DACTYLOLOGY RECOGNITION SYSTEM

Major Project Report submitted in partial fulfillment of the requirements for the award of the Degree of B.E in Computer Science and Engineering

By

A.Meghna Reddy 160618733069

Under the Guidance of

Dr. M. Swapna
Associate Professor, Department of Computer Science & Engineering

Department of Computer Science and Engineering
Stanley College of Engineering & Technology for Women (Autonomous)

Chapel Road, Abids, Hyderabad – 500001
(Affiliated to Osmania University, Hyderabad, Approved by AICTE, Accredited by NBA & NAAC with A Grade)

2021-22

Stanley College of Engineering & Technology for Women

(Autonomous)

Chapel Road, Abids, Hyderabad – 500001

(Affiliated to Osmania University, Hyderabad, Approved by AICTE, Accredited by NBA & NAAC with A Grade)

CERTIFICATE

This is to certify that major project report entitled ***Dactylology Recognition System*** being submitted by

A. Meghna Reddy 160618733069

In partial fulfillment for the award of the Degree of Bachelor of Engineering in Computer Science & Engineering to the Osmania University, Hyderabad is a record of bonafide work carried out under my guidance and supervision. The results embodied in this project report have not been submitted to any other University or Institute for the award of any Degree or Diploma.

Guide

Dr. M. Swapna
Associate Professor, Dept of CSE

Head of the Department

Dr Y V S S Pragathi
Prof & HoD, Dept of CSE

Project Coordinator

K. Srilatha
Assistant Professor, CSE

External Examiner

DECLARATION

We hereby declare that major project work entitled ***Dactylology Recognition System*** submitted to the Osmania University, Hyderabad, is a record of original work done by us. This project work is submitted in partial fulfillment of the requirements for the award of the degree of the B.E in Computer Science and Engineering.

A. Meghna Reddy 160618733069

ACKNOWLEDGEMENT

First of all, we are grateful to **The Almighty God** for establishing us to complete this Major Project. We pay our respects and love to our parents and all our family members and friends for their love and encouragement throughout our career.

We wish to express our sincere thanks to **Shri. Kodali Krishna Rao,** Correspondent and Secretary, Stanley College of Engineering and Technology for Women, for providing us with all necessary facilities.

We place on record our sincere gratitude to **Prof. Satya Prasad Lanka**, Principal, for his constant encouragement.

We deeply express our sincere thanks to our Head of the Department, **Prof Y V S S Pragathi**, for encouraging and allowing us to present the Major Project on the topic "**Dactylology Recognition System"** at our department premises for the fulfillment of the requirements leading to the award of the B.E. degree.

It is our privilege to express sincere regards to our project guide **Dr. M.Swapna**, for the valuable inputs, able guidance, encouragement, whole-hearted co-operation and constructive criticism throughout the duration of our project.

We take this opportunity to thank all our faculty, who have directly or indirectly helped our project. Last but not least, we express our thanks to our friends for their cooperation and support.

ABSTRACT

Sign Language is one of the oldest and most natural form of language for interaction between normal and deaf & mute people, but since most people do not know sign language and interpreters are very difficult to find, we have come up with a real time method using neural networks for fingerspelling based American sign Language. Sign language is a visual language and consists of 3 major components: Fingerspelling, Word level vocabulary and Facial expressions.
In our project we basically focus on producing a model which can recognize Fingerspelling based hand gestures in order to form a complete word by combining each gesture.

We follow 4 steps in the hand gesture recognition: i) Data Acquisition, ii) Data preprocessing, iii) Feature extraction iv) Gesture classification. We aim to collect 600 RGB images each for 26 alphabets and apply a filter called Gaussian blur to each of the images. This filter gives a black& white outline of the hand gesture which makes it easier for computation. The images are then passed through convolution neural network classifier which predicts the class of the hand gestures. Our method provides 95% accuracy for the 26 letters of the alphabet

Keywords: *Dactylology, Image Recognition, Gaussian Blur Filter, Image Processing, Deep Learning, Convolutional Neural networks.*

Tables of Contents

List of Figures

CHAPTER 1

INTRODUCTION

1.1 About Project

Sign language is a form of communication used by people with impaired hearing and speech. People use sign language gestures as a means of non-verbal communication to express their thoughts and emotions.

Gesture based communications (otherwise called marked dialects) are dialects that utilize the visual-manual methodology to pass on significance. Gesture based communications are communicated through manual verbalizations in blend with non-manual components. Gesture based communications are undeniable characteristic dialects with their own language structure and vocabulary. Gestures are performed by deaf and dumb community to perform sign language. People utilize gesture-based communication for their correspondence when broadcasting audio is impossible, or typing and writing is troublesome, yet there is the possibility of vision. Around then communication via gestures is the just path for trading data between individuals.

Sign languages generally do not have any linguistic relation to the spoken languages of the lands in which they arise. The correlation between sign and spoken languages is complex and varies depending on the country more than the spoken language. We use a custom recorded American Sign Language data set based on an existing data set for training the model to recognize gestures. We propose to use a CNN (Convolutional Neural Networks) named Inception to extract spatial features from the video stream for Sign Language Recognition (SLR).

Some countries such as Belgium, the UK, the USA or India may have more than one sign language. Hundreds of sign languages are in use around the world, for instance, Japanese Sign Language, British Sign Language (BSL), Spanish Sign Language, Turkish Sign Language. Sign language is a visual language and consists of 3 major components: • Fingerspelling - Used to spell words letter by letter. Word level sign vocabulary - Used for the majority of communication.

