RECOGNITION OF OBJECTS IN REAL TIME VIDEOS USING MACHINE LEARNING

by

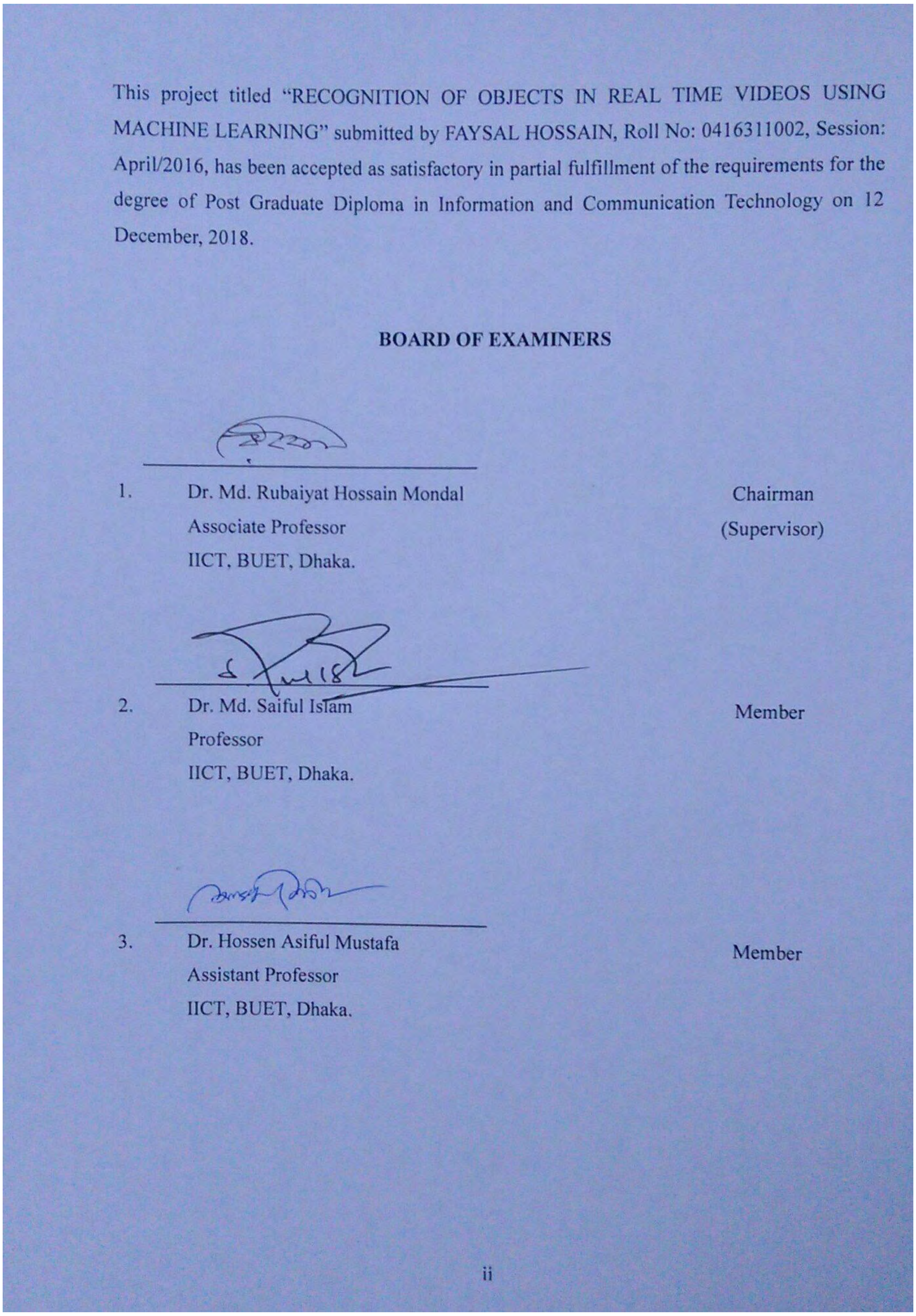
FAYSAL HOSSAIN

POST GRADUATE DIPLOMA IN INFORMATION AND COMMUNICATION TECHNOLOGY

Institute of Information and Communication Technology (IICT)

BANGLADESH UNIVERSITY OF ENGINEERING AND TECHNOLOGY (BUET)

12 December, 2018



This project titled "RECOGNITION OF OBJECTS IN REAL TIME VIDEOS USING MACHINE LEARNING" submitted by FAYSAL HOSSAIN, Roll No: 0416311002, Session: April/2016, has been accepted as satisfactory in partial fulfillment of the requirements for the degree of Post Graduate Diploma in Information and Communication Technology on 12 December, 2018.

BOARD OF EXAMINERS

[illegible]

1. Dr. Md. Rubaiyat Hossain Mondal — Chairman (Supervisor)
   Associate Professor
   IICT, BUET, Dhaka.

[illegible]

2. Dr. Md. Saiful Islam — Member
   Professor
   IICT, BUET, Dhaka.

[illegible]

3. Dr. Hossen Asiful Mustafa — Member
   Assistant Professor
   IICT, BUET, Dhaka.

ii

CANDIDATE'S DECLARATION

It is hereby declared that this report or any part of it has not been submitted elsewhere for the award of any degree or diploma.

Faysal Hossain

FAYSAL HOSSAIN

ID: 0416311002

DEDICATION

To

My Parents and Family Members

Table of Contents

List of Figures

List of Abbreviations and Key Terms

CNN	Convolutional Neural Network
CPU	Central Processing Unit
GPU	Graphics Processing Unit
FC	Fully Connected
ReLU	Rectified Linear Unit
RCNN	Regional Convolutional Neural Network
F-RCNN	Faster Regional Convolutional Neural Network
SSD	Single Shot Detector
OpenCV	Open Source Computer Vision
IoU	Intersection over Union
NMS	Non-maximum Suppression
RoI	Region of Interest

Acknowledgment

First of all, I would like to convey my gratitude to Almighty Allah for giving me the opportunity to accomplish this project. I want to thank my supervisor Dr. Md. Rubaiyat Hossain Mondal, Associate Professor, IICT, BUET for giving me the chance to explore such an interesting field of research and providing help and advice whenever I needed it. Without his proper guidance, advice, continual encouragement and active involvement in this process of this work, it would have not been feasible.

A big thank also goes to all the teachers, officers and staffs of Information and Communication Technology (IICT) for giving me their kind support and information during the study.

Finally, I am very grateful to my parents and family members whose continuous support all over my life has brought me this far in my career.

Abstract

Deep Convolutional Neural Network (CNN) has recently made ground-breaking advances on several vision tasks such as objects detection and recognition, classification and semantic segmentation of images. It has achieved state-of-the-art performance on several image recognition benchmarks. The goal of this project is to develop a system capable of detecting and recognizing objects in real time video without substantial memory requirements using Deep CNN. Different deep learning-based methodologies have been proposed to achieve this, and a thorough study of them is undertaken here. A common paradigm to address the problem is to train object detector models with image data sets and apply these detectors in an exhaustive manner across all locations and scales. In this work, saliency-inspired CNN models are used for recognition which predict a set of class-agnostic bounding boxes along with a single score for each box, corresponding to its likelihood of containing object of interest. Python as a programming language, TensorFlow library for computing and OpenCV for computer vision, are used to complete the project. Region-based object detector model such as Faster Convolution Neural Network (Faster-RCNN) inception v2 and MobileNet Single Shot MultiBox Detector (SSD) are used to localize and recognize the objects and compare the accuracy of those predefined models to get the clear concept of model performance in using for different aspects and situations.

CHAPTER 1

Introduction

1.1 Overview

Object detection is the process of automatically locating and identifying objects contained on images. This is one of the challenging and exciting tasks in computer vision. It is difficult to detect the same object on different platforms because of variations in orientation, lighting, background and occlusion. Now, with the advance of deep learning and neural network, convolutional neural networks are currently the state-of-the-art solution for object detection. It can tackle such problems without coming up with various heuristics in real-time.

1.2 Motivation

With the rapid advancement in technology and vast amount of image data in the world, computer vision is playing a key role in revolutionizing the industrial environment. According to the InfoTrends, still cameras and mobile devices captured more than 1.2 trillion images in 2017 [33]. With this same estimate, in 2020 the figure will increase to 1.4 trillion. Going beyond consumer devices, there are cameras all over the world that capture images and record videos for automation purposes for self-driving cars, monitoring pedestrians, and traffic signals on the road and experiment in medical science. Robots need to understand a visual scene in order to smartly build devices in sort waste for engineers, doctors and space explorers alike. To effectively manage all these tasks, it is required to have some idea about its contents. Automated processing of visual contents is useful for a wide variety of image-related tasks. A dynamic environment is required to accomplish the all kinds of computer vision and machine learning processes.

1.3 Objective

The goal of this project is to develop a system capable of detecting and recognizing objects in real time video using machine learning and computer vision techniques. To achieve the goal the following steps will be carried out:

- To train a Convolutional Neural Network (CNN) based models with custom data sets.
- To capture real time video frames of objects using cameras.
- To classify and identify the object from video stream with the help of models and trained data set.
- To study and analysed the model performance, accuracy and computation time during training and testing.

1.4 Structure of Project Report

The project report begins with two theoretical chapters. Since convolutional object detection is a combination of several fields of computer science, it is required to discuss several theoretical topics to understand the basic concept behind this project. In the beginning of Chapter 2, a short introduction to machine learning and neural networks is provided. This chapter ends by introducing convolutional neural networks and computer vision. In Chapter 3, a discussion is provided on how convolutional networks can be used for object detection. Moreover, a review on the relevant literature and methods are also provided in Chapter 3. In Chapter 4, system configuration and project environment are discussed. All kinds of experimental setup are used for testing object detection model with required software and hardware. In Chapter 5, the methodology of the project is discussed step by step. In this chapter, every step required to complete the whole project is described in detail. In Chapter 6, the model evaluation is completed by discussing the accuracy and limitations. In Chapter 7, a review of the project and some concluding remarks is provided and potential improvements for future works are mentioned. Finally, necessary code and common errors are included in appendix.

CHAPTER 2

Research Background

2.1 Overview

In this chapter, the necessary theoretical knowledge for understanding the methods is discussed. First, a relevant discussion about machine learning, neural networks, and computer vision is provided in detail. Finally, how these disciplines are combined in convolutional neural networks is explained.

2.2 Machine Learning

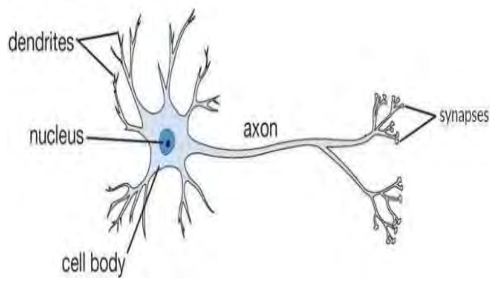
Machine learning is the science of getting computers to act using statistical techniques to give computer systems the ability to learn with data without being explicitly programmed [2]. Machine learning is closely related to computational statistics, which also focuses on prediction making through the use of computers. It has strong ties to mathematical optimization, which delivers methods, theory and application domains to the field. Machine learning algorithms are often categorized as supervised or unsupervised [32].

Supervised algorithms require machine learning skills to provide both input and desired output, in addition to furnishing feedback about the accuracy of predictions during algorithm training. Data scientists determine which variables or features the model should analyze and use to develop predictions. The algorithm is applied based on training data to learn new data [8][10].

Unsupervised algorithms do not need to be trained with desired outcome data. Instead of this, an iterative approach called deep learning is used to review data and arrive at conclusions. Unsupervised learning algorithms are used for more complex processing tasks than supervised learning systems, including image recognition, speech-to-text and natural language processing. These algorithms work by combing through millions of examples of training data and automatically identifying often subtle correlations between many variables. These algorithms have only become feasible in the age of big data, as it is required massive amounts of training data [8][32].

2.3 Neural Network

An Artificial Neural Network (ANN) is an information processing paradigm that is inspired by the way of biological nervous systems, such as the brain, process information [32]. The key element of this paradigm is the novel structure of the information processing system. It is composed of a large number of highly interconnected processing elements (neurones) working in unison to solve specific problems. An ANN is configured for a specific application, such as pattern recognition or data classification through a learning process.



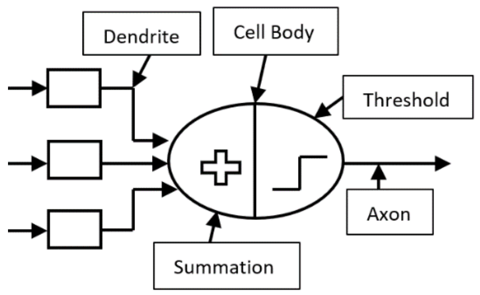


Figure 2.1: Human Neuron [36]

Figure 2.2: Artificial Neuron Concept [36]

In the human brain, a typical neuron collects signals from others through a host of fine structures called dendrites as shown in Figure 2.1. The neuron sends out spikes of electrical activity through a long, thin stand known as an axon, which splits into thousands of branches. At the end of each branch, a structure called a synapse converts the activity from the axon into electrical effects that inhibit or excite activity from the axon into electrical effects that inhibit or excite activity in the connected neurones [36]. When a neuron receives excitatory input that is sufficiently large compared with its inhibitory input, it sends a spike of electrical activity down its axon. Learning occurs by changing the effectiveness of the synapses so that the influence of one neuron changes the another [8]. An artificial neuron concept that is derived from biological neuron is shown in Figure 2.2.

2.3.1 Historical Background

Neural network simulations appear to be a recent development. However, this field was established before the advent of computers, and has survived at least one major setback and several eras. Following an initial period of enthusiasm, the field survived a period of frustration and disrepute. During this period, when funding and professional support was minimal, important advances were made by relatively few researchers. These pioneers were able to develop convincing technology which surpassed the limitations identified by Minsky and Papert [10]. They summed up a general feeling of frustration against neural networks among researchers, and thus accepted by most without further analysis. Currently, the neural network field enjoys a resurgence of interest and a corresponding increase in funding. The first artificial neuron was produced in 1943 by the neurophysiologist Warren McCulloch and the logician Walter Pits. But, the technology available at that time did not allow them to do too much.

2.3.2 Architecture of Neural Networks

A neural network is a combination of artificial neurons which are typically grouped into layers. In a fully-connected feed-forward multi-layer network, shown in Figure 2.3, each output of a layer of neurons is fed as input to each neuron of the next layer. Thus, some layers process the

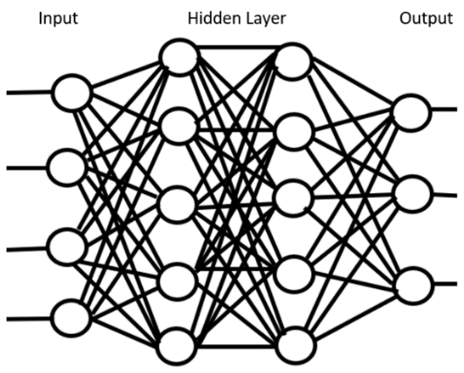


Figure 2.3: Architecture of multilayer Neural Network [8].

original input data, while some process data received from other neurons. Each neuron has a number of weights equal to the number of neurons in the previous layer. A multi-layer network typically includes an input layer, one or more hidden layers and an output layer. The input layer usually merely passes data along without modifying it. Most of the computation happens in the hidden layers. The output layer converts the hidden layer activations to an output, such as a classification [36]. A multilayer feed-forward network with at least one hidden layer can function as a universal approximator, i.e., can be constructed to compute almost any function.

2.3.3 Backpropagation

In order to train a neural network to perform a task, the weights of each unit should be adjusted in such a way that the error between the desired output and the actual output is reduced. A neural network is trained by selecting the weights of all neurons so that the network learns to approximate target outputs from known inputs. It is difficult to solve the neuron weights of a multi-layer network analytically. The back-propagation algorithm provides a simple and effective solution to solving the weights iteratively. The classical version uses gradient descent as optimization method. Gradient descent can be quite time-consuming and is not guaranteed to find the global minimum of error, but with proper configuration (known in machine learning as hyperparameters) works well enough in practice [36][10]. In the first phase of the algorithm, an input vector is propagated forward through the neural network. Before this, the weights of the network neurons have been initialized to some values, for example small random values. The received output of the network is compared to the desired output (which should be known for the training examples) using a loss function. The gradient of the loss function is then computed. This gradient is also called the error value [10]. When using mean squared error as the loss function, the output layer error value is simply the difference between the current and desired output. The error values are then propagated back through the network to calculate the error values of the hidden layer neurons. The hidden neuron loss function gradients can be solved using the chain rule of derivatives. Finally, the neuron weights are updated by calculating the gradient of the weights and subtracting a proportion of the gradient from the weights [8]. This ratio is called the learning rate. The learning rate can be fixed or dynamic. After the weights have been updated, the algorithm continues by executing the phases again with different input until the weights converge.

2.3.4 Activation Function

Activation function is an extremely important feature of the artificial neural networks. It basically decides whether a neuron should be activated or not, whether the information that the neuron is receiving is relevant for the given information or should be ignored. Different types of activation function are available for neural network such as: Identity, Binary Step, Sigmoid, Tanh, ReLU, Leaky ReLU, SoftMax [8][10], etc. ReLU stands for Rectified Linear Units, and is the most used activation function in the world right now. It is used in almost all the convolutional neural networks or deep learning. As shown in Figure 2.4, ReLU function is nonlinear, which means that it can easily backpropagate the errors and multiples layers of neurons being activated by the ReLU function [10]. The main advantage of using the ReLU function over other activation functions is that it does not activate all the neurons at the same time. That means, if the input is negative, it will convert it to zero and the neuron does not get activated.

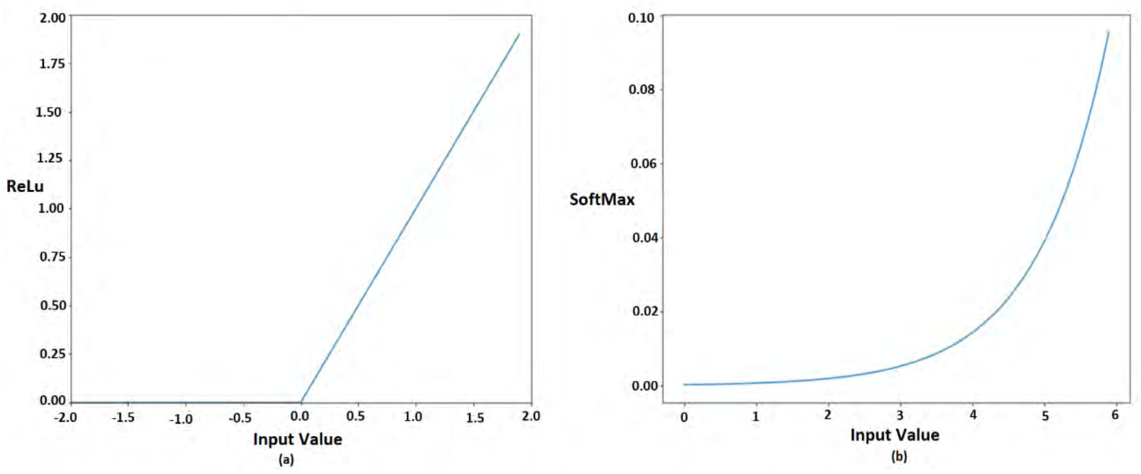


Figure 2.4: Activation function (a) ReLU (b) SoftMax

The SoftMax function, shown in Figure 2.4(b), is a type of sigmoid function but is handy when working with classification problems. The SoftMax function would squeeze the outputs for each class between 0 and 1 and would also divide by the sum of the outputs [2]. This essentially gives the probability of the input being in a particular class.

2.4 Convolutional Neural Network

Convolutional neural networks (CNN) are named after the mathematical operation convolution. Convolution is often encountered in the context of image processing, with the intensity of a given pixel and 2-dimensional weighting function. The weighting function is usually non-zero only for a few values in the close neighbourhood to the central pixel and therefore the sum has to compute only over those values instead of the whole image. The weighting function is called kernel, often defined as a small square matrix whose size is called the kernel size [10].

2.4.1 Basic Structure of CNN

Convolutional Neural Networks have a different architecture than regular Neural Networks. First of all, the layers are organised in 3 dimensions: width, height and depth. Further, the neurons in one layer do not connect to all the neurons in the next layer but only to a small region of it. Lastly, the final output will be reduced to a single vector of probability scores, organized along the depth dimension.

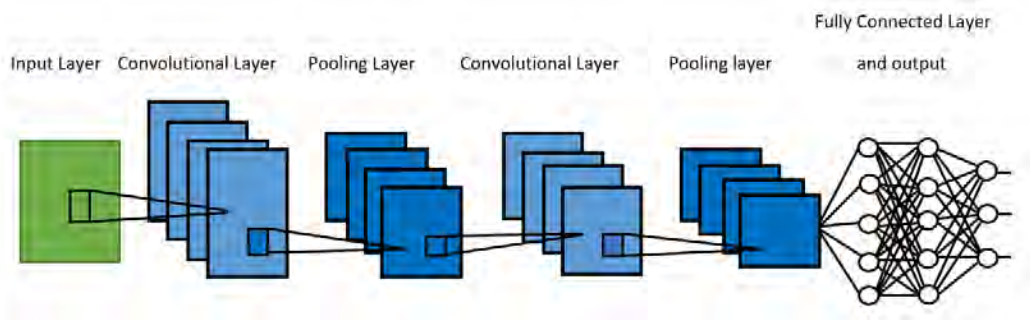


Figure 2.5: Structure of Convolutional Neural Network

A more detailed overview of what CNNs do would be that it takes the image, pass it through a series of convolutional, nonlinear, pooling (down sampling), and fully connected layers, and get an output and the output can be a single class or a probability of classes that best describes the image. An example of CNN architecture is presented in Figure 2.5.

2.4.2 Convolutional Layer

The first layer in a CNN is always a Convolutional Layer. The input is a 32 x 32 x 3 array of pixel values as shown as small cell in Figure 2.6. The top left of the pixel cell with red box is filter that covers a 5 x 5 area and the region that it is covered over is called the receptive field. This filter is also an array of numbers (the numbers are called weights or parameters). A very important note is that the depth of this filter has to be the same as the depth of the input; so the dimensions of this filter is 5 x 5 x 3. For example, the first position of the filter would be the top left corner in Figure 2.6. As the filter is sliding, or convolving, around the input image, it is multiplying the values in the filter with the original pixel values of the image with computing element wise multiplications. These multiplications are all summed up mathematically and there would be 75 multiplications in total and result will be a single number. This number is just representative of when the filter is at the top left of the image. This process is repeated for every location on the input volume and every unique location on the input volume produces a number. After sliding the filter over all the locations, there will be 28 x 28 x 1 array of numbers, which is called an activation map or feature map.

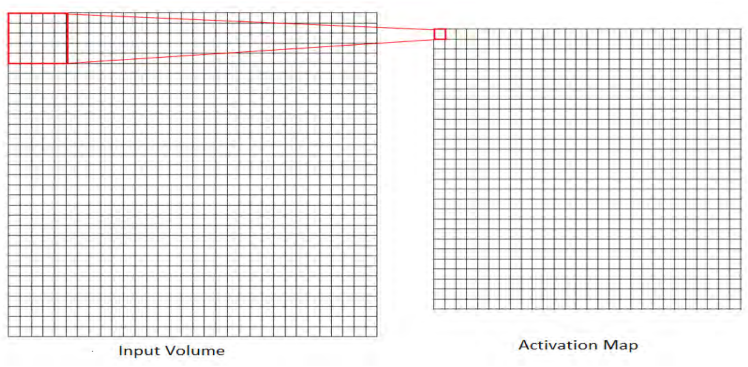
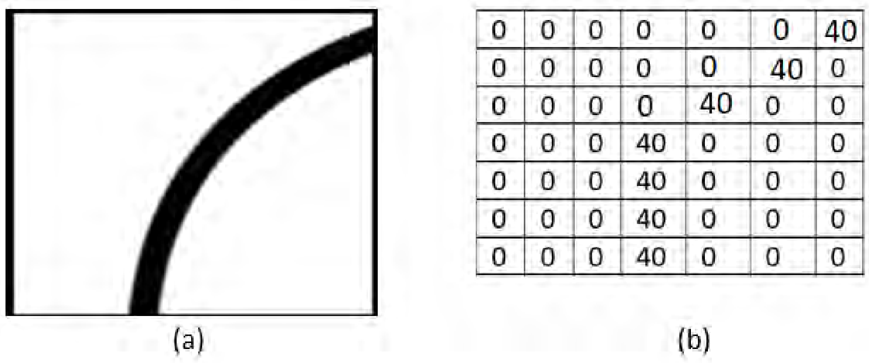


Figure 2.6: Visualization of 5x5 filter convolving around of input volume and producing activation map.

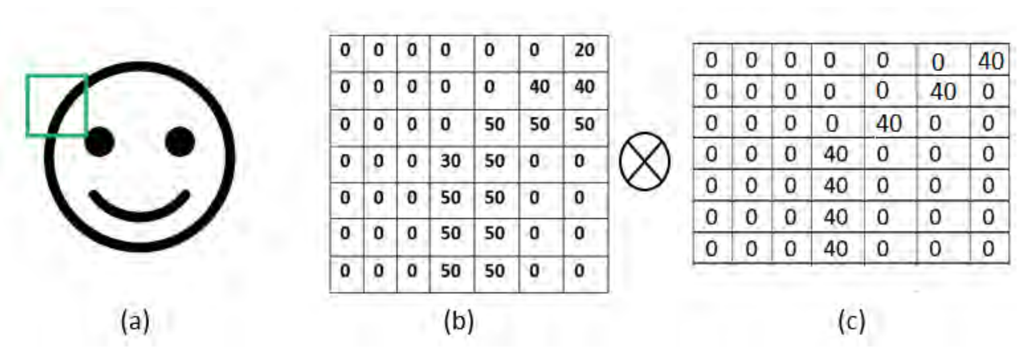
Filters used to create feature map can be thought of as feature identifiers that means different filters can identify different features like straight edges, simple colours, and curves. A curve detector filter, shown in Figure 2.7, will have a pixel structure in which there will be higher numerical values along the area that is a shape of a curve.



0	0	0	0	0	0	40
0	0	0	0	0	40	0
0	0	0	0	40	0	0
0	0	0	40	0	0	0
0	0	0	40	0	0	0
0	0	0	40	0	0	0
0	0	0	40	0	0	0

Figure 2.7: (a) Visualization of curve and (b) corresponding pixel representation

When this filter is at the top left corner of the input volume, it is computing multiplications between the filter and pixel values at that region. Figure 2.8(a) shows an example of an image that need to classify, and a curve filter is put at the top left corner indicating green bounding box to multiply the values in the filter with the original pixel values of the image shown in Figure 2.8(b) and (c).



0	0	0	0	0	0	20
0	0	0	0	0	40	40
0	0	0	0	50	50	50
0	0	0	30	50	0	0
0	0	0	50	50	0	0
0	0	0	50	50	0	0
0	0	0	50	50	0	0

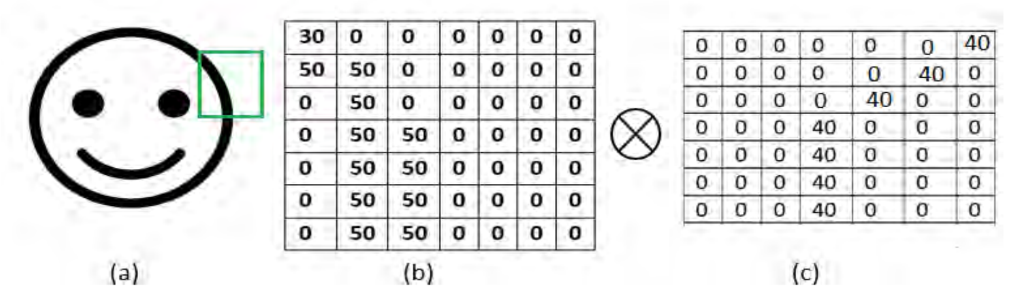
0	0	0	0	0	0	40
0	0	0	0	0	40	0
0	0	0	0	40	0	0
0	0	0	40	0	0	0
0	0	0	40	0	0	0
0	0	0	40	0	0	0
0	0	0	40	0	0	0

Figure 2.8: Convolutional operation with filter

Summation of multiplications=(50x40)+(50x40)+(50x40)+(30x40)+(50x40)+(40x40)+(20x40)

=11600

Basically, in the input image, if there is a shape that generally resembles the curve that this filter is representing, then all of the multiplications summed together will result in a large value. But, when filter is moved to others region, the result is very low. This is because there was not anything in the image section that responded to the curve detector filter. The output of this convolutional layer is an activation map. So, in the simple case of a one filter convolution (and if that filter is a curve detector), the activation map will show the areas that are at mostly likely to be right aligned curves in the picture. In this example, the top left value of 26 x 26 x 1 activation map is 11600. This high value means that it is likely that there is some sort of curve in the input volume that caused the filter to activate. The top right value in of activation map will be 0 because there was not anything in the input volume that caused the filter to activate; simply said, there was not a right aligned curve in that region of the original image shown in Figure 2.9.



30	0	0	0	0	0	0
50	50	0	0	0	0	0
0	50	0	0	0	0	0
0	50	50	0	0	0	0
0	50	50	0	0	0	0
0	50	50	0	0	0	0
0	50	50	0	0	0	0

0	0	0	0	0	0	40
0	0	0	0	0	40	0
0	0	0	0	40	0	0
0	0	0	40	0	0	0
0	0	0	40	0	0	0
0	0	0	40	0	0	0
0	0	0	40	0	0	0

Figure 2.9: Convolutional operation of different pixels with same filter

Summation of multiplications = (0x40)+(0x40)+(0x40)+(0x40)+(0x40)+(0x40)+(0x40)

= 0

It is important to notice that this is just for one filter that is going to detect lines that curve outward and to the right. There may have other filters for lines that curve to the left aligned or for straight edges. The more the filters, the greater the depth of the activation map, and the more information can be gathered about the input volume.

2.4.3 Pooling and Stride

To make the network more manageable for classification, it is useful to decrease the activation map size in the deep end of the network. Generally, the deep layers of the network require less information about exact spatial locations of features, but require more filter matrixes to recognize multiple high-level patterns. By reducing the height and width of the data volume, the depth of the data volume can be increased and the computation time can be kept at a reasonable level [10][29].

There are two ways of reducing the data volume size. One way is to include a pooling layer after a convolutional layer. The layer effectively down-samples the activation maps. Pooling has the added effect of making the resulting network more translation invariant by forcing the detectors to be less precise. However, pooling can destroy information about spatial relationships between subparts of patterns. Typical pooling method used in this network is max-pooling shown in the Figure 2.10. Max-pooling simply outputs the maximum value within a rectangular neighbourhood of the activation map [29].

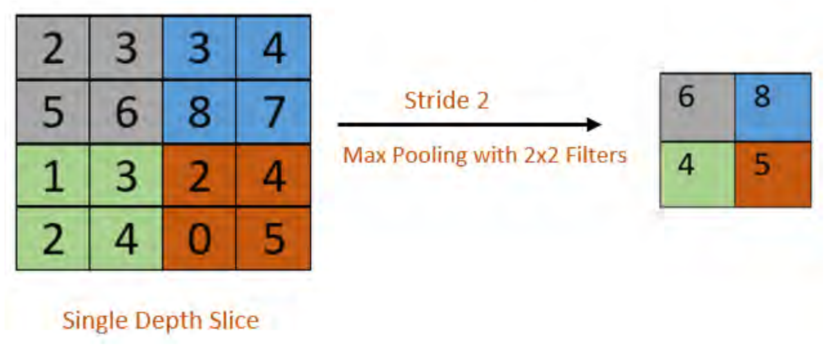


Figure 2.10: Max Pooling and Stride

Another way of reducing the data volume size is adjusting the stride parameter of the convolution operation. The stride parameter controls whether the convolution output is calculated for a neighbourhood centred on every pixel of the input image (stride 1) or for every nth pixel (stride n). It has shown that pooling layers can often be discarded without loss in accuracy by using convolutional layers with larger stride value [10].

2.4.4 Regularization and Augmentation

Regularization refers to methods that are used to reduce overfitting by introducing additional constraints or information to the machine learning system. A classical way of using regularization in neural networks is adding a penalty term to the loss function that penalizes certain types of weights. The parameter sharing feature of convolutional networks is another example of regularization [10]. There are several regularization techniques that are specific to deep neural networks. A popular technique called dropout attempts to reduce the co-adaptation of neurons. This is achieved by randomly dropping out neurons during training, meaning that a slightly different neural network is used for each training sample or minibatch. This causes the system not to depend too much on any single neuron or connection and provides an effective yet computationally inexpensive way of implementing regularization. In convolutional networks, dropout is typically used in the final fully-connected layers. Overfitting can also be reduced by increasing the amount of training data. When it is not possible to acquire more actual samples, data augmentation is used to generate more samples from the existing data [2]. For classification using convolutional networks, this can be achieved by computing transformations of the input images that do not alter the perceived object classes, yet provide additional challenge to the system. The images can be, for example, flipped, rotated or sub sampled with different crops and scales. Also, noise can be added to the input images [29].

2.5 Computer Vision

Computer vision is the method of enabling computers to see, identify, and process images in the same way that human vision does, and to provide appropriate output [32]. It is a challenging task to enable computers to recognize images of different objects in real life. Computer vision can be closely linked with artificial intelligence, as the computer must interpret what it sees, and then, perform appropriate analysis or act accordingly. Computer vision deals with the extraction of meaningful information from the contents of digital images or video. This is distinct from mere image processing, which involves manipulating visual information on the pixel level. Applications of computer vision include image classification, visual detection, 3D scene reconstruction from 2D images, image retrieval, augmented reality, machine vision and traffic automation [4][7][24]. Today, machine learning is a necessary component of many computer vision algorithms that can be described as a combination of image processing and machine learning. Effective solutions require algorithms that can cope with the vast amount of information contained in visual images, and critically for many applications, can carry out the computation in real time [20].

2.5 Summary

Machine learning and deep neural network researches are transforming the modern technology by many amazing advances in the recent years. The model performances and accuracy can be increased by following different types of methods such as max pooling, augmentation, dropout etc. Different types of machine learning and neural network algorithms are used to solve computer vision problem.

CHAPTER 3

Object Detection Models

3.1 Overview

This chapter discusses the different object detection models that utilize convolutional neural networks to detect objects. In particular, this chapter describes the Faster RCNN and SSD models that combine CNNs with region proposal classification and Single Shot Multi Box detector respectively.

3.2 Region-based Convolutional Neural Network (R-CNN)

The R-CNN model [21] proposal intuitively begin with the region search and then perform the classification using the selective search [14] method to extract 2000 regions from the image to capture object location. These 2000 region proposals are warped into a square and fed into a convolutional neural network that produces a 4096-dimensional feature vector as output. The CNN acts as a feature extractor and the output dense layer consists of the features extracted from the image and the extracted features are fed into an SVM to classify the presence of the object within that candidate region proposal. Figure 3.1 presents the architecture of R-CNN where different region of an input image re-extracted, wrapped and convolved to generate feature vectors

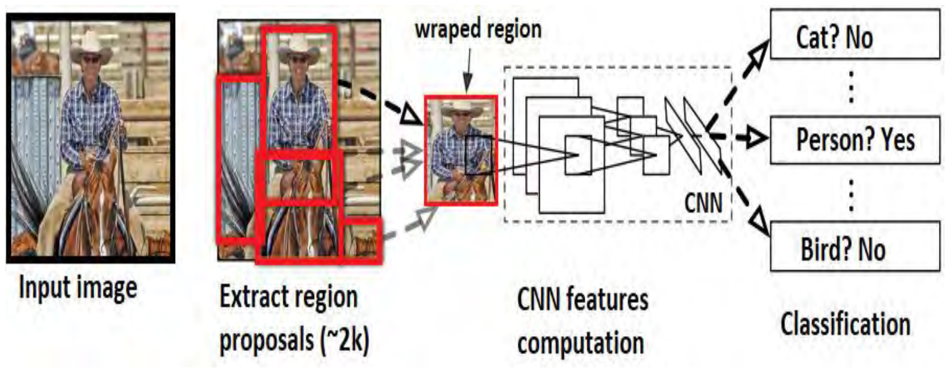


Figure 3.1: The architecture of R-CNN [21].

A linear regressor is used to adapt the shapes of the bounding box for a region proposal and to reduce the localization errors. In addition to predicting the presence of an object within the region proposals, the algorithm also predicts four values which are offset values to increase the precision of the bounding box. For example, given a region proposal, the algorithm would have predicted the presence of a person but the face of that person within that region proposal could have been cut in half. Therefore, the offset values help in adjusting the bounding box of the region proposal. Figure 3.2 shows the RCNN work flow where the output of each convolutional networks is fed into SVM and bounding box regressor.

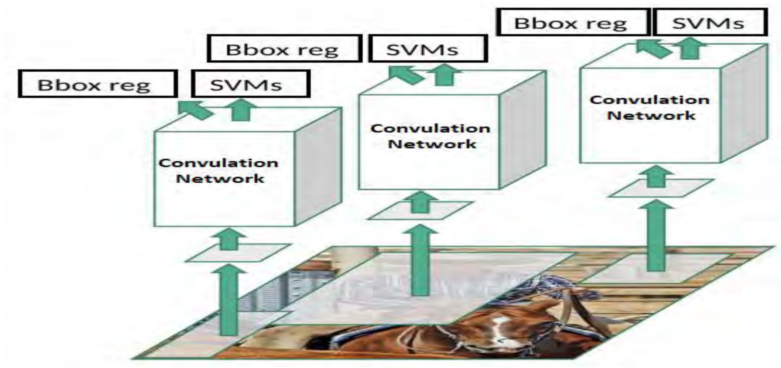


Figure 3.2: Region-based Convolution Network [29].

The CNN model in [29] is trained on the 2012 ImageNet dataset of the original challenge of image classification. It is fine-tuned using the region proposals corresponding to an IoU greater than 0.5 with the ground-truth boxes. Two versions are produced, one version is using the 2012 PASCAL VOC dataset and the other the 2013 ImageNet dataset with bounding boxes. The SVM classifiers are also trained for each class of each data set. The best R-CNNs models have achieved a 62.4% mAP score over the PASCAL VOC2012 test dataset (22.0 points increase w.r.t. the second position result on the leader board) and a 31.4% mAP score over the 2013 ImageNet dataset (7.1 points increase w.r.t. the second position result on the leader board) [11].

There are some drawbacks in RCNN model: it still takes a huge amount of time to train the network as it would have to classify 2000 region proposals per image [21]; it cannot be implemented real time as it takes around 47 seconds for each test image; the selective search [14] algorithm is a fixed algorithm and no learning is happening at that stage and it is not appropriate for real time region proposals application.

3.3 Fast Region-based Convolutional Network (Fast R-CNN)

Girshick et. al. [21] solved some of the drawbacks of R-CNN to build a faster object detection algorithm and it was called Fast R-CNN [11]. The approach is similar to the R-CNN algorithm. But, instead of feeding the region proposals to the CNN, the input image is fed to the CNN to generate a convolutional feature map. From the convolutional feature map, it identifies the region of proposals and warp them into squares by using a RoI pooling layer and reshape them into a fixed size so that it can feed into a fully connected layer. From the RoI feature vector, SoftMax layer is used to predict the class of the proposed region and the offset values for the bounding box. The work flow of Fast RCNN model is shown in Figure 3.3.

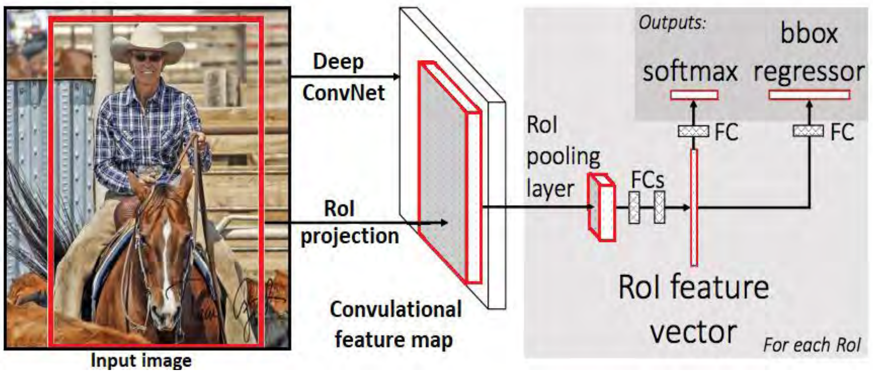


Figure 3.3: The architecture of Fast R-CNN [11].

