

**OBJECT RECOGNITION THROUGH COMPUTER VISION
FOR VISUALLY IMPAIRED**

*An Project report submitted in partial fulfillment of the requirements for
the award of the degree of*

**BACHELOR OF TECHNOLOGY
IN
ELECTRONICS AND COMMUNICATION ENGINEERING**

Submitted by

B. Lakshmikanth (317126512126)

P. Sandhya (317126512160)

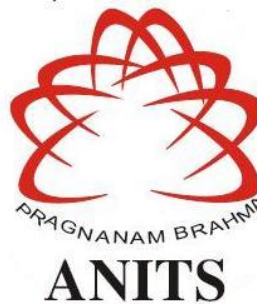
K. Yashwanth Surya (317126512180)

T. Indu priya (318126512128)

Under the guidance of

Dr.G.Prasanna

Assistant Professor



DEPARTMENT OF ELECTRONICS AND COMMUNICATION ENGINEERING

**ANIL NEERUKONDA INSTITUTE OF TECHNOLOGY AND SCIENCES
(UGC AUTONOMOUS)**

**DEPARTMENT OF ELECTRONICS AND COMMUNICATION ENGINEERING
ANIL NEERUKONDA INSTITUTE OF TECHNOLOGY AND SCIENCES
(UGC AUTONOMOUS)**

*(Permanently Affiliated to AU, Approved by AICTE and Accredited by NBA & NAAC with
'A' Grade)*

Sangivalasa, Bheemilimandal, Visakhapatnam dist.(A.P)



ANITS

CERTIFICATE

This is to certify that the project report entitled "OBJECT RECOGNITION THROUGH COMPUTER VISION FOR VISUALLY IMPAIRED" submitted by B.Lakshmikanth (317126512126), P.Sandhya (317126512160), K.Yashwanth Surya (317126512180), T.Indu priya (318126512L28) in partial fulfillment of the requirements for the award of the degree of Bachelor of Technology in Electronics & Communication Engineering of Andhra University, Visakhapatnam is a record of bonafide work carried out under my guidance and supervision.

Project Guide

[Signature]
Dr.G.Prasanna

Assistant Professor
Department of E.C.E
ANITS

**Assistant Professor
Department of E.C.E.
Anil Neerukonda**

Institute of Technology & Sciences
Sangivalasa, Visakhapatnam - 531 162

Head of the Department

[Signature]
Dr. V.Rajyalakshmi
Professor & HOD

Department of E.C.E
ANITS

**Head of the Department
Department of E C E**

**Anil Neerukonda Institute of Technology & Sciences
Sangivalasa - 531 162**

ACKNOWLEDGEMENT

We would like to express our deep gratitude to our project guide **Dr.G.Prasanna**, Assistant Professor, Department of Electronics and Communication Engineering, ANITS, for his/her guidance with unsurpassed knowledge and immense encouragement. We are grateful to **Dr. V. Rajyalakshmi**, Head of the Department, Electronics and Communication Engineering, for providing us with the required facilities for the completion of the project work.

We are very much thankful to the **Principal and Management, ANITS, Sangivalasa**, for their encouragement and cooperation to carry out this work.

We express our thanks to all **teaching faculty** of Department of ECE, whose suggestions during reviews helped us in accomplishment of our project. We would like to thank **all non-teaching staff** of the Department of ECE, ANITS for providing great assistance in accomplishment of our project.

We would like to thank our parents, friends, and classmates for their encouragement throughout our project period. At last but not the least, we thank everyone for supporting us directly or indirectly in completing this project successfully.

PROJECT STUDENTS

B.Lakshmikanth (317126512126)

P.Sandhya (317126512160)

K.Yashwanth Surya (317126512180)

T.Indu priya (318126512L28)

CONTENTS

ABSTRACT	05
LIST OF FIGURES	07
LIST OF TABLES	08
LIST OF ABBREVIATIONS	09
CHAPTER 1 INTRODUCTION	10
1.1 Project objective	10
1.2 Project outline	11
CHAPTER 2 OPENCV	11
2.1 Introduction	14
2.2 Basic operations on an image	21
CHAPTER 3 OBJECT RECOGNITION MODULE	32
3.1 Haar Cascade Algorithm	32
3.2 Single Shot Detector	40
CHAPTER 4 WEBSERVER USING DJANGO	40
4.1 Introduction and Installation	45
4.2 Views in Django	52
4.3 Project MVT Structure	54
CHAPTER 5 HARDWARE MODEL	54
5.1 Components	57
5.2 Implementation	59
CHAPTER 6 RESULTS AND DISCUSSIONS	60
CONCLUSION	62
REFERENCES	63
PAPER PUBLICATION	

ABSTRACT

Vision is one of the very essential human senses and it plays the most important role in human perception of our environment, unfortunately there are many people who are visually impaired. Blind people today rely on sighted guides, seeing-eye dogs and canes even a century after these came into existence. The present work aims to aid the blind through a wrist wearable. The wearable is designed to capture the user's environment through a camera and recognize the objects present in image. These identified objects are informed to user through an audio output. The object recognition is achieved through OpenCV.

LIST OF FIGURES

Fig. 2.1	The display of a normal vision of surroundings	13
Fig. 2.2	The extracting of an image	15
Fig.2.3	The original image	17
Fig. 2.4	The resizing of an image	17
Fig. 2.5	The rotation of an image	18
Fig. 2.6	The creating the bounding box/rectangle of an Image	19
Fig. 2.7	The text of an image	20
Fig. 3.1	The haarcascade features	22
Fig. 3.2	The integral values of an image	24
Fig. 3.3	The Integral Image is used here to calculate the haar value	25
Fig. 3.4	The Haar calculation from Integral Image	26
Fig.3.5	The ratio of alpha and error rate of an image	27
Fig.3.6	The output weights	28
Fig.3.7	A flowchart of cascade classifiers	28
Fig.3.8	Architecture of Single Shot MultiBox detector (input is 300x300x3)	31
Fig.3.9	VGG architecture (input is 224x224x3)	35
Fig.3.10	Architecture of multi-scale convolutional prediction of the location	34
Fig3.11	Diagram explaining IOU	34
Fig.3.12	SSD default boxes at 8x8 and 4x4 feature map	35
Fig.3.13	Images from Pascal VOC dataset	35
Fig.3.14	VGG Feature Map Visualization	36
Fig.3.15	Example of hard negative mining	37
Fig.3.16	NMS example	38
Fig.4.1	server of Django	38
Fig.4.2	The installed apps	43
Fig.4.3	The normal view of Django	44
Fig.4.4	The classification of Django views	45
Fig.4.5	The template for the Django	46

Fig.4.6	The project structure	52
Fig.5.1	The raspberry pi (hardware tool used in project)	55
Fig.5.2	A Power Bank	57
Fig.5.3	Flowchart/procedure of program	58

LIST OF TABLES

Table 3.1	Comparison of performance of various models	39
-----------	---	----

LIST OF ABBREVIATIONS

OpenCV	Open ComputerVision
IoU	Intersection over Union
CPU	Central Processing Unit
GPU	Graphics Processing Unit
COCO	Common Objects in Context
mAP	Mean Average Precision
CNNs	Convolutional neural networks
SSD	Single Shot Detector

Chapter 1

INTRODUCTION

1.1 Project Objective

Blind people lead a normal life with their own style of doing things. But, they definitely face troubles due to inaccessible infrastructure and in new environment. It is difficult for others to help the visually impaired all the time.

In 2015, there were an estimated 253 million people with visual impairment worldwide. Of these, 36 million were blind and a further 217 million had moderate to severe visual impairment (MSVI). By 2020, it is projected to 76 million people.

The need for identifying objects in the surroundings among blind people and a broader look at the advanced technology available in today's world is the reason to develop this project.

Object recognition is one of the fundamental tasks in computer vision. It is the process of finding or identifying instances of objects (for example faces, dogs or buildings) in digital images or videos.

Globally, the causes of vision impairment are cataract, uncorrected refractive errors, trachoma, diabetic retinopathy, corneal opacity, age related muscular degeneration and eye injuries. It limits visually impaired people to navigate and perform everyday tasks. Advancement in technologies such as hand gesture, text recognition, Eye-ring project are came up with solutions. However, these solutions have some disadvantages like expensive, heavyweight, less accuracy, less speed, etc. So, the objective of the project is to design and implement a system that captures the user's environment through a camera module. The images that are captured are transformed to Raspberry pi and it is analyzed using Haar Cascade and SSD Model. Then the recognized objects are given to the user via audio through a speaker or bluetooth headset. The image data is transferred to data server for further use.

1.2 Project Outline

The project outline consists of:

Chapter 1: Introduction of project and its objectives.

Chapter 2: This chapter mainly focuses on Open ComputerVision library and some basic operations performed on an image using OpenCV library.

Chapter 3: In this chapter object recognition model is proposed. To evaluate the algorithm a household objects database is made, the dataset contains images of objects such as door, mobile phone, chair etc.

Chapter 4: The web server implementation using a python library Django is discussed in this chapter.

Chapter 5: The system is implemented using Raspberry Pi Hardware, the implementation details are discussed. Object and color recognition results with are portrayed.

Chapter 2

OPENCV

2.1 Introduction

OpenCV was started at Intel in 1999 by **Gary Bradsky**, and the first release came out in 2000. **Vadim Pisarevsky** joined Gary Bradsky to manage Intel's Russian software OpenCV team. In 2005, OpenCV was used on Stanley, the vehicle that won the 2005 DARPA Grand Challenge. Later, its active development continued under the support of Willow Garage with Gary Bradsky and Vadim Pisarevsky leading the project. OpenCV now supports a multitude of algorithms related to Computer Vision and Machine Learning and is expanding day by day.

OpenCV supports a wide variety of programming languages such as C++, Python, Java, etc., and is available on different platforms including Windows, Linux, OS X, Android, and iOS. Interfaces for high-speed GPU operations based on CUDA and OpenCL are also under active development.

OpenCV-Python is the Python API for OpenCV, combining the best qualities of the OpenCV C++ API and the Python language.

OpenCV-Python

OpenCV-Python is a library of Python bindings designed to solve computer vision problems. Python is a general purpose programming language started by **Guido van Rossum** that became very popular very quickly, mainly because of its simplicity and code readability. It enables the programmer to express ideas in fewer lines of code without reducing readability. Compared to languages like C/C++, Python is slower. That said, Python can be easily extended with C/C++, which allows us to write computationally intensive code in C/C++ and create Python wrappers that can be used as Python modules. This gives us two advantages: first, the code is as fast as the original C/C++ code (since it is the actual C++ code working in background) and second, it is easier to code in Python than C/C++. OpenCV-Python is a Python wrapper for the original OpenCV C++ implementation.

OpenCV-Python makes use of **Numpy**, which is a highly optimized library for numerical operations with MATLAB-style syntax. All the OpenCV array structures are converted to and

from Numpy arrays. This also makes it easier to integrate with other libraries that use Numpy such as SciPy and Matplotlib.

Applications of OpenCV: There are lots of applications which are solved using OpenCV, some of them are listed below

- face recognition
- Automated inspection and surveillance
- number of people – count (foot traffic in a mall, etc)
- Vehicle counting on highways along with their speeds
- Interactive art installations
- Anomaly (defect) detection in the manufacturing process (the odd defective products)
- Street view image stitching
- Video/image search and retrieval
- Robot and driver-less car navigation and control
- object recognition
- Medical image analysis
- Movies – 3D structure from motion
- TV Channels advertisement recognition

OpenCV Functionality

- Image/video I/O, processing, display (core, imgproc, highgui)
- Object/feature detection (objdetect, features2d, nonfree)
- Geometry-based monocular or stereo computer vision (calib3d, stitching, videostab)
- Computational photography (photo, video, superres)
- Machine learning & clustering (ml, flann)
- CUDA acceleration (gpu)

Image-Processing

Image processing is a method to perform some operations on an image, in order to get an enhanced image and or to extract some useful information from it. If we talk about the basic definition of image processing then **“Image processing is the analysis and manipulation of a digitized image, especially in order to improve its quality”**.

Digital-Image:

An image may be defined as a two-dimensional function $f(x, y)$, where x and y are spatial(plane) coordinates, and the amplitude of f at any pair of coordinates (x, y) is called the intensity or grey level of the image at that point. In another word An image is nothing more than a two-dimensional matrix (3-D in case of colored images) which is defined by the mathematical function $f(x, y)$ at any point is giving the pixel value at that point of an image, the pixel value describes how bright that pixel is, and what color it should be. Image processing is basically signal processing in which input is an image and output is image or characteristics according to requirement associated with that image.

Image processing basically includes the following three steps:

1. Importing the image
2. Analyzing and manipulating the image
3. Output in which result can be altered image or report that is based on image analysis

2.2 Basic Operations on an Image



Fig (2.1):The display of a normal vision of surroundings

Reading an image

```
# Importing the OpenCV library

import cv2

# Reading the image using imread() function

image = cv2.imread('image.png')

# Extracting the height and width of an image

h, w = image.shape[:2]

# Displaying the height and width

print("Height = {}, Width = {}".format(h, w))
```

Note – OpenCV arranges the channels in BGR order. So the 0th value will correspond to Blue pixel and not Red.

Extracting the RGB values of a pixel

```
# Extracting RGB values.

# Here we have randomly chosen a pixel

# by passing in 100, 100 for height and width.

(B, G, R) = image[100, 100]

# Displaying the pixel values

print("R = {}, G = {}, B = {}".format(R, G, B))

# We can also pass the channel to extract

# the value for a specific channel

B = image[100, 100, 0]
```

```
print("B = {}".format(B))
```

Extracting the Region of Interest (ROI)

```
# We will calculate the region of interest
```

```
# by slicing the pixels of the image
```

```
roi = image[100 : 500, 200 : 700]
```



Fig(2.2):The extracting of an image

Resizing the Image

```
# resize() function takes 2 parameters,  
# the image and the dimensions  
resize = cv2.resize(image, (800, 800))
```



Fig(2.3): The original image

The problem with this approach is that the aspect ratio of the image is not maintained. So we need to do some extra work in order to maintain a proper aspect ratio.

```
# Calculating the ratio  
ratio = 800 / w  
  
# Creating a tuple containing width and height  
dim = (800, int(h * ratio))  
  
# Resizing the image  
resize_aspect = cv2.resize(image, dim)
```



Fig(2.4):The resizing of an image

Rotating the Image

Calculating the center of the image

```
center = (w // 2, h // 2)
```

Generating a rotation matrix

```
matrix = cv2.getRotationMatrix2D(center, -45, 1.0)
```

Performing the affine transformation

```
rotated = cv2.warpAffine(image, matrix, (w, h))
```



Fig(2.5): The rotation of an image

Drawing a Rectangle

It is an in-place operation.

```
# We are copying the original image,
```

```
# as it is an in-place operation.
```

```
output = image.copy()
```

```
# Using the rectangle() function to create a rectangle.
```

```
rectangle = cv2.rectangle(output, (1500, 900),
```

```
(600, 400), (255, 0, 0), 2)
```



Fig(2.6):The creating the bounding box/rectangle of an Image

It takes in 5 arguments –

- Image
- Top-left corner co-ordinates
- Bottom-right corner co-ordinates
- Color (in BGR format)
- Line width

Displaying text

It is also an in-place operation

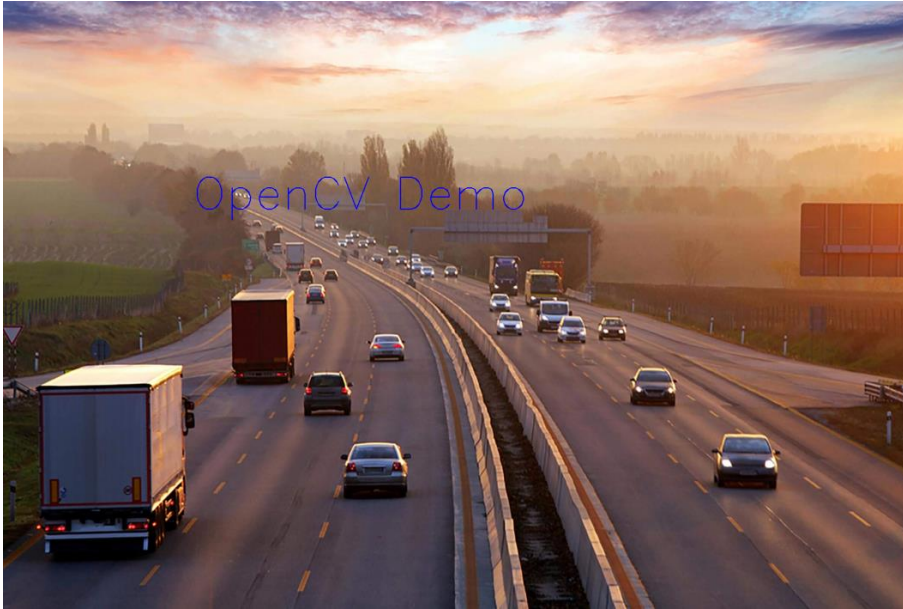
Copying the original image

```
output = image.copy()
```

Adding the text using putText() function

```
text = cv2.putText(output, 'OpenCV Demo', (500, 550),
```

```
cv2.FONT_HERSHEY_SIMPLEX, 4, (255, 0, 0), 2)
```



Fig(2.7):shows the text of an image

It takes in 7 arguments –

1. Image
2. Text to be displayed
3. Bottom-left corner co-ordinates, from where the text should start
4. Font
5. Font size
6. Color (BGR format)
7. Line width

Chapter 3

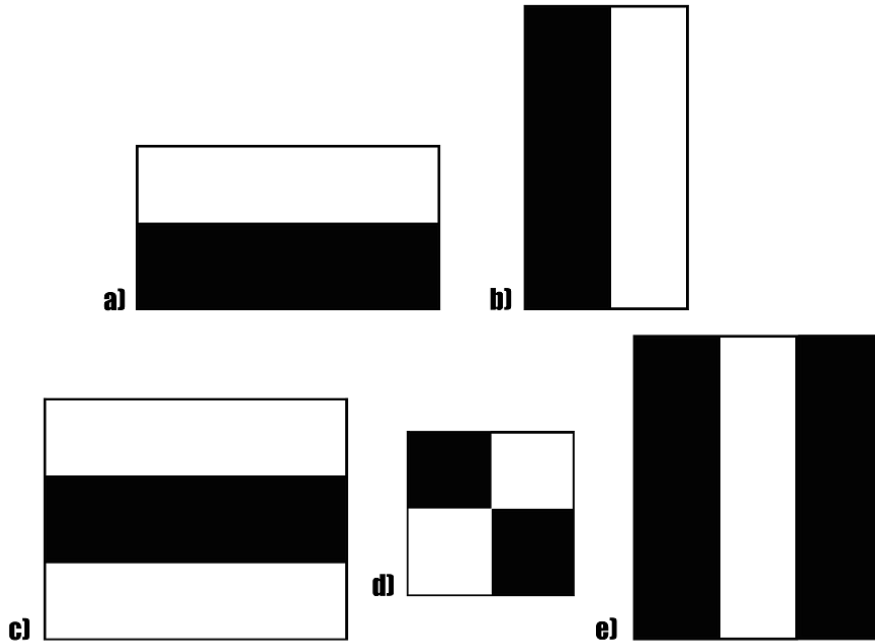
OBJECT RECOGNITION MODULE

3.1 Haar Cascade Algorithm

It is an Object Detection Algorithm used to identify faces in an image or a real time video. The algorithm uses edge or line detection features proposed by Viola and Jones in their research paper “Rapid Object Detection using a Boosted Cascade of Simple Features” published in 2001. The algorithm is given a lot of positive images consisting of faces, and a lot of negative

images not consisting of any face to train on them. The model created from this training is available at the OpenCV GitHub repository

Features



Figure(3.1) :shows the haar cascade features

Fig (a & b) : shows the edge features

Fig (c & e) :shows the line features

Fig(d): shows the four-rectangle features

Fig. A sample of Haar features used in the Original Research Paper published by Viola and Jones.

The first contribution to the research was the introduction of the haar features shown above. These features on the image make it easy to find out the edges or the lines in the image, or to pick areas where there is a sudden change in the intensities of the pixels.

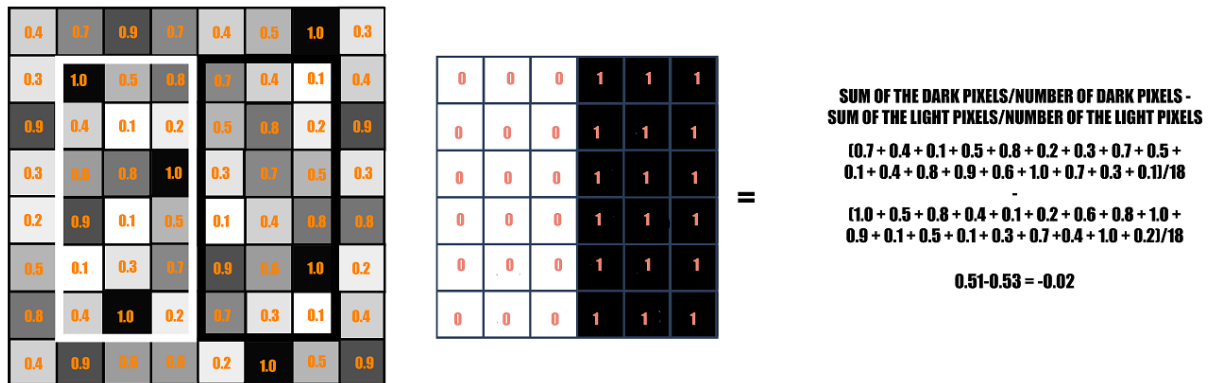


Figure 3.2 shows the integral values of an image that converts the original image into an integral image

Fig. The rectangle on the left is a sample representation of an image with pixel values 0.0 to 1.0. The rectangle at the center is a haar kernel which has all the light pixels on the left and all the dark pixels on the right. The haar calculation is done by finding out the difference of the average of the pixel values at the darker region and the average of the pixel values at the lighter region. If the difference is close to 1, then there is an edge detected by the haar feature.

A sample calculation of Haar value from a rectangular image section has been shown here. The darker areas in the haar feature are pixels with values 1, and the lighter areas are pixels with values 0. Each of these is responsible for finding out one particular feature in the image. Such as an edge, a line or any structure in the image where there is a sudden change of intensities. For ex. in the image above, the haar feature can detect a vertical edge with darker pixels at its right and lighter pixels at its left.

The objective here is to find out *the sum of all the image pixels lying in the darker area of the haar feature* and *the sum of all the image pixels lying in the lighter area of the haar feature*. And then find out their difference. Now if the image has an edge separating dark pixels on the right and light pixels on the left, then the haar value will be closer to 1. That means, we say that there is an edge detected if the haar value is closer to 1. In the example above, there is no edge as the haar value is far from 1.

This is just one representation of a particular haar feature separating a vertical edge. Now there are other haar features as well, which will detect edges in other directions and any other image

structures. To detect an edge anywhere in the image, the haar feature needs to traverse the whole image.

The haar feature continuously traverses from the top left of the image to the bottom right to search for the particular feature. This is just a representation of the whole concept of the haar feature traversal. In its actual work, the haar feature would traverse pixel by pixel in the image. Also all possible sizes of the haar features will be applied.

Depending on the feature each one is looking for, these are broadly classified into three categories. The first set of *two rectangle features* is responsible for finding out the edges in a horizontal or in a vertical direction (as shown above). The second set of *three rectangle features* are responsible for finding out if there is a lighter region surrounded by darker regions on either side or vice-versa. The third set of *four rectangle features* is responsible for finding out change of pixel intensities across diagonals.

Now, the haar features traversal on an image would involve a lot of mathematical calculations. As we can see for a single rectangle on either side, it involves 18 pixel value additions (for a rectangle enclosing 18 pixels). Imagine doing this for the whole image with all sizes of the haar features. This would be a hectic operation even for a high performance machine.

To tackle this, they introduced another concept known as *The Integral Image* to perform the same operation. An Integral Image is calculated from the Original Image in such a way that each pixel in this is the sum of all the pixels lying in its left and above in the Original Image. The calculation of a pixel in the Integral Image can be seen in the above GIF. The last pixel at the bottom right corner of the Integral Image will be the sum of all the pixels in the Original Image.

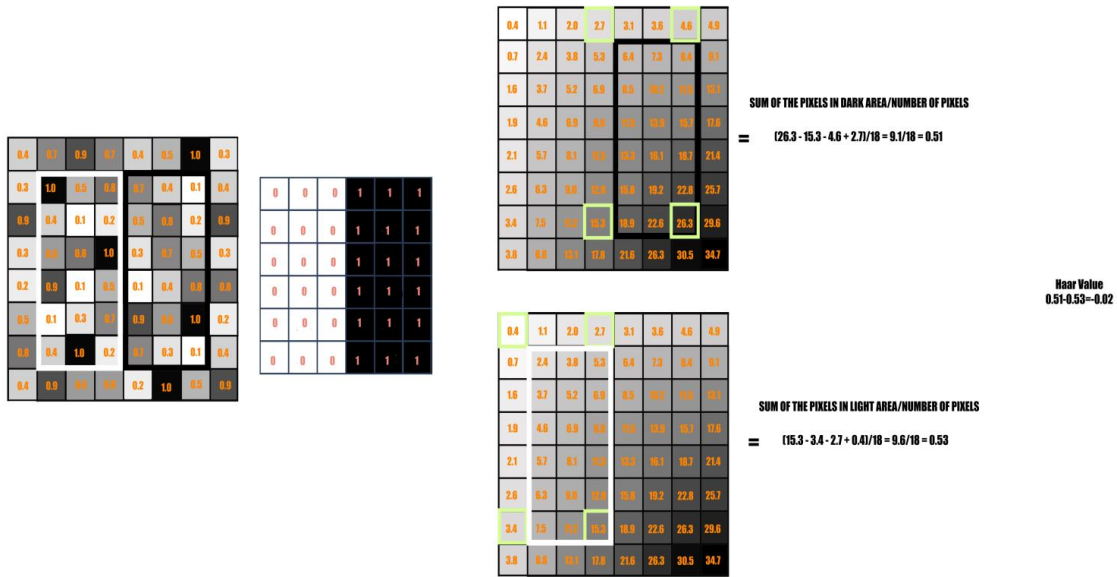
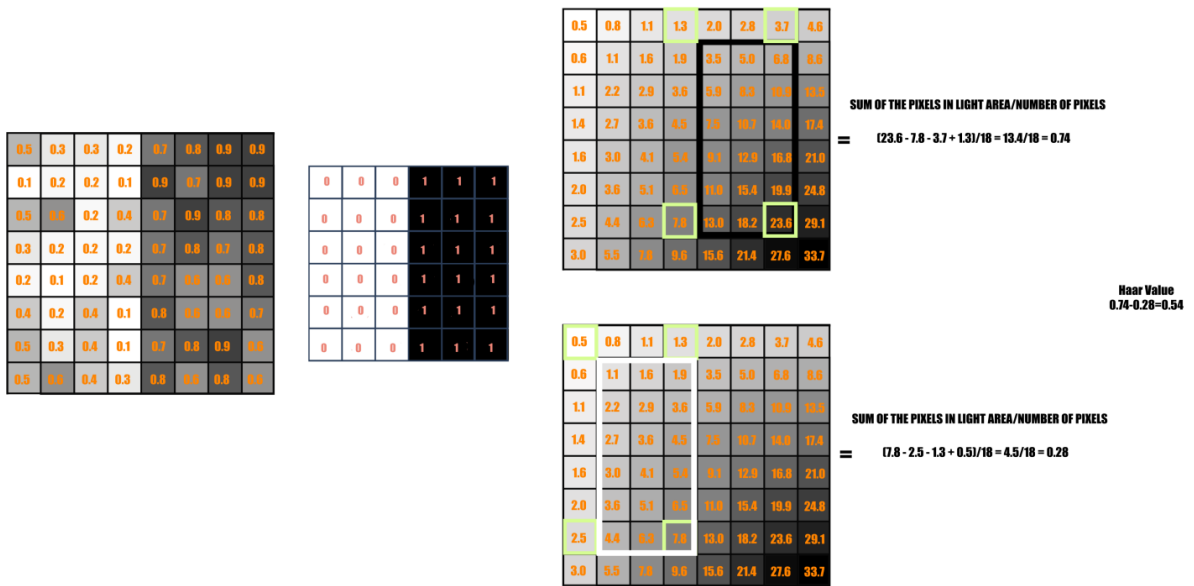


Fig (3.3) The Integral Image is used here to calculate the haar value.

