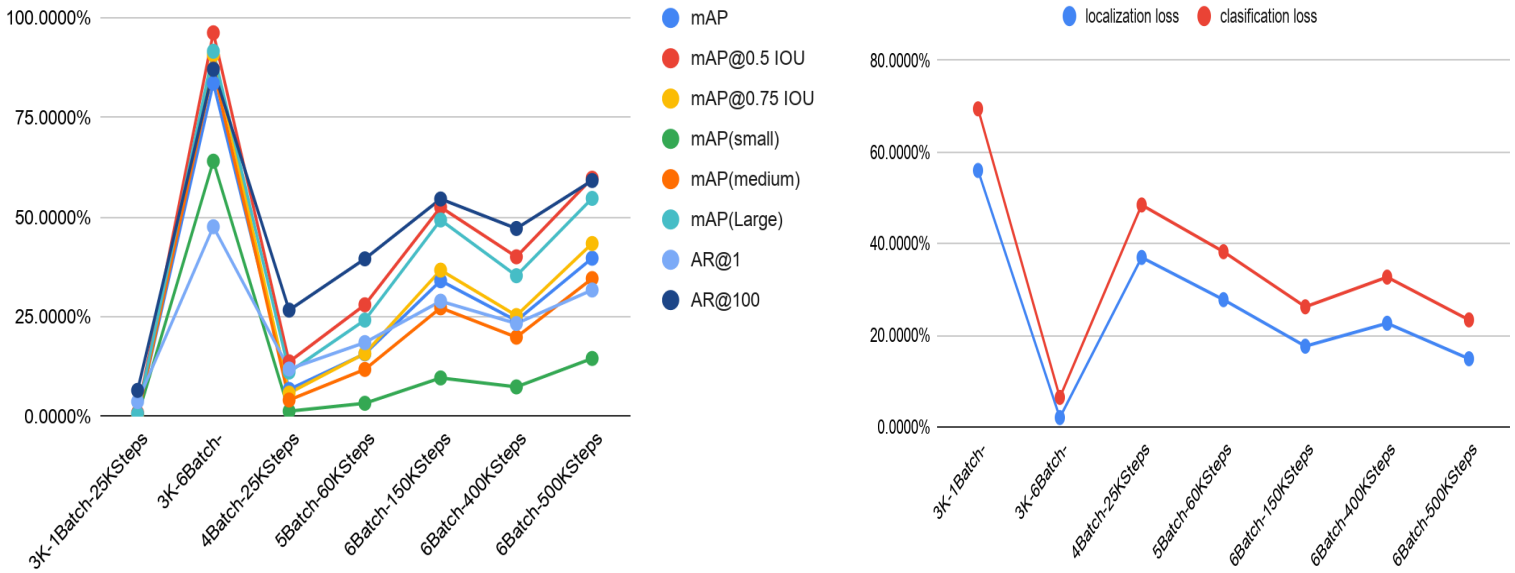


Fig 4.6.2 - Training Data Mathematical

For the statistics based on the validation data and graph view Fig 4.6.1 above.
For the statistics based on the Testing Data and graph view Fig 4.6.2 above.

As can be seen, the data's general trend is that the accuracy of the CNN improves when increasing either the batch size or the step count. This can be seen across all four graphs.

From the above figures, It was noticed that most of the networks performed about the same or slightly worse on the training dataset compared with the validation datasets. This means that most of the networks seem to be under-fitted as they perform worse on seen data rather than unseen data, see Fig 4.6.3 below.
This also can be that the training data has 70,000 images, whereas the validation data has 3,000 images. So since it averages them, the greater the number of images tested, the more accurate the results.

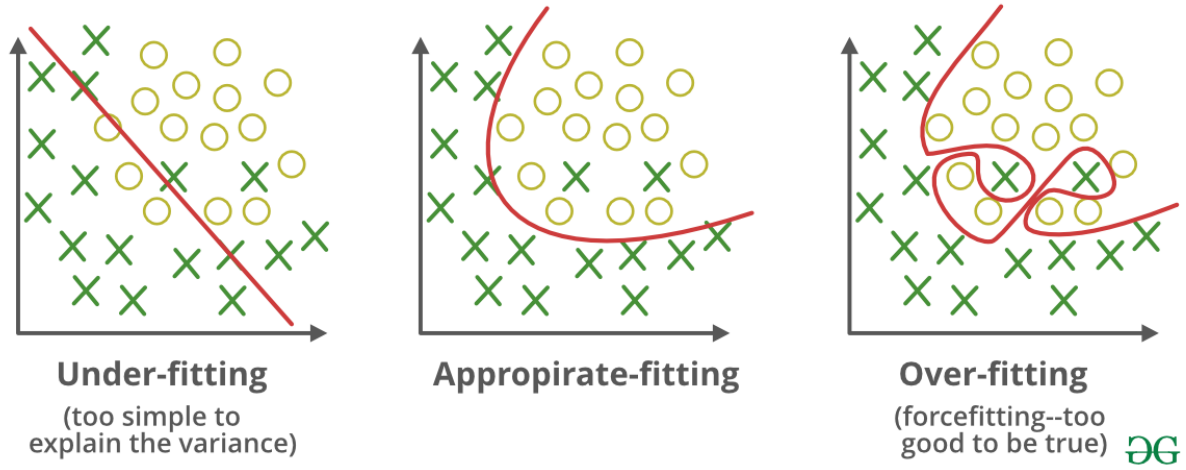


Fig 4.6.3 - Overfitting diagram [76]

Now for the exceptions.

In Fig 4.6.1 above, it may notice that the model (3K Images - 500K Steps) performs better than expected. This is because it was trained with larger batch size and a higher number of steps, meaning that it looked at more images each time it adjusted the weights. It also adjusted the weights and biases 20 times more than the previous model. **This is an exciting discovery as it means that a well-tuned model with a small number of training images can perform better than a poorly tuned model with 23 times more images.**

Again in Fig 4.6.2 above, the model (3K Images - 500K Steps) performs really well, getting pretty close to a perfect score. This is because it was tested on its training data. When this is compared to the other models tested on their training data, it can be noticed that the other models archive approximately the same result in the training data and validation data.

This shows that they are very generalised, maybe even too generalised. This is not the case in this model, which gets almost five times the score in the training data. This shows that this model has started to memorise the training data. This hints at the fact that this model could be overfitted, See Fig 4.6.3 above. This means that it has become too fixed to the data and therefore cannot predict new data as well. In the below diagram, it can be imagined that if more data is added to the graph, it could easily misclassify it.

The following exception is when we reach the model (Batch Size 6 - 400K steps).
This model has decreased accuracy when compared to the 150K steps model. At first, this seemed unusual as it has been trained for longer than the previous model.
Initially, I thought this was because the AI overfit the training data.
I investigated this further by running the same test and calculations on the training data set see Fig 4.6.2 above. This showed that all the networks performed about the same on the training set. This means the error with the 500K steps CNN is not caused by overfitting.

The reason I think that the 400,000 steps AI perform poorly is that it never finished training. It was originally meant to train for 500,000 steps before windows update had other plans. This means that the learning rate curve never started to flatten out at the bottom, so it never made the fine adjustments to the weights and biases necessary to get high precision.

Another possible reason is that TensorFlow kept on warning that the GPU was out of VRAM while running that training run. See Fig 4.6.4 in appendix. It would appear that this is not the problem as the other CNN with a batch size of 6 showed the same warnings while training and did not have any problems. This shows that the 8 GB GDDR 6 VRAM on my GPU is only just sufficient for a batch size of 6. [68]

To prove what caused this, further training was carried out by duplicating the model then resuming the training so that it could repeat the full 500K steps.
This is displayed in all the data as 70K images - batch size 6 - 500K Steps.

This model has confirmed my theory that the learning rate caused it. As can be seen in Figure 4.6.1 above, this model clearly performs better than any others. This is expected as it has the highest number of images, batch size, steps and training time.

Graphs

Running tests and forming graphs helped us obtain the data shown in Fig 4.6.5 - 4.6.11 in appendix. In theory, when using a larger batch size, it takes a lot longer to train, but it becomes a lot more accurate when it settles on a value. We decided to test out this theory and obtained the above results based on the following parameters.

- X Images - How many images used to train the model.
- Batch Size Y - How many images were being used trained in one step.
- Z Steps - How long the training lasts. Each step will update the weights and biases based on looking at batch size Y images.

As can be seen, all the graphs of the learning rate are the same, except for the values on the x-axis are different. This is because TensorFlow adjusts the learning rate as its training. For the first 2000 steps, it sets the learning rate to 0.13333. See Fig 4.3.1 in appendix. It then adjusts the learning rate to a cosine decay function which is scaled to the remaining number of steps before finishing.

While learning this, we found that **TensorFlow does not use the number of epochs, instead, it overrides it when it starts. It will just repeat through the data set until the total steps have been reached.**

Total loss is what the AI is trying to minimise. Total loss is a combination of classification loss, localisation loss and regularisation loss. Classification loss is a measure of how many incorrect guesses it makes. Localisation loss is a measure of how many objects it did not see. Regularisation loss is a measure of how large the weights are, which is used to prompt the AI to use smaller weights. The graph of regularisation loss indicates how likely the AI is to be overfitting.

Normalisation total loss is the total loss with each argument converted to the same scale to avoid floating-point rounding by having large or very small numbers.[77] [78]
Learning rate affects how much the training algorithm tweaks each weight and bias in each step. Therefore, high learning rates mean the AI is making large coarse adjustments to the weights and biases. Therefore, it has to reduce it. Otherwise, it would never make fine adjustments and be inaccurate.

See Fig 4.6.9 - 4.6.12. As can be see, this method confirms the mathematical evaluation, as the network's performance increases as the total steps increases until it reaches 400,000 steps, where it gets slightly worse.
It can be noticed from the above metrics that our model performs very well but not perfectly. This is because it was not trained for as long as it could have been, and some of the images in the COCO Dataset are challenging. For example, in Fig 4.6.13 in appendix, on the left is a picture of people all the way in the background and on the right is a dog. Both these are really hard for even a person to be able to see.

If you would like to reproduce our results, run the commands in Fig 4.6.15 in appendix if TensorFlow and the object detection API is installed and downloaded.

5. Conclusion

Overall there was much to be taken from the entire development of the A.I. The main aim and goals were achieved, which were to implement a Convolutional Neural Network using classification.

Learning how to use Tensorflow greatly added to the success rate of the object detection A.I developed for this project. This was because Tensorflow allowed us to build and train our models with immediate model iterations and high quality executions. We used Single Shot MultiBox Detector (SSD), to determine what and where exactly the objects are inside the image.

Even though we initially encountered some problems with Tensorflow, this was overcome and we were able to continue with the development of our object detection A.I. A better understanding of Python was developed by working on implementing our neural network. Tensorflow was much easier to use than Matlab.

We were able to use what we learned about object detection to create model parameters which we used for our model: SSD ResNet50 V1 FPN 640x640 (RetinaNet50).
This was the model that was chosen because it has a good balance of speed, accuracy and output resolution. This is a fully convolutional neural network which means that it can only perform convolution operations. See section 3.1.

We were also able to test out and train the majority of our model parameters. The base model had a map score of 34, which performed less than our model, which had a map score of 37. This is because we had trained it on six instead of 80 classes. As seen in section 4.1. These six classes are:

- Dog
- Person
- Bicycle
- Bus
- Car
- Truck

Here is a table of results for our best performing models. For more detailed results, see Fig 4.6.1 and 4.6.2 in results.

Results of performance on validation data

Models	mAP	mAP@ 0.5 IOU	mAP@ 0.75 IOU	mAP (small)	mAP (medium)	mAP (Large)	AR@1	AR@100	localization loss	classification loss
70K Images Batch Size 6 150K Steps	**33.2172%**	**52.2373%**	35.5855%	13.0523%	28.7876%	47.5977%	28.0879%	**54.7798%**	18.9524%	27.4378%
70K Images Batch Size 6 500K Steps	**37.1248%**	**56.3542%**	40.7161%	14.2981%	34.3656%	51.3718%	29.9675%	**57.8401%**	17.1319%	25.7544%

As seen from this table of results this is what can be concluded:

70k Images, Batch Size 6, 150k Steps	**70k Images, Batch Size 6, 500k Steps**
Precision - 33.22%	Precision- 37.12%
Due to the lower step count it decreased the accuracy of the A.I	As seen from results due to the high step count the A.I was a lot more accurate.
Took 34 hours to train	Took 175 hours to train

These were the best performing models that were trained. This shows that the A.I can detect objects to a relatively high degree level of accuracy.

In collusion with the results in section 4.6, the network becomes more accurate as the number of steps becomes more accurate until it starts overfitting. At this point, we can prevent overfitting by increasing the bach size. The more batch size increases or steps, the longer it will take to train, and the more accurate the network becomes.
This means that this network is ideal for the chosen use case to detect people or vehicles approaching clients property. This means that the client could reliably have this AI watching a CCTV camera system so that it would alert a security guard if anyone drove down the driveway. It could even specifically only trigger on trucks so that when someone walks down the driveway, it does not go off.
If we wanted to adapt this model to another use case that required more accuracy, you would increase the batch size and the step count. This would mean the training takes a lot longer but is more accurate. To increase the batch size higher than we did, you would need a professional GPU, for example, one or several Quadros.