Non-Manual features - Facial expressions and tongue, mouth and body position. American Sign Language (ASL) is a characteristic language that fills in as the prevalent sign language of Deaf people. American Sign Language (ASL) is a visual language. With marking. the brain processes phonetic data through the eyes. The shape, arrangement, and movement of the hands, just as facial expressions and body movements, all play significant parts in passing on data, ASL has the same linguistic properties as spoken languages.

Problem Statement:

Understanding the precise meaning of deaf and dumb people's symbolic gestures and converting it into understandable language (Text) and finally into speech using Google Text to Speech (gTTS) module.

1.2 Objectives of the Project

The aim of this project is to use the corresponding gesture to recognize Alphabets in Indian Sign Language. The identification of gestures and sign languages is a well-studied subject in American Sign Language, but it has received little attention in Indian Sign Language. We want to solve this issue, but instead of using high-end technologies like gloves or the Kinect, we want to recognize gestures from photographs (which can be accessed from a webcam), and then use computer vision and machine learning techniques to extract specific features and classify them.

For interaction between normal people and D&M people a language barrier is created as a sign language structure which is different from normal text. So, they depend on vision-based communication for interaction.

If there is a common interface that converts the sign language to text the gestures can be easily understood by the other people. So, research has been made for a vision-based interface system where D&M people can enjoy communication without really knowing each other's language.

The aim is to develop a user-friendly human computer interface (HCI) where the computer understands the human sign language. There are various sign languages all over the world, namely American Sign Language (ASL), French Sign Language, British Sign Language (BSL), Indian Sign language, Japanese Sign Language and work has been done on other languages all around the world. In recent years there has been tremendous research done on hand gesture recognition. With the help of literature survey done we realized the basic steps in hand gesture recognition are: - Data acquisition, Data preprocessing, Feature extraction, Gesture classification

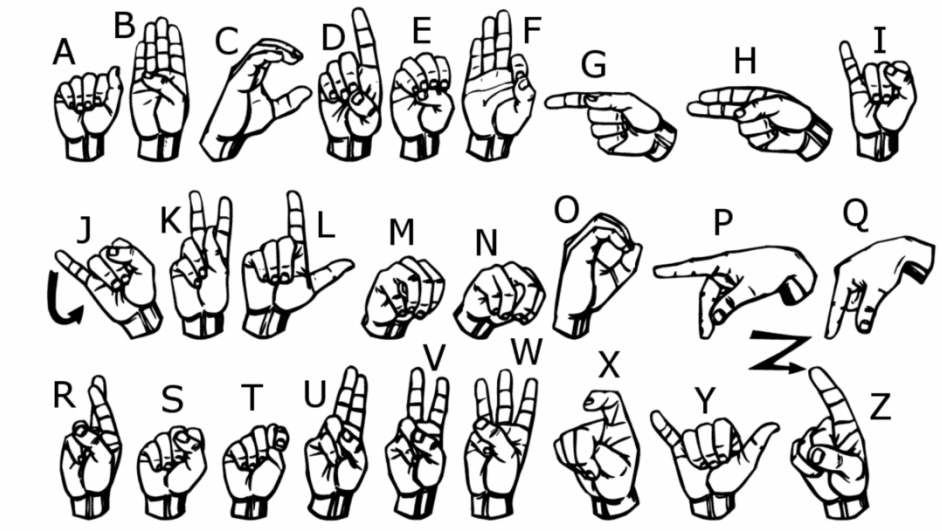


Fig 1.1: ASL Sign Language

1.3 Scope of the Project

We are planning to achieve higher accuracy even in case of complex backgrounds by trying out various background subtraction algorithms. We are also thinking of improving preprocessing to predict gestures in low light conditions with a higher accuracy.

1.4 Advantages

The output of the hand gestures will be displayed in the text format on the screen. This makes the system more efficient and the communication with the hearing and speech impaired people easier. Simple, fast and easy to implement. The system successfully recognizes the static and dynamic gestures using the web camera. The use of gTTS converts the text to audio, which makes this system user friendly. A spell checker is also added to make the process of converting sign language much faster.

1.5 Disadvantages

- Performance might decrease when irrelevant objects might overlap with the hand. A well lit room with a clear background is required to give an accurate prediction. Thus, performance reduces in dark environments.

- Irrelevant objects might overlap with the hand Wrong object extraction appeared if the extraction appeared if the objects were larger than the hand. Performance recognition algorithm decreases when the distance is greater than 1.5 meters between the user and the camera. System limitations restrict the applications such as the arm must be vertical, the palm is facing the camera and the finger color must be basic color such as either red or green or blue, Ambient light affects the color threshold.

- Proper displaying of the sign letters is required for its recognition. The amount of time taken to move recognized characters to sentences is more.

1.6 Applications

The proposed framework will assist the hard of hearing and nearly deaf with conveying individuals from the local area. For instance, there have been occurrences where the individuals who are hard of hearing have experienced issues speaking with people on call when out of luck. In spite of the fact that responders may be preparing on the essentials of ASL, it is ridiculous to expect everybody to turn out to be completely familiar with gesture-based communication. Down the line, headways like these in PC acknowledgment could help a specialist on call in comprehension and aiding those that can't impart through discourse. Another application is to empower the hard of hearing and nearly deaf equivalent admittance to video discussions, regardless of whether in an expert setting or while attempting to speak with their medical care suppliers through Tele wellbeing. Rather than utilizing essential talk, these progressions would permit the meeting debilitated admittance to viable video correspondence.

1.7 Hardware and Software Requirