

**A SMART IOT INTERACTION SYSTEM WITH OBJECT
DETECTION AND GAZE ESTIMATION**

at

**Sathyabama Institute of Science and Technology
(DEEMED TO BE UNIVERSITY)**

Submitted in partial fulfillment of the requirements for the award of Bachelor
of Engineering Degree in Computer Science and Engineering

By

Sandeep Kumar(38110496,

VishalKumarGupta(3811064)



**DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING
SCHOOL OF COMPUTING SATHYABAMA INSTITUTE OF SCIENCE
AND TECHNOLOGY JEPPIAAR NAGAR, RAJIV GANDHI SALAI,
CHENNAI – 600119,**

APRIL 2021



SATHYABAMA

INSTITUTE OF SCIENCE AND TECHNOLOGY
(DEEMED TO BE UNIVERSITY)

Accredited with Grade “A” by NAAC

(Established under Section 3 of UGC Act, 1956)

JEPPIAAR NAGAR, RAJIV GANDHI SALAI, CHENNAI- 600119

www.sathyabamauniversity.ac.in



SCHOOL OF COMPUTING

BONAFIDE CERTIFICATE

This is to certify that this

Project Report is the

bonafide work Sandeep

Kumar(38110496)

who carried out the project entitled “A SMART IoT INTERACTION SYSTEM WITH OBJECT DETECTION AND GAZE ESTIMATION” under our supervision from November 2019 to April 2020.

Internal Guide

Dr. Albert Mayan, Ph.D.,

Head of the Department

Dr. S.VIGNESHWARI, M.E., Ph.D.,

Dr. L. LAKSHMANAN M.E., Ph.D.,

Submitted for Viva voce Examination held on

Internal Examiner

External Examiner

DECLARATION

**I Sandeep Kumar(38110496, hereby
declare ProjectTraining Report on "A
SMART IoT INTERACTION DEVICE
WITH OBJECT DETECTION**

AND GAZE ESTIMATION" done under the guidance of the faculty Dr. Albert mayan L M.E., Ph.D., at Sathyabama Institute of Science and Technology is submitted in partial fulfillment of the requirements for the award of Bachelor of Engineering degree in Computer Science and Engineering.

DATE:

PLACE: CHENNAI

SIGNATURE OF THE CANDIDATE

ACKNOWLEDGEMENT

I am pleased to acknowledge my sincere thanks to Board of Management of **SATHYABAMA** for their kind encouragement in doing this project and for completing it successfully. I am grateful to them.

I convey my thanks to **Dr. T. SASIKALA, M.E., Ph.D., Dean, School of Computing** and **Dr. S. VIGNESHWARI, M.E., Ph.D., Head of the Department, Department of Computer Science and Engineering** for providing me necessary support and details at the right time during the progressive reviews.

I would like to express my sincere and deep sense of gratitude to my Project Guide **Dr. ALBERT MAYAN M.E., Ph.D.**, for her valuable guidance, suggestions and constant encouragement paved way for the successful completion of my project work.

I wish to express my thanks to all Teaching and Non-teaching staff members of the Department of **Computer Science and Engineering** who were helpful in many ways For the Completion of the project.

ABSTRACT

(IoT) attempts to help people Internet-connected devices, applications, and services anytime and anywhere. However, how providing an efficient and intuitive method of interaction between people and IoT devices is still an open challenge. In this paper, we propose a novel interaction system called *Watch & Do*, where users can control an IoT device by gazing at it and doing simple gestures. The proposed system mainly consists of: 1) object detection module; 2) gaze estimation module; 3) hand gesture recognition module; and 4) IoT controller module. The target device is identified by various deep learning-based gaze estimation and object detection techniques. Afterwards, hand gesture recognition is applied to generate an IoT device control command which is transmitted to the IoT platform. The experimental results and case studies demonstrate the feasibility of the proposed system and imply the future research directions.

TABLE OF CONTENTS

CHAPTER No	TITLE	PAGE
	ABSTRACT	1
	LIST OF FIGURES	4
1	INTRODUCTION	5
	1.1 About the Project	5
2	SYSTEM ANALYSIS	6
	2.1 Existing system	6
	2.2 Proposed system	6
	2.3 Literature survey	7
3	REQUIREMENTS SPECIFICATION	9
	3.1 Introduction	9
	3.2 Hardware and Software specification	10
	3.3 Technologies Used	11
	3.3.1 Python	11

	3.3.3.1Introduction to Python	12
4	DESIGN AND IMPLEMENTATION	34
	4.1 Constraint in analysis	34
	4.2 Constraint in design	35
	5.1Architecture Diagram	36
6	SYSTEM DESIGN – DETAILED	44
	6.1 Steps of process	44
	6.2 Steps explanation	44
7	CODING AND TESTING	45
	7.1Coding	45
	7.2Coding standards	45
	7.3Test procedure	47
	7.4Test data and output	48
	SOURCE CODE	56
	SNAP SHOTS	60
	PAPER WORK	61
	REFERENCE	62

LIST OF FIGURES

FIGURE NO	FIGURE NAME	PAGE NO
	System Design	36
	Sequence Diagram	38
	Use Case Diagram	40
	Activity Diagram	42
	Collaboration Diagram	43

CHAPTER 1

INTRODUCTION

To detect target device is identified by various deep learning-based gaze estimation and object detection techniques and user can control the target IOT device by hand gesture.

The Internet of Things (IoT) attempts to help people access internet-connected devices, applications, and services anytime and anywhere. However, how providing an efficient and intuitive method of interaction between people and IoT devices is still an open challenge. In this work, we propose a novel interaction system called Watch & Do, where users can control an IoT device by gazing at it and doing simple gestures. The proposed system mainly consists of four categories: 1) object detection module, 2) gaze estimation module, 3) hand gesture recognition module and IoT controller module. The target device is identified by various deep learning-based gaze estimation and object detection techniques. Afterwards, hand gesture recognition is applied to generate an IoT device control command which is transmitted to the IoT platform. The experimental results and case studies demonstrate the feasibility of the proposed system and imply the future research directions.

CHAPTER 2

SYSTEM ANALYSIS

EXISTING SYSTEM

The target device is identified by various Algorithms are used for gaze estimation and object detection techniques. Afterwards, hand gesture recognition is applied to generate an IoT device control command which is transmitted to the IoT platform. These algorithms results are not accuracy. The experimental results and case studies demonstrate the feasibility of the proposed system and imply the future research directions.

PROPOSED SYSTEM

Proposed system works as follows. First, the Watch module records the opposite side of the user to detect and recognize the types of IoT devices installed in the room. Second, the Watch module detects the user's head region and then computes a fine-grained head pose information (i.e., pitch, yaw, and roll) to estimate the user's gaze position. With this information, the proposed system can identify the target device. Then, the Do module captures the user's hand gestures. A combination of hand gesture information and the type of selected IoT device is then translated into an IoT command and transmitted to IoT platforms. The target device is identified by deep learning Algorithms are used for gaze estimation and object detection techniques.

LITERATURE SURVEY

Charith perera [1] proposed content aware for computing for internet of things and Understanding sensor data is one of the main challenges that the IoT would face. This vision has been supported and heavily invested by governments, interest groups, companies, and research institutes. The IoT has gained significant attention over the last few years. With the advances in sensor hardware technology and cheap materials, sensors are expected to be attached to all the objects, Understanding sensor data is one of the main challenges that the IoT would face. This vision has been supported and heavily invested by governments, interest groups, companies, and research institutes. The IoT has gained significant attention over the last few years. With the advances in sensor hardware technology and cheap materials, sensors are expected to be attached to all the objects.

Luigi atzori [2] proposed a theory Each IoT element so as to make it possible to reach them from any other node of the network, looks more suitable for the traditional Internet paradigm. The IoT has the potential to add a new dimension to this process by enabling communications with and among smart objects, thus leading to the vision of „anytime, anywhere, any media, anything” communications. Technologies make the IoT concept feasible but do not fit well with the scalability and efficiency requirements. It is emphasized that „to the extent that everyday objects become information security risks, the IoT could distribute those risks far more widely than the Internet has to date”.

Chuyu Wang[3] proposed multi touch in air device free finger tracking and gesture recognition. Gesture recognition has gained considerable attention in emerging applications to provide a better user experience for human-computer interaction. Experiments validate that RF-finger can achieve as high as 88% and 92% accuracy for finger tracking and multi-touch gesture recognition. These solutions are either easily affected by the environmental noise or incapable of sensing fine-grained gestures at the finger level. We experimentally investigate the impact of tag array deployment on the signal quality.

Nataniel Ruiz Eunji Chong [4] proposed fine grain head pose We show that a multi-loss deep network can directly, accurately and robustly predict head rotation from image intensities. Landmark-to-pose methods are studied in this work to show their dependence on extraneous factors such as head model and landmark detection accuracy. Head pose recovery inherently is a two step process with numerous opportunities for error. Generic head models can introduce errors for any given participant, and the process of deforming the head model to adapt to each participant requires significant amounts of data and can be computationally expensive.

Gururaj Kulkarni [5] proposed enabling technologies protocols IoT has potential to broaden its horizon by enabling communication between smart objects. IoT will change everything drastically if implemented successfully, We presented some handsome application of IoT and its comfort in life. Some important issues that needed to be resolved have been discussed before wide acceptance of this technology There are various issues which need thorough research to improve the quality of life. An overview of IoT technologies, protocols and applications and related issues with a comparison of other papers.

Arkady Zaslavsky [6] proposed symatic driven configuration we have introduced the CASCoM approach that allows non-IT experts to configure IoT middleware efficiently and effectively Our model supports single-click configuration by eliminating sequences of manual activities needed to be carried out by users otherwise, the problem we addressed in this paper is comprehensively analyzed and presented with use-case scenarios in detail, Our objective is to automate and simplify the configuration of an IoT middleware and improve its usability so non-IT experts can use it efficiently and effectively

CHAPTER 3

REQUIREMENT SPECIFICATIONS

INTRODUCTION

THE RECENT developments in network infrastructure and smart devices have resulted in the rapid spread of the Internet of Things (IoT) applications and services. The IoT is defined as a network of Internet-connected things (e.g., computers, vehicles, and sensors) that exchange data and information among themselves with and other services [1]. The number of Internet-connected devices is now dramatically growing. According to a recent study on the prediction of IoT market share, the number of IoT devices will approach 100 billion and the total amount of data generated by the users and devices will reach 35 ZB by 2020 [2]. It is therefore expected that the success of IoT will allow the users and things to be connected anytime, anywhere using any path, network, or service [3]. These characteristics of the IoT ecosystem will improve the quality of services in various application domains such as health-care (e.g., remote patient monitoring and treatment), transportation (e.g., smart transportation systems), and home automation (e.g., smart appliances). A typical architecture of the IoT comprises a perception layer, a network layer, and an application layer [4]. Numerous studies have been conducted to address the challenging issues in each layer of the architecture. The perception layer first interacts with physical devices like RFID [5], sensors, and actuators and then connects the devices to the network of IoT. The network layer of the IoT is responsible for transmitting data between different things, applications, and services using heterogeneous networks and communication protocols. Finally, the application layer

exploits the data from the underlying layers to build and provide the required services. Interoperability of IoT devices, applications, services, and users is an important factor for the successful implementation of IoT. Therefore, the network layer and its corresponding technologies (e.g., Bluetooth [6], Wi-Fi [7], 6LoWPAN [8], ZigBee [9], Z-Wave [10], MQTT [11], and CoAP [12], etc.) have been usually considered the most important components since they are closely related to the connectivity, interoperability, and scalability of IoT architectures. Similarly, the IoT standards [13], [14], frameworks [15]-[17], and platforms in the application layer aim to maximize the interoperability by abstracting the layers of the IoT architecture and providing an efficient user interface (UI).

