Major Project

ON

Object Detection Using Single Shot Multibox Detector

Submitted in partial fulfillment for the award of the degree of

BACHELOR OF TECHNOLOGY

in

INFORMATION TECHNOLOGY

By

P. Manideep 15211A1258

Under the Esteemed Guidance of

K. Madhavi , M.tech

Assistant Professor

Department of Information Technology

CHAPTER 1

1.INTRODUCTION

When we're shown an image, our brain instantly recognizes the objects contained in it. On the other hand, it takes a lot of time and training data for a machine to identify these objects. Look around and you will see a lot of day to day objects. You recognize them almost instantaneously and involuntarily. You don't have to wait for a few minutes after looking at a table to understand that it is in fact a table. Machines, on the other hand, find it very difficult to do this task. People have been working for decades to find a solution to this problem, but they have only been able to achieve an accuracy of around 65%. Why is it so hard for machines to recognize and categorize objects like humans? What's so difficult here? We do it every day and we get it right almost every single time. What's the missing link? This is actually the holy grail of computer vision!

Let's take a look at how humans recognize and categorize objects. The processing of visual data happens in the ventral visual stream. It is a hierarchy of areas in the brain which helps in object recognition. Humans can easily recognize different sized objects and put them in the same category. This happens because of the invariances we develop. Whenever we look at an object, our brain extracts the features and in such a way that the size, orientation, illumination, perspective etc. don't matter. You remember an object by its shape and inherent features. It doesn't matter how the object is placed, how big or small or in whichever orientation.

Object detection refers to the capability of computer and software systems to locate objects in an image/scene and identify each object. Object detection has been widely used for face detection, vehicle detection, pedestrian counting, web images, security systems and driverless cars.

Like human brain composed of neurons and the feedforward processing, the artificial neurons are used in the deep neural networks for performing object detection.

1.1 INPUT DESIGN

Before proceeding with any object detection tasks, dataset is required. So, the input design involves the developer collecting images consisting of objects on which he is building the model. The next step is to sanitize the images i.e. remove any unnecessary images which may not lead to improvement in efficiency of the object detection model. Later we work on dividing the dataset into training and testing dataset. The division could be 75%-25% or 80%-20% depending on the programmer and the environment where it is being deployed. Here in the current project being implemented, first ratio is used.

The input design considers the following things:

- What kind of images should be used as input
- What is the number of objects being considered?
- What kind of neural network should be used to improve the model efficiency?
- Are we overfitting the model with inputs which reduces efficiency?
- Are there any initial weights associated with the inputs?

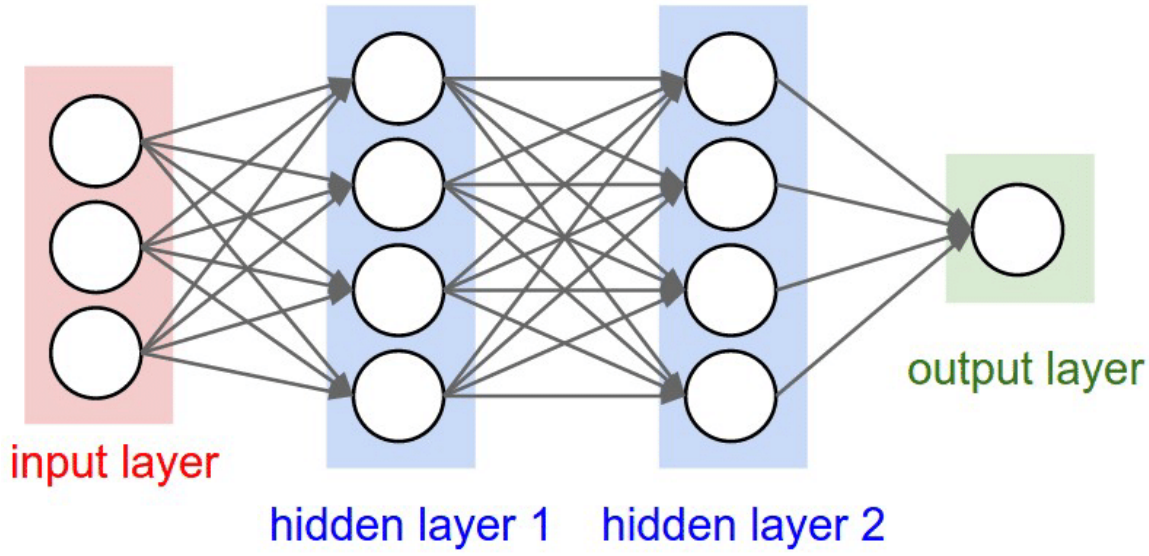

Fig 1.1: Input design for the neural network

Here the input layer will contain the images being fed and the images should be less in storage size and should also be feature rich so that the feature map layer present in the hidden layers can easily extract the features and this will, in turn, increase the efficiency of the model.

1.2 OBJECT DETECTION WORKFLOW

- **Image Classification**

  Image classification is among the most common computer vision problems. An algorithm looks at an image and classifies it as an object. Image classification performs a lot of operations, like face detection to detection of cancer in medicines.

- **Object classification and localization**

  The object localization algorithms would not only help to know the presence of an object, but also the location of the object. A bounding box is drawn around the object in the image.

- **Multiple object detection and localization**

  There could be multiple objects in the image and this is something that would be very common in self-driving cars. The algorithm would not only need to detect other cars but motorcycles, pedestrians, trees, and other objects. When it comes to the context of deep learning, the basic algorithmic difference would be choosing the relevant inputs and outputs.

1.3 OBJECTIVES

The clear objective of the project is to able to detect objects in a given image, video input or webcam feed. The first objective is to use the best possible model which is implemented on deep neural network architecture. The various available models include Faster RCNN which stands for Recursive Neural Networks. This model is only suitable for highly powerful computing machines and it requires large amount of time to train. The other model is YOLO which stands for You Only Look Once. This model has great accuracy of upto 95% and is also fast in terms of training but the same can't be said for mobile device. This model is not optimized for mobile devices which limits its widespread implantation.

The model used here is SSD-Mobilenet. SSD stands for Single Shot Detector. This model localizes and identifies the image in the frame in single step thereby reducing the number of layers in the neural network and hence improving speed.

After the model is selected, the next objective is to gather dataset comprising of the objects to be detected. Once the training is done our next objective is to test the model for accuracy. The testing dataset is used for this and we get the accuracy

for our model. Next objective is to optimize the model as a TensorFlow serving and deploy that to the environment we want to use it for.

1.4 APPLICATIONS

- **FACIAL RECOGNITION**

A deep learning facial recognition system called the "DeepFace" has been developed by a group of researchers in the Facebook, which identifies human faces in a digital image very effectively. Google uses its own facial recognition in Google photos, which automatically segregates all the photos based on the person in the image. There are various components involved in the Facial recognition like eyes, nose, mouth and the eyebrows

- **PEOPLE COUNTING**

Object detection can also be used for people counting, it is used for analyzing store performance or crowd statistics during festivals. These tens to be more difficult as people move out of the frame quickly.

- **INDUSTRIAL QUALITY CHECK**

Object detection is also used in industrial procresses to identify products. Finding a specific object through visual is a basic task that is involved in multiple industrial processes like sorting, inventory management, machining, quality management, packing etc.

- **SECURITY**

Object Detection plays a very important role in security. Be it face ID of the Apple or the retina scan used in all the sci-fi movies . It is alos used by the government to access the security feed and match it with their existing database to find any criminals or to detect the robber's vehicle

- **SELF DRIVING**

Self-driving cars are the Future, there's no doubt in that. But the working behind it is very tricky as it combines a variety of techniques to perceive their surroundings, including radar, laser light, GPS, odometry and computer vision.

CHAPTER 2

2.LITERATURE SURVEY

2.1 RELATED WORK

Early works on object detection were based on template matching techniques and simple part-based models.