There is a yearly AI challenge on the COCO dataset. The last year to compete in BBOX object detection is 2017. The best way to demonstrate the accuracy of our network is to compare our mathematical scores with the leaderboards for this challenge. Please note that this is not a fair comparison because our network was only trained and evaluated on the six classes, not all 80.

position when sorted by AP if our network was on the leaderboard	name	mAP	mAP@ 0.5 IOU	mAP@ 0.75 IOU	mAP (small)	mAP (med)	mAP (Large)	AR@1	AR@1 0	AR@1 00	AR (small)	AR (med)	AR (Large)
1	Megvii (Face++)	52.5%	72.9%	58.7%	34.5%	55.6%	64.9%	39.3%	64.5%	69.0%	51.3%	72.7%	82.4%
2	UCenter	51.0%	70.6%	55.7%	32.6%	54.0%	64.0%	39.5%	64.0%	67.9%	49.0%	72.1%	82.7%
3	MSRA	50.4%	71.5%	56.3%	33.7%	52.9%	62.3%	37.9%	63.7%	69.0%	51.8%	72.3%	82.6%
4	FAIR Mask R-CNN	50.3%	71.9%	55.8%	32.6%	53.7%	62.4%	38.2%	62.2%	66.1%	47.7%	70.8%	79.9%
5	bharat_umd	48.1%	69.5%	53.5%	31.1%	51.5%	60.1%	36.6%	60.4%	64.8%	45.7%	69.7%	79.0%
6	Trimps-Soushen+QINIU	48.0%	67.9%	53.3%	30.9%	51.2%	60.5%	37.4%	61.2%	65.4%	47.2%	69.0%	79.8%
7	DANet	45.7%	67.4%	51.0%	28.3%	48.4%	58.7%	35.9%	58.8%	62.7%	43.0%	66.8%	78.2%
8	BUPT-Priv	43.5%	66.2%	47.4%	25.3%	47.7%	56.0%	33.9%	54.4%	58.1%	37.4%	63.0%	73.4%
9	DL61	42.1%	63.2%	46.7%	24.5%	45.8%	54.4%	33.8%	56.4%	60.7%	39.0%	65.5%	77.1%
10	DeNet	42.1%	61.2%	45.2%	22.3%	46.0%	58.0%	34.7%	55.0%	58.3%	37.3%	62.7%	75.6%
11	IL	41.6%	62.9%	45.6%	23.1%	44.9%	54.6%	34.0%	55.3%	59.1%	37.4%	63.6%	75.8%
12	VCA	41.5%	64.0%	45.4%	23.5%	44.9%	53.8%	33.6%	55.5%	59.2%	37.3%	63.9%	76.4%
13	LDL	41.3%	64.1%	45.1%	24.4%	44.2%	53.5%	33.6%	54.8%	58.0%	38.7%	62.2%	72.9%
14	PingAn AI Lab	41.1%	61.3%	45.6%	23.1%	45.3%	54.6%	33.7%	55.2%	59.8%	36.1%	65.0%	78.0%
15	DeepInsight	39.6%	60.8%	43.1%	20.4%	42.7%	52.7%	32.2%	52.1%	55.7%	33.0%	59.9%	74.1%
16	DGIST-FATRC	38.1%	59.8%	41.2%	17.7%	41.6%	52.7%	31.4%	50.4%	53.9%	29.8%	58.9%	73.0%
17	EE297 - Team 2 - Model	37.1%	56.4%	40.7%	14.3%	34.4%	51.4%	30.0%	52.6%	57.8%	31.1%	58.8%	71.4%
18	mcc_lab	35.2%	55.2%	37.9%	14.7%	39.0%	50.7%	31.0%	44.7%	45.7%	21.1%	51.0%	66.0%
19	ANLV	33.9%	55.2%	36.6%	19.4%	36.8%	42.8%	29.3%	46.3%	48.4%	30.3%	52.7%	60.1%
20	MCPRL	26.7%	46.5%	27.7%	10.2%	30.6%	37.5%	25.9%	37.5%	38.3%	15.5%	44.5%	54.9%
21	drl	23.4%	43.5%	22.9%	5.4%	24.4%	38.6%	23.1%	32.6%	33.3%	9.5%	35.9%	54.6%

Fig 5.1 - Top 20 leaderboard of Coco BBox Challenge 2017 [79]

As can be seen in Fig 5.1 above, if we were placed on the coco leaderboards, we would come in 17th place, which is very good considering that “DGIST-FATRC” used “ four models ensemble with the joint Soft-NMS”[80]

Soft-NMS is built into faster RCNN and should increase our mAP score by about 1.1%. This would come at the cost of some performance.[80], [81]

This is an excellent result as our score could easily be improved by any of the above methods.

6. Code/Appendix

Fig 3.1.1 - TF object detection models ordered by their speed. [70]

Model name	Speed (ms)	COCO mAP	Outputs
CenterNet MobileNetV2 FPN 512x512	6	23	Boxes
SSD MobileNet v2 320x320	19	20	Boxes
SSD MobileNet V2 FPNLite 320x320	22	22	Boxes
CenterNet Resnet50 V1 FPN 512x512	27	31	Boxes
CenterNet Resnet50 V2 512x512	27	30	Boxes
CenterNet Resnet101 V1 FPN 512x512	34	34	Boxes
EfficientDet D0 512x512	39	34	Boxes
SSD MobileNet V2 FPNLite 640x640	39	28	Boxes
SSD ResNet50 V1 FPN 640x640 (RetinaNet50)	46	34	Boxes
SSD MobileNet V1 FPN 640x640	48	29	Boxes
Faster R-CNN ResNet50 V1 640x640	53	29	Boxes
EfficientDet D1 640x640	54	38	Boxes
Faster R-CNN ResNet101 V1 640x640	55	32	Boxes
SSD ResNet101 V1 FPN 640x640 (RetinaNet101)	57	36	Boxes
Faster R-CNN ResNet152 V1 640x640	64	32	Boxes
Faster R-CNN ResNet50 V1 1024x1024	65	31	Boxes
Faster R-CNN ResNet50 V1 800x1333	65	32	Boxes
EfficientDet D2 768x768	67	42	Boxes
CenterNet HourGlass104 512x512	70	42	Boxes
Faster R-CNN ResNet101 V1 1024x1024	72	37	Boxes
CenterNet HourGlass104 Keypoints 512x512	76	40	Boxes
Faster R-CNN ResNet101 V1 800x1333	77	37	Boxes
SSD ResNet152 V1 FPN 640x640 (RetinaNet152)	80	35	Boxes
Faster R-CNN ResNet152 V1 1024x1024	85	38	Boxes
SSD ResNet50 V1 FPN 1024x1024 (RetinaNet50)	87	38	Boxes
EfficientDet D3 896x896	95	45	Boxes
Faster R-CNN ResNet152 V1 800x1333	101	37	Boxes
SSD ResNet101 V1 FPN 1024x1024 (RetinaNet101)	104	40	Boxes
SSD ResNet152 V1 FPN 1024x1024 (RetinaNet152)	111	40	Boxes
EfficientDet D4 1024x1024	133	49	Boxes
CenterNet HourGlass104 1024x1024	197	45	Boxes
Faster R-CNN Inception ResNet V2 640x640	206	38	Boxes
CenterNet HourGlass104 Keypoints 1024x1024	211	43	Boxes
EfficientDet D5 1280x1280	222	50	Boxes

Fig 3.3.1 - Diagram of CNN Model

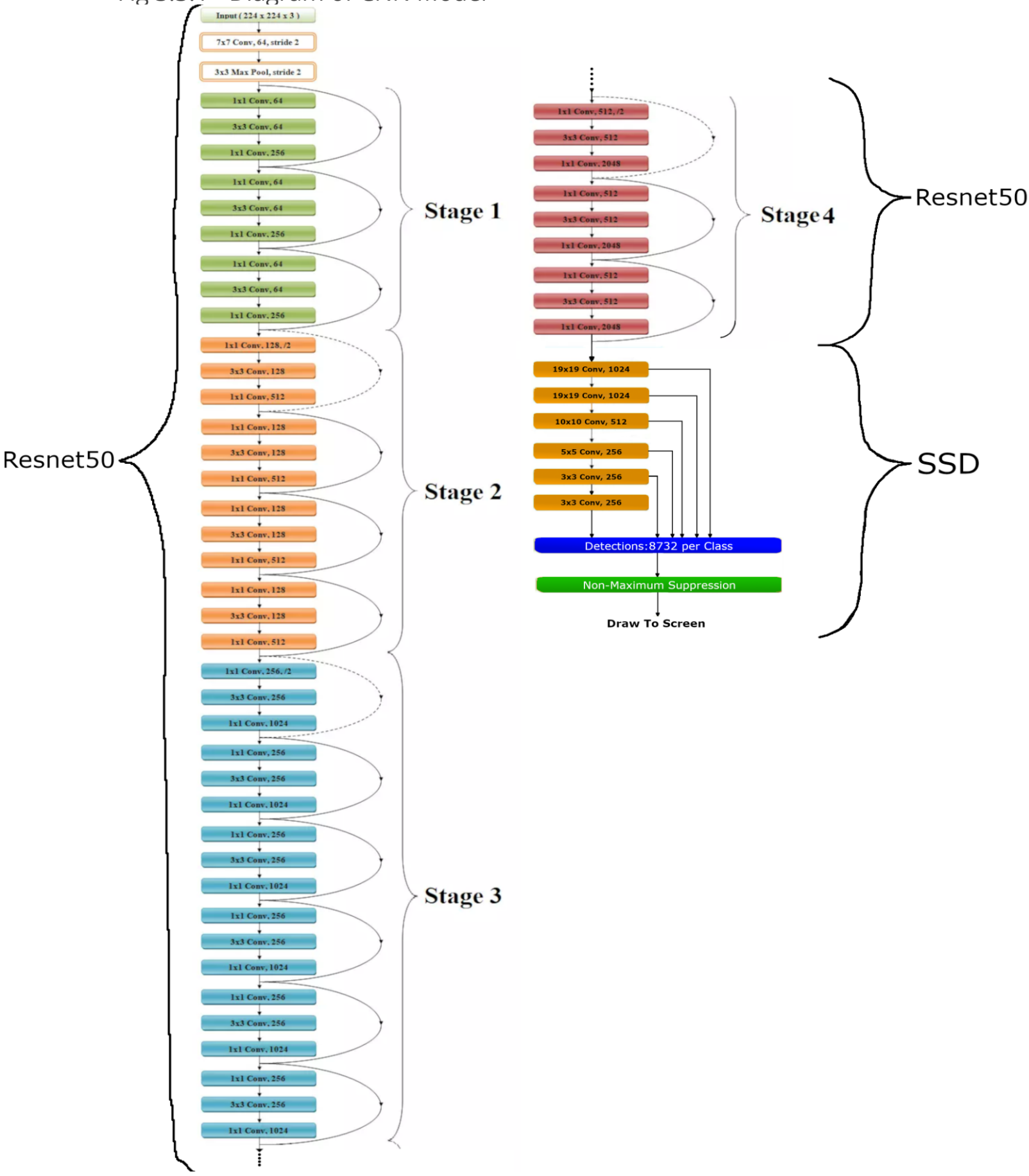


Fig 4.2.1 - example image label - 000000010363.xml

```
<annotation>
  <folder>VOC2014_instance/cat</folder>
  <filename>000000010363.jpg</filename>
  <source>
    <database>MS COCO 2014</database>
    <annotation>MS COCO 2014</annotation>
    <image>Flickr</image>
    <url>http://images.cocodataset.org/val2017/000000010363.jpg</url>
  </source>
  <size>
    <width>640</width>
    <height>361</height>
    <depth>3</depth>
  </size>
  <segmented>0</segmented>
  <object>
    <name>cat</name>
    <bndbox>
      <xmin>245.51</xmin>
      <ymin>108.3</ymin>
      <xmax>471.64</xmax>
      <ymax>243.98000000000002</ymax>
    </bndbox>
    <difficult>0</difficult>
  </object>
  <object>
    <name>bottle</name>
    <bndbox>
      <xmin>147.18</xmin>
      <ymin>28.8</ymin>
      <xmax>159.84</xmax>
      <ymax>66.41</ymax>
    </bndbox>
    <difficult>0</difficult>
  </object>
  <object>
    <name>car</name>
    <bndbox>
      <xmin>90.83</xmin>
      <ymin>27.17</ymin>
      <xmax>640.0</xmax>
      <ymax>361.0</ymax>
    </bndbox>
    <difficult>0</difficult>
  </object>
  <object>
    <name>bicycle</name>
    <bndbox>
      <xmin>492.55</xmin>
      <ymin>0.0</ymin>
      <xmax>635.9</xmax>
      <ymax>52.9</ymax>
    </bndbox>
    <difficult>0</difficult>
  </object>
</annotation>
```

Fig 4.2.2 - The preprocessing script - moveImages.py

```
#Created by Team 2 EE297
import os
import argparse
from pprint import pprint
import xml.etree.ElementTree as ET
from shutil import copyfile

# construct the argument parse and parse the arguments
ap = argparse.ArgumentParser()
ap.add_argument("-x", "--xmlPath", required=True,
	help="path to input xml directory")
ap.add_argument("-i", "--imgPath", required=True,
	help="path to input img directory")
ap.add_argument("-t", "--type", required=True,
	help="path to input video file")
args = vars(ap.parse_args())

#xmlPath = 'CocoDataset/trainLabels'
#imgPath = 'CocoDataset/train2017'
#type = 'train'

xmlPath = args["xmlPath"]
imgPath = args["imgPath"]
type = args["type"]

if not os.path.exists('images'):
    os.makedirs('images')
if not os.path.exists("images\\"+type):
    os.makedirs("images\\"+type)
if os.path.exists('tmp'):
	tmpFiles = os.listdir('tmp')
	for file in tmpFiles:
		if file.endswith(".xml"):
			os.remove(os.path.join('tmp', file))
else:
    os.makedirs('tmp')


classes = []
files = []
labels = os.listdir(xmlPath)
for curClass in labels:
	if curClass=="other":
		continue
	classes.append(curClass)
	curPath = os.path.join(xmlPath, curClass)
	curFiles = os.listdir(curPath)
	files += [curPath + "\\" + file for file in curFiles]

print( "found "+str(len(files))+" files" )
pprint( classes )

for file in files:
	newFileName = "tmp\\"+os.path.basename(file)
	copyfile(file, newFileName)
	tree = ET.parse(newFileName)
```

```
        root = tree.getroot()
        for object in root.findall('object'):
                name = object.find('name').text
                if name not in classes:
                        root.remove(object)
        tree.write(newFileName)

files = os.listdir("tmp")
for file in files:
        imgFile = os.path.splitext(file)[0]+".jpg"
        copyfile("tmp//"+file, "images\\"+type+"\\"+file)
        copyfile(os.path.join(imgPath, imgFile), "images\\"+type+"\\"+imgFile)
#pprint(files)
```