HARDWARE AND SOFTWARE SPECIFICATION

HARDWARE REQUIREMENTS

- Hard Disk : 500GB and Above
- RAM : 4GB and Above
- Processor : I3 and Above
- Webcam - 2
- Arduino UNO, Gesture Sensors, relay circuit board

SOFTWARE REQUIREMENTS

- ✓ Operating System : Windows (64 bit)
- ✓ Software : Python and Anaconda
- ✓ Tools : Jupyter Note Book and Spyder IDE

TECHNOLOGIES USED

- Python
- Region Based Convolution Neural Network & CNN

Introduction to Python

Python is a widely used general-purpose, high level programming language. It was initially designed by Guido van Rossum in 1991 and developed by Python Software Foundation. It was mainly developed for emphasis on code readability, and its syntax allows programmers to express concepts in fewer lines of code.

Python is a programming language that lets you work quickly and integrate systems more efficiently.

It is used for:

- web development (server-side),
- software development,
- mathematics,
- System scripting.

What can Python do?

- Python can be used on a server to create web applications.
- Python can be used alongside software to create workflows.
- Python can connect to database systems. It can also read and modify files.
- Python can be used to handle big data and perform complex mathematics.
- Python can be used for rapid prototyping, or for production-ready software development.

Why Python?

- Python works on different platforms (Windows, Mac, Linux, Raspberry Pi, etc).
- Python has a simple syntax similar to the English language.
- Python has syntax that allows developers to write programs with fewer lines than some other programming languages.
- Python runs on an interpreter system, meaning that code can be executed as soon as it is written. This means that prototyping can be very quick.
- Python can be treated in a procedural way, an object-orientated way or a functional way.

Good to know

- The most recent major version of Python is Python 3, which we shall be using in this tutorial. However, Python 2, although not being updated with anything other than security updates, is still quite popular.
- Python 2.0 was released in 2000, and the 2.x versions were the prevalent releases until December 2008. At that time, the development team made the decision to release version 3.0, which contained a few relatively small but significant changes that were not backward compatible with the 2.x versions. Python 2 and 3 are very similar, and some features of Python 3 have been backported to Python 2. But in general, they remain not quite compatible.
- Both Python 2 and 3 have continued to be maintained and developed, with periodic release updates for both. As of this writing, the most recent versions available are 2.7.15 and 3.6.5. However, an official End Of Life date of January 1, 2020 has been established for Python 2, after which time it will no longer be maintained.
- Python is still maintained by a core development team at the Institute, and Guido is still in charge, having been given the title of BDFL (Benevolent Dictator For Life) by the Python community. The name Python, by the way, derives not from the snake, but from the British comedy troupe Monty Python's Flying Circus, of which Guido was, and presumably still is, a fan. It is common to find references to Monty Python sketches and movies scattered throughout the Python documentation.

- It is possible to write Python in an Integrated Development Environment, such as Thonny, Pycharm, Netbeans or Eclipse which are particularly useful when managing larger collections of Python files.

Python Syntax compared to other programming languages

- Python was designed to for readability, and has some similarities to the English language with influence from mathematics.
- Python uses new lines to complete a command, as opposed to other programming languages which often use semicolons or parentheses.
- Python relies on indentation, using whitespace, to define scope; such as the scope of loops, functions and classes. Other programming languages often use curly-brackets for this purpose.

Python is Interpreted

- Many languages are compiled, meaning the source code you create needs to be translated into machine code, the language of your computer's processor, before it can be run. Programs written in an interpreted language are passed straight to an interpreter that runs them directly.
- This makes for a quicker development cycle because you just type in your code and run it, without the intermediate compilation step.
- One potential downside to interpreted languages is execution speed. Programs that are compiled into the native language of the computer processor tend to run more quickly than interpreted programs. For some applications that are particularly computationally intensive, like graphics processing or intense number crunching, this can be limiting.
- In practice, however, for most programs, the difference in execution speed is measured in milliseconds, or seconds at most, and not appreciably noticeable to a human user. The expediency of coding in an interpreted language is typically worth it for most applications.

- For all its syntactical simplicity, Python supports most constructs that would be expected in a very high-level language, including complex dynamic data types, structured and functional programming, and object-oriented programming.
- Additionally, a very extensive library of classes and functions is available that provides capability well beyond what is built into the language, such as database manipulation or GUI programming.
- Python accomplishes what many programming languages don't: the language itself is simply designed, but it is very versatile in terms of what you can accomplish with it.

Regional CNN

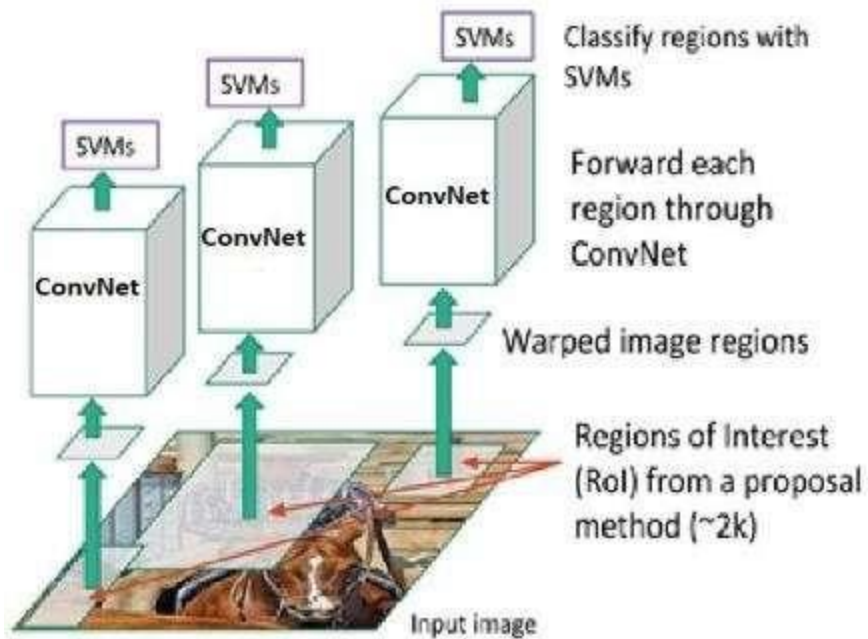
In this chapter we will talk about the Region based methods for object detection. In particular, R-CNN (Regional CNN), the original application of CNNs to this problem, along with its descendants Fast R-CNN, and Faster R-CNN. In classification, there's generally an image with a single object as the focus and the task is to say what that image is. But when we look at the world around us, we carry out far more complex tasks. We see complicated sights with multiple overlapping objects, and different backgrounds and we not only classify these different objects but also identify their boundaries, differences, and relations to one another. The goal of a Region based CNN is to take an image input to detect and localize the object in the image. All three above mentioned architectures were used for object detection and localization for a driving scenario to detect cars, pedestrians and traffic signs etc. and performed a time study to conclude which of the three architectures can be used as a detector for a real-world driving scenario for detection of objects present on road. We will also discuss their architecture and describe how they perform the task of object detection and localization given the input image.

R-CNN

R-CNN was first introduced by Ross Girshick, Jeff Donahue, Trevor Darrell and Jitendra Malik in 2014. Region based Convolutional Network (R-CNN) combines two ideas: (1) one can apply high-capacity Convolutional Networks (CNNs) to bottom-up region proposals in order to localize and segment objects and (2) When labelling data is scarce, supervised pre-training for an auxiliary task, followed by domain-specific-fine-tuning, boosts performance significantly. They combined region proposals with CNNs and called the resulting model an R-CNN or Region-based Convolutional Network.

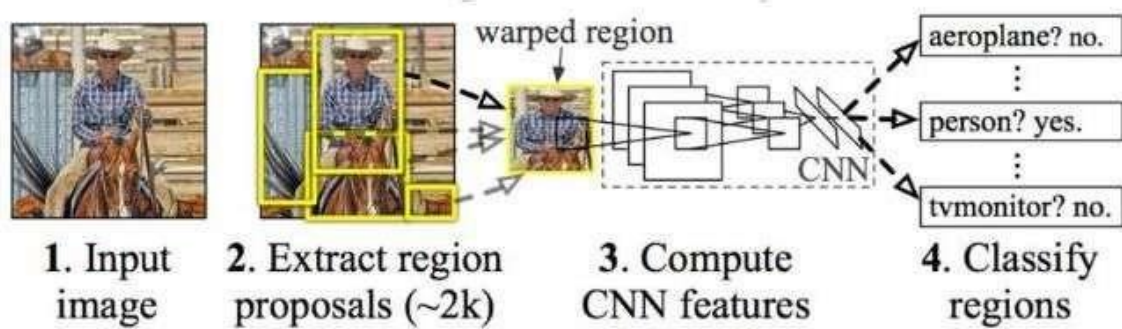
R-CNN Architecture

In R-CNN we input an image and then using the basic feature detection techniques such as edge detection etc. we get Region Proposals Also referred as Regions of interest. This process is also known as selective search.



R-CNN Architecture

In the next step after these region proposals are extracted these wrapped image regions go through a trained CNN in our implementation AlexNet and then on the final layer the classification is done Using a Support Vector Machine(SVM) classifier which classifies whether this is an object and if so what. Pictorial representation of above process is shown in Figure.



Procedure for Object Detection and Localisation in R-CNN

R-CNN works really well but is really quite slow for implementation in real time object detection. The reasons being

1. It requires a forward pass of the CNN (AlexNet) for every single region proposal for every single image that's around 2000 forward passes per image.
2. It must train three different models separately -the CNN to generate image features, the classifier that predicts the class and the regression model to tighten the bounding boxes. This makes the pipeline extremely hard to train.

Convolutional Neural Networks (CNN)

Introduction

Convolutional neural networks (CNN) sounds like a weird combination of biology and math with a little CS sprinkled in, but these networks have been some of the most

influential innovations in the field of computer vision. 2012 was the first year that neural nets grew to prominence as Alex Krizhevsky used them to win that year's ImageNet competition (basically, the annual Olympics of computer vision), dropping the classification error record from 26% to 15%, an astounding improvement at the time. Ever since then, a host of companies have been using deep learning at the core of their services. Facebook uses neural nets for their automatic tagging algorithms, Google for their photo search, Amazon for their product recommendations, Pinterest for their home feed personalization, and Instagram for their search infrastructure.

However, the classic, and arguably most popular, use case of these networks is for image processing. Within image processing, let's take a look at how to use these CNNs for image classification.

The Problem Space

Image classification is the task of taking an input image and outputting a class (a cat, dog, etc) or a probability of classes that best describes the image. For humans, this task of recognition is one of the first skills we learn from the moment we are born and is one that comes naturally and effortlessly as adults. Without even thinking twice, we're able to quickly and seamlessly identify the environment we are in as well as the objects that surround us. When we see an image or just when we look at the world around us, most of the time we are able to immediately characterize the scene and give each object a label, all without even consciously noticing. These skills of being able to quickly

recognize patterns, generalize from prior knowledge, and adapt to different image environments are ones that we do not share with our fellow machines.