The reason "Fast R-CNN" is faster than R-CNN is because it does not require to feed 2000 region proposals to the convolutional neural network every time [11]. Instead, the convolution

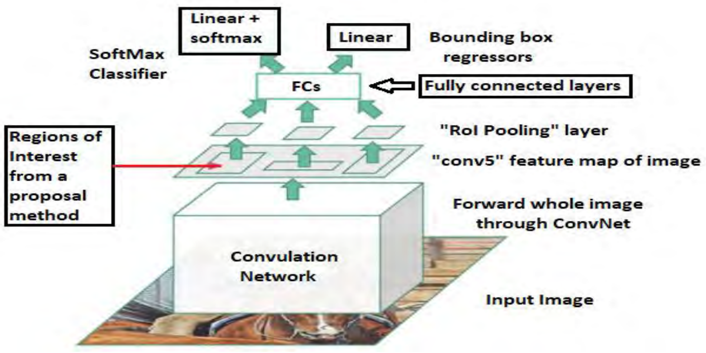


Figure 3.4: Region-based Fast Convolution Network [29].

operation is done only once per image and a feature map is generated from it; then region of interest is applied as shown in Figure 3.4. The best Fast R-CNNs have reached mAP scores of 70.0% for the 2007 PASCAL VOC test dataset, 68.8% for the 2010 PASCAL VOC test dataset and 68.4% for the 2012 PASCAL VOC test dataset [11].

3.4 Faster Region-based Convolutional Neural Network (Faster R-CNN)

Faster R-CNN works to combat the somewhat complex training pipeline that both R-CNN [21] and Fast R-CNN[11] exhibited. The same authors [11] insert a region proposal network (RPN) after the last convolutional layer. This network is able to just look at the last convolutional feature map and produce region proposals from that. From that stage, the same pipeline as R-CNN is used (ROI pooling, FC, and then classification and regression heads). Faster R-CNN, is composed of two networks: region proposal network (RPN) for generating region proposals and a network using these proposals to detect objects. The entire system is a single, unified network for object detection. The main different here with Fast R-CNN is that the later uses selective search [14] to generate region proposals. The time cost of generating region proposals is much smaller in RPN than selective search, when RPN shares the most computation with the object detection network. Briefly, RPN ranks region boxes (called anchors) and proposes the ones most likely containing objects [29]. Figure 3.5 shows the work flow of Faster RCNN model and how anchors are used to detect objects.

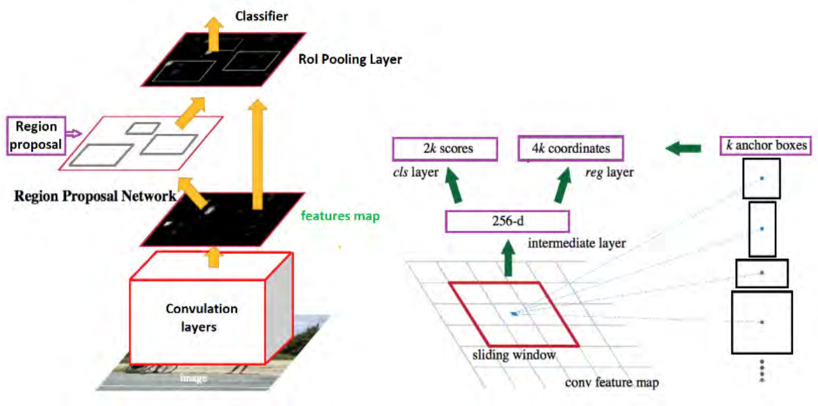


Figure 3.5: An illustration of Faster R-CNN model [5].

Anchor is a box which plays an important role in Faster R-CNN. In the default configuration of Faster R-CNN, there are 9 anchors at a position of an image. An anchor is a combination of sliding window centre, scale, ratio. For example, 3 scales and 3 ratios for k=9 anchors at each sliding position. The sheer size is hardly smaller than the combination of sliding window and pyramid [29]. This is why, it has a coverage as good as other state of the art methods.

The bright side here is that region proposal network can be used from the method in Fast RCNN [11] to significantly reduce number. When the anchor boxes are detected, they are selected by applying a threshold over the "objectness" score to keep only the relevant boxes. These anchor boxes and the feature maps computed by the initial CNN model feeds a Fast R-CNN model [11]. Faster R-CNN uses RPN to avoid the selective search [14] method, it accelerates the training and testing processes, and improve the performances using iterative process [5].

The best Faster R-CNNs have obtained mAP scores of 78.8% over the 2007 PASCALVOC test dataset and 75.9% over the 2012 PASCAL VOC test dataset. The model has been trained with PASCAL VOC and COCO datasets. This models is 34 times faster than the Fast R-CNN[5][11]

3.5 Single Shot MultiBox Detector (SSD)

Single-Shot MultiBox Detector model is developed to predict all at once with the bounding boxes and the class probabilities with end-to-end CNN architecture. Single Shot means that the input image is observed at once and the tasks of object localization and classification are done in a single forward pass of the network. MultiBox is the name of a technique for bounding box regression developed by Szegedy et al [15] and detector is an object detector network that also classifies those detected objects. The model takes an image as input which passes through multiple convolutional layers with different sizes of filter (5x5 ,3x3 and 1x1). Feature maps from convolutional layers at different position of the network are used to predict the bounding boxes. They are processed by a specific convolutional layer with 3x3 filters called extra featurelayers to produce a set of bounding boxes like to the anchor boxes of the Fast RCNN[11]. Each box has 4 parameters: the coordinates of the centre, the width and the height.

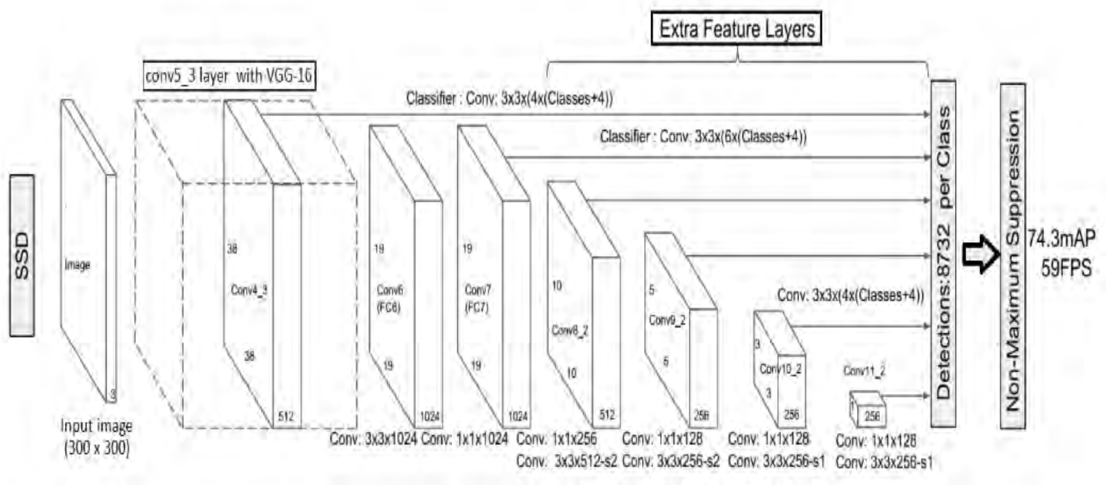


Figure 3.6: Architecture of Single Shot MultiBox detector (input is 300x300x3) [15]

At the same time, it produces a vector of probabilities corresponding to the confidence over each class of object. Every convolution layers have individual classifier for prediction and connected to last stage separately as shown in Figure 3.6. The input image is rescaled in every convolution layer for better detection and SSD model can perform 8732 detections per class.

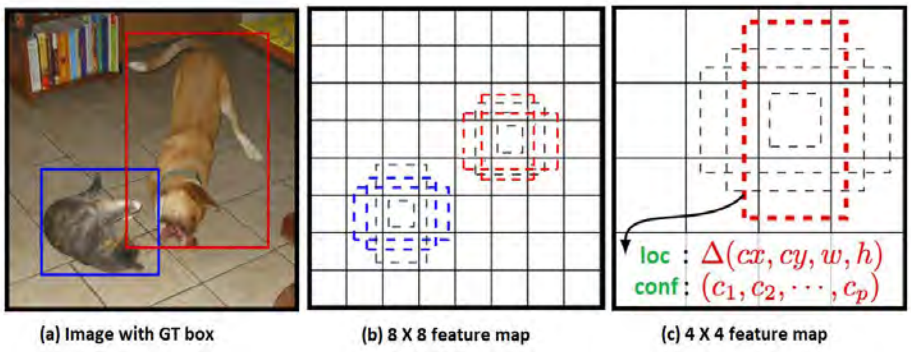


Figure 3.7: SSD Framework. (a) The model takes an image and its ground truth (GT) bounding boxes. Small sets of boxes with different aspect ratios are fixed by the different feature map (b) and (c) [15].

Figure 3.7 shows the bounding box that describes the ground truth and features are extracted from the ground truth area. During training, the box localizations are modified to best match the ground truth. The Non-Maximum Suppression method is also used at the end of the SSD [15] model to keep the most relevant bounding boxes with the help of IoU technique. The maximum coverage area of ground truth by bounding box give the maximum IoU score. The maximum IoU scoring bounding box is kept and other boxes are deleted.

3.6 Summary

Faster RCNN and Single Shot MultiBox Detector models are popular object detector models in deep learning and computer vision field. Faster RCNN uses the region proposal technique where SSD uses the single forward path multi box technique. Both models can be used in real time situation but SSD model with sacrificing some accuracy is Faster than RCNN model. This is, because, computation cost of SSD model is less than Faster RCNN model.

CHAPTER 4

Prerequisite Dependencies and Environment Setup

4.1 Overview

In this chapter, a discussion is provided about software and hardware required to complete the project and environment setup for object detection model.

4.2 Prerequisite of Software and Hardware

This project used several software libraries, packages and programs to utilize machine learning. Python was the choice of programming language, and TensorFlow was used for the deep learning computations, which in turn has a list of dependencies. Anaconda IDE consisting of Jupiter notebook and spyder are used to implement the idea easily. TensorFlow offers a version for CPU usage and another for GPU; this project used the GPU version. This version requires extra programs from the GPU designer NVIDIA, such as CUDA 9.0 Toolkit, cuDNN 7.0.5 and their GPU drivers. So far, NVIDIA is the leading GPU designer for deep learning (also crypto mining and other similar high complex tasks) since they also write programs that are compatible with their cards that enable much of this capacity. The card used for this project was a NVIDIA GeForce mx150.

4.3 Environment setup

To install tensorflow in GPU, the flowing steps were carried out:

1) Prerequisite tools:
   - Nvidia Graphics Card
   - Anaconda with python 3.6 (or 3.5)
   - CUDA Tool kit (version 9.1)
   - CuDNN (version 7.0.5)

2) CUDA Tool Kit [28] installation:
    - A user profile account has been created in Nvidia website [28] to download CUDA Version 9.1
    - CUDNN 7.0.5 has also downloaded for CUDA tool kit 9.1
    - After downloading CUDA 9.1, it has installed in the PC
    - An environment variable path is created for CUDA by adding following paths:
      C:\Program Files\NVIDIA GPU Computing Toolkit\CUDA\v9.1\bin
      C:\Program Files\NVIDIA GPU Computing Toolkit\CUDA\v9.1\libnvvp
      C:\ProgramFiles\NVIDIA GPU Computing Toolkit\CUDA\v9.1\extras\CUPTI\libx64
    - CUDNN zip file has been extracted and a path is added to the bin folder. For example,

      C:\cuda\bin

3) It is required to update the GPU driver if needed
4) Anaconda is downloaded [38] with Python 3.7 and installed in PC
5) Anaconda environment and tensorflow installation:
    - A virtual environment is created named tensorflow_gpu by invoking the following command:
      C:> conda create -n tensorflow_gpu pip python=3.5
    - tensorflow_gpu environment is activated by issuing the following command:
      C:> activate tensorflow_gpu
    - As result, the prompt is changed to *(tensorflow_gpu)C:>*
    - GPU version of TensorFlow is installed by issuing the following command on a single line:
      (tensorflow_gpu)C:> pip install --ignore-installed --upgrade tensorflow-gpu

6) Testing the installation process:
    - Anaconda prompt is opened and commend is run by typing *python*
    - When interpreter opens, the following commands are issued:

```
>>> import tensorflow as tf
  >>> hello = tf.constant('Hello, TensorFlow!')
  >>> sess = tf.Session()
```

>>> print(sess.run(hello))

the output is: `hello, TensorFlow!`

That means tensorflow is installed correctly.

7) Other necessary packages are installed by issuing the following commands:

   (tensorflow_gpu) C:\> conda install -c anaconda protobuf

   (tensorflow_gpu) C:\> pip install lxml

   (tensorflow_gpu) C:\> pip install Cython

   (tensorflow_gpu) C:\> pip install jupyter

   (tensorflow_gpu) C:\> pip install matplotlib

   (tensorflow_gpu) C:\> pip install pandas

   (tensorflow_gpu) C:\> pip install opencv-python

8) Downloading process of TensorFlow Object Detection API repository from GitHub: A working directory is created where TensorFlow object detection framework as well as training images, training data, trained classifier, configuration files, and everything else needed for the object detection classifier are kept. The full TensorFlow object detection API repository is downloaded from https://github.com/tensorflow/models by clicking the "Clone or Download" button and downloading the zip file. The zip file named model (model master) is extracted into working directory folder. The objection models are downloaded from tensorflow model zoo following this link https://github.com/tensorflow/models/blob/master/research/object_detection/g3doc/detection_model_zoo.md

9) A PYTHONPATH variable is created that points to the \models, \models\research, and \models\research\slim directories. It is done by issuing the following commands:

   (tensorflow_gpu)C:\>set PYTHONPATH=C:\tensorflow_gpu\models; C:\tensorflow_gpu \ models\research;C:\tensorflow_gpu\models\research\slim

10) The Tensorflow Object Detection API uses Protobufs to configure model and training parameters. Before the framework can be used, the Protobuf libraries is compiled. Every .proto file in the \object_detection\protos directory is called out individually by the command. In the Anaconda Command Prompt, by changing directories to the \models\research\ object_detection directory and the following command is issued: "protoc object_detection/protos/*.proto --python_out=."

