

“HAND GESTURE CONTROL”

A Project Report

Submitted By

PRITISH ARORA - 1614110978

PRATIK MALI - 1614110352

SYED AUSAF HAIDER - 1614110213

NOOR SHAIKH - 1714110769

In partial fulfilment of the requirement

For the degree of

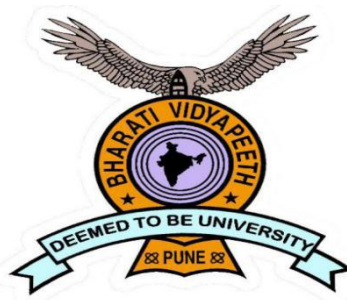
BACHELOR OF TECHNOLOGY

In

COMPUTER ENGINEERING

Under the guidance of

PROF.N.S.Patil



DEPARTMENT OF COMPUTER ENGINEERING

BHARATI VIDYAPEETH

(DEEMED TO BE UNIVERSITY),

COLLEGE OF ENGINEERING, PUNE- 43

2019-2020



CERTIFICATE

This is to certify that the project report titled **HAND GESTURE CONTROL**, has been carried out by the following students in partial fulfillment of the degree of **BACHELOR OF TECHNOLOGY** in Computer Engineering of Bharati Vidyapeeth (Deemed to be) University Pune, during the academic year 2019-2020.

Team:

1. PRITISH ARORA (1614110978)
2. PRATIK MALI(1614110352)
3. SYED AUSAF HAIDER(1614110213)
4. NOOR SHAIKH(1714110769)

Name of Guide

Prof. N.S.Patil

Prof. S.B Vanjale

Department of Computer Engineering

Name and Signature of External
Examiner

Date:

Place: Pune

Acknowledgement

I am extremely grateful and remain indebted to my guide Prof N.S Patil for being a source of inspiration and for her constant support in the Design, Implementation and Evaluation of the project. I am thankful to her for constant constructive criticism and invaluable suggestions, which benefited me a lot while developing the project on “HAND GESTURE CONTROL”, She has been a constant source of inspiration and motivation for hard work, he has been very co-operative throughout this project work. Through this column, it would be my utmost pleasure to express my warm thanks to her for the encouragement, co-operation and consent without which we might not be able to accomplish this project. We would also like to expand our deepest gratitude to all those who have directly and indirectly guided us in completion this assignment.

INDEX

Sr no	Title	Pg.no
	Title page	
	Certificate	
	Acknowledgement	
	Table of contents	
	List of figures	
	List of Tables	
	Abstract	
Chapter1	INTRODUCTION	1
Chapter2	LITERATURE REVIEW	5
	2.1 EXISITING SYSTEMS	7
	2.2 SIGNIFICANCE OF MODEL	8
	2.3 FUTURE SCOPE	9
Chapter3	CASE STUDIES	10
	3.1 MICROSOFT KINECT AND ITS EFFECTS	11
	3.2 GESTURE RECOGNITION USING KINECT	13
	3.2 HGC USING LEAP MOTION	15
Chapter4	PROPOSED SYSTEM	19
	4.1 PROBLEM DEFINATION	20
	4.2 PROPOSED SYSTEM	20
	4.3 WORKING	21
	4.4 YOLOV3	21
Chapter5	SOFTWARE REQUIREMENT SPECIFICATION	30
Chapter6	DESIGN METHODOLOGY	32
Chapter7	IMPLEMENTATION	36
Chapter8	APPLICATION	46
	8.1 APPLICATION	47
	8.2 DRAWBACKS	48
	Conclusion	
	References	

LIST OF FIGURES

Sr. No.	Topic	Page No.
1	Kinect Sensor	12
2	Kinect for xbox	14
3	Angular Regions of palm plane	17
4	Features of ROI	15
5	Architecture	20
6	Scope of AI,ML and DL	33
7	Yolov3	21
8	Yolov3: Input Image	23
9	Yolov3: Grid	23
10	8D Vector	24
11	Yolov3:Working	24
12	Yolov3: Anchor Boxes 1	25
13	Yolov3: Anchor Boxes 2	26
14	Yolov3:Final Anchor Boxes	26
15	Yolov3:Vector for Anchor Boxes	27

16	8D Vector for AB	27
17	Yolov3: Training	28
18	Architecture	37
19	Bounding box	38
20	Vector	38
21	Training	41
22	Real-time Detection	42
23	Server Execution	43
24	Client application interface	43

LIST OF TABLES

1	Input Gestures	39
2	Result	41

ABSTRACT

Computer vision as a field is an intellectual frontier. Like any frontier, it is exciting and disorganized, and there is often no reliable authority to appeal to. Many useful ideas have no theoretical grounding, and some theories are useless in practice; developed areas are widely scattered, and often one looks completely inaccessible from the other.

The goal of computer vision is to understand the content of digital images. Typically, this involves developing methods that attempt to reproduce the capability of human vision. Using this concept our goal is to create a hand gesture control system that is a system that can detect and recognize a gesture being made by the users bare hands and then triggering certain functions of a corresponding application on a target mobile device.

CHAPTER 1

INTRODUCTION

INTRODUCTION

Hand gesture recognition is very significant for human-computer interaction. In this work, we present a novel real-time method for hand gesture recognition. In our framework, the hand region is extracted from the background with the background subtraction method. Then, the palm and fingers are segmented so as to detect and recognize the fingers. Finally, a rule classifier is applied to predict the labels of hand gestures. The experiments on the data set of 1300 images show that our method performs well and is highly efficient. Moreover, our method shows better performance than a state-of-art method on another data set of hand gestures.

1. Introduction

As we know, the vision-based technology of hand gesture recognition is an important part of human-computer interaction (HCI). In the last decades, keyboard and mouse play a significant role in human-computer interaction. However, owing to the rapid development of hardware and software, new types of HCI methods have been required. In particular, technologies such as speech recognition and gesture recognition receive great attention in the field of HCI.

Gesture is a symbol of physical behavior or emotional expression. It includes body gesture and hand gesture. It falls into two categories: static gesture and dynamic gesture. For the former, the posture of the body or the gesture of the hand denotes a sign. For the latter, the movement of the body or the hand conveys some messages. Gesture can be used as a tool of communication between computer and human. It is greatly different from the traditional hardware based methods and can accomplish human-computer interaction through gesture recognition. Gesture recognition determines the user intent through the recognition of the gesture or movement of the body or body parts. In the past decades, many researchers have strived to improve the hand gesture recognition technology. Hand gesture recognition has great value in many applications such as sign language recognition, augmented reality (virtual reality), sign language interpreters for the disabled, and robot control.

2. What is Gesture recognition?

Gesture recognition is a topic in computer science and language technology with the goal of interpreting human gestures via mathematical algorithms. Gestures can originate from any bodily motion or state but commonly originate from the face or hand. Current[when?] focuses in the field include emotion recognition from face and hand gesture recognition. Users can use simple gestures to control or interact with devices without physically touching them. Many approaches have been made using cameras and computer vision algorithms to interpret sign language. However, the identification and recognition of posture, gait, proxemics, and human behaviors is also the subject of gesture recognition techniques.[1] Gesture recognition can be seen as a way for computers to begin to understand human body language, thus building a richer bridge between machines and humans than primitive text user interfaces or even GUIs (graphical user interfaces), which still limit the majority of input to keyboard and mouse and interact naturally without any mechanical devices. Using the concept of gesture recognition, it is possible to point a finger at this point will move accordingly. This could make conventional input on devices such and even redundant.

Gesture recognition can be conducted with techniques from computer vision and image processing.

The literature includes ongoing work in the computer vision field on capturing gestures or more general human pose and movements by cameras connected to a computer.

Gesture recognition and pen computing: Pen computing reduces the hardware impact of a system and also increases the range of physical world objects usable for control beyond traditional digital objects like keyboards and mice. Such implementations could enable a new range of hardware that does not require monitors. This idea may lead to the creation of holographic display. The term gesture recognition has been used to refer more narrowly to non-text-input handwriting symbols, such as inking on a graphics tablet, multi-touch gestures, and mouse gesture recognition. This is computer interaction through the drawing of symbols with a pointing device cursor.

COMPUTER VISION

Computer vision is an **interdisciplinary scientific field** that deals with how computers can be made to gain high-level understanding from **digital images** or **videos**. From the perspective of **engineering**, it seeks to automate tasks that the **human visual system** can do.

Computer vision tasks include methods for **acquiring, processing, analyzing** and understanding digital images, and extraction of **high-dimensional** data from the real world in order to produce numerical or symbolic information, e.g. in the forms of decisions. Understanding in this context means the transformation of visual images (the input of the retina) into descriptions of the world that can interface with other thought processes and elicit appropriate action. This image understanding can be seen as the disentangling of symbolic information from image data using models constructed with the aid of geometry, physics, statistics, and learning theory.

The **scientific discipline** of computer vision is concerned with the theory behind artificial systems that extract information from images. The image data can take many forms, such as video sequences, views from multiple cameras, or multi-dimensional data from a medical scanner. The technological discipline of computer vision seeks to apply its theories and models to the construction of computer vision systems.

Sub-domains of computer vision include **scene reconstruction**, event detection, **video tracking**, **object recognition**, **3D pose estimation**, learning, indexing, **motion estimation**, and **image restoration**.

Areas of **artificial intelligence** deal with autonomous planning or deliberation for robotic systems to **navigate through an environment**. A detailed understanding of these environments is required to navigate through them. Information about the environment could be provided by a computer vision system, acting as a vision sensor and providing high-level information about the environment and the robot.

Artificial intelligence and computer vision share other topics such as **pattern recognition** and learning techniques. Consequently, computer vision is sometimes seen as a part of the artificial intelligence field or the computer science field in general.