Fig 4.3.1 - Pipeline.config for 70k-6Batch-400Ksteps

```
model {
  ssd {
    num_classes: 6
    image_resizer {
      fixed_shape_resizer {
        height: 640
        width: 640
      }
    }
    feature_extractor {
      type: "ssd_resnet50_v1_fpn_keras"
      depth_multiplier: 1.0
      min_depth: 16
      conv_hyperparams {
        regularizer {
          l2_regularizer {
            weight: 0.0004
          }
        }
        initializer {
          truncated_normal_initializer {
            mean: 0.0
            stddev: 0.03
          }
        }
        activation: RELU_6
        batch_norm {
          decay: 0.997
          scale: true
          epsilon: 0.001
        }
      }
      override_base_feature_extractor_hyperparams: true
      fpn {
        min_level: 3
        max_level: 7
      }
    }
    box_coder {
      faster_rcnn_box_coder {
        y_scale: 10.0
        x_scale: 10.0
        height_scale: 5.0
        width_scale: 5.0
      }
    }
    matcher {
      argmax_matcher {
        matched_threshold: 0.5
        unmatched_threshold: 0.5
        ignore_thresholds: false
        negatives_lower_than_unmatched: true
        force_match_for_each_row: true
        use_matmul_gather: true
      }
    }
```

```
    similarity_calculator {
      iou_similarity {
      }
    }
    box_predictor {
      weight_shared_convolutional_box_predictor {
        conv_hyperparams {
          regularizer {
            l2_regularizer {
              weight: 0.0004
            }
          }
          initializer {
            random_normal_initializer {
              mean: 0.0
              stddev: 0.01
            }
          }
          activation: RELU_6
          batch_norm {
            decay: 0.997
            scale: true
            epsilon: 0.001
          }
        }
        depth: 256
        num_layers_before_predictor: 4
        kernel_size: 3
        class_prediction_bias_init: -4.6
      }
    }
    anchor_generator {
      multiscale_anchor_generator {
        min_level: 3
        max_level: 7
        anchor_scale: 4.0
        aspect_ratios: 1.0
        aspect_ratios: 2.0
        aspect_ratios: 0.5
        scales_per_octave: 2
      }
    }
    post_processing {
      batch_non_max_suppression {
        score_threshold: 1e-08
        iou_threshold: 0.6
        max_detections_per_class: 100
        max_total_detections: 100
        use_static_shapes: false
      }
      score_converter: SIGMOID
    }
    normalize_loss_by_num_matches: true
    loss {
      localization_loss {
        weighted_smooth_l1 {
        }
      }
```

```
      classification_loss {
        weighted_sigmoid_focal {
          gamma: 2.0
          alpha: 0.25
        }
      }
      classification_weight: 1.0
      localization_weight: 1.0
    }
    encode_background_as_zeros: true
    normalize_loc_loss_by_codesize: true
    inplace_batchnorm_update: true
    freeze_batchnorm: false
  }
}
train_config {
  batch_size: 6
  data_augmentation_options {
    random_horizontal_flip {
    }
  }
  data_augmentation_options {
    random_crop_image {
      min_object_covered: 0.0
      min_aspect_ratio: 0.75
      max_aspect_ratio: 3.0
      min_area: 0.75
      max_area: 1.0
      overlap_thresh: 0.0
    }
  }
  sync_replicas: true
  optimizer {
    momentum_optimizer {
      learning_rate {
        cosine_decay_learning_rate {
          learning_rate_base: 0.04
          total_steps: 500000
          warmup_learning_rate: 0.013333
          warmup_steps: 2000
        }
      }
      momentum_optimizer_value: 0.9
    }
    use_moving_average: false
  }
  fine_tune_checkpoint:
"pre-trained-models/ssd_resnet50_v1_fpn_640x640_coco17_tpu-8/checkpoint/ckpt-0"
  num_steps: 500000
  startup_delay_steps: 0.0
  replicas_to_aggregate: 8
  max_number_of_boxes: 100
  unpad_groundtruth_tensors: false
  fine_tune_checkpoint_type: "detection"
  use_bfloat16: false
  fine_tune_checkpoint_version: V2
}
train_input_reader {
```

```
  label_map_path: "annotations/label_map.pbtxt"
  tf_record_input_reader {
    input_path: "annotations/train.record"
  }
}
eval_config {
  metrics_set: "coco_detection_metrics"
  use_moving_averages: false
}
eval_input_reader {
  label_map_path: "annotations/label_map.pbtxt"
  shuffle: false
  num_epochs: 1
  tf_record_input_reader {
    input_path: "annotations/test.record"
  }
}
```

Fig 4.3.2 - console output while training

```
I0430 18:55:34.074477 11448 model_lib_v2.py:679] Step 327400 per-step time 1.057s loss=0.547
INFO:tensorflow:Step 327500 per-step time 0.981s loss=0.725
I0430 18:57:14.976833 11448 model_lib_v2.py:679] Step 327500 per-step time 0.981s loss=0.725
INFO:tensorflow:Step 327600 per-step time 0.940s loss=0.520
I0430 18:58:54.492838 11448 model_lib_v2.py:679] Step 327600 per-step time 0.940s loss=0.520
INFO:tensorflow:Step 327700 per-step time 1.097s loss=0.540
I0430 19:00:37.188071 11448 model_lib_v2.py:679] Step 327700 per-step time 1.097s loss=0.540
INFO:tensorflow:Step 327800 per-step time 0.996s loss=0.458
I0430 19:02:21.412800 11448 model_lib_v2.py:679] Step 327800 per-step time 0.996s loss=0.458
INFO:tensorflow:Step 327900 per-step time 1.052s loss=0.523
I0430 19:04:03.260918 11448 model_lib_v2.py:679] Step 327900 per-step time 1.052s loss=0.523
INFO:tensorflow:Step 328000 per-step time 0.991s loss=0.559
I0430 19:05:50.958060 11448 model_lib_v2.py:679] Step 328000 per-step time 0.991s loss=0.559
INFO:tensorflow:Step 328100 per-step time 1.069s loss=0.650
I0430 19:07:40.670145 11448 model_lib_v2.py:679] Step 328100 per-step time 1.069s loss=0.650
INFO:tensorflow:Step 328200 per-step time 1.059s loss=0.573
I0430 19:09:22.047366 11448 model_lib_v2.py:679] Step 328200 per-step time 1.059s loss=0.573
INFO:tensorflow:Step 328300 per-step time 1.149s loss=0.530
I0430 19:11:03.138919 11448 model_lib_v2.py:679] Step 328300 per-step time 1.149s loss=0.530
INFO:tensorflow:Step 328400 per-step time 0.986s loss=0.519
I0430 19:12:41.476730 11448 model_lib_v2.py:679] Step 328400 per-step time 0.986s loss=0.519
INFO:tensorflow:Step 328500 per-step time 0.973s loss=0.525
I0430 19:14:19.673863 11448 model_lib_v2.py:679] Step 328500 per-step time 0.973s loss=0.525
INFO:tensorflow:Step 328600 per-step time 1.058s loss=0.528
I0430 19:15:56.690136 11448 model_lib_v2.py:679] Step 328600 per-step time 1.058s loss=0.528
INFO:tensorflow:Step 328700 per-step time 1.027s loss=0.606
I0430 19:17:38.043495 11448 model_lib_v2.py:679] Step 328700 per-step time 1.027s loss=0.606
INFO:tensorflow:Step 328800 per-step time 1.298s loss=0.607
I0430 19:19:21.263647 11448 model_lib_v2.py:679] Step 328800 per-step time 1.298s loss=0.607
INFO:tensorflow:Step 328900 per-step time 1.017s loss=0.536
I0430 19:21:06.569310 11448 model_lib_v2.py:679] Step 328900 per-step time 1.017s loss=0.536
INFO:tensorflow:Step 329000 per-step time 1.035s loss=0.603
I0430 19:22:50.104161 11448 model_lib_v2.py:679] Step 329000 per-step time 1.035s loss=0.603
INFO:tensorflow:Step 329100 per-step time 0.984s loss=0.560
I0430 19:24:35.699778 11448 model_lib_v2.py:679] Step 329100 per-step time 0.984s loss=0.560
INFO:tensorflow:Step 329200 per-step time 1.063s loss=0.523
I0430 19:26:24.849883 11448 model_lib_v2.py:679] Step 329200 per-step time 1.063s loss=0.523
INFO:tensorflow:Step 329300 per-step time 0.965s loss=0.640
I0430 19:28:08.986803 11448 model_lib_v2.py:679] Step 329300 per-step time 0.965s loss=0.640
INFO:tensorflow:Step 329400 per-step time 1.081s loss=0.476
I0430 19:29:56.613634 11448 model_lib_v2.py:679] Step 329400 per-step time 1.081s loss=0.476
INFO:tensorflow:Step 329500 per-step time 1.076s loss=0.527
I0430 19:31:44.691948 11448 model_lib_v2.py:679] Step 329500 per-step time 1.076s loss=0.527
INFO:tensorflow:Step 329600 per-step time 1.043s loss=0.505
I0430 19:33:29.337977 11448 model_lib_v2.py:679] Step 329600 per-step time 1.043s loss=0.505
INFO:tensorflow:Step 329700 per-step time 1.166s loss=0.507
I0430 19:35:12.470080 11448 model_lib_v2.py:679] Step 329700 per-step time 1.166s loss=0.507
INFO:tensorflow:Step 329800 per-step time 1.106s loss=0.520
I0430 19:36:55.909403 11448 model_lib_v2.py:679] Step 329800 per-step time 1.106s loss=0.520
INFO:tensorflow:Step 329900 per-step time 0.928s loss=0.474
I0430 19:38:32.040274 11448 model_lib_v2.py:679] Step 329900 per-step time 0.928s loss=0.474
INFO:tensorflow:Step 330000 per-step time 1.008s loss=0.473
I0430 19:40:08.577508 11448 model_lib_v2.py:679] Step 330000 per-step time 1.008s loss=0.473
INFO:tensorflow:Step 330100 per-step time 1.031s loss=0.475
I0430 19:41:50.137417 11448 model_lib_v2.py:679] Step 330100 per-step time 1.031s loss=0.475
INFO:tensorflow:Step 330200 per-step time 1.017s loss=0.616
I0430 19:43:27.372262 11448 model_lib_v2.py:679] Step 330200 per-step time 1.017s loss=0.616
INFO:tensorflow:Step 330300 per-step time 1.023s loss=0.562
I0430 19:45:07.251070 11448 model_lib_v2.py:679] Step 330300 per-step time 1.023s loss=0.562
INFO:tensorflow:Step 330400 per-step time 1.222s loss=0.492
I0430 19:46:49.416431 11448 model_lib_v2.py:679] Step 330400 per-step time 1.222s loss=0.492
INFO:tensorflow:Step 330500 per-step time 1.266s loss=0.545
I0430 19:48:36.145564 11448 model_lib_v2.py:679] Step 330500 per-step time 1.266s loss=0.545
INFO:tensorflow:Step 330600 per-step time 0.996s loss=0.549
I0430 19:50:22.357280 11448 model_lib_v2.py:679] Step 330600 per-step time 0.996s loss=0.549
INFO:tensorflow:Step 330700 per-step time 1.145s loss=0.541
I0430 19:52:04.740659 11448 model_lib_v2.py:679] Step 330700 per-step time 1.145s loss=0.541
INFO:tensorflow:Step 330800 per-step time 0.883s loss=0.478
I0430 19:53:48.736914 11448 model_lib_v2.py:679] Step 330800 per-step time 0.883s loss=0.478
INFO:tensorflow:Step 330900 per-step time 0.983s loss=0.551
I0430 19:55:30.612444 11448 model_lib_v2.py:679] Step 330900 per-step time 0.983s loss=0.551
INFO:tensorflow:Step 331000 per-step time 1.034s loss=0.485
I0430 19:57:10.669915 11448 model_lib_v2.py:679] Step 331000 per-step time 1.034s loss=0.485
INFO:tensorflow:Step 331100 per-step time 0.900s loss=0.462
I0430 19:58:54.671563 11448 model_lib_v2.py:679] Step 331100 per-step time 0.900s loss=0.462
INFO:tensorflow:Step 331200 per-step time 0.972s loss=0.474
I0430 20:00:38.951684 11448 model_lib_v2.py:679] Step 331200 per-step time 0.972s loss=0.474
INFO:tensorflow:Step 331300 per-step time 1.039s loss=0.558
I0430 20:02:25.084730 11448 model_lib_v2.py:679] Step 331300 per-step time 1.039s loss=0.558
INFO:tensorflow:Step 331400 per-step time 0.918s loss=0.618
I0430 20:04:07.008391 11448 model_lib_v2.py:679] Step 331400 per-step time 0.918s loss=0.618
```

Fig 4.4.1 - IOU Diagram

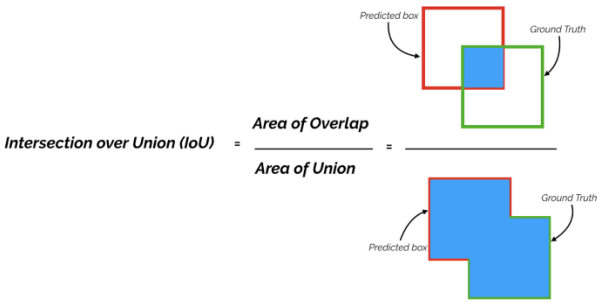


Fig 4.4.2 - Precision vs Recall diagram

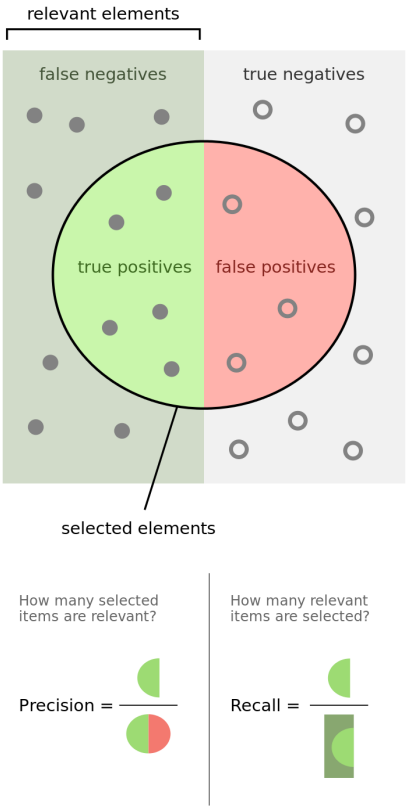


Fig 4.4.3 - object_detection_video.py

```
#!/usr/bin/env python
#Created by Team 2 EE297
#Detect Objects Using specified Neural Network inside a video file
# loads the video a frame at a time, then run it through the detection model and visualises
the
# detection results, including the keypoints then save it to the output video
#
# please note this will take a long time before it returns
#
#================================

import os
import argparse
import tensorflow as tf
import numpy as np
from object_detection.utils import label_map_util
from object_detection.utils import config_util
from object_detection.utils import visualization_utils as viz_utils
from object_detection.builders import model_builder

# construct the argument parse and parse the arguments
ap = argparse.ArgumentParser()
ap.add_argument("-m", "--model", required=True,
	help="path to model directory")
ap.add_argument("-v", "--video", required=True,
	help="path to the input video")
args = vars(ap.parse_args())

#constructs the paths
PATH_TO_CKPT = os.path.join(args["model"], 'checkpoint/')
PATH_TO_CFG = os.path.join(args["model"], 'pipeline.config')
PATH_TO_LABELS = '../annotations/label_map.pbtxt'

# %%
# Load the model
# ~~~~~~~~~~~~~~
# Next, we load the model

os.environ['TF_CPP_MIN_LOG_LEVEL'] = '2'    # Suppress TensorFlow logging
tf.get_logger().setLevel('ERROR')           # Suppress TensorFlow logging (2)

# Enable GPU dynamic memory allocation
gpus = tf.config.experimental.list_physical_devices('GPU')
for gpu in gpus:
	tf.config.experimental.set_memory_growth(gpu, True)

# Load pipeline config and build a detection model
configs = config_util.get_configs_from_pipeline_file(PATH_TO_CFG)
model_config = configs['model']
detection_model = model_builder.build(model_config=model_config, is_training=False)

# Restore checkpoint
ckpt = tf.compat.v2.train.Checkpoint(model=detection_model)
ckpt.restore(os.path.join(PATH_TO_CKPT, 'ckpt-0')).expect_partial()

@tf.function
```

```
def detect_fn(image):
        """Detect objects in image."""

        image, shapes = detection_model.preprocess(image)
        prediction_dict = detection_model.predict(image, shapes)
        detections = detection_model.postprocess(prediction_dict, shapes)

        return detections, prediction_dict, tf.reshape(shapes, [-1])


# %%
# Load label map data (for plotting)
# ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
# Label maps correspond index numbers to category names, so that when our CNN
# predicts `3`, we know that this corresponds to `Car`.
# Here we use label_map_util function
category_index = label_map_util.create_category_index_from_labelmap(PATH_TO_LABELS,

                                    use_display_name=True)

# %%
# Define input video
# ~~~~~~~~~~~~~~~~~~~~~~
# We will use `OpenCV <https://pypi.org/project/opencv-python/>`_ to open the video
import cv2
cap = cv2.VideoCapture(args["video"])
fps = cap.get(cv2.CAP_PROP_FPS)

#the output video is placed in the model directory with the name of the input video
#and with the framerate of the input video
newVideoName = os.path.basename(args["video"]).split('.')[0]+".avi"
newVideoPath = os.path.join(args["model"], newVideoName)
print(newVideoPath) #debug

# Define the codec and create VideoWriter object
fourcc = cv2.VideoWriter_fourcc(*'DIVX')
out = cv2.VideoWriter(newVideoPath,fourcc, fps, (800,600))

# Main Loop
# ~~~~~~~~~~~~~~~~~~~~~~~~~~
#runs until finished or killed by ctrl+c
while True:
        # loads a single frame at time
        ret, image_np = cap.read()

        #checks if there is a next frame
        if ret == False:
                break

        # Expand dimensions since the model expects images to have shape: [1, None, None, 3]
        image_np_expanded = np.expand_dims(image_np, axis=0)

        #feeds the image into the CNN and gets back the predictions
        input_tensor = tf.convert_to_tensor(np.expand_dims(image_np, 0), dtype=tf.float32)
        detections, predictions_dict, shapes = detect_fn(input_tensor)

        #offsets the label
        label_id_offset = 1
```

```
        #duplicates the image so that the original doesnt get modified
        image_np_with_detections = image_np.copy()

        #uses visualization_utils to overlay the predictions onto the new
image(image_np_with_detections)
        viz_utils.visualize_boxes_and_labels_on_image_array(
                image_np_with_detections,
                detections['detection_boxes'][0].numpy(),
                (detections['detection_classes'][0].numpy() + label_id_offset).astype(int),
                detections['detection_scores'][0].numpy(),
                category_index,
                use_normalized_coordinates=True,
                max_boxes_to_draw=200,
                min_score_thresh=.30,
                agnostic_mode=False)

        # append the new frame(image_np_with_detections) to the output video
        # while resizing it the the desired dimensions
        out.write(cv2.resize(image_np_with_detections, (800, 600)))
#finished so releasing resources
print("Done")
cap.release()
out.release()
```