Finally, the following commands are run from the C:\tensorflow_gpu\models\research directory:

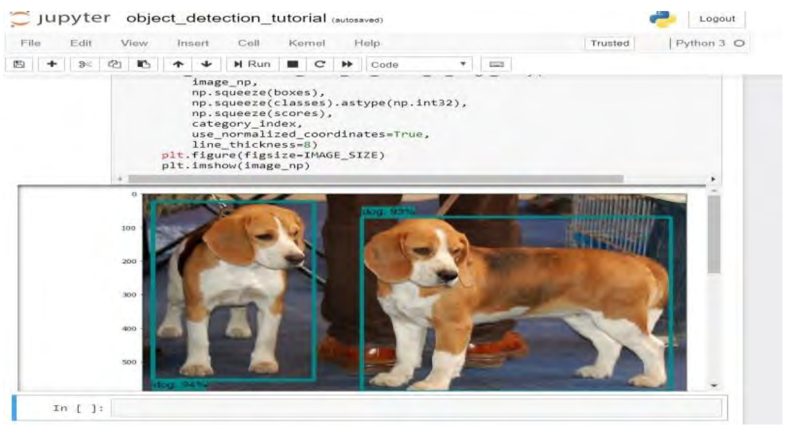
(tensorflow) C:\tensorflow_gpu\models\research> python setup.py build

(tensorflow) C:\tensorflow_gpu\models\research> python setup.py install

11) The TensorFlow Object Detection API is now all set up to use pre-trained models for object detection, or to train a new one. The installation is verified by launching the object_detection_tutorial.ipynb script with Jupyter. From the \object_detection directory, issuing this command:

    (tensorflow_gpu)C:\tensorflow_gpu\models\research\object_detection>jupyter notebook object_detection_tutorial.ipynb

    This opens the script with default web browser and allows to step through the code one section at a time. It can be stepped through each section by clicking the "Run" button in the upper toolbar. The output of this API would be objects surrounding box as shown in the Figure 4.1.



```
        image_np,
        np.squeeze(boxes),
        np.squeeze(classes).astype(np.int32),
        np.squeeze(scores),
        category_index,
        use_normalized_coordinates=True,
        line_thickness=8)
    plt.figure(figsize=IMAGE_SIZE)
    plt.imshow(image_np)
```

Figure 4.1: Output of objection API

4.4 Summary

It is important to set up the system accurately to run the object detection API. Every command step should follow correctly to complete the procedures. Moreover, Nvidia GPU configuration is mandatory to execute the model because the more powerful GPU is required less time to train a model.

CHAPTER 5

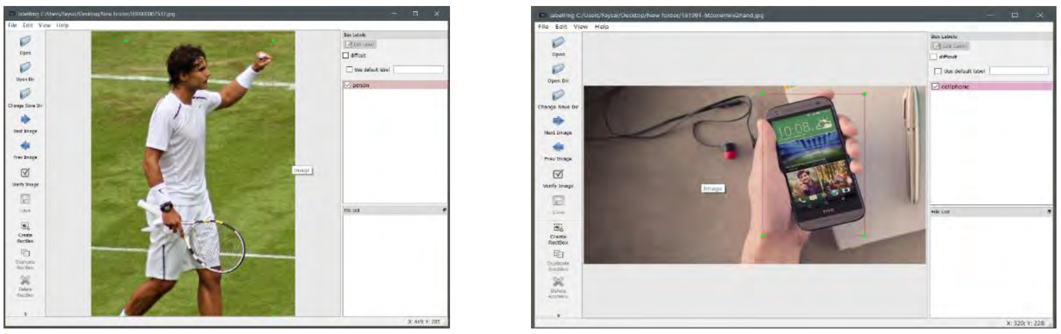
Practical Implementation

5.1 Overview

In this chapter, the overall working procedure are discussed gradually. How images are labelled and how to generate TF record and configure classifier are described step by step.

5.2 Image Gather and Labelling

There are many image data sets available online such as COCO, Pascal VOC etc. In order to create custom object recognized classifier, images from 4 objects such as person, watch, cell phone, and book are gathered as training data set. Total 400 images are collected from four objects and divided into two parts as training and test data set where 70 percent for training and 30 percent for test from each object. First of all, it is required to label the images to classify by the object detection. LabelImage [37] is an easy and great tool for graphical image annotation shown in Figure 5.1 which can be downloaded from GitHub link [37].



(a) (b)

Figure 5.1(a) and (b): Labelling the image

One has to download and install LabelImg and next one, has to point it to \images\train directory, and then, draw a box around each object in each image. The process is repeated for all the images in the \images\test directory.

LabelImg saves a .xml file containing the label data for each image. These .xml files will be used to generate TFRecords, which are one of the inputs to the TensorFlow trainer. For each labelled image, there will be one .xml file in the \test and \train directories. Moreover, it can be checked if the size of each bounding box is correct by running sizeChecker.py by using the following command:

(tensorflow_gpu)C:\tensorflow_gpu\models\research\object_detection>

python sizeChecker.py –move

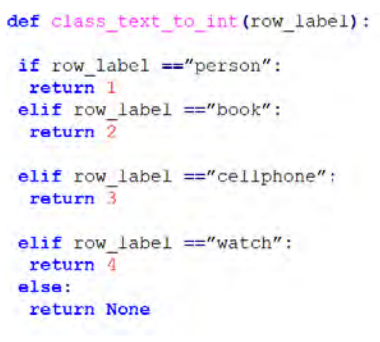
5.3 Generating Tensorflow Record

When images labelling is completed, it generates .xml file with information of every images. It requires a TFRecords that serve as input data to the TensorFlow training model. For this purpose, the xml_to_csv.py and generate_tfrecord.py scripts are used to work with our directory structure. First, the image .xml data are used to create .csv files containing all the data for the train and test images. From the \object_detection folder, the following command is issued in the Anaconda command prompt:

(tensorflow_gpu)C:\tensorflow_gpu\models\research\object_detection>python xml_to_csv.py

This creates a train_labels.csv and test_labels.csv file in the \object_detection\images folder.

Next, it is required to change the label map in generate_tfrecord.py file starting at line 31, where each object is assigned an ID number. This same number assignment is used when configuring the labelmap.pbtxt file in following way:



```
def class_text_to_int(row_label):

 if row_label =="person":
  return 1
 elif row_label =="book":
  return 2

 elif row_label =="cellphone":
  return 3

 elif row_label =="watch":
  return 4
 else:
  return None
```

Then, to generate the TFRecord files, these following commands are issued from the \object_detection folder:

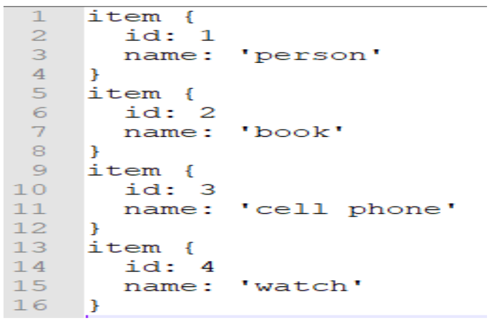
pythongenerate_tfrecord.py --csv_input=images\train_labels.csv --image_dir=images \train --output_path=train.record

python generate_tfrecord.py --csv_input=images\test_labels.csv --image_dir=images \test --output_path=test.record

These generate a train.record and a test.record file in \object_detection. These will be used to train the new object detection classifier.

5.4 Create Label Map

The last thing to do before training is to create a label map and edit the training configuration file. The label map tells the trainer what each object is by defining a mapping of class names to class ID numbers. Use a text editor to create a new file and save it as labelmap.pbtxt in the C:\tensorflow1\models\research\object_detection\ training folder. (it is necessary to keep the file type is .pbtxt, not .txt). In the text editor, the label map is created in following way for 4 objects detector:



```
item {
  id: 1
  name: 'person'
}
item {
  id: 2
  name: 'book'
}
item {
  id: 3
  name: 'cell phone'
}
item {
  id: 4
  name: 'watch'
}
```

5.5 Configure Classifier

Finally, the object detection training pipeline must be configured. It defines which model and what parameters will be used for training. This is the last step before running training.

It is required to navigate at C:\tensorflow_gpu\models\research\object_detection\samples\configs and copy the faster_rcnn_inception_v2_pets.config file (the choosing model) into the

\object_detection\training directory. Then, open the file with a text editor. There are several changes to make to the .config file, mainly changing the number of classes and examples, and adding the file paths to the training data.

The following changes are required to the faster_rcnn_inception_v2_pets.config file. Note: The paths must be entered with single forward slashes (NOT backslashes), or TensorFlow will give a file path error when trying to train the model. Also, the paths must be in double quotation marks ("), not single quotation marks ('). The following changes are made to configure the classifier:

- Line 9. It is required to change num_classes to the number of different objects for the classifier to detect. For the above basketball, shirt, and shoe detector, it would be num_classes : 4
- Line 110. It is required to change fine_tune_checkpoint to:

  fine_tune_checkpoint:
  "C:/tensorflow_gpu/models/research/object_detection/faster_rcnn_inception_v2_coco_2018_01_28/model.ckpt"

- Lines 126 and 128. In the train_input_reader section, change input_path and label_map_path to:

  input_path: "C:/tensorflow_gpu/models/research/object_detection/train.record"

  label_map_path:
  "C:/tensorflow_gpu/models/research/object_detection/training/labelmap.pbtx"

- Line 132. It is required to change num_examples to the number of images that are in the \images\test directory.
- Lines 140 and 142. In the eval_input_reader section, change input_path and label_map_path to:

  input_path :
  "C:/tensorflow1/models/research/object_detection/test.record"

label_map_path:

"C:/tensorflow1/models/research/object_detection/training/labelmap.pbtxt"

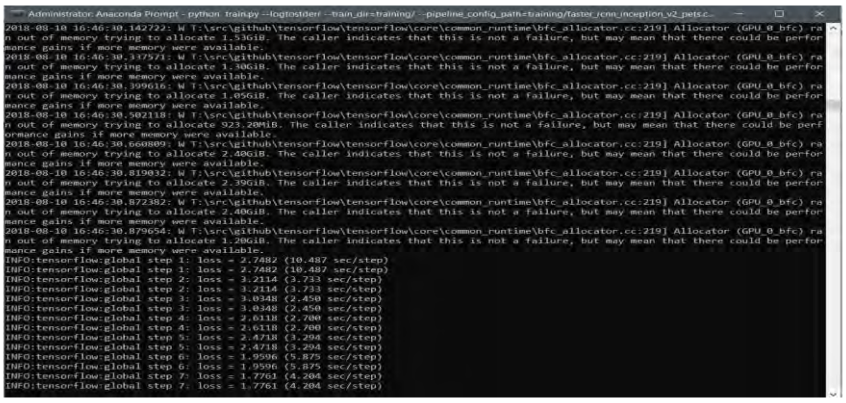
File should be saved after the changes and the training job is all configured and ready to go.

5.6 Training the Classifier

From the \object_detection directory, the following command needs to be issued to begin training:

python train.py --logtostderr --train_dir=training/ --pipeline_config_path=training/ faster_rcnn_inception_v2_pets.config

If everything has been set up correctly, TensorFlow will initialize the training. The initialization can take up to 30 seconds before the actual training begins. When training begins, it almost will look like Figure 5.2.



```
Administrator: Anaconda Prompt - python  train.py --logtostderr --train_dir=training/ --pipeline_config_path=training/faster_rcnn_inception_v2_pets.c...
2018-08-10 16:46:30.142722: W T:\src\github\tensorflow\tensorflow\core\common_runtime\bfc_allocator.cc:219] Allocator (GPU_0_bfc) ra
n out of memory trying to allocate 1.53GiB. The caller indicates that this is not a failure, but may mean that there could be perfor
mance gains if more memory were available.
2018-08-10 16:46:30.337571: W T:\src\github\tensorflow\tensorflow\core\common_runtime\bfc_allocator.cc:219] Allocator (GPU_0_bfc) ra
n out of memory trying to allocate 1.30GiB. The caller indicates that this is not a failure, but may mean that there could be perfor
mance gains if more memory were available.
2018-08-10 16:46:30.399616: W T:\src\github\tensorflow\tensorflow\core\common_runtime\bfc_allocator.cc:219] Allocator (GPU_0_bfc) ra
n out of memory trying to allocate 1.05GiB. The caller indicates that this is not a failure, but may mean that there could be perfor
mance gains if more memory were available.
2018-08-10 16:46:30.502118: W T:\src\github\tensorflow\tensorflow\core\common_runtime\bfc_allocator.cc:219] Allocator (GPU_0_bfc) ra
n out of memory trying to allocate 923.20MiB. The caller indicates that this is not a failure, but may mean that there could be perf
ormance gains if more memory were available.
2018-08-10 16:46:30.660809: W T:\src\github\tensorflow\tensorflow\core\common_runtime\bfc_allocator.cc:219] Allocator (GPU_0_bfc) ra
n out of memory trying to allocate 2.40GiB. The caller indicates that this is not a failure, but may mean that there could be perfor
mance gains if more memory were available.
2018-08-10 16:46:30.819032: W T:\src\github\tensorflow\tensorflow\core\common_runtime\bfc_allocator.cc:219] Allocator (GPU_0_bfc) ra
n out of memory trying to allocate 2.39GiB. The caller indicates that this is not a failure, but may mean that there could be perfor
mance gains if more memory were available.
2018-08-10 16:46:30.872382: W T:\src\github\tensorflow\tensorflow\core\common_runtime\bfc_allocator.cc:219] Allocator (GPU_0_bfc) ra
n out of memory trying to allocate 2.40GiB. The caller indicates that this is not a failure, but may mean that there could be perfor
mance gains if more memory were available.
2018-08-10 16:46:30.879654: W T:\src\github\tensorflow\tensorflow\core\common_runtime\bfc_allocator.cc:219] Allocator (GPU_0_bfc) ra
n out of memory trying to allocate 1.20GiB. The caller indicates that this is not a failure, but may mean that there could be perfor
mance gains if more memory were available.
INFO:tensorflow:global step 1: loss = 2.7482 (10.487 sec/step)
INFO:tensorflow:global step 1: loss = 2.7482 (10.487 sec/step)
INFO:tensorflow:global step 2: loss = 3.2114 (3.733 sec/step)
INFO:tensorflow:global step 2: loss = 3.2114 (3.733 sec/step)
INFO:tensorflow:global step 3: loss = 3.0348 (2.450 sec/step)
INFO:tensorflow:global step 3: loss = 3.0348 (2.450 sec/step)
INFO:tensorflow:global step 4: loss = 2.6118 (2.700 sec/step)
INFO:tensorflow:global step 4: loss = 2.6118 (2.700 sec/step)
INFO:tensorflow:global step 5: loss = 2.4718 (3.294 sec/step)
INFO:tensorflow:global step 5: loss = 2.4718 (3.294 sec/step)
INFO:tensorflow:global step 6: loss = 1.9596 (5.875 sec/step)
INFO:tensorflow:global step 6: loss = 1.9596 (5.875 sec/step)
INFO:tensorflow:global step 7: loss = 1.7761 (4.204 sec/step)
INFO:tensorflow:global step 7: loss = 1.7761 (4.204 sec/step)
```