- **Viola and Jones (2001)]:** First is the introduction of a new image representation called the "integral image" which allows the features used by our detector to be computed very quickly. Second is a learning algorithm, based on AdaBoost, which selects a small number of critical visual features from a larger set and yields extremely efficient classifiers. Third contribution is a method for combining increasingly more complex classifiers in a "cascade" which allows background regions of the image to be quickly discarded while spending more computation on promising object-like regions.
- **Romdhani et al. (2001) :** Proposed to compute a set of reduced set vectors from the original support vectors. These reduced set vectors are then tested against the test sample sequentially making early rejections possible.
- **Fleuret and Geman (2001):** Starting from training examples, we recursively find larger and larger arrangements which are “decomposable,” which implies the probability of an arrangement appearing on an object decays slowly with its size.
- **Schneiderman and Kanade (2000):** It detects facial features and ignores anything else, such as buildings, trees and other parts of the body.
- **Sung and Poggio (1998):** The technique models the distribution of human face patterns by means of a few view-based “face” and “nonface” model clusters. At each image location, a difference feature vector is computed between the local image pattern and the distribution-based model.
- **Rowley et al. (1998):** A retinally connected neural network examines small windows of an image and decides whether each window contains a face. The system arbitrates between multiple networks to improve performance over a single network.
- **Fischler and Elschlager (1973):** A scale checking and adjusting algorithm is proposed to automatically adjust the perspective scales during the tracking process.
- Later, methods based on statistical classifiers (e.g., Neural Networks, SVM, Adaboost, Bayes, etc.) were introduced.
- **Osuna et al. (1997):** Training a SVM is equivalent to solving a linearly constrained quadratic programming (QP) problem in a number of variables equal to the number of data points. This optimization problem is known to be challenging when the number of data points exceeds few thousands.

This initial successful family of object detectors, all of them based on statistical classifiers, set the ground for most of the following research in terms of training and evaluation procedures and classification techniques.

2.2 EXISTING SYSTEM

- **Faster RCNN:** Faster R-CNN has two networks: region proposal network (RPN) for generating region proposals and a network using these proposals to detect objects.

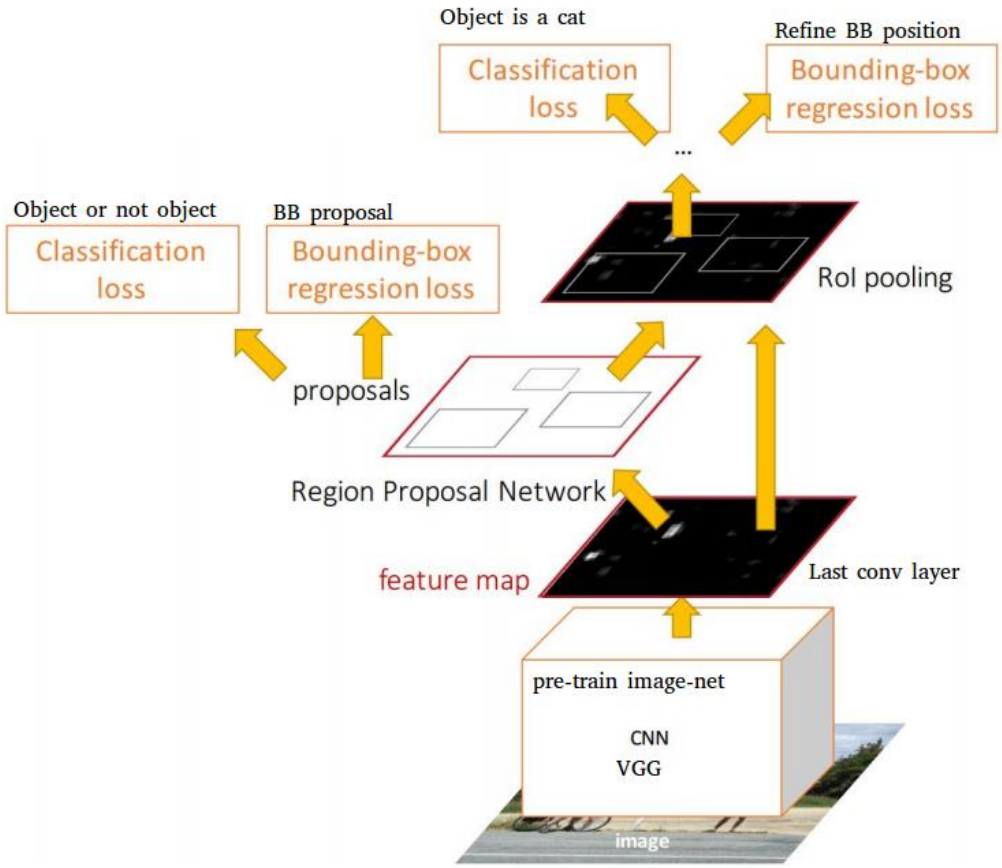

Fig 2.1: Architecture of Faster RCNN

- **Yolo V3:** Single Neural network is applied to the complete image. The network divides the image into regions and predicts bounding boxes and probabilities of each region.

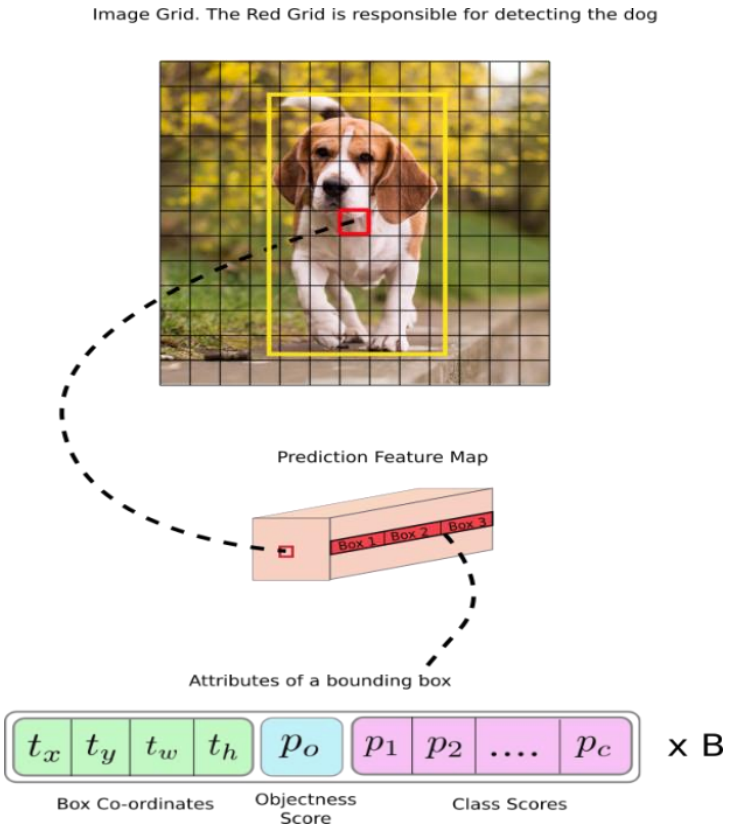

Fig 2.2: Architecture of YOLO V3

2.2.1 LIMITATIONS OF EXISTING SYSTEM

- Takes lot of computational power to train.
- Expensive computation for cropping multiple images
- Accuracy of the bounding box is poor

2.3 PROPOSED SYSTEM

SSD: Single Shot MultiBox Detector (by C. Szegedy et al.) was released at the end of November 2016 and reached new records in terms of performance and precision for object detection tasks, scoring over 74% mAP (*mean Average Precision*) at 59 frames per second on standard datasets such as PascalVOC and COCO. To better understand SSD, let's start by explaining where the name of this architecture comes from:

- **Single Shot:** this means that the tasks of object localization and classification are done in a *single forward pass* of the network
- **MultiBox:** this is the name of a technique for bounding box regression developed by Szegedy et al. (we will briefly cover it shortly)
- **Detector:** The network is an object detector that also classifies those detected objects.

2.3.1 ADVANTAGES OVER EXISTING SYSTEM

- It is clearly faster than Faster-RCNN and optimized for low end devices.
- Multiple images can be cropped at lost cost.
- Accuracy of the bounding box is decent in SSD.

CHAPTER 3

3. SOFTWARE RQUIREMENTS SPECIFICATIONS

3.1 User Requirements

- List of objects to be detected using the model
- Collection of images which include the objects to be detected
- Good object detection Algorithm which best suits the needs
- Good amount of resources to understand the underlying techniques used.

3.2 Software Requirements

Operating System	Windows 7 and later.
Coding Language	Python 3.5, XML
IDE	Pycharm Community Edition, Python Idle, Notepad ++, Anaconda
API	TensorFlow Object Detection API
Tools	TensorFlow Toolkit, TensorBoard, Protobu v3.4, LabellMg

3.3 Hardware Requirements

Computer System	Dell Inspiron 15 7000 Gaming
Processor	Processor Intel(R) Core(TM) i5-7300H CPU @ 2.50GHz, 2496 MHz, 4 Core(s), 4 Logical Processor(s)
RAM	4 GB(8GB Preferable)
Storage	NVMe 256 GB SSD
Mouse	Dell
Screen	15.6-inch FHD
Graphics Card	Nvidia GTX 1060 Max-Q architecture, Inte HD graphics-630

CHAPTER 4

4.SYSTEM ANALYSIS

4.1 ANALYSIS MODEL

In computer vision there are two main systems to consider when developing an object recognition system:

- Instance-level object recognition: This is the easiest to achieve as it is just about recognizing an instance of an object in a scene. It involves finding corresponding points between an object model in the database and that in a scene.
- Category-level object recognition: This involves classifying an object as belonging to a particular known category of objects. This is more difficult and requires special feature selection and learning algorithms.

Instance-level object recognition:

There are 4 main stages:

1. Feature detection
2. Feature matching
3. Feature integration
4. Model fitting and verification

Feature detection: What is a feature?

- A feature in computer vision is an easily localizable structure in an image, that is, a feature must be easy to find in a given image.
- It must have a well-defined pose i.e. position that is easy to recover from measurements done on the image.
- Repeatability is one important attribute of a feature and means that the same feature must be detectable even when an image/object undergoes different transformations such as projective distortion, scaling, rotation or translation.
- Borrowing from the primary visual cortex, feature detectors are heavily used in computer vision to speed up image analysis.
- One easily identifiable features are edges: Edges indicate an existing boundary between an object and the background, in terms of computer vision an edge is a region of high gradient magnitude, usually an abrupt change in gradient magnitude indicates edge presence

Feature matching: When a feature point is detected, a descriptor is extracted around that point to summarize image appeared in the immediate neighborhood.

- Descriptors are usually real-valued vectors that are normalized using L2-norm, but they can also be binary.
- They are placed in an indexing structure like the k-d tree or many others such as locality-sensitive hashing so that observed descriptors can be matched to their corresponding look-alike features in the database.

Feature integration:

- As previously seen multiple features are detected and matched per given scene.
- There is need for multiple features consistent with a particular object pose to be grouped into a singular perception.
- This can be achieved with a generalized hough transform approach were each feature votes for an object center, scale and orientation.
- The hough voting bins with enough votes i.e. 4 and above are considered as object instance hypotheses.

Model fitting and verification:

- The object instance hypotheses are analyzed by fitting a model to the observation.
- This is done by RANSAC - Random sample consensus algorithm.
- When fitting a homography motion model matrix 4-point correspondences are sufficient to give a good solution.
- Model verification looks at a detailed probabilistic analysis of inliers and outliers, to accept or reject a hypothesis.

4.2 Algorithm

When an input image is given and a set of ground truth labels, SSD does the following:

1) Pass the image through a series of convolutional layers, yielding several sets of feature maps at different scales (e.g. 10x10, then 6x6, then 3x3, etc.)
2) For each location in each of these feature maps, use a 3x3 convolutional filter to evaluate a small set of default bounding boxes. These default bounding boxes are essentially equivalent to Faster R-CNN's anchor boxes.
3) For each box, simultaneously predict

a) the bounding box offset
b) the class probabilities

4) During training, match the ground truth box with these predicted boxes based on IoU(Intersection over union). The best predicted box will be labeled a "positive," along with all other boxes that have an IoU with the truth >0.5.
5) Instead of using all the negative examples, sort the results using the highest confidence loss for each default box and pick the top ones so that the ratio between the negatives and positives is at most 3:1.

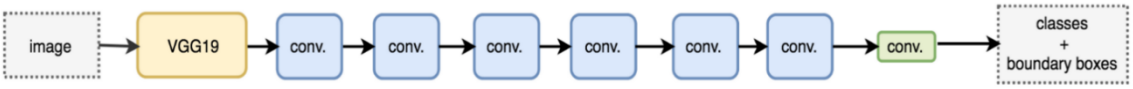

Fig 4.1 : Single shot prediction for both classification and location.

4.3 MODULE DESIGN AND ORGANIZATION

An object recognition system must have the following components to perform the task:

- Model Base
- Feature Detector
- Hypothesizer
- Hypothesis Verifier

The central issues that should be considered in designing an object recognition system are:

- **Object or model representation:** How should objects be represented in the model database?
  What are the important attributes or features of objects that must be captured in these models?
  For some objects, geometric descriptions may be available and may also be efficient.
  The representation of an object should capture all relevant information without any redundancies.
  While for another class one may have to rely on generic or functional features and should organize this information in a form that allows easy access by different components of the object recognition system.

- **Feature extraction**: Which features should be detected, and how can they be detected reliably?

Most features can be computed in two-dimensional images, but they are related to three-dimensional characteristics of objects.

- **Feature-model matching:** How can features in images be matched to models in the database?

  In most object recognition tasks, there are many features and numerous objects.
  An exhaustive matching approach will solve the recognition problem but may be too slow to be useful.
  Effectiveness of features and efficiency of a matching technique must be considered in developing a matching approach.

- **Hypotheses formation:** How can a set of likely objects based on the feature matching be selected, and how can probabilities be assigned to each possible object? The hypothesis formation step is basically a heuristic to reduce the size of the search space. This step uses knowledge of the application domain to assign some kind of probability or confidence measure to different objects in the domain. This measure reflects the likelihood of the presence of objects based on the detected features.

- **Object verification:** How can object models be used to select the most likely object from the set of probable objects in a given image? The presence of each likely object can be verified by using their models. One must examine each plausible hypothesis to verify the presence of the object or ignore it.

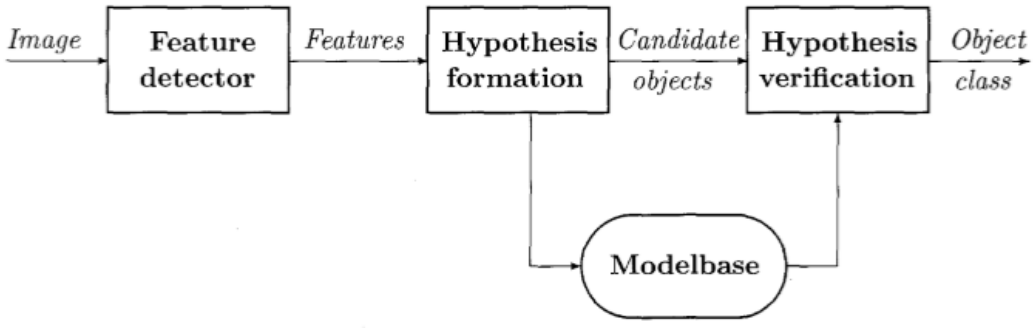

Fig 4.2 : Different components of an object recognition system are shown.

CHAPTER 5

5.SYSTEM DESIGN

5.1 SYSTEM ARCHITECTURE

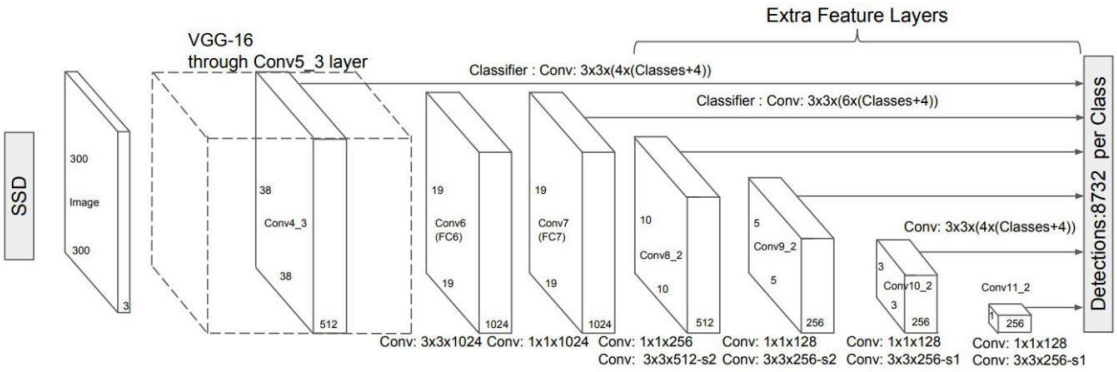

Fig 5.1 : Architecture of Single Shot MultiBox detector (input is 300x300x3)

5.2 UML Diagrams

A diagram is graphical representation of a set of elements; most often rendered as a connected graph of vertices a thing. Each UML diagram is designed to let developers and customers view a software system from a different perspective and in varying degrees of abstraction. UML diagrams commonly created in visual modeling tool include:

1. Class Diagram
2. Usecase Diagram
3. Sequence Diagram
4. Activity Diagram

Use Case Diagram :

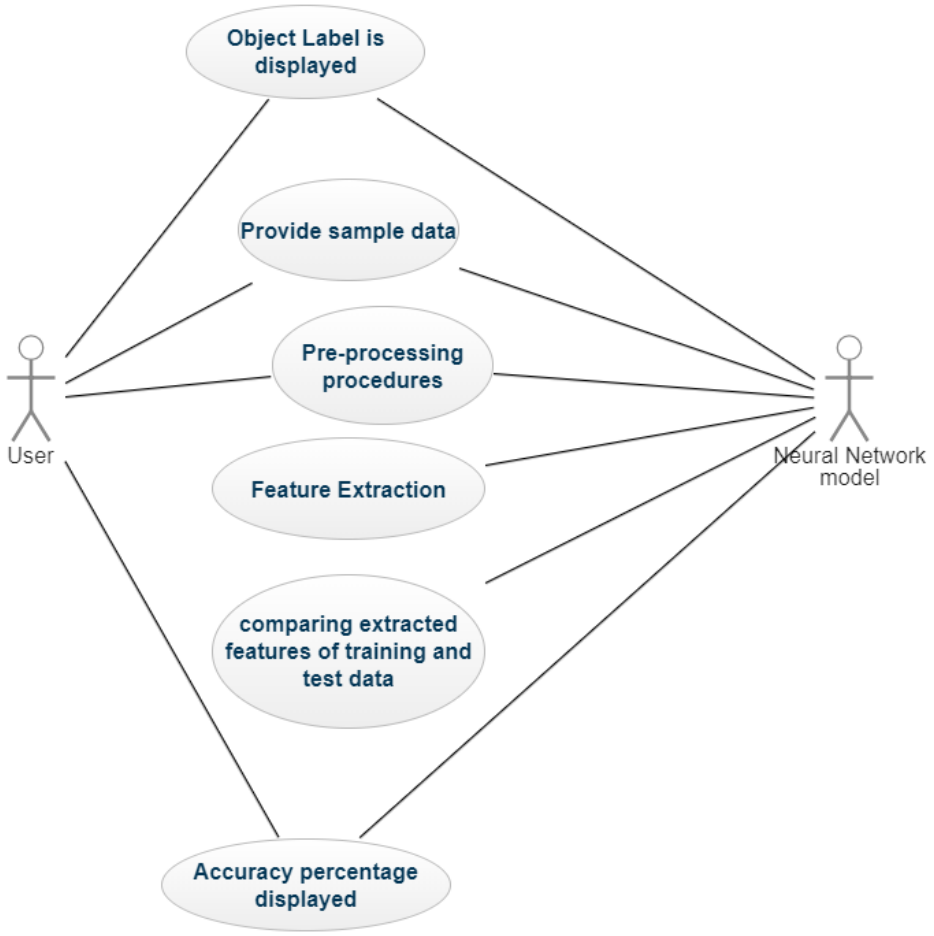

Fig 5.2: Use case diagram showing how a user interacts with the neural network

Class Diagram:

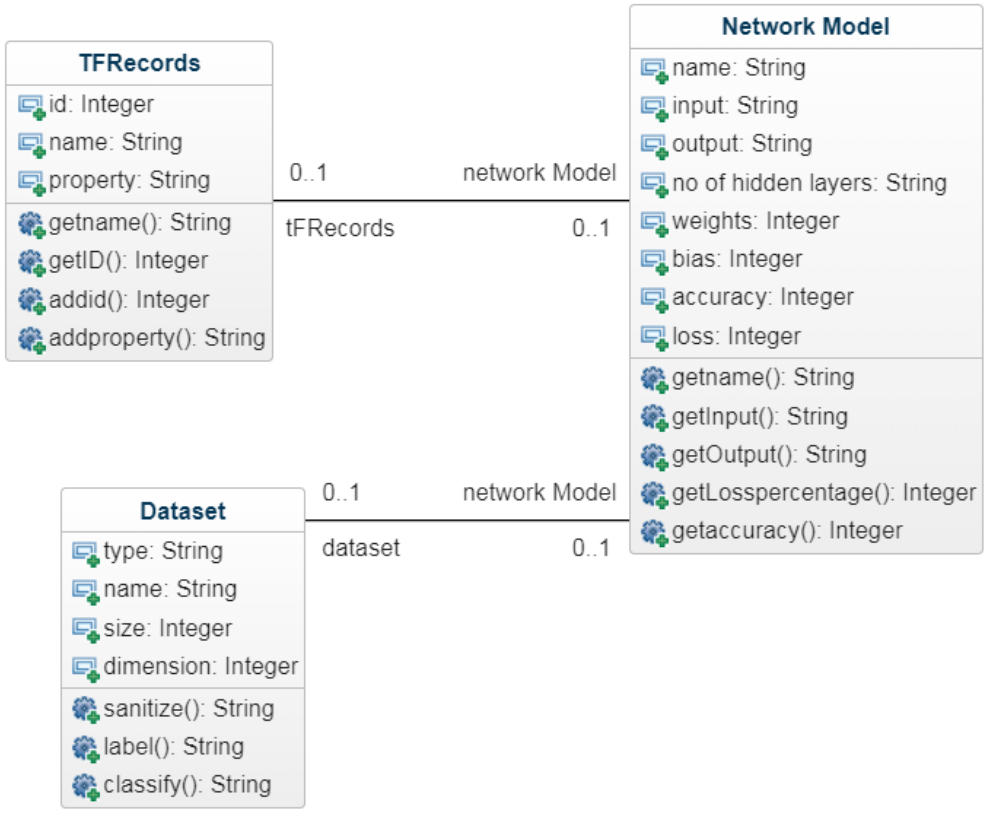

Fig 5.3 : Class diagram for the neural network

Sequence Diagram:

