Visvesvaraya Technological University
Belgaum, Karnataka- 590014

A Project Report On

"Live Object Detection in Videos Using CNN"

Submitted in the partial fulfilment of the requirements for the award of the Degree of

BACHELOR OF ENGINEERING

In

INFORMATION SCIENCE AND ENGINEERING

ACCREDITED BY NBA

Submitted by

Abhishek Goyal (1DS15IS004) **Ayush Agarwal (1DS15IS016)**

G Satish Reddy (1DS15IS027) **K R Krishna Vamshi (1DS15IS035)**

Under the Guidance of

Mrs. Sharon Christa

Asst. Professor, Dept. of ISE

2018-2019

DEPARTMENT OF INFORMATION SCIENCE AND ENGINEERING

DAYANANDA SAGAR COLLEGE OF ENGINEERING

SHAVIGE MALLESHWARA HILLS, KUMARASWAMY LAYOUT, BANGALORE-78

DAYANANDA SAGAR COLLEGE OF ENGINEERING

Shavige Malleshwara Hills, Kumaraswamy Layout

Bangalore-560078

Department of Information Science and Engineering

ACCREDITED BY NBA

2018-2019

Certificate

This is to certify that the Project Work entitled —**"Live Object Detection in Videos Using CNN"** is a bonafide work carried out by **Abhishek Goyal** (1DS15IS004), **Ayush Agarwal** (1DS15IS016), **G Satish Reddy** (1DS15IS027), and **K R Krishna Vamshi** (1DS15IS035) in partial fulfilment for the 8th semester of Bachelor of Engineering in Information Science & Engineering of the Visvesvaraya Technological University, Belgaum during the year 2018-2019. The Project Report has been approved as it satisfies the academics prescribed for the Bachelor of Engineering degree.

_____	_____	_____
Signature of Guide [Mrs. Sharon Christa]	Signature of HOD [Dr. K.N. Rama Mohan Babu]	Signature of Principal [Dr. C.P.S Prakash]

Name of the Examiners	Signature with Date
1._____	_____
2._____	_____

ACKNOWLEDGEMENT

It is great pleasure for us to acknowledge the assistance and support of a large number of individuals who have been responsible for the successful completion of this project.

We take this opportunity to express our sincere gratitude to **Dayananda Sagar College of Engineering** for having provided us with a great opportunity to pursue our Bachelor Degree in this institution.

In particular we would like to thank **Dr. C. P. S Prakash,** Principal, Dayananda Sagar College of Engineering for his constant encouragement and advice.

Special thanks to **Dr. K.N. Rama Mohan Babu,** HOD, Department of Information Science & Engineering, Dayananda Sagar College of Engineering for his motivation and invaluable support well through the development of this project.

We are highly indebted to our internal guide **Mrs. Sharon Christa,** Asst. Professor, Department of Information Science & Engineering, Dayananda Sagar College of Engineering for his constant support and guidance. He has been a great source of inspiration throughout the course of this project.

Finally, we gratefully acknowledge the support of our families during the completion of the project.

Abhishek Goyal (1DS15IS004)

Ayush Agarwal (1DS15IS016)

G Satish Reddy (1DS15IS027)

K R Krishna Vamshi (1DS15IS035)

ABSTRACT

The purpose of this project is to develop a computer vision model using deep learning technique that can detect objects in a live video. Recognizing objects in a video is a difficult task and the difficulty increases if the inference is expected in the real-time. The project begins with analyzing advantages and disadvantages of existing object detection architectures like single shot detector, R-CNN etc. on the self-prepared dataset. The goal is to come up with an optimized object detection architecture using advantages of existing architectures. The developed architecture may be used for various applications in real-time with better accuracy and lesser time taken for generating the inference.

The superiority of the architecture will be demonstrated by developing a computer vision model that recognizes objects belonging to different classes. The model will be deployed to detect objects in a live streaming video. For faster inference, the full capabilities of the hardware will be utilized. To accomplish this, concepts such as multithreading will be used.

CONTENTS

1. INTRODUCTION…….. 1

1.1. Overview 1

1.2. Problem Statement 2

1.3. Objectives 2

1.4. Motivation 2

2. LITERATURE SURVEY……... 3

3. REQUIREMENTS…… 5

3.1 Functional Requirements 5

3.2 Non Functional Requirements 5

3.3 Software Requirements 5

3.4 Hardware Requirements 5

4. SYSTEM ANALYSIS & DESIGN..……..6

4.1 Analysis 6

4.2 System Design 6

4.2.1 System Architecture Diagram 6

4.2.1.1 XML Dataset 8

4.2.1.2. Data Flow Diagram 9

4.2.1.3. Flowchart 11

4.2.1.4 Use Case Diagram 12

4.2.1.5 Sequence Diagram 12

5. IMPLEMENTATION…… 14

5.1 Introduction 14

5.2 Overview of System Implementation 14

5.2.1 Usability Aspect 15

5.2.1.1 The project is implemented using PYTHON 15

5.2.2 Technical Aspect 15

5.2.2.1 The project is implemented by using PYTHON from Jupyter Notebook 15

5.3 Implementation Support 17

5.3.1 Installation of Jupyter Notebook 17

6. PSEUDO CODE…… … 18

6.1 model_faster_rcnn.py 19

6.2 model_ssd_mobilenet.py 22

7. TESTING …….. 25

7.1: Test Cases 25

8. RESULTS…… 27

8.1: Results using Faster-RCNN 28

8.2: Results using SSD 29

9. CONCLUSION & FUTURE SCOPE……... ... …… 30

10. REFERENCES …… 31

LIST OF FIGURES

Fig. No.	Topic	Page No.
4.2.1	System Architecture Diagram	7
4.2.1.2	Dataflow diagram	10
4.2.1.3	Flowchart	11
4.2.1.4	Use Case Diagram	12
4.2.1.5	Sequence Diagram	13
8.1.1	Faster R-CNN in outdoor environment	28
8.1.2	Faster R-CNN in indoor environment	28
8.2.1	SSD Mobilenet in outdoor environment	29
8.2.2	SSD Mobilenet in indoor environment	29

LIST OF TABLES

Table No.	Topic	Page No.
7.1	Test Cases for model_rcnn.py	25
7.2	Test Cases for model_ssd_mobilenet.py	26

CHAPTER 1

INTRODUCTION

1.1 Overview:

Looking for lost keys in a messy and untidy house is time consuming. It happens to all of us and till date remains an incredibly irritating experience. But what if a simple computer algorithm could locate our keys in a matter of seconds?

The power of object detection algorithms comes into picture here. While this was a simple example, the applications of object detection span many industries, from vehicle detection in cities to surveillance. In short, there are powerful deep learning algorithms.