What We See

```
00 02 22 97 38 15 00 40 00 75 04 05 07 78 52 12 50 77 91 08
49 49 59 40 17 81 18 57 60 87 17 40 98 43 69 48 04 56 62 00
81 48 31 75 55 79 14 29 93 71 40 67 53 88 30 09 49 13 34 65
52 70 95 23 04 60 11 42 69 24 68 56 01 32 56 71 37 02 36 91
22 31 16 71 51 67 63 89 41 92 36 54 22 40 40 28 66 33 13 80
24 47 32 60 99 03 45 02 44 75 33 53 78 36 84 20 38 17 12 50
32 98 81 28 64 23 67 10 26 39 40 67 59 34 70 66 18 38 64 70
67 26 20 69 02 62 12 20 95 63 94 39 63 08 40 91 66 49 94 21
24 55 58 05 86 73 99 24 97 17 78 78 96 83 14 88 34 89 63 72
21 36 29 09 75 00 74 44 20 45 35 14 00 61 33 97 34 31 33 96
78 17 53 28 22 75 31 67 15 94 03 80 04 62 16 14 09 53 56 92
16 35 05 42 96 35 31 47 55 88 08 24 00 17 54 24 36 29 28 37
86 56 00 48 35 71 89 07 05 44 94 37 44 60 21 58 81 54 17 58
19 80 81 68 05 94 47 69 28 73 92 13 88 52 17 77 04 89 55 40
04 52 08 23 97 35 99 14 07 97 57 32 16 26 26 79 33 27 82 66
88 36 68 87 57 62 20 72 03 66 33 67 46 55 12 32 63 93 53 69
04 42 16 73 38 25 39 11 24 94 72 18 08 46 29 32 40 62 74 36
20 69 36 41 72 30 23 88 34 62 99 69 82 67 59 85 74 04 36 16
20 73 35 29 78 31 90 01 74 31 49 71 48 86 81 16 23 07 05 34
01 70 84 71 83 51 94 69 16 92 38 48 61 43 82 01 89 19 67 48
```

What Computers See

Inputs and Outputs

When a computer sees an image (takes an image as input), it will see an array of pixel values. Depending on the resolution and size of the image, it will see a 32 x 32 x 3 array of numbers (The 3 refers to RGB values). Just to drive home the point, let's say we have a color image in JPG form and its size is 480 x 480. The representative array will be 480 x 480 x 3. Each of these numbers is given a value from 0 to 255 which describe the pixel intensity at that point. These numbers, while meaningless to us when we perform image classification, are the only inputs available to the computer. The idea is that you give the computer this array of numbers and it will output numbers that describe the probability of the image being a certain class (.80 for cat, .15 for dog, .05 for bird, etc).

Now that we know the problem as well as the inputs and outputs, let's think about how to approach this. What we want the computer to do is to be able to differentiate between all the images it's given and figure out the unique features that make a dog a dog or that make a cat a cat. This is the process that goes on in our minds subconsciously as well. When we look at a picture of a dog, we can classify it as such if the picture has identifiable features such as paws or 4 legs. In a similar way, the computer is able to perform image classification by looking for low level features such as edges and curves, and then building up to more abstract concepts through a series of convolutional layers. This is a general overview of what a CNN does. Let's get into the specifics.

Biological Connection

But first, a little background. When you first heard of the term convolutional neural networks, you may have thought of something related to neuroscience or biology, and you would be right. Sort of. CNNs do take a biological inspiration from the visual cortex. The visual cortex has small regions of cells that are sensitive to specific regions of the visual field. This idea was expanded upon by a fascinating experiment by Hubel and Wiesel in 1962 ([Video](#)) where they showed that some individual neuronal cells in the brain responded (or fired) only in the presence of edges of a certain orientation. For example, some neurons fired when exposed to vertical edges and some when shown horizontal or diagonal edges. Hubel and Wiesel found out that all of these neurons were organized in a columnar architecture and that together, they were able to produce visual perception. This idea of specialized components inside of a system having specific tasks

(the neuronal cells in the visual cortex looking for specific characteristics) is one that machines use as well, and is the basis behind CNNs.

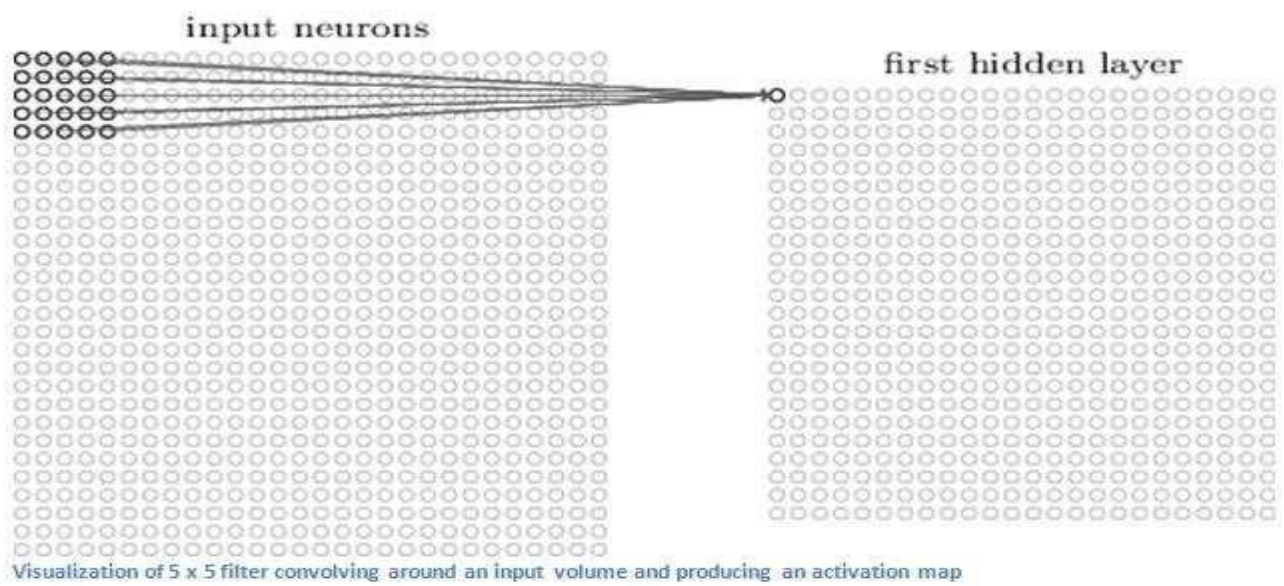
Structure

Back to the specifics. A more detailed overview of what CNNs do would be that you take the image, pass it through a series of convolutional, nonlinear, pooling (down sampling), and fully connected layers, and get an output. As we said earlier, the output can be a single class or a probability of classes that best describes the image. Now, the hard part understands what each of these layers do. So let's get into the most important one.

First Layer – Math Part

The first layer in a CNN is always a Convolutional Layer. First thing to make sure you remember is what the input to this conv (I'll be using that abbreviation a lot) layer is. Like we mentioned before, the input is a $32 \times 32 \times 3$ array of pixel values. Now, the best way to explain a conv layer is to imagine a flashlight that is shining over the top left of the image. Let's say that the light this flashlight shines covers a 5×5 area. And now, let's imagine this flashlight sliding across all the areas of the input image. In machine learning terms, this flashlight is called a filter (or sometimes referred to as a neuron or a kernel) and the region that it is shining over is called the receptive field. Now this filter is also an array of numbers (the numbers are called weights or parameters). A very important note is that the depth of this filter has to be the same as the depth of the input

(this makes sure that the math works out), so the dimension of this filter is $5 \times 5 \times 3$. Now, let's take the first position the filter is in for example. It would be the top left corner. As the filter is sliding, or convolving, around the input image, it is multiplying the values in the filter with the original pixel values of the image (aka computing element wise multiplications). These multiplications are all summed up (mathematically speaking, this would be 75 multiplications in total). So now you have a single number. Remember, this number is just representative of when the filter is at the top left of the image. Now, we repeat this process for every location on the input volume. (Next step would be moving the filter to the right by 1 unit, then right again by 1, and so on). Every unique location on the input volume produces a number. After sliding the filter over all the locations, you will find out that what you're left with is a $28 \times 28 \times 1$ array of numbers, which we call an activation map or feature map. The reason you get a 28×28 array is that there are 784 different locations that a 5×5 filter can fit on a 32×32 input image. These 784 numbers are mapped to a 28×28 array.



Let's say now we use two $5 \times 5 \times 3$ filters instead of one. Then our output volume would be $28 \times 28 \times 2$. By using more filters, we are able to preserve the spatial dimensions better. Mathematically, this is what's going on in a convolutional layer.

First Layer – High Level Perspective

However, let's talk about what this convolution is actually doing from a high level. Each of these filters can be thought of as **feature identifiers**. When I say features, I'm talking about things like straight edges, simple colors, and curves. Think about the simplest characteristics that all images have in common with each other. Let's say our first filter is $7 \times 7 \times 3$ and is going to be a curve detector. (In this section, let's ignore the fact that the filter is 3 units deep and only consider the top depth slice of the filter and the image, for simplicity.) As a curve detector, the filter will have a pixel structure in which there will be higher numerical values along the area that is a shape of a curve (Remember, these filters that we're talking about as just numbers!).

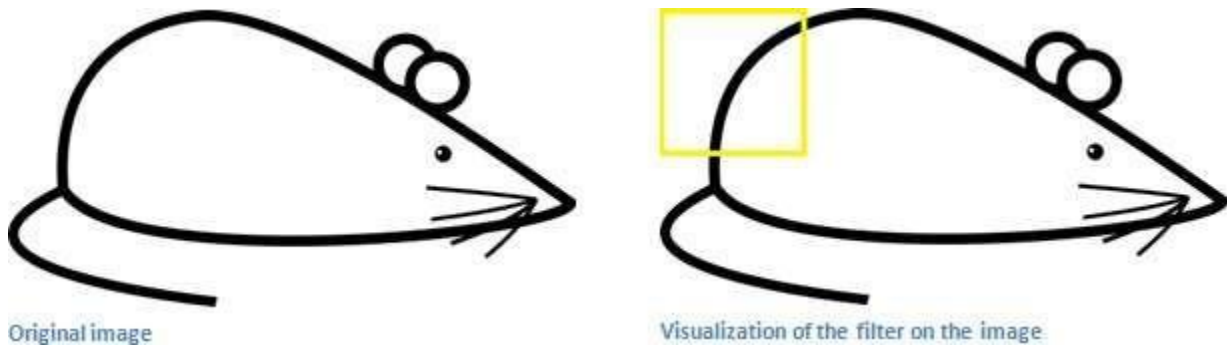
0	0	0	0	0	30	0
0	0	0	0	30	0	0
0	0	0	30	0	0	0
0	0	0	30	0	0	0
0	0	0	30	0	0	0
0	0	0	30	0	0	0
0	0	0	0	0	0	0