Figure 5.2: Starting part of training screen

Each step of training reports the loss. It will start high and get lower and lower as training progresses. For this training on the Faster-RCNN-Inception-V2 model, it started at about 3.0 and quickly dropped below 0.8. The loss numbers will be different if a different model is used. MobileNet-SSD starts with a loss of about 20, it should be trained until the loss is consistently under 2.

The progress of the training job can be viewed by using TensorBoard. To do this, by opening the Anaconda Prompt with activation of virtual environment following command should be issued from C:\tensorflow_gpu\models\research\object_detection directory.

(tensorflow_gpu)C:\tensorflow_gpu\models\research\object_detection>tensorboard --logdir=training

This will create a webpage on local machine at PCName:6006, which can be viewed through a web browser. The TensorBoard page provides information and graphs that show how the training is progressing. One important graph is the Loss graph, which shows the overall loss of the classifier over time.

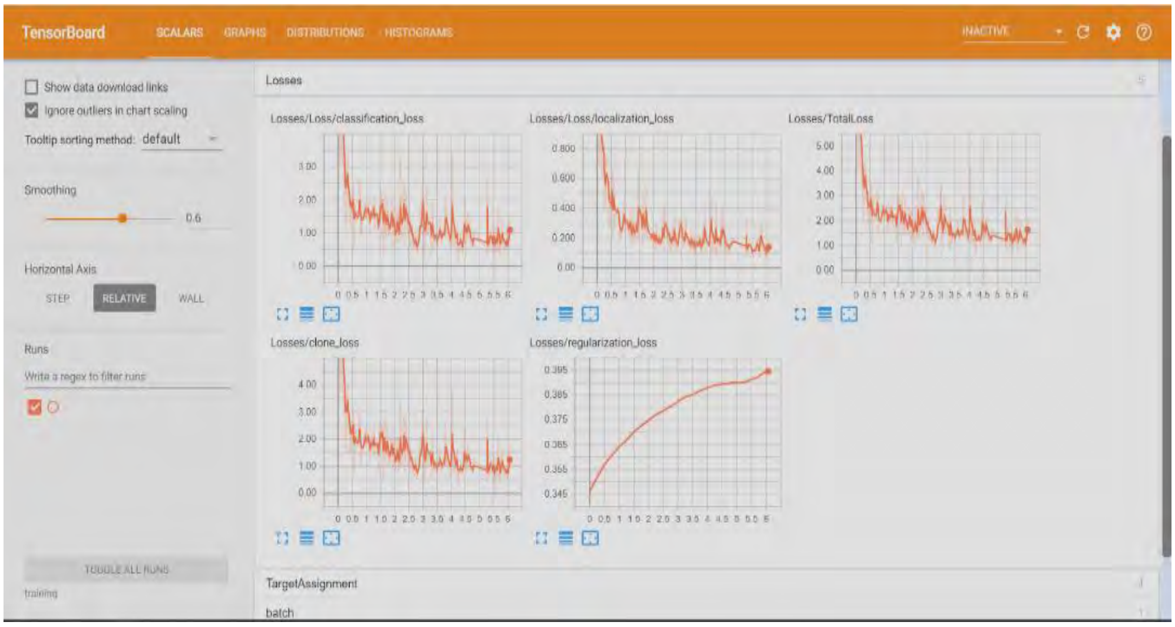


Figure 5.3: Tensor board display on browser

The training routine periodically saves checkpoints about every five minutes. The training can be terminated by pressing Ctrl+C in the command prompt window. It is good to typically wait until just after a checkpoint has been saved to terminate the training. The checkpoint at the highest number of steps will be used to generate the frozen inference graph.

5.7 Export Inference Graph

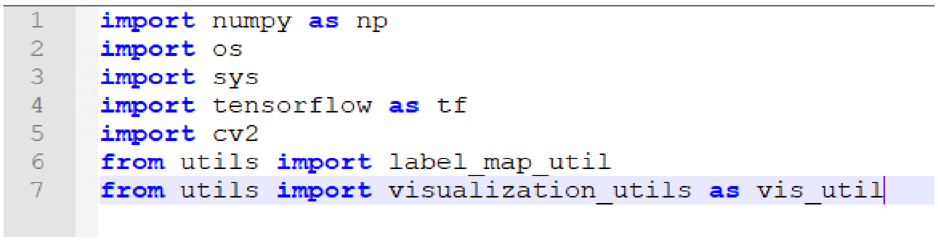
When the training is completed, the last step is to generate the frozen inference graph (.pb file). From the \object_detection folder, the following command is issued, where “XXXX” in “model.ckpt-XXXX” is replaced with the highest-numbered .ckpt file in the training folder:

python export_inference_graph.py --input_type image_tensor --pipeline_config_path training/faster_rcnn_inception_v2_pets.config --trained_checkpoint_prefix training/ model.ckpt-XXXX --output_directory inference_graph

This creates a frozen_inference_graph.pb file in the \object_detection\inference_graph folder. The .pb file contains the object detection classifier.

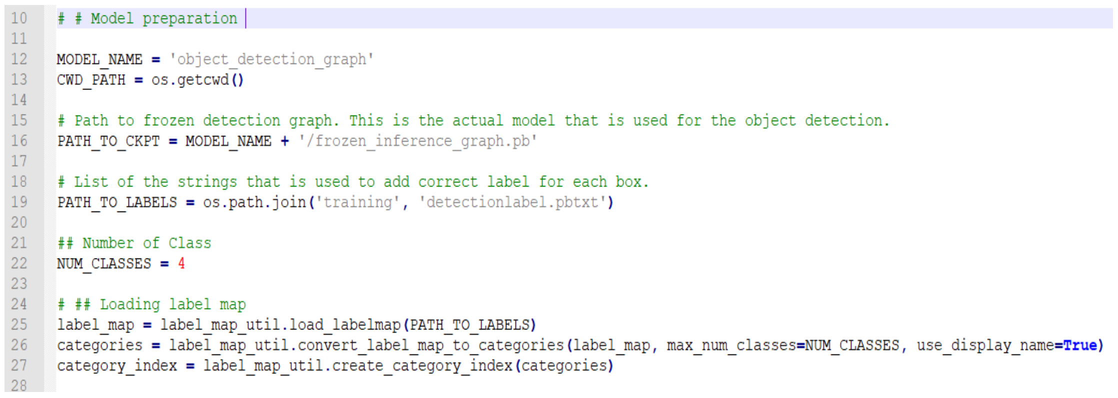
5.8 Main Code

The code can be dissected in to smaller parts for clear understanding (see Appendix II for full code). First comes the different libraries and packages that are needed for the code. The following libraries (line 1 to line 7) like numpy, tensorflow, open cv, object detection utils etc. are included for the code.



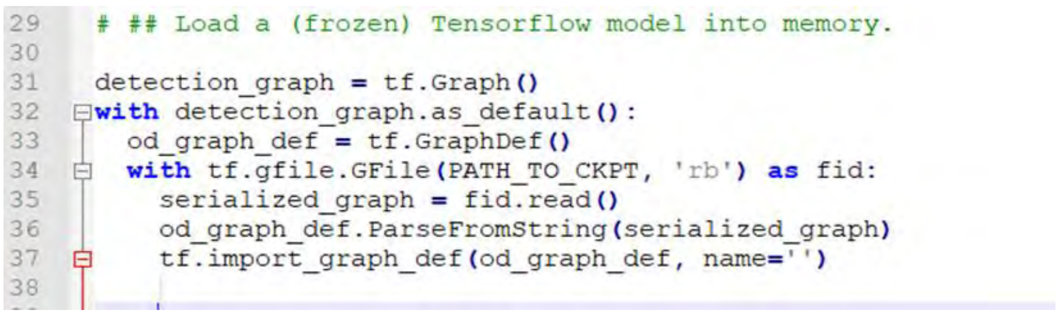
```
1  import numpy as np
2  import os
3  import sys
4  import tensorflow as tf
5  import cv2
6  from utils import label_map_util
7  from utils import visualization_utils as vis_util
```

The next following part (line 12 to line 27) are variables that will be used in the document, such as filenames and paths for the model and label map. The reason for having paths and names separate like this is because it makes it possible to easily switch models by simply changing the filenames at the start of the file.



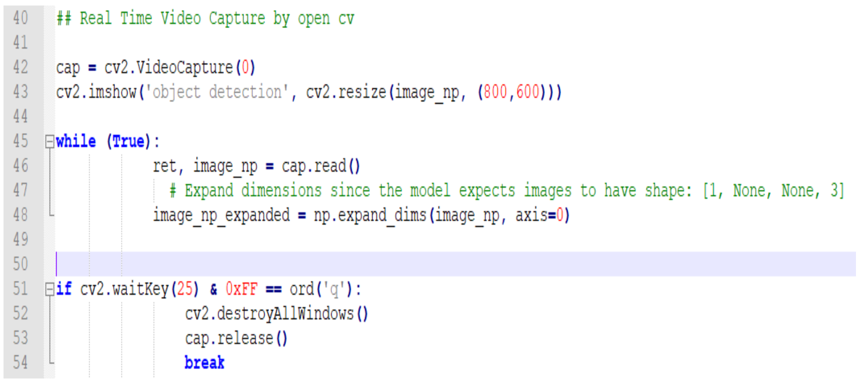
```
10  # # Model preparation
11  
12  MODEL_NAME = 'object_detection_graph'
13  CWD_PATH = os.getcwd()
14  
15  # Path to frozen detection graph. This is the actual model that is used for the object detection.
16  PATH_TO_CKPT = MODEL_NAME + '/frozen_inference_graph.pb'
17  
18  # List of the strings that is used to add correct label for each box.
19  PATH_TO_LABELS = os.path.join('training', 'detectionlabel.pbtxt')
20  
21  ## Number of Class
22  NUM_CLASSES = 4
23  
24  # ## Loading label map
25  label_map = label_map_util.load_labelmap(PATH_TO_LABELS)
26  categories = label_map_util.convert_label_map_to_categories(label_map, max_num_classes=NUM_CLASSES, use_display_name=True)
27  category_index = label_map_util.create_category_index(categories)
28  
```

With the basics covered, next comes the main part of the code (line 31 to line 37) that is run. First the model is loaded into the memory, and a TensorFlow session is initiated.



```
29  # ## Load a (frozen) Tensorflow model into memory.
30
31  detection_graph = tf.Graph()
32  with detection_graph.as_default():
33    od_graph_def = tf.GraphDef()
34    with tf.gfile.GFile(PATH_TO_CKPT, 'rb') as fid:
35      serialized_graph = fid.read()
36      od_graph_def.ParseFromString(serialized_graph)
37      tf.import_graph_def(od_graph_def, name='')
38
```

With the use of the library called CV2 in following code (line 42 to line 54), a video feed can be opened, with the window size of 800 by 600 pixels. A package with FPS tracking is also

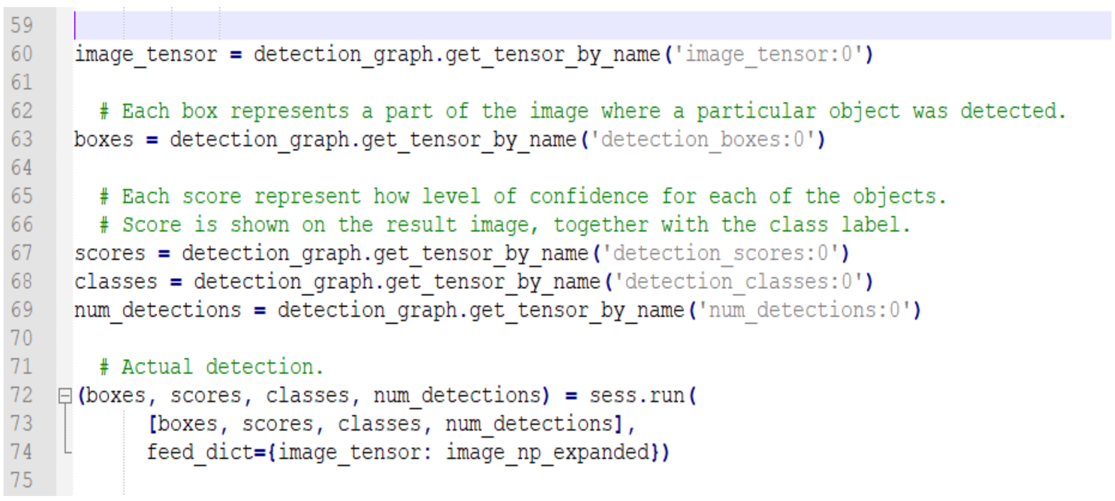


```
40  ## Real Time Video Capture by open cv
41
42  cap = cv2.VideoCapture(0)
43  cv2.imshow('object detection', cv2.resize(image_np, (800,600)))
44
45  while (True):
46              ret, image_np = cap.read()
47                # Expand dimensions since the model expects images to have shape: [1, None, None, 3]
48              image_np_expanded = np.expand_dims(image_np, axis=0)
49
50
51  if cv2.waitKey(25) & 0xFF == ord('q'):
52                  cv2.destroyAllWindows()
53                  cap.release()
54                  break
```

started.