With the Integral Image, only 4 constant value additions are needed each time for any feature size (with respect to the 18 additions earlier). This reduces the time complexity of each addition gradually, as the number of additions does not depend on the number of pixels enclosed anymore.

In the above image, there is no edge in the vertical direction as the haar value is -0.02, which is very far from 1. Let's see one more example, where there might be an edge present in the image.



Fig(3.4): The Haar calculation from Integral Image.

This is a case where there is a sudden change of pixel intensities moving vertically from the left towards the right in the image.

Again repeating the same calculation done above, but this time just to see what haar value is calculated when there is a sudden change of intensities moving from left to right in a vertical direction. The haar value here is 0.54, which is closer to 1 in comparison to the case earlier.

AdaBoost is a popular boosting technique which helps you combine multiple “weak classifiers” into a single “strong classifier”. A weak classifier is simply a classifier that performs poorly, but performs better than random guessing. A simple example might be classifying a person as male or female based on their height. You could say anyone over 5’ 9” is a male and anyone under that is a female. You’ll misclassify a lot of people that way, but your accuracy will still be greater than 50%.

AdaBoost can be applied to any classification algorithm, so it’s really a technique that builds on top of other classifiers as opposed to being a classifier itself.

What does AdaBoost do for you? There are really two things it figures out for you:

1. It helps you choose the training set for each new classifier that you train based on the results of the previous classifier.
2. It determines how much weight should be given to each classifier's proposed answer when combining the results.

Training Set Selection

Each weak classifier should be trained on a random subset of the total training set. The subsets can overlap—it's not the same as, for example, dividing the training set into ten portions. AdaBoost assigns a "weight" to each training example, which determines the probability that each example should appear in the training set. Examples with higher weights are more likely to be included in the training set, and vice versa. After training a classifier, AdaBoost increases the weight on the misclassified examples so that these examples will make up a larger part of the next classifier's training set, and hopefully the next classifier trained will perform better on them.

The equation for this weight update step is detailed later on.

Classifier Output Weights

After each classifier is trained, the classifier's weight is calculated based on its accuracy. More accurate classifiers are given more weight. A classifier with 50% accuracy is given a weight of zero, and a classifier with less than 50% accuracy (kind of a funny concept) is given negative weight.

Formal Definition

To learn about AdaBoost, I read through a tutorial written by one of the original authors of the algorithm, Robert Schapire. Let's look first at the equation for the final classifier.

$$H(x) = \text{sign} \left(\sum_{t=1}^T \alpha_t h_t(x) \right)$$

The final classifier consists of 'T' weak classifiers. $h_t(x)$ is the output of weak classifier 't' (in this paper, the outputs are limited to -1 or +1). α_t is the weight applied to classifier 't' as determined by AdaBoost. So the final output is just a linear combination of all of the weak classifiers, and then we make our final decision simply by looking at the sign of this sum.

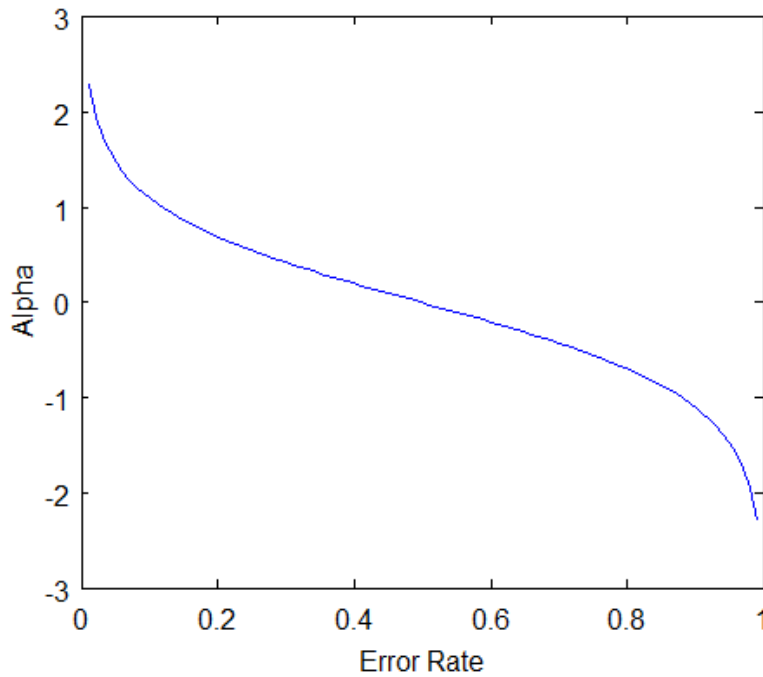
The classifiers are trained one at a time. After each classifier is trained, we update the probabilities of each of the training examples appearing in the training set for the next classifier.

The first classifier ($t = 1$) is trained with equal probability given to all training examples. After it's trained, we compute the output weight (α) for that classifier.

$$\alpha_t = \frac{1}{2} \ln \left(\frac{1 - \epsilon_t}{\epsilon_t} \right)$$

The output weight, α_t , is fairly straightforward. It's based on the classifier's error rate, ' ϵ_t '. ϵ_t is just the number of misclassifications over the training set divided by the training set size.

Here's a plot of what α_t will look like for classifiers with different error rates.



Figure(3.5) :shows the ratio of alpha and error rate of an image

There are three bits of intuition to take from this graph:

1. The classifier weight grows exponentially as the error approaches 0. Better classifiers are given exponentially more weight.
2. The classifier weight is zero if the error rate is 0.5. A classifier with 50% accuracy is no better than random guessing, so we ignore it.
3. The classifier weight grows exponentially negative as the error approaches 1. We give a negative weight to classifiers with worse worse than 50% accuracy. “Whatever that classifier says, do the opposite!”.

After computing the alpha for the first classifier, we update the training example weights using the following formula.

$$D_{t+1}(i) = \frac{D_t(i) \exp(-\alpha_t y_i h_t(x_i))}{Z_t}$$

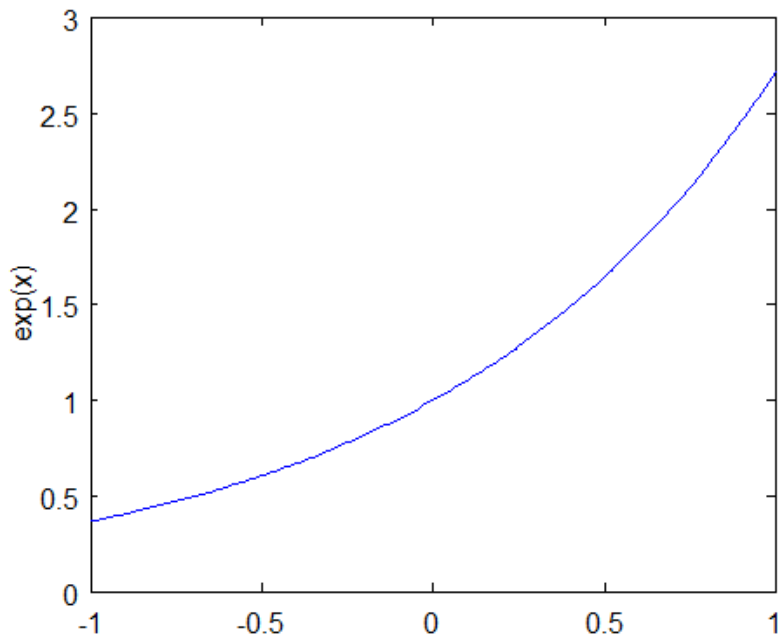
The variable D_t is a vector of weights, with one weight for each training example in the training set. 'i' is the training example number. This equation shows you how to update the weight for the i th training example.

The paper describes D_t as a distribution. This just means that each weight $D(i)$ represents the probability that training example i will be selected as part of the training set.

To make it a distribution, all of these probabilities should add up to 1. To ensure this, we normalize the weights by dividing each of them by the sum of all the weights, Z_t . So, for example, if all of the calculated weights added up to 12.2, then we would divide each of the weights by 12.2 so that they sum up to 1.0 instead.

This vector is updated for each new weak classifier that's trained. D_t refers to the weight vector used when training classifier 't'.

This equation needs to be evaluated for each of the training samples 'i' (x_i, y_i). Each weight from the previous training round is going to be scaled up or down by this exponential term.



Fig(3.6):The output weights

The function $\exp(x)$ will return a fraction for negative values of x , and a value greater than one for positive values of x . So the weight for training sample i will be either increased or decreased depending on the final sign of the term “ $-\alpha * y * h(x)$ ”. For binary classifiers whose output is constrained to either -1 or $+1$, the terms y and $h(x)$ only contribute to the sign and not the magnitude.

y_i is the correct output for training example ‘ i ’, and $h_t(x_i)$ is the predicted output by classifier t on this training example. If the predicted and actual output agree, $y * h(x)$ will always be $+1$ (either $1 * 1$ or $-1 * -1$). If they disagree, $y * h(x)$ will be negative.

Ultimately, misclassifications by a classifier with a positive α will cause this training example to be given a larger weight. And vice versa.

Note that by including α in this term, we are also incorporating the classifier’s effectiveness into consideration when updating the weights. If a weak classifier misclassifies an input, we don’t take that as seriously as a strong classifier’s mistake.

Practical Application

One of the biggest applications of AdaBoost that I’ve encountered is the Viola-Jones face detector, which seems to be the standard algorithm for detecting faces in an image. The Viola-Jones face detector uses a “rejection cascade” consisting of many layers of classifiers. If at any layer the detection window is `_not_` recognized as a face, it’s rejected and we move on to the next window. The first classifier in the cascade is designed to discard as many negative windows as possible with minimal computational cost.

In this context, AdaBoost actually has two roles. Each layer of the cascade is a strong classifier built out of a combination of weaker classifiers, as discussed here. However, the principles of AdaBoost are also used to find the best features to use in each layer of the cascade.

Implementing Cascading Classifiers

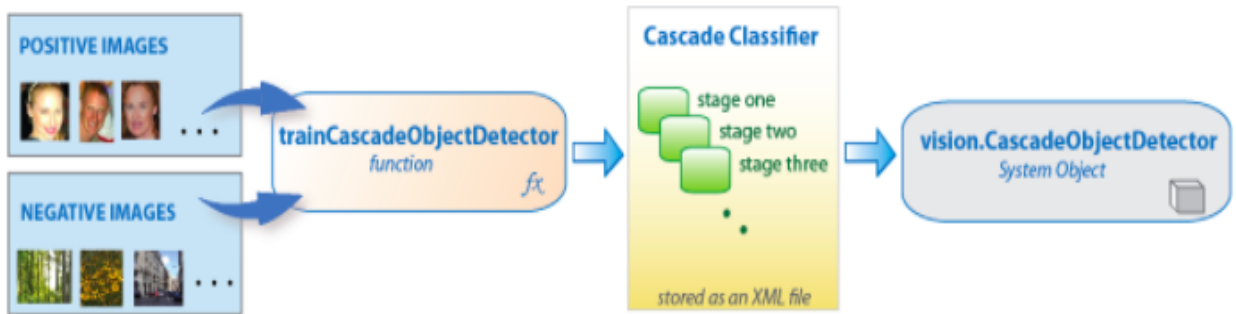


Fig (3.7): shows A flowchart of cascade classifiers.

The cascade classifier is made up of a series of stages, where each stage is a collection of weak learners. Weak learners are trained using boosting, which allows for a highly accurate classifier from the mean prediction of all weak learners.

