B.E. PROJECT ON

TOUCHLESS TYPING USING CNNs

SUBMITTED IN PARTIAL FULFILLMENT OF REQUIREMENTS OF AWARD OF

B.E. (COMPUTER ENGINEERING)

DEGREE OF UNIVERSITY OF DELHI

SUBMITTED BY:

Shivam Kantival	364/CO/13
Tanay Jha	376/CO/13
Ujjwal Peshin	380/CO/13

GUIDED BY:
Dr. Sushama Nagpal

COMPUTER ENGINEERING (COE)
NETAJI SUBHAS INSTITUTE OF TECHNOLOGY
UNIVERSITY OF DELHI

2013-17

नेताजी सुभाष प्रौद्योगिकी संस्थान
NETAJI SUBHAS INSTITUTE OF TECHNOLOGY
(Formerly, Delhi Institute of Technology)

Azad Hind Fauj Marg, Sector-3, Dwarka, New Delhi-110 045
Telephone: 2509 9036-42, 2509 9050 Fax: 2509 9022 Website: http://www.nsit.ac.in

CERTIFICATE

The project titled **"Touchless Typing using CNNs"** by **Shivam Kantival (364/CO/13)**, **Tanay Jha (376/CO/13)** and **Ujjwal Peshin (380/CO/13)** is a record of bonafide work carried out by them, in the Division of Computer Engineering. Netaji Subhas Institute of Technology, New Delhi, under the supervision and guidance of Dr. Sushama Nagpal in partial fulfillment of requirement for the award of the degree of Bachelor of Engineering in Computer Engineering, University of Delhi in the academic year 2016-2017.

Dr. Sushama Nagpal
Division of Computer Engineering
Netaji Subhas Institute of Technology
New Delhi

Dated:

CANDIDATE'S DECLARATION

This is to certify that the work which is being hereby presented by us in this project titled **"Touchless Typing using CNNs"** in partial fulfillment of the award of the Bachelor of Engineering submitted at the Department of Computer Engineering, Netaji Subhas Institute of Technology, Delhi, is a genuine account of our work carried out during the period from January 2017 to May 2017 under the guidance of Dr. Sushama Nagpal, Department of Computer Engineering, Netaji Subhas Institute of Technology, Delhi. The matter embodied in the project report to the best of our knowledge has not been submitted for the award of any other degree elsewhere.

Dated:

Shivam Kantival Tanay Jha Ujjwal Peshin

This is to certify that the above declaration by the students is true to the best of my knowledge.

Dr. Sushama Nagpal

ACKNOWLEDGEMENT

No significant achievement can be done by solo performance especially when starting a project from ground up. This B.E. Project, on such a revolutionary idea, has by no means been an exception. It took many very special people to enable it and support it. Here we would like to acknowledge their precious co-operation and express our sincere gratitude to them.

Dr. Sushama Nagpal has been very supportive and involved in yet another student project. It was her support that helped the project to start in its earliest and most vulnerable stages. Her name opened many doors for us and persuaded many people. She was always found with energy and enthusiasm to make sure that we were provided with everything we needed. No amount of words can express thanks to her. She was the one who backed us in providing any assistance we needed during the project work.

We are also thankful to our friends who motivated us at each and every step of this project. Without their interest in our project we could not have gone so far.

And most of all, we would like to thank our wonderful parents who motivated us from day one of the project. You were the lights that led us.

It was a great pleasure and honour to spend our time with all of them and there could not be any better payment for the efforts put into completing this B.E project than their valuable presence. They are all very special to us.

ABSTRACT

Even though a lot of progress has been made to provide input to a device by able-bodied people, their has no significant progress concerning as to how input will be provided by physically disabled people. We present one such technique, touchless typing, which is a novel technique to interact with computers and has been proposed in the below sections. The input characters are taken in a touchless fashion, i.e. using a custom made front facing camera which can filter out visible spectrum and only detects Infrared wavelength. This enables us to use any IR emitter for tracking and providing input as a character. This is fed into a deep convolutional neural network built using LeNet architecture which has the ability to recognize different characters.

The convolutional neural network has the ability to differentiate between different characters as they have been entered into the system via the custom made camera.

It provides an accuracy of 98.4 % on a custom made test set.

TABLE OF CONTENTS

LIST OF FIGURES

CHAPTER - 1

DETAILED PROBLEM STATEMENT

1.1 Introduction

Humans have had tremendous success in the field of technology. Humans have been trying to find more and more ways of making the computer self sufficient because of the major developments in science and education. Humans interact with the computer in many ways, which leading to several ways of human computer interaction (HCI) by creating many intuitive interfaces between them. [1]

Gesture recognition is an advanced field of Human-Computer Interaction, as it allows communication with computers without any mechanical devices.Hand gesture recognition has become an important and vital part of HCI due to its usefulness in Virtual Reality, Teaching and Learning, Sign Language recognition and gaming.[1]

The major difficulties in hand gesture recognition is that, while a human has the ability to discern patterns, for a machine it is merely data. The main job has been to make a machine learn the ability to discern pattern from this data, which is already a trivial task for humans. Hence comes the ability of artificial neural networks. Just like a human has biological neural networks, a machine has the ability to learn patterns using artificial neural networks. This is a fundamental machine learning algorithm which is used to estimate values given some input data, which is passed through a system of interconnected 'neurons'. [1]

Artificial Neural Networks comprises of a set of input values, hidden layer and output layer. There may be varying number of neurons in these layers. The output of a neuron is weighted sum of inputs with a bias. The function of the entire neural network is simply to compute the resultant output of all neurons. [2]

An artificial neural network can be trained with known examples of a problem before it is tested for its inference capability on unknown instances of the problem. ANNs possess the capability to generalize the given set of data, as a result they can predict new outcomes from past trends. A very important feature of these networks is their adaptive nature, where programming is replaced by learning in solving problems. They are robust, fault tolerant and can recall full patterns from partial, incomplete or noisy patterns.[3]

The model proposed takes input from the user using an IR emitter and a webcam which can detect IR and this is used to generate a 28X28 image contains the drawn character. This is passed through a CNN which detects the character drawn. The benefits over present systems include reliability and cost effective nature of the project.

An approach like this can help the disabled people who knew to write, but later on were unable to write on paper by using hand because of some difficulties.

1.2 Motivation

One big motivating factor for the production of this project stem from the need to help the physically disabled people.
About 15% of the world's population lives with some form of disability, of whom 2-4% experience significant difficulties in functioning. The global disability prevalence is higher than previous WHO estimates, which date from the 1970s and suggested a figure of around 10%. This global estimate for disability is on the rise due to population ageing, war and the rapid spread of chronic diseases, as well as improvements in the methodologies used to measure disability.

Also, on larger screens, like TVs or desktops, where there is no inherent touch-control system and the device itself is sometimes out of reach, a gesture-based system makes a lot of sense as it provides an intuitive control system.

If your hands are dirty or if you really hate smudges, touch-free controls are a huge benefit over conventional means of communication with the device.Touch-free screens would also make using touchscreen devices in the winter much easier when you're wearing gloves. For casual gamers who play games like Cut the Rope and Fruit Ninja, this project can be extended to assist their gameplay and make it more natural.

One big future use can be with a 3D display and projections, as that display can be combined with this project to create immersive experiences with futuristic devices.

1.3 Problem Statement

Develop a system capable of taking input from user who uses an IR emitter to generate a character in front of a camera capable of detecting IR feed and produces a prediction of the character. The system should be platform independent.

Generated code should be modular enough so that user can tweak it with minimum file changes. The cohesion should be high for the modules.

INPUT TO THE SYSTEM

A single input stream by the user using an IR emitter which is used to make the character in front of the camera stream.

OUTPUT OF THE SYSTEM

A character recognized by the provided system.

1.4 Definitions

Word	Meaning
IR	Infrared radiation (IR), sometimes referred to simply as infrared, is a region of the electromagnetic radiation spectrum where wavelengths range from about 700 nanometers (nm) to 1 millimeter (mm). Infrared waves are longer than those of visible light, but shorter than those of radio waves. The frequencies of IR are higher than those of microwaves, but lower than those of visible light, ranging from about 300 GHz to 400 THz.
HCI	HCI (human-computer interaction) is the study of how people interact with computers and to what extent computers are or are not developed for successful interaction with human beings. Computer system developers might argue that computers are extremely complex products to design and make and that the demand for the services that computers can provide has always out driven the demand for ease-of-use.
ANN	Artificial Neurons are similar to their biological counterparts. The input connections of the artificial neurons are summed up to determine the strength of their output, which is the result of the sum being fed into an activation function. The resultant of this function is then passed as the input to other neurons through more connections, each of which are weighted and these weights determine the behavior of the network.
CNN	In machine learning, a convolutional neural network (CNN, or ConvNet) is a type of feed-forward artificial neural network in which the connectivity pattern between its neurons is inspired by the organization of the animal visual cortex. Individual cortical neurons respond to stimuli in a restricted region of space known as the *receptive field*.
HSV	A way to characterize a color is in terms of the HSV model.The *hue* (H) of a color refers to which pure color it resembles. All tints, tones and shades of red have the same hue.The *saturation* (S) of a color describes how white the color is. A pure red is with a saturation of 1; tints of red have saturations less than 1; and white has a saturation of 0.The *value* (V) of a color, also called its *lightness*, describes how dark the color is. A value of 0 is black, with increasing lightness moving away from black.

CHAPTER-2
LITERATURE SURVEY

2.1 Tracking

2.1.1 OpenCV

OpenCV (*Open Source Computer Vision*) is a library of programming functions mainly aimed at real-time computer vision. Originally developed by Intel, it was later supported by Willow Garage and is now maintained by Itseez.[6]

Officially launched in 1999, the OpenCV project was initially an Intel Research initiative to advance CPU-intensive applications, part of a series of projects including real-time ray tracing and 3D display walls. The main contributors to the project included a number of optimization experts in Intel Russia, as well as Intel's Performance Library Team. In the early days of OpenCV, the goals of the project were described as:

- Advance vision research by providing not only open but also optimized code for basic vision infrastructure. No more reinventing the wheel.
- Disseminate vision knowledge by providing a common infrastructure that developers could build on, so that code would be more readily readable and transferable.
- Advance vision-based commercial applications by making portable, performance-optimized code available for free—with a license that did not require code to be open or free itself.

The first alpha version of OpenCV was released to the public at the IEEE Conference on Computer Vision and Pattern Recognition in 2000, and five betas were released between 2001 and 2005. In August 2012, support for OpenCV was taken over by a non-profit foundation OpenCV.org, which maintains a developer and user site.[6]

OpenCV's application areas include:

- 2D and 3D feature toolkits
- Facial recognition system
- Gesture recognition
- Human–computer interaction (HCI)
- Object identification
- Motion tracking

To support some of the above areas, OpenCV includes a statistical machine learning library that contains:

- Boosting
- Decision tree learning
- Gradient boosting trees
- k-nearest neighbor algorithm
- k-nearest neighbor algorithm
- Artificial neural networks

OpenCV is written in C++ and its primary interface is in C++, but it still retains a less comprehensive though extensive older C interface. There are bindings in Python, Java and MATLAB/OCTAVE. The API for these interfaces can be found in the online documentation.[6]

OpenCV runs on a variety of platforms. Desktop: Windows, Linux, macOS, FreeBSD, NetBSD, OpenBSD; Mobile: Android, iOS, Maemo, BlackBerry 10. The user can get official releases from SourceForge or take the latest sources from GitHub.[6]

OpenCV

2.1.2 Hardware Requirements

The hardware required to run the project consists of the following:

- A CPU with preferably two cores.
- A webcam with IR filter removed.

- An IR emitting device.

A **webcam** [15] is a video camera that feeds or streams its image in real time to or through a computer to a computer network. When "captured" by the computer, the video stream may be saved, viewed or sent on to other networks via systems such as the internet, and emailed as an attachment. When sent to a remote location, the video stream may be saved, viewed or on sent there. Unlike an IP camera (which connects using Ethernet or Wi-Fi), a webcam is generally connected by a USB cable, or similar cable, or built into computer hardware, such as laptops.

An **infrared emitter, or IR emitter** [16], is a source of light energy in the infrared spectrum. It is a light emitting diode (LED) that is used in order to transmit infrared signals from a remote control. In general, the more they are in quantity and the better the emitters are, the stronger and wider the resulting signal is. A remote with strong emitters can often be used without directly pointing at the desired device. Infrared emitters are also partly responsible for limits on the range of frequencies that can be controlled. An IR emitter generates infrared light that transmits information and commands from one device to another. Typically one device receives the signal then passes the infrared (IR) signal through the emitter to another device.

2.1.3 Previous Work

Object detection is a computer technology related to computer vision and image processing that deals with detecting instances of semantic objects of a certain class (such as humans, buildings, or cars) in digital images and videos. Well-researched domains of object detection include face detection and pedestrian detection. Object detection has applications in many areas of computer vision, including image retrieval and video surveillance.

Every object class has its own special features that helps in classifying the class – for example all circles are round. Object class detection uses these special features. For example, when looking for circles, objects that are at a particular distance from a point

(i.e. the center) are sought. Similarly, when looking for squares, objects that are perpendicular at corners and have equal side lengths are needed.
The previous work done in this field consists of creating the system by tracking a colored marker to draw the object. [1][2]

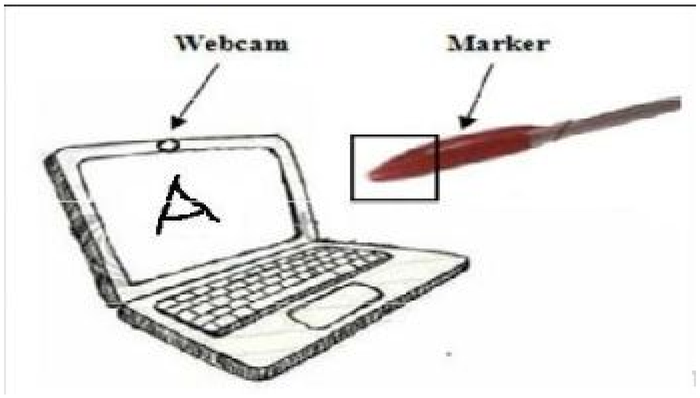
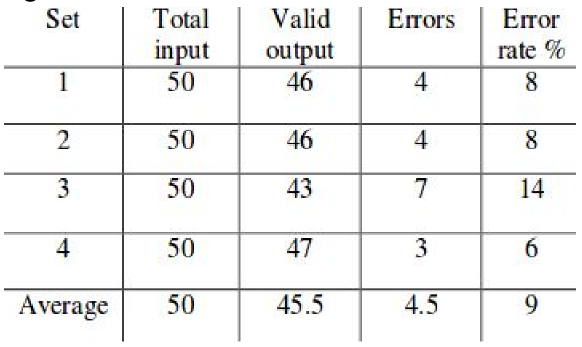


Figure 1 : Webcam Tracking

The shortcomings of this approach are:

- The surrounding environment should not have the color which is being tracked by the code originally.
- The tracking is dependent on the lighting conditions of the surrounding environment. If the environment is not properly lighted then the color may not be detected.
- The object should not be placed very far, as it will hamper the detection process.
- The tracking is heavily dependent on the quality of the camera.

Moreover the accuracy obtained by the already implemented system are not very promising and have been given below:



Set	Total input	Valid output	Errors	Error rate %
1	50	46	4	8
2	50	46	4	8
3	50	43	7	14
4	50	47	3	6
Average	50	45.5	4.5	9

Figure 2 : Accuracy obtained in above Model

Hence it is quite evident that the work done in this field is not up to the mark and there is a lot of scope of improvement in it. We have employed better tracking techniques and used better machine learning models in order to improve the accuracy as well as giving a better user experience.

2.2 Recognition

2.2.1 Machine Learning [10]

Machine learning is the subfield of computer science that, according to Arthur Samuel in 1959, gives "computers the ability to learn without being explicitly programmed." Evolved from the study of pattern recognition and computational learning theory in artificial intelligence, machine learning explores the study and construction of algorithms that can learn from and make predictions on data – such algorithms overcome following strictly static program instructions by making data-driven predictions or decisions, through building a model from sample inputs.
Within the field of data analytics, machine learning is a method used to devise complex models and algorithms that lend themselves to prediction; in commercial use, this is known as predictive analytics.
Machine learning tasks are typically classified into three broad categories, depending on the nature of the learning "signal" or "feedback" available to a learning system. These are:

- **Supervised learning:** The computer is presented with example inputs and their desired outputs, given by a "teacher", and the goal is to learn a general rule that maps inputs to outputs.
- **Unsupervised learning:** No labels are given to the learning algorithm, leaving it on its own to find structure in its input. Unsupervised learning can be a goal in itself (discovering hidden patterns in data) or a means towards an end (feature learning).
- **Reinforcement learning:** A computer program interacts with a dynamic environment in which it must perform a certain goal (such as driving a vehicle or playing a game against an opponent). The program is provided feedback in terms of rewards and punishments as it navigates its problem space.

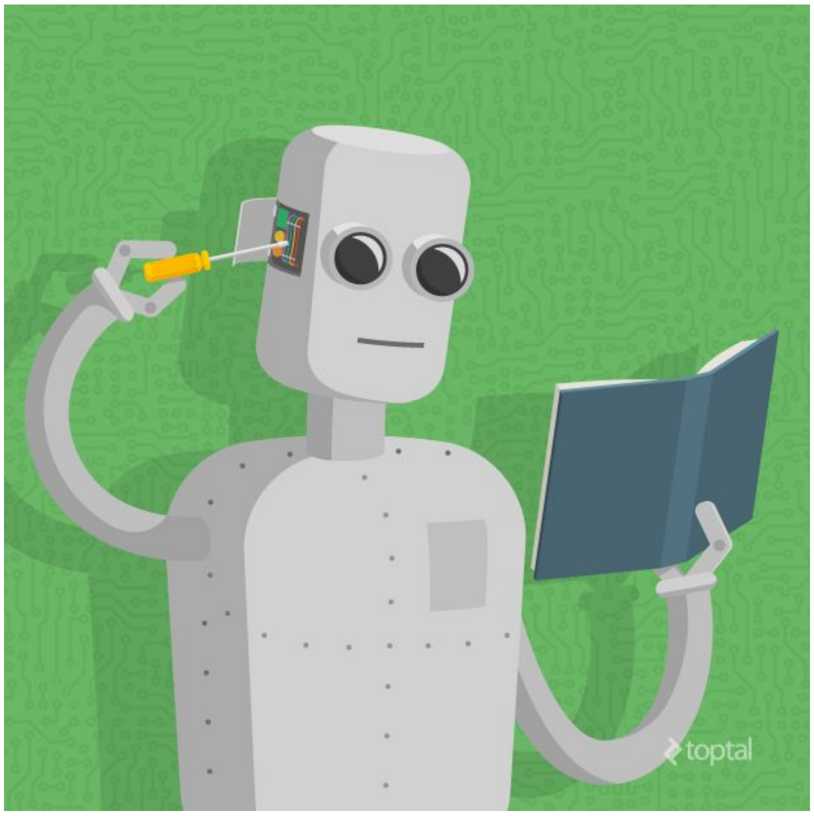
Another categorization of machine learning tasks arises when one considers the desired *output* of a machine-learned system:

- In **classification**, inputs are divided into two or more classes, and the learner must produce a model that assigns unseen inputs to one or more (multi-label classification) of these classes.
- In **regression**, also a supervised problem, the outputs are continuous rather than discrete.

- In **clustering**, a set of inputs is to be divided into groups. Unlike in classification, the groups are not known beforehand, making this typically an unsupervised task.
- **Dimensionality reduction** simplifies inputs by mapping them into a lower-dimensional space

There are numerous areas where machine learning is being used:

- Adaptive websites
- Bioinformatics
- Brain-machine interfaces
- Classifying DNA sequences
- Computer vision, including object recognition
- Detecting credit card fraud
- Game playing
- Natural language processing
- Search engines
- Software engineering



2.2.2 Deep Learning

Deep learning is the study of artificial neural networks and related machine learning algorithm that contain more than one hidden layer. These deep nets:

- use a cascade of many layers of nonlinear processing units for feature extraction and transformation. Each successive layer uses the output from the previous layer as input. The algorithms may be supervised or unsupervised and applications include pattern analysis (unsupervised) and classification (supervised).
- are based on the (unsupervised) learning of multiple levels of features or representations of the data. Higher level features are derived from lower level features to form a hierarchical representation.
- are part of the broader machine learning field of learning representations of data.
- learn multiple levels of representations that correspond to different levels of abstraction; the levels form a hierarchy of concepts.

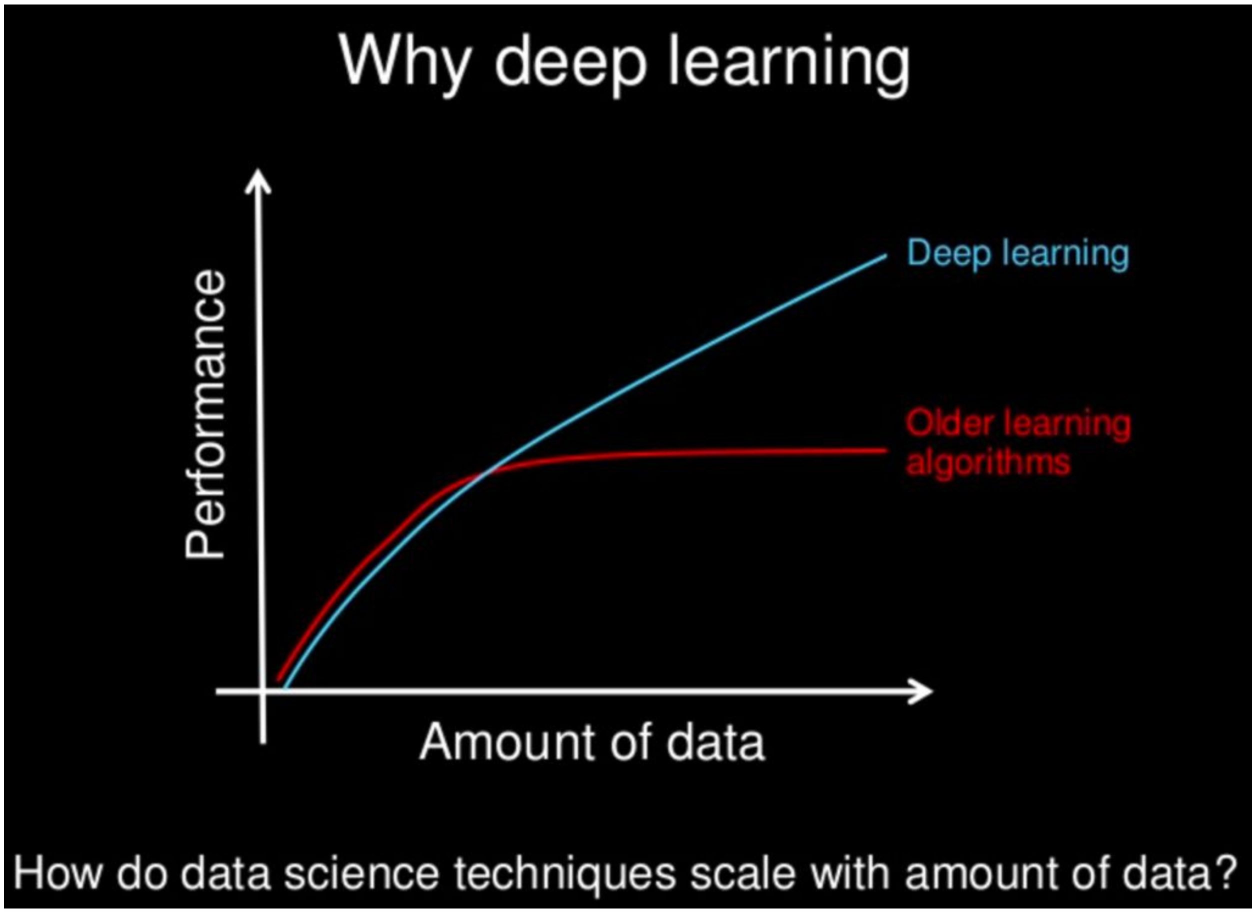
In a simple case, there might be two sets of neurons: one set that receives an input signal and one that sends an output signal. When the input layer receives an input it passes on a modified version of the input to the next layer. In a deep network, there are many layers between the input and the output (and the layers are not made of neurons but it can help to think of it that way), allowing the algorithm to use multiple processing layers, composed of multiple linear and nonlinear transformations.

Various deep learning architectures such as deep neural networks, convolutional deep neural networks, deep belief networks and recurrent neural networks have been applied to fields like computer vision, automatic speech recognition, natural language processing, audio recognition and bioinformatics where they have been shown to produce state-of-the-art results on various tasks.

Deep learning algorithms are based on distributed representations. The underlying assumption behind distributed representations is that observed data are generated by the interactions of factors organized in layers. Deep learning adds the assumption that these layers of factors correspond to levels of abstraction or composition. Varying numbers of layers and layer sizes can be used to provide different amounts of abstraction.

Deep learning is often presented as a step towards realising strong AI and thus many organizations have become interested in its use for particular applications. In December 2013, Facebook hired Yann LeCun to head its new artificial intelligence (AI) lab that was to have operations in California, London, and New York. The AI lab will develop deep learning techniques to help Facebook do tasks such as automatically tagging uploaded pictures with the names of the people in them.

A main criticism of deep learning concerns the lack of theory surrounding many of the methods. Learning in the most common deep architectures is implemented using gradient descent; while gradient descent has been understood for a while now, the theory surrounding other algorithms, such as contrastive divergence is less clear (i.e., Does it converge? If so, how fast? What is it approximating?) Deep learning methods are often looked at as a black box, with most confirmations done empirically, rather than theoretically.



2.2.3 Neural Networks [11]

Neural networks (NNs) are a computational model used in machine learning, computer science and other research disciplines, which is based on a large collection of connected simple units called artificial neurons, loosely analogous to axons in a biological brain. Connections between neurons carry an activation signal of varying strength.If the combined incoming signals are strong enough, the neuron becomes activated and the signal travels to other neurons connected to it. Such systems can be trained from examples, rather than explicitly programmed, and excel in areas where the solution or feature detection is difficult to express in a traditional computer program. Like other machine learning methods, neural networks have been used to solve a wide variety of tasks, like computer vision and speech recognition, that are difficult to solve using ordinary rule-based programming.

Typically, neurons are connected in layers, and signals travel from the first (input), to the last (output) layer. Modern neural network projects typically have a few thousand to a few million neural units and millions of connections; their computing power is similar to a worm brain, several orders of magnitude simpler than a human brain. The goal of the neural network is to solve problems in the same way that a human would, although several neural network categories are more abstract.

A key advance that came later was the backpropagation algorithm which effectively solved the exclusive-or problem, and more generally the problem of quickly training multi-layer neural networks. Neural networks, as used in artificial intelligence, have traditionally been viewed as simplified models of neural processing in the brain, even though the relation between this model and the biological architecture of the brain is debated; it's not clear to what degree artificial neural networks mirror brain function.

The tasks artificial neural networks are applied to tend to fall within the following broad categories:

- Function approximation, or regression analysis, including time series prediction.
- Classification, including pattern and sequence recognition.
- Data processing, including filtering, clustering and compression.
- Robotics, including directing manipulators, prosthesis.
- Control, including computer numerical control.

A common criticism of neural networks, particularly in robotics, is that they require a large diversity of training for real-world operation. This is not surprising, since any learning

machine needs sufficient representative examples in order to capture the underlying structure that allows it to generalize to new cases.
To implement large and effective software neural networks, considerable processing and storage resources need to be committed. While the brain has hardware tailored to the task of processing signals through a graph of neurons, simulating even a most simplified form on von Neumann architecture may compel a neural network designer to fill many millions of database rows for its connections – which can consume vast amounts of computer memory and hard disk space.

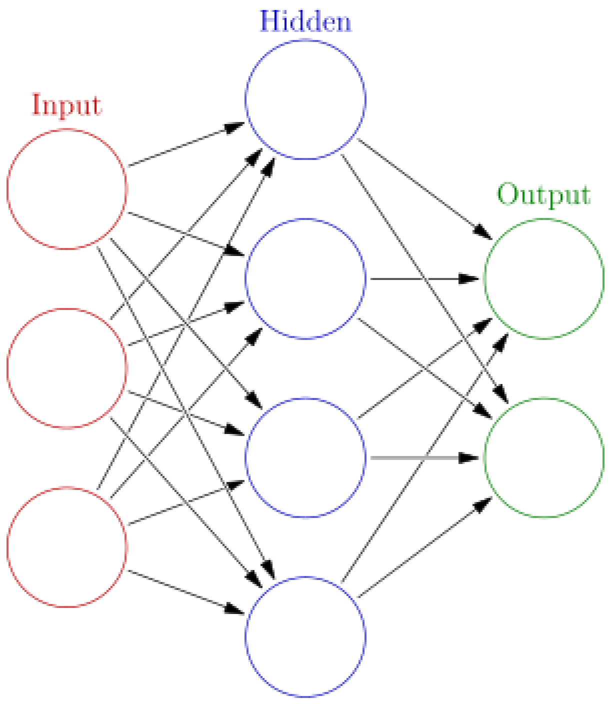


Figure 3 : Neural Network Structure

2.2.4 Convolutional Neural Network [12]

Convolutional Neural Network (CNN) are similar to normal Neural Network apart from the fact that they make explicit assumption that their inputs will always be images. They then make the forward function more efficient to implement and vastly reduce the amount of parameters in the network. CNNs model animal visual perception, hence they can be applied to visual recognition task.

CNNs consist of multiple layers of receptive fields. These are small neuron collections which process portions of input image. The outputs of these collections are then tiled so that their input regions overlap, to obtain a higher-resolution representation of the original image; this is repeated for every such layer.
The CNNs can have a variety of architecture. One of the commonly used architecture is [INPUT-CONV-RELU-POOL-FC]. In more detail:

- **INPUT** holds the raw pixel value of the image.
- **CONV** layer will compute the outputs of the neurons that are connected to the local regions in the input.
- **RELU** will apply an element wise activation function, such as max(0, x) thresholding at zero.
- **POOL** layer will perform a down sampling operation along the spatial dimensions.

- **FC** (i.e. fully connected) layer will compute the class scores.

In this way, CNN's transform the original image layer by layer from the original pixel values to the final class scores.

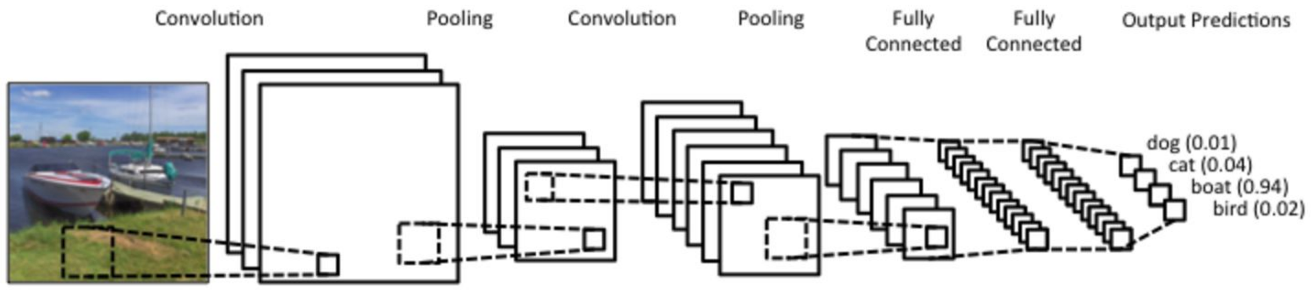


Figure 4 : Convolutional Neural Network in Working

2.2.5 Transfer Learning [13]

Transfer learning focuses on storing knowledge gained while solving one problem and applying it to a different but related problem. For example, knowledge gained while trying to recognize cars could apply when trying to recognize trucks.

In practice, very few people train an entire CNN from scratch, because it is relatively rare to have a dataset of sufficient size. Instead it is common to pre-train a CNN on a very large dataset, and then use the CNN either as an initialization or a fixed feature extractor for the task of interest.

- **CNN as fixed feature extractor:** Here we take a CNN pre trained on another dataset and remove the last fully connected layer, then treat the rest of the CNN as fixed feature extractor for the new dataset.
- **Fine tuning the CNN:** The second strategy is to not only replace and retrain the classifier on top of the CNN on the new dataset, but to also fine-tune the weights of the pre trained network by continuing the back-propagation.
- **Pre trained models:** Since training on modern CNN takes around 2-3 weeks, people release their final CNN checkpoints for the benefit of others.

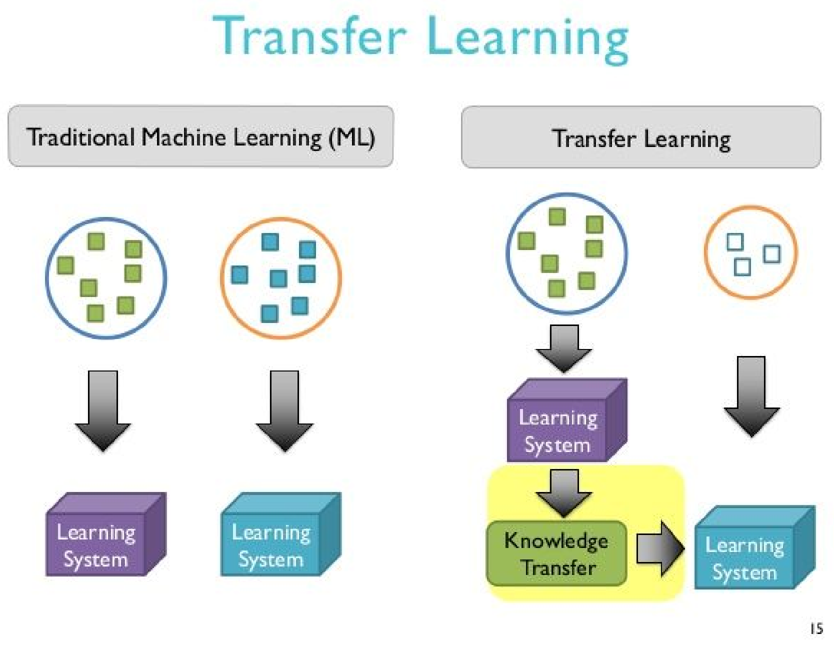


Figure 5 : Traditional Machine Learning vs Transfer Learning

2.2.6 LeNet vs Inception Architecture

LeNet Architecture [14][18]

It is the year 1994, and this is one of the very first convolutional neural networks, and what propelled the field of Deep Learning. This pioneering work by Yann LeCun was named LeNet5 after many previous successful iterations since they year 1988!

The LeNet5 architecture was fundamental, in particular the insight that image features are distributed across the entire image, and convolutions with learnable parameters are an effective way to extract similar features at multiple location with few parameters.

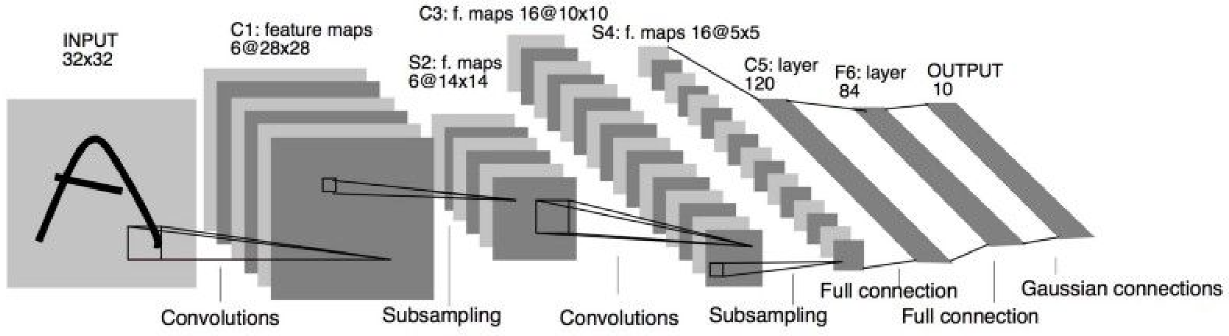


Figure 6 : LeNet Architecture

LeNet5 features can be summarized as:

- convolutional neural network use sequence of 3 layers: convolution, pooling, non-linearity –> This may be the key feature of Deep Learning for images since this paper!
- use convolution to extract spatial features
- subsample using spatial average of maps
- non-linearity in the form of tanh or sigmoids
- multi-layer neural network (MLP) as final classifier
- sparse connection matrix between layers to avoid large computational cost

Overall this network was the origin of much of the recent architectures, and a true inspiration for many people in the field.

GoogLeNet and Inception [14][20]

Christian Szegedy from Google begun a quest aimed at reducing the computational burden of deep neural networks, and devised the GoogLeNet the first Inception architecture. Christian thought a lot about ways to reduce the computational burden of deep neural nets while obtaining state-of-art performance. He and his team came up with the Inception module:

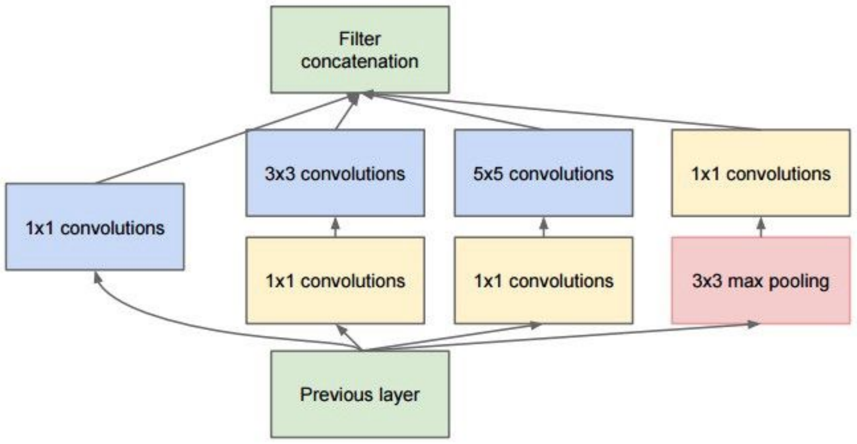


Figure 7 : Inception Model

which at a first glance is basically the parallel combination of 1×1, 3×3, and 5×5 convolutional filters. But the great insight of the inception module was the use of 1×1 convolutional blocks (NiN) to reduce the number of features before the expensive parallel blocks. This is commonly referred as “bottleneck”.
GoogLeNet used a stem without inception modules as initial layers, and an average pooling plus softmax classifier similar to NiN. This classifier is also extremely low number of operations, compared to the ones of AlexNet and VGG. This also contributed to a very efficient network design.

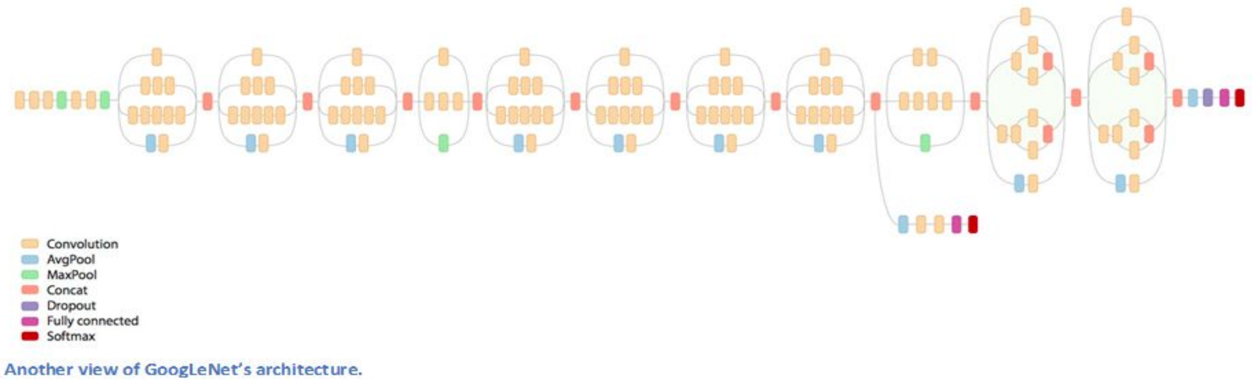


Figure 8 : GoogLeNet Architecture

2.2.7 Tensorflow [21]

TensorFlow is an open source software library for machine learning across a range of tasks, and developed by Google to meet their needs for systems capable of building and training neural networks to detect and decipher patterns and correlations, analogous to the learning and reasoning which humans use. It is currently used for both research and production at Google products, often replacing the role of its closed-source predecessor, DistBelief. TensorFlow was originally developed by the Google Brain team for internal Google use before being released under the Apache 2.0 open source license on November 9, 2015.

TensorFlow is Google Brain's second generation machine learning system, released as open source software on November 9, 2015. While the reference implementation runs on single devices, TensorFlow can run on multiple CPUs and GPUs (with optional CUDA extensions for general-purpose computing on graphics processing units).

In May 2016 Google announced its tensor processing unit (TPU), a custom ASIC built specifically for machine learning and tailored for TensorFlow. The TPU is a programmable AI accelerator designed to provide high throughput of low-precision arithmetic (e.g., 8-bit), and oriented toward using or running models rather than training them. Google announced they had been running TPUs inside their data centers for more than a year, and have found them to deliver an order of magnitude better-optimized performance per watt for machine learning.

Among the applications for which TensorFlow is the foundation, are automated image captioning software, such as DeepDream. Google officially implemented RankBrain on 26 October 2015, backed by TensorFlow. RankBrain now handles a substantial number of search queries, replacing and supplementing traditional static algorithm based search results.

CHAPTER - 3
DETAILED PROBLEM STATEMENT

3.1 Introduction

This document is a Software Requirement Specification (SRS) for the Touchless Typing using CNNs. SRS will be used for the extensions. The document is prepared as per the standard IEEE standard [9] for Software Requirement Specification.

3.1.1 Purpose

The purpose of this document is to present a detailed description of the Touchless Typing using CNNs. It will explain the purpose and features of the system, the interfaces of the system, what the system will do, the constraints under which it must operate and how the system will react to external stimuli. Through this document, the workload needed for development, validation and verification will ease. To be specific, this document is going to describe functionality, external interfaces, performance, attributes and the design constraints of the system which is going to be developed.

3.1.2 Scope

This software system is a touchless interface for physically disabled people or people who have their hands dirty and don’t want to spoil the screen of their device. This system can be very helpful to enable war veterans and disabled people to be a part of the community and not feel unwanted. It removes the dependency of having a physical access of the device, which has installed this service, at all times.

3.1.3 Technologies Used

Programming Language:

Python: Python is a general-purpose interpreted, interactive, object-oriented, and high-level programming language. Its high-level built in data structures, combined with dynamic typing and dynamic binding, make it very attractive for Rapid Application Development, as well as for use as a scripting or glue language to connect existing components together. Python's simple, easy to learn syntax emphasizes readability and therefore reduces the cost of program maintenance. Python supports modules and packages, which encourages program modularity and code reuse. The Python interpreter and the extensive standard library are available in source or binary form without charge for all major platforms, and can be freely distributed.

Libraries: Since the product will require a lot of libraries based on the requirements, developers are open to choose any set of libraries as found fit for high cohesion and modularity.

3.1.4 Overview

The next chapter, the Overall Description section, of this document gives an overview of the functionality of the product. It describes the informal requirements and is used to establish a context for the technical requirements specification in the next chapter.

The third chapter, Requirements Specification section, of this document is written primarily for the developers and describes in technical terms the details of the functionality of the product.

Both sections of the document describe the same software product in its entirety, but are intended for different audiences and thus use different language.

3.2 Overall Description

This section will give the overall product description of the specific requirements of the product, 'Touchless Typing Using CNNs' which will be developed.
Although we do not explain every requirement in detail, this will describe the factors which will influence the final product.

3.2.1 Product Perspective

This is a software product which intends to remove human interaction with a physical mechanical device, which will lead to a further enhancement of Human - Computer Interaction. The product, while having similar systems which have been developed, aims to provide a level of reliability and cost effective measure which had not been provided earlier. The product will be currently deployed to computers and has a later scope of being expanded to mobile systems also.
The main advantage of this product remains in the fact that it provides a higher degree of reliability than the current existing systems and is also cost effective in nature. Hence, there is no sacrifice of cost effectivity for reliability.

3.2.2 Product Functions

The product aims to provide functionality which have been explained in the introduction above. The functionality of the product are explained as below:
The main tracking function takes input from the user from a camera feed and provides result of some computations in the CLI.

The function uses several modules to implement different stages of tracking and recognition, which can be defined as the following:

1. **Tracking:** takes input from the user from the camera feed provided to the user and copies the tracked coordinates to a file.
2. **Image Preprocessing:** takes coordinates from file and plots an image which is cropped to 28X28 image.

3. **Character recognition:** uses pre trained model to recognize characters from input 28X28 image.
4. **Interrupt Generation:** generates keyboard interrupt in the system according to the recognized character.

3.2.3 User Characteristics

1. User must have a basic knowledge of the alphabet and numeric symbols.
2. User is accustomed to basic computer usage.
3. User must be able to hold the remote.

3.2.4 Assumptions

1. User has bought a camera capable of IR.
2. User has an IR emitter.
3. User has python installed in command line and has all dependencies installed, which will be provided in the documentation.

<u>3.3 Specific Requirements</u>

With this section and later, we will describe the requirements of the product in detail. Basically, we will categorize requirements in three which are namely external interface requirements, functional requirements and nonfunctional requirements. Except non-functional requirements, requirements of the product will be detailed under this section with brief information.

3.3.1 External Interface Requirements

1. **User Interface Requirements:** The interface provided to the user must be understandable to a basic user using the system. A README file is to be provided for a proper explanation of the installation of the system and how to run the system.
2. **Hardware Interface Requirements:** The hardware required to run the project consists of the following:

- A CPU with preferably two cores.
- A webcam with IR filter removed.
- An IR emitting device.

The first requirement is not very demanding and is commonly available in most of the places nowadays.

[7]To fulfill the second requirement, a few changes need to be done on the normal webcam:

- unscrew the webcam casing.
- look for the ir filter (tiny colour film like thing).
- with help of cutter or scissors carefully get off the IR filter.
- replace the ir filter with a clean unused negative film.
- screw the setup,adjust the focus back.
- Finally, to make this an IR-only camera, the visible light needs to be filter-out. To do that, a filter is made for the front of the lens out of several layers of fully-exposed 35mm color negative film.

The third requirement is that of an IR emitting device. A simple TV remote is an IR emitter. Hence we can modify the TV remote to make it work as an IR emitter.

3. **Communication Interface Requirements:** Communication has no involvement with the project. Hence there are no communication interface requirements.

3.3.2. Functional Requirements

Main function: Takes input from user as stream of video feed and provides output as prediction of character produced.
Infrared Capable Camera and IR emitter : Takes input from user.
Frame Processing : Captures contour and geometric center from the video feed and feeds it into a file.
Data Plotting : Plots captured coordinate points to form an image.
Image Preprocessing : crops and dilates the image to a 28X28 binary image
CNN model : model used for character recognition

3.3.3 Performance Requirements

1. Prediction should be available within 1 second.
2. Webcam quality should be good.

3.3.4 Design Constraints

1. Hardware Constraints - The computer which has installed the system should have 2 cores preferably. The system should have a webcam which has the ability to detect IR sensor. The user should also have an IR emitter.
2. The language used should have good implementation of machine learning libraries.

3.3.5 Non Functional Requirements

1. Safety : Direct Exposure of IR to eyes may lead to loss of eyesight.
2. Usability : The system shall be easy to use and understand. User will only need to give inputs about the application. All other functions will be automated and internal to generation process.

3. Maintainability – Component driven development and modular structure shall ensure maintainable code.
4. Portability - The system should be compatible with other systems and the end-user part should be fully portable and any system should be able to use the features of the application, including any hardware platform that is available or will be available in the future.

3.3.6 Other Requirements

Schedule and Budgets

The model of this project is to be proposed by 2nd June 2017. Due to lack of funding, a large scale system could not be implemented.

3.4 Change Management Process

The SRS is developed in such a manner that any desired changes can be introduced by the designing party in the near future according to the suitability.

3.5 Design Document

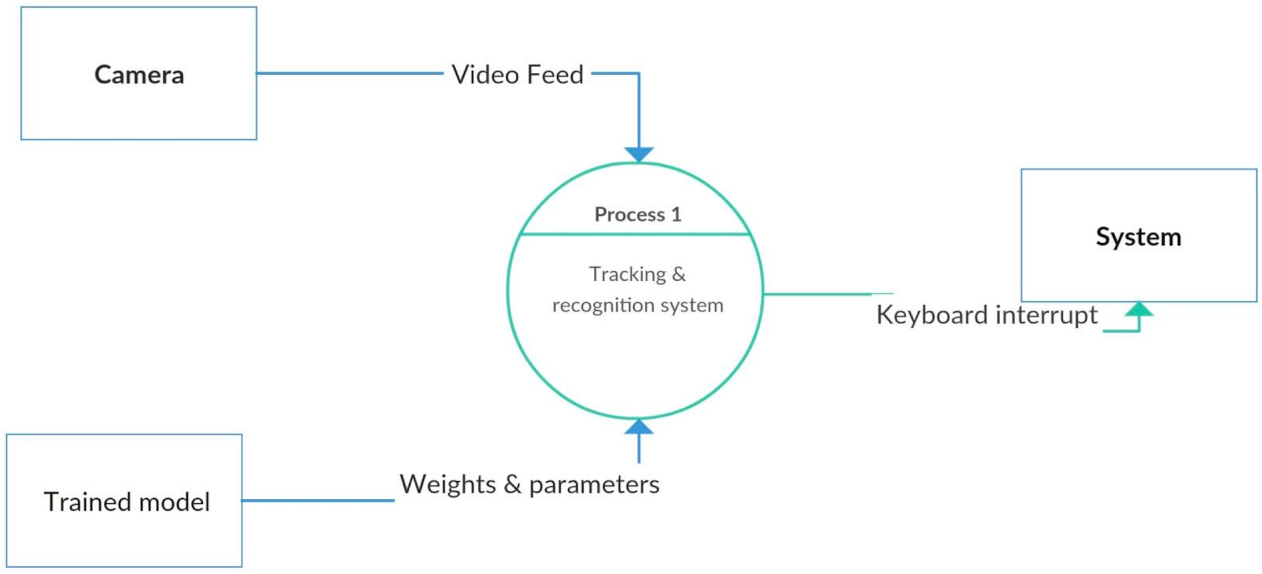


Figure 9 : Context Diagram

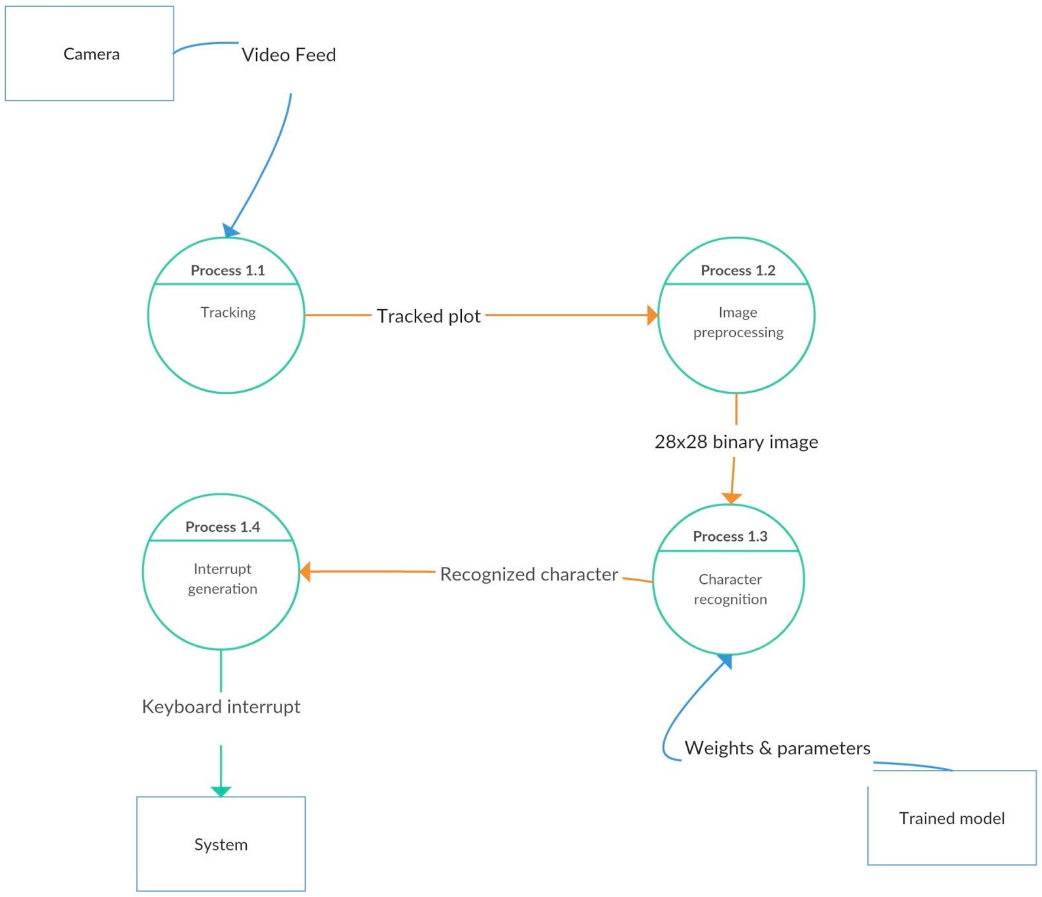


Figure 10 : Data Flow Level 1 in Touchless Typing using CNNs

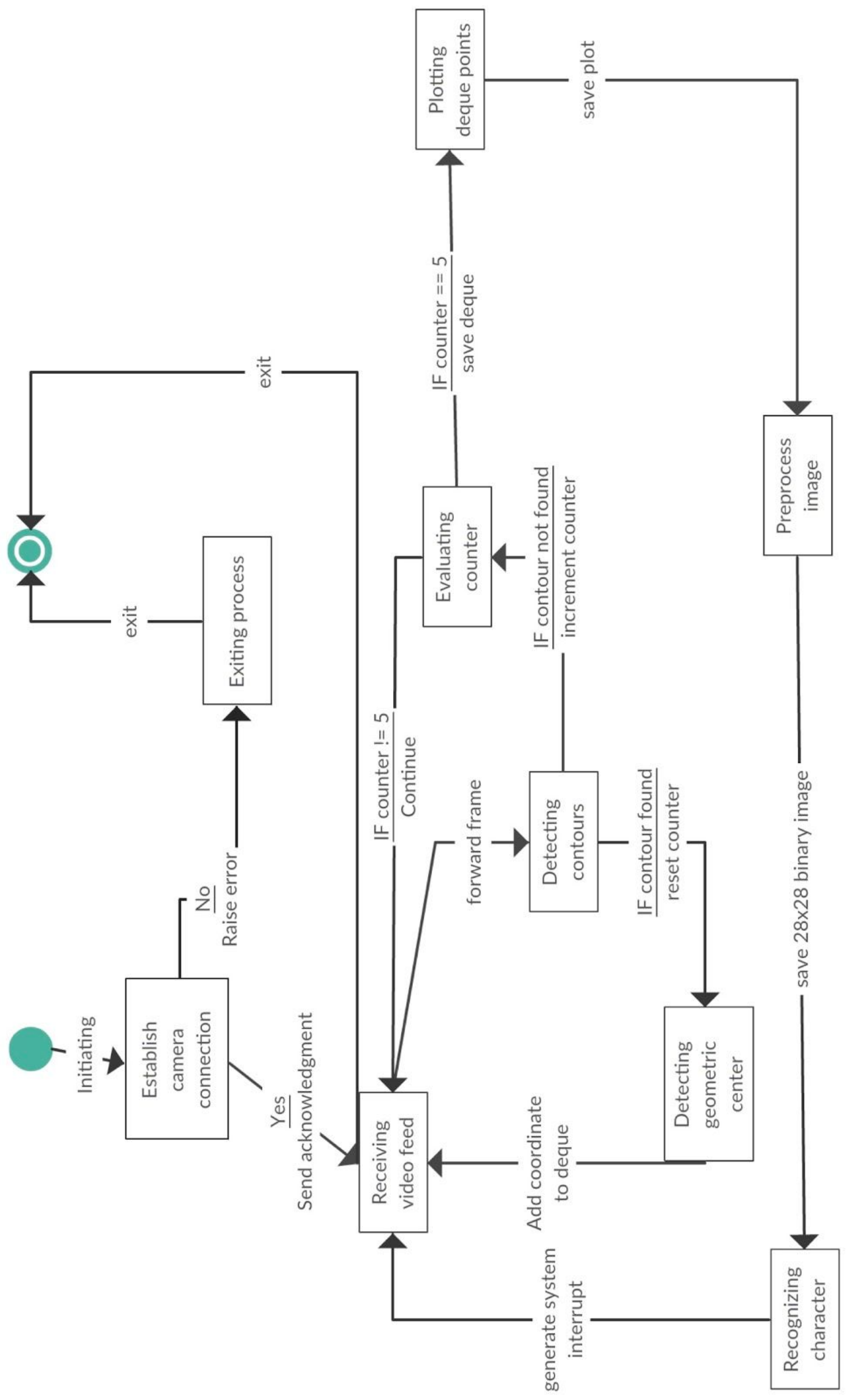


Figure 11 : State Transition Diagram in Touchless Typing using CNNs

CHAPTER - 4

MODEL PROPOSED

CHAPTER-4
Proposed model

In the final model user draws the character to be recognized aerially using an infrared emitter. The movement of this infrared marker is captured by a camera capable of viewing infrared spectrum light and neglects the visible spectrum waveforms. The video feed from this camera is further processed to track and plot the movement of the marker. The plot thus generated is normalised and fed to a pre-trained CNN network and the recognised character is used as per the requirement.

4.1 Block diagram

The system has the following modules working in tandem to generate the required results.

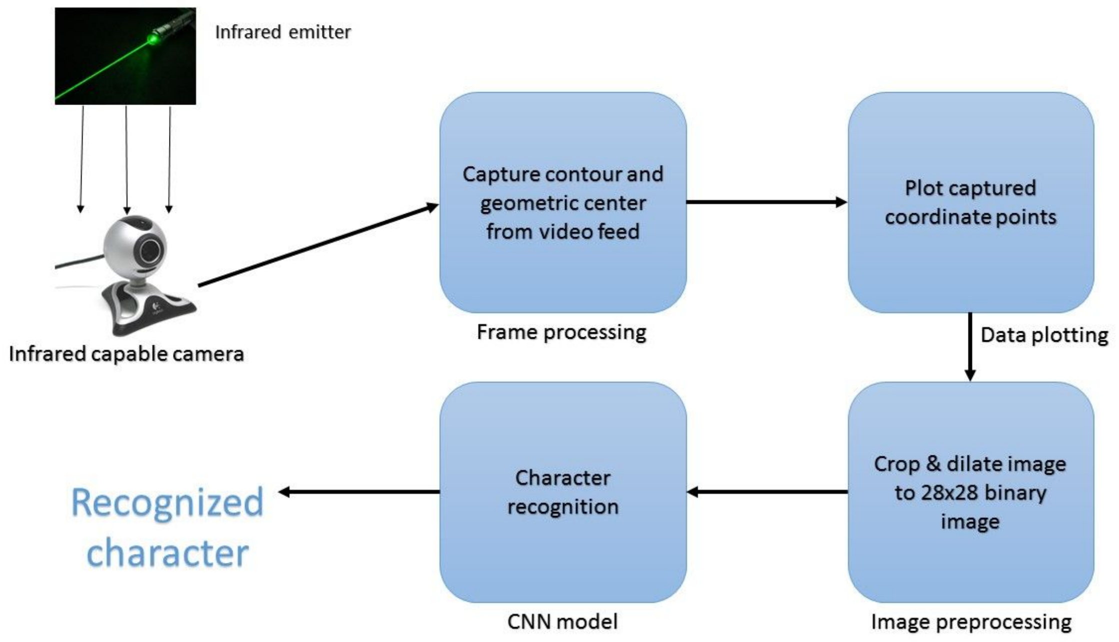


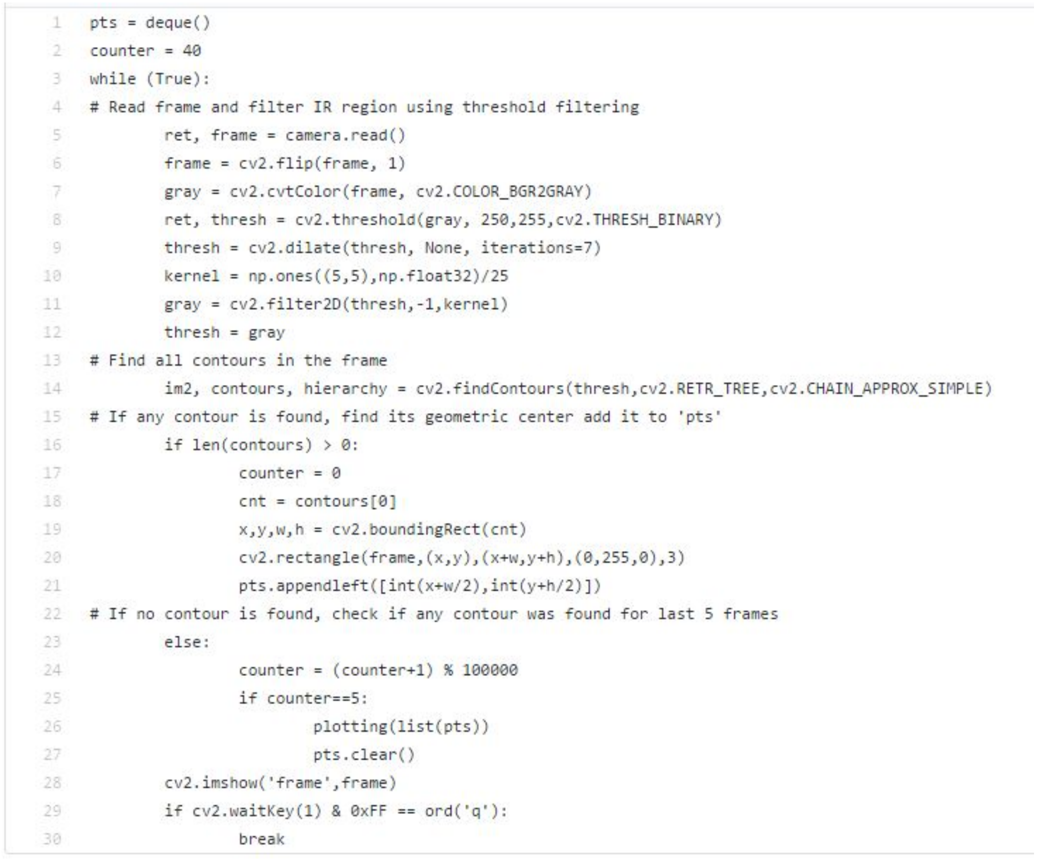
Figure 12 : Block Diagram

4.2 System Modules

4.2.1 IR tracking

This module receives the video frames from the camera, detects the region in which IR is present and draws a contour around it. It then finds the minimum enclosing rectangle around this contour and finds the geometric center of the contour.

The centers of contours found are added to a deque. Once this modules detects that no contours have been found for 5 simultaneous frames, it automatically passes this deque to module described in 4.2.2 for plotting. All the functions used here for image processing are provided in OPENCV[6].



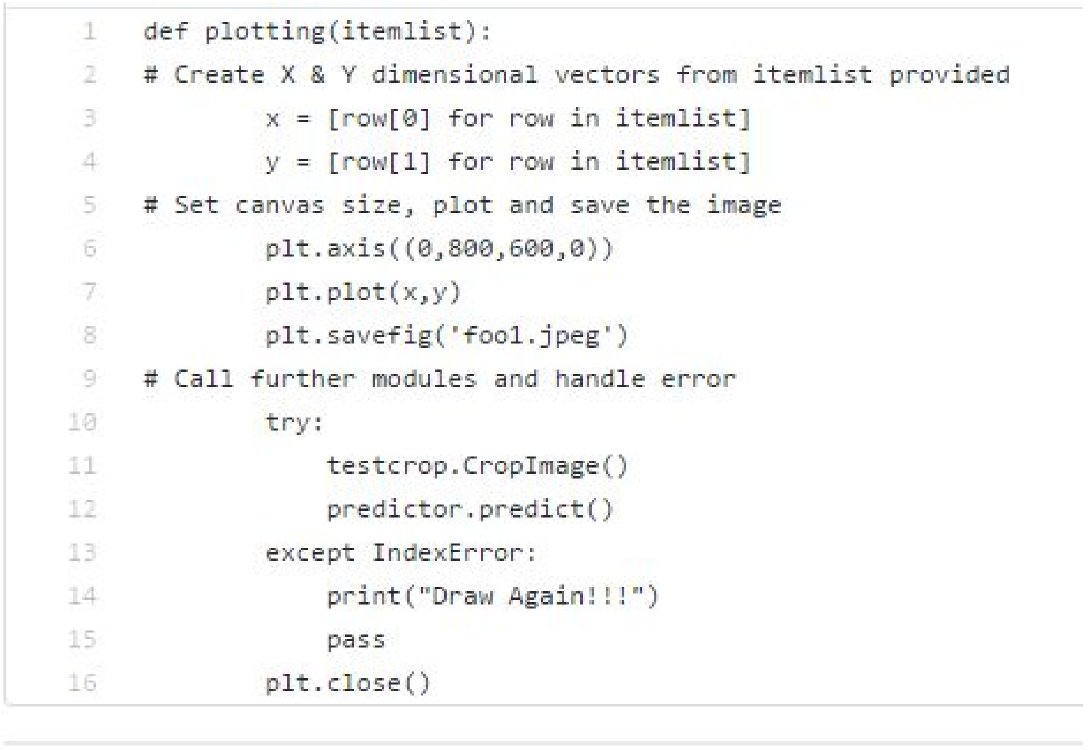
```
pts = deque()
counter = 40
while (True):
# Read frame and filter IR region using threshold filtering
        ret, frame = camera.read()
        frame = cv2.flip(frame, 1)
        gray = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY)
        ret, thresh = cv2.threshold(gray, 250,255,cv2.THRESH_BINARY)
        thresh = cv2.dilate(thresh, None, iterations=7)
        kernel = np.ones((5,5),np.float32)/25
        gray = cv2.filter2D(thresh,-1,kernel)
        thresh = gray
# Find all contours in the frame
        im2, contours, hierarchy = cv2.findContours(thresh,cv2.RETR_TREE,cv2.CHAIN_APPROX_SIMPLE)
# If any contour is found, find its geometric center add it to 'pts'
        if len(contours) > 0:
                counter = 0
                cnt = contours[0]
                x,y,w,h = cv2.boundingRect(cnt)
                cv2.rectangle(frame,(x,y),(x+w,y+h),(0,255,0),3)
                pts.appendleft([int(x+w/2),int(y+h/2)])
# If no contour is found, check if any contour was found for last 5 frames
        else:
                counter = (counter+1) % 100000
                if counter==5:
                        plotting(list(pts))
                        pts.clear()
        cv2.imshow('frame',frame)
        if cv2.waitKey(1) & 0xFF == ord('q'):
                break
```

Figure 13 : Code Snippet of IR Tracking

4.2.2 Plotting function [18]

This module uses the ‘pts’ deque passed to it by the module 4.2.1. This module uses MATPLOTLIB[11] library for plotting the tracked points to a 800x600 (wxh) white canvas.

This plot is saved as a temporary file named “foo1.jpeg”, which is further used by modules 4.2.3 & 4.2.4.



```
def plotting(itemlist):
# Create X & Y dimensional vectors from itemlist provided
        x = [row[0] for row in itemlist]
        y = [row[1] for row in itemlist]
# Set canvas size, plot and save the image
        plt.axis((0,800,600,0))
        plt.plot(x,y)
        plt.savefig('foo1.jpeg')
# Call further modules and handle error
        try:
            testcrop.CropImage()
            predictor.predict()
        except IndexError:
            print("Draw Again!!!")
            pass
        plt.close()
```

Figure 14 : Code Snippet of Plotting Function

The output plot of this module is as shown in the figure, we can see the dimensions of the image generated and the tracked points plotted to a blank background.

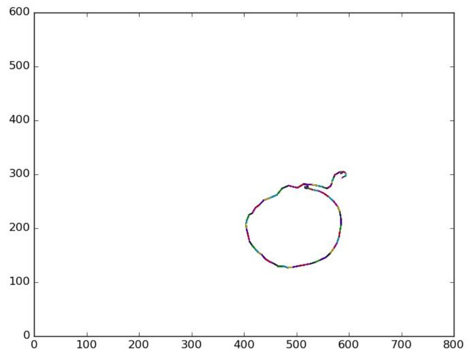
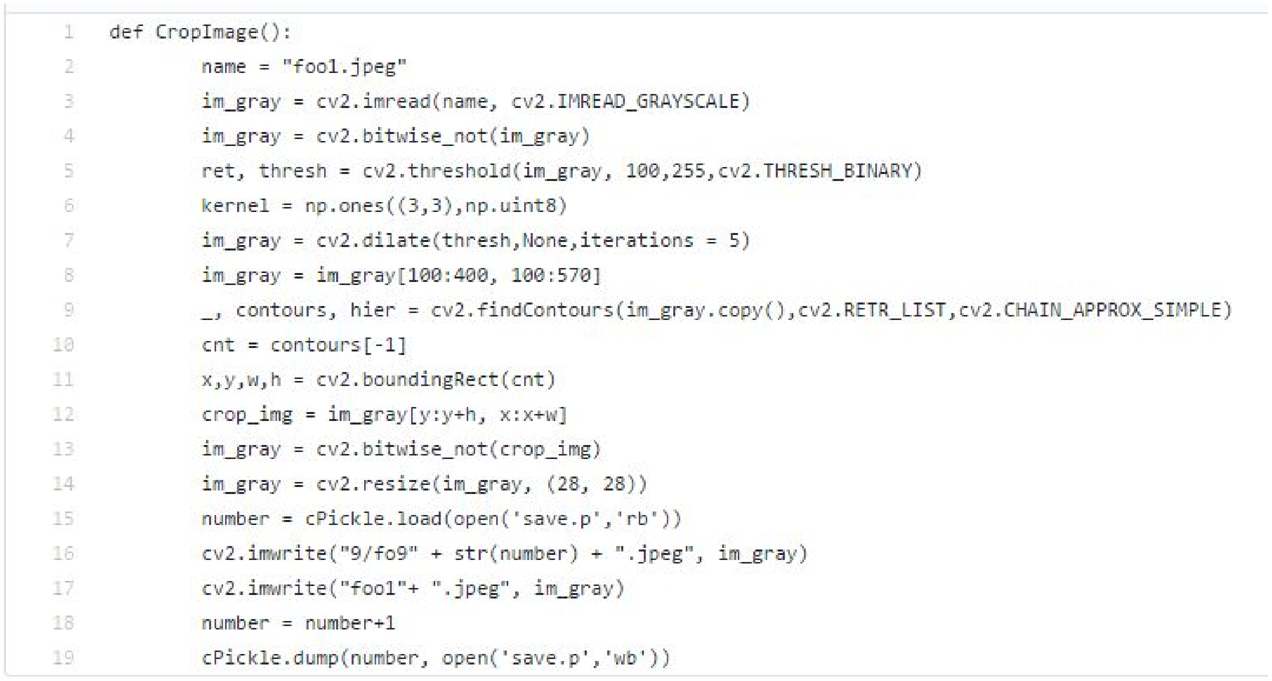


Figure 15 : Output of Plotting Function

4.2.3 Image Preprocessing

This module itself is called by by the module 4.2.2. No argument is passed to this module as an input, instead it uses the temporary file updated by module 4.2.3 as an input. The plot in image “foo1.jpeg” is read using the OPENCV[6]’s ‘imread’ function and further processed to create a binary image that is accepted by the CNN classifier.

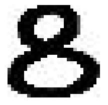
The module first converts the image to a binary space using thresholding function, then finds the area of interest by finding a minimal enclosing rectangle around the plotted pixels. This sub spaced image of the original canvas is then resized to a 28x28 pixel image. Necessary dilation is added to the pixels to achieve better prediction.



```
def CropImage():
        name = "foo1.jpeg"
        im_gray = cv2.imread(name, cv2.IMREAD_GRAYSCALE)
        im_gray = cv2.bitwise_not(im_gray)
        ret, thresh = cv2.threshold(im_gray, 100,255,cv2.THRESH_BINARY)
        kernel = np.ones((3,3),np.uint8)
        im_gray = cv2.dilate(thresh,None,iterations = 5)
        im_gray = im_gray[100:400, 100:570]
        _, contours, hier = cv2.findContours(im_gray.copy(),cv2.RETR_LIST,cv2.CHAIN_APPROX_SIMPLE)
        cnt = contours[-1]
        x,y,w,h = cv2.boundingRect(cnt)
        crop_img = im_gray[y:y+h, x:x+w]
        im_gray = cv2.bitwise_not(crop_img)
        im_gray = cv2.resize(im_gray, (28, 28))
        number = cPickle.load(open('save.p','rb'))
        cv2.imwrite("9/fo9" + str(number) + ".jpeg", im_gray)
        cv2.imwrite("foo1"+ ".jpeg", im_gray)
        number = number+1
        cPickle.dump(number, open('save.p','wb'))
```

Figure 16 : Code Snippet of Image Processing module

The figure attached shows the output generated by this module. We obtain a 28x28 dimensional binary image that has dilated pixels for achieving better accuracy and match the dataset used for transfer learning[13].



module output

4.2.4 Character prediction

This module uses a LeNet architecture[18] CNN[12] trained using transfer learning[13] to predict the character drawn by the user. This module picks up from where 4.2.3 left off. It reads the 28x28 binary image saved by module 2.4.3.

This binary image is fed to the CNN[12] which outputs the predicted character along with the prediction confidence value.

The CNN[12] used is trained using images from MNIST[19] dataset, and images normalised to match them with those generated with module 4.2.3.