Our brain instantly recognizes the objects contained when we're shown an image. A lot of time is taken and huge amount of training data is required for a machine to identify these objects. But with the recent advances deep learning and hardware aspects, the computer vision field has become more intuitive.

A Convolutional Neural Network is a deep learning algorithm which can take in an input image, assign importance (biases and learnable weights) to various objects in the image and be able to differentiate one from the other. The pre-processing required in a convolutional network is much lower as compared to other classification algorithms. While in primitive methods filters are manually designed, with enough training, convolutional networks have the ability to learn the filters and characteristics.

Given that there are many techniques for object detection which have been developed, this project aims at reviewing and researching the efficiency of the major techniques used for live object detection in videos using Convolutional Neural Networks.

1.2 Problem Statement

For developing any object detection application, the main challenge is to select an object detection architecture that achieves the right speed-memory-accuracy balance for a given application and platform. There is a huge trade-off between accuracy, speed and memory usage in existing convolutional object detection systems.

1.3 Objectives

1. To prepare an image dataset for certain classes of objects and annotate them with the ground truth labels.
2. Systematic analysis of advantages and disadvantages of existing object detection architectures developed using CNN.
3. Developing an optimized CNN architecture and testing its speed v/s accuracy trade-off with the prepared dataset using Transfer Learning approach.
4. Deployment of the most efficient model for live detection of objects in videos using CNN.

1.4 Motivation

Machine Learning is going to shape our future more powerfully than any other innovation this century. Anyone who does not understand it will soon find themselves feeling left behind, waking up in a world full of technology that feels more and more like magic. The development of various techniques to identify objects has confused people on which architecture is best for a given situation. This project reviews two state-of-the-arts object detection models. The analysis criteria are the accuracy of detection, time taken and storage used.

CHAPTER 2

LITERATURE SURVEY

Various Books and information materials from the web regarding Machine learning and Convolutional Neural Networks have been studied through in order to achieve the required information concern to this project. Among them, following are the key points extracted through:

i) The **Research paper** on **Faster R-CNN** by **Shaoqing Ren, Kaiming He, Ross Girshick and Jian Sun** gave us a brief knowledge of Real time object detection with Region Proposal Networks and helped us to choose the appropriate approach.

This paper introduces a Region Proposal Network (RPN) that shares full-image convolutional features with the detection network, thus enabling nearly cost-free region proposals. An RPN is a fully-convolutional network that simultaneously predicts object bounds and objectness scores at each position. RPNs are trained end-to-end to generate high-quality region proposals, which are used by Fast R-CNN for detection.

ii) **The Research paper "R-FCN: Object Detection via Region Based Fully Convolutional Networks** by **Jifeng Dai, Yi Li, Kaiming He and Jian Sun** gave us a brief knowledge of region based fully convolutional networks for accurate and efficient object detection.

This paper presents region-based, fully convolutional networks for accurate and efficient object detection. Position-sensitive score maps are proposed to address a dilemma between translation-invariance in image classification and translation-variance in object detection. This method can thus naturally adopt fully convolutional image classifier backbones, such as the latest Residual Networks (ResNets), for object detection.

iii) The **Research paper "Scalable High Quality Object Detection"** by Christian Szegedy**, Scott Reed, Dumitru Erhan, Dragomir Anguelov and Sergey Ioffe** gave us an introductory knowledge on learning-based proposal methods and their effectiveness.

The fundamental idea is to train a convolutional network that outputs the coordinates of the object bounding boxes directly. However, this is just half of the story, since we would also like to rank the proposals by their likelihood of being an accurate bounding box for an object of interest.

iv) **"SSD: Single Shot Multibox Detector"** by Wei Liu, Dragomir Anguelov and Alexander C Berg provided us information on the SSD method for detecting objects in images using a single deep neural network.

SSD discretizes the output space of bounding boxes into a set of default boxes over different aspect ratios and scales per feature map location. At prediction time, the network generates scores for the presence of each object category in each default box and produces adjustments to the box to better match the object shape.

v) From the arXiv **paper "You Only Look Once: Unified, Real-Time Object Detection**" by **Joseph Redmon, Santosh Divvala** and Ali Farhadi, we learnt about new approach to object detection.

In Yolo, we frame object detection as a regression problem to spatially separated bounding boxes and associated class probabilities. A single neural network predicts bounding boxes and class probabilities directly from full images in one evaluation. Since the whole detection pipeline is a single network, it can be optimized end-to-end directly on detection performance.

vi) The https://medium.com/ website provided us the required real time implementation knowledge in order to code the model.

vii) The https://towardsdatascience.com/ websites helped us understand many concepts related to machine learning that helped us to choose appropriate algorithms for the work.

CHAPTER 3

REQUIREMENTS

3.1 Functional Requirements

- The model should be able to recognise objects which it is trained on.

3.2 Non Functional Requirements

- The accuracy of the predicted value must be precise.
- The model should never fail in the middle of operation.
- The model should work consistently across various platforms.

3.2.1.1 Software Requirements

- OS Version: Windows 7(64 bit) or newer versions.
- Coding Language: Python 3.6
- Platform: Jupyter Notebook

3.2.1.2 Hardware Requirements

- Processor: i5 or i7 Intel Processor
- Primary Storage: 8 GB RAM or above (Recommended 16 GB)
- Secondary Storage: Any standard HDD or SDD
- Web-Cam

CHAPTER 4

SYSTEM ANALYSIS & DESIGN

4.1 Analysis

The procedure of breaking a difficult topic or substance into small parts to gain a better knowledge of the problem is known as analysis. Analysts in the field of engineering look at the structures and requirements, mechanisms, and systems dimensions. Analysis is an activity of exploration. The project lifecycle begins in the analysis phase.

4.2 System Design

The definition of the architecture of a system, components, modules, interfaces, and data for a system to fulfil specified requirements is system design. Systems design could be seen as the application of systems theory to product development.

The design phase produces the overall design of the software. The goal of design phase is to figure out the modules that should be in the system to fulfil all the system requirements in an efficient manner. It will contain the details of all these modules, their working with other modules and the desired output from each module. The output of the design process is a description of the software architecture.

4.2.1 System Architecture Diagram

The definition of the structure and operation and more views of a system is known as system architecture. A formal description and rendition of a system, organized in a way so that it supports reasoning about the working and behaviors of the system is called architecture description.

System architecture comprises of system components that work together and implement the overall system.

The below figure shows a general block diagram describing the activities performed by this project.

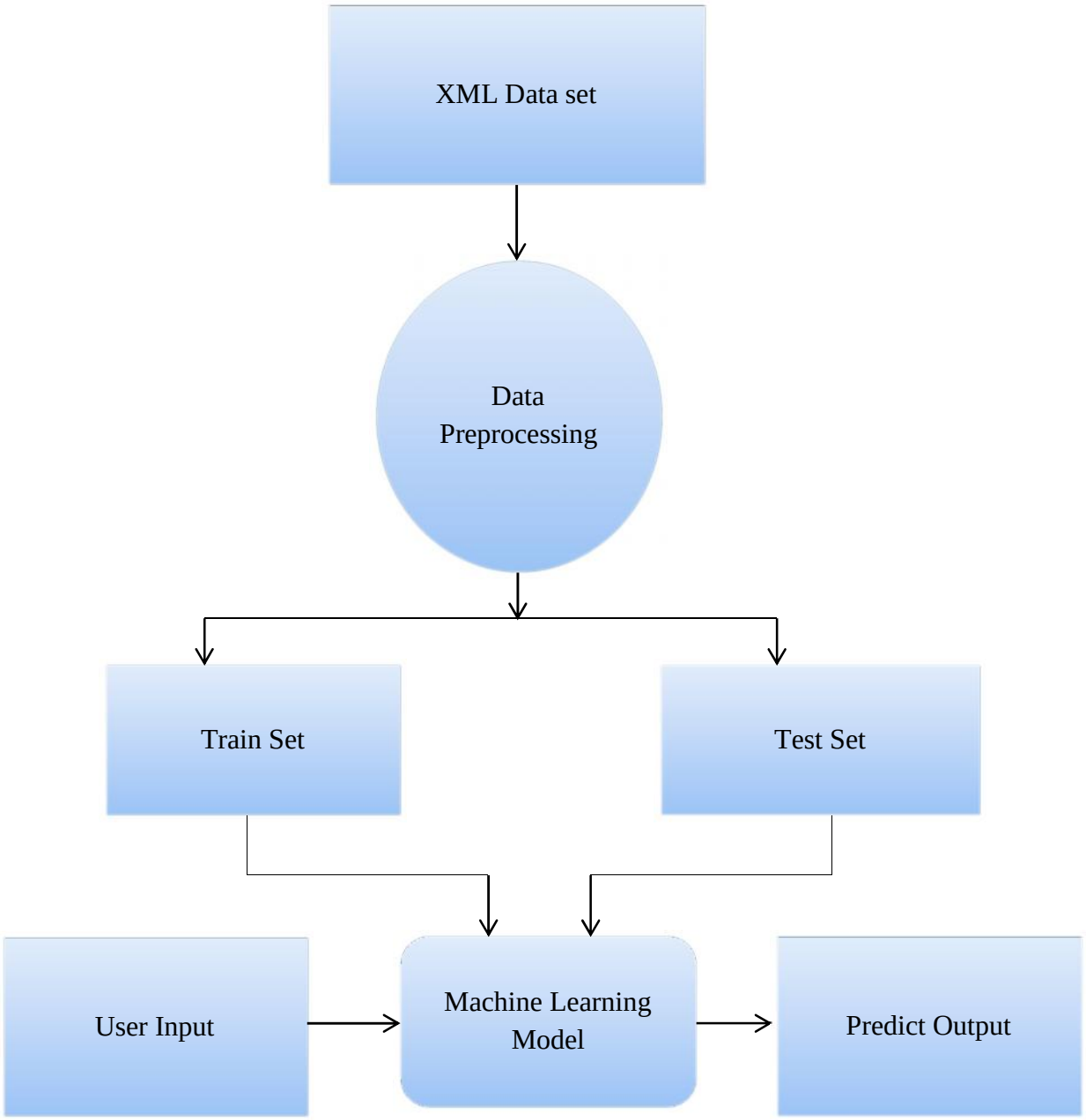

Fig. 4.2.1: System Architecture Diagram

Key Components of this Project are:

i XML data set

ii model.py

4.2.1.1 XML Data set

A markup language that defines a set of rules for encoding documents in a format that is both human-readable and machine-readable is known as Extensible Markup Language.

Emphasizing simplicity, generality, and usability across the Internet are the design goals of XML. It is a textual data format with strong support via Unicode for different human languages. The language is widely used for the representation of arbitrary data structures such as those used in web services, although the design of XML focuses on documents.

Several schema systems exist to aid in the definition of XML-based languages, while programmers have developed many application programming interfaces (APIs) to aid the processing of XML data.

XML stores data in plain text format. This provides a software and hardware independent way of storing, transporting, and sharing data. With XML, data can be available to all kinds of reading machines like people, computers, voice machines, news feeds, etc.

XML dataset addresses extraction of data of interest through an easy to use plaintext declaration that follows the structure of the XML document. The declaration is indented, matching the XML structure, the data we are interested in is tagged against a dataset.

4.2.1.2 Data Flow Diagram

A data flow diagram is the graphical representation of the flow of data through an information system. DFD is very useful in understanding a system and can be efficiently used during analysis. A DFD shows the flow of data through a system. It views a system as a function that transforms the inputs into desired outputs. Any complex systems will not perform this transformation in a single step and a data will typically undergo a series of transformations before it becomes the output.

With a data flow diagram, users are able to visualize how the system will operate that the system will accomplish and how the system will be implemented, old system data flow diagrams can be drawn up and compared with a new systems data flow diagram to draw comparisons to implement a more efficient system.

Data flow diagrams can be used to provide the end user with a physical idea of where they input, ultimately as an effect upon the structure of the whole system.

In the perspective of application development Data Flow Diagram (DFD) is a special chart type which lets graphically illustrate the "flow" of data through various application component. So the Data Flow Diagrams can be successfully used for visualization of data processing or structured design.

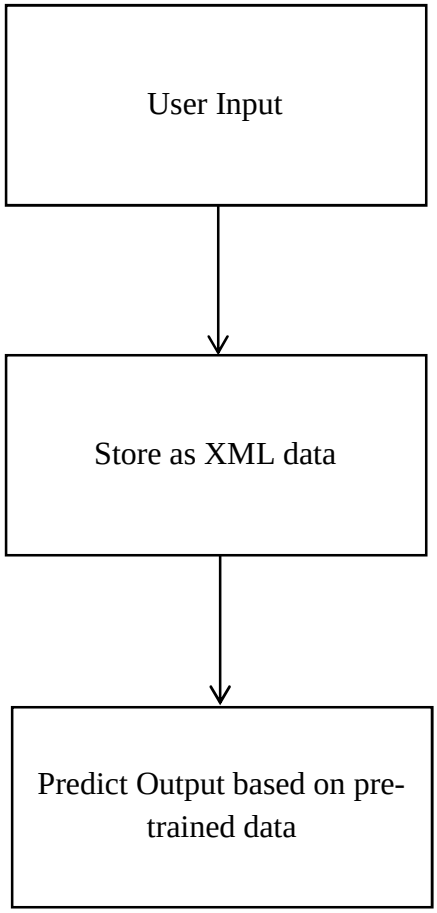

Fig. 4.2.1.2: Data Flow Diagram

4.2.1.3 Flow Chart

Flowcharts are used in designing and documenting simple processes or programs. Like other types of diagrams, they help visualize what is going on and thereby help understand a process, and perhaps also find flaws, bottlenecks, and other less-obvious features within it. There are many different types of flowcharts, and each type has its own repertoire of boxes and notational conventions. The two most common types of boxes in a flowchart are:

- A processing step, usually called activity, and denoted as a rectangular box.
- A decision usually denoted as a diamond.

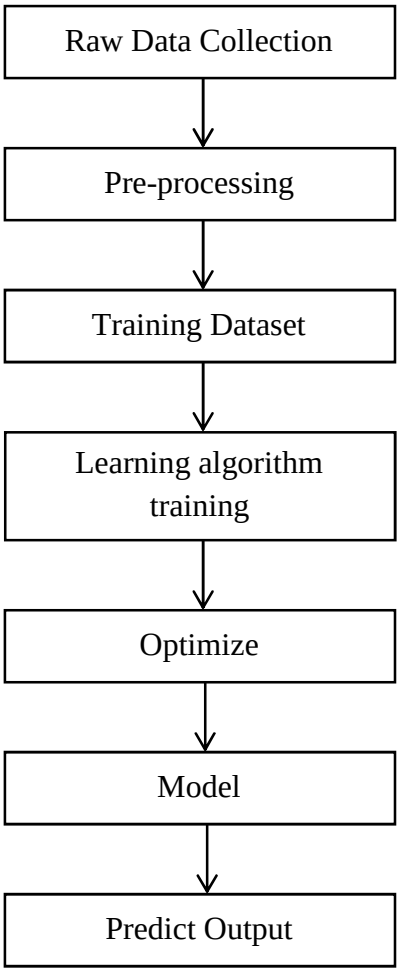

Fig. 4.2.1.3: Flowchart

4.2.1.4 Use Case Diagram

The external objects that interact directly with the system are called actors. Actors include humans, external devices and other software systems. The important thing about actors is that they are not under control of the application. In this project, user of the system is the actor. To find use cases, for each actor, list the fundamentally different ways in which the actor uses the system. Each of these ways is a use case.

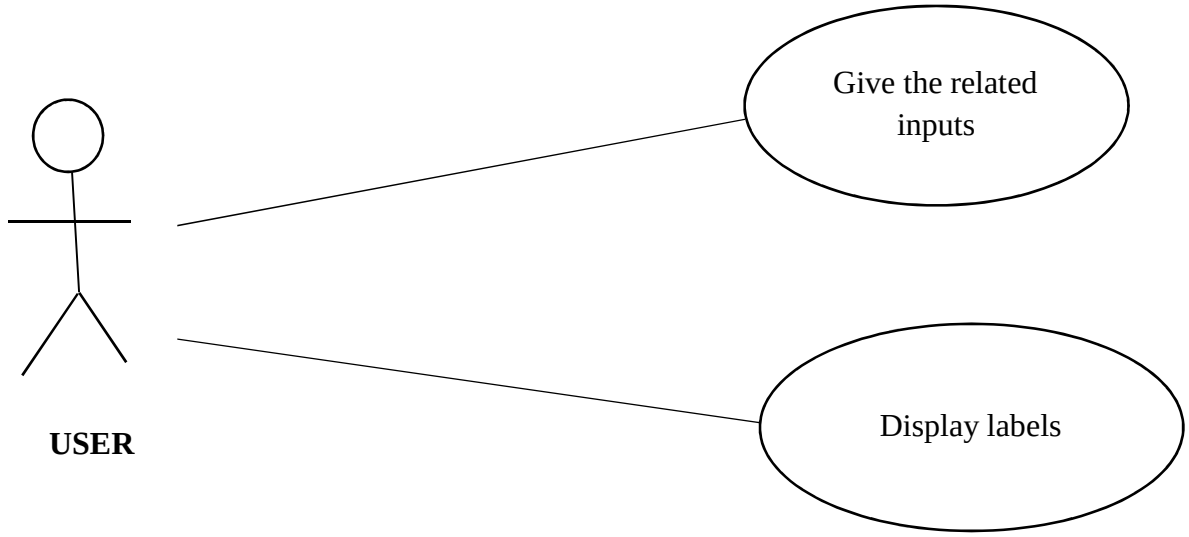

Fig. 4.2.1.4 Use Case Diagram

4.2.1.5 Sequence Diagram

A sequence diagram in a Unified Modelling Language (UML) is a kind of interaction diagram that shows how processes operate with one another and in what order. It shows the participants in an interaction and the sequence of messages among them; each participant is assigned a column in a table.