Fig 4.4.4 - object_detection_camera.py

```
#!/usr/bin/env python
# Created by Team 2 EE297
# Detect Objects Using specified Neural Network inside a video stream(webcam)
# loads the video, a frame at a time, then run it through the detection model and visualises
the
# detection results, including the key points, then displays it on the screen
#
#================================
import cv2
import os
import argparse
import tensorflow as tf
import numpy as np
from object_detection.utils import label_map_util
from object_detection.utils import config_util
from object_detection.utils import visualization_utils as viz_utils
from object_detection.builders import model_builder

# construct the argument parse and parse the arguments
ap = argparse.ArgumentParser()
ap.add_argument("-m", "--model", required=True,
	help="path to model directory")
ap.add_argument("-w", "--webcam", type=int, required=True,
	help="the port number the webcam is connected to normally 0")
args = vars(ap.parse_args())

PATH_TO_CKPT = os.path.join(args["model"], 'checkpoint/')
PATH_TO_CFG = os.path.join(args["model"], 'pipeline.config')
PATH_TO_LABELS = '../annotations/label_map.pbtxt'
# %%
# Load the model
# ~~~~~~~~~~~~~~
# Next, we load the model

os.environ['TF_CPP_MIN_LOG_LEVEL'] = '2'    # Suppress TensorFlow logging
tf.get_logger().setLevel('ERROR')                  # Suppress TensorFlow logging (2)

# Enable GPU dynamic memory allocation
gpus = tf.config.experimental.list_physical_devices('GPU')
for gpu in gpus:
	tf.config.experimental.set_memory_growth(gpu, True)

# Load pipeline config and build a detection model
configs = config_util.get_configs_from_pipeline_file(PATH_TO_CFG)
model_config = configs['model']
detection_model = model_builder.build(model_config=model_config, is_training=False)

# Restore checkpoint
ckpt = tf.compat.v2.train.Checkpoint(model=detection_model)
ckpt.restore(os.path.join(PATH_TO_CKPT, 'ckpt-0')).expect_partial()

@tf.function
def detectObject(image):
	image, shapes = detection_model.preprocess(image)
	prediction_dict = detection_model.predict(image, shapes)
	detections = detection_model.postprocess(prediction_dict, shapes)
```

```
        return detections, prediction_dict, tf.reshape(shapes, [-1])


# %%
# Load label map data (for displaying)
# ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
# Label maps correspond index numbers to category names, so that when our CNN
# predicts `3`, we know that this corresponds to `Car`.
# Here we use label_map_util function
category_index = label_map_util.create_category_index_from_labelmap(PATH_TO_LABELS,

                                    use_display_name=True)

# %%
# Define the video stream
# ~~~~~~~~~~~~~~~~~~~~~~~
# We will use `OpenCV <https://pypi.org/project/opencv-python/>`_ to capture the video stream
cap = cv2.VideoCapture(args["webcam"], cv2.CAP_DSHOW)

# Main Loop
# ~~~~~~~~~~~~~~~~~~~~~~~~~~~~
#runs until q is pressed or killed by ctrl+c
while True:
        # Read frame from camera
        ret, image_np = cap.read()

        # Expand dimensions since the model expects images to have shape: [1, None, None, 3]
        image_np_expanded = np.expand_dims(image_np, axis=0)

        #feeds the image into the CNN and gets back the predictions
        input_tensor = tf.convert_to_tensor(np.expand_dims(image_np, 0), dtype=tf.float32)
        detections, predictions_dict, shapes = detectObject(input_tensor)

        #offsets the label
        label_id_offset = 1

        #duplicates the image so that the original doesnt get modified
        image_np_with_detections = image_np.copy()

        #uses visualization_utils to overlay the predictions onto the new
image(image_np_with_detections)
        viz_utils.visualize_boxes_and_labels_on_image_array(
                image_np_with_detections,
                detections['detection_boxes'][0].numpy(),
                (detections['detection_classes'][0].numpy() + label_id_offset).astype(int),
                detections['detection_scores'][0].numpy(),
                category_index,
                use_normalized_coordinates=True,
                max_boxes_to_draw=200,
                min_score_thresh=.30,
                agnostic_mode=False)

        # Display the output frame with the models input dimensions
        cv2.imshow('object detection', cv2.resize(image_np_with_detections, (800, 600)))

        if cv2.waitKey(25) & 0xFF == ord('q'):
                break
```

```
#finished so releasing resources
cap.release()
cv2.destroyAllWindows()
```

Fig 4.6.4 - Out of memory error

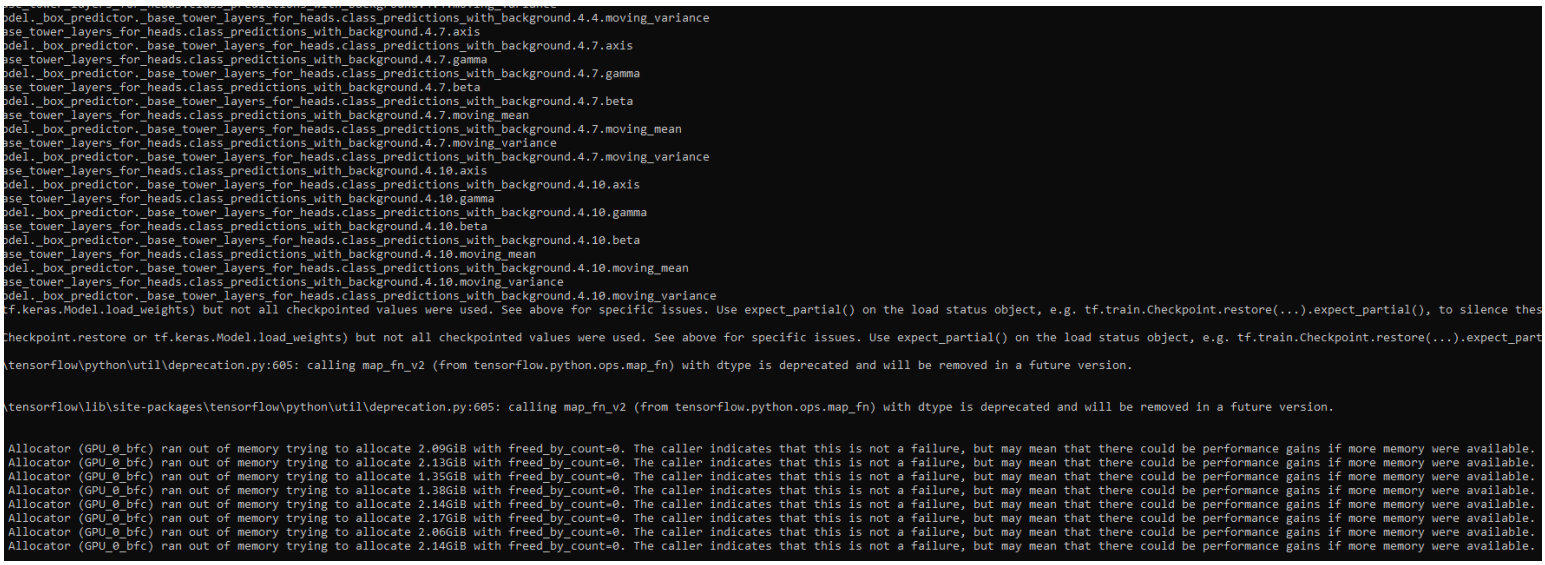


Fig 4.6.5 - 4,000 images - Batch of 1 - 25,000 Steps

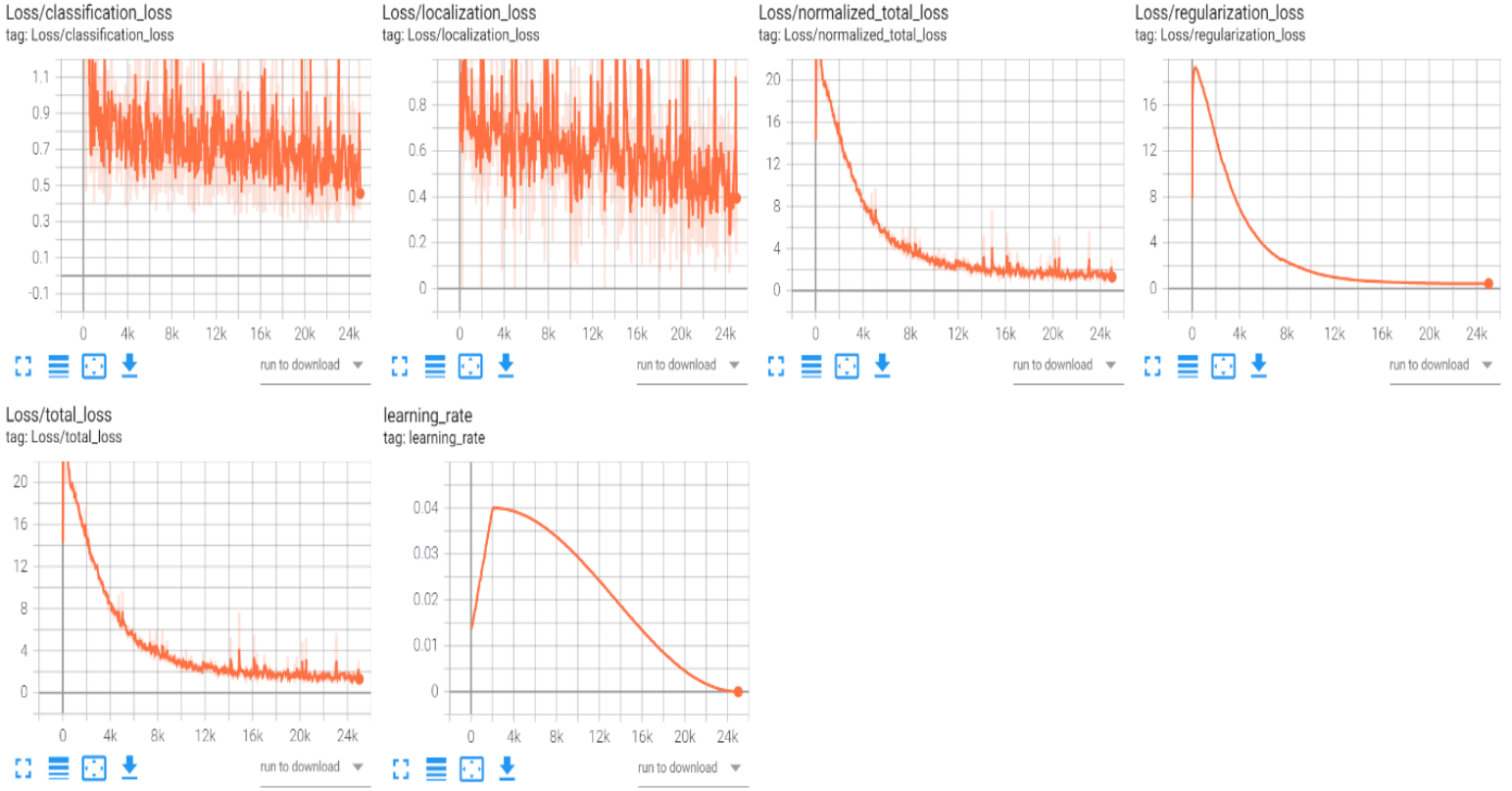
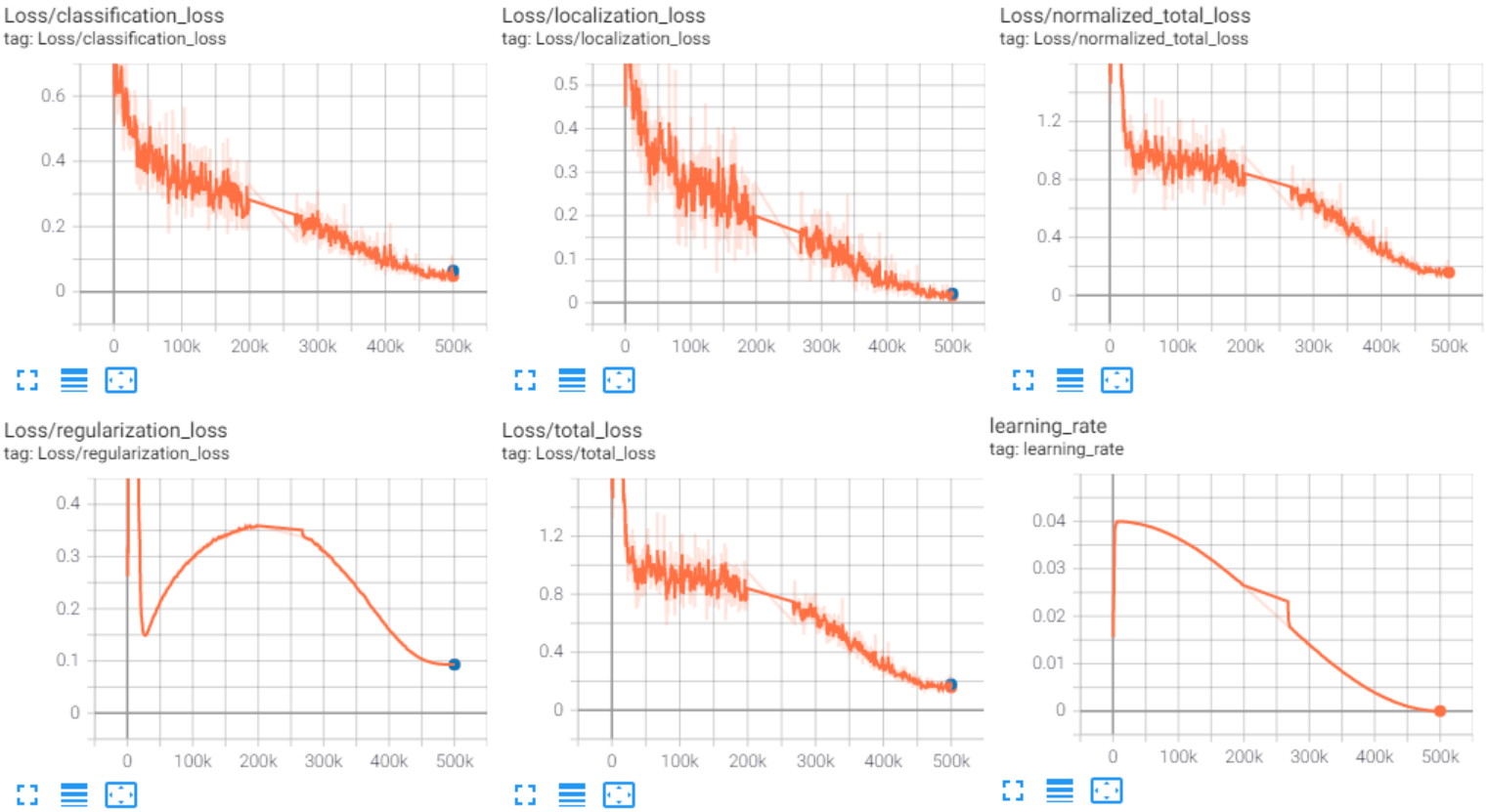


Fig 4.6.6 - 4,000 images - Batch of 6 - 500,000 steps



Note graph is missing some data in the center as it got corrupted by a memory leak, so it was removed.