Based on this prediction, the classifier either decides to indicate an object was found (positive) or move on to the next region (negative). Stages are designed to reject negative samples as fast as possible, because a majority of the windows do not contain anything of interest.

It's important to maximize a low false negative rate, because classifying an object as a non-object will severely impair your object detection algorithm. A video below shows Haar cascades in action. The red boxes denote "positives" from the weak learners.

Haar cascades are one of many algorithms that are currently being used for object detection. One thing to note about Haar cascades is that it is very important to reduce the false negative rate, so make sure to tune hyper parameters accordingly when training your model.

3.2 Single Shot Detector

The paper about 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

Architecture

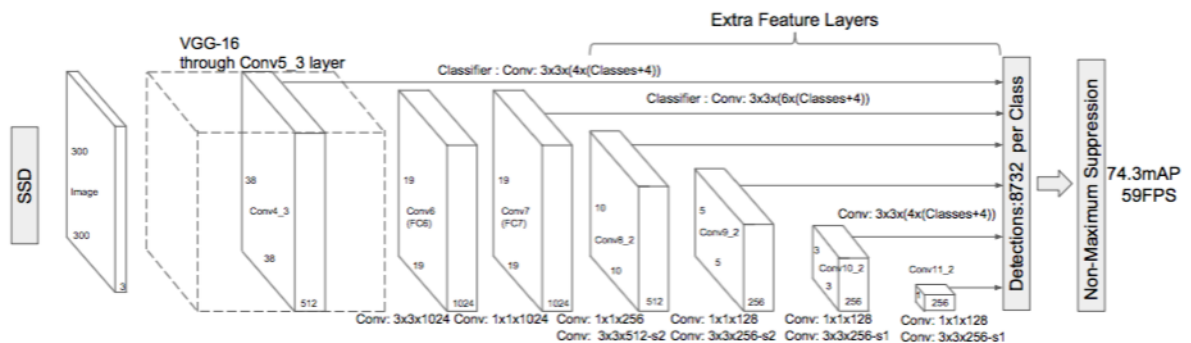


Fig (3.8): Architecture of Single Shot MultiBox detector (input is 300x300x3)

As you can see from the diagram above, SSD's architecture builds on the venerable VGG-16 architecture, but discards the fully connected layers. The reason VGG-16 was used as the *base network* is because of its strong performance in high quality image classification tasks and its popularity for problems where *transfer learning* helps in improving results. Instead of the original VGG fully connected layers, a set of *auxiliary* convolutional layers (from *conv6* onwards) were added, thus enabling to extract features at multiple scales and progressively decrease the size of the input to each subsequent layer.

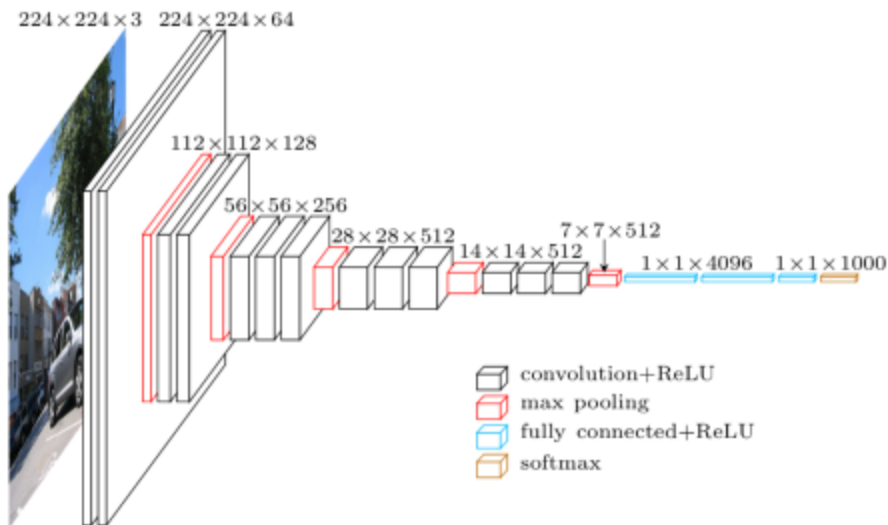


Fig (3.9): The VGG architecture (input is 224x224x3)

MultiBox

The bounding box regression technique of SSD is inspired by Szegedy’s work on MultiBox, a method for fast *class-agnostic* bounding box coordinate proposals. Interestingly, in the work done on MultiBox an Inception-style convolutional network is used. The 1x1 convolutions that you see below help in dimensionality reduction since the number of dimensions will go down (but “width” and “height” will remain the same).

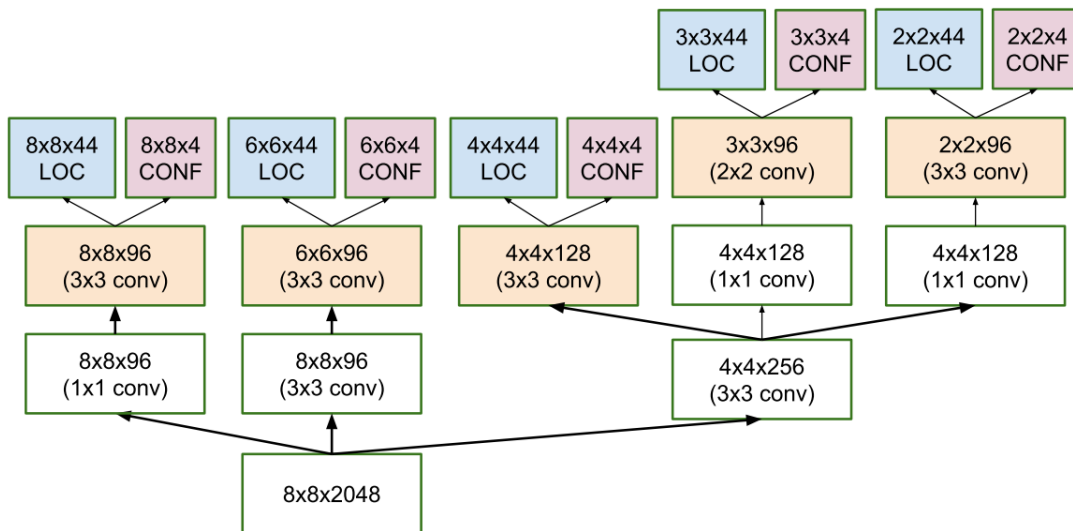


Fig (3.10): Architecture of multi-scale convolutional prediction of the location and confidences of multibox

MultiBox’s loss function also combined two critical components that made their way into SSD:

- **Confidence Loss:** this measures how confident the network is of the *objectness* of the computed bounding box. Categorical cross-entropy is used to compute this loss.
- **Location Loss:** this measures how *far away* the network’s predicted bounding boxes are from the ground truth ones from the training set. L2-Norm is used here.

Without delving too deep into the math (read the paper if you are curious and want a more rigorous notation), the expression for the loss, which measures how far off our prediction “landed”, is thus:

$$\text{multibox_loss} = \text{confidence_loss} + \alpha * \text{location_loss}$$

The *alpha* term helps us in balancing the contribution of the location loss. As usual in deep learning, the goal is to find the parameter values that most optimally reduce the loss function, thereby bringing our predictions closer to the ground truth.

MultiBox Priors AndIoU

The logic revolving around the bounding box generation is actually more complex than stated earlier. In MultiBox, the researchers created what we call *priors* (or *anchors* in Faster-R-CNN terminology), which are pre-computed, fixed size bounding boxes that closely match the distribution of the original ground truth boxes. In fact those *priors* are selected in such a way that their Intersection over Union ratio (aka IoU, and sometimes referred to as Jaccard index) is greater than 0.5. As you can infer from the image below, an IoU of 0.5 is still not good enough but it does however provide a strong starting point for the bounding box regression algorithm — it is a much better strategy than starting the predictions with random coordinates! **Therefore MultiBox starts with the priors as predictions and attempt to regress closer to the ground truth bounding boxes.**

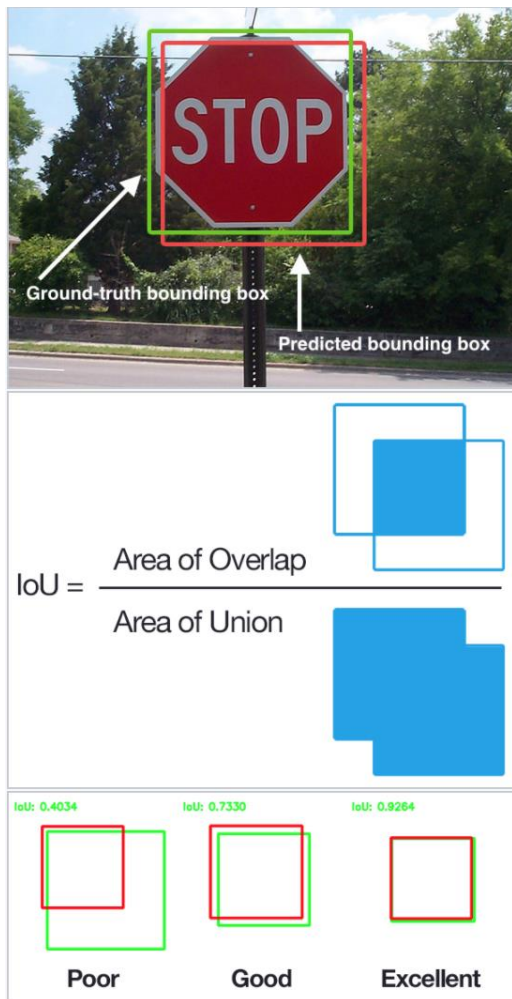


Fig (3.11): Diagram explaining IoU

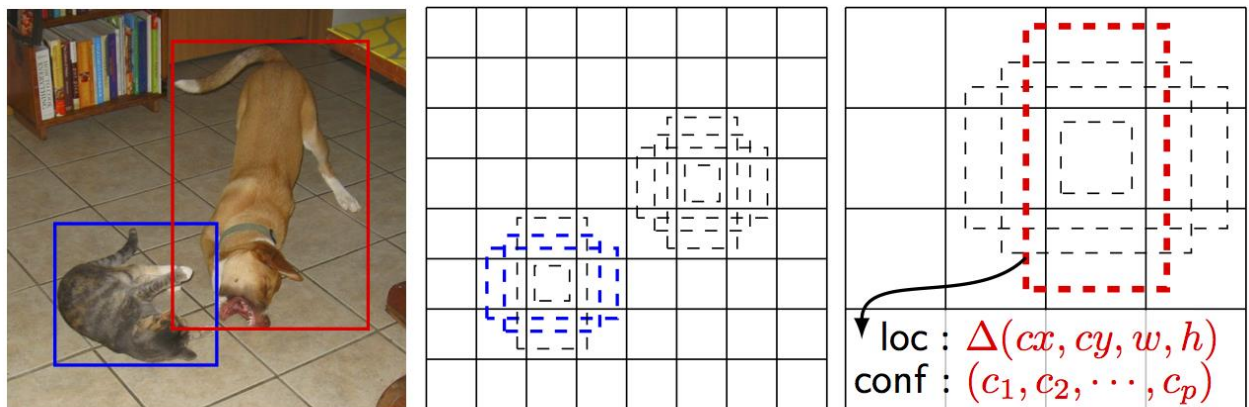
The resulting architecture (check MultiBox architecture diagram above again for reference) contains 11 priors per feature map cell (8x8, 6x6, 4x4, 3x3, 2x2) and only one on the 1x1 feature map, resulting in a total of 1420 priors per image, thus enabling robust coverage of input images at multiple scales, to detect objects of various sizes.

At the end, MultiBox only retains the top K predictions that have minimised both location (*LOC*) and confidence (*CONF*) losses.

SSD Improvements

Back onto SSD, a number of tweaks were added to make this network even more capable of localizing and classifying objects.

Fixed Priors: unlike MultiBox, every feature map cell is associated with a set of default bounding boxes of different dimensions and aspect ratios. These priors are manually (but carefully) chosen, whereas in MultiBox, they were chosen because their IoU with respect to the ground truth was over 0.5. This in theory should allow SSD to generalise for any type of input, without requiring a pre-training phase for prior generation. For instance, assuming we have configured 2 diagonally opposed points $(x1, y1)$ and $(x2, y2)$ for each b default bounding boxes per feature map cell, and c classes to classify, on a given feature map of size $f = m * n$, SSD would compute $f * b * (4 + c)$ values for this feature map.



(a) Image with GT boxes (b) 8×8 feature map (c) 4×4 feature map
Fig (3.12): shows the SSD default boxes at 8x8 and 4x4 feature maps

Location Loss: SSD uses smooth L1-Norm to calculate the location loss. While not as precise as L2-Norm, it is still highly effective and gives SSD more room for maneuver as it does not try to be “pixel perfect” in its bounding box prediction (i.e. a difference of a few pixels would hardly be noticeable for many of us).

Classification: MultiBox does not perform object classification, whereas SSD does. Therefore, for each predicted bounding box, a set of c class predictions are computed, for every possible class in the dataset.

Training & Running SSD

Datasets

You will need training and test datasets with ground truth bounding boxes and assigned class labels (only one per bounding box). The Pascal VOC and COCO datasets are a good starting point.