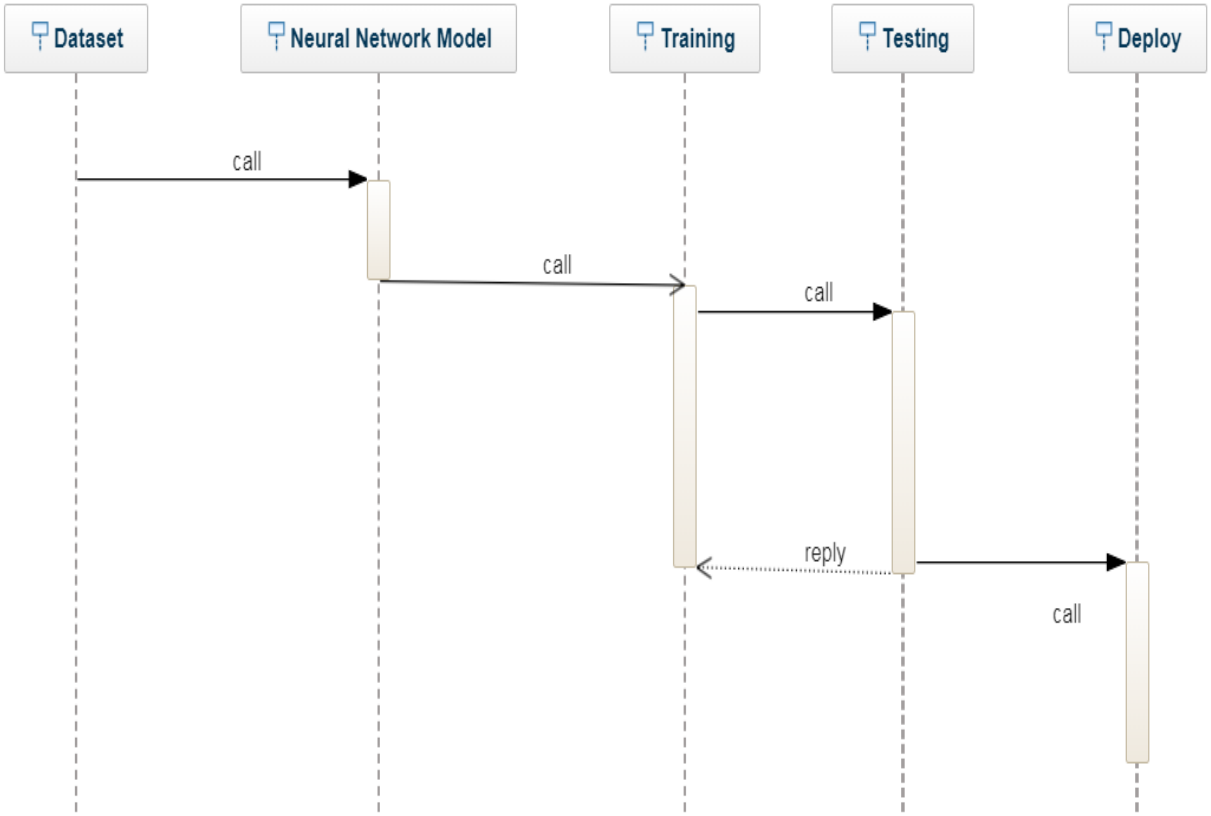

Fig 5.4: Sequence diagram for object detection

Activity Diagram:

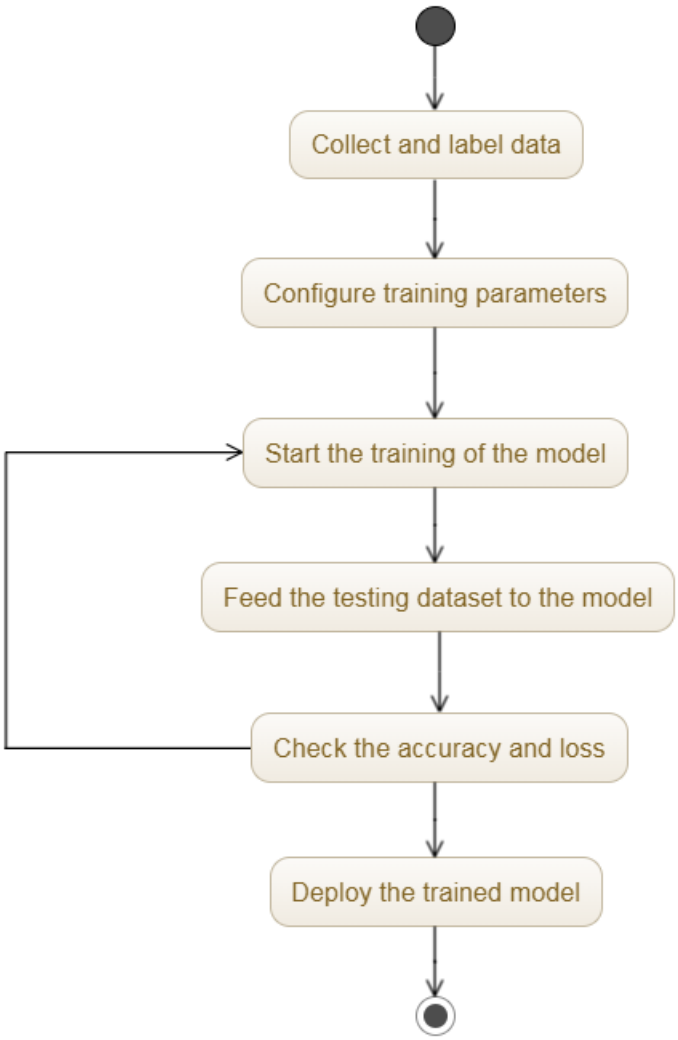

Fig 5.5 : Activity diagram for object detection

CHAPTER 6

6. IMPLEMENTATION & CODING

6.1 INTRODUCTION

- Humans can easily detect and identify objects present in an image.
- The human visual system is fast and accurate and can perform complex tasks like identifying multiple objects and detect obstacles with little conscious thought.
- With the availability of large amounts of data, faster GPUs, and better algorithms, we can now easily train computers to detect and classify multiple objects within an image with high accuracy.
- In the explanation of key functions, we will explore terms such as object detection, object localization, loss function for object detection and localization.

6.2 EXPLANATION OF KEY FUNCTIONS

Object Localization

- An image classification or image recognition model simply detect the probability of an object in an image.
- In contrast to this, object localization refers to identifying the location of an object in the image.
- An object localization algorithm will output the coordinates of the location of an object with respect to the image.
- In computer vision, the most popular way to localize an object in an image is to represent its location with the help of bounding boxes. Fig. 1 shows an example of a bounding box.

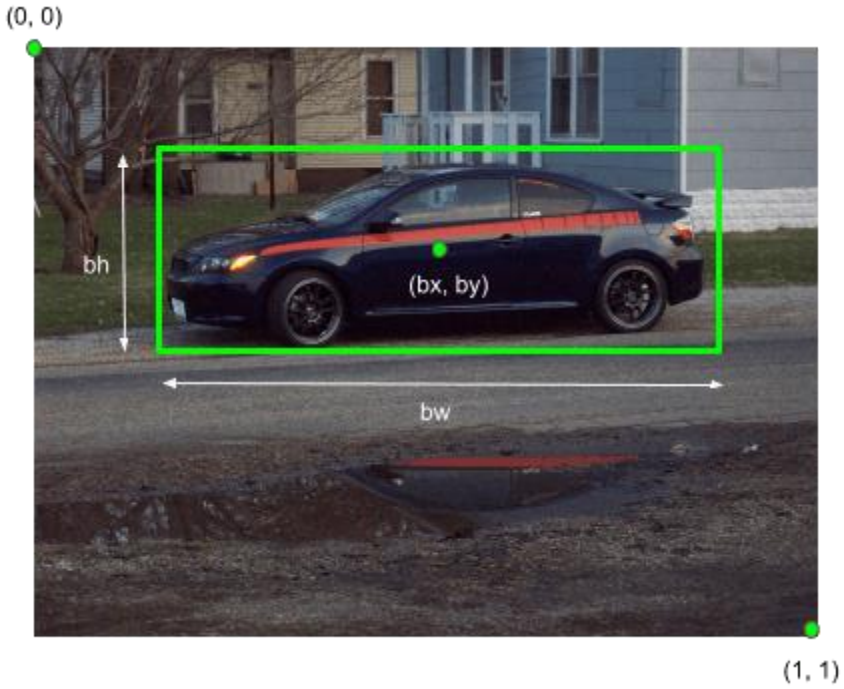

Fig 6.1 : Bounding box representation used for object localization

A bounding box can be initialized using the following parameters:

bx, by : coordinates of the center of the bounding box

bw : width of the bounding box w.r.t the image width

bh : height of the bounding box w.r.t the image height

Loss Function

- At its core, a loss function is incredibly simple: it's a method of evaluating how well your algorithm models your dataset.
- If your predictions are totally off, your loss function will output a higher number.
- If they're pretty good, it'll output a lower one.
- As you change pieces of your algorithm to try and improve your model, your loss function will tell you if you're getting anywhere.
- The loss function is the bread and butter of modern Machine Learning
- It takes your algorithm from theoretical to practical and transforms neural networks from glorified matrix multiplication into Deep Learning.

Object Detection

- Object detection is the process of finding instances of real-world objects such as faces, bicycles, and buildings in images or videos.
- Object detection algorithms typically use extracted features and learning algorithms to recognize instances of an object category.
- It is commonly used in applications such as image retrieval, security, surveillance, and advanced driver assistance systems (ADAS).

You can detect objects using a variety of models, including:

- Deep learning object detection
- Feature-based object detection

6.3 METHODS OF IMPLEMENTATION

Object Detection can be done via multiple ways:

- Feature-Based Object Detection
- Viola Jones Object Detection
- SVM Classifications with HOG Features
- Deep Learning Object Detection

Deep Learning Object Detection

Steps followed were

1) Install TensorFlow-GPU 1.5
2) Set up TensorFlow Directory and Anaconda Virtual Environment
3) Download TensorFlow Object Detection API.
4) Download the SSD-Mobilenet FPN.

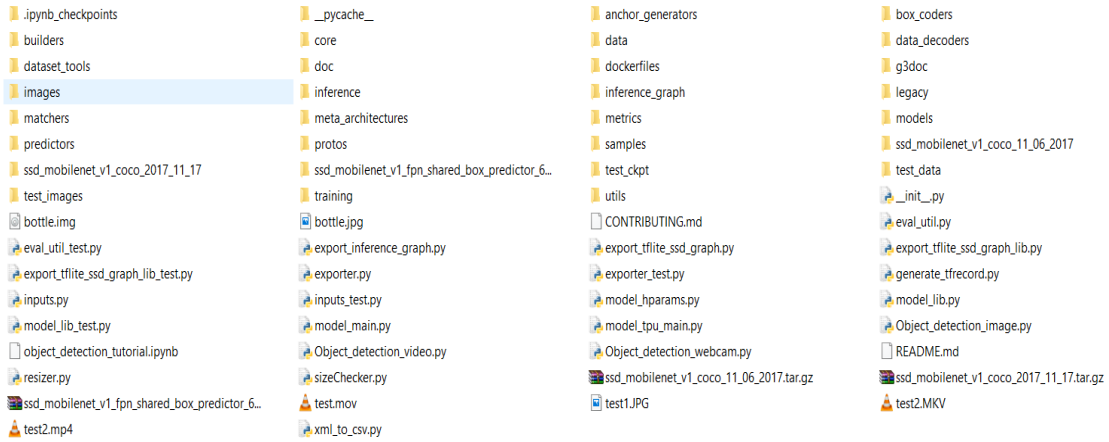

Fig 6.2 : downloaded files in the TensorFlow directory

5) Set up new Anaconda virtual environment

```
C:\> conda create -n tensorflow1 pip python=3.5
```

Then, activate the environment by issuing:

```
C:\> activate tensorflow1
```

Install tensorflow-gpu in this environment by issuing:

```
(tensorflow1) C:\> pip install --ignore-installed --upgrade tensorflow-gpu
```

Install the other necessary packages by issuing the following commands:

```
(tensorflow1) C:\> conda install -c anaconda protobuf
(tensorflow1) C:\> pip install pillow
(tensorflow1) C:\> pip install lxml
(tensorflow1) C:\> pip install Cython
(tensorflow1) C:\> pip install jupyter
(tensorflow1) C:\> pip install matplotlib
(tensorflow1) C:\> pip install pandas
(tensorflow1) C:\> pip install opencv-python
```