Fig 4.6.7 - 70,000 images -Batch of 4 images - 25,000 Steps

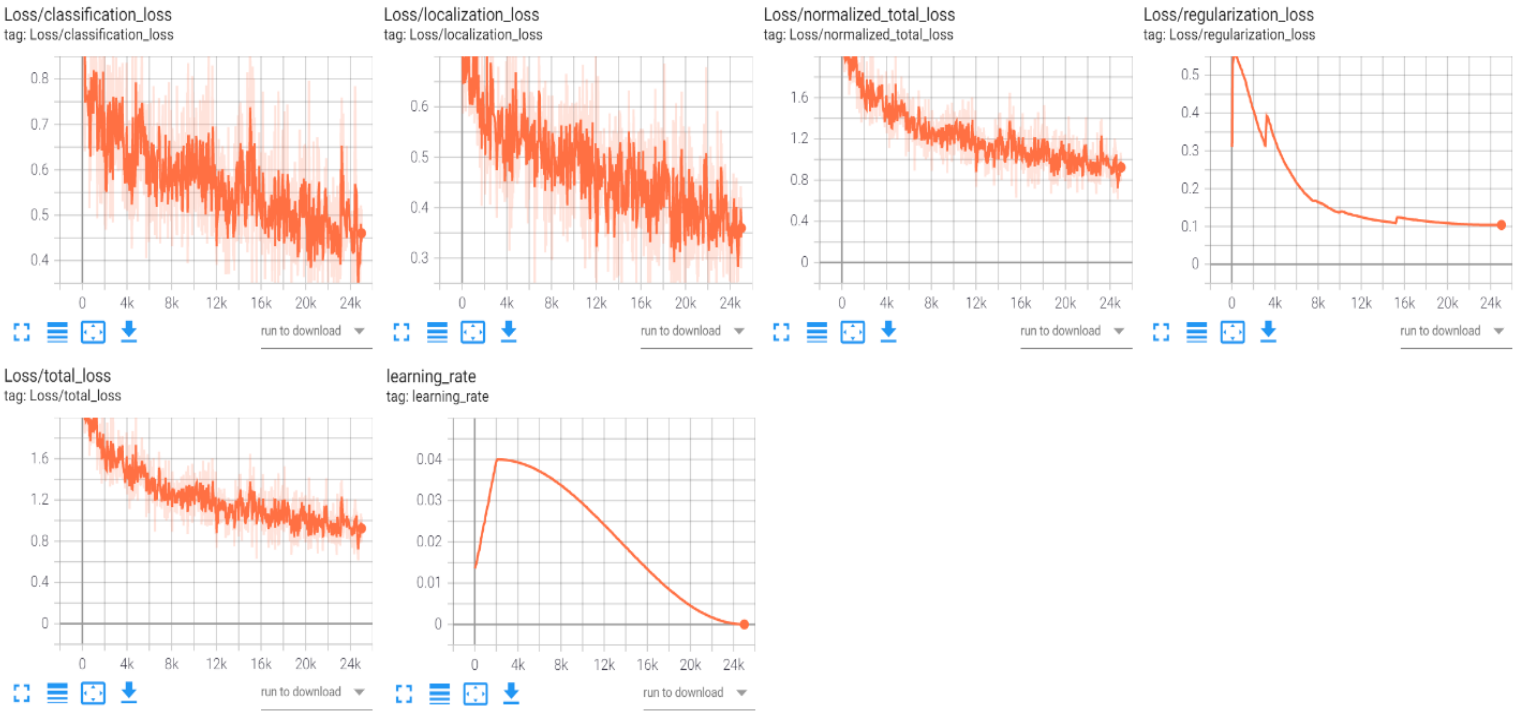


Fig 4.6.8 - 70,000 images - Batch of 5 images - 60,000 Steps

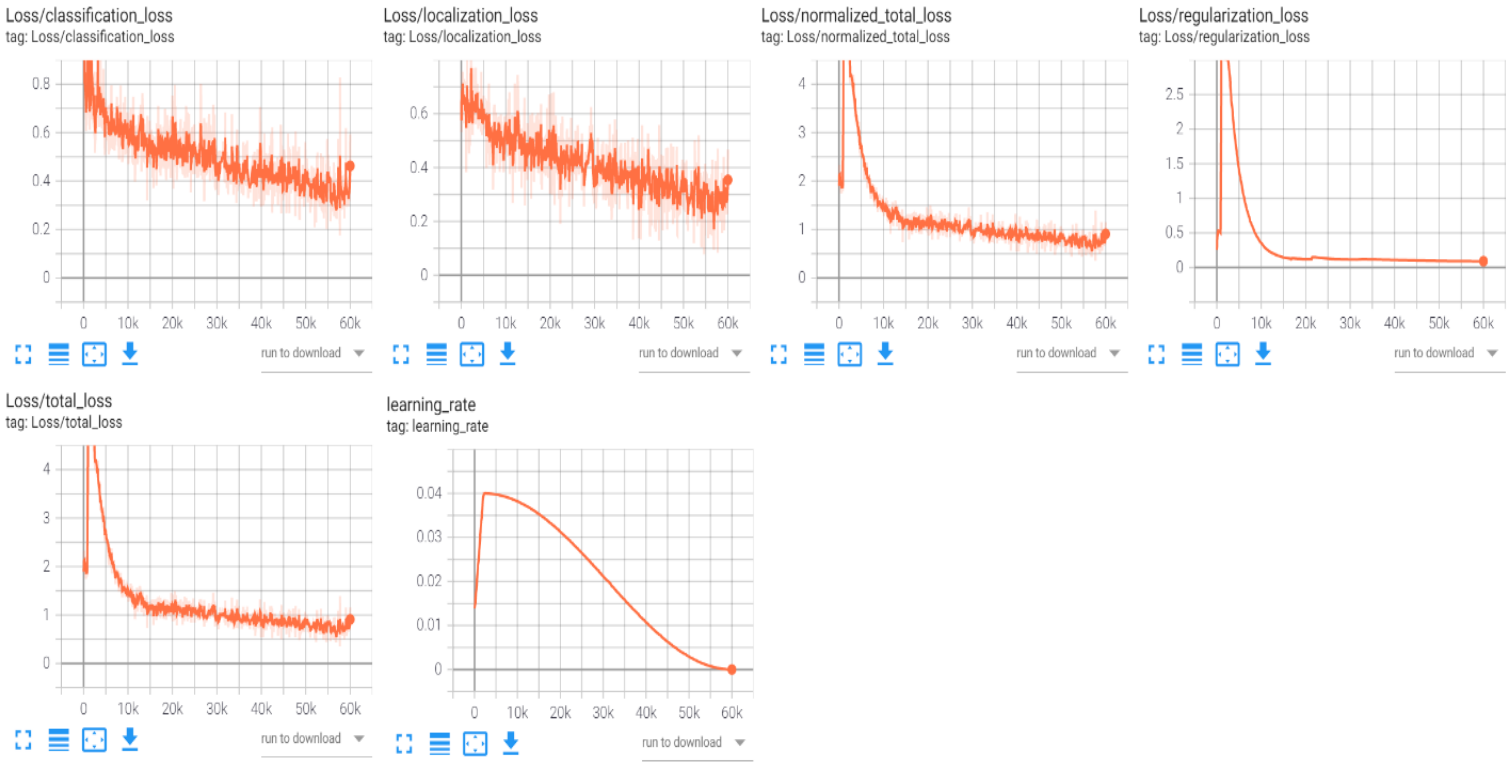


Fig 4.6.9 - 70,000 Images - Batch of 6 images - 150,000 Steps

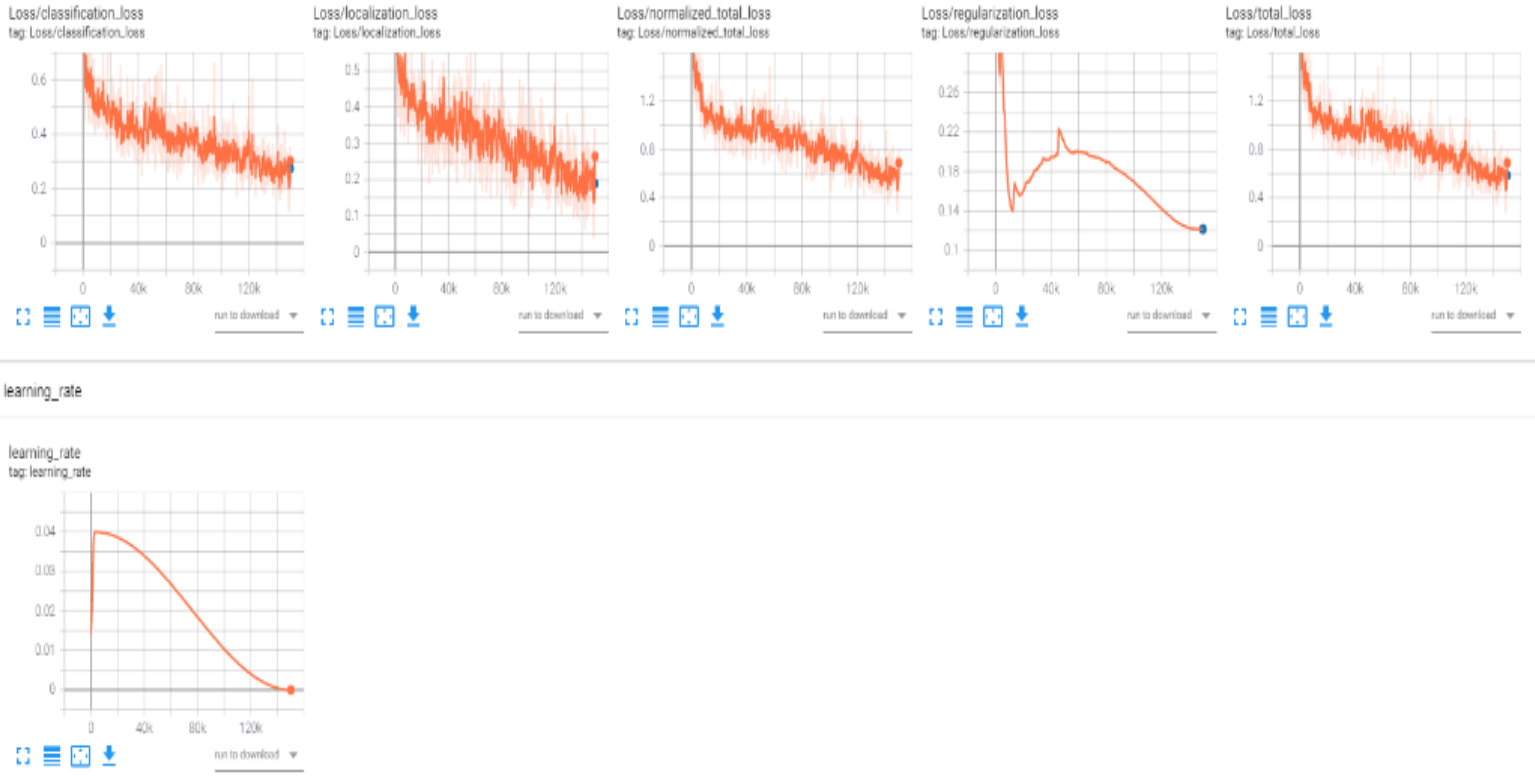


Fig 4.6.10 - 70,00 images - Batch of 6 images - 400,000 steps - halted

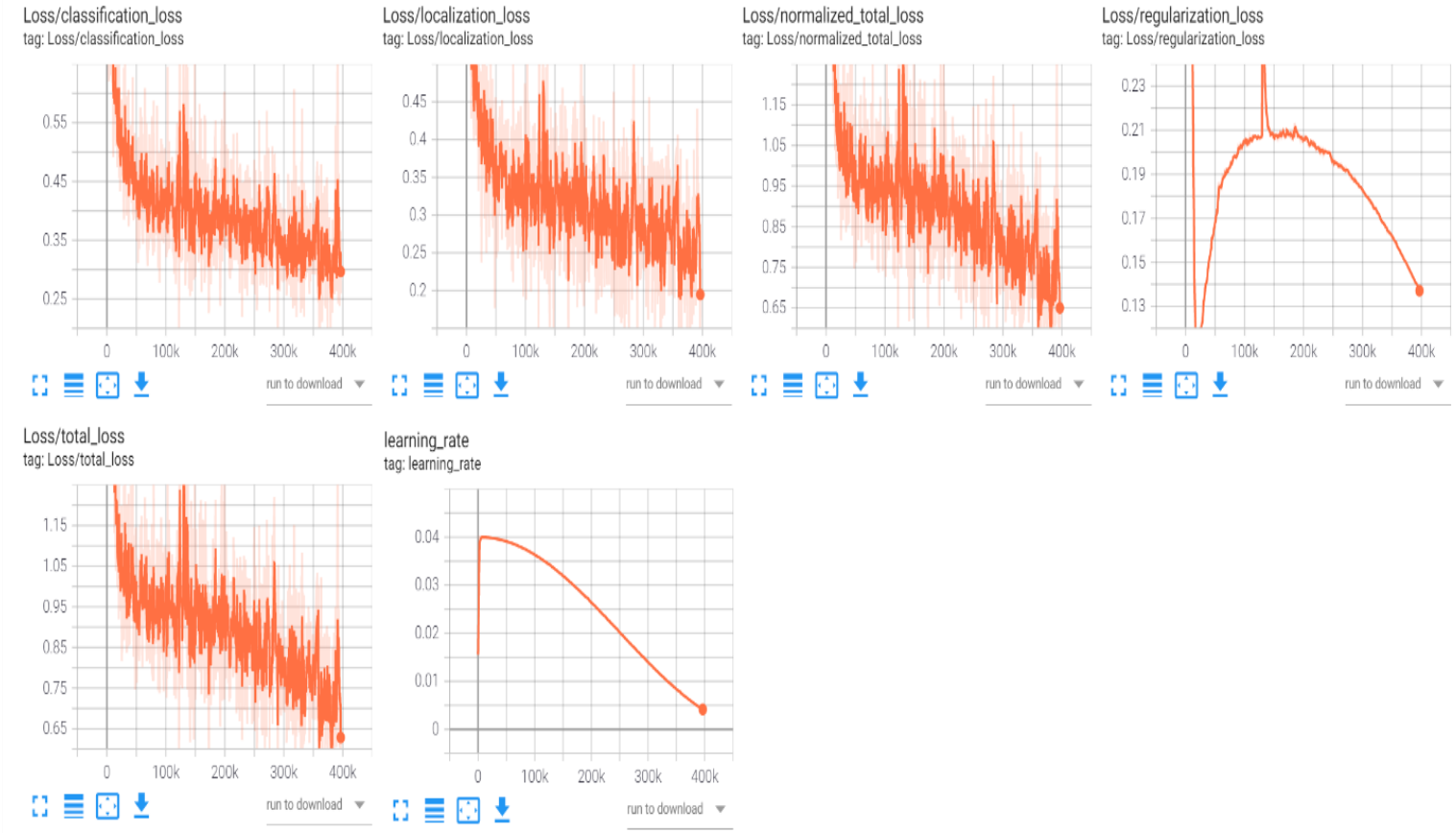


Fig 4.6.11 - 70,000 images - batch of 6 images - 500,000

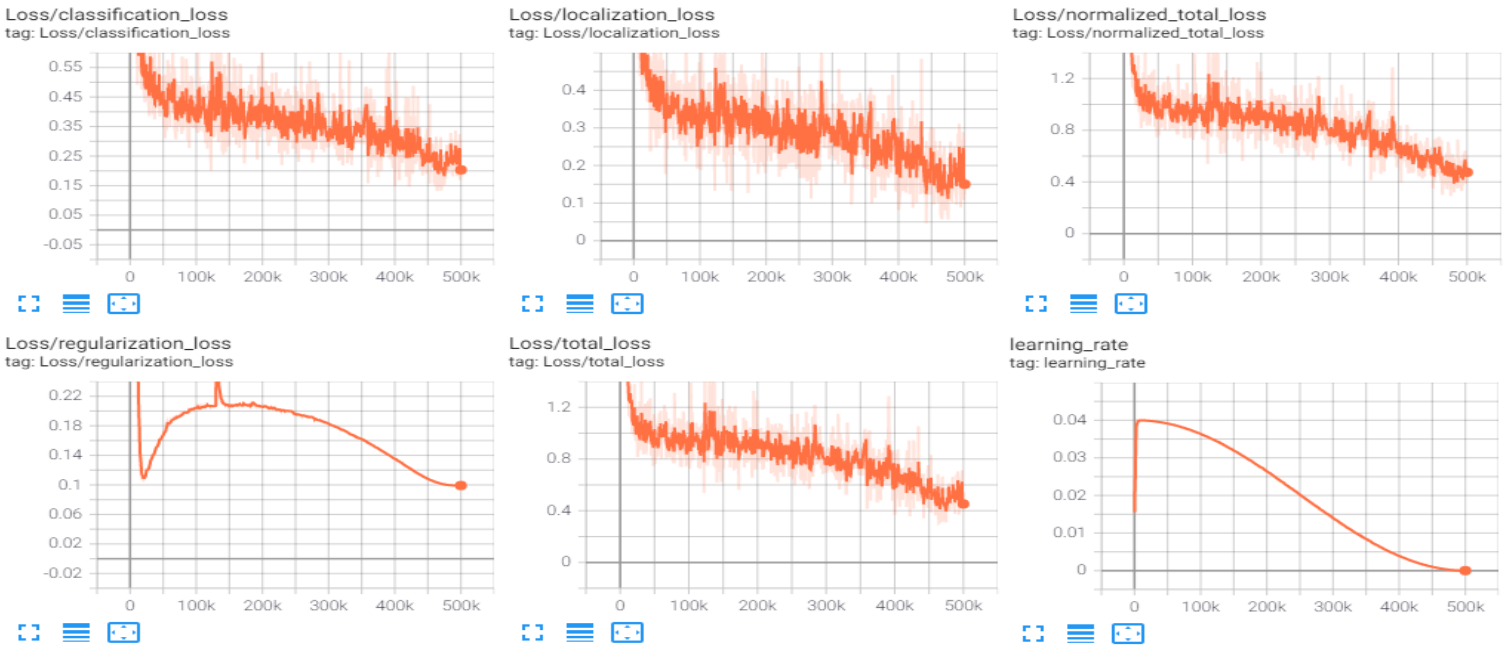
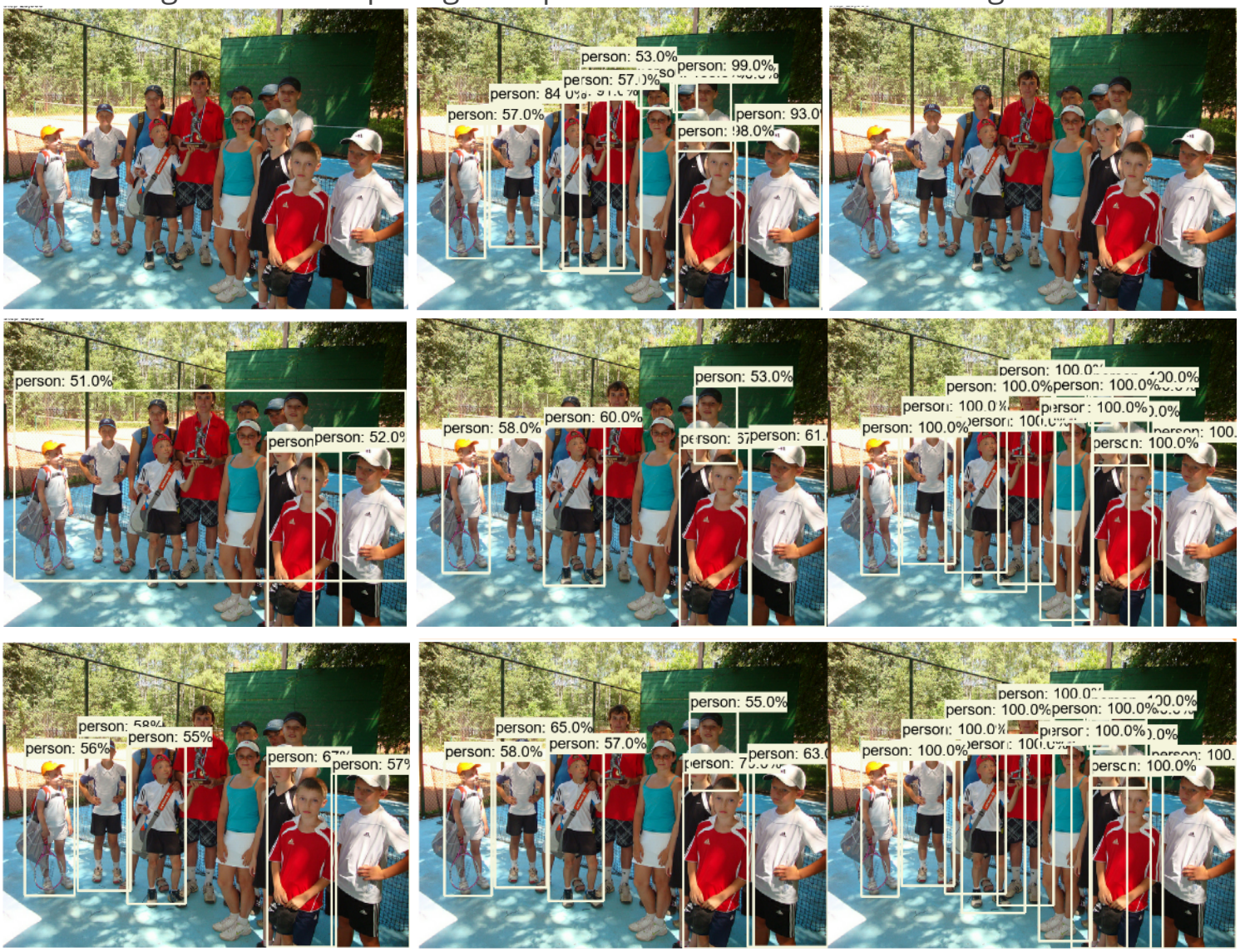


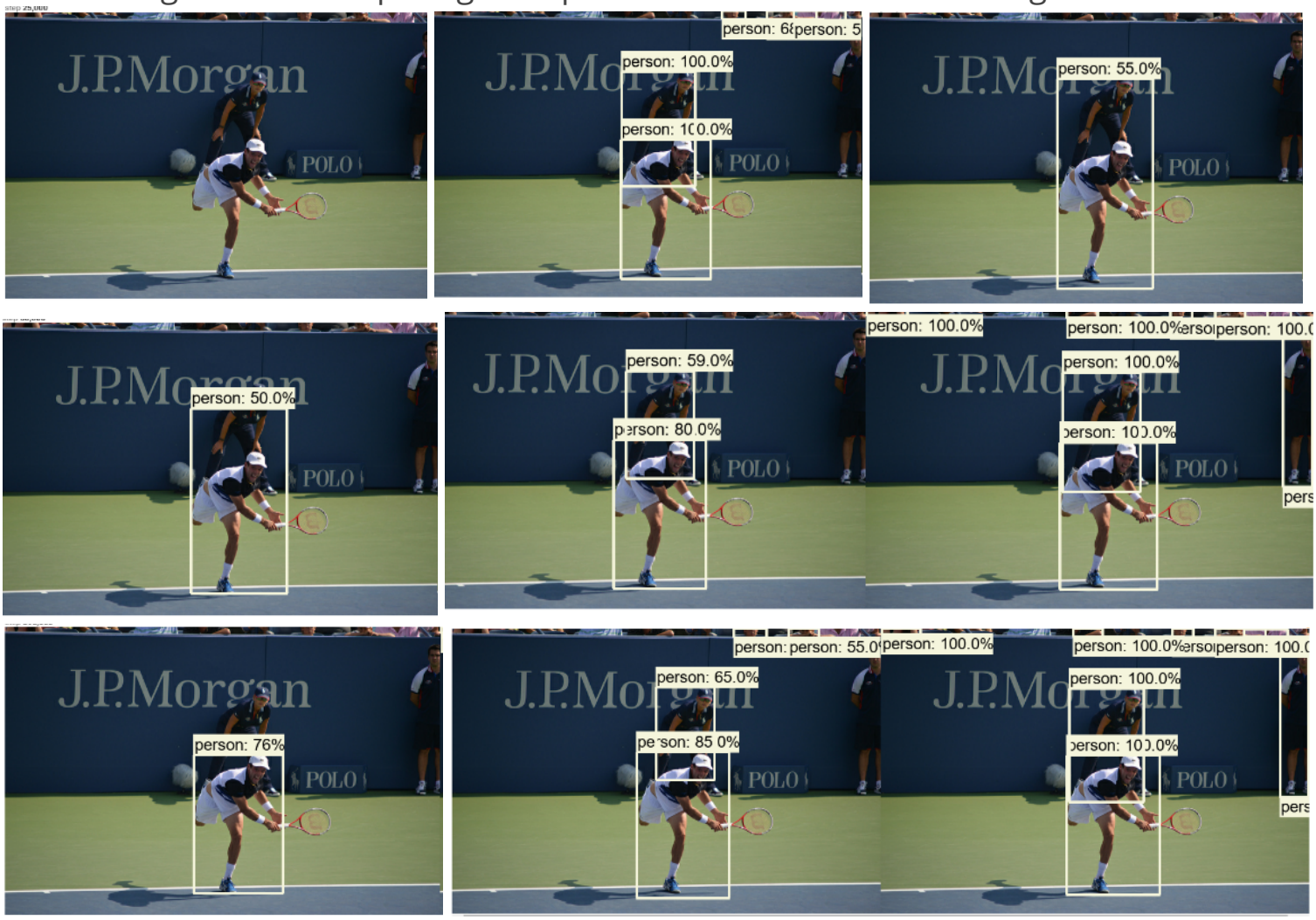
Fig 4.6.12 - Comparing CNN performance on validation images 1



4K-1Batch-25KSteps	4K-6Batch-500KSteps	70K-4Batch-25KSteps
70K-5Batch-60KSteps	70K-6Batch-150KSteps	Ground Truth
70K-6Batch-400KSteps	70K-6Batch-500KSteps	Ground Truth

Important Note: 4K-1Batch-25KSteps and 4K-6Batch-500KSteps has an unfair advantage as it has seen this image during training.

Fig 4.6.13 - Comparing CNN performance on validation images 2



4K-1Batch-25KSteps	4K-6Batch-500KSteps	70K-4Batch-25KSteps
70K-5Batch-60KSteps	70K-6Batch-150KSteps	Ground Truth
70K-6Batch-400KSteps	70K-6Batch-500KSteps	Ground Truth

Important Note: 4K-1Batch-25KSteps and 4K-6Batch-500KSteps has an unfair advantage as it has seen this image during training.

Fig 4.6.14 - Comparing CNN performance on validation images 4

4K-1Batch-25KSteps	4K-6Batch-500KSteps	70K-4Batch-25KSteps
70K-5Batch-60KSteps	70K-6Batch-150KSteps	Ground Truth
70K-6Batch-400KSteps	70K-6Batch-500KSteps	Ground Truth

Important Note: 4K-1Batch-25KSteps and 4K-6Batch-500KSteps has an unfair advantage as it has seen this image during training.