CHAPTER 2

LITERATURE REVIEW

LITERATURE REVIEW

Humans glance at an image and instantly know what objects are in the image, where they are, and how they interact. The human visual system is fast and accurate, allowing us to perform complex tasks like driving with little conscious thought. Deep learning is an artificial intelligence function that imitates the workings of the human brain in processing data and creating patterns for use in decision making. Deep learning is a subset of machine learning in artificial intelligence (AI) that has networks capable of learning unsupervised from data that is unstructured or unlabelled. Fast, accurate algorithms for object detection would allow computers to drive cars without specialized sensors, enable assistive devices to convey real-time scene information to human users, and unlock the potential for general purpose, responsive robotic systems. Current detection systems repurpose classifiers to perform detection. To detect an object, these systems take a classifier for that object and evaluate it at various locations and scales in a test image. Systems like deformable parts models (DPM) use a sliding window approach where the classifier is run at evenly spaced locations over the entire image. More recent approaches like R-CNN use region proposal methods to first generate potential bounding boxes in an image and then run a classifier on these proposed boxes. After classification, post-processing is used to refine the bounding boxes, eliminate duplicate detections, and rescore the boxes based on other objects in the scene . These complex pipelines are slow and hard to optimize because each individual component must be trained separately. We reframe object detection as a single regression problem, straight from image pixels to bounding box coordinates and class probabilities. Using our system, you only look once (YOLO) at an image to predict what objects are present and where they are.

This development has made possible robust recognition which is like to identify fingers and hand gesture recognition in bad condition such as dark light and rough background. At the same time, the interface for natural interaction was required from many users. However, it doesn't reach to this requirement of users. Some of researchers propose media player manipulation interaction using hand motion gesture and glove-based hand gesture interaction. Vision-based hand gesture recognition system was proposed as well. However, the research approaches that propose the mounting of addition-al device on the body are usually troublesome and not natural. Recently, new horizons are open to the HCI field with the development of sensors and technology [2] such as Kinect, Depth-Sense and Leap motion.

2.1. EXISTING SYSTEMS

Many hand gesture recognition techniques are proposed to interact with objects for natural user interface (NUI). They are implemented using various devices and techniques. Vision based, glove based and depth based are widely used in hand gesture recognition.

Glove-based hand gesture recognition

Kenn [6] presents an interface for wearable computing applications using glove-based finger recognition. His paper implemented finger identification for hand gesture. The research provides rich user experience based on accurate recognition rate. However, it is not practical in all situations because the user cannot always carry a glove. The glove is both unnatural and heavy. This condition is not suitable for natural interaction.

Depth-based hand gesture recognition

Yang et al, [3] proposed a hand motion gesture recognition system using kinect depth data. They designed hand motion gestures like wave, forward/backward, move up/down, left/right in a media player application. This system shows the possibility of using kinect for hand motion gesture recognition in a contactless UI. However, the implemented system was not able to recognize fingertip. Therefore, they were not able to provide delicate and natural gestures, such as pinch gesture, spread gesture and flick finger. Raheja et al, [4] proposed a method to recognize and track fingertips and centre of palm using kinect. They track the fingertip by calculating depth image segmentation of hand regions.

Finger Identification Method

We describe a finger identification [4] method for hand gesture recognition. Our system makes interaction based on predefined hand gestures such as mouse dragging gesture and mouse left/right click gesture, so that the finger identification is important for such delicate recognition.

Remote Video Monitoring

As a business owner, one of your top priorities is protecting your property against theft and break-ins as well as dishonest employees. Thanks to advancements in security technology, you can rely on remote security experts to monitor your system live and react quickly to any activity on your site. These solutions work with surveillance cameras you may already have installed, or complete security towers that can be installed at your business.

2.2.SIGNIFICANCE OF MODEL

Gestures are expressive, meaningful body motions, physical movements of the fingers, hands, arms, head, face, or body with the intent to convey information or interact with the environment. In the HCI literature the word gesture has been used to identify many types of hand movements for control of computer process. However, getting one's hand to the place to start creation or manipulation is not considered a gesture, because it is a necessity to move your hand, but it does not contribute to the final product as such. The way you move your hand to reach this point is not important. According to McNeill [17] the dynamic gesture movement consists of three parts :

Approach: body begins to move

Stroke: the gesture itself

Return: return to balanced posture

Gesture is widely divided into static and dynamic. But some gestures have both static and dynamic elements, where the pose is important in one or more of the gesture phases. We present YOLO, a new approach to object detection. Prior work on object detection repurposes classifiers to perform detection. Instead, we frame object detection as a regression problem to spatially separated bounding boxes and associated class probabilities. A single neural network predicts bounding boxes and class probabilities directly from full images in one evaluation. Since the whole detection pipeline is a single network, it can be optimized end-to-end directly on detection performance. Our unified architecture is extremely fast. Our base YOLO model processes images in real-time at 45 frames per second.

2.3.FUTURE SCOPE

Constructing an efficient hand gesture recognition system is an important aspect for easily interaction between human and machine. Object detection is breaking into a wide range of industries, with use cases ranging from personal security to productivity in the workplace. Object detection and recognition is applied in many areas of computer vision, including image retrieval, security, surveillance, automated vehicle systems and machine inspection. Significant challenges stay on the field of object recognition. The possibilities are endless when it comes to future use cases for object detection. Our model has several advantages over classifier-based systems. It looks at the whole image at test time so its predictions are informed by global context in the image. It also makes predictions with a single network evaluation unlike systems like R-CNN which require thousands for a single image. This makes it extremely fast, more than 1000x faster than R-CNN and 100x faster than Fast R-CNN.

2.4.LITERATURE REVIEW

Sr. No.	Title	Volume /Date	Author	Demerits
1	Gesture recognition using depth-based hand tracking for contactless controller application	Consumer Electronics (ICCE), 2012 IEEE International Conference on, pp. 297 -298, 2012	Y. Cheoljong, J. Yujeong, B. Jounghoon, H. David and K. Hanseok	Uses Multiple sensors which increase the types of inputs and hence requires more processing power.
2	A glove-based gesture interface for wearable computing Applications	IFAWC 4th international forum on applied wearable computing 2007, pp. 169-177, 2007.	H. Kenn, F. Megan and R. Sugar	Requires user to wear a glove laid with a circuit, if damaged then no detection can be made.
3	Hand Gesture for Taking Self Portrait	Human-Computer Interaction, Part II, HCII 2011, LNCS 6762, pp. 238–247, 2011	S. Chu and J. Tanaka	Number of gestures defined is very less, application is minimal.
4	Finger Identification and Hand Gesture Recognition Techniques for Natural User Interface	University of Tsukuba	Unseok Lee and Jiro Tanaka	Uses Microsoft Kinect sensor
5	FreeDigiter: a contact-free device for gesture control	Eighth International Symposium on Wearable Computers, 31 Oct.-3 Nov. 2004	Christian Metzger, Matt Anderson and Thad Starner.	Senses via an infrared proximity sensor and a dual axis accelerometer

CHAPTER 3

CASE STUDIES

CASE STUDIES

CASE STUDY I : Microsoft Kinect Sensor and Its Effect

3.1.1. Kinect Sensor



Fig 1. Kinect sensor

The Kinect sensor incorporates several advanced sensing hardware. Most notably, it contains a depth sensor, a color camera, and a four-microphone array that provide full-body 3D motion capture, facial recognition, and voice recognition capabilities. The arrangement of the infrared (IR) projector, the color camera, and the IR camera. The depth sensor consists of the IR projector combined with the IR camera, which is a monochrome complementary metaloxide semiconductor (CMOS) sensor. The depth-sensing technology is licensed from the Israeli company PrimeSense (www.primesense.com). Although the exact technology is not disclosed, it is based on the structured light principle. The IR projector is an IR laser that passes through a diffraction grating and turns into a set of IR dots. Figure 2 shows the IR dots seen by the IR camera. The relative geometry between the IR projector and the IR camera as well as the projected IR dot pattern are known. If we can match a dot observed in an image with a dot in the projector pattern, we can reconstruct it in 3D using triangulation. Because the dot pattern is relatively random, the matching between the IR image and the projector pattern can be done in a straightforward way by comparing small neighborhoods using, for example, normalized cross correlation.

3.1.2. Kinect Skeletal Tracking

In skeletal tracking, a human body is represented by a number of joints representing body parts such as head, neck, shoulders, and arms (see Figure 5a). Each joint is represented by its 3D coordinates. The goal is to determine all the 3D parameters of these joints in real time to

allow fluent interactivity and with limited computation resources allocated on the Xbox 360 so as not to impact gaming performance.

3.1.3. Head-Pose and Facial-Expression Tracking

Head-pose and facial-expression tracking has been an active research area in computer vision for several decades. It has many applications including human-computer interaction, performance-driven facial animation, and face recognition. Most previous approaches focus on 2D images, so they must exploit some appearance and shape models because there are few distinct facial features. They might still suffer from lighting and texture variations, occlusion of profile poses, and so forth. Related research has also focused on fitting morphable models to 3D facial scans. These 3D scans are usually obtained by high-quality laser scanners or structured light systems. Fitting these high-quality range data with a morphable face model usually involves the well-known iterative closest point (ICP) algorithm and its variants. The results are generally good, but these capturing systems are expensive to acquire or operate and the capture process is long. A Kinect sensor produces both 2D color video and depth images at 30 fps, combining the best of both worlds. However, the Kinect's depth information is not very accurate.