Below section shows the sequence diagrams in this application

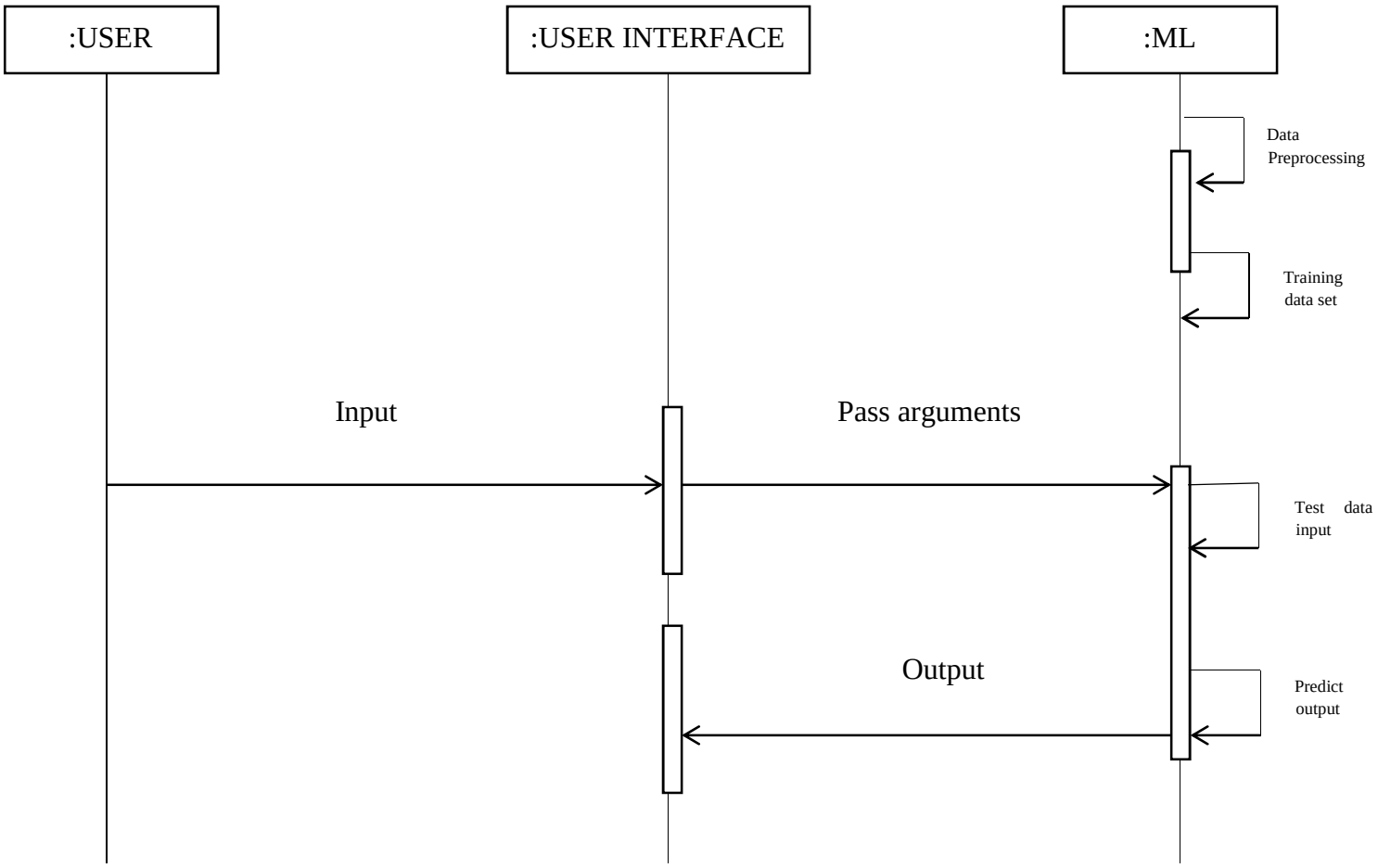

Fig. 4.2.1.5 Sequence Diagram

CHAPTER 5

IMPLEMENTATION

5.1 Introduction

The realizing of an application or execution of a plan, idea, model, design, specification, standard, algorithm, or policy is known as implementation. In other words, an implementation is a realization of a technical specification or algorithm as a program, software component, or other computer system through programming and deployment. Many implementations may exist for a given specification or standard.

Implementation is one of the most important phases of the Software Development Life Cycle (SDLC). It encompasses all the processes involved in getting new software or hardware operating properly in its environment, including installation, configuration, running, testing, and making necessary changes. Specifically, it involves coding the system using a particular programming language and transferring the design into an actual working system.

5.2 Overview of System Implementation

This project is implemented considering the following aspects:

1. Usability Aspect.
2. Technical Aspect.

5.2.1 Usability Aspect

The usability aspect of implementation of the project is realized using two principles:

5.2.1.1 The project is implemented using PYTHON

Python is an interpreted, high-level, general-purpose programming language. Created by Guido van Rossum and first released in 1991, Python has a design philosophy that emphasizes code readability, notably using significant whitespace. It provides constructs that enable clear programming on both small and large scales.

Python features a dynamic tape system and automatic memory management. It supports multiple programming paradigms, including object-oriented, imperative, functional and procedural and has a large and comprehensive standard library.

Python interpreters are available for many operating systems. CPython, the reference implementation of Python, is open-source software and has a community-based development model. Python and CPython are managed by the non-profit Python Software Foundation.

Rather than having all of its functionality built into its core, Python was designed to be highly extensible. This compact modularity has made it particularly popular as a means of adding programmable interfaces to existing applications. Van Rossum's vision of a small core language with a large standard library and easily extensible interpreter stemmed from his frustrations with ABC, which espoused the opposite approach.

5.2.2 Technical Aspect

The technical aspect of implementation of the project is realized using following principle:

5.2.2.1 Anaconda

Anaconda is a free and open-source distribution of the Python and R programming languages for scientific computing (data science, machine learning applications, large-scale data processing, predictive analytics, etc.), that aims to simplify package management and deployment. Package

versions are managed by the package management system conda. The Anaconda distribution is used by over 13 million users and includes more than 1400 popular data-science packages suitable for Windows, Linux, and MacOS.

5.2.2.2 NumPy

NumPy is the fundamental package for scientific computing with Python. It contains among other things:

- a powerful N-dimensional array object
- sophisticated (broadcasting) functions
- tools for integrating C/C++ and Fortran code
- useful linear algebra, Fourier transform, and random number capabilities

Besides its obvious scientific uses, NumPy can also be used as an efficient multi-dimensional container of generic data. Arbitrary data-types can be defined. This allows NumPy to seamlessly and speedily integrate with a wide variety of databases.

5.2.2.3 Matplotlib

Matplotlib is a Python 2D plotting library which produces publication quality figures in a variety of hardcopy formats and interactive environments across platforms. Matplotlib can be used in Python scripts, the Python and IPython shells, the Jupyter notebook, web application servers, and four graphical user interface toolkits.

Matplotlib tries to make easy things easy and hard things possible. You can generate plots, histograms, power spectra, bar charts, errorcharts, scatterplots, etc., with just a few lines of code. For examples, see the sample plots and thumbnail gallery.

For simple plotting the pyplot module provides a MATLAB-like interface, particularly when combined with IPython. For the power user, you have full control of line styles, font properties, axes properties, etc, via an object oriented interface or via a set of functions familiar to MATLAB users.

5.2.2.4 OpenCV

OpenCV (Open Source Computer Vision Library) is an open source computer vision and machine learning software library. OpenCV was built to provide a common infrastructure for computer vision applications and to accelerate the use of machine perception in the commercial products. Being a BSD-licensed product, OpenCV makes it easy for businesses to utilize and modify the code.

The library has more than 2500 optimized algorithms, which includes a comprehensive set of both classic and state-of-the-art computer vision and machine learning algorithms. These algorithms can be used to detect and recognize faces, identify objects, classify human actions in videos, track camera movements, track moving objects, extract 3D models of objects, produce 3D point clouds from stereo cameras, stitch images together to produce a high resolution image of an entire scene, find similar images from an image database, remove red eyes from images taken using flash, follow eye movements, recognize scenery and establish markers to overlay it with augmented reality, etc.

5.3 Implementation Support

5.3.1 Installation of Jupyter Notebook

Following are the requirements for installation of Jupyter Notebook on Windows Operating System:

- Microsoft Windows 7/8/10 (32-bit or 64-bit)
- 2 GB RAM minimum, 4 GB RAM recommended
- 1 GB of available disk space minimum, 2 GB Recommended
- 1280x800 minimum screen resolution
- Python 3.3 or greater or Python 2.7

To install Jupyter Notebook on Windows, we should proceed as follows:

1. Download Anaconda.
2. Install the version of Anaconda which you downloaded, following the instructions on the download page.
3. Jupyter Notebook is automatically installed. To run the notebook: jupyter notebook

CHAPTER 6

PSEUDO CODE

Pseudo Code uses the structural conventions of a normal programming language but the intention is for human reading rather than machine reading. Omission of details that are essential for machine understanding of the algorithm such as variable declarations, specific system code and some sub-routines is pseudocode. Augmentation of programming language is done with natural language description details where convenient or with compact mathematical notation. The purpose of using pseudocode is that it is easier for people to understand than conventional programming language code and that the key principles of an algorithm are efficiently and environment independently described. Pseudocode is commonly used in textbooks and scientific publications that are documenting various algorithms and also in plan of computer program development, for sketching the structure of the program before the actual coding takes place.

Steps:

1. Firstly, we imported Python libraries.
2. Loading of trained model was done.
3. Next, we created a detection mask function to obtain bounding boxes.
4. Finally, using OpenCV, model is run real-time.

6.1 model_faster_rcnn.py

```
import numpy as np
import os
import six.moves.urllib as urllib
import sys
import tarfile
import tensorflow as tf
import zipfile

from distutils.version import StrictVersion
from collections import defaultdict
from io import StringIO
from matplotlib import pyplot as plt
from PIL import Image

# 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

print(tf.__version__)
if StrictVersion(tf.__version__) < StrictVersion('1.12.0'):
  raise ImportError('Please upgrade your TensorFlow installation to v1.12.*.')
%matplotlib inline
from utils import label_map_util
from utils import visualization_utils as vis_util
MODEL_NAME = 'faster_rcnn_inception_v2_coco'
MODEL_FILE = MODEL_NAME + '.tar.gz'
# 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', 'labelmap.pbtxt')
tar_file = tarfile.open(MODEL_FILE)
for file in tar_file.getmembers():
  file_name = os.path.basename(file.name)
  if 'frozen_inference_graph.pb' in file_name:
    tar_file.extract(file, os.getcwd())
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='')
category_index = label_map_util.create_category_index_from_labelmap(PATH_TO_LABELS, use_display_name=True)
def run_inference_for_single_image(image, graph):
    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
import cv2
cap = cv2.VideoCapture(0)
try:
    with detection_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)
            while True:
                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)
                # 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('object_detection', cv2.resize(image_np, (800, 600)))
                if cv2.waitKey(25) & 0xFF == ord('q'):
                    cap.release()
                    cv2.destroyAllWindows()
                    break
except Exception as e:
    print(e)
    cap.release()
```

6.2 model_ssd_mobilenet.py

```
import numpy as np
import os
import six.moves.urllib as urllib
import sys
import tarfile
import tensorflow as tf
import zipfile

from distutils.version import StrictVersion
from collections import defaultdict
from io import StringIO
from matplotlib import pyplot as plt
from PIL import Image

# 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.*.')
%matplotlib inline
from utils import label_map_util
from utils import visualization_utils as vis_util
MODEL_NAME = 'ssd_mobilenet_v1_coco'
MODEL_FILE = MODEL_NAME + '.tar.gz'

# 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', 'labelmap.pbtxt')
tar_file = tarfile.open(MODEL_FILE)
for file in tar_file.getmembers():
  file_name = os.path.basename(file.name)
  if 'frozen_inference_graph.pb' in file_name:
    tar_file.extract(file, os.getcwd())
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='')
category_index = label_map_util.create_category_index_from_labelmap(PATH_TO_LABELS, use_display_name=True)
def run_inference_for_single_image(image, graph):
    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
import cv2
cap = cv2.VideoCapture(0)
try:
```

```
    with detection_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)

            while True:
                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)
                # 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('object_detection', cv2.resize(image_np, (800, 600)))
                if cv2.waitKey(25) & 0xFF == ord('q'):
                    cap.release()
                    cv2.destroyAllWindows()
                    break
except Exception as e:
    print(e)
    cap.release()
```

CHAPTER 7

TESTING

The discovery of errors is the purpose of testing. Discovery of every possible conceivable fault or weakness in a work product is called Testing. Testing provides a way to check the functionality of components, sub assemblies, assemblies and a finished product. Exercise of software with the intent of ensuring that the software system meets its requirements and user expectations and does not fail in an unacceptable manner is the process of testing. The system has been verified and validated by running the test data and live data.

7.1: Test Cases

Table 7.1: Test Cases for **model_rcnn.py**

Test Case Id	**Description**	**Input Object**	**Expected o/p**	**Actual o/p**	**Result**
1	Object with high exposure to light	Cup	Object label “Cup”	Object label “Cup”	Pass
2	Object placed near to camera	Chair	Object label “Chair”	Object label “Chair”	Pass
3	Object with low exposure to light	Cup	Object label “Cup”	Object label “Cup”	Pass
4	Object placed far from camera	Chair	Object label “Chair”	Object label “Chair”	Pass
5	Objects with high exposure to light	Bag, Mobile, Table	Object labels “Bag, Mobile, Table”	Object labels “Bag, Mobile, Table”	Pass
6	Objects placed near to camera	Sofa, Table, Chair	Object labels “Sofa, Table, Chair”	Object labels “Sofa, Table, Chair”	Pass

7	Objects with low exposure to light	Bag, Mobile, Table	Object labels “Bag, Mobile, Table”	Object labels “Bag, Mobile, Table”	Pass
8	Objects placed far from camera	Sofa, Table, Chair	Object labels “Sofa, Table, Chair”	Object labels “Sofa, Table, Chair”	Pass

Table 7.2: Test Cases for **model_ssd_mobilenet.py**

Test Case Id	**Description**	**Input Object**	**Expected o/p**	**Actual o/p**	**Result**
1	Object with high exposure to light	Cup	Object label “Cup”	Object label “Cup”	Pass
2	Object placed near to camera	Chair	Object label “Chair”	Object label “Chair”	Pass
3	Object with low exposure to light	Cup	Object label “Cup”	Object label “Cup”	Pass
4	Object placed far from camera	Chair	Object label “Chair”	Object label “Chair”	Pass
5	Objects with high exposure to light	Bag, Mobile, Table	Object labels “Bag, Mobile, Table”	Object labels “Bag, Mobile, Table”	Pass
6	Objects placed near to camera	Sofa, Table, Chair	Object labels “Sofa, Table, Chair”	Object labels “Sofa, Table, Chair”	Pass
7	Objects with low exposure to light	Bag, Mobile, Table	Object labels “Bag, Mobile, Table”	Object labels “Bag, Mobile, Table”	Pass
8	Objects placed far from camera	Sofa, Table, Chair	Object labels “Sofa, Table, Chair”	Object labels “Sofa, Table, Chair”	Pass

CHAPTER 8

RESULTS

The notebook is stored in the object_detection folder. Path to frozen detection graph is supplied. This is the actual model that is used for the object detection. List of the strings that is used to add correct label for each box are provided.