Pixel representation of filter

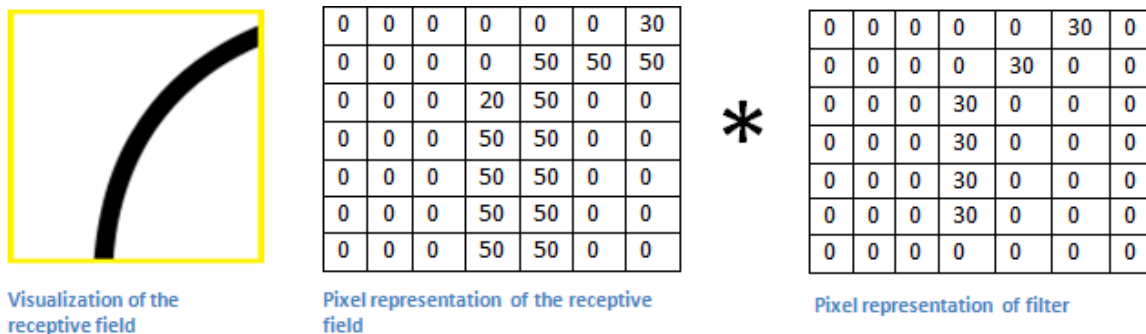


Visualization of a curve detector filter

at that region. Now let's take an example of an image that we want to classify, and let's put our filter at the top left corner. Now, let's go back to visualizing this mathematically. When we have this filter at the top left corner of the input volume, it is computing multiplications between the filter and pixel values

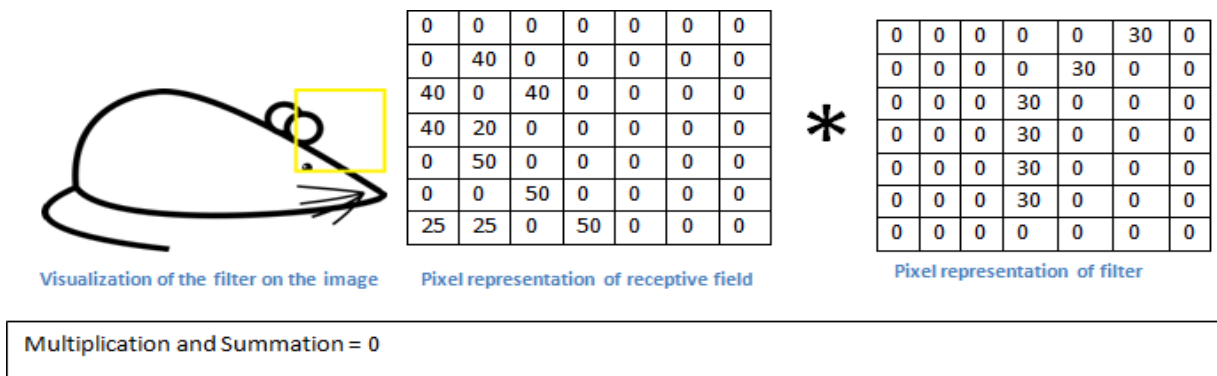


Remember, what we have to do is multiply the values in the filter with the original pixel values of the image.



Multiplication and Summation = $(50 \times 30) + (50 \times 30) + (50 \times 30) + (20 \times 30) + (50 \times 30) = 6600$ (A large number!)

Basically, in the input image, if there is a shape that generally resembles the curve that this filter is representing, then all of the multiplications summed together will result in a large value! Now let's see what happens when we move our filter.

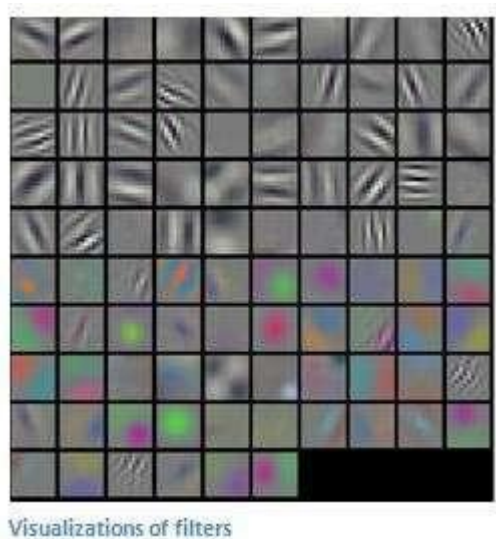


The value is much lower! This is because there wasn't anything in the image section that responded to the curve detector filter. Remember, the output of this conv layer is an activation map. So, in the simple case of a one filter convolution (and if that filter is a curve detector), the activation map will show the areas in which there are mostly likely to be curves in the picture. In this example, the top left value of our 26 x 26 x 1 activation map (26 because of the 7x7 filter instead of 5x5) will be 6600. This high value means that it is likely that there is some sort of curve in the input volume that caused the filter to activate. The top right value in our activation map will be 0 because there wasn't anything in the input volume that caused the filter to activate (or more simply said, there wasn't a curve in that region of the original image). Remember, this is just for one filter. This is just a filter that is going to detect lines that curve outward and to the right. We can have other filters for lines that curve to the left or for straight edges. The more filters, the

greater the depth of the activation map, and the more information we have about the input volume.

Disclaimer:

The filter I described in this section was simplistic for the main purpose of describing the math that goes on during a convolution. In the picture below, you'll see some examples of actual visualizations of the filters of the first conv layer of a trained network. Nonetheless, the main argument remains the same. The filters on the first layer convolve around the input image and "activate" (or compute high values) when the specific feature it is looking for is in the input volume.



Going Deeper Through the Network

Now in traditional convolutional neural network architecture, there are other layers that are interspersed between these conv layers. I'd strongly encourage those

interested to read up on them and understand their function and effects, but in a general sense, they provide nonlinearities and preservation of dimension that help to improve the robustness of the network and control over fitting. A classic CNN architecture would look like this.

Input -> Conv -> ReLU -> Conv -> ReLU -> Pool -> ReLU -> Conv -> ReLU -> Pool -> Fully Connected

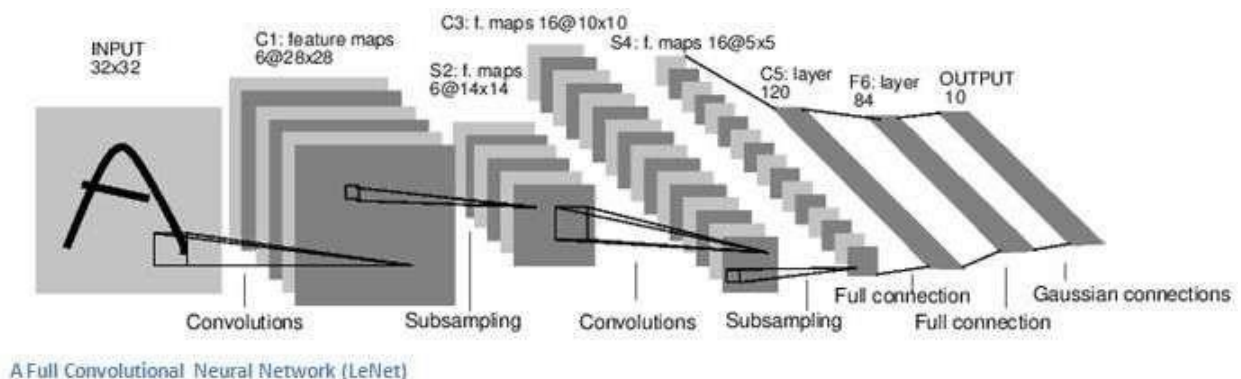
curve and straight edge) or squares (combination of several straight edges). As you go through the network and go through more conv layers, you get activation maps that represent more and more complex features. By the end of the network, you may have some filters that activate when there is handwriting in the image, filters that activate when they see pink objects, etc. If you want more information about visualizing filters in ConvNets, Matt Zeiler and Rob Fergus had an excellent research paper discussing the topic. The last layer, however, is an important one and one that we will go into later on. Let's just take a step back and review what we've learned so far. We talked about what the filters in the first conv layer are designed to detect. They detect low level features such as edges and curves. As one would imagine, in order predicting whether an image is a type of object, we need the network to be able to recognize higher level features such as hands or paws or ears. So let's think about what the output of the network is after the first conv layer. It would be a $28 \times 28 \times 3$ volume (assuming we use three $5 \times 5 \times 3$ filters). When we go through another conv layer, the output of the first conv layer becomes the input of the 2nd conv layer. Now, this is a little bit harder to visualize. When we were talking about the first layer, the input was just the original image. However, when we're talking about the 2nd conv layer, the input is the

activation map(s) that result from the first layer. So each layer of the input is basically describing the locations in the original image for where certain low level features appear. Now when you apply a set of filters on top of that (pass it through the 2nd conv layer), the output will be activations that represent higher level features. Types of these features could be semicircles (combination of a. Jason Yosinski also has a [video](#) on YouTube that provides a great visual representation. Another interesting thing to note is that as you go deeper into the network, the filters begin to have a larger and larger receptive field, which means that they are able to consider information from a larger area of the original input volume (another way of putting it is that they are more responsive to a larger region of pixel space).

Fully Connected Layer

Now that we can detect these high level features, the icing on the cake is attaching a **fully connected layer** to the end of the network. This layer basically takes an input volume (whatever the output is of the conv or ReLU or pool layer preceding it) and outputs an N dimensional vector where N is the number of classes that the program has to choose from. For example, if you wanted a digit classification program, N would be 10 since there are 10 digits. Each number in this N dimensional vector represents the probability of a certain class. For example, if the resulting vector for a digit classification program is [0 .1 .1 .75 0 0 0 0 0 .05], then this represents a 10% probability that the image is a 1, a 10% probability that the image is a 2, a 75% probability that the image is a 3, and a 5% probability that the image is a 9 (Side note: There are other ways that you can represent the output, but I am just showing the softmax approach). The way this fully

connected layer works is that it looks at the output of the previous layer (which as we remember should represent the activation maps of high level features) and determines which features most correlate to a particular class. For example, if the program is predicting that some image is a dog, it will have high values in the activation maps that represent high level features like a paw or 4 legs, etc. Similarly, if the program is predicting that some image is a bird, it will have high values in the activation maps that represent high level features like wings or a beak, etc. Basically, a FC layer looks at what high level features most strongly correlate to a particular class and has particular weights so that when you compute the products between the weights and the previous layer, you get the correct probabilities for the different classes.



Training (AKA: What Makes this Stuff Work)

Now, this is the one aspect of neural networks that I purposely haven't mentioned yet and it is probably the most important part. There may be a lot of questions you had while reading. How do the filters in the first conv layer know to look for edges and curves? How does the fully connected layer know what activation maps to look at?

How do the filters in each layer know what values to have? The way the computer is able to adjust its filter values (or weights) is through a training process called **backpropagation**.