3.1.4. Conclusion

The Kinect sensor offers an unlimited number of opportunities for old and new applications. This article only gives a taste of what is possible. Thus far, additional research areas include hand-gesture recognition,⁶ human-activity recognition,⁷ body biometrics estimation (such as weight, gender, or height),⁸ 3D surface reconstruction,⁹ and healthcare applications.¹⁰ Here, I have included just one reference per application area, not trying to be exhaustive.

CASE STUDY II:Gesture Recognition using Microsoft Kinect

3.2.1. Introduction

Kinect® depth camera for recognition of some common gestures. Kinect® interprets a 3D scene information using a projected infrared structured light (fig.1). This 3D scanner system called Light Coding employs a variant of image-based 3D reconstruction. The Kinect® sensor is a horizontal bar connected to a small base with a motorized pivot and is designed to



Fig 2. Kinect for Xbox

be positioned lengthwise above or below the video display. It also has a RGB camera. The depth sensor consists of an infrared laser projector combined with a monochrome CMOS sensor, which captures video data in 3D under any ambient light conditions. Fig 2 shows the RGB camera image and the depth image of the same scene. The depth map is visualized here using colour gradients from white (near) to blue (far). The monochrome depth sensing video stream is in VGA resolution with 2,048 levels of sensitivity. It is shown in fig.

3.2.2. Pre – processing

The first step that we need to do is to isolate the human making the gestures from the background scene. This is done by background subtraction from the depth image of the scene. This was done by using auto thresholding, proposed by Riddler and Calvard , on the depth histogram. shows a typical depth histogram corresponding to video frame shown in fig. 4. The threshold is found from the valley following the first large peak of the histogram. This enables the foreground to be extracted as shown in Fig. 4. The next step is to figure out the position of hand with respect to rest of the body. The histogram of the extracted foreground from the image is shown in fig. 4. As it relates to body parts very close to each other, the histogram does not reveal any more details. To bring in more clarity, we carry out histogram equalization.

3.2.3. Features from ROI

A region of interest (ROI) is created by placing a 14x14 grid on the extracted foreground. The gesture is parameterized using depth variation and motion information content of each cell of the grid. The depth information is extracted by dividing the entire depth gray scale range (0-255) into 10 bins (0-20, 21-30, 31- 40, 41-50, 51-60, 61-80, 81-100, 101-120, 121-165, 166-255). For each cell the number of pixels lying in each bin is counted, and is normalized. This can be visualized as segmenting the histogram of each cell into the 10 specified levels and storing the normalized count of pixels in each bin.

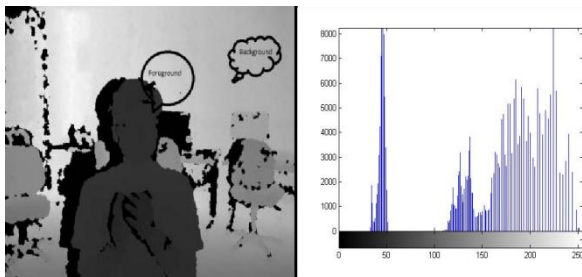


Fig 4. Features of ROI

3.2.4. Training and Testing

A multiclass SVM was used to train the system for the classification of the 8 gestures. A matrix was generated for the entire training data set. Each frame of the video was represented by a row of the matrix. The columns represent the feature points. The number of columns was 1960 (10 bins for 196 cells) for depth related data, and another 196 for normalized count of motion related data for each cell. Thus, the dimension of SVM training matrix was $N \times 2156$ (N being the number of images in the entire training data set). The number of classes was 8 (different gestures). The training data was created with 5 subjects.

3.2.5. CONCLUSIONS

It is shown that using simply the depth images, it is possible to classify hand gestures. For illustration we picked up 8 gestures. But it appears fairly easy to extend it to larger number of gestures. The accuracy of the results could be improved by making use of the skin color information of the color camera. The method is not compute intensive, as very few calculations are involved to extract the features. This would be much faster than a method dealing with RGB components for shape and optical flow for motion.

CASE STUDY III:HAND GESTURE RECOGNITION WITH LEAP MOTION AND KINECT DEVICES

3.3.1. Introduction

The recent introduction of the Leap Motion device has opened new opportunities for gesture recognition. Differently from the Kinect, this device is explicitly targeted to hand gesture recognition and directly computes the position of the fingertips and the hand orientation. Compared to depth cameras like the Kinect and similar devices, it produces a more limited amount of information (only a few keypoints instead of the complete depth map) and its interaction zone is rather limited but the extracted data is more accurate and it is not necessary to perform image processing tasks to extract the relevant points. The Leap Motion software recognizes a few movement patterns only, like swipe and tap, but the exploitation of Leap Motion data for gesture recognition purposes is still an almost unexplored field.

To detect gestures from the data acquired by the Leap Motion. A set of relevant features is extracted from the data produced by the sensor and fed into a SVM classifier in order to recognize the performed gestures. The same gestures have also been acquired with a Kinect and this paper shows both the comparison of the performance that can be obtained from the data of the two devices, and how to combine them together to improve the recognition accuracy.

3.3.2. GENERAL OVERVIEW

In the first step, the hand attributes are gathered from the Leap Motion and the depth acquired from the Kinect. Then a set of relevant features is extracted from the data acquired. The two sensors provide different data, therefore different features set have been extracted from each of the two sensors. Finally a multi-class Support Vector Machine classifier is applied to the extracted features in order to recognize the performed gesture.

3.3.3. FEATURES EXTRACTION FROM LEAP MOTION DATA

Leap device that will be used in the proposed gesture recognition system, namely:

- **Position of the fingertips** F_i , $i = 1, \dots, N$ represent the 3D positions of the detected fingertips (N is the number of recognized fingers). Note that the device is not able to associate each 3D position to a particular finger.

- **Palm center C** roughly corresponds to the center of the palm region in the 3D space.
- **Hand orientation** based on two unit vectors, **h** is pointing from the palm center to the fingers, while **n** is perpendicular to the hand (palm) plane pointing downward from the palm center. However, their estimation is not very accurate and depends on the fingers arrangement.

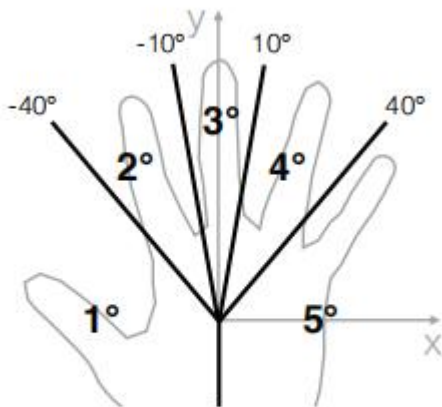


Fig. 3. Angular regions in the palm plane.

3.3.4. FEATURES EXTRACTION FROM KINECT DATA AND GESTURE CLASSIFICATION

Gestures have been acquired with both a Leap Motion and a Kinect device. For the features extraction from Kinect data, we employed a pipeline made of two main steps, i.e., the hand is firstly extracted from the acquired depth and color data and then two different types of features are computed from the 3D points corresponding to the hand..

Assume a training set containing data from M users, to perform the grid search we divide the space of parameters (C, γ) of the RBF kernel with a regular grid and for each couple of parameters the training set is divided into two parts, one containing $M - 1$ users for training and the other one the remaining user for validation and performance evaluation. We repeat the procedure changing each time the user used for the validation and we select the couple of parameters that give the best accuracy on average. Finally we train the SVM on all the

M users of the training set with the optimal parameters.

3.3.5. EXPERIMENTAL RESULTS

The performed gestures have been acquired at the same time with both a Leap Motion device and a Kinect sensor. Fingertip distance features allow to obtain an accuracy of about 76%, they are able to recognize the majority of the gestures but G2 and G3 are easily confused. This is due to the limited accuracy of the hand direction estimation from the Leap Motion software that causes an unreliable matching between the fingertip of these gestures and the corresponding angular region. This inaccuracy is partially solved with the other two features but a slightly lower overall accuracy is obtained: 74.2% from the fingertip angles and 73% from the elevations. An interesting observation is that the 3 feature descriptors capture different properties of the performed gesture and by combining them together it is possible to improve the recognition accuracy, e.g., by combining distances and elevations, an accuracy of about 80% can be reached. .By combining all the 3 features together, 81% of accuracy can be obtained, and this represents the best accuracy that can be extracted from Leap Motion data with the proposed approach..

3.3.6. CONCLUSIONS

Different feature sets have been used to deal with the different nature of data provided by the two devices, the Leap Motion provides a higher level but more limited data description while Kinect provides the full depth map. Even if the data provided by the Leap Motion is not completely reliable, since some fingers might not be detected, the proposed set of features and classification algorithm allows to obtain a good overall accuracy. The more complete description provided by the depth map of the Kinect allows to capture other properties missing in the Leap Motion output and by combining the two devices a very good accuracy can be obtained. Experimental results show also that the assignment of each finger to a specific angular region leads to a considerable increase of performance.

CHAPTER 4

PROPOSED SYSTEM

PROPOSED SYSTEM

4.1. Problem definition

The intention of this project is to design a intelligent, computer vision based system which can make use of commonly available resources to establish control of a device or an application in a efficient manner. The goal here is to establish an interface between a human and a computing device in which interaction can be done using simple and intuitive hand gestures.

4.2. Proposed Solution

As observed, most of the existing systems for establishing hand gesture control need adequate hardware like a hand glove or Kinect sensor, In proposed system, we attempt to leverage the power of a simple camera sensor which is commonly available in every smart phone these days. The images of the gestures can be captured from these camera sensors and these images can be pre-processed on the mobile devices which reduces the processing overhead for the server and hence the resultant image can be sent to a server which is nothing but a vision system working on the principles of computer vision. The server will implement YOLOv3 algorithm on the received input image make a prediction about the gesture contained in the image and generate a flag or a label string which corresponds to the gesture predicted by the server. This label string can then be sent back to the respective client mobile device which intern may trigger a function call therefore achieving control of the device (may or may not be a specific application).