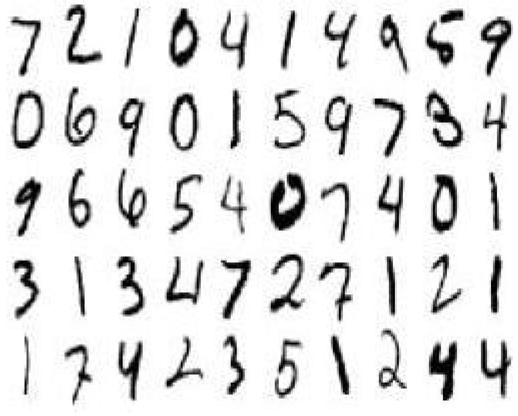


Figure 17 : MNIST example images

4.3 EVOLUTION TO CURRENT MODEL

4.3.1 USING CONVEX HULL

In an application where you want to track a user's movement a skin color histogram can be a helpful tool. This histogram subtracts the background from an image, only leaving parts of the image that contain skin. A much simpler method to detect skin would be to find pixels that are in a certain RGB or HSV range. The problem with this is that changing light environments and skin colors can really mess with the skin detection. Using a histogram tends to be more accurate and takes into account the current light environment.

Once a histogram for the skin color is formed, we can detect the contours having the color of the skin. Hence the hand can easily be detected.

In order to detect the fingertip another function which finds the convex hull can be used. The fingertip will be the farthest point on the convex hull.

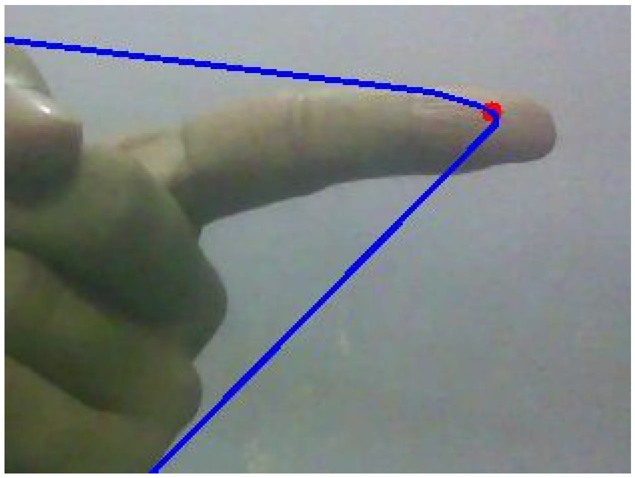
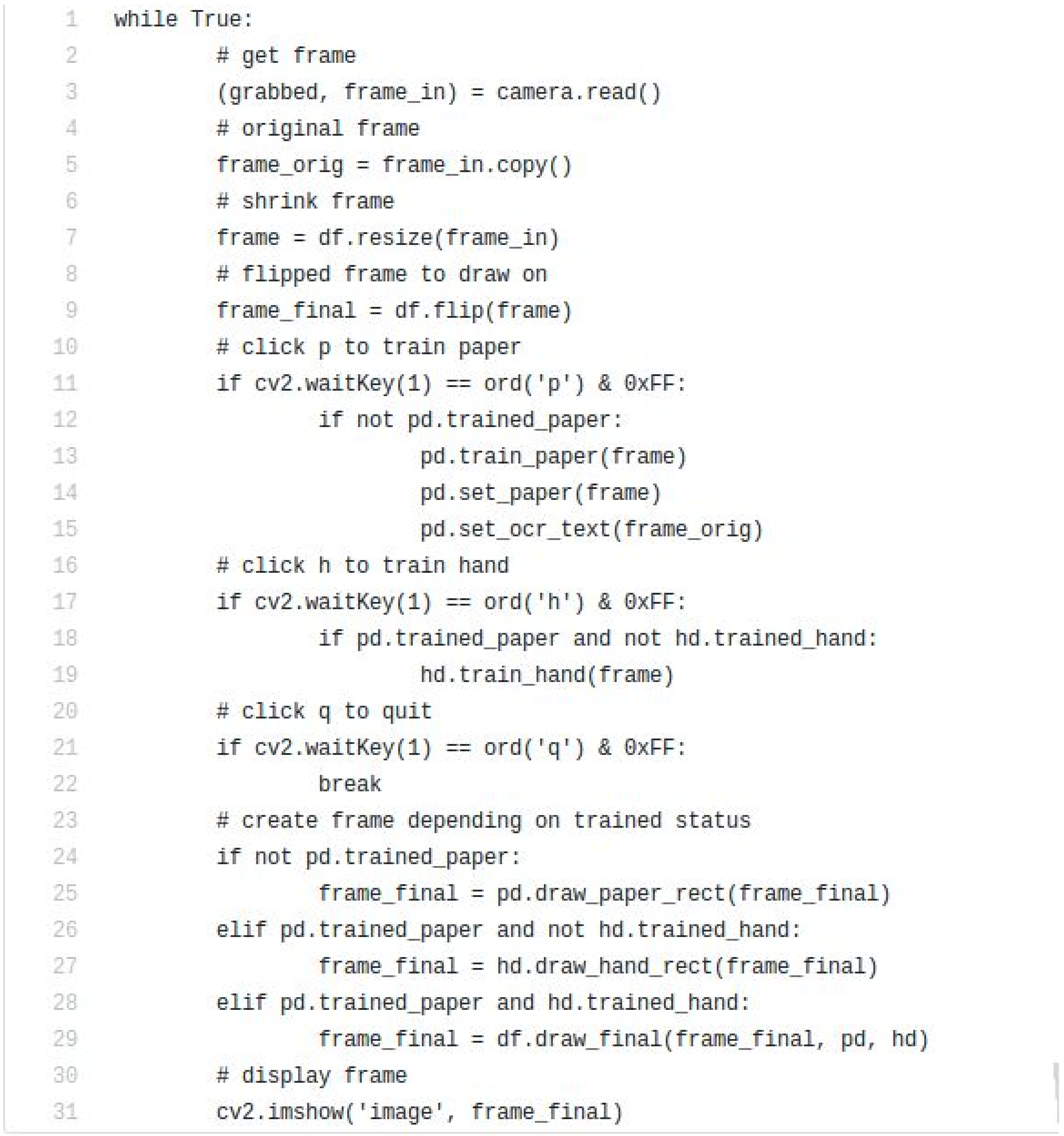


Figure 18 : Finger Tracking using Convex Hull

The main code section implementing the convex hull idea is given below.



```
while True:
        # get frame
        (grabbed, frame_in) = camera.read()
        # original frame
        frame_orig = frame_in.copy()
        # shrink frame
        frame = df.resize(frame_in)
        # flipped frame to draw on
        frame_final = df.flip(frame)
        # click p to train paper
        if cv2.waitKey(1) == ord('p') & 0xFF:
                if not pd.trained_paper:
                        pd.train_paper(frame)
                        pd.set_paper(frame)
                        pd.set_ocr_text(frame_orig)
        # click h to train hand
        if cv2.waitKey(1) == ord('h') & 0xFF:
                if pd.trained_paper and not hd.trained_hand:
                        hd.train_hand(frame)
        # click q to quit
        if cv2.waitKey(1) == ord('q') & 0xFF:
                break
        # create frame depending on trained status
        if not pd.trained_paper:
                frame_final = pd.draw_paper_rect(frame_final)
        elif pd.trained_paper and not hd.trained_hand:
                frame_final = hd.draw_hand_rect(frame_final)
        elif pd.trained_paper and hd.trained_hand:
                frame_final = df.draw_final(frame_final, pd, hd)
        # display frame
        cv2.imshow('image', frame_final)
```

Figure 19 : Code Snippet of Convex Hull finger tracking

The shortcomings of this approach are:

- Anything matching with the skin color will be detected by the program.
- The fingers have to be kept at an awkward angle so that the fingertip is the farthest point in the contour.

- The tracking is dependent on the lighting conditions of the surrounding environment. If the environment is not properly lighted then the skin color may not be detected.

4.3.2 Colored Object Tracking

Tracking a colored object is very easy using OpenCV. This is the first method we used in order to create characters which would be the input for the predictor.
OpenCV provides a function to find contours in the video feed having HSV value in the specified range. Hence, we can easily specify the HSV value of a particular color, thereby finding all the instances of the color in the video feed.
Another function allows us to choose the contour having the largest area out of all the detected contours.
Then a function is used to find the center of the circular contour formed by the previous function and now this center can be tracked at the speed at which the video can be processed, thereby completing the tracking of the object.

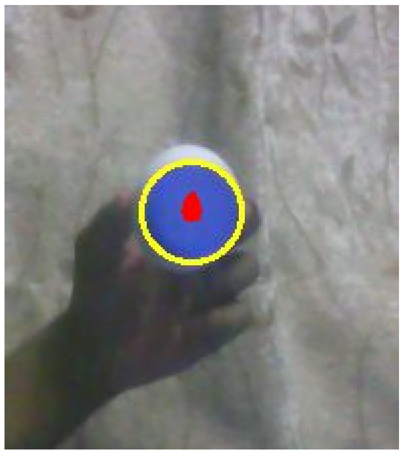
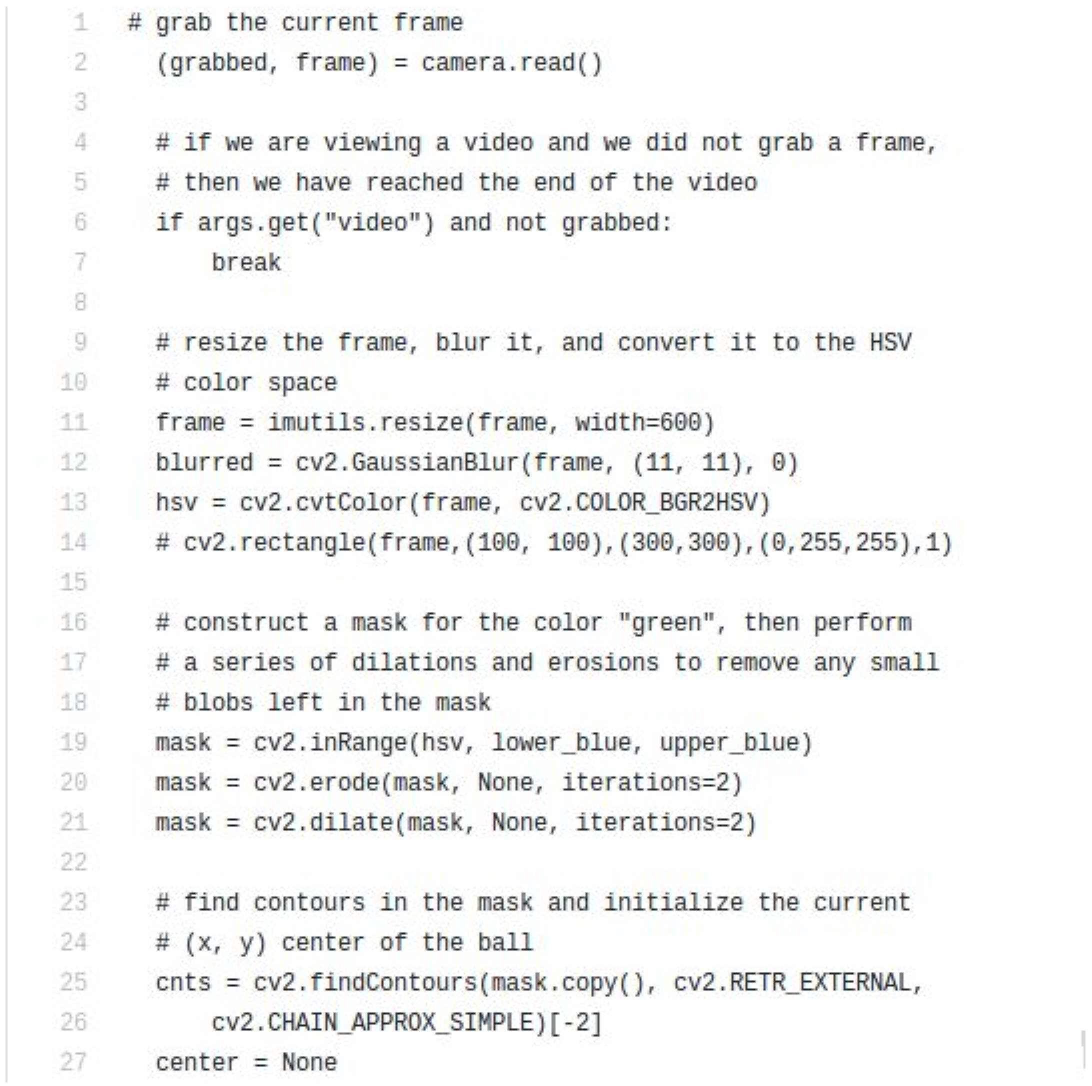