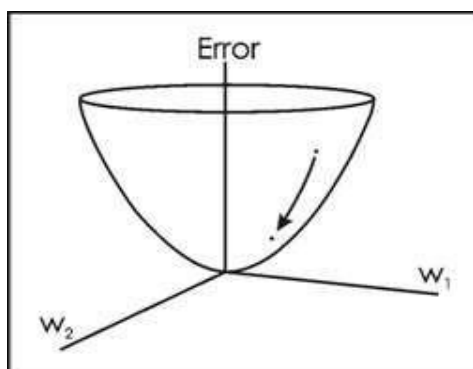
Before we get into backpropagation, we must first take a step back and talk about what a neural network needs in order to work. At the moment we all were born, our minds were fresh. We didn't know what a cat or dog or bird was. In a similar sort of way, before the CNN starts, the weights or filter values are randomized. The filters don't know to look for edges and curves. The filters in the higher layers don't know to look for paws and beaks. As we grew older however, our parents and teachers showed us different pictures and images and gave us a corresponding label. This idea of being given an image and a label is the training process that CNNs go through. Before getting too into it, let's just say that we have a training set that has thousands of images of dogs, cats, and birds and each of the images has a label of what animal that picture is. Back to backprop.

So backpropagation can be separated into 4 distinct sections, the forward pass, the loss function, the backward pass, and the weight update. During the **forward pass**, you take a training image which as we remember is a $32 \times 32 \times 3$ array of numbers and pass it through the whole network. On our first training example, since all of the weights or filter values were randomly initialized, the output will probably be something like $[.1 \ .1 \ .1 \ .1 \ .1 \ .1 \ .1 \ .1 \ .1]$, basically an output that doesn't give preference to any number in particular. The network, with its current weights, isn't able to look for those low level features or thus isn't able to make any reasonable conclusion about what the classification might be. This goes to the **loss function** part of backpropagation.

Remember that what we are using right now is training data. This data has both an image and a label. Let's say for example that the first training image inputted was a 3. The label for the image would be [0 0 0 1 0 0 0 0 0]. A loss function can be defined in many different ways but a common one is MSE (mean squared error), which is $\frac{1}{2}$ times (actual - predicted) squared.

$$E_{total} = \sum \frac{1}{2} (target - output)^2$$

Let's say the variable L is equal to that value. As you can imagine, the loss will be extremely high for the first couple of training images. Now, let's just think about this intuitively. We want to get to a point where the predicted label (output of the ConvNet) is the same as the training label (This means that our network got its prediction right). In order to get there, we want to minimize the amount of loss we have. Visualizing this as just an optimization problem in calculus, we want to find out which inputs (weights in our case) most directly contributed to the loss (or error) of the network.



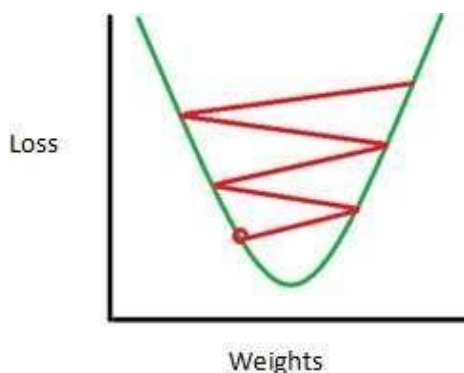
One way of visualizing this idea of minimizing the loss is to consider a 3-D graph where the weights of the neural net (there are obviously more than 2 weights, but let's go for simplicity) are the independent variables and the dependent variable is the loss. The task of minimizing the loss involves trying to adjust the weights so that the loss decreases. In visual terms, we want to get to the lowest point in our bowl shaped object. To do this, we have to take a derivative of the loss (visual terms: calculate the slope in every direction) with respect to the weights.

This is the mathematical equivalent of a dL/dW where W are the weights at a particular layer. Now, what we want to do is perform a **backward pass** through the network, which is determining which weights contributed most to the loss and finding ways to adjust them so that the loss decreases. Once we compute this derivative, we then go to the last step which is the **weight update**. This is where we take all the weights of the filters and update them so that they change in the opposite direction of the gradient.

$$w = w_i - \eta \frac{dL}{dW}$$

w = Weight
w_i = Initial Weight
η = Learning Rate

The **learning rate** is a parameter that is chosen by the programmer. A high learning rate means that bigger steps are taken in the weight updates and thus, it may take less time for the model to converge on an optimal set of weights. However, a learning rate that is too high could result in jumps that are too large and not precise enough to reach the optimal point.



Consequence of a high learning rate where the jumps are too large and we are not able to minimize the loss.

The process of forward pass, loss function, backward pass, and parameter update is one training iteration. The program will repeat this process for a fixed number of iterations for each set of training images (commonly called a batch). Once you finish the parameter update on the last training example, hopefully the network should be trained well enough so that the weights of the layers are tuned correctly.

Testing

Finally, to see whether or not our CNN works, we have a different set of images and labels (can't double dip between training and test!) and pass the images through the CNN. We compare the outputs to the ground truth and see if our network works!

How Companies Use CNNs

Data, data, data. The companies that have lots of this magic 4 letter word are the ones that have an inherent advantage over the rest of the competition. The more training data that you can give to a network, the more training iterations you can make, the more weight updates you can make, and the better tuned to the network is when it goes to production. Facebook (and Instagram) can use all the photos of the billion users it currently has, Pinterest can use information of the 50 billion pins that are on its site, Google can use search data, and Amazon can use data from the millions of products that are bought every day. And now you know the magic behind how they use it.

CHAPTER 4

Design and Implementation Constraints

1 Constraints in Analysis

- ◆ Constraints as Informal Text
- ◆ Constraints as Operational Restrictions
- ◆ Constraints Integrated in Existing Model Concepts
- ◆ Constraints as a Separate Concept
- ◆ Constraints Implied by the Model Structure

Constraints in Design

- ◆ Determination of the Involved Classes
- ◆ Determination of the Involved Objects
- ◆ Determination of the Involved Actions
- ◆ Determination of the Require Clauses
- ◆ Global actions and Constraint Realization

Constraints in Implementation

A hierarchical structuring of relations may result in more classes and a more complicated structure to implement. Therefore it is advisable to transform the hierarchical relation structure to a simpler structure such as a classical flat one. It is rather straightforward to transform the developed hierarchical model into a bipartite, flat model, consisting of classes on the one hand and flat relations on the other. Flat relations are preferred at the design level for reasons of simplicity and implementation ease. There is no identity or functionality associated with a flat relation. A flat relation corresponds with the relation concept of entity-relationship modeling and

Other Nonfunctional Requirements

4.2.1 Performance Requirements

The application at this side controls and communicates with the following three main general components.

- embedded browser in charge of the navigation and accessing to the web service;
- Server Tier: The server side contains the main parts of the functionality of the proposed architecture. The components at this tier are the following.

Web Server, Security Module, Server-Side Capturing Engine, Preprocessing Engine, Database System, Verification Engine, Output Module.

Safety Requirements

1. The software may be safety-critical. If so, there are issues associated with its integrity level
2. The software may not be safety-critical although it forms part of a safety-critical system. For example, software may simply log transactions.
3. If a system must be of a high integrity level and if the software is shown to be of that integrity level, then the hardware must be at least of the same integrity level.
4. There is little point in producing 'perfect' code in some language if hardware and system software (in widest sense) are not reliable.
5. If a computer system is to run software of a high integrity level then that system should not at the same time accommodate software of a lower integrity level.
6. Systems with different requirements for safety levels must be separated.

CHAPTER 5

Architecture Diagram:

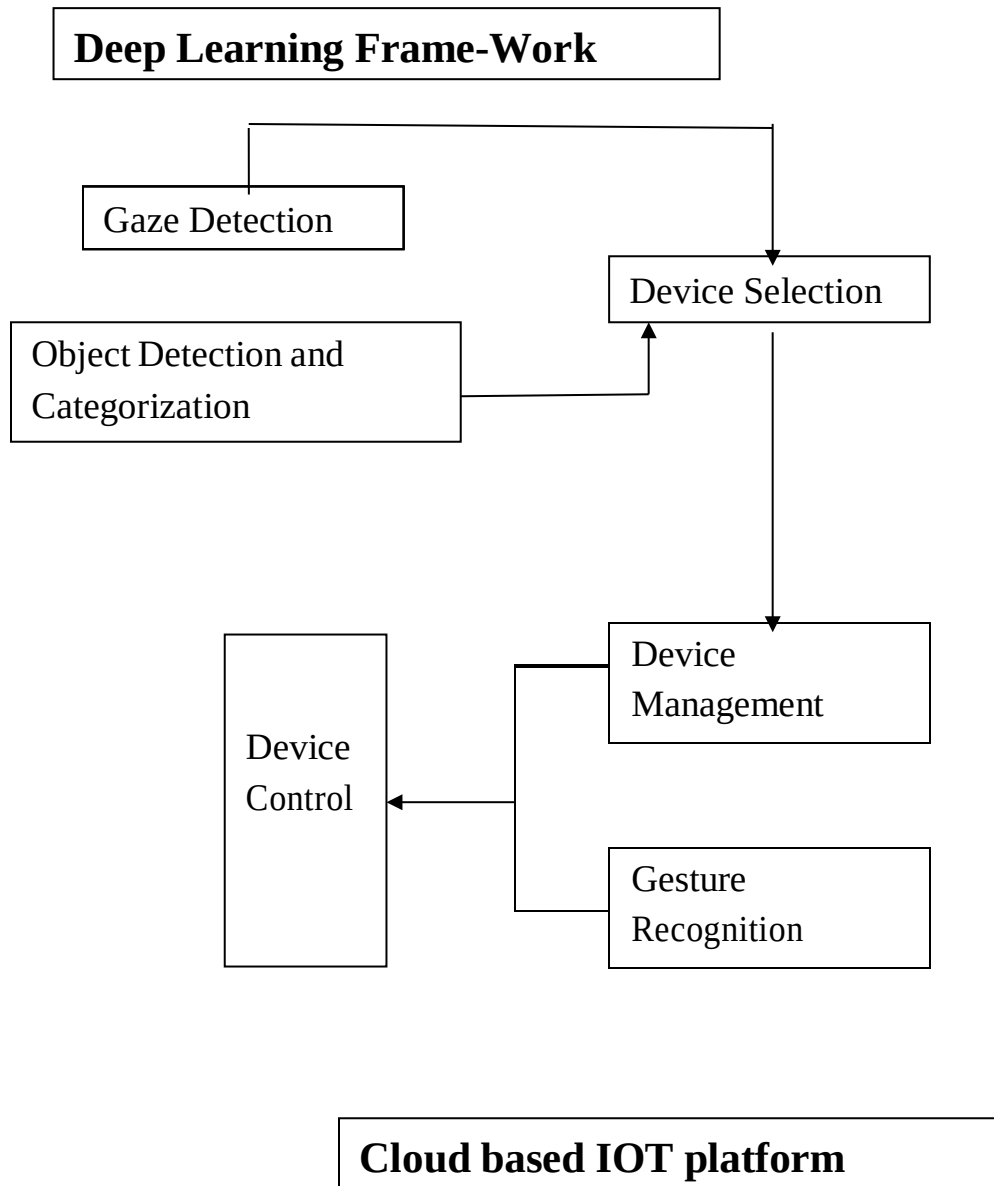
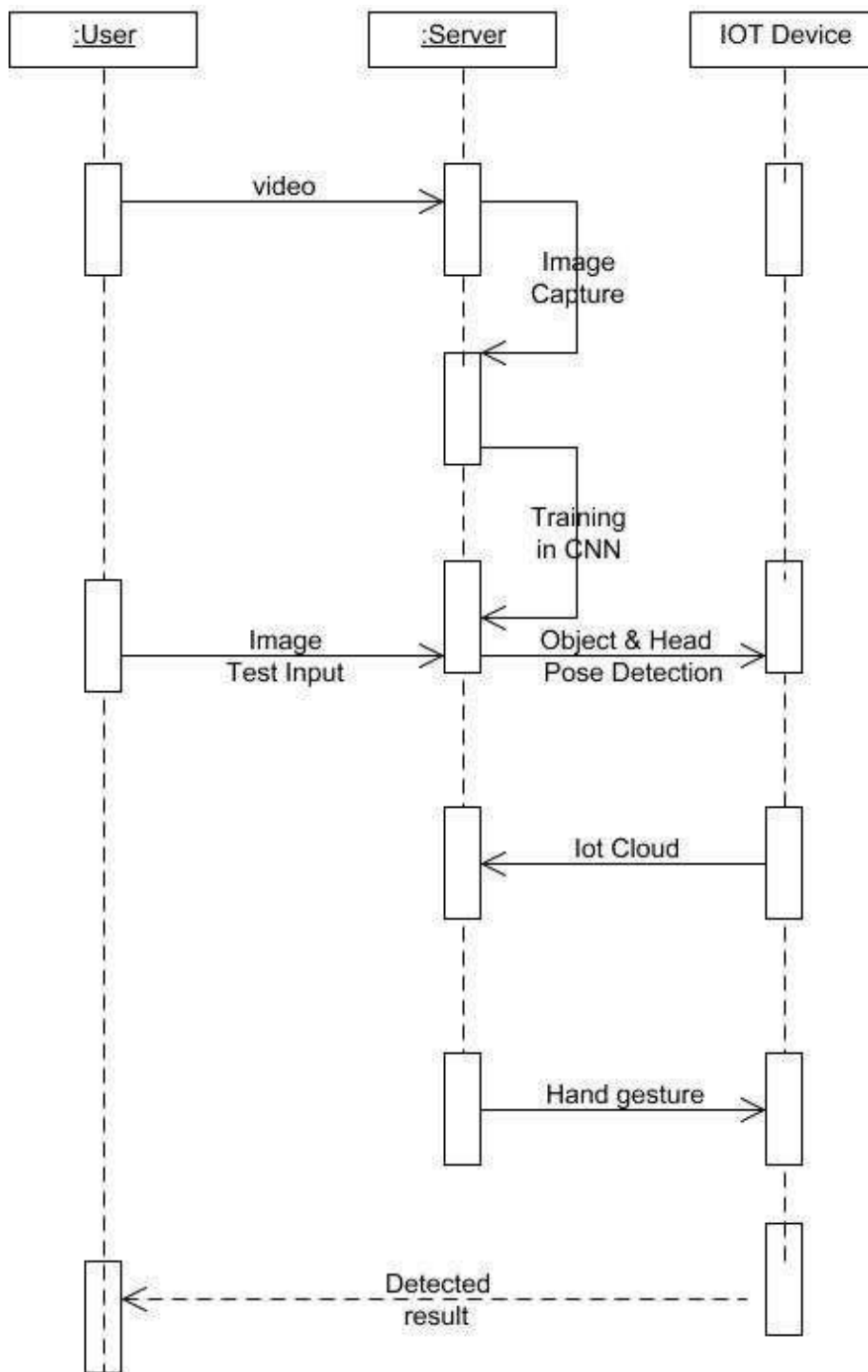