4.3. WORKING:

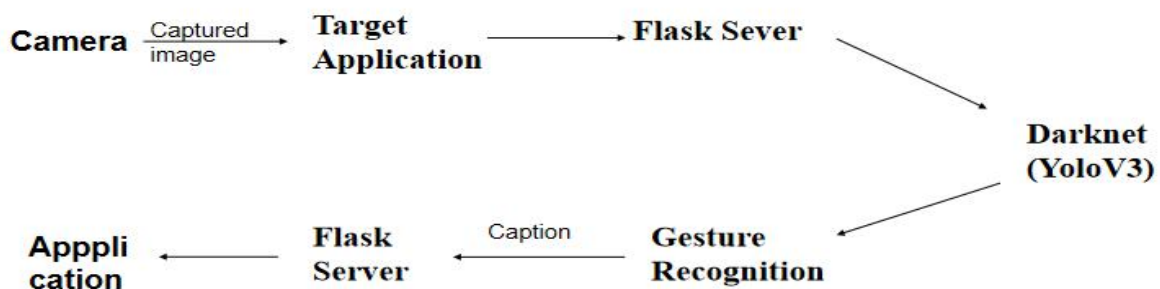


Fig 5. Architecture

1. Image is captured from the camera sensor.
2. Image is stored and pre-processed so that the size of image is reduced and also the processing overhead is brought to an optimal.
3. The resultant is sent over a network to the Flask Server.
4. The flask server activates the darknet framework which is nothing but a script that implements YOLOv3 to make predictions.
5. The flask server then generates a Caption or a label for the detected gesture.
6. The server then sends this caption to application.
7. A corresponding function call is made on basis of the caption received by the application.

4.4. YOLOV3:

YOU ONLY LOOK ONCE:

We present YOLO, a new approach to object detection. Prior work on object detection repurposes classifiers to perform detection. Instead, we frame object detection as a regression problem to spatially separated bounding boxes and associated class probabilities. A single neural network predicts bounding boxes.

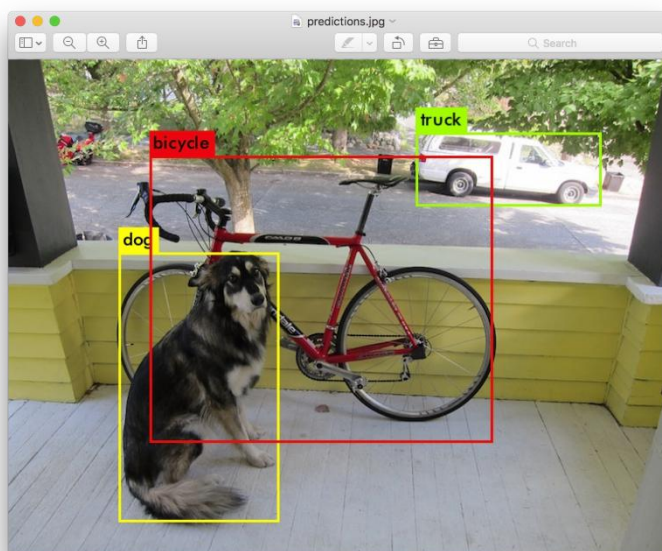


Fig 7: YOLO v3

YOLO is refreshingly simple: see Figure 1. A single convolutional network simultaneously predicts multiple bounding boxes and class probabilities for those boxes. YOLO trains on full images and directly optimizes detection performance.

Second, YOLO reasons globally about the image when making predictions. Unlike sliding window and region proposal-based techniques, YOLO sees the entire image during training and test time so it implicitly encodes contextual information about classes as well as their appearance. Fast R-CNN, a top detection method [14], mistakes background patches in an image for objects because it can't see the larger context. YOLO makes less than half the number of background errors compared to Fast R-CNN.

Third, YOLO learns generalizable representations of objects. When trained on natural images and tested on artwork, YOLO outperforms top detection methods like DPM and R-CNN by a wide margin. Since YOLO is highly generalizable it is less likely to break down when applied to new domains or unexpected inputs.

YOLOv3 is extremely fast and accurate. In mAP measured at .5 IOU YOLOv3 is on par with Focal Loss but about 4x faster. Moreover, you can easily tradeoff between speed and accuracy simply by changing the size of the model, no retraining required!

ALGORITHM (YOLOV3):

Now that we have grasp on why YOLO is such a useful framework, let's jump into how it actually works. In this section, I have mentioned the steps followed by YOLO for detecting objects in a given image.

- YOLO first takes an input image:

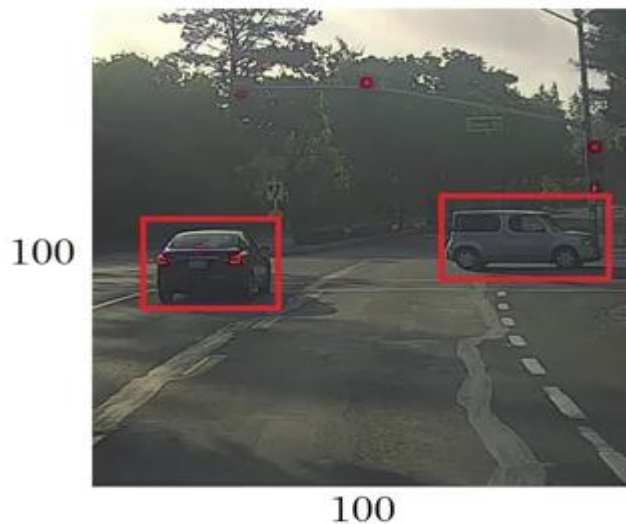


Fig 8: YOLOv3: Input Image

- The framework then divides the input image into grids (say a 3 X 3 grid):

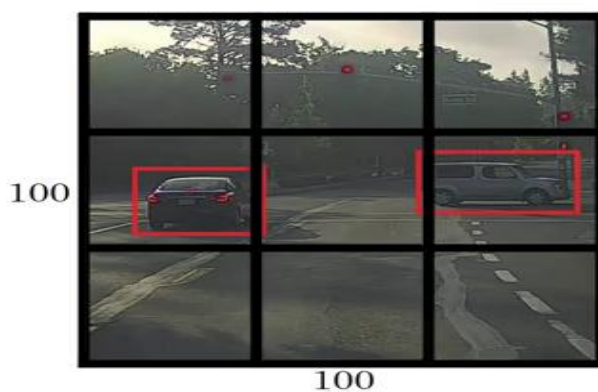


Fig 9: YOLOv3: Grid

- Image classification and localization are applied on each grid. YOLO then predicts the bounding boxes and their corresponding class probabilities for objects (if any are found, of course).

Pretty straightforward, isn't it? Let's break down each step to get a more granular understanding of what we just learned.

We need to pass the labelled data to the model in order to train it. Suppose we have divided the image into a grid of size 3 X 3 and there are a total of 3 classes which we want the objects to be classified into. Let's say the classes are Pedestrian, Car, and Motorcycle respectively. So, for each grid cell, the label y will be an eight dimensional vector:

y =	pc
	bx
	by
	bh
	bw
	c1
	c2
	c3

Fig 10: 8 dimensional vector

Here,

- p_c defines whether an object is present in the grid or not (it is the probability)
- b_x, b_y, b_h, b_w specify the bounding box if there is an object
- c_1, c_2, c_3 represent the classes. So, if the object is a car, c_2 will be 1 and c_1 & c_3 will be 0, and so on

Since there is an object in this grid, p_c will be equal to 1. b_x, b_y, b_h, b_w will be calculated relative to the particular grid cell we are dealing with. Since car is the second class, $c_2 = 1$ and c_1 and $c_3 = 0$. So, for each of the 9 grids, we will have an eight dimensional output vector. This output will have a shape of $3 \times 3 \times 8$.

So now we have an input image and it's corresponding target vector. Using the above example (input image – $100 \times 100 \times 3$, output – $3 \times 3 \times 8$), our model will be trained as follows:

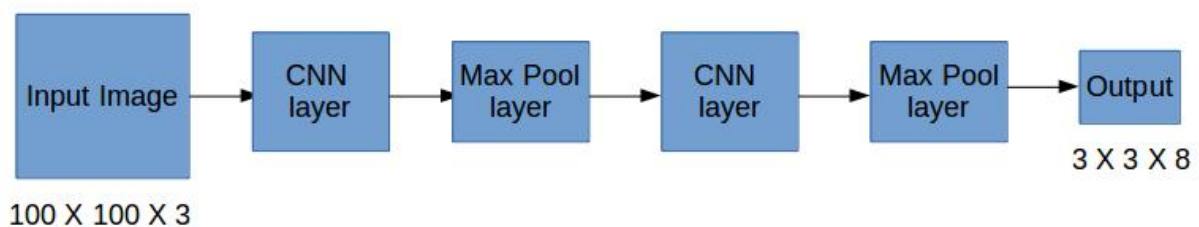


Fig 11: Yolov3: Working

We will run both forward and backward propagation to train our model. During the testing phase, we pass an image to the model and run forward propagation until we get an output y .

In order to keep things simple, I have explained this using a 3 X 3 grid here, but generally in real-world scenarios we take larger grids (perhaps 19 X 19).

Even if an object spans out to more than one grid, it will only be assigned to a single grid in which its mid-point is located. We can reduce the chances of multiple objects appearing in the same grid cell by increasing the more number of grids (19 X 19, for example).

ANCHOR BOXES

Anchor Boxes

We have seen that each grid can only identify one object. But what if there are multiple objects in a single grid? That can so often be the case in reality. And that leads us to the concept of anchor boxes. Consider the following image, divided into a 3 X 3 grid:

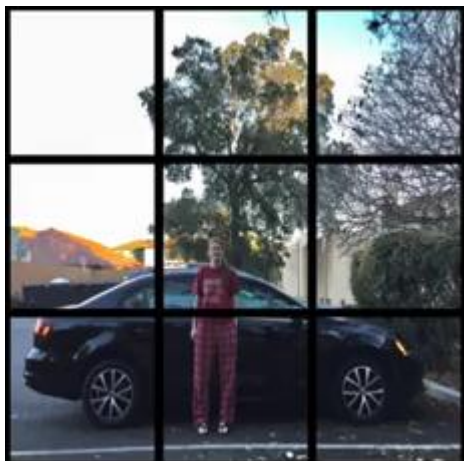


Fig 12: Yolov3: Anchor boxes 1

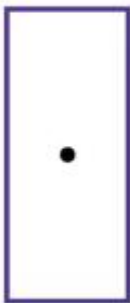
Remember how we assigned an object to a grid? We took the midpoint of the object and based on its location, assigned the object to the corresponding grid. In the above example, the midpoint of both the objects lies in the same grid. This is how the actual bounding boxes for the objects will be:



Fig 13: Yolov3: Anchor Boxes 2

We will only be getting one of the two boxes, either for the car or for the person. But if we use anchor boxes, we might be able to output both boxes! How do we go about doing this? First, we pre-define two different shapes called anchor boxes or anchor box shapes. Now, for each grid, instead of having one output, we will have two outputs. We can always increase the number of anchor boxes as well. I have taken two here to make the concept easy to understand:

Anchor box 1:



Anchor box 2:



Fig 14. Yolov3: Anchor boxes

This is how the y label for YOLO without anchor boxes looks like:

y =	pc
	bx
	by
	bh
	bw
	c1
	c2
	c3

Fig 15. Yolov3: Vector for anchor box

What do you think the y label will be if we have 2 anchor boxes? I want you to take a moment to ponder this before reading further. Got it? The y label will be:

y =	pc
	bx
	by
	bh
	bw
	c1
	c2
	c3
	pc
	bx
	by
	bh
	bw
	c1
	c2
	c3

Fig 16. Yolov3: 8 Dim. Vector

The first 8 rows belong to anchor box 1 and the remaining 8 belongs to anchor box 2. The objects are assigned to the anchor boxes based on the similarity of the bounding boxes and the anchor box shape. Since the shape of anchor box 1 is similar to the bounding box for the person, the latter will be assigned to anchor box 1 and the car will be assigned to anchor box 2. The output in this case, instead of 3 X 3 X 8 (using a 3 X 3 grid and 3 classes), will be 3 X 3 X 16 (since we are using 2 anchors).

So, for each grid, we can detect two or more objects based on the number of anchors. Let's combine all the ideas we have covered so far and integrate them into the YOLO framework.

Training

The input for training our model will obviously be images and their corresponding y labels. Let's see an image and make its y label:

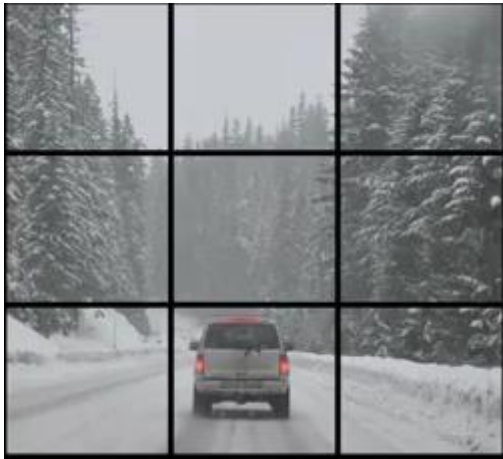


Fig 17. YOLOv3: Training

Consider the scenario where we are using a 3 X 3 grid with two anchors per grid, and there are 3 different object classes. So the corresponding y labels will have a shape of 3 X 3 X 16. Now, suppose if we use 5 anchor boxes per grid and the number of classes has been increased to 5. So the target will be 3 X 3 X 10 X 5 = 3 X 3 X 50. This is how the training process is done – taking an image of a particular shape and mapping it with a 3 X 3 X 16 target (this may change as per the grid size, number of anchor boxes and the number of classes).

Testing

The new image will be divided into the same number of grids which we have chosen during the training period. For each grid, the model will predict an output of shape 3 X 3 X 16 (assuming this is the shape of the target during training time). The 16 values in this prediction will be in the same format as that of the training label. The first 8 values will correspond to anchor box 1, where the first value will be the probability of an object in that grid. Values 2-5 will be the bounding box coordinates for that object, and the last three values will tell us which class the object belongs to. The next 8 values will be for anchor box 2 and in the same format, i.e., first the probability, then the bounding box coordinates, and finally the classes.

Finally, the Non-Max Suppression technique will be applied on the predicted boxes to obtain a single prediction per object.

That brings us to the end of the theoretical aspect of understanding how the YOLO algorithm works, starting from training the model and then generating prediction boxes for the objects. Below are the exact dimensions and steps that the YOLO algorithm follows:

- Takes an input image of shape (608, 608, 3)
- Passes this image to a convolutional neural network (CNN), which returns a (19, 19, 5, 85) dimensional output
- The last two dimensions of the above output are flattened to get an output volume of (19, 19, 425):
 - Here, each cell of a 19 X 19 grid returns 425 numbers
 - $425 = 5 * 85$, where 5 is the number of anchor boxes per grid
 - $85 = 5 + 80$, where 5 is (pc, bx, by, bh, bw) and 80 is the number of classes we want to detect
- Finally, we do the IoU and Non-Max Suppression to avoid selecting overlapping boxes

CHAPTER 5

SOFTWARE REQUIREMENT SPECIFICATION

SOFTWARE REQUIERMENT SPECIFICATION:

- Platform
 - Windows operating system
 - Linux
- Software Requirements
 - Darknet framework
 - Python
 - Anaconda IDE
 - Android studio
- Hardware Specifications
 - Processor: Intel i5 or more
 - Motherboard: Intel® Chipset Motherboard.
 - RAM: 8 GB or more
 - Hard disk: 16 GB hard disk recommended
 - Graphics card

CHAPTER 6

DESIGN METHODOLOGY

6.1.PLAN OF ATTACK:

Machine learning

Machine learning is a field of computer science that gives computers the ability to learn without being programmed explicitly. The power of machine learning is that you can determine how to differentiate using models, rather than using human judgment. The basic steps that lead to machine learning and will teach you how it works are described below in a big picture:

1. Gathering data
2. Preparing that data
3. Choosing a model
4. Training
5. Evaluation
6. Hyper parameter tuning
7. Prediction.

The link is to create a system that answers a particular question. This question answering system called a model is created via a process termed as training. The main goal of training is to create an accurate model that answers our questions correctly, at least for most of the times. But in order to train a model, you also need to collect data on what you'd want to train on. This is where you start and then the rest follows.

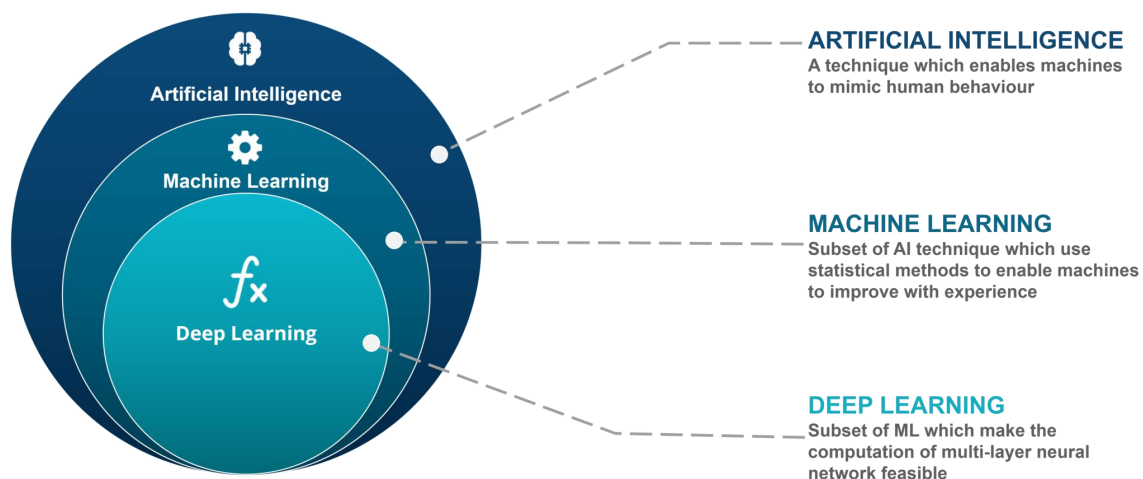


Fig 6: Scope of AI, ML AND DL

The detailed information to the next steps is given below:

1.Gathering Data:-

Once you know exactly what you want and the equipment are in hand, it takes you to the first real step of machine learning- Gathering Data. This step is very crucial as the quality and quantity of data gathered will directly determine how good the predictive model will turn out to be. The data collected is then tabulated and called as Training Data.

2. Data Preparation:

After the training data is gathered, you move on to the next step of machine learning: Data preparation, where the data is loaded into a suitable place and then prepared for use in machine learning training. Here, the data is first put all together and then the order is randomized as the order of data should not affect what is learned.

This is also a good enough time to do any visualizations of the data, as that will help you see if there are any relevant relationships between the different variables, how you can take their advantage and as well as show you if there are any data imbalances present. Also, the data now has to be split into two parts. The first part that is used in training our model, will be the majority of the dataset and the second will be used for the evaluation of the trained model's performance. The other forms of adjusting and manipulation like normalization, error correction, and more take place at this step. Tool Used: OpenLabel For labeling classes in image dataset.