Processing for a single image is done and a reframe is required to translate mask from box coordinates to image coordinates and fit the image size. Convention is followed by adding back the batch dimension and inference is run. All outputs are float32 numpy arrays and are converted to appropriate types.

Handles to input and output tensors are obtained and dimensions are expanded since the model expects images to have shape. Actual detection is done and the results are visualized.

8.1 Results Using Faster-RCNN

Fig 8.1.1 Faster R-CNN in outdoor environment

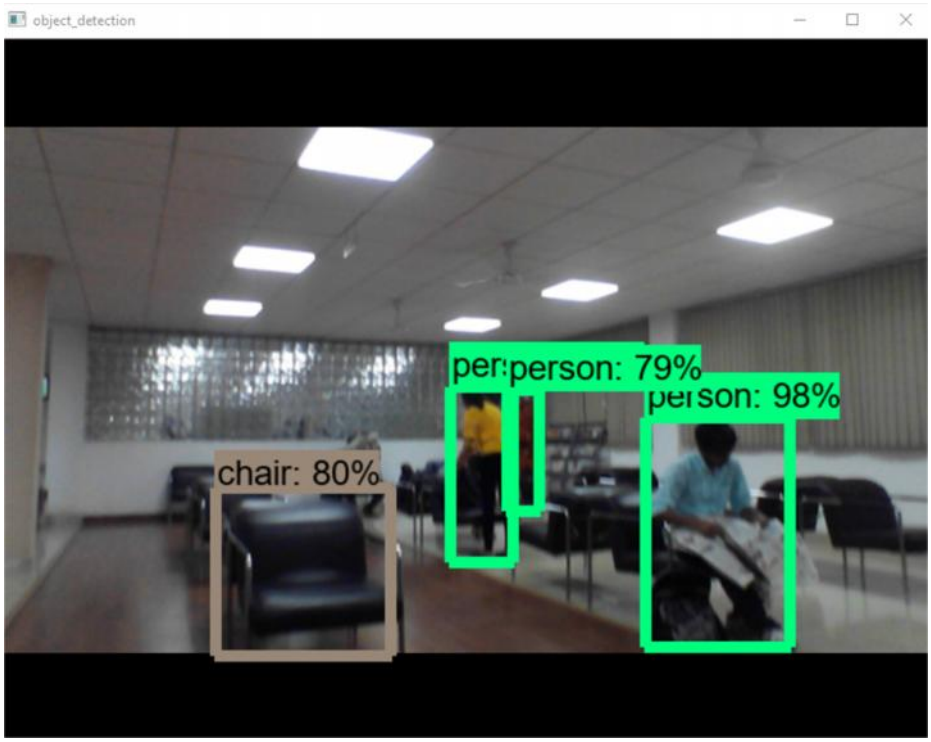

Fig. 8.1.2 Faster R-CNN in indoor environment

8.2 Results Using SSD

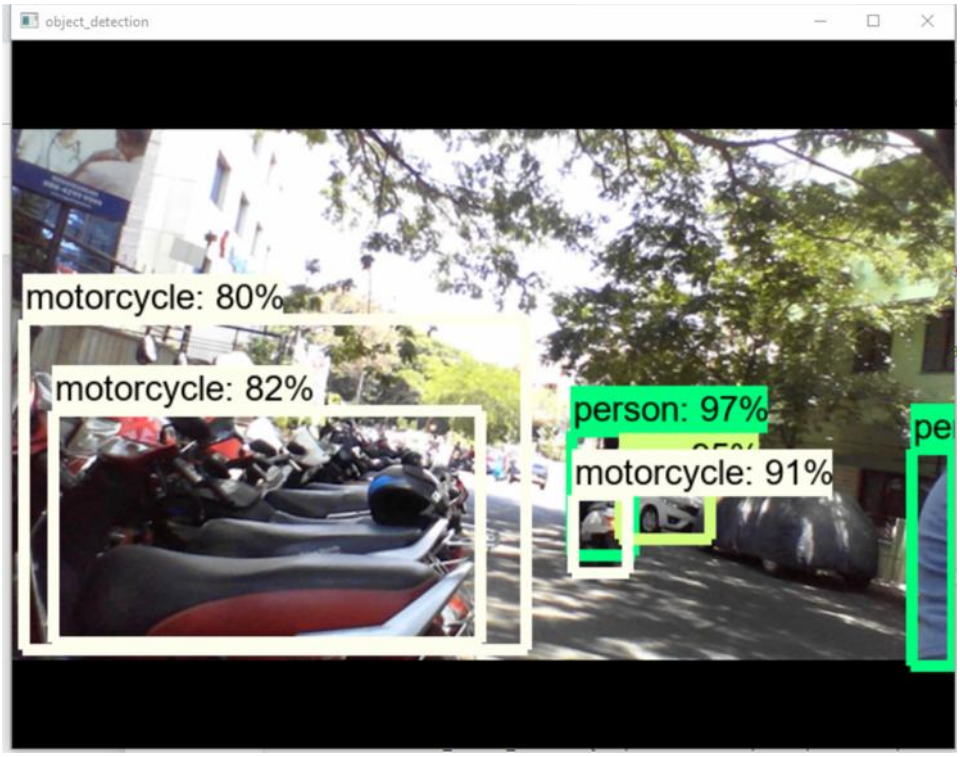

Fig. 8.2.1 SSD in outdoor environment

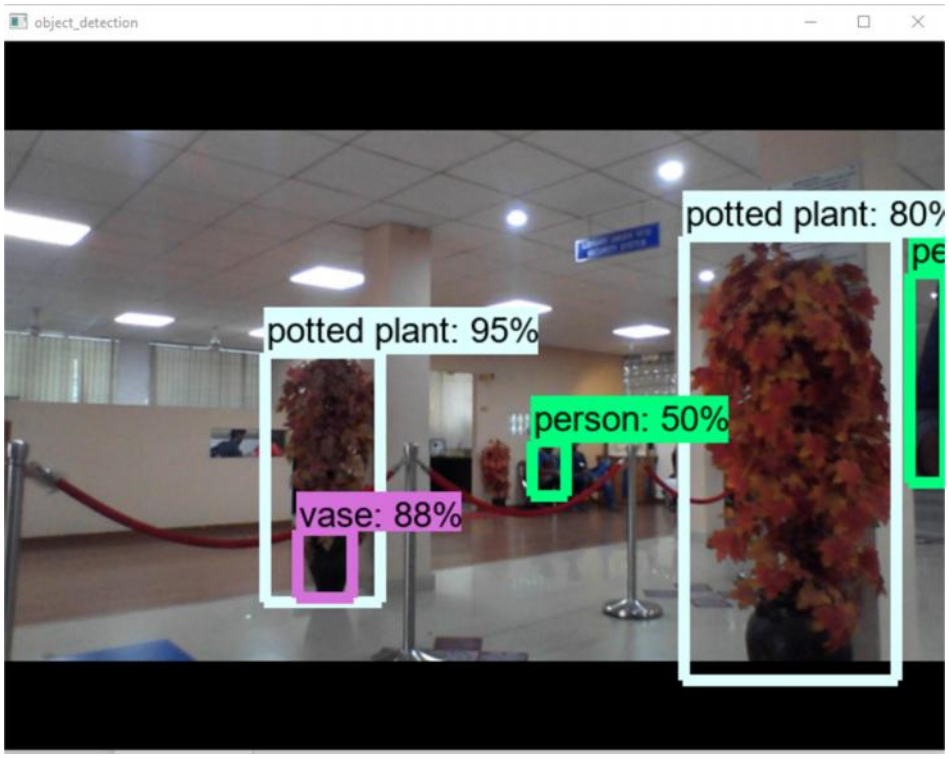

Fig. 8.2.2 SSD in indoor environment

CHAPTER 9

CONCLUSION AND FUTURE SCOPE

The difficulty for a beginner to the field of deep learning to know what type of network to use is high. There are so many types of networks to choose from and new methods being published and discussions are happening every day. Making things worse, most neural networks are flexible enough that they work and make a prediction even when used with the wrong type of data or prediction problem.

Convolutional Neural Networks were designed to map image data to an output variable. They have proven so effective that they are the go to method for any type of prediction problem which involves image data as an input.

Region based detectors like Faster R-CNN demonstrate a small accuracy advantage if real-time speed is not needed. Single shot detectors are here for real time processing. But applications need to verify whether it meets their accuracy requirement.

Chapter 10

REFERENCES

Reference Papers:

[1] Shaoqing Ren, Kaiming He, Ross Girshick, Jian Sun, “Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks”, Conference on Neural Information Processing Systems 2015.
DOI 10.1109/TPAMI.2016.2577031

[2] Jifeng Dai, Yi Li, Kaiming He, Jian Sun, “R-FCN: Object Detection via Region-based Fully Convolutional Networks
arXiv: 1605.06409v2 [cs.CV] 21 Jun 2016

[3] Christian Szegedy, Scott Reed, Dumitru Erhan, Dragomir Anguelov, Sergey Ioffe, “Scalable High Quality Object Detection”
arXiv:1412.1441v3 [cs.CV] 9 Dec 2015

[4] Wei Liu, Dragomir Anguelov, Dumitru Erhan, Christian Szegedy, Scott Reed, Cheng-Yang Fu, Alexander C. Berg, “SSD: Single Shot MultiBox Detector”
arXiv:1512.02325v5 [cs.CV] 29 Dec 2016

[5] Joseph Redmon, Santosh Divvala, Ross Girshick, Ali Farhadi, “You Only Look Once: Unified, Real-Time Object Detection”
arXiv: 1506.02640v5 [cs.CV] 9 May 2016

[6] Jonathan Huang, Vivek Rathod, Chen Sun, Menglong Zhu, Anoop Korattikara, Alireza Fathi, Ian Fischer, Zbigniew Wojna, Yang Song, Sergio Guadarrama, Kevin Murphy. Speed/accuracy trade-offs for modern convolutional object detectors , *Huang et al., CVPR 2017.*