Fig 4.6.15 - image from coco evaluation data - image number 28993.jpg

Fig 4.6.16 - Commands Used

```
cd C:\Users\donov>cd
Documents\tensorflow\workspace\beforeYOLO\preprocessing\CocoDataset\Dataset_to_VOC_converter-master\
activate tensorflow

python anno_coco2voc.py --anno_file=..\annotations_trainval2017\annotations\instances_train2017.json --type=instance
--output_dir=..\trainLabels\
python anno_coco2voc.py --anno_file=..\annotations_trainval2017\annotations\instances_val2017.json --type=instance
--output_dir=..\testLabels\

moved all uneeded classes to other
cd ..\..\
python moveImages.py --xmlPath "CocoDataset/trainLabels" --imgPath "CocoDataset/train2017" --type "train"
python moveImages.py --xmlPath "CocoDataset/testLabels" --imgPath "CocoDataset/val2017" --type "test"

C:\Users\donov\Documents\tensorflow\workspace\beforeYOLO\preprocessing\images\test>labelImg

move images to C:\Users\donov\Documents\tensorflow\workspace\beforeYOLO\images

python generate_tfrecord.py -x ..\images\test -l ..\annotations\label_map.pbtxt -o ..\annotations\test.record
python generate_tfrecord.py -x ..\images\train -l ..\annotations\label_map.pbtxt -o ..\annotations\train.record

downloaded pre-trained base
configure pipeline
copy train script

used for all plotting of data
tensorboard --logdir=my_ssd_resnet50_v1_fpn

training model
done
python model_main_tf2.py --model_dir=model-3K-1Batch-25KSteps/my_ssd_resnet50_v1_fpn
--pipeline_config_path=model-3K-1Batch-25KSteps/my_ssd_resnet50_v1_fpn/pipeline.config
python model_main_tf2.py --model_dir=model-70K-4Batch-25KSteps/my_ssd_resnet50_v1_fpn
--pipeline_config_path=model-70K-4Batch-25KSteps/my_ssd_resnet50_v1_fpn/pipeline.config
python model_main_tf2.py --model_dir=model-70K-5Batch-60KSteps/my_ssd_resnet50_v1_fpn
--pipeline_config_path=model-70K-5Batch-60KSteps/my_ssd_resnet50_v1_fpn/pipeline.config
python model_main_tf2.py --model_dir=model-70K-6Batch-150KSteps/my_ssd_resnet50_v1_fpn
--pipeline_config_path=model-70K-6Batch-150KSteps/my_ssd_resnet50_v1_fpn/pipeline.config
python model_main_tf2.py --model_dir=model-70K-6Batch-400KSteps/my_ssd_resnet50_v1_fpn
--pipeline_config_path=model-70K-6Batch-400KSteps/my_ssd_resnet50_v1_fpn/pipeline.config

python model_main_tf2.py --model_dir=model-3K-6Batch-500KSteps/my_ssd_resnet50_v1_fpn
--pipeline_config_path=model-3K-6Batch-500KSteps/my_ssd_resnet50_v1_fpn/pipeline.config
python model_main_tf2.py --model_dir=model-70K-6Batch-500KSteps/my_ssd_resnet50_v1_fpn
--pipeline_config_path=model-70K-6Batch-500KSteps/my_ssd_resnet50_v1_fpn/pipeline.config


todo
python model_main_tf2.py --model_dir=model-3K-1Batch-400KSteps/my_ssd_resnet50_v1_fpn
--pipeline_config_path=model-3K-1Batch-400KSteps/my_ssd_resnet50_v1_fpn/pipeline.config
python model_main_tf2.py --model_dir=model-70K-4Batch-400KSteps/my_ssd_resnet50_v1_fpn
--pipeline_config_path=model-70K-4Batch-400KSteps/my_ssd_resnet50_v1_fpn/pipeline.config
python model_main_tf2.py --model_dir=model-70K-5Batch-400KSteps/my_ssd_resnet50_v1_fpn
--pipeline_config_path=model-70K-5Batch-400KSteps/my_ssd_resnet50_v1_fpn/pipeline.config



evaluate model
export model
python .\exporter_main_v2.py --input_type image_tensor --pipeline_config_path
.\model-3K-1Batch-25KSteps\my_ssd_resnet50_v1_fpn\pipeline.config --trained_checkpoint_dir
.\model-3K-1Batch-25KSteps\my_ssd_resnet50_v1_fpn\ --output_directory .\exported-models\model-3K-1Batch-25KSteps
```

```
python .\exporter_main_v2.py --input_type image_tensor --pipeline_config_path
.\model-3K-6Batch-500KSteps\my_ssd_resnet50_v1_fpn\pipeline.config --trained_checkpoint_dir
.\model-3K-6Batch-500KSteps\my_ssd_resnet50_v1_fpn\ --output_directory .\exported-models\model-3K-6Batch-500KSteps
python .\exporter_main_v2.py --input_type image_tensor --pipeline_config_path
.\model-70K-4Batch-25KSteps\my_ssd_resnet50_v1_fpn\pipeline.config --trained_checkpoint_dir
.\model-70K-4Batch-25KSteps\my_ssd_resnet50_v1_fpn\ --output_directory .\exported-models\model-70K-4Batch-25KSteps
python .\exporter_main_v2.py --input_type image_tensor --pipeline_config_path
.\model-70K-5Batch-60KSteps\my_ssd_resnet50_v1_fpn\pipeline.config --trained_checkpoint_dir
.\model-70K-5Batch-60KSteps\my_ssd_resnet50_v1_fpn\ --output_directory .\exported-models\model-70K-5Batch-60KSteps
python .\exporter_main_v2.py --input_type image_tensor --pipeline_config_path
.\model-70K-6Batch-150KSteps\my_ssd_resnet50_v1_fpn\pipeline.config --trained_checkpoint_dir
.\model-70K-6Batch-150KSteps\my_ssd_resnet50_v1_fpn\ --output_directory .\exported-models\model-70K-6Batch-150KSteps
python .\exporter_main_v2.py --input_type image_tensor --pipeline_config_path
.\model-70K-6Batch-400KSteps\my_ssd_resnet50_v1_fpn\pipeline.config --trained_checkpoint_dir
.\model-70K-6Batch-400KSteps\my_ssd_resnet50_v1_fpn\ --output_directory .\exported-models\model-70K-6Batch-400KSteps
python .\exporter_main_v2.py --input_type image_tensor --pipeline_config_path
.\model-70K-6Batch-500KSteps\my_ssd_resnet50_v1_fpn\pipeline.config --trained_checkpoint_dir
.\model-70K-6Batch-500KSteps\my_ssd_resnet50_v1_fpn\ --output_directory .\exported-models\model-70K-6Batch-500KSteps

cd runAI
export car video
python object_detection_video.py -m ..\exported-models\model-3K-1Batch-25KSteps -v car.mp4
python object_detection_video.py -m ..\exported-models\model-3K-6Batch-500KSteps -v car.mp4
python object_detection_video.py -m ..\exported-models\model-70K-4Batch-25KSteps -v car.mp4
python object_detection_video.py -m ..\exported-models\model-70K-5Batch-60KSteps -v car.mp4
python object_detection_video.py -m ..\exported-models\model-70K-6Batch-150KSteps -v car.mp4
python object_detection_video.py -m ..\exported-models\model-70K-6Batch-400KSteps -v car.mp4
python object_detection_video.py -m ..\exported-models\model-70K-6Batch-500KSteps -v car.mp4

export dog video
python object_detection_video.py -m ..\exported-models\model-3K-1Batch-25KSteps -v dog.mp4
python object_detection_video.py -m ..\exported-models\model-3K-6Batch-500KSteps -v dog.mp4
python object_detection_video.py -m ..\exported-models\model-70K-4Batch-25KSteps -v dog.mp4
python object_detection_video.py -m ..\exported-models\model-70K-5Batch-60KSteps -v dog.mp4
python object_detection_video.py -m ..\exported-models\model-70K-6Batch-150KSteps -v dog.mp4
python object_detection_video.py -m ..\exported-models\model-70K-6Batch-400KSteps -v dog.mp4
python object_detection_video.py -m ..\exported-models\model-70K-6Batch-500KSteps -v dog.mp4

export donovan video
python object_detection_video.py -m ..\exported-models\model-3K-1Batch-25KSteps -v donovan.mp4
python object_detection_video.py -m ..\exported-models\model-3K-6Batch-500KSteps -v donovan.mp4
python object_detection_video.py -m ..\exported-models\model-70K-4Batch-25KSteps -v donovan.mp4
python object_detection_video.py -m ..\exported-models\model-70K-5Batch-60KSteps -v donovan.mp4
python object_detection_video.py -m ..\exported-models\model-70K-6Batch-150KSteps -v donovan.mp4
python object_detection_video.py -m ..\exported-models\model-70K-6Batch-400KSteps -v donovan.mp4
python object_detection_video.py -m ..\exported-models\model-70K-6Batch-500KSteps -v donovan.mp4

used for batch converting videos
convert videos
FOR /F "tokens=*" %G IN ('dir /b *.avi') DO ffmpeg -i "%G" -codec copy "%~nG.mp4"

test realtime detection
python object_detection_camera.py -m ..\exported-models\model-70K-4Batch-25KSteps -w 0



cd to model directory
activate tensorfloweval
changed numpy to 1.17.5 to avoid error

used for eval data
```

```
python model_main_tf2.py --sample_1_of_n_eval_examples=10 --model_dir=model-3K-1Batch-25KSteps/my_ssd_resnet50_v1_fpn
--pipeline_config_path=model-3K-1Batch-25KSteps/my_ssd_resnet50_v1_fpn/pipeline.config
--checkpoint_dir=model-3K-1Batch-25KSteps/my_ssd_resnet50_v1_fpn
python model_main_tf2.py --sample_1_of_n_eval_examples=10 --model_dir=model-3K-6Batch-500KSteps/my_ssd_resnet50_v1_fpn
--pipeline_config_path=model-3K-6Batch-500KSteps/my_ssd_resnet50_v1_fpn/pipeline.config
--checkpoint_dir=model-3K-6Batch-500KSteps/my_ssd_resnet50_v1_fpn
python model_main_tf2.py --sample_1_of_n_eval_examples=1 --model_dir=model-70K-4Batch-25KSteps/my_ssd_resnet50_v1_fpn
--pipeline_config_path=model-70K-4Batch-25KSteps/my_ssd_resnet50_v1_fpn/pipeline.config
--checkpoint_dir=model-70K-4Batch-25KSteps/my_ssd_resnet50_v1_fpn
python model_main_tf2.py --sample_1_of_n_eval_examples=1 --model_dir=model-70K-5Batch-60KSteps/my_ssd_resnet50_v1_fpn
--pipeline_config_path=model-70K-5Batch-60KSteps/my_ssd_resnet50_v1_fpn/pipeline.config
--checkpoint_dir=model-70K-5Batch-60KSteps/my_ssd_resnet50_v1_fpn
python model_main_tf2.py --sample_1_of_n_eval_examples=1 --model_dir=model-70K-6Batch-150KSteps/my_ssd_resnet50_v1_fpn
--pipeline_config_path=model-70K-6Batch-150KSteps/my_ssd_resnet50_v1_fpn/pipeline.config
--checkpoint_dir=model-70K-6Batch-150KSteps/my_ssd_resnet50_v1_fpn
python model_main_tf2.py --sample_1_of_n_eval_examples=1 --model_dir=model-70K-6Batch-400KSteps/my_ssd_resnet50_v1_fpn
--pipeline_config_path=model-70K-6Batch-400KSteps/my_ssd_resnet50_v1_fpn/pipeline.config
--checkpoint_dir=model-70K-6Batch-400KSteps/my_ssd_resnet50_v1_fpn
python model_main_tf2.py --sample_1_of_n_eval_examples=1 --model_dir=model-70K-6Batch-500KSteps/my_ssd_resnet50_v1_fpn
--pipeline_config_path=model-70K-6Batch-500KSteps/my_ssd_resnet50_v1_fpn/pipeline.config
--checkpoint_dir=model-70K-6Batch-500KSteps/my_ssd_resnet50_v1_fpn
```

used for training data

```
python model_main_tf2.py --sample_1_of_n_eval_examples=1 --model_dir=model-3K-1Batch-25KSteps/my_ssd_resnet50_v1_fpn
--pipeline_config_path=model-3K-1Batch-25KSteps/my_ssd_resnet50_v1_fpn/pipeline_evalTraining.config
--checkpoint_dir=model-3K-1Batch-25KSteps/my_ssd_resnet50_v1_fpn
python model_main_tf2.py --sample_1_of_n_eval_examples=1 --model_dir=model-3K-6Batch-500KSteps/my_ssd_resnet50_v1_fpn
--pipeline_config_path=model-3K-6Batch-500KSteps/my_ssd_resnet50_v1_fpn/pipeline_evalTraining.config
--checkpoint_dir=model-3K-6Batch-500KSteps/my_ssd_resnet50_v1_fpn
python model_main_tf2.py --sample_1_of_n_eval_examples=10 --model_dir=model-70K-4Batch-25KSteps/my_ssd_resnet50_v1_fpn
--pipeline_config_path=model-70K-4Batch-25KSteps/my_ssd_resnet50_v1_fpn/pipeline_evalTraining.config
--checkpoint_dir=model-70K-4Batch-25KSteps/my_ssd_resnet50_v1_fpn
python model_main_tf2.py --sample_1_of_n_eval_examples=10 --model_dir=model-70K-5Batch-60KSteps/my_ssd_resnet50_v1_fpn
--pipeline_config_path=model-70K-5Batch-60KSteps/my_ssd_resnet50_v1_fpn/pipeline_evalTraining.config
--checkpoint_dir=model-70K-5Batch-60KSteps/my_ssd_resnet50_v1_fpn
python model_main_tf2.py --sample_1_of_n_eval_examples=10
--model_dir=model-70K-6Batch-150KSteps/my_ssd_resnet50_v1_fpn
--pipeline_config_path=model-70K-6Batch-150KSteps/my_ssd_resnet50_v1_fpn/pipeline_evalTraining.config
--checkpoint_dir=model-70K-6Batch-150KSteps/my_ssd_resnet50_v1_fpn
python model_main_tf2.py --sample_1_of_n_eval_examples=10
--model_dir=model-70K-6Batch-400KSteps/my_ssd_resnet50_v1_fpn
--pipeline_config_path=model-70K-6Batch-400KSteps/my_ssd_resnet50_v1_fpn/pipeline_evalTraining.config
--checkpoint_dir=model-70K-6Batch-400KSteps/my_ssd_resnet50_v1_fpn
python model_main_tf2.py --sample_1_of_n_eval_examples=10
--model_dir=model-70K-6Batch-500KSteps/my_ssd_resnet50_v1_fpn
--pipeline_config_path=model-70K-6Batch-500KSteps/my_ssd_resnet50_v1_fpn/pipeline_evalTraining.config
--checkpoint_dir=model-70K-6Batch-500KSteps/my_ssd_resnet50_v1_fpn
```

attempt confusion matrix

```
python infer_detections.py --input_tfrecord_paths=annotations/test.record
--output_tfrecord_path=exported-models/model-70K-6Batch-400KSteps/saved_model/test_detections.tfrecord
--inference_graph=exported-models/model-70K-6Batch-400KSteps/saved_model/saved_model.pb
```

---

7. References

[1] Label Your Data, “The History of Machine Learning: How Did It All Start?,” Jul. 06, 2020. https://labelyourdata.com/articles/history-of-machine-learning-how-did-it-all-start/ (accessed Apr. 29, 2021).

[2] Y. Taigman, M. Yang, M. ’aurelio Ranzato, and L. Wolf, “DeepFace: Closing the Gap to Human-Level Performance in Face Verification,” *2014 IEEE Conference on Computer Vision and Pattern Recognition*. 2014, doi: 10.1109/cvpr.2014.220.

[3] Expert. ai Team, “What is Machine Learning? A definition,” May 05, 2020. https://www.expert.ai/blog/machine-learning-definition/ (accessed May 10, 2021).

[4] “How Does Machine Learning Work? - dummies,” Jul. 11, 2019. https://www.dummies.com/programming/big-data/data-science/how-does-machine-learning-work/ (accessed May 10, 2021).