Fig: 5.1

Sequence Diagram:

A Sequence diagram is a kind of interaction diagram that shows how processes operate with one another and in what order. It is a construct of Message Sequence diagrams are sometimes called event diagrams, event sceneries and timing diagram.



Use Case Diagram:

Unified Modeling Language (UML) is a standardized general-purpose modeling language in the field of software engineering. The standard is managed and was created by the Object Management Group. UML includes a set of graphic notation techniques to create visual models of software intensive systems. This language is used to specify, visualize, modify, construct and document the artifacts of an object oriented software intensive system under development.

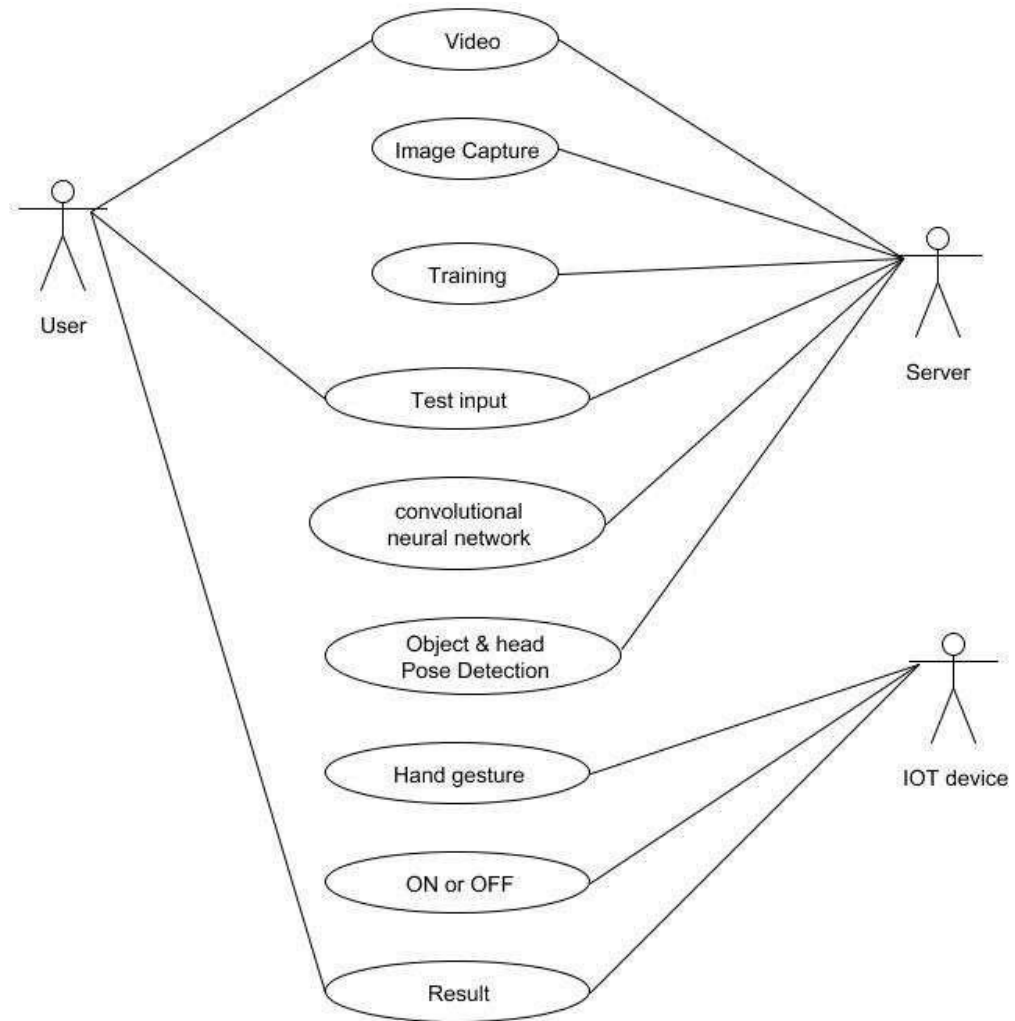
USECASE DIAGRAM

A Use case Diagram is used to present a graphical overview of the functionality provided by a system in terms of actors, their goals and any dependencies between those use cases.

Use case diagram consists of two parts:

Use case: A use case describes a sequence of actions that provided something of measurable value to an actor and is drawn as a horizontal ellipse.

Actor: An actor is a person, organization or external system that plays a role in one or more interaction with the system.



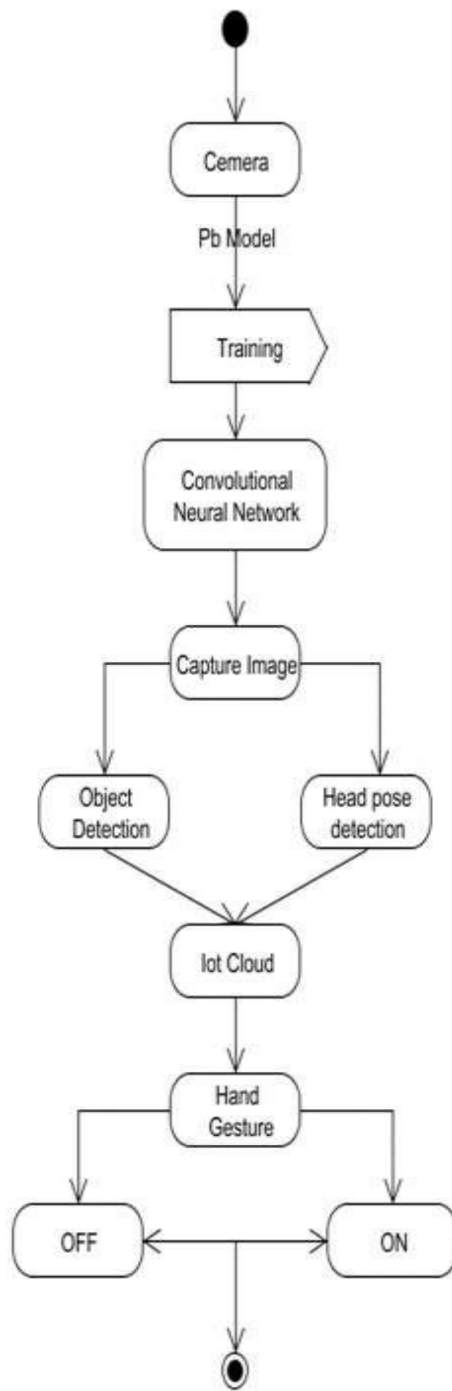
Activity Diagram:

Activity diagram is a graphical representation of workflows of stepwise activities and actions with support for choice, iteration and concurrency. An activity diagram shows the overall flow of control.

The most important shape types:

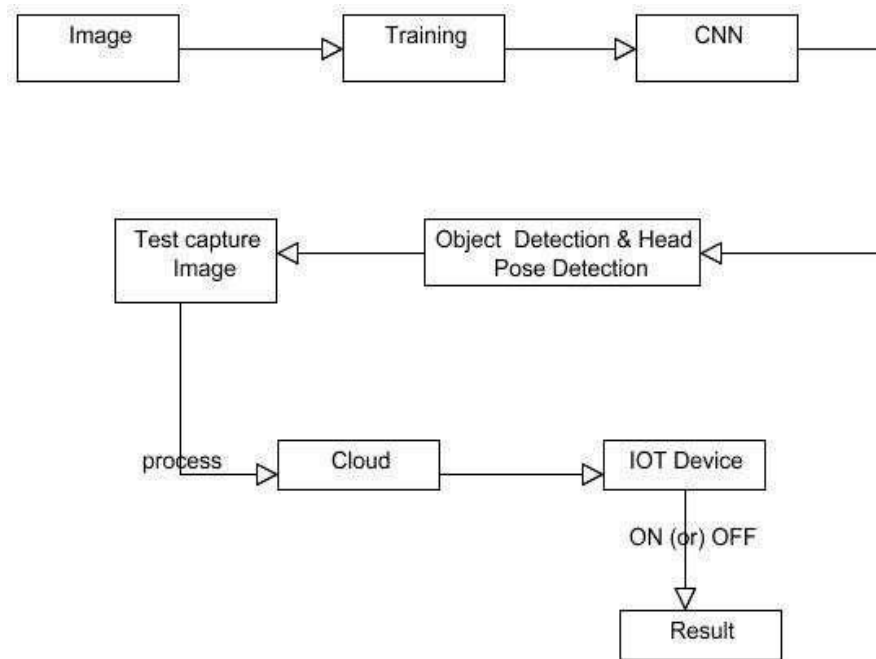
- Rounded rectangles represent activities.