3. Choosing a model:

The next step that follows in the workflow is choosing a model among the many that researchers and data scientists have created over the years. Make the choice of the right one that should get the job done. Algorithm used: YOLOV3

4. Training:

After the previous steps are completed, you then move onto what is often considered the bulk of machine learning called training where the data is used to incrementally improve the model's ability to predict. Tool Used: Darknet Framework.

The training process involves initializing some random values for say A and B of our model, predict the output with those values, then compare it with the model's prediction and then adjust the values so that they match the predictions that were made previously.

This process then repeats and each cycle of updating is called one training step.

5. Evaluation:

Once training is complete, you now check if it is good enough using this step. This is where that dataset you set aside earlier comes into play. Evaluation allows the testing of the model against data that has never been seen and used for training and is meant to be representative of how the model might perform when in the real world.

6. Parameter Tuning:

Once the evaluation is over, any further improvement in your training can be possible by tuning the parameters. There were a few parameters that were implicitly assumed when the training was done. Another parameter included is the learning rate that defines how far the line is shifted during each step, based on the information from the previous training step. These values all play a role in the accuracy of the training model, and how long the training will take.

For models that are more complex, initial conditions play a significant role in the determination of the outcome of training. Differences can be seen depending on whether a model starts off training with values initialized to zeroes versus some distribution of values, which then leads to the question of which distribution is to be used. Since there are many considerations at this phase of training, it's important that you define what makes a model good. These parameters are referred to as hyper parameters. The adjustment or tuning of these parameters depends on the dataset, model, and the training process. Once you are done with these parameters and are satisfied you can move on to the last step.

7. Prediction:

Machine learning is basically using data to answer questions. So this is the final step where you get to answer few questions. This is the point where the value of machine learning is realized. Here you can finally use your model to predict the outcome of what you want. The above-mentioned steps take you from where you create a model to where you predict its output and thus acts as a learning path.

CHAPTER 7

IMPLEMENTATION

Our proposed architecture uses neural network to generate a prediction label from the input image of a hand. The label is then sent to the client music player application over the network. This received label is then used to initiate a function call of the application. The goal of this system is to provide real time detection and hence establish control of the music player application. It functions on a server-client architecture. The server system has a capable Graphics Processing Unit which is vital to the system as higher accuracy and immediate detection is an important feature. The detection of the hand gesture is performed using YOLO(v3) algorithm. This algorithm has proven to be one of the fastest when it comes to static detection. It is based on CNN i.e. Convolutional neural networks but unlike CNN, it uses the input image only once. The architecture of the proposed system is shown in fig 3.1

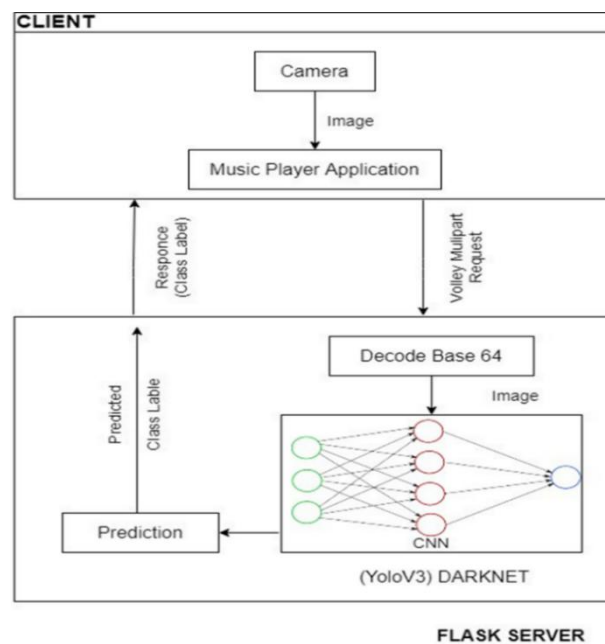


Fig. 7.1: Architecture

Firstly, the flask server is connected to the mobile device. The android application then initiates the camera after one of the audio files is first played manually by touch. The captured image is then passed on to the flask server using Volley-Multipart-Request. HTTP works as a request-response protocol between a client and server. We employ the POST method of HTTP. Volley is an HTTP library that makes networking for Android apps easier and most importantly, faster. One of the biggest advantages of Volley is that it holds all responses in memory during parsing along with Effective request cache and memory management. On receiving the image, the flask server then evokes the python script for darknet framework. The image is provided as input to the script.

Prior work on object detection repurposes classifiers to perform detection. Instead, we frame object detection as a regression problem to spatially separated bounding boxes and associated class probabilities. A single neural network predicts bounding boxes. YOLO trains on full images and directly optimizes detection performance. This unified model has several benefits over traditional methods of object detection. Second, YOLO reasons globally about the image when making predictions. Unlike sliding window and region proposal-based techniques, YOLO sees the entire image during training and test time so it implicitly encodes contextual information about classes as well as their appearance. Third, YOLO learns generalizable

representations of objects. The biggest advantage of using YOLO is its superb speed – it's incredibly fast and can process 45 frames per second. YOLO uses the following steps: -

YOLO first takes an input image. The framework then divides the input image into grids (say a 3X3 grid)

The network predicts 4 coordinates for each bounding box, t_x , t_y , t_w , t_h . If the cell is offset from the top left corner of the image by $(c_x; c_y)$ and the bounding box prior has width and height p_w , p_h , then the predictions correspond to:

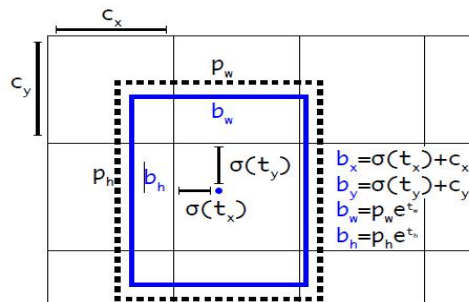


Fig. 7.2: Yolo: Bounding Box

Image classification 13 and localization are applied on each grid. YOLO then predicts the bounding boxes and their corresponding class probabilities for objects. We need to pass the labelled data to the model in order to train it. Then, a vector is generated for each image using the notations to detect objects (if any) in the image. Each box predicts the classes the bounding box may contain using multilabel classification. During training we use binary cross-entropy loss for the class predictions.

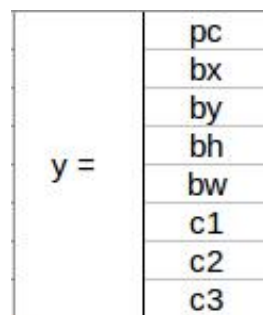


Fig. 7.3: Yolo: Vector

Here, pc defines whether an object is present in the grid (probability) bx , by , bh , bw specify the bounding box if there is an object $c1$, $c2$, $c3$ represent the classes. So, if the object is a car, $c2$ will be 1 and $c1$ & $c3$ will be 0, and so on.

We use a technique called as Intersection over Union which help us to predict bounding boxes. It calculates the intersection over union of the actual bounding box and the predicted bounding box ($IoU = \text{Area of the intersection} / \text{Area of the union}$)

Another technique is introduced to overcome multiple detections by the object detection algorithm. This is called Non-Max Suppression which ensures only single detection per

object. It essentially suppresses boxes with higher IoU to improve the output of YOLO significantly.

Finally, using these vectors we can identify classes of the object detected and use it for further application. We generate appropriate labels for the classes.

The resultant label generated is then converted to string and sent using response class to the android application. The music player application on receiving the response string then triggers the function that corresponds to the hand gesture label. And hence an interface where a user can interact with the application by making hand gestures is established.

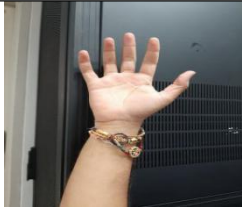

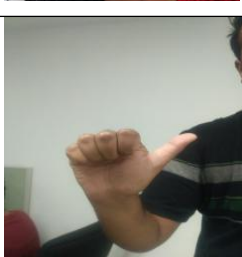
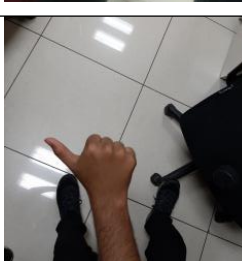
7. IMPLEMENTATION

After the architecture of the system was defined, the implementation of the system was carried out in phases as follows-

7.1 Use Case Definition:

The Music player was defined as the use case. Firstly, the most convenient static hand gestures were identified. These gestures were then assigned to a specific operation of the music player. The table shows the corresponding gestures defined for given operations.

Table 7.1: Input Gestures

Sr.no	Command	Gesture
1.	Play	
2.	Pause	
3.	Forward	
4.	Previous	

7.2 Collection of data:

For the given problem statement the data set was created manually by capturing pictures as no data set particular was found specific to the problem on the internet. The goal of create a dataset which would contribute in providing a more accurate model. Therefore, pictures of the multiple people's hand were taken in randomly varying light conditions. Also, the dataset contained pictures of all required hand gestures from different angles and all potential deviations of the ideal gestures. For example, different pictures of the play gestures contained hands of varying space between the fingers skin tones, light conditions and different angles of the hand gesture etc.

7.3 Data Preparation:

In this step of preparation, the Dataset containing more than 2000 images was firstly cleaned by going through each picture individually to make sure that the captured picture would not deviate the model in the training phase.

7.4 Data Labelling:

The model is based on CNNs which follow the supervised approach of machine learning. So, the cleaned dataset was then labelled using openCV in python. Labelling of data was done in the form of creating bounding boxes around the gesture in the images. The marked bounding boxes generate an annotation file which contains the class of the object, x- centre, y-centre, width and height of the bounding box.