[7] A. Krizhevsky, I. Sutskever, and G. E. Hinton. Imagenet classification with deep convolutional neural networks. In Advances in neural information processing systems, pages 1097–1105, 2012.

[8] Pratik Dubal, Rohan Mahadev, Suraj Kothawade, Kunal Dargan, Rishabh Iyer. Deployment of Customized Deep Learning based Video Analytics On Surveillance Cameras. arXiv:1805.10604v2 [cs.CV] 27 Jun 2018.

[9] J. Dai, Y. Li, K. He, and J. Sun. R-fcn: Object detection via region-based fully convolutional networks. arXiv preprint
arXiv:1605.06409, 2016.

[10] C.-Y. Fu, W. Liu, A. Ranga, A. Tyagi, and A. C. Berg. Dssd: Deconvolutional single shot detector. arXiv preprint
arXiv:1701.06659, 2017.

[11] R. Girshick. Fast r-cnn. In Proceedings of the IEEE International Conference on Computer Vision, pages 1440–1448,

[12] T. Y. Lin and P. Dollar. Ms coco api. https://github. com/pdollar/coco, 2016.

[13] S. Ioffe and C. Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift.
arXiv preprint arXiv:1502.03167, 2015.

[14] M. Abadi, A. Agarwal, P. Barham, E. Brevdo, Z. Chen, C. Citro, G. S. Corrado, A. Davis, J. Dean, M. Devin, et al. Tensorflow: Large-scale machine learning on heterogeneous systems, 2015. Software available from tensorflow. org, 1, 2015.

[15] S. Bell, C. L. Zitnick, K. Bala, and R. Girshick. Insideoutside net: Detecting objects in context with skip pooling and recurrent neural networks. arXiv preprint
arXiv:1512.04143, 2015.

[16] A. Canziani, A. Paszke, and E. Culurciello. An analysis of deep neural network models for practical applications. arXiv preprint
arXiv:1605.07678, 2016.

[17] A. Howard, M. Zhu, B. Chen, D. Kalenichenko, W. Wang, T. Weyand, M. Andreetto, and H. Adam. Mobilenets: Efficient convolutional neural networks for mobile vision applications. arXiv preprint
arXiv:1704.04861, 2017.

[18] Y. Jia, E. Shelhamer, J. Donahue, S. Karayev, J. Long, R. Girshick, S. Guadarrama, and T. Darrell. Caffe: Convolutional architecture for fast feature embedding. In Proceedings of the 22nd ACM international conference on Multimedia, pages 675–678.
ACM, 2014.

[19] T.-Y. Lin, P. Dollar, R. Girshick, K. He, B. Hariharan, and S. Belongie. Feature pyramid networks for object detection.
arXiv preprint arXiv:1612.03144, 2016.

[20] A. Shrivastava, A. Gupta, and R. Girshick. Training regionbased object detectors with online hard example mining. arXiv preprint
arXiv:1604.03540, 2016.

Reference Websites

The https://towardsdatascience.com/ websites helped us understand many concepts related to machine learning that helped us to choose appropriate algorithms for the work.

Data has been collected from the following websites:

- https://www.google.com/imghp?hl=en
- https://pixabay.com/
- https://www.datacamp.com/