- Diamonds represent decisions.
- Bars represent the start or end of concurrent activities.
- A black circle represents the start of the workflow.
- An encircled circle represents the end of the workflow.



Collaboration Diagram:

UML Collaboration Diagrams illustrate the relationship and interaction between software objects. They require use cases, system operation contracts and domain model to already exist. The collaboration diagram illustrates messages being sent between classes and objects.



CHAPTER 6

Steps of process

. Object detection

.Gaze estimation

.Image recognition

Steps explanation

Object Detection:

The Watch module also records the opposite side of the user to detect the IoT devices installed in the room and recognizes their types. For object detection and recognition, recent deep learning approaches have shown excellent performance in terms of accuracy. The Deep Learning framework divides each input image into $S \times S$ grids and each grid predicts N bounding boxes and their confidence scores which indicate whether the bounding box contains an object or not.

Gaze Estimation:

We use the orientation of the face (i.e., pitch, yaw, and roll) and the distance between the Watch module and the user as head pose information for training a gaze estimator. A deep learning-based approach called Deep Gaze was employed to compute the value of face orientation.

The DeepGaze uses a convolutional neural network (CNN) to output the values of face orientation from the given face image.

Hand Gesture Recognition:

Gesture recognition is performed by a commercial gesture sensor module. This sensor supports the following nine hand gestures: “Up”, “Down”, “Left” .

CHAPTER 7

CODING AND TESTING

CODING

Once the design aspect of the system is finalized the system enters into the coding and testing phase. The coding phase brings the actual system into action by converting the design of the system into the code in a given programming language. Therefore, a good coding style has to be taken whenever changes are required it easily screws into the system.

CODING STANDARDS

Coding standards are guidelines to programming that focus on the physical structure and appearance of the program. They make the code easier to read, understand and maintain. This phase of the system actually implements the blueprint developed during the design phase. The coding specification should be in such a way that any programmer must be able to understand the code and can bring about changes whenever felt necessary. Some of the standards needed to achieve the above-mentioned objectives are as follows:

Program should be simple, clear and easy to understand.

Naming conventions

Value conventions

Script and comment procedure

Message box format

Exception and error handling

NAMING CONVENTIONS

Naming conventions of classes, data member, member functions, procedures etc., should be **self-descriptive**. One should even get the meaning and scope of the variable by its name. The conventions are adopted for **easy understanding** of the intended message by the user. So it is customary to follow the conventions. These conventions are as follows:

Class names

Class names are problem domain equivalence and begin with capital letter and have mixed cases.

Member Function and Data Member name

Member function and data member name begins with a lowercase letter with each subsequent letters of the new words in uppercase and the rest of letters in lowercase.

VALUE CONVENTIONS

Value conventions ensure values for variable at any point of time. This involves the following:

- Proper default values for the variables.
- Proper validation of values in the field.
- Proper documentation of flag values.

SCRIPT WRITING AND COMMENTING STANDARD

Script writing is an art in which indentation is utmost important. Conditional and looping statements are to be properly aligned to facilitate easy understanding. Comments are included to minimize the number of surprises that could occur when going through the code.

MESSAGE BOX FORMAT

When something has to be prompted to the user, he must be able to understand it properly. To achieve this, a specific format has been adopted in displaying messages to the user. They are as follows:

- X - User has performed illegal operation.
- ! - Information to the user.

TEST PROCEDURE

SYSTEM TESTING

Testing is performed to identify errors. It is used for quality assurance. Testing is an integral part of the entire development and maintenance process. The goal of the testing during phase is to verify that the specification has been accurately and completely incorporated into the design, as well as to ensure the correctness of the design itself. For example the design must not have any logic faults in the design is detected before coding commences, otherwise the cost of fixing the faults will be considerably higher as reflected. Detection of design faults can be achieved by means of inspection as well as walkthrough.

Testing is one of the important steps in the software development phase. Testing checks for the errors, as a whole of the project testing involves the following test cases:

- Static analysis is used to investigate the structural properties of the Source code.
- Dynamic testing is used to investigate the behavior of the source code by executing the program on the test data.

TEST DATA AND OUTPUT

UNIT TESTING

Unit testing is conducted to verify the functional performance of each modular component of the software. Unit testing focuses on the smallest unit of the software design (i.e.), the module. The white-box testing techniques were heavily employed for unit testing.

FUNCTIONAL TESTS

Functional test cases involved exercising the code with nominal input values for which the expected results are known, as well as boundary values and special values, such as logically related inputs, files of identical elements, and empty files.

Three types of tests in Functional test:

- Performance Test
- Stress Test
- Structure Test

PERFORMANCE TEST

It determines the amount of execution time spent in various parts of the unit, program throughput, and response time and device utilization by the program unit.

STRESS TEST

Stress Test is those test designed to intentionally break the unit. A Great deal can be learned about the strength and limitations of a program by examining the manner in which a programmer in which a program unit breaks.

STRUCTURED TEST

Structure Tests are concerned with exercising the internal logic of a program and traversing particular execution paths. The way in which White-Box test strategy was employed to ensure that the test cases could Guarantee that all independent paths within a module have been have been exercised at least once.

- Exercise all logical decisions on their true or false sides.
- Execute all loops at their boundaries and within their operational bounds.
- Exercise internal data structures to assure their validity.
- Checking attributes for their correctness.
- Handling end of file condition, I/O errors, buffer problems and textual errors in output information

INTEGRATION TESTING

Integration testing is a systematic technique for construction the program structure while at the same time conducting tests to uncover errors associated with interfacing. i.e., integration testing is the complete testing of the set of modules which makes up the product. The objective is to take untested modules and build a program structure tester should identify critical modules. Critical modules should be tested as early as possible. One approach is to wait until all the units have passed testing, and then combine them and then tested. This approach is evolved from unstructured testing of small programs. Another strategy is to construct the product in increments of tested units. A small set of modules are integrated together and tested, to which another module is added and tested in combination. And so on. The advantages of this approach are that, interface dispenses can be easily found and corrected.

The major error that was faced during the project is linking error. When all the modules are combined the link is not set properly with all support files. Then we checked out for interconnection and the links. Errors are localized to the new module and its intercommunications. The product development can be staged, and modules integrated in as they complete unit testing. Testing is completed when the last module is integrated and tested.

TESTING TECHNIQUES / TESTING STRATEGIES

TESTING

Testing is a process of executing a program with the intent of finding an error. A good test case is one that has a high probability of finding an as-yet -undiscovered error. A successful test is one that uncovers an as-yet- undiscovered error. System testing is the stage of implementation, which is aimed at ensuring that the system works accurately and efficiently as expected before live operation commences. It verifies that the whole set of programs hang together. System testing requires a test consists of several key activities and steps for run program, string, system and is important in adopting a successful new system. This is the last chance to detect and correct errors before the system is installed for user acceptance testing.

The software testing process commences once the program is created and the documentation and related data structures are designed. Software testing is essential for correcting errors. Otherwise the program or the project is not said to be complete. Software testing is the critical element of software quality assurance and represents the ultimate the review of specification design and coding. Testing is the process of executing the program with the intent of finding the error. A good test case design is one that as a probability of finding an yet undiscovered error. A successful test is one that uncovers an yet undiscovered error. Any engineering product can be tested in one of the two ways:

WHITE BOX TESTING

This testing is also called as Glass box testing. In this testing, by knowing the specific functions that a product has been design to perform test can be conducted that demonstrate each function is fully operational at the same time searching for errors in each function. It is a test case design method that uses the control structure of the procedural design to derive test cases. Basis path testing is a white box testing.

Basis path testing:

- Flow graph notation
- Cyclometric complexity
- Deriving test cases
- Graph matrices Control

BLACK BOX TESTING

In this testing by knowing the internal operation of a product, test can be conducted to ensure that “all gears mesh”, that is the internal operation performs according to specification and all internal components have been adequately exercised. It fundamentally focuses on the functional requirements of the software.

The steps involved in black box test case design are:

- Graph based testing methods
- Equivalence partitioning
- Boundary value analysis

- Comparison testing

SOFTWARE TESTING STRATEGIES:

A software testing strategy provides a road map for the software developer. Testing is a set activity that can be planned in advance and conducted systematically. For this reason a template for software testing a set of steps into which we can place specific test case design methods should be strategy should have the following characteristics:

- Testing begins at the module level and works “outward” toward the integration of the entire computer based system.
- Different testing techniques are appropriate at different points in time.
- The developer of the software and an independent test group conducts testing.
- Testing and Debugging are different activities but debugging must be accommodated in any testing strategy.

INTEGRATION TESTING:

Integration testing is a systematic technique for constructing the program structure while at the same time conducting tests to uncover errors associated with. Individual modules, which are highly prone to interface errors, should not be assumed to work instantly when we put them together. The problem of course, is “putting them together”- interfacing. There may be the chances of data lost across on another’s sub functions, when combined may not produce the desired major function; individually

acceptable impression may be magnified to unacceptable levels; global data structures can present problems.

PROGRAM TESTING:

The logical and syntax errors have been pointed out by program testing. A syntax error is an error in a program statement that in violates one or more rules of the language in which it is written. An improperly defined field dimension or omitted keywords are common syntax error. These errors are shown through error messages generated by the computer. A logic error on the other hand deals with the incorrect data fields, out-off-range items and invalid combinations. Since the compiler s will not deduct logical error, the programmer must examine the output. Condition testing exercises the logical conditions contained in a module. The possible types of elements in a condition include a Boolean operator, Boolean variable, a pair of Boolean parentheses A relational operator or on arithmetic expression. Condition testing method focuses on testing each condition in the program the purpose of condition test is to deduct not only errors in the condition of a program but also other a errors in the program.

SECURITY TESTING:

Security testing attempts to verify the protection mechanisms built in to a system well, in fact, protect it from improper penetration. The system security must be tested for invulnerability from frontal attack must also be tested for invulnerability from rear attack. During security, the tester places the role of individual who desires to penetrate system.

VALIDATION TESTING

At the culmination of integration testing, software is completely assembled as a package. Interfacing errors have been uncovered and corrected and a final series of software test-validation testing begins. Validation testing can be defined in many ways, but a simple definition is that validation succeeds when the software functions in manner that is reasonably expected by the customer. Software validation is achieved through a series of black box tests that demonstrate conformity with requirement. After validation test has been conducted, one of two conditions exists.

- * The function or performance characteristics confirm to specifications and are accepted.
- * A validation from specification is uncovered and a deficiency created.

Deviation or errors discovered at this step in this project is corrected prior to completion of the project with the help of the user by negotiating to establish a method for resolving deficiencies. Thus the proposed system under consideration has been tested by using validation testing and found to be working satisfactorily. Though there were deficiencies in the system they were not catastrophic

USER ACCEPTANCE TESTING

User acceptance of the system is key factor for the success of any system. The system under consideration is tested for user acceptance by constantly keeping in touch with prospective system and user at the time of developing and making changes whenever required. This is done in regarding to the following points.