The OpenCv python script will create .txt-file for each .jpg-image-file - in the same directory and with the same name, but with .txt-extension, and put to file: object number and object coordinates on this image, for each object in new line:

```
<object-class> <x_center> <y_center> <width> <height>
```

Where:

<object-class> - integer object number from 0 to (classes-1)

<x_center> <y_center> <width> <height> - float values relative to width and height of image, it can be equal from (0.0 to 1.0]

for example: $\langle x \rangle = \langle \text{absolute_x} \rangle / \langle \text{image_width} \rangle$ or $\langle \text{height} \rangle = \langle \text{absolute_height} \rangle / \langle \text{image_height} \rangle$

attention: <x_center> <y_center> - are center of rectangle (are not top-left corner)

7.5 Training and testing:

Training is the most important part of the implementation. In this phase the required inputs provided to the neural network and it starts adjusting its weight values till a point where the predictions made by the neural network matches the expected outcome. The model selected

for the application is Convolutional neural networks as mentioned earlier and the algorithm applied is Yolov3. The open source darknet framework provides a neural net which is customized by defining the classes and setting optimum batch sizes for the objects and modifying the the filters for different layers. These parameters of the darknet are first set to optimum values as per requirement.

Second step for training is to create a input train.txt file which will be used as input interface for the darknet. Create file train.txt in directory of the darknet build, with filenames of the labelled images, each filename in new line, with path relative to darknet.exe.

The training process here follows the transfer learning approach. Pre-trained weight values for some random objects is provided to the neural net and as the learning process goes on and the avg. loss parameter decreases the weight values are adjusted to suit the current application.

The following image represents the training process of the darknet framework:

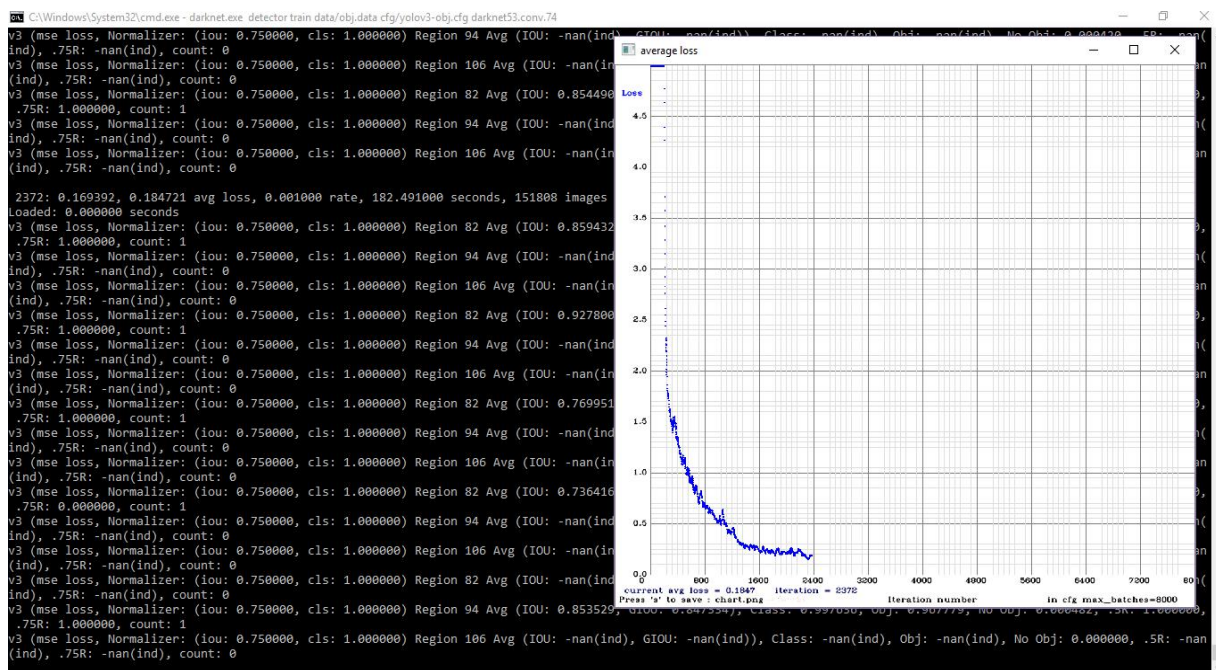


Fig 7.4: Training

For validation of the results of neural net a valid.txt file contains filenames of the labelled images similar to train.txt. The validation set contains 20% of the total dataset.

The graph in the Fig 4.1 represents the avg. loss value on the y-axis and number of iterations on the x-axis. The graph basically provides an insight on how well the neural net has learned. The avg. loss values corresponds to the error in the detection, lesser the value of avg loss better the accuracy of the prediction made by the model.

As the iterations progress the weights of the neural network are updated at regular intervals of 100 iterations (can be customized as needed). When the value of avg. loss becomes constant the training can be stopped. For our particular application 2400 iterations were performed and the latest weight file was stored to make predictions on real time data (fig 4.2).

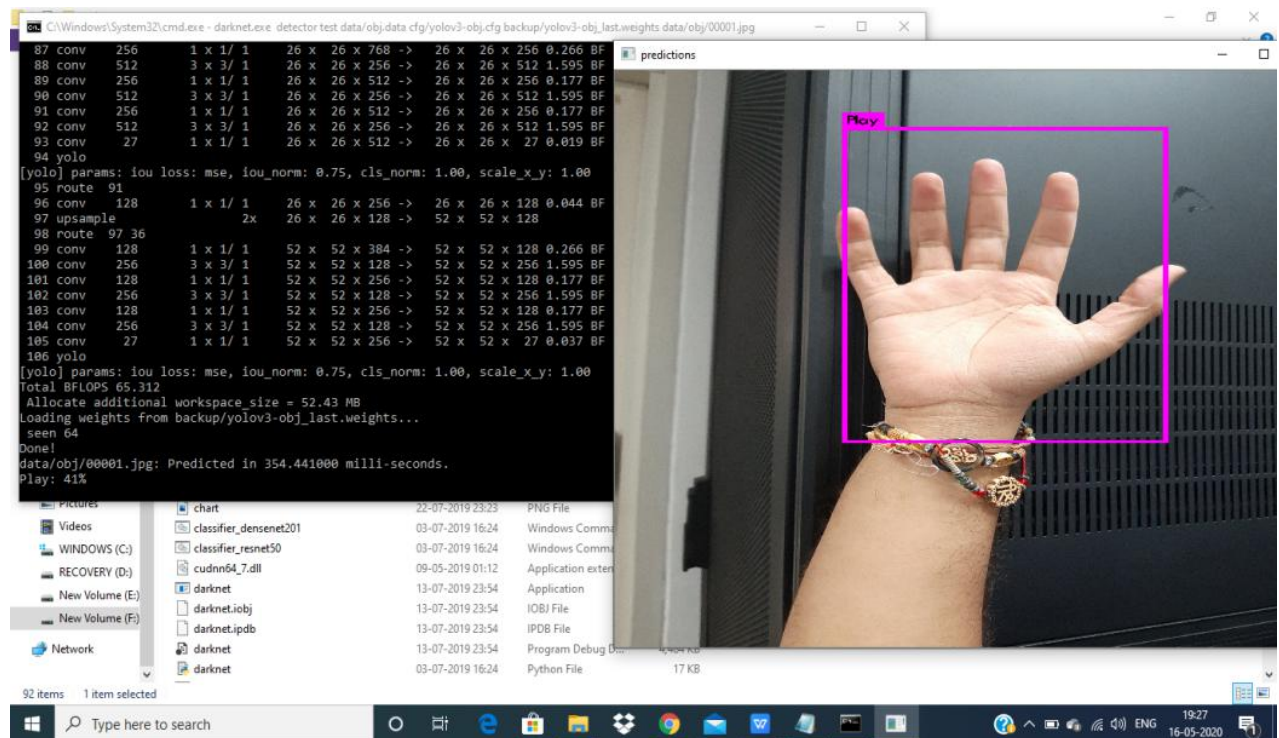


Fig 7.4: Real time detection

7.6 Integration with Android application:

A sample music player android app was developed which used request-response http methods to send image using VolleyMultipartRequest as base64 string to the Server.

A Python flask sever was created which achieved integration with the target android application by accepting the image from the client application. The server First decoded the base64string and then passed the decoded image to the trained neural network in the darknet framework. The output generated label string which represented the name of the detected gesture. This Label string is then passed using http response to the android application.