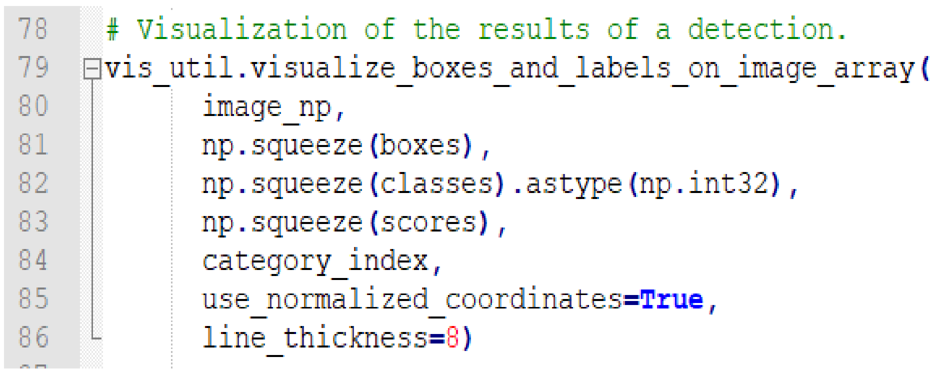
np.expand_dims() expand the dimension since the model expects images to have shape [1,None,None,3]

Afterwards 5 tensors are called from the graph with the TensorFlow package (tf.Tensor) (line 60 to line 74). This is the information that will be used for the detection. Next the object detection API is used to do the object detection and identification, the variables that was created on line 72-74 is used and will return updated information.



```
59
60  image_tensor = detection_graph.get_tensor_by_name('image_tensor:0')
61
62    # Each box represents a part of the image where a particular object was detected.
63  boxes = detection_graph.get_tensor_by_name('detection_boxes:0')
64
65    # Each score represent how level of confidence for each of the objects.
66    # Score is shown on the result image, together with the class label.
67  scores = detection_graph.get_tensor_by_name('detection_scores:0')
68  classes = detection_graph.get_tensor_by_name('detection_classes:0')
69  num_detections = detection_graph.get_tensor_by_name('num_detections:0')
70
71    # Actual detection.
72  (boxes, scores, classes, num_detections) = sess.run(
73        [boxes, scores, classes, num_detections],
74        feed_dict={image_tensor: image_np_expanded})
75
```

With the help of numpy and another TensorFlow utility (line 79 to line 86) that creates an overlay on top of the original frame information that was gathered through TensorFlow. It draws a box around the object that was found, writes what type of object it is and the confidence that said identification is correct. There are also some additional settings like the thickness of the border and format of the coordinates. The function does not provide an easy way to change fonts. So, it is needed to open the file where the visualize_boxes_and_labels_on_image_array method was written and manually changed the font with some hardcode. Lastly, the newly created frame is returned and subsequently shown to the user.



```
78   # Visualization of the results of a detection.
79   vis_util.visualize_boxes_and_labels_on_image_array(
80           image_np,
81           np.squeeze(boxes),
82           np.squeeze(classes).astype(np.int32),
83           np.squeeze(scores),
84           category_index,
85           use_normalized_coordinates=True,
86           line_thickness=8)
```

5.9 Summary

How the object detector model is implemented practically is described in this chapter completely. There are some procedures strictly followed from image labelling to model training. Tensor board shows the update of training information.

CHAPTER 6

Evaluation

6.1 Overview

This chapter describes the evaluation part of the object detection models. The total loss curve is shown as the metric of evaluation. The training and testing parts are discussed separately.

6.2 Training

This section provides the discussion about training and testing accuracy of mobileNet SSD and Faster RCNN inception v2 pre-trained model. During the training, Tensorboard shows the performance of every model in graphical representation. Figure 6.1 shows the total loss of SSD model. In this model average loss is reduced to below 1.5 after completing 25 thousand steps. To complete those training steps, it takes approximate 6 hours and 30 minutes.

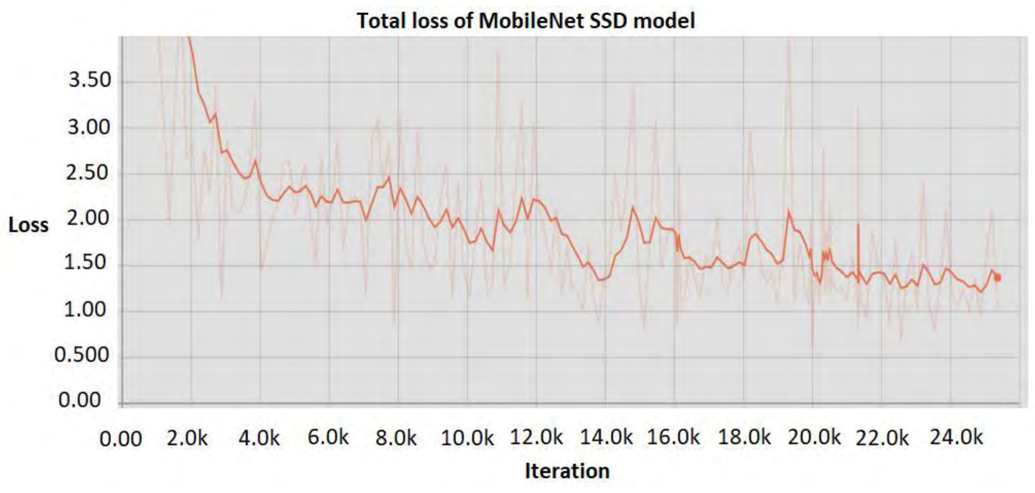


Figure 6.1: Total loss of mobileNet SSD model

Total loss of Faster RCNN model shown in Figure 6.2, is reduced to below 0.1 by completing almost 17 thousand steps within 3 hours training.

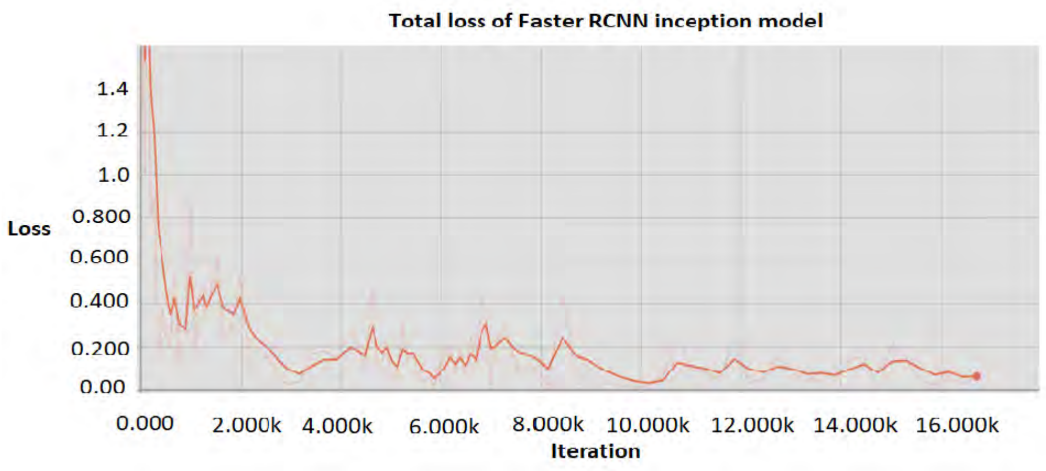


Figure 6.2: Total loss of Faster RCNN inception model v2

6.3 Testing

When training is completed both models are tested through camera from laptop. The testing device has the same configuration as training device. The output of the two classifier models are shown in the Figure 6.3 and Figure 6.4. Faster R-CNN model can successfully detect the all objects with high score where SSD mobileNet cannot detect all objects. So, accuracy of the RCNN model is higher than SSD model in this scenario.

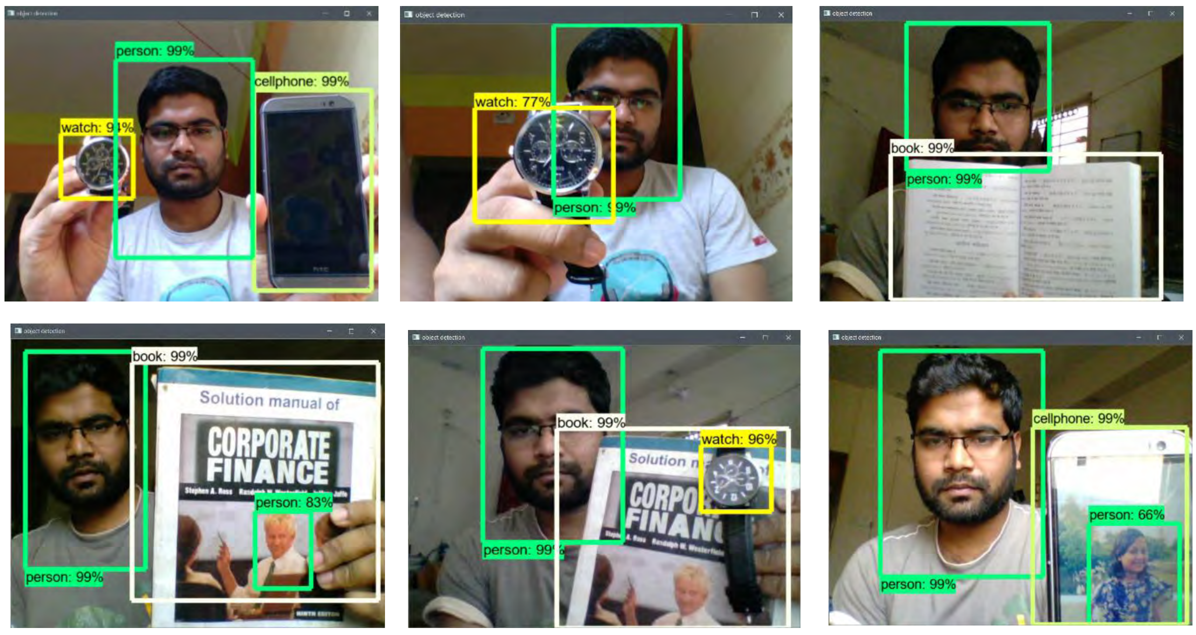


Figure 6.3: Faster RCNN inception model

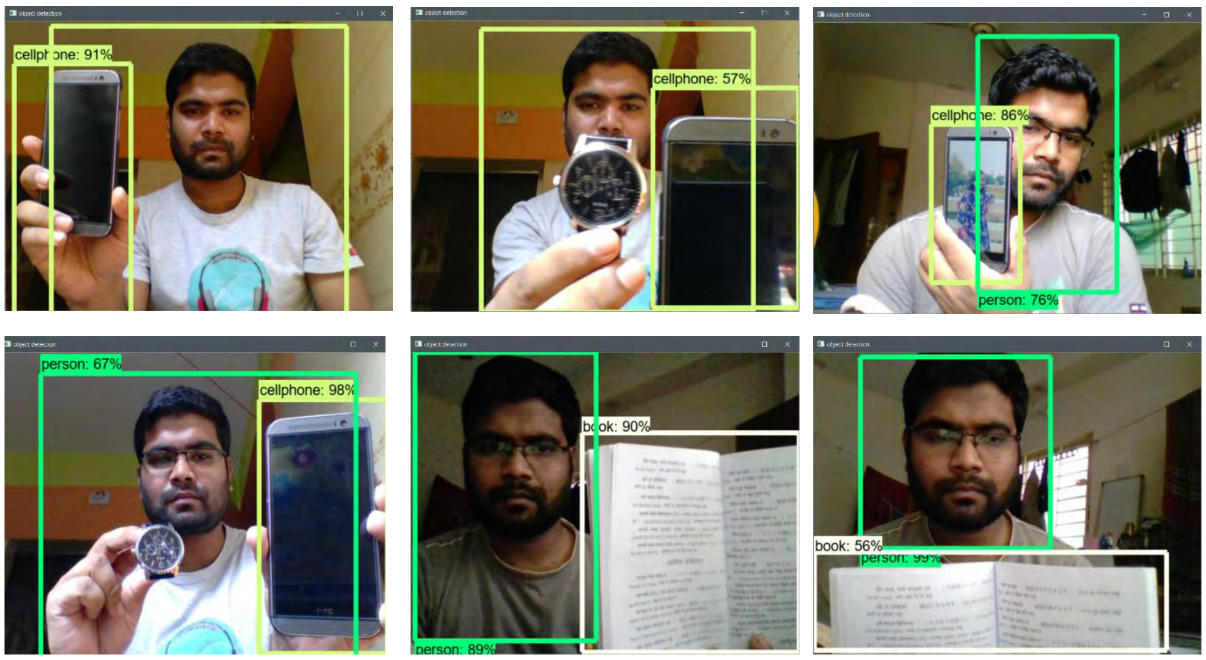


Figure 6.4: SSD mobileNet model

By comparing these two models, it has been shown that Faster RCNN model is more powerful than SSD model. It takes significantly less time and less steps to achieve more accuracy than SSD model. During testing, Faster RCNN model can detect all objects accurately where SSD model fails to detect some objects. This is because, Faster RCNN model has low training accuracy than SSD model. Another reason is camera resolution; camera of this laptop is not good in resolution. That's why, model sometimes fails to classify the objects. Faster RCNN model is slightly slower than SSD model in real time response. The accuracy of both the models can be increased by increasing training data and training period.