SOURCECODE

```
1#!/usr/bin/env python
2# coding: utf-8
3
4# In[1]:
5
6
7import numpy as np
8import os
9import six.moves.urllib as urllib
10import sys
11import tarfile
12import tensorflow.compat.v1 as tf
13tf.disable_v2_behavior()
14import zipfile
15
16from collections import defaultdict
17from io import StringIO
18from matplotlib import pyplot as plt
19from PIL import Image
20
21import cv2
22cap = cv2.VideoCapture(0)
23
24# This is needed since the notebook is stored in the object_detection folder.
25sys.path.append("..")
26
27
28# ## Object detection imports
29# Here are the imports from the object detection module.
30
31# In[3]:
32
33from utils import label_map_util
34
35from utils import visualization_utils as vis_util
36
37
38# # Model preparation
39
40# ## Variables
41#
```

```

42 # Any model exported using the `export_inference_graph.py` tool can be loaded here simply
43 #
44 # By default we use an "SSD with Mobilenet" model here. See the [detection model zoo](http
45
46 # In[4]:
47 MODEL_NAME = 'ssd_mobilenet_v2_coco_2018_03_29'
48 # What model to download.
49 #MODEL_NAME = 'ssd_mobilenet_v1_coco_11_06_2017'
50 MODEL_FILE = MODEL_NAME + '.tar.gz'
51 DOWNLOAD_BASE = 'http://download.tensorflow.org/models/object_detection/'
52
53 # Path to frozen detection graph. This is the actual model that is used for the object de
54 PATH_TO_CKPT = 'F:/Watch & Do Object Detection and Gaze Estimation mod1/object_detection/'
55
56 # List of the strings that is used to add correct label for each box.
57 PATH_TO_LABELS = os.path.join('data', 'mscoco_complete_label_map.pbtxt')
58
59 NUM_CLASSES = 90
60
61
62 # ## Download Model
63
64 # In[5]:
65
66 opener = urllib.request.URLopener()
67 opener.retrieve(DOWNLOAD_BASE + MODEL_FILE, MODEL_FILE)
68 tar_file = tarfile.open(MODEL_FILE)
69 for file in tar_file.getmembers():
70     file_name = os.path.basename(file.name)
71     if 'frozen_inference_graph.pb' in file_name:
72         tar_file.extract(file, os.getcwd())
73
74
75 # ## Load a (frozen) Tensorflow model into memory.
76
77 # In[6]:
78
79 detection_graph = tf.Graph()
80 with detection_graph.as_default():
81     od_graph_def = tf.GraphDef()
82     with tf.io.gfile.GFile(PATH_TO_CKPT, 'rb') as fid:
83         serialized_graph = fid.read()

```

```

84 od_graph_def.ParseFromString(serialized_graph)
85 tf.import_graph_def(od_graph_def, name='')
86
87
88 ## Loading label map
89 # Label maps map indices to category names, so that when our convolution network predicts
90
91
92 # In[7]:
93
94 label_map = label_map_util.load_labelmap(PATH_TO_LABELS)
95 categories = label_map_util.convert_label_map_to_categories(label_map, max_num_classes=NUM
96 category_index = label_map_util.create_category_index(categories)
97
98
99 ## Helper code
100
101 # In[8]:
102
103 def load_image_into_numpy_array(image):
104     (im_width, im_height) = image.size
105     return np.array(image.getdata()).reshape(
106         (im_height, im_width, 3)).astype(np.uint8)
107
108
109 # # Detection
110
111 # In[9]:
112
113 # For the sake of simplicity we will use only 2 images:
114 # image1.jpg
115 # image2.jpg
116 # If you want to test the code with your images, just add path to the images to the TEST_I
117 PATH_TO_TEST_IMAGES_DIR = 'test_images'
118 TEST_IMAGE_PATHS = [ os.path.join(PATH_TO_TEST_IMAGES_DIR, 'image{}.jpg'.format(i)) for i
119
120 # Size, in inches, of the output images.
121 IMAGE_SIZE = (12, 8)
122
123
124 # In[10]:
125

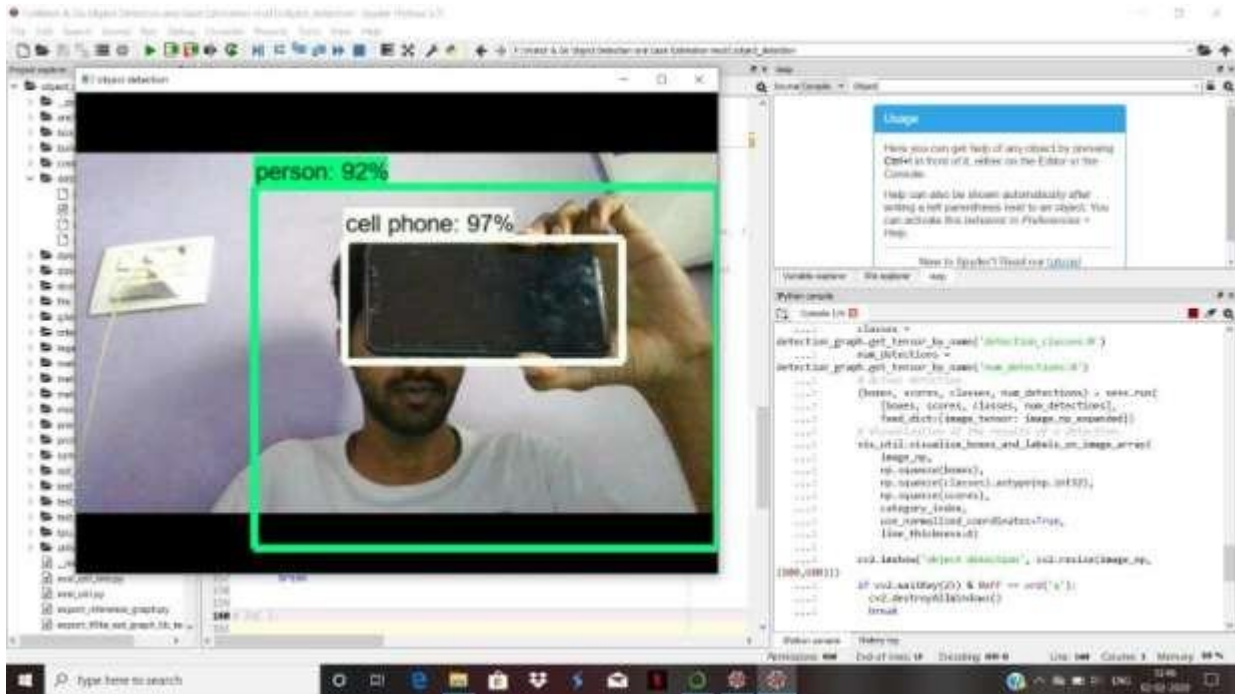
```

```

125
126 with detection_graph.as_default():
127     with tf.Session(graph=detection_graph) as sess:
128         while True:
129             ret, image_np = cap.read()
130             # Expand dimensions since the model expects images to have shape: [1, None, None, 3]
131             image_np_expanded = np.expand_dims(image_np, axis=0)
132             image_tensor = detection_graph.get_tensor_by_name('image_tensor:0')
133             # Each box represents a part of the image where a particular object was detected.
134             boxes = detection_graph.get_tensor_by_name('detection_boxes:0')
135             # Each score represent how level of confidence for each of the objects.
136             # Score is shown on the result image, together with the class label.
137             scores = detection_graph.get_tensor_by_name('detection_scores:0')
138             classes = detection_graph.get_tensor_by_name('detection_classes:0')
139             num_detections = detection_graph.get_tensor_by_name('num_detections:0')
140             # Actual detection.
141             (boxes, scores, classes, num_detections) = sess.run(
142                 [boxes, scores, classes, num_detections],
143                 feed_dict={image_tensor: image_np_expanded})
144             # Visualization of the results of a detection.
145             vis_util.visualize_boxes_and_labels_on_image_array(
146                 image_np,
147                 np.squeeze(boxes),
148                 np.squeeze(classes).astype(np.int32),
149                 np.squeeze(scores),
150                 category_index,
151                 use_normalized_coordinates=True,
152                 line_thickness=8)
153
154             cv2.imshow('object detection', cv2.resize(image_np, (800,600)))
155             if cv2.waitKey(25) & 0xFF == ord('q'):
156                 cv2.destroyAllWindows()
157                 break
158
159
160 # In[ ]:
161
162
163
164
165

```

Screenshots:



PAPER WORK

15

ORIGINALITY REPORT

0%

SIMILARITY INDEX

0%

INTERNET SOURCES

0%

PUBLICATIONS

0%

STUDENT PAPERS

PRIMARY SOURCES

1

Submitted to Postgraduate Schools -
Limkokwing University of Creative Technology
Student Paper

<1%

Exclude quotes On

Exclude matches Off

Exclude bibliography On

REFERENCES

- [1] O. B. Sezer, E. Dogdu, and A. M. Ozbayoglu, "Context-aware computing, learning, and big data in Internet of Things: A survey," *IEEE Internet Things J.*, vol. 5, no. 1, pp. 1-27, Feb. 2018.
- [2] J. B. B. Neto, T. H. Silva, R. M. Assunç ao, R. A. F. Mini, and A. A. F. Loureiro, "Sensing in the collaborative Internet of Things," *IEEE Sensors J.*, vol. 15, no. 3, pp. 6607-6632, Mar. 2015.
- [3] C. Perera, A. Zaslavsky, M. Compton, P. Christen, and D. Georgakopoulos, "Semantic-driven configuration of Internet of Things middleware," in *Proc. SKG*, Beijing, China, 2013, pp. 66-73.
- [4] R. Mahmoud, T. Yousuf, F. Aloul, and I. Zuolkernan, "Internet of Things (IoT) security: Current status, challenges and prospective measures," in *Proc. ICITST*, London, U.K., 2015, pp. 336-341.
- [5] Y. Zou et al., "GRfid: A device-free RFID-based gesture recognition system," *IEEE Trans. Mobile Comput.*, vol. 16, no. 2, pp. 381-393, . 2017.
- [6] K. Wu et al., "HJam: Attachment transmission in WLANs," in *Proc. IEEE INFOCOM*, Orlando, FL, USA, 2012, pp. 1449-1457.
- [7] Y. Zou, W. Liu, K. Wu, and L. M. Ni, "Wi-Fi radar: Recognizing human behavior with commodity Wi-Fi," *IEEE Commun. Mag.*, vol. 55, no. 10, pp. 105-111, Oct. 2017.
- [8] K. Wu et al., "CSI-based indoor localization," *IEEE Trans. Parallel Distrib. Syst.*, vol. 24, no. 7, pp. 1300-1309, Jul. 2013.
- [9] W. Kluge et al., "A fully integrated 2.4-GHz IEEE 802.15.4-compliant transceiver for ZigBeeTM applications," *IEEE J. Solid-State Circuits*, vol. 41, no. 12, pp. 2767-2775, Dec. 2006.
- [10] H. B. Pandya and T. A. Champaneria, "Internet of Things: Survey and case studies," in *Proc. EESCO*, Visakhapatnam, India, 2015, pp. 1-6.