On receiving the response string, certain if conditions then allow the corresponding intended functions like play or pause to be called. Hence achieving an integration between the Darknet framework and the android application.

```

C:\Windows\System32\cmd.exe
77 conv 512 1 x 1/ 1 13 x 13 x1024 -> 13 x 13 x 512 0.177 BF
78 conv 1024 3 x 3/ 1 13 x 13 x 512 -> 13 x 13 x1024 1.595 BF
79 conv 512 1 x 1/ 1 13 x 13 x1024 -> 13 x 13 x 512 0.177 BF
80 conv 1024 3 x 3/ 1 13 x 13 x 512 -> 13 x 13 x1024 1.595 BF
81 conv 27 1 x 1/ 1 13 x 13 x1024 -> 13 x 13 x 27 0.009 BF
82 yolo
[yolo] params: iou loss: mse, iou_norm: 0.75, cls_norm: 1.00, scale_x_y: 1.00
83 route 79
84 conv 256 1 x 1/ 1 26 x 26 x 512 -> 26 x 26 x 256 0.044 BF
85 upsample 2x 13 x 13 x 256 -> 26 x 26 x 256
86 route 85 61
87 conv 256 1 x 1/ 1 26 x 26 x 768 -> 26 x 26 x 256 0.266 BF
88 conv 512 3 x 3/ 1 26 x 26 x 256 -> 26 x 26 x 512 1.595 BF
89 conv 256 1 x 1/ 1 26 x 26 x 512 -> 26 x 26 x 256 0.177 BF
90 conv 512 3 x 3/ 1 26 x 26 x 256 -> 26 x 26 x 512 1.595 BF
91 conv 256 1 x 1/ 1 26 x 26 x 512 -> 26 x 26 x 256 0.177 BF
92 conv 512 3 x 3/ 1 26 x 26 x 256 -> 26 x 26 x 512 1.595 BF
93 conv 27 1 x 1/ 1 26 x 26 x 512 -> 26 x 26 x 27 0.019 BF
94 yolo
[yolo] params: iou loss: mse, iou_norm: 0.75, cls_norm: 1.00, scale_x_y: 1.00
95 route 91
96 conv 128 1 x 1/ 1 26 x 26 x 256 -> 26 x 26 x 128 0.044 BF
97 upsample 2x 26 x 26 x 128 -> 52 x 52 x 128
98 route 97 36
99 conv 128 1 x 1/ 1 52 x 52 x 384 -> 52 x 52 x 128 0.266 BF
100 conv 256 3 x 3/ 1 52 x 52 x 128 -> 52 x 52 x 256 1.595 BF
101 conv 128 1 x 1/ 1 52 x 52 x 256 -> 52 x 52 x 128 0.177 BF
102 conv 256 3 x 3/ 1 52 x 52 x 128 -> 52 x 52 x 256 1.595 BF
103 conv 128 1 x 1/ 1 52 x 52 x 256 -> 52 x 52 x 128 0.177 BF
104 conv 256 3 x 3/ 1 52 x 52 x 128 -> 52 x 52 x 256 1.595 BF
105 conv 27 1 x 1/ 1 52 x 52 x 256 -> 52 x 52 x 27 0.037 BF
106 yolo
[yolo] params: iou loss: mse, iou_norm: 0.75, cls_norm: 1.00, scale_x_y: 1.00
Total BFLOPS 65.312
Allocate additional workspace_size = 52.43 MB
Loading weights from ./backup/yolov3-obj_2500.weights...
seen 64
Done!
Loaded - names list: data/obj.names, classes = 4
(3516, 4688, 3)
*** 1 Results, color coded by confidence ***
(3516, 4688, 3)
Play: 95.0%
Play

```

Fig 7.5: Server Execution

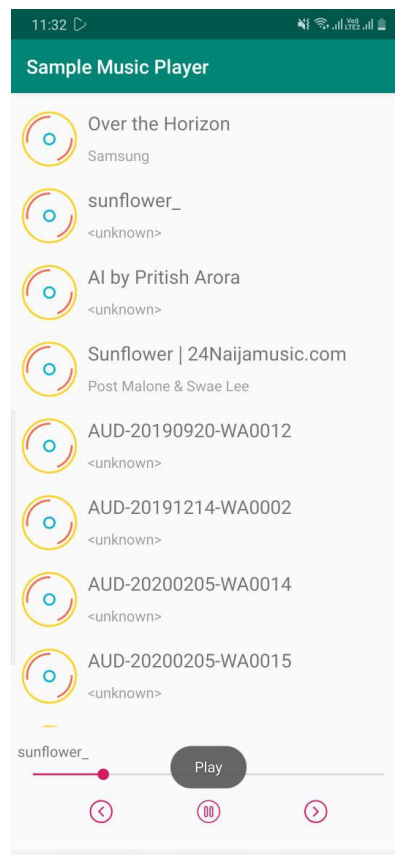
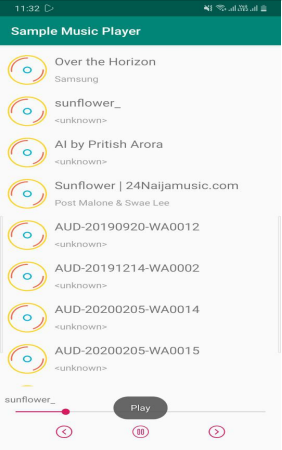
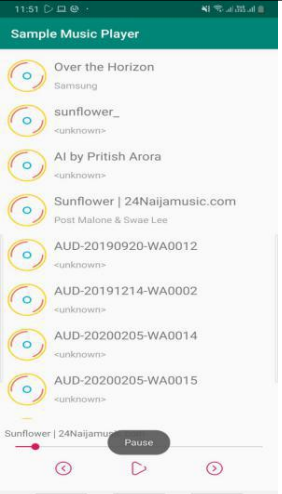
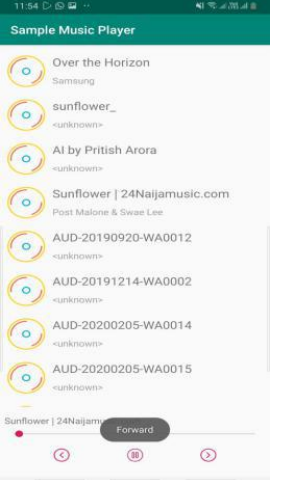
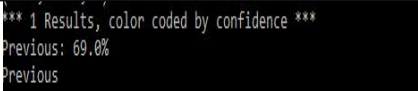


Fig 7.6: Client Application interface

7.7 RESULT –

Table 7.2: Result

S.No.	Gesture	Server	Client
1	Play	<pre> Loaded - names list: data/obj.names, classes = 4 (3516, 4688, 3) *** 1 Results, color coded by confidence *** (3516, 4688, 3) Play: 95.0% Play </pre>	
2	Pause	<pre> Loaded - names list: data/obj.names, classes = 4 (3516, 4688, 3) *** 1 Results, color coded by confidence *** Pause: 75.0% Pause </pre>	
3	Forward	<pre> Loaded - names list: data/obj.names, classes = 4 (3516, 4688, 3) *** 1 Results, color coded by confidence *** Forward: 45.0% Forward </pre>	

4	Previous		
---	----------	--	---

The above table shows the detection made by the server and the reflection of the resultant detection on the client application. The input images are similar to the ones shown in Table 4.1.

In today's world, there are many services available to provide data. Some need physical contact to every application and some without it. Using physical touch (talk, hand gesture etc.). There are applications that are regulated using the current and Intelligent input facility by hand gesture. User of this system can manage remote application without having to use mouse and keyboard. The program includes a flexibility of describing common user-interest movements. Command which makes the application more physically useful for challenged people, as the gesture can be described according to the its viability

CHAPTER 8

APPLICATIONS

8.1.APPLICATIONS

- Optical character recognition.
- Self driving cars.
- Tracking objects in factories.
- Face detection and face recognition.
- Smile detection.
- Medical imaging.
- Ball tracking in sports.
- Surveillance.
- Online images.

8.2. DRAWBACKS OF HAND GESTURE RECOGNITION

- (1) Irrelevant object might overlap with the hand.
- (2) Wrong object extraction appeared if the objects larger than the hand.
- (3) The proposed method is susceptible to errors, especially in shapes like square and circular.
- (4) System limitations restrict the applications such as; gestures are made with the right hand only, the arm must be vertical, the palm is facing the camera, background is plain and uniform.
- (5) Required long time for Learning; several hours for learning 42 characters, and four days to learn ten words.
- (6) Large difference in recognition rate for both registered and unregistered objects.
- (7) Recognition limited to numbers only.
- (8) The system doesn't reflect the dynamic gesture characteristics.
- (9) The edge method less effective since some important features was lost.
- (10) Similar gestures have different orientation histograms. Different gestures have similar orientation histograms. Any objects dominate the image will be represented as orientation histograms.
- (11) The database samples, variation in scale, translation and rotation led to a misclassification.
- (12) The feature vector with highest score evaluated first by the classifier then the system selects that features hypothesis.

CONCLUSION

In this report we conclude can conclude that image processing when combined with artificial intelligence gives rise to a much more useful and revolutionary technology called as computer vision which is essentially making sense out of the raw images. This computer vision concept can be implemented using neural networks and digital signal convolution to achieve tasks like object recognition which can be leveraged to detect certain hand gestures. Using this hand gesture recognition implementation, interface between user and device can be developed where the user can interact with the device using just his/her hands. This approach can be termed as “Hand gesture control”, it shows promise and with more data it might be possible to improve the accuracy which will not only help visually impaired people, but also it has significant impact on the field of robotics, security and forensics.

REFERENCES

1. [Wikipedia.com](https://www.wikipedia.com)
2. W3schools.com
3. Y. Cheoljong, J. Yujeong, B. Jounghoon, H. David and K. Hanseok. "Gesture recognition using depth-based hand tracking for contactless controller application", Consumer Electronics (ICCE), 2012 IEEE International Conference on, pp. 297 -298, 2012
6. H. Kenn, F. Megan and R. Sugar. "A glove-based gesture interface for wearable computing applications", Proceedings of the IFAWC 4th international forum on applied wearable computing 2007, pp. 169-177, 2007.
5. S. Chu and J. Tanaka. "Hand Gesture for Taking Self Portrait", Human-Computer Interaction, Part II, HCII 2011, LNCS 6762, pp. 238–247, 2011.
4. Unseok Lee and Jiro Tanaka. "Finger Identification and Hand Gesture Recognition Techniques for Natural User Interface" University of Tsukuba
17. D. McNeill. "So you think gestures are nonverbal?", Psychological Review, vol92 (3), pp. 350-373, 1985.

Other Refernces:

- <https://pjreddie.com/darknet/yolo/>
- <https://towardsdatascience.com/yolo-v3-object-detectionb>