Figure 20 : Colored Object Tracking

The snippet of the main function performing this task has been attached below for reference. The OpenCV functions:

- cv2.inRange
- cv2.erode
- cv2.dialate

Together find the contour and smoothen them so that the small contours are removed and only the large ones are picked to work with.



```
# grab the current frame
  (grabbed, frame) = camera.read()

  # if we are viewing a video and we did not grab a frame,
  # then we have reached the end of the video
  if args.get("video") and not grabbed:
      break

  # resize the frame, blur it, and convert it to the HSV
  # color space
  frame = imutils.resize(frame, width=600)
  blurred = cv2.GaussianBlur(frame, (11, 11), 0)
  hsv = cv2.cvtColor(frame, cv2.COLOR_BGR2HSV)
  # cv2.rectangle(frame,(100, 100),(300,300),(0,255,255),1)

  # construct a mask for the color "green", then perform
  # a series of dilations and erosions to remove any small
  # blobs left in the mask
  mask = cv2.inRange(hsv, lower_blue, upper_blue)
  mask = cv2.erode(mask, None, iterations=2)
  mask = cv2.dilate(mask, None, iterations=2)

  # find contours in the mask and initialize the current
  # (x, y) center of the ball
  cnts = cv2.findContours(mask.copy(), cv2.RETR_EXTERNAL,
      cv2.CHAIN_APPROX_SIMPLE)[-2]
  center = None
```

Figure 21 : Code Snippet of Colored Object Tracking

The shortcomings of this approach are:

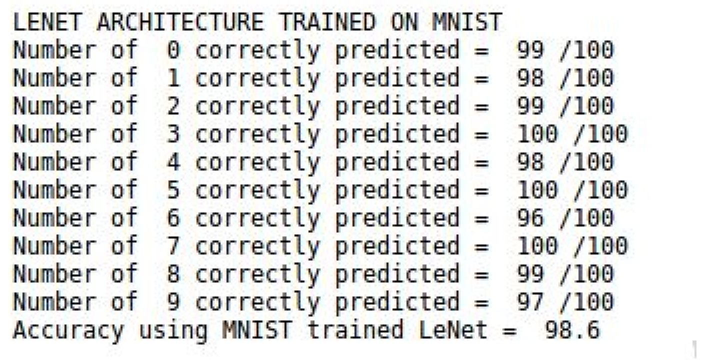
- The surrounding environment should not have the color which is being tracked by the code originally.
- The tracking is dependent on the lighting conditions of the surrounding environment. If the environment is not properly lighted then the color may not be detected.
- The object should not be placed very far, as it will hamper the detection process.
- The tracking is heavily dependent on the quality of the camera.

CHAPTER-5
CONCLUSIONS AND RESULTS

The final system was successfully created which can take input from the IR emitting device and predict the corresponding digit.

The prediction was done using convolutional neural networks. CNN can have different architectures. Some of the most common architectures used with CNN are LeNet, AlexNet, GoogLeNet etc. We tried the LeNet and GoogLeNet architecture in our project.

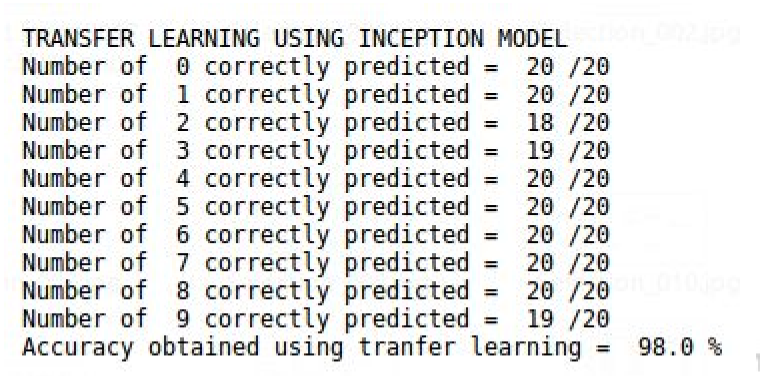
- Using LeNet Architecture trained on MNIST dataset



```
LENET ARCHITECTURE TRAINED ON MNIST
Number of  0 correctly predicted =  99 /100
Number of  1 correctly predicted =  98 /100
Number of  2 correctly predicted =  99 /100
Number of  3 correctly predicted =  100 /100
Number of  4 correctly predicted =  98 /100
Number of  5 correctly predicted =  100 /100
Number of  6 correctly predicted =  96 /100
Number of  7 correctly predicted =  100 /100
Number of  8 correctly predicted =  99 /100
Number of  9 correctly predicted =  97 /100
Accuracy using MNIST trained LeNet =  98.6
```

Figure 22 : Accuracy Achieved by LeNet Architecture trained on MNIST

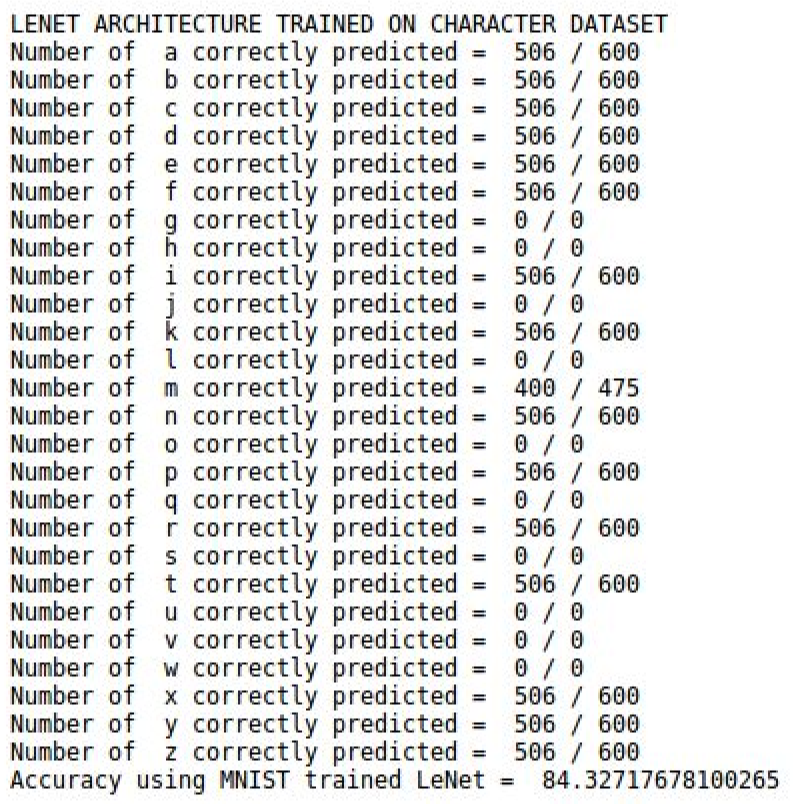
- Using transfer learning with the Inception Model



```
TRANSFER LEARNING USING INCEPTION MODEL
Number of  0 correctly predicted =  20 /20
Number of  1 correctly predicted =  20 /20
Number of  2 correctly predicted =  18 /20
Number of  3 correctly predicted =  19 /20
Number of  4 correctly predicted =  20 /20
Number of  5 correctly predicted =  20 /20
Number of  6 correctly predicted =  20 /20
Number of  7 correctly predicted =  20 /20
Number of  8 correctly predicted =  20 /20
Number of  9 correctly predicted =  19 /20
Accuracy obtained using tranfer learning =  98.0 %
```

Figure 23 : Accuracy Achieved by Inception Model trained on MNIST

- Using LeNet Architecture for character dataset



```
LENET ARCHITECTURE TRAINED ON CHARACTER DATASET
Number of  a correctly predicted =  506 / 600
Number of  b correctly predicted =  506 / 600
Number of  c correctly predicted =  506 / 600
Number of  d correctly predicted =  506 / 600
Number of  e correctly predicted =  506 / 600
Number of  f correctly predicted =  506 / 600
Number of  g correctly predicted =  0 / 0
Number of  h correctly predicted =  0 / 0
Number of  i correctly predicted =  506 / 600
Number of  j correctly predicted =  0 / 0
Number of  k correctly predicted =  506 / 600
Number of  l correctly predicted =  0 / 0
Number of  m correctly predicted =  400 / 475
Number of  n correctly predicted =  506 / 600
Number of  o correctly predicted =  0 / 0
Number of  p correctly predicted =  506 / 600
Number of  q correctly predicted =  0 / 0
Number of  r correctly predicted =  506 / 600
Number of  s correctly predicted =  0 / 0
Number of  t correctly predicted =  506 / 600
Number of  u correctly predicted =  0 / 0
Number of  v correctly predicted =  0 / 0
Number of  w correctly predicted =  0 / 0
Number of  x correctly predicted =  506 / 600
Number of  y correctly predicted =  506 / 600
Number of  z correctly predicted =  506 / 600
Accuracy using MNIST trained LeNet =  84.32717678100265
```

Figure 24 : Accuracy Achieved by LeNet Architecture trained on Character Dataset

CHAPTER - 6
FUTURE SCOPE

- The project in the current stage recognizes only digits and some mathematical operators, this can be extended to include other characters as well.

- Once the character recognition is included in the project, it can be extended to complete sentence detection (using a combination of LSTMs and CNNs).

- Using better computing power we can train our predictor more extensively, to get even better prediction accuracy, for eg. using a GPU.

- Better quality camera can be used to make the process of taking input more efficient.

CHAPTER - 7

REFERENCES

[1] Aditya G. Joshi, Darshana V. Kotle, Ashish V. Bandgar, Aakash S. Kadlak, N. S. Patil , "Touchless Writer A Hand Gesture Recognizer For English Characters", *International Journal of Advanced Computational Engineering and Networking, ISSN: 2320-2106, Volume-3, Issue-5, May-2015*

[2] Bikash Chandra Karmokar, M. A. Parvez Mahmud, Md. Kibria Siddiquee, Kawser Wazed Nafi, Tonny Shekha Kar, "Touchless Written English Characters Recognition using Neural Network", *International Journal of Computer & Organization Trends –Volume2 Issue3- 2012*

[3] Aditya G. Joshi, Darshana V. Kolte , Ashish V. Bandgar, Aakash S. Kadlak, Ms. P. S. Khude, "Touchless Writer: Recognizing various Hand Gestures using Artificial Neural Network (ANN) through Webcam And Objects", *International Journal of Advance Foundation and Research in Computer (IJAFRC) Volume 2, Special Issue (NCRTIT 2015), January 2015. ISSN 2348 - 4853*

[4] Python - Tutorial, *https://www.tutorialspoint.com/python/*

[5] What is Python? Executive Summary, *https://www.python.org/doc/essays/blurb/*

[6] OpenCV, *https://en.wikipedia.org/wiki/OpenCV*

[7] Sonam Sawant, "Turn Webcam to IR Sensitive Camera", *http://www.instructables.com/id/turn-webcam-to-IR-sensitive-camera/*

[8] Adrian Rosebrock, "OpenCV Track Object Movement", *http://www.pyimagesearch.com/2015/09/21/opencv-track-object-movement/*

[9]Ben Meline, "Finger Tracking with OpenCV and Python", http://www.benmeline.com/finger-tracking-with-opencv-and-python/

[10] Machine learning, *https://en.wikipedia.org/wiki/Machine_learning*

[11] Artificial neural network, *https://en.wikipedia.org/wiki/Artificial_neural_network*

[12] CS231n Convolutional Neural Networks for Visual Recognition,

http://cs231n.github.io/convolutional-networks/

[13] Transfer Learning, http://cs231n.github.io/transfer-learning/

[14] Eugenio Culurciello, “Neural Network Architectures”,

https://medium.com/towards-data-science/neural-network-architectures-156e5bad51ba
[15] Webcam, https://en.wikipedia.org/wiki/Webcam
[16]What is an Infrared Emitter?,
http://www.futureelectronics.com/en/optoelectronics/infrared-emitters.aspx
[17] Matplotlib, https://matplotlib.org/

[18] LeCun et al., “Gradient-Based Learning Applied to Document Recognition”

[19] Yann LeCun, Corinna Cortes, Christopher J.C. Burges, “THE MNIST DATABASE of handwritten digits”, http://yann.lecun.com/exdb/mnist/

[20] Christian Szegedy, Vincent Vanhoucke, Sergey Ioffe, Jonathon Shlens, Zbigniew Wojna, “Rethinking the Inception Architecture for Computer Vision”, https://arxiv.org/pdf/1512.00567.pdf

[21] TensorFlow, https://en.wikipedia.org/wiki/TensorFlow