Fig 6.3 : downloading the libraries

6) Configure PYTHONPATH environment variable

7) Test TensorFlow setup to verify it works

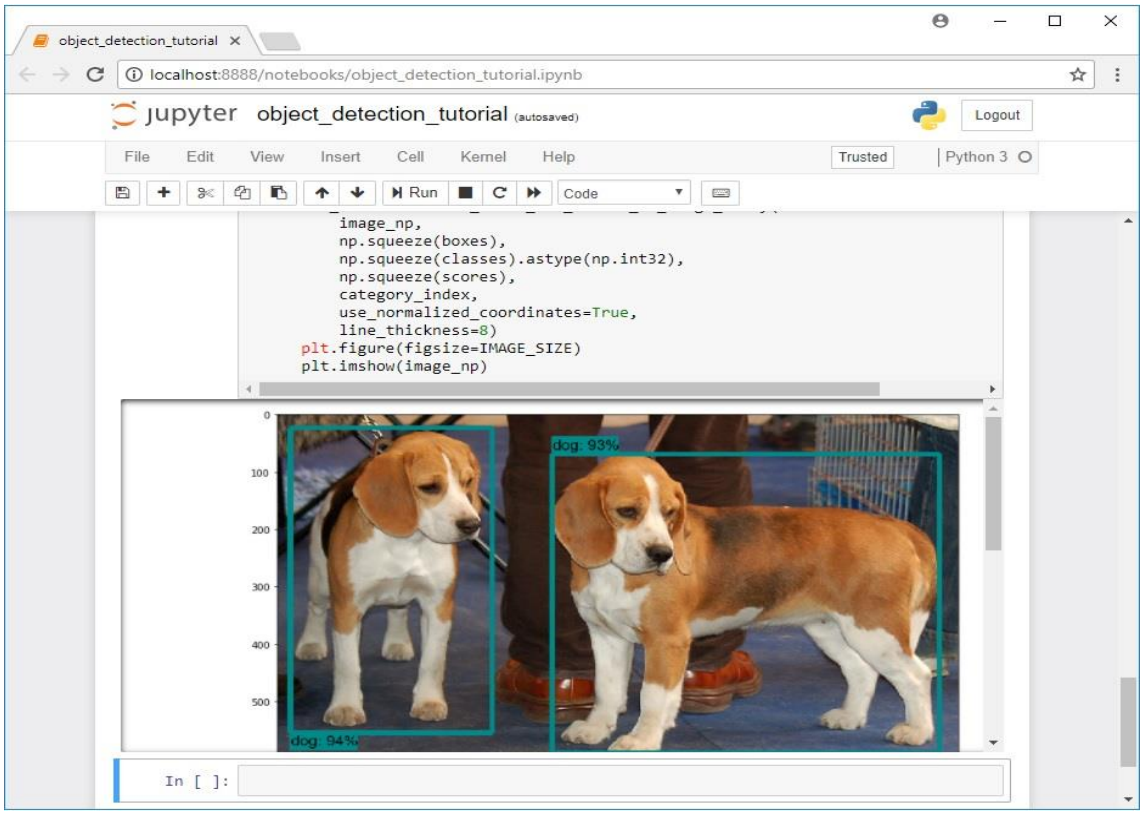

Fig 6.4 : Testing the object detection model

8) Gather and Label Pictures

Fig 6.5 : Labelling the images using LabellMg

9) Generate Training Data

10) Create Label Map and Configure Training

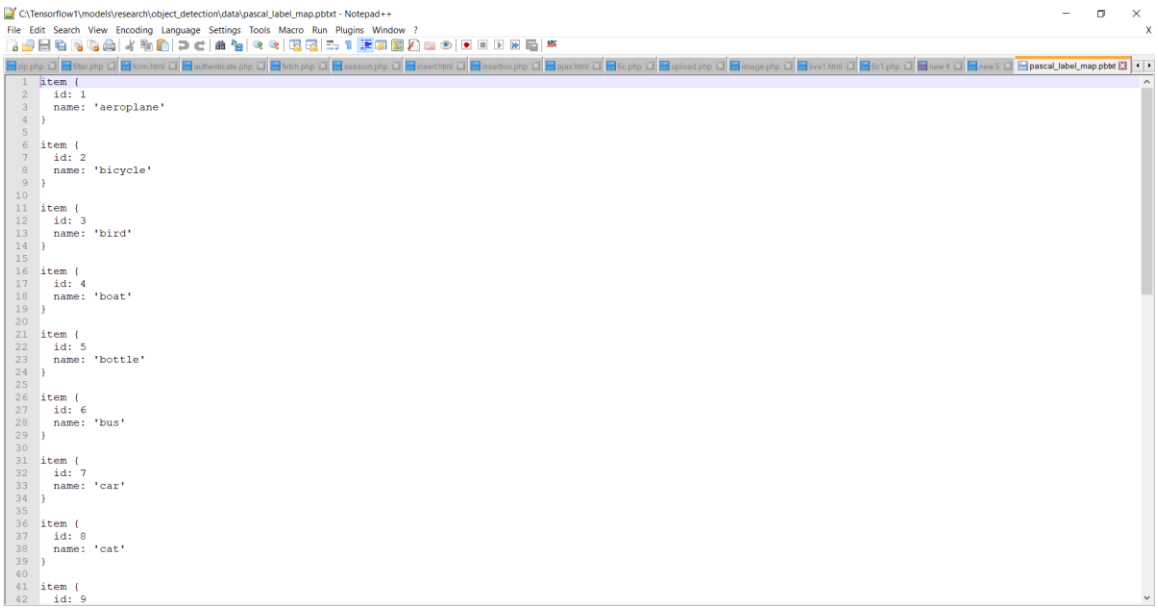

Fig 6.6 : Creating the label map used for training

11) Configure training

12) Run the Training

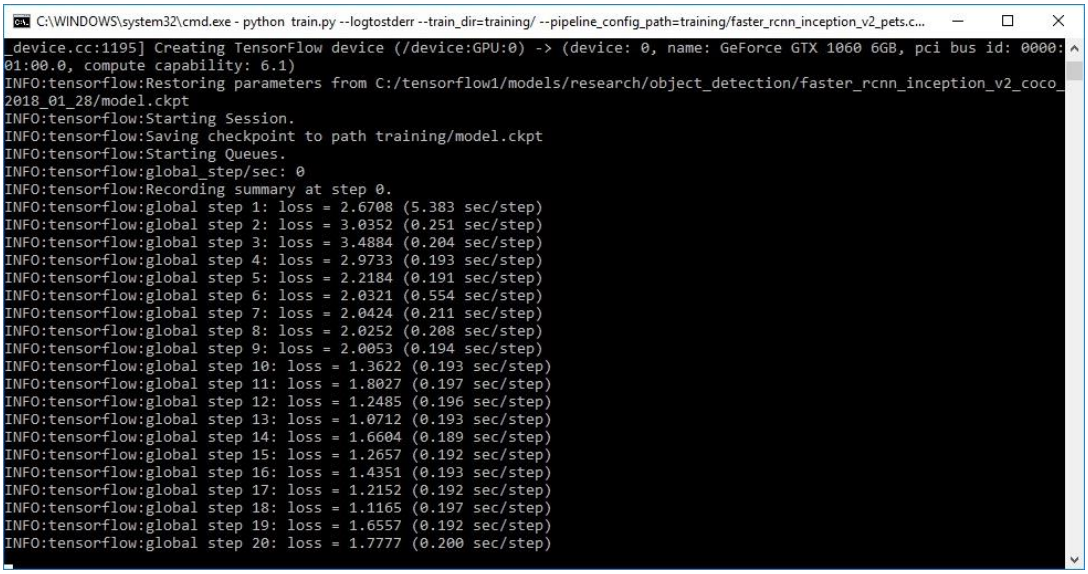

Fig 6.7 : The model being trained for the training image dataset

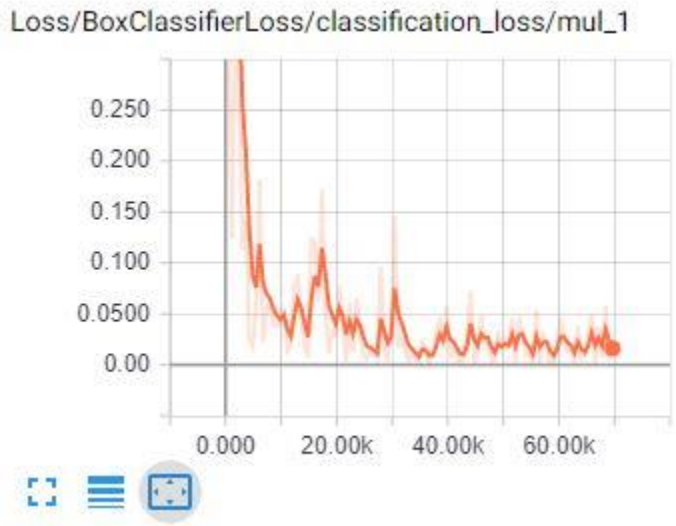

Fig 6.8 : Loss Graph generated during the training of the model

6.4 PYTHON SCRIPT

```
# -*- coding: utf-8 -*-

#!/usr/bin/env python
# coding: utf-8

# # Object Detection

# # Imports

# In[1]:


import numpy as np
import os
import sys
import tensorflow as tf

from distutils.version import StrictVersion
from IPython import get_ipython

# This is needed since the notebook is stored in the
object_detection folder.
sys.path.append("..")
from object_detection.utils import ops as utils_ops

if StrictVersion(tf.__version__) < StrictVersion('1.12.0'):
  raise ImportError('Please upgrade your TensorFlow installation to
v1.12.*.')


# ## Env setup

# In[2]:

# This is needed to display the images.
get_ipython().run_line_magic('matplotlib', 'inline')


# ## Object detection imports
# Here are the imports from the object detection module.

# In[3]:


from utils import label_map_util

from utils import visualization_utils as vis_util
```

```
# # Model preparation

# In[4]:


# What model to download.
MODEL_NAME = 'ssd_mobilenet_v1_coco_2017_11_17'

# Path to frozen detection graph. This is the actual model that is
used for the object detection.
PATH_TO_FROZEN_GRAPH = 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('data', 'mscoco_label_map.pbtxt')




# In[5]:


detection_graph = tf.Graph()
with detection_graph.as_default():
  od_graph_def = tf.GraphDef()
  with tf.gfile.GFile(PATH_TO_FROZEN_GRAPH, '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 maps map indices to category names, so that when our
convolution network predicts `5`, we know that this corresponds to
`airplane`.  Here we use internal utility functions, but anything
that returns a dictionary mapping integers to appropriate string
labels would be fine

# In[6]:


category_index =
label_map_util.create_category_index_from_labelmap(PATH_TO_LABELS,
use_display_name=True)




# In[7]:
```

```
def run_inference_for_single_image(image, graph):
  with graph.as_default():
    with tf.Session() as sess:
      # Get handles to input and output tensors
      ops = tf.get_default_graph().get_operations()
      all_tensor_names = {output.name for op in ops for output in
op.outputs}
      tensor_dict = {}
      for key in [
          'num_detections', 'detection_boxes', 'detection_scores',
          'detection_classes', 'detection_masks'
      ]:
        tensor_name = key + ':0'
        if tensor_name in all_tensor_names:
          tensor_dict[key] =
tf.get_default_graph().get_tensor_by_name(
              tensor_name)
      if 'detection_masks' in tensor_dict:
        # The following processing is only for single image
        detection_boxes = tf.squeeze(tensor_dict['detection_boxes'],
[0])
        detection_masks = tf.squeeze(tensor_dict['detection_masks'],
[0])
        # Reframe is required to translate mask from box coordinates
to image coordinates and fit the image size.
        real_num_detection =
tf.cast(tensor_dict['num_detections'][0], tf.int32)
        detection_boxes = tf.slice(detection_boxes, [0, 0],
[real_num_detection, -1])
        detection_masks = tf.slice(detection_masks, [0, 0, 0],
[real_num_detection, -1, -1])
        detection_masks_reframed =
utils_ops.reframe_box_masks_to_image_masks(
            detection_masks, detection_boxes, image.shape[0],
image.shape[1])
        detection_masks_reframed = tf.cast(
            tf.greater(detection_masks_reframed, 0.5), tf.uint8)
        # Follow the convention by adding back the batch dimension
        tensor_dict['detection_masks'] = tf.expand_dims(
            detection_masks_reframed, 0)
      image_tensor =
tf.get_default_graph().get_tensor_by_name('image_tensor:0')

      # Run inference
      output_dict = sess.run(tensor_dict,
                             feed_dict={image_tensor:
np.expand_dims(image, 0)})

      # all outputs are float32 numpy arrays, so convert types as
appropriate
      output_dict['num_detections'] =
int(output_dict['num_detections'][0])
      output_dict['detection_classes'] = output_dict[
          'detection_classes'][0].astype(np.uint8)
```

```
      output_dict['detection_boxes'] =
output_dict['detection_boxes'][0]
      output_dict['detection_scores'] =
output_dict['detection_scores'][0]
      if 'detection_masks' in output_dict:
        output_dict['detection_masks'] =
output_dict['detection_masks'][0]
  return output_dict


# In[8]:

import cv2
cap = cv2.VideoCapture(0)

while(True):


  # Expand dimensions since the model expects images to have shape:
[1, None, None, 3]
  ret,image_np=cap.read()
  image_np_expanded = np.expand_dims(image_np, axis=0)
  # Actual detection.
  output_dict = run_inference_for_single_image(image_np,
detection_graph)
  # Visualization of the results of a detection.
  vis_util.visualize_boxes_and_labels_on_image_array(
      image_np,
      output_dict['detection_boxes'],
      output_dict['detection_classes'],
      output_dict['detection_scores'],
      category_index,
      instance_masks=output_dict.get('detection_masks'),
      use_normalized_coordinates=True,
      line_thickness=8)


  cv2.imshow('image',image_np)
  if cv2.waitKey(25) & 0xFF==ord('q'):
      break
cv2.destroyAllWindows()
cap.release()


# In[ ]:




# In[ ]:
```