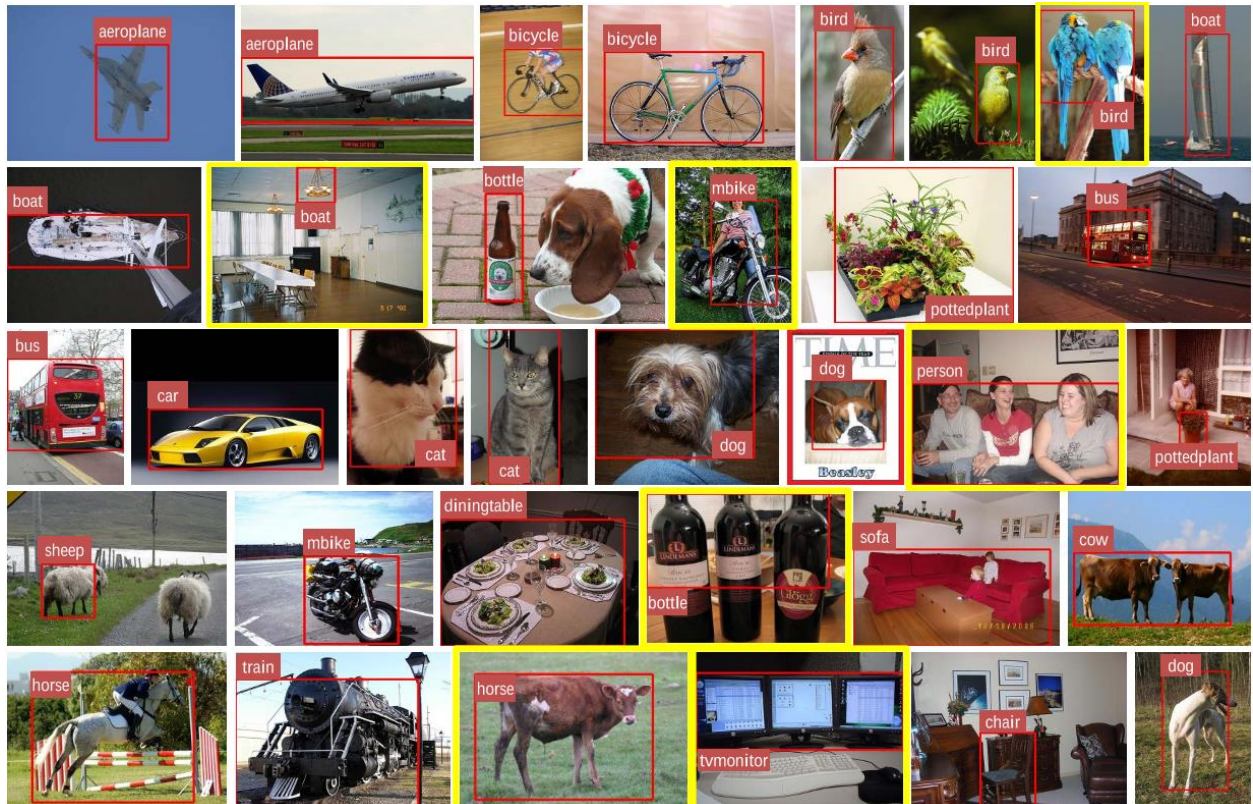


Fig (3.13) : Images from Pascal VOC dataset

Default Bounding Boxes

It is recommended to configure a varied set of default bounding boxes, of different scales and aspect ratios to ensure most objects could be captured. The SSD paper has around 6 bounding boxes per feature map cell.

Feature Maps

Features maps (i.e. the results of the convolutional blocks) are a representation of the dominant features of the image at different scales, therefore running MultiBox on multiple feature maps increases the likelihood of any object (large and small) to be eventually detected, localized and

appropriately classified. The image below shows how the network “sees” a given image across its feature maps:

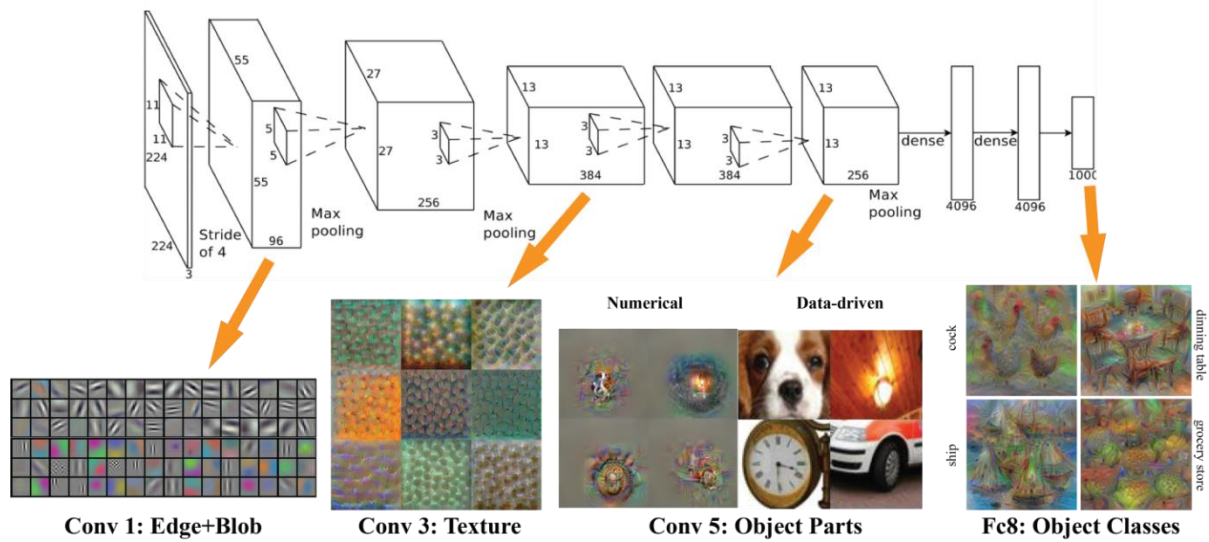


Fig (3.14): VGG Feature Map Visualization

Hard Negative Mining

During training, as most of the bounding boxes will have low IoU and therefore be interpreted as *negative* training examples, we may end up with a disproportionate amount of negative examples in our training set. Therefore, instead of using all negative predictions, it is advised to keep a ratio of negative to positive examples of around 3:1. The reason why you need to keep negative samples is because the network also needs to learn and be explicitly told what constitutes an incorrect detection.

Model summary	minival mAP	test-dev mAP
(Fastest) SSD w/MobileNet	19.3	18.8
SSD w/Inception V2	22	21.6
Faster R-CNN w/Resnet 101, 100 Proposals	32	31.9
R-FCN w/Resnet 101, 300 Proposals	30.4	30.3
Faster R-CNN w/Inception Resnet V2, 300 Proposals	35.7	35.6

Table 3.1: Comparison of performance of various models

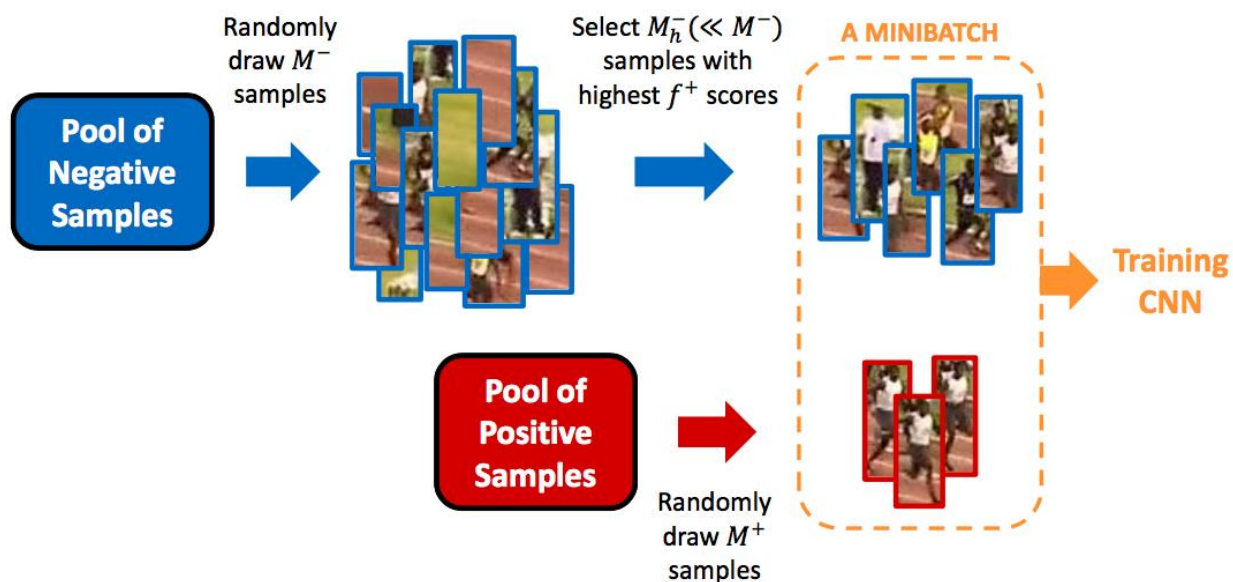
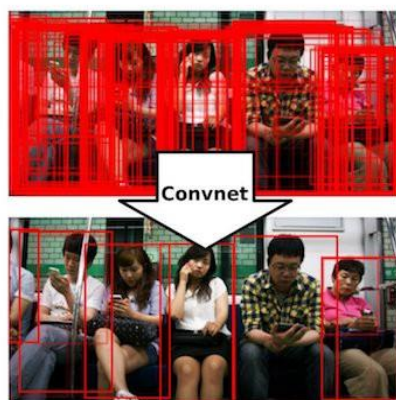


Fig (3.15): Example of hard negative mining

Non-Maximum Suppression (NMS)

Given the large number of boxes generated during a forward pass of SSD at inference time, it is essential to prune most of the bounding boxes by applying a technique known as *non-maximum suppression*: boxes with a confidence loss threshold less than ct (e.g. 0.01) and IoU less than lt (e.g. 0.45) are discarded, and only the top N predictions are kept. This ensures only the most likely predictions are retained by the network, while the more noisier ones are removed.



Fig(3.16) : NMS example

Chapter 4

Web server using Django

4.1 Introduction and Installation

Django is a Python-based web framework which allows you to quickly create web application without all of the installation or dependency problems that you normally will find with other frameworks. When you're building a website, you always need a similar set of components: a way to handle user authentication (signing up, signing in, signing out), a management panel for your website, forms, a way to upload files, etc. Django gives you ready-made components to use.

Why Django?

- Django is a rapid web development framework that can be used to develop fully fleshed web applications in a short period of time.
- It's very easy to switch database in Django framework.
- It has built-in admin interface which makes easy to work with it.
- Django is fully functional framework that requires nothing else.
- It has thousands of additional packages available.
- It is very scalable.

Django architecture

Django is based on MVT (Model-View-Template) architecture. MVT is a software design pattern for developing a web application.

MVT Structure has the following three parts –

Model: Model is going to act as the interface of your data. It is responsible for maintaining data. It is the logical data structure behind the entire application and is represented by a database (generally relational databases such as MySQL, Postgres).

View: The View is the user interface — what you see in your browser when you render a website. It is represented by HTML/CSS/Javascript and Jinja files.

Template: A template consists of static parts of the desired HTML output as well as some special syntax describing how dynamic content will be inserted.

Installation of Django

- Install python3 if not installed in your system (according to configuration of your system and OS) from here. Try to download the latest version of python its python3.6.4 this time. **Note-** Installation of Django in Linux and Mac is similar, here I am showing it in windows for Linux and mac just open terminal in place of command prompt and go through the following commands.
- **Install pip-** Open command prompt and enter following command-
python -m pip install -U pip
- **Install virtual environment-** Enter following command in cmd-
pip install virtualenv
- **Set Virtual environment-** Setting up the virtual environment will allow you to edit the dependency which generally your system wouldn't allow. Follow these steps to set up a virtual environment-
 1. Create a virtual environment by giving this command in cmd-
virtualenv env_site
 2. Change directory to env_site by this command-
cd env_site
 3. Go to Script directory inside env_site and activate virtual environment-
cd Script

activate
- **Install Django-** Install django by giving following command-
pip install django

Creating a Project

Let's check how to create a basic project using Django after you have installed it in your pc.

- To initiate a project of Django on Your PC, open Terminal and Enter the following command
django-admin startproject projectName

- A New Folder with name projectName will be created. To enter in the project using terminal enter command

`cd projectName`

Now run

Python manage.py runserver

Now visit <http://localhost:8000/>,

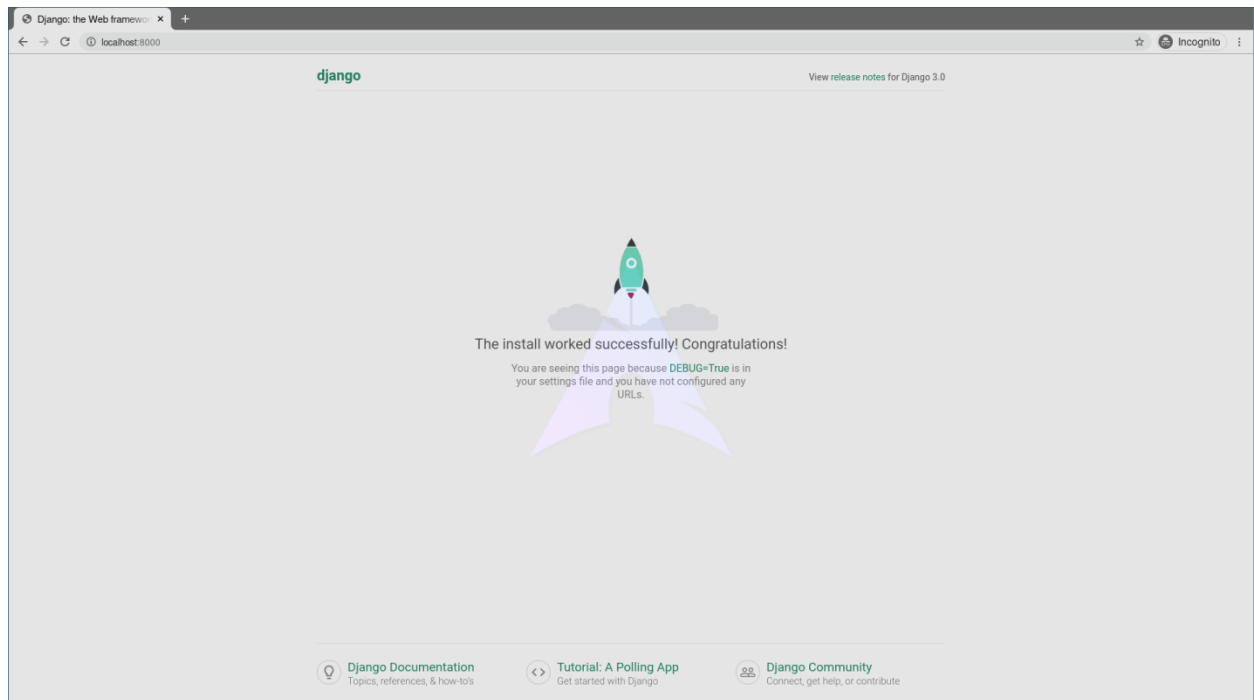


Fig4.1 :server of Django

Creating an App

Django is famous for its unique and fully managed app structure. For every functionality, an app can be created like a completely independent module. This article will take you through how to create a basic app and add functionalities using that app.

- To create a basic app in your Django project you need to go to directory containing manage.py and from there enter the command :

`python manage.py startappprojectApp`

Now you can see your directory structure as under :

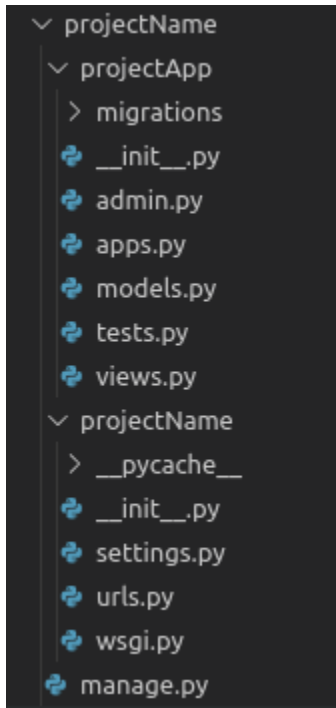


Fig (4.2): Shows the installed apps

To consider the app in your project you need to specify your project name in INSTALLED_APPS list as follows in settings.py:

```
# Application definition
```

```
INSTALLED_APPS = [  
  
    'django.contrib.admin',  
  
    'django.contrib.auth',  
  
    'django.contrib.contenttypes',  
  
    'django.contrib.sessions',  
  
    'django.contrib.messages',  
  
    'django.contrib.staticfiles',
```

```
        'projectApp'  
    ]
```

So, we have finally created an app but to render the app using urls we need to include the app in our main project so that urls redirected to that app can be rendered. Let us explore it. Move to projectName->projectName -> urls.py and add below code in the header

```
from django.urls import include
```

Now in the list of URL patterns, you need to specify app name for including your app urls. Here is the code for it –

```
from django.contrib import admin
```

```
from django.urls import path, include
```

```
urlpatterns = [
```

```
    path('admin/', admin.site.urls),
```

```
    # Enter the app name in following syntax for this to work
```

```
    path("", include("projectApp.urls")),
```

```
]
```

4.2 Views in Django

Django Views are one of the vital participants of MVT Structure of Django. A View is the user interface — what you see in your browser when you render a website. It is represented by HTML/CSS/Javascript and Jinja files. As per Django Documentation, A view function is a Python function that takes a Web request and returns a Web response. This **response** can be the HTML contents of a Web page, or a redirect, or a 404 error, or an XML document, or an image, anything that a web browser can display.

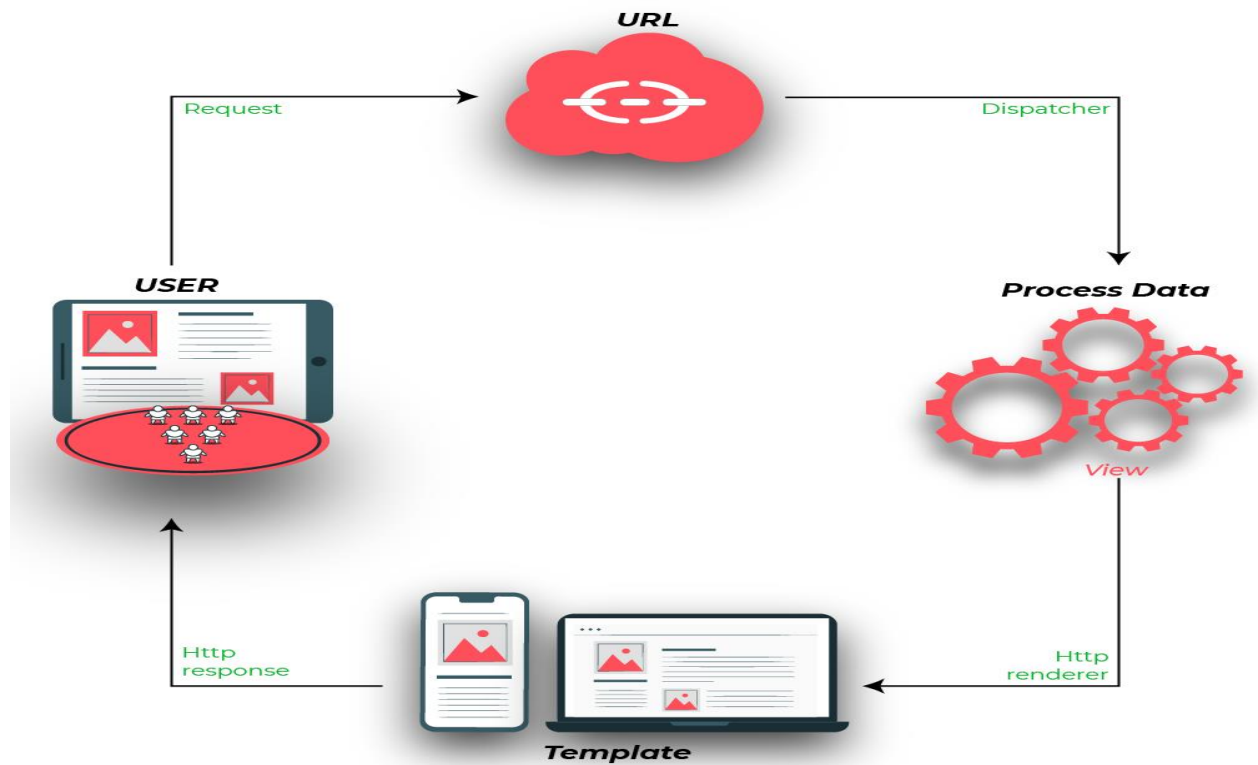


Fig (4.3): shows the normal view of Django

Django View Example

Illustration of **How to create and use a Django view** using an Example. Consider a project named geeksforgeeks having an app named geeks.

After you have a project ready, we can create a view in geeks/views.py,

```
# import Http Response from django
from django.http import HttpResponse

# get datetime
import datetime

# create a function
def geeks_view(request):
```

```
# fetch date and time

now = datetime.datetime.now()

# convert to string

html = "Time is {}".format(now)

# return response

return HttpResponse(html)
```

Let's step through this code one line at a time:

- First, we import the class **HttpResponse** from the `django.http` module, along with Python's `datetime` library.
- Next, we define a function called `geeks_view`. This is the view function. Each view function takes an `HttpRequest` object as its first parameter, which is typically named `request`.
- The view returns an `HttpResponse` object that contains the generated response. Each view function is responsible for returning an **HttpResponse** object.

Let's get this view to working, in `geeks/urls.py`,

```
from django.urls import path

# importing views from views..py

from .views import geeks_view

urlpatterns = [

    path("", geeks_view),

]
```

Types of Views

Django views are divided into two major categories :-

- Function Based Views
- Class Based Views

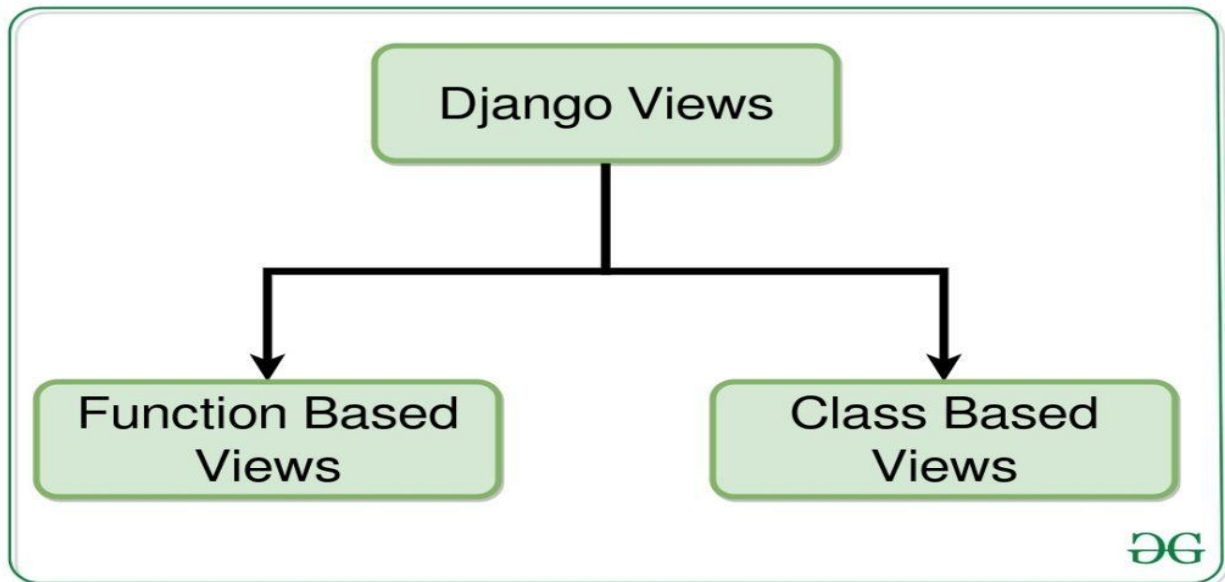


Fig (4.4): shows the classification of Django views

Function Based Views

Function based views are written using a function in python which receives as an argument `HttpRequest` object and returns an `HttpResponse` Object. Function based views are generally divided into 4 basic strategies, i.e., CRUD (Create, Retrieve, Update, Delete). CRUD is the base of any framework one is using for development.

Function based view Example –

Let's Create a function based view list view to display instances of a model. let's create a model of which we will be creating instances through our view. In `geeks/models.py`,

```
# import the standard Django Model
# from built-in library
from django.db import models

# declare a new model with a name "GeeksModel"
class GeeksModel(models.Model):
```



```

# fields of the model

title = models.CharField(max_length = 200)

description = models.TextField()

# renames the instances of the model

# with their title name

def __str__(self):

    return self.title

```

After creating this model, we need to run two commands in order to create Database for the same.

```

Python manage.py makemigrations
Python manage.py migrate

```

Now if you want to see your model and its data in the admin panel, then you need to register your model. Let's register this model. In geeks/admin.py,

```

from django.contrib import admin

from .models import GeeksModel

# Register your models here.

admin.site.register(GeeksModel)

```

Now we have everything ready for back end. Verify that instances have been created from <http://localhost:8000/admin/geeks/geeksmodel/>

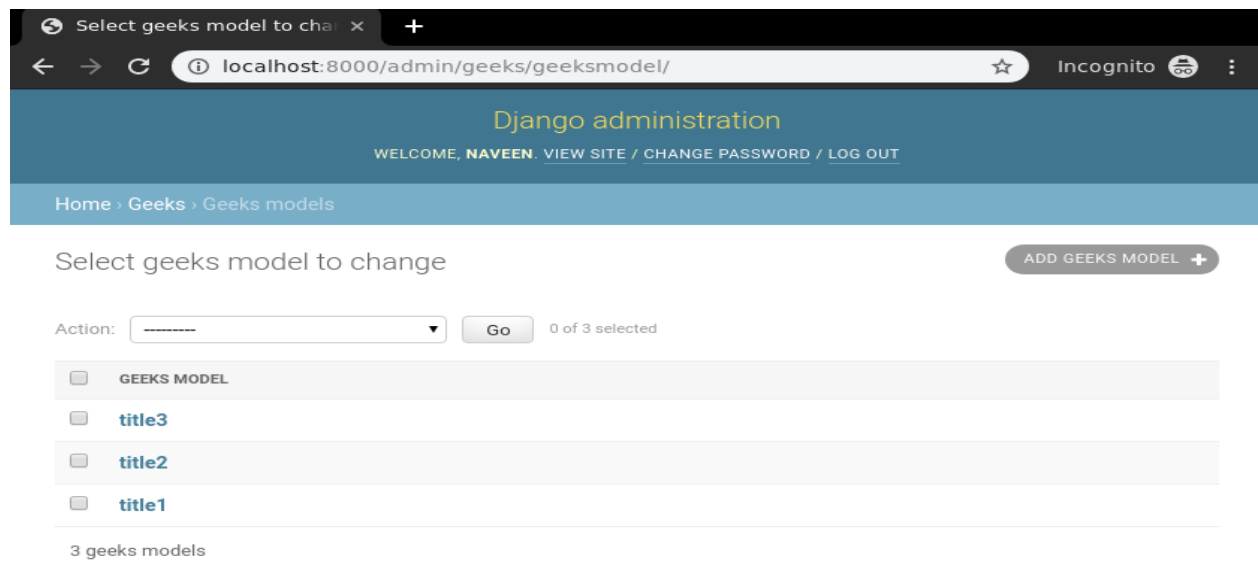


Fig (4.5): shows the template for the django

Let's create a view and template for the same. In `geeks/views.py`,

```
from django.shortcuts import render
```

```
# relative import of forms
```

```
from .models import GeeksModel
```

```
def list_view(request):
```

```
    # dictionary for initial data with
```

```
    # field names as keys
```

```
    context = { }
```

```
    # add the dictionary during initialization
```

```
    context["dataset"] = GeeksModel.objects.all()
```

```
    return render(request, "list_view.html", context)
```

Create a template in templates/list_view.html,

```
<div class="main">

    {% for data in dataset %}.

    {{ data.title }}<br/>
    {{ data.description }}<br/>
    <hr/>

    {% endfor %}

</div>
```

Class Based Views

Class-based views provide an alternative way to implement views as Python objects instead of functions. They do not replace function-based views, but have certain differences and advantages when compared to function-based views:

- Organization of code related to specific HTTP methods (GET, POST, etc.) can be addressed by separate methods instead of conditional branching.
- Object oriented techniques such as mixins (multiple inheritance) can be used to factor code into reusable components.