6.4 Summary

This project has been successfully run to detect and recognize the object in front of the camera module in both trained models. Any model can be used to train object detection classifier based on situation. If it is planning on using the object detector on a device with low computational power (such as a smart phone or Raspberry Pi), SDD-MobileNet model is appropriate. If it is possible to run the detector on a decently powered laptop or desktop PC with Nvidia GPU, one of the RCNN models can be used.

CHAPTER 7

Conclusion and Future work

7.1 Conclusion

In this project, mobileNet SSD and Faster RCNN object detection models are studied. Descriptions are made on how the model works and which model is better for real time object detection. These models can be trained with any number of object category. The accuracy can be increased by increasing the training period and training data set. It is better if powerful Nvidia GPU is available. More powerful GPU gives more cuda computing scope to get better result in less time.

7.2 Future work

There is a huge scope for future work for improving the application base. It is to be mentioned that the primary focus for the future work is not only to improve the algorithm itself, but to improve upon the usability and application itself. A very short list of probable and promising future aspect of continuing this work is mentioned below:

- Face Recognition
- Human movement detection and recognition
- Human facial expression recognition
- Hand digit recognition
- Traffic signal recognition and monitor
- Sign language recognition
- Gender classification

References

[1] Karpathy, A., CS231n Convolutional Neural Networks for Visual Recognition. Convolutional Neurons. http://cs231n.github.io/ convolutional-networks/. Accessed 24 January 2018

[2] Bishop, C., M., Pattern Recognition and Machine Learning (Information Science and Statistics). Springer-Verlag New York, Inc., Secaucus, NJ, USA, 2006.

[3] Deng, J. et al. ,Imagenet: A large-scale hierarchical image database. Computer Vision and Pattern Recognition, 2009. CVPR 2009. IEEE Conference on. IEEE. 2009, pp. 248–255.

[4] Lin, T., Y., et al. Microsoft coco: Common objects in context. In: European Conference on Computer Vision. Springer. 2014, pp. 740–755.

[5] Ren, S. et al., Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks., CoRRabs/1506.01497(2015). url: http://arxiv.org/abs/1506.01497.

[6] Redmon, J. et al., You only look once: Unified, real-time object detection. In: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition. 2016, pp. 779–788.

[7] Dei, J., Li, Y., He, K., Sun, J. R-FCN: Object Detection via Region-based Fully Convolutional Networks. In arXiv: 1605.06409, 2017.

[8] Ketkar, N., Deep Learning with Python: A Hands-on Introduction, Bangalore, Karnataka, India, ISBN-13 (electronic): 978-1-4842-2766-4, DOI 10.1007/978-1-4842-2766-4

[9] Zeiler, M., D., Fergus,R., Visualizing and Understanding Convolutional Networks. In arXiv:1311.2901v3,2013

[10] Goodellow, I., Bengio, Y., Courville, A., Deep Learning (Adaptive Computation and Machine Learning), An MIT Press Book, url: https://www.deeplearningbook.org/ Accessed: 2018-05-28.

[11] Ross, G., Fast R-CNN, Proceedings of the IEEE International Conference on Computer Vision. 2015, pp. 1440–1448.

[12] Zeiler, M. and Fergus, R., Visualizing and understanding convolutional networks. In: European conference on computer vision. Springer. 2014, pp. 818–833.

[13] Simonyan, K. and Zisserman, A., Very deep convolutional networks for large-scale image recognition. In: arXiv preprint arXiv:1409.1556 (2014)

[14] Uijlings, J. R. R. et al, Selective Search for Object Recognition, url: http://disi.unitn.it/ uijlings/SelectiveSearch.html

[15] Liu, W., Szegedy, C., et al, SSD: Single Shot MultiBox Detector, In arXiv:1512.02325

[16] Erhan, D., Szegedy, C., Toshev, A., and Anguelov, D. Scalable object detection using deep neural networks. In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (2014), pp. 2147

[17] Xiang, Y., Choi, W., Lin,Y. and Savarese, S., Subcategory-aware convolutional neural networks for object proposals and detection. In arXiv:1604.04693, 2016.

[18] Girshick, R., Fast R-CNN, url: http://arxiv.org/abs/ 1504.08083

[19] Mitchell, T., M., Machine Learing, ISBN-13: 978-0070428072

[20] Henriques, J., F., Carreira, J., Caseiro, R. and Batista, J., Beyond Hard Negative Mining: Efficient Detector Learning via Block-Circulant Decomposition. proceedings of the IEEE International Conference on Computer Vision,2013

[21] Girshick, R., Donahue, J., Malik, J., Rich feature hierarchies for accurate object detection and semantic segmentation Tech report (v5), url: http://arxiv.org: 1311.2524

[22] Chellappa, R., et al "Towards the design of an end-to-end automated system for image and video-based recognition," 2016 Information Theory and Applications Workshop (ITA), La Jolla, CA, 2016, pp. 1-7.doi: 10.1109/ITA.2016.7888183.

[23] Nikan, S. and Ahmadi, M., "Effectiveness of various classification techniques on human face recognition," 2014 International Conference on High Performance Computing & Simulation (HPCS), Bologna, 2014, pp. 651-655. doi: 10.1109/HPCSim.2014.6903749.

[24] Wright, J., Yang, A. Y., Ganesh, A., Sastry, S. S. and Ma, Y., "Robust face recognition via sparse representation," IEEE Trans. Pattern Anal. Mach. Intell., vol. 31, pp. 210-227, February 2009.

[25] Huang, G. B., Zhou, H., Ding, X. and Zhang, R., Extreme learning machine for regression and multiclass classification,IEEE Trans. Syst., Man, Cybern., Syst., vol. 45, pp. 513-529, April 2012.

[26] Nikan, S. and Ahmadi, M., Study of the Effectiveness of Various Feature Extractors for Human Face Recognition for Low Resolution Images, in: Proc. International Conf. on Artificial Intell. and Software Eng. (AISE14). Phuket, pp. 1-6, January 2014.

[27] Wu, J., Ma, L. and Hu, X., Delving deeper into convolutional neural networks for camera relocalization, 2017 IEEE International Conference on Robotics and Automation(ICRA),Singapore,2017,pp. 5644-5651. doi: 10.1109/ICRA.2017.7989663.

[28] Nvidia cuda gpu. https://developer.nvidia.com/cuda-gpus. Accessed: 2018-05-28.

[29] Stanford Lecture:http://cs231n.github.io/. Accessed: 2018-05-28.

[30] TensorFlow gpu install.https://www.tensorflow.org/install/gpu.Accessed: 2018-05-28.

[31] Objection API installation. https://github.com/tensorflow/models/blob/master/research/object_detection/g3doc/installation.md. Accessed: 2018-05-28.

[32] Wikipedia link: https://en.wikipedia.org/wiki/Machine_learning

[33] InfoTrend: link: https://www.infortrend.com, Accessed: 2018-05-28.

[34] SSD github link:https://github.com/weiliu89/caffe/tree/ssd. Accessed: 2018-05-28.

[35] Liu,W. et al. SSD:Single Shot MultiBox Detector. Retrieved July3, 2018, url: http://web.cs. ucdavis.edu /~yjlee/teaching/ecs289gwinter2018/ SSD.pdf

[36] Neural network url: https://www.doc.ic.ac.uk/~nd/s urprise_96/journal/ vol4/cs11 / report. html. Accessed: 2018-05-28.

[37] LabelImage url: https://github.com/tzutalin/labelImg. Accessed: 2018-05-28.

[38] Anaconda download url: https://www.anaconda.com/download/ Accessed: 2018-05-28.

APPENDIX I: Main Code

File name: object_detection_spyder.py

```
import numpy as np

import os

import six.moves.urllib as urllib

import sys

import tarfile

import tensorflow as tf

import zipfile

from collections import defaultdict

from io import StringIO

from PIL import Image

import cv2

from utils import label_map_util

from utils import visualization_utils as vis_util


# # Model preparation

MODEL_NAME = 'object_detection_graph'

CWD_PATH = os.getcwd()


# Path to frozen detection graph. This is the actual model that is used for the object detection.

PATH_TO_CKPT = MODEL_NAME + '/frozen_inference_graph.pb'

#PATH_TO_CKPT                                                    =
os.path.join(CWD_PATH,MODEL_NAME,'frozen_inference_graph.pb')
```

```
# List of the strings that is used to add correct label for each box.
PATH_TO_LABELS = os.path.join('training', 'detectionlabel.pbtxt')
# Grab path to current working directory
NUM_CLASSES = 4



# ## Load a (frozen) Tensorflow model into memory.
detection_graph = tf.Graph()
with detection_graph.as_default():
  od_graph_def = tf.GraphDef()
  with tf.gfile.GFile(PATH_TO_CKPT, 'rb') as fid:
    serialized_graph = fid.read()
    od_graph_def.ParseFromString(serialized_graph)
    tf.import_graph_def(od_graph_def, name='')

# ## Loading label map
label_map = label_map_util.load_labelmap(PATH_TO_LABELS)
categories = label_map_util.convert_label_map_to_categories(label_map,
max_num_classes=NUM_CLASSES, use_display_name=True)
category_index = label_map_util.create_category_index(categories)

# ## Helper code
def load_image_into_numpy_array(image):
  (im_width, im_height) = image.size
  return np.array(image.getdata()).reshape(
      (im_height, im_width, 3)).astype(np.uint8)


# # Detection
cap = cv2.VideoCapture(0)
with detection_graph.as_default():
    with tf.Session(graph=detection_graph) as sess:
        ret=True
        while (ret):
            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)
              image_tensor=detection_graph.get_tensor_by_name('image_tensor:0')
 # Each box represents a part of the image where a particular object was detected.
            boxes = detection_graph.get_tensor_by_name('detection_boxes:0')
        # Each score represent how level of confidence for each of the objects.
        # Score is shown on the result image, together with the class label.
            scores = detection_graph.get_tensor_by_name('detection_scores:0')
            classes = detection_graph.get_tensor_by_name('detection_classes:0')
            num_detections = detection_graph.get_tensor_by_name('num_detections:0')
              # Actual detection.
            (boxes, scores, classes, num_detections) = sess.run(
                  [boxes, scores, classes, num_detections],
                  feed_dict={image_tensor: image_np_expanded})
```

```
# Visualization of the results of a detection.
            vis_util.visualize_boxes_and_labels_on_image_array(
                  image_np,
                  np.squeeze(boxes),
                  np.squeeze(classes).astype(np.int32),
                  np.squeeze(scores),
                  category_index,
                  use_normalized_coordinates=True,
                  line_thickness=8)
            cv2.imshow('object detection', cv2.resize(image_np, (800,600)))
            if cv2.waitKey(25) & 0xFF == ord('q'):
                cv2.destroyAllWindows()
                cap.release()
                break
```

APPENDIX II: Common Error

1. ModuleNotFoundError: No module named 'deployment'

This error occurs when object_detection_tutorial.ipynb or train.py are tried to run but don't have the PATH and PYTHONPATH environment variables set up correctly. Then, issue "activate tensorflow_gpu" to re-enter the environment.It is required to use "echo %PATH%" and "echo %PYTHONPATH%" to check the environment variables and make sure that are set up correctly. Also, make sure to have run these commands from the \models\research directory: setup.py build and setup.py install

2. ImportError: cannot import name 'preprocessor_pb2'

ImportError: cannot import name 'string_int_label_map_pb2'(or similar errors with other pb2 files)

This occurs when the protobuf files (in this case, preprocessor.proto) have not been compiled. Re-run the protoc command. Check the \object_detection\protos folder to make sure there is a name_pb2.py file for every name.proto file.

3. 'protoc' is not recognize as internal and external command

This occurs when following command is tried to run "protoc object_detection/protos/*.proto --python_out=."

It isrequired to give the full path where protoc is located in bin folder then run the command.

4. Unsuccessful TensorSliceReader constructor: Failed to get "file path" … The filename, directory name, or volume label syntax is incorrect.

This error occurs when the filepaths in the training configuration file (faster_rcnn_inception_v2_pets.config or similar) have not been entered with backslashes instead of forward slashes. Open the .config file and make sure all file paths are given in the following format: "C:/path/to/model.file"

5. ValueError: Tried to convert 't' to a tensor and failed. Error: Argument must be a dense tensor: range(0, 3) - got shape [3], but wanted [].

The issue is with models/research/object_detection/utils/learning_schedules.py Currently it is

```
rate_index = tf.reduce_max(tf.where(tf.greater_equal(global_step, boundaries),
                      range(num_boundaries),[0] * num_boundaries))
```

Wrap list() around the range() like this:

```
rate_index = tf.reduce_max(tf.where(tf.greater_equal(global_step, boundaries),
                     list(range(num_boundaries)),[0] * num_boundaries))
```