CHAPTER 7

7.TESTING

7.1 TEST CASE

- Once we have trained the model using the labelled images which were in the training dataset the next step is to test the model.
- Testing is done on the 25% of images which weren't labelled, and which were in the testing dataset.
- Now we first export the trained model as frozen graph.
- This is the file which contains all the training parameters that were used to train the model and optimize its accuracy.
- These get stored along with other metadata about the model and this file should be placed in environments where we want to deploy the model and test the object detection functionality.

- The object detection classifier is all ready to go! We have written Python scripts to test it out on an image, video, or webcam feed.

- To test your object detector, move a picture of the object or objects into the \object_detection folder, and change the IMAGE_NAME variable in the Object_detection_image.py to match the file name of the picture. Alternatively, you can use a video of the objects (using Object_detection_video.py), or just plug in a USB webcam and point it at the objects (using Object_detection_webcam.py).

- To run any of the scripts, type "idle" in the Anaconda Command Prompt (with the "tensorflow1" virtual environment activated) and press ENTER. This will open IDLE, and from there, you can open any of the scripts and run them.

Fig 7.1: Objects being detected by the model

CHAPTER 8

8. SCREENSHOTS

8.1 OUTPUT SCREEN

The output design revolves around how the model classifies the objects, how it detects them. It's important to consider that the object it is detecting is displayed to the user, so that user can know which object is detected.

To do this we use the bounding box technique where we represent the detected object inside a box with label of the object on the top right-hand side. This clear specifies the object being detected.

In order to provide more information, the bounding box also contains a percentage representing the accuracy of the detection.

With this information the user has all the data about the model and he can use this to tweak the model in order to improve the model.

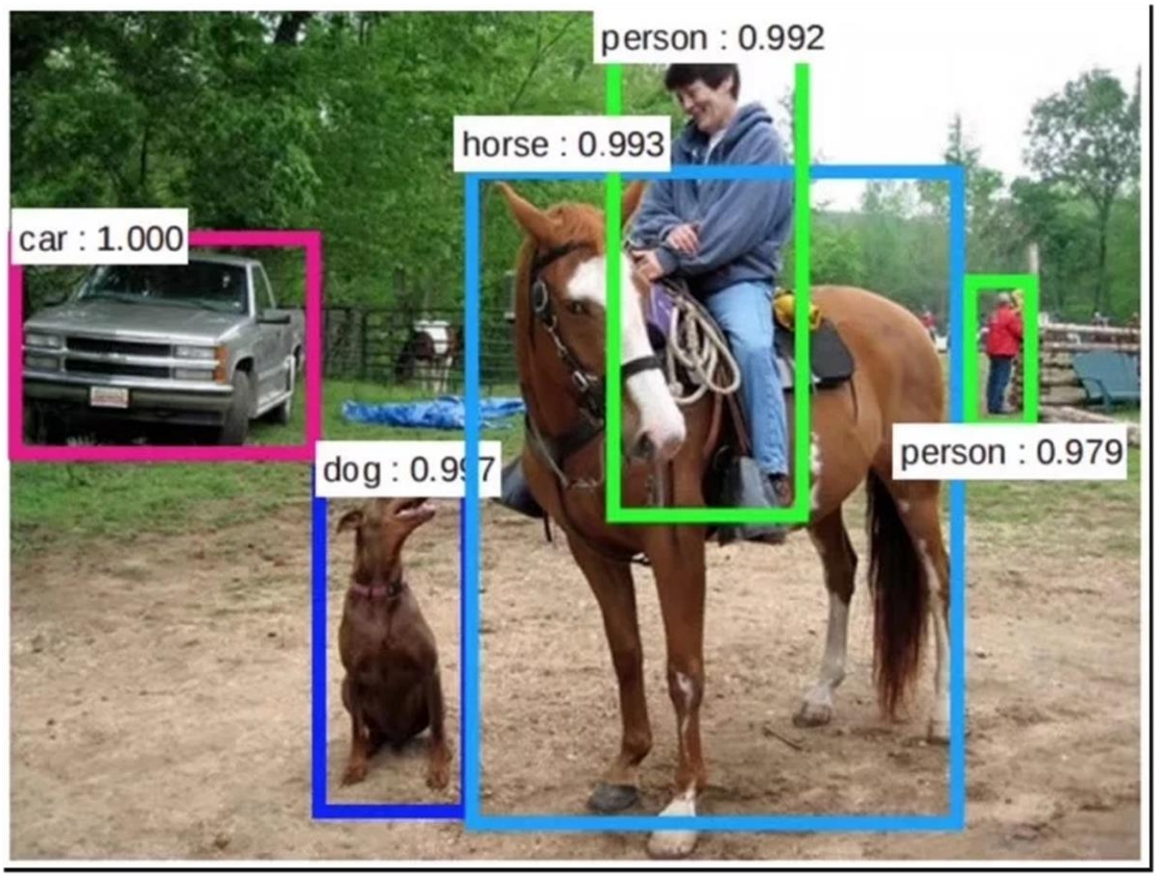

Fig 8.1 : Output displaying the detection of objects by the model

CHAPTER 9

9. CONCLUSION

We developed an object detector model using deep learning neural networks. This model can be used to detect the objects in images, videos or even webcam feeds. The accuracy of this model is more than 75%. The training time for this model was around 5-6 hrs. Many images belonging to different areas consisting of different objects were labelled and then given to the model for training. This model can easily be optimized and deployed on low computing devices such as mobile phones, tablets etc. This model uses artificial neural networks to extract the feature information, then perform feature mapping, the SSD MobileNet FPN performs localization and classification in the same layer of neural network and hence is fast compared to other models.

From here the object detection model can further be trained on even larger sets of data and can include more objects. This model can then be used in other applications such as :

- Autonomous vehicles
- Surveillance systems
- Develop advanced robots
- Retrieval (search engines, management)
- Facial recognition attendance system
- Industry Quality check

CHAPTER 10

10. BIBLIOGRAPHY

- https://www.frontiersin.org/articles/10.3389/frobt.2015.00029/full
- https://ieeexplore.ieee.org/document/5255236
- https://www.researchgate.net/publication/3192800_Example_Based_Learning_for_View-Based_Human_Face_Detection
- https://www.edureka.co/blog/tensorflow-object-detection-tutorial/
- https://github.com/EdjeElectronics/TensorFlow-Object-Detection-API-Tutorial-Train-Multiple-Objects-Windows-10
- https://medium.com/@jonathan_hui/what-do-we-learn-from-single-shot-object-detectors-ssd-yolo-fpn-focal-loss-3888677c5f4d