Class-based views are simpler and efficient to manage than function-based views. A function-based view with tons of lines of code can be converted into a class-based views with few lines only. This is where Object-Oriented Programming comes into impact.

Class based view Example –

In geeks/views.py,

```
from django.views.generic.list import ListView
from .models import GeeksModel
class GeeksList(ListView):
    # specify the model for list view
    model = GeeksModel
```

Now create a URL path to map the view. In `geeks/urls.py`,
from `django.urls` import `path`

```
# importing views from views..py
from .views import GeeksList
urlpatterns = [
    path("", GeeksList.as_view()),
]
```

Create a template in `templates/geeks/geeksmodel_list.html`,

```
<ul>
    <!-- Iterate over object_list -->
    {% for object in object_list %}
    <!-- Display Objects -->
    <li>{{ object.title }}</li>
    <li>{{ object.description }}</li>

    <hr/>
    <!-- If object_list is empty -->
    {% empty %}
    <li>No objects yet.</li>
    {% endfor %}
</ul>
```

4.3 Project MVT Structure

Django is based on **MVT (Model-View-Template)** architecture. MVT is a software design pattern for developing a web application.

MVT Structure has the following three parts –

Model: Model is going to act as the interface of your data. It is responsible for maintaining data. It is the logical data structure behind the entire application and is represented by a database (generally relational databases such as MySQL, Postgres).

View: The View is the user interface — what you see in your browser when you render a website. It is represented by HTML/CSS/Javascript and Jinja files.

Template: A template consists of static parts of the desired HTML output as well as some special syntax describing how dynamic content will be inserted.

Project Structure :

A Django Project when initialised contains basic files by default such as `manage.py`, `view.py`, etc. A simple project structure is enough to create a single page application. Here are the major files and there explanations. Inside the `geeks_site` folder (project folder) there will be following files-

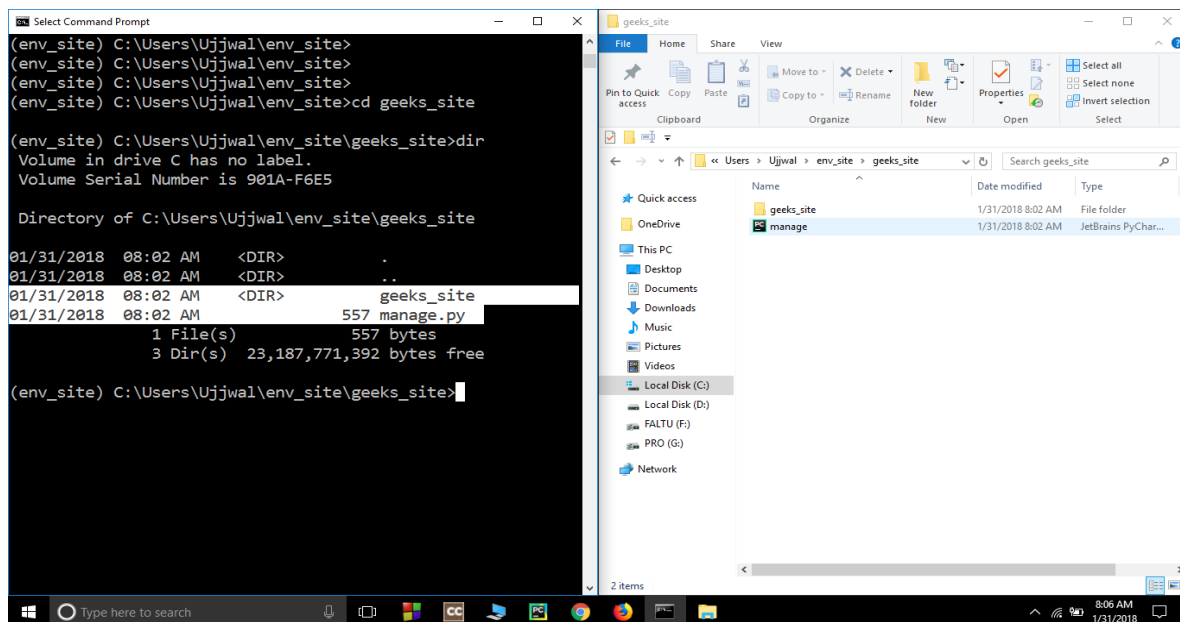


Fig (4.6): shows the project structure

manage.py-This file is used to interact with your project via the command line(start the server, sync the database... etc). For getting the full list of command that can be executed by manage.py type this code in the command window-

```
$ python manage.py help
```

folder (geeks_site) – This folder contains all the packages of your project. Initially it contains four files –

- **_init_.py** – It is python package.
- **settings.py** – As the name indicates it contains all the website settings. In this file we register any applications we create, the location of our static files, database configuration details, etc.
- **urls.py** – In this file we store all links of the project and functions to call.
- **wsgi.py** – This file is used in deploying the project in WSGI. It is used to help your Django application communicate with the web server.

Chapter 5

Hardware Model

5.1 Components

Raspberry Pi:

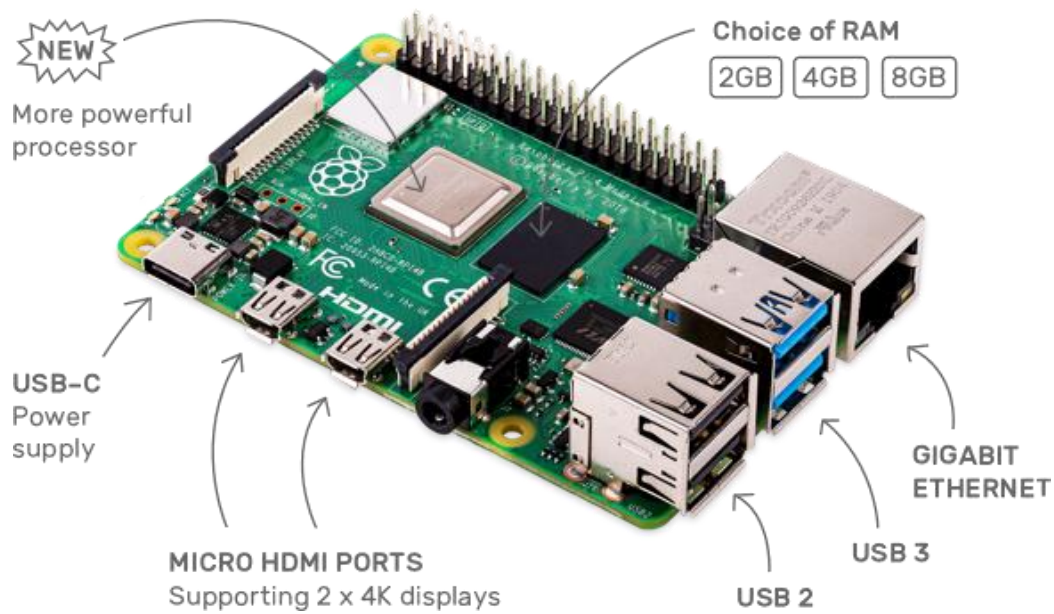


Fig (5.1): shows the raspberry pi (hardware tool used in project)

Raspberry Pi a series of small single-board computers developed in the United Kingdom by the Raspberry Pi Foundation in association with Broadcom. The Raspberry Pi project originally leaned towards the promotion of teaching basic computer science in schools and in developing countries. The original model became more popular than anticipated, selling outside its target market for uses such as robotics. It is widely used in many areas, such as for weather monitoring, because of its low cost, modularity, and open design. It is typically used by computer and electronic hobbyists, due to its adoption of HDMI and USB devices.

Specifications

- Broadcom BCM2711, Quad core Cortex-A72 (ARM v8) 64-bit SoC @ 1.5GHz
- 2GB, 4GB or 8GB LPDDR4-3200 SDRAM (depending on model)

- 2.4 GHz and 5.0 GHz IEEE 802.11ac wireless, Bluetooth 5.0, BLE
- Gigabit Ethernet
- 2 USB 3.0 ports; 2 USB 2.0 ports.
- Raspberry Pi standard 40 pin GPIO header (fully backwards compatible with previous boards)
- 2 × micro-HDMI ports (up to 4k60 supported)
- 2-lane MIPI DSI display port
- 2-lane MIPI CSI camera port
- 4-pole stereo audio and composite video port
- H.265 (4k60 decode), H264 (1080p60 decode, 1080p30 encode)
- OpenGL ES 3.1, Vulkan 1.0
- Micro-SD card slot for loading operating system and data storage
- 5V DC via USB-C connector (minimum 3A*)
- 5V DC via GPIO header (minimum 3A*)
- Power over Ethernet (PoE) enabled (requires separate PoE HAT)
- Operating temperature: 0 – 50 degrees C ambient

* A good quality 2.5A power supply can be used if downstream USB peripherals consume less than 500mA in total.

Camera Module:

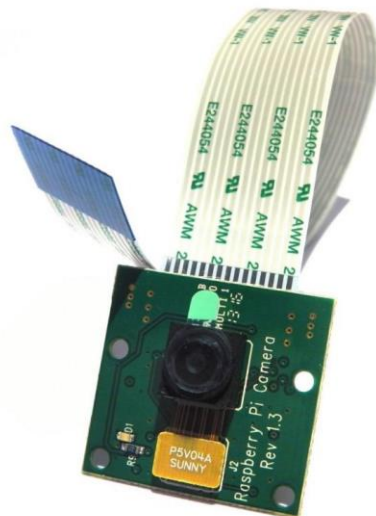


Fig :(5.2): shows the camera module connector on Raspberry pi

The Raspberry Pi Camera Board plugs directly into the CSI connector on the Raspberry Pi. It's able to deliver a crystal clear 5MP resolution image or 1080p HD video recording at 30fps! Latest Version 1.3! Custom designed and manufactured by the Raspberry Pi Foundation in the UK, the Raspberry Pi Camera Board features a 5MP (2592x1944 pixels) Omnivision 5647 sensor in a fixed focus module. The module attaches to Raspberry Pi, by way of a 15 Pin Ribbon Cable, to the dedicated 15-pin MIPI Camera Serial Interface (CSI), which was designed especially for interfacing to cameras. The CSI bus is capable of extremely high data rates, and it exclusively carries pixel data to the BCM2835 processor. The board itself is tiny, at around 25mm x 20mm x 9mm, and weighs just over 3g, making it perfect for mobile or other applications where size and weight are important. The sensor itself has a native resolution of 5 megapixels, and has a fixed focus lens onboard. In terms of still images, the camera is capable of 2592 x 1944 pixel static images, and also supports 1080p @ 30fps, 720p @ 60fps and 640x480p 60/90 video recording. The camera is supported in the latest version of Raspbian, the Raspberry Pi's preferred operating system.

The Raspberry Pi Camera Board Features:

- Fully Compatible with Both the Model A and Model B Raspberry Pi
- 5MP Omnivision 5647 Camera Module
- Still Picture Resolution: 2592 x 1944
- Video: Supports 1080p @ 30fps, 720p @ 60fps and 640x480p 60/90 Recording
- 15-pin MIPI Camera Serial Interface - Plugs Directly into the Raspberry Pi Board
- Size: 20 x 25 x 9mm
- Weight 3g

Audio Output and Power Supply:

The audio output is given through regular bluetooth headset or a speaker. The power supply is provided through a power bank using a USB type A to USB type C cable.



Fig (5.3): A Power Bank

5.2 Implementation

- The Image is captured using Camera Module (Pi camera).
- Image is transferred to Raspberry Pi and it is analyzed using **Haar Cascade** and **SSD model**.
- Recognized Objects in the image are Output is given via Audio through a speaker or bluetooth headset.
- More automation assistant features are added like object searching, answering questions etc.
- Image Data is transferred to Data Server and Stored for further uses.