[5] “Machine Learning,” Feb. 25, 2021. https://www.sas.com/en_ie/insights/analytics/machine-learning.html (accessed May 10, 2021).

[6] J. Brownlee, “Supervised and Unsupervised Machine Learning Algorithms,” Mar. 15, 2016. https://machinelearningmastery.com/supervised-and-unsupervised-machine-learning-algorithms/ (accessed May 07, 2021).

[7] IBM Cloud Education, “What is Supervised Learning?” https://www.ibm.com/cloud/learn/supervised-learning (accessed May 08, 2021).

[8] A. Wilson, “A Brief Introduction to Supervised Learning - Towards Data Science,” *Towards Data Science*, Sep. 29, 2019. https://towardsdatascience.com/a-brief-introduction-to-supervised-learning-54a3e3932590 (accessed May 08, 2021).

[9] K. Rungta, “Unsupervised Machine Learning: What is, Algorithms, Example,” Jan. 01, 2020. https://www.guru99.com/unsupervised-machine-learning.html (accessed May 08, 2021).

[10] “Unsupervised Machine Learning.” https://www.datarobot.com/wiki/unsupervised-machine-learning/ (accessed May 08, 2021).

[11] IBM Cloud Education, “What is Unsupervised Learning?” https://www.ibm.com/cloud/learn/unsupervised-learning (accessed May 08, 2021).

[12] “Unsupervised Machine learning - Javatpoint.” https://www.javatpoint.com/unsupervised-machine-learning (accessed May 08, 2021).

[13] B. Dickson, “What is semi-supervised machine learning?,” Jan. 04, 2021. https://bdtechtalks.com/2021/01/04/semi-supervised-machine-learning/ (accessed May 08, 2021).

[14] A. Ye, “Supervised Learning, But A Lot Better: Semi-Supervised Learning,” *Towards Data Science*, Jun. 06, 2020. https://towardsdatascience.com/supervised-learning-but-a-lot-better-semi-supervised-learning-a42dff534781 (accessed May 08, 2021).

[15] J. Brownlee, “What Is Semi-Supervised Learning,” Apr. 08, 2021. https://machinelearningmastery.com/what-is-semi-supervised-learning/ (accessed May 08, 2021).

[16] “10 Companies Using Machine Learning in Cool Ways.” https://www.wordstream.com/blog/ws/2017/07/28/machine-learning-applications (accessed May 07, 2021).

[17] N. Duggal, “Top 10 Machine Learning Applications in 2021,” Apr. 22, 2020. https://www.simplilearn.com/tutorials/machine-learning-tutorial/machine-learning-applications (accessed May 07, 2021).

[18] “The Amazing Ways Tesla Is Using Artificial Intelligence And Big Data.” https://bernardmarr.com/default.asp?contentID=1251 (accessed May 07, 2021).

[19] “Autopilot AI.” https://www.tesla.com/autopilotAI (accessed May 07, 2021).

[20] “Applications of Machine Learning - Javatpoint.” https://www.javatpoint.com/applications-of-machine-learning (accessed May 07, 2021).

[21] K. Keshari, “Top 10 Applications of Machine Learning,” Jan. 28, 2019. https://www.edureka.co/blog/machine-learning-applications/ (accessed May 07, 2021).

[22] Marek, “Neural Networks, Part 1: Background,” Jan. 19, 2014. http://www.marekrei.com/blog/neural-networks-part-1-background/ (accessed Apr. 30, 2021).

[23] “What is a neuron?,” Nov. 22, 2016. https://qbi.uq.edu.au/brain/brain-anatomy/what-neuron (accessed May 08, 2021).

[24] “Artificial Intelligence - Neural Networks.” https://www.tutorialspoint.com/artificial_intelligence/artificial_intelligence_neural_networks.htm (accessed May 08, 2021).

[25] “Feedforward Neural Networks.” https://brilliant.org/wiki/feedforward-neural-networks/ (accessed May 08, 2021).

[26] R. Miikkulainen, “Topology of a Neural Network,” in *Encyclopedia of Machine Learning*, Springer, Boston, MA, 2011, pp. 988–989.

[27] G. Naitzat, A. Zhitnikov, and L.-H. Lim, “Topology of deep neural networks,” Apr. 13, 2020.

[28] DeepAI, “Feed Forward Neural Network,” May 17, 2019. https://deepai.org/machine-learning-glossary-and-terms/feed-forward-neural-network (accessed May 08, 2021).

[29] S. Bhattacharyya, “Neural networks: evolution, topologies, learning algorithms and applications,” *Cross-Disciplinary Applications of Artificial Intelligence and Pattern Recognition: Advancing Technologies*, pp. 450–498, Jan. 2011, Accessed: May 08, 2021. [Online].

[30] S. M. Shamsuddin, A. O. Ibrahim, and C. Ramadhena, “Artificial Neural Networks - Architectures and Applications,” in *Artificial Neural Networks - Architectures and Applications*, IntechOpen, 2013.

[31] “Neural Networks, Manifolds, and Topology.” https://colah.github.io/posts/2014-03-NN-Manifolds-Topology/ (accessed May 08, 2021).
[32] “[No title].” https://www.ling.upenn.edu/courses/cogs501/Rosenblatt1958.pdf (accessed May 10, 2021).
[33] A. Bhardwaj, “What is a Perceptron? – Basics of Neural Networks - Towards Data Science,” *Towards Data Science*, Oct. 11, 2020. https://towardsdatascience.com/what-is-a-perceptron-basics-of-neural-networks-c4cfea20c590 (accessed May 10, 2021).
[34] “Website.” https://www.simplilearn.com/what-is-perceptron-tutoria (accessed May 10, 2021).
[35] K. Rungta, “Back Propagation Neural Network: Explained With Simple Example,” Jan. 01, 2020. https://www.guru99.com/backpropogation-neural-network.html (accessed May 10, 2021).
[36] Uniqtech, “Multilayer Perceptron (MLP) vs Convolutional Neural Network in Deep Learning,” *Data Science Bootcamp*, Dec. 22, 2018. https://medium.com/data-science-bootcamp/multilayer-perceptron-mlp-vs-convolutional-neural-network-in-deep-learning-c890f487a8f1 (accessed May 10, 2021).
[37] “Multilayer perceptron (MLP) - Introduction to neural networks.” https://www.coursera.org/lecture/intro-to-deep-learning/multilayer-perceptron-mlp-yy1NV (accessed May 10, 2021).
[38] A. Gad, “A Comprehensive Guide to the Backpropagation Algorithm in Neural Networks,” Mar. 27, 2021. https://neptune.ai/blog/backpropagation-algorithm-in-neural-networks-guide (accessed May 10, 2021).
[39] “A Beginner’s Guide to Backpropagation in Neural Networks.” http://wiki.pathmind.com/backpropagation (accessed May 10, 2021).
[40] “Designing Your Neural Networks - KDnuggets.” https://www.kdnuggets.com/designing-your-neural-networks.html/ (accessed May 10, 2021).
[41] J. Brownlee, “Gradient Descent For Machine Learning,” Mar. 22, 2016. https://machinelearningmastery.com/gradient-descent-for-machine-learning/ (accessed May 10, 2021).
[42] “Gradient Descent: An Introduction to 1 of Machine Learning’s Most Popular Algorithms.” https://builtin.com/data-science/gradient-descent (accessed May 10, 2021).
[43] “Gradient Descent — ML Glossary documentation.” https://ml-cheatsheet.readthedocs.io/en/latest/gradient_descent.html (accessed May 10, 2021).
[44] “Activation functions in Neural Networks - GeeksforGeeks,” Jan. 29, 2018. https://www.geeksforgeeks.org/activation-functions-neural-networks/ (accessed May 10, 2021).
[45] “Activation functions in Neural Networks - GeeksforGeeks,” Jan. 29, 2018. https://www.geeksforgeeks.org/activation-functions-neural-networks/ (accessed May 10, 2021).
[46] “Activation functions in Neural Networks - GeeksforGeeks,” Jan. 29, 2018. https://www.geeksforgeeks.org/activation-functions-neural-networks/ (accessed May 10, 2021).
[47] “Difference between linear and nonlinear neural networks?” https://www.kaggle.com/discussion/196486 (accessed May 10, 2021).
[48] S. Sharma, “Activation Functions in Neural Networks - Towards Data Science,” *Towards Data Science*, Sep. 06, 2017. https://towardsdatascience.com/activation-functions-neural-networks-1cbd9f8d91d6 (accessed May 10, 2021).
[49] S. Saha, “A Comprehensive Guide to Convolutional Neural Networks — the ELI5 way,” *Towards Data Science*, Dec. 15, 2018. https://towardsdatascience.com/a-comprehensive-guide-to-convolutional-neural-networks-the-eli5-way-3bd2b1164a53 (accessed May 10, 2021).
[50] S. Asiri, “Building a Convolutional Neural Network for Image Classification with Tensorflow,” *Medium*, Nov. 12, 2019. https://medium.com/@sidathasiri/building-a-convolutional-neural-network-for-image-classification-with-tensorflow-f1f2f56bd83b (accessed May 10, 2021).
[51] *TensorFlow 2.0 Complete Course - Python Neural Networks for Beginners Tutorial*. 2020.
[52] “#006 CNN Convolution On RGB Images,” Nov. 06, 2018. http://datahacker.rs/convolution-rgb-image/ (accessed May 10, 2021).
[53] C. T. B. Miap, “An introduction to Convolutional Neural Networks - Towards Data Science,” *Towards Data Science*, May 27, 2019. https://towardsdatascience.com/an-introduction-to-convolutional-neural-networks-eb0b60b58fd7 (accessed May 10, 2021).
[54] M. Z. Alom *et al.*, “A State-of-the-Art Survey on Deep Learning Theory and Architectures,” *Electronics*, vol. 8, no. 3. p. 292, 2019, doi: 10.3390/electronics8030292.
[55] “An intuitive guide to Convolutional Neural Networks,” Apr. 24, 2018. https://www.freecodecamp.org/news/an-intuitive-guide-to-convolutional-neural-networks-260c2de0a050/ (accessed May 10, 2021).
[56] B. Dickson, “What are convolutional neural networks (CNN)?,” Jan. 06, 2020. https://bdtechtalks.com/2020/01/06/convolutional-neural-networks-cnn-convnets/ (accessed May 10, 2021).
[57] K. He, X. Zhang, S. Ren, and J. Sun, “Deep Residual Learning for Image Recognition,” *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 2016, doi: 10.1109/cvpr.2016.90.
[58] C.-F. Wang, “The Vanishing Gradient Problem,” *Towards Data Science*, Jan. 08, 2019. https://towardsdatascience.com/the-vanishing-gradient-problem-69bf08b15484 (accessed May 07, 2021).
[59] P. Ruiz, “Understanding and visualizing ResNets - Towards Data Science,” *Towards Data Science*, Oct. 08, 2018. https://towardsdatascience.com/understanding-and-visualizing-resnets-442284831be8 (accessed May 07, 2021).
[60] “A Beginner’s Guide to Object Detection,” Apr. 19, 2018. https://www.datacamp.com/community/tutorials/object-detection-guide (accessed May 06, 2021).

[61] “How single-shot detector (SSD) works?” https://developers.arcgis.com/python/guide/how-ssd-works/ (accessed May 06, 2021).
[62] “Lecture 11 | Detection and Segmentation,” Aug. 11, 2017. https://www.youtube.com/watch?v=nDPWywWRIRo (accessed May 06, 2021).
[63] W. Liu *et al.*, “SSD: Single Shot MultiBox Detector,” *Computer Vision – ECCV 2016*. pp. 21–37, 2016, doi: 10.1007/978-3-319-46448-0_2.
[64] M. R. Ronchi, “Describing Common Human Visual Actions in Images.” http://people.vision.caltech.edu/~mronchi/projects/Cocoa/ (accessed May 06, 2021).
[65] T.-Y. Lin *et al.*, “Microsoft COCO: Common Objects in Context,” May 01, 2014.
[66] CasiaFan, “CasiaFan/Dataset_to_VOC_converter.” https://github.com/CasiaFan/Dataset_to_VOC_converter (accessed Apr. 27, 2021).
[67] “Training Custom Object Detector — TensorFlow 2 Object Detection API tutorial documentation.” https://tensorflow-object-detection-api-tutorial.readthedocs.io/en/latest/training.html#convert-xml-to-record (accessed Apr. 27, 2021).
[68] “GeForce GTX 1070.” https://www.nvidia.com/en-gb/geforce/graphics-cards/geforce-gtx-1070/specifications/ (accessed Apr. 29, 2021).
[69] “Performance and Scalability of GPU-Based Convolutional Neural Networks.” https://ieeexplore.ieee.org/document/5452452 (accessed May 04, 2021).
[70] tensorflow, “tensorflow/models.” https://github.com/tensorflow/models (accessed Apr. 28, 2021).
[71] T. C. Arlen, “Understanding the mAP Evaluation Metric for Object Detection,” *Medium*, Mar. 01, 2018. https://medium.com/@timothycarlen/understanding-the-map-evaluation-metric-for-object-detection-a07fe6962cf3 (accessed Apr. 28, 2021).
[72] S. Yohanandan, “mAP (mean Average Precision) might confuse you!,” *Towards Data Science*, Jun. 09, 2020. https://towardsdatascience.com/map-mean-average-precision-might-confuse-you-5956f1bfa9e2 (accessed Apr. 28, 2021).
[73] “FFmpeg.” https://ffmpeg.org/ (accessed Apr. 28, 2021).
[74] “Detect Objects Using Your Webcam — TensorFlow 2 Object Detection API tutorial documentation.” https://tensorflow-object-detection-api-tutorial.readthedocs.io/en/latest/auto_examples/object_detection_camera.html#sphx-glr-auto-examples-object-detection-camera-py (accessed Apr. 29, 2021).
[75] G. F. Elsayed, I. Goodfellow, and J. Sohl-Dickstein, “Adversarial Reprogramming of Neural Networks,” Jun. 28, 2018.
[76] “Underfitting and Overfitting in Machine Learning - GeeksforGeeks,” Nov. 23, 2017. https://www.geeksforgeeks.org/underfitting-and-overfitting-in-machine-learning/ (accessed May 04, 2021).
[77] J. Jordan, “Normalizing your data (specifically, input and batch normalization),” Jan. 27, 2018. https://www.jeremyjordan.me/batch-normalization/ (accessed Apr. 30, 2021).
[78] X. Ma, H. Huang, Y. Wang, S. Romano, S. Erfani, and J. Bailey, “Normalized Loss Functions for Deep Learning with Noisy Labels,” Jun. 24, 2020.
[79] “COCO - Common Objects in Context.” https://cocodataset.org/#detection-leaderboard (accessed May 10, 2021).
[80] “COCO - Common Objects in Context.” https://cocodataset.org/#detection-leaderboard (accessed May 10, 2021).
[81] N. Bodla, B. Singh, R. Chellappa, and L. S. Davis, “Soft-NMS — Improving Object Detection with One Line of Code,” *Proc. IEEE Int. Conf. Comput. Vis.*, pp. 5562–5570, Oct. 2017, Accessed: May 10, 2021. [Online].