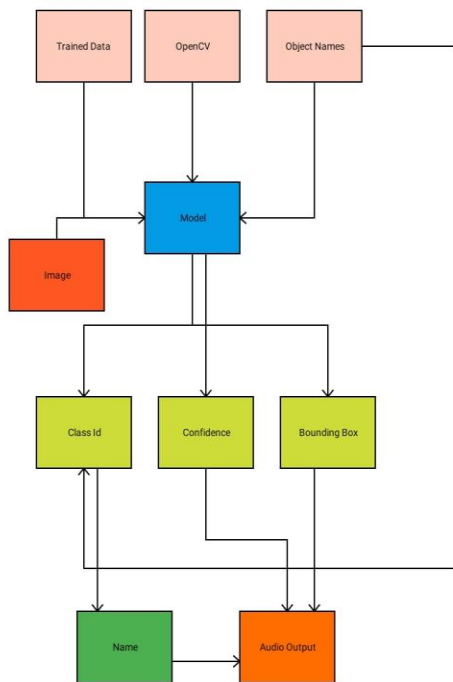


Fig (5.4): the Flowchart/procedure of program

The images that are captured are transferred to Raspberry pi and it is analyzed using Haar Cascade and SSD Model. Then the recognized objects are given to the user via audio through a

speaker or bluetooth headset. The image data is transferred to Data Server for further use. Initially, an integral image that allows very fast feature evaluation can be computed from an image using few operations per pixel. The integral image is calculated from the original image in such a way that each pixel in this is equal to the sum of all the pixels lying in its left and above in the original image.

Now a method is taken for constructing a classifier by selecting a small number of important features using AdaBoost. We use AdaBoost which selects the best features and trains the classifiers that use them. During a detection phase, a window of the target is moved over the input image and each subsection of the image and Haar features are calculated. The difference is then compared to a learned threshold that separates non-objects from objects.

At last a cascade structure for combining successively more complex classifiers which increases the speed of detector by focusing more on promising regions of the image is taken. It consists a multiple stages, each stage is trained using a technique called Boosting. Boosting provides the ability to train a highly accurate classifier by taking a weighted average of the decisions made by the weak learners.

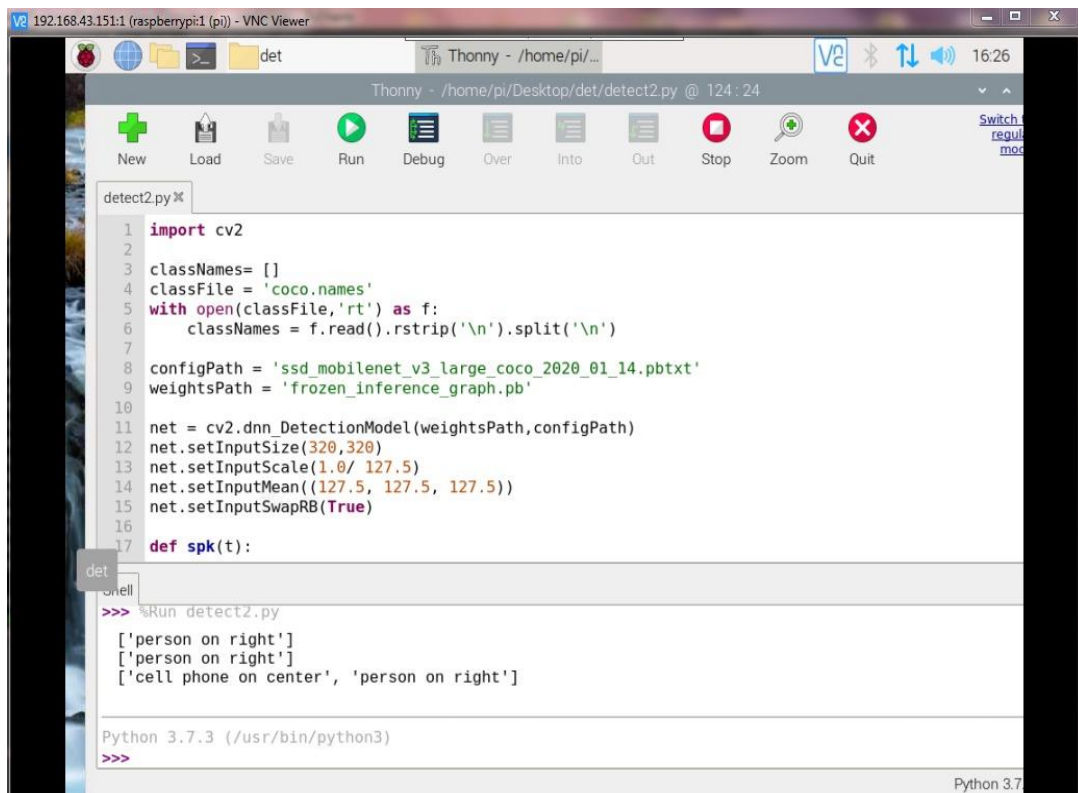
Results and Discussions

There are some key takeaways from the above discussed chapters. Before concluding the report, we need to mention those points from each and every chapter. The key points we have obtained from the project are:

- OpenCV-Python is a library of Python bindings designed to solve computer vision problems. Compared to languages like C/C++, Python is slower. That said, Python can be easily extended with C/C++, which allows us to write computationally intensive code in C/C++ and create Python wrappers that can be used as Python modules. This gives us two advantages: first, the code is as fast as the original C/C++ code (since it is the actual C++ code working in background) and second, it is easier to code in Python than C/C++. OpenCV-Python is a Python wrapper for the original OpenCV C++ implementation.
- Haarcascade features are used for knowing the qualities of an object by which objects can be recognized like shape, size, and edges. These can be formed by locating unique pixels of the objects where darker and lighter pixels locate next to each other forming a feature. Likewise, all features of the object are detected by running a rectangular window over the image. By selecting the best features from all the features we can detect the object by matching them with these features.
- SSD's architecture builds on the venerable VGG-16 architecture, but discards the fully connected layers. The reason VGG-16 was used as the *base network* is because of its strong performance in high quality image classification tasks and its popularity for problems where *transfer learning* helps in improving results. Instead of the original VGG fully connected layers, a set of *auxiliary* convolutional layers (from *conv6* onwards) were added, thus enabling to extract features at multiple scales and progressively decrease the size of the input to each subsequent layer.
- Django is a Python-based web framework which allows you to quickly create web application without all of the installation or dependency problems that you normally will find with other frameworks. When you're building a website, you always need a similar set of components: a way to handle user authentication (signing up, signing in,

signing out), a management panel for your website, forms, a way to upload files, etc. Django gives you ready-made components to use.

- Raspberry Pi is a series of small single-board computers It is widely used in many areas, such as for weather monitoring, because of its low cost, modularity, and open design. It is typically used by computer and electronic hobbyists, due to its adoption of HDMI and USB devices.
- The images that are captured are transformed to Raspberry pi and it is analyzed using Haar Cascade and SSD Model. Then the recognized objects are given to the user via audio through a speaker or bluetooth headset. The image data is transferred to Data Server for further use. Image is processed at 0.3 seconds per image, hence 200 images can be analyzed per minute.



The screenshot shows a Thonny IDE window titled 'Thonny - /home/pi/Desktop/det/detect2.py @ 124:24'. The code in the editor is as follows:

```
1 import cv2
2
3 classNames= []
4 classFile = 'coco.names'
5 with open(classFile,'rt') as f:
6     classNames = f.read().rstrip('\n').split('\n')
7
8 configPath = 'ssd_mobilenet_v3_large_coco_2020_01_14.pbtxt'
9 weightsPath = 'frozen_inference_graph.pb'
10
11 net = cv2.dnn_DetectionModel(weightsPath,configPath)
12 net.setInputSize(320,320)
13 net.setInputScale(1.0/ 127.5)
14 net.setInputMean((127.5, 127.5, 127.5))
15 net.setInputSwapRB(True)
16
17 def spk(t):
```

The console output shows the following text:

```
>>> %Run detect2.py
['person on right']
['person on right']
['cell phone on center', 'person on right']
```

The bottom of the window shows 'Python 3.7.3 (/usr/bin/python3)' and a prompt '>>>'.

Fig 6.1: Output in text

CONCLUSION

To conclude with, we have proposed an assistive system for Visually Impaired group of people in the society, which helps them recognize objects. Haar Cascade along with SSD Model is found to be better in terms of accuracy and speed as compared to other approaches. Apart from simulation on Python Programming, we have also done hardware implementation using Raspberry Pi board, and other hardware equipments and hardware implementations are done successfully. The processing time for Object recognition module is very appropriate for this application. The Haar Cascade along with SSD Model object recognition algorithm is highly optimized and hence the processing time can be improved using hardware such as FPGA, DSP, and GPU etc. Generic object recognition to aid visually impaired people is in its very early stage of development, and has a very bright future; it is a great service for humanity to develop such a system.

Our future work includes, searching objects in real time through voice commands, date and time information through voice commands, phone call or sending SMS through voice, Text recognition etc.

REFERENCES

1. Docs, OpenCV. Cascade Classifier Training
2. <https://cocodataset.org/#home>
3. <https://github.com/opencv/opencv/wiki/TensorFlow-Object-Detection-API>
4. Docs, Raspberry Pi. Working with GPIO Pins
5. Viola, Paul & Jones, Michael. (2001). Rapid Object Detection using a Boosted Cascade of Simple Features. *IEEE Conf Comput Vis Pattern Recognit.* 1. I-511. 10.1109/CVPR.2001.990517.
6. Lin, T. Y., Maire, M., Belongie, S., Hays, J., Perona, P., Ramanan, D., ... & Zitnick, C. L. (2014, September). Microsoft coco: Common objects in context. In *European conference on computer vision* (pp. 740-755). Springer, Cham.
7. Liu, W., Anguelov, D., Erhan, D., Szegedy, C., Reed, S., Fu, C. Y., & Berg, A. C. (2016, October). Ssd: Single shot multibox detector. In *European conference on computer vision* (pp. 21-37). Springer, Cham.
8. Patterson, G., & Hays, J. (2016, October). Coco attributes: Attributes for people, animals, and objects. In *European Conference on Computer Vision* (pp. 85-100). Springer, Cham.
9. Howard, A. G., Zhu, M., Chen, B., Kalenichenko, D., Wang, W., Weyand, T., ... & Adam, H. (2017). Mobilenets: Efficient convolutional neural networks for mobile vision applications. *arXiv preprint arXiv:1704.04861*.
10. Jeong, J., Park, H., & Kwak, N. (2017). Enhancement of SSD by concatenating feature maps for object detection. *arXiv preprint arXiv:1705.09587*.
11. Mathworks, Mathworks. "Train a Cascade Object Detector" *Mathworks*, 2017, www.mathworks.com/help/vision/ug/train-a-cascade-object-detector.html
12. Docs, OpenCV. "Face Detection Using Haar Cascades." *OpenCV: Face Detection Using Haar Cascades*, 4 Aug. 2017, docs.opencv.org/3.3.0/d7/d8b/tutorialpyfacedetection.html.
13. Huang, J., Rathod, V., Sun, C., Zhu, M., Korattikara, A., Fathi, A., ... & Murphy, K. (2017). Speed/accuracy trade-offs for modern convolutional object detectors.

- In Proceedings of the IEEE conference on computer vision and pattern recognition* (pp. 7310-7311).
14. A. Heredia and G. Barros-Gavilanes, "Video processing inside embedded devices using SSD-Mobilenet to count mobility actors," 2019 IEEE Colombian Conference on Applications in Computational Intelligence (ColCACI), Barranquilla, Colombia, 2019, pp. 1-6, doi: 10.1109/ColCACI.2019.8781798.
 15. S. Kanimozhi, G. Gayathri and T. Mala, "Multiple Real-time object identification using Single shot Multi-Box detection," 2019 International Conference on Computational Intelligence in Data Science (ICCIDS), Chennai, India, 2019, pp. 1-5, doi: 10.1109/ICCIDS.2019.8862041.
 16. Y. -C. Chiu, C. -Y. Tsai, M. -D. Ruan, G. -Y. Shen and T. -T. Lee, "Mobilenet-SSDv2: An Improved Object Detection Model for Embedded Systems," 2020 International Conference on System Science and Engineering (ICSSE), Kagawa, Japan, 2020, pp. 1-5, doi: 10.1109/ICSSE50014.2020.9219319

PAPER PUBLICATION

ICT Academy is an initiative of the Government of India in collaboration with the state Governments and Industries. ICTACT is a not-for-profit society, the first of its kind pioneer venture under the Public-Private-Partnership (PPP) model that endeavors to train the higher education teachers and students thereby exercises on developing the next generation teachers and industry ready students.

ICTACT Journal on Image and Video Processing (IJIVP) is a peer – reviewed International Journal published quarterly. IJIVP welcomes Scientists, Researchers, Academicians and Engineers to submit their original research papers which is neither published nor currently under review by other journals or conferences. Papers should emphasize original results relating to both theoretical and application issues of Image and Video Processing. Review articles, focusing on multi disciplinary views, are also welcome.

ISSN Number (Print): 0976 9099, ISSN Number (Online): 0976-9102

The paper focuses on the method used for achieving object recognition through haar cascade algorithm integrated with single shot detector model using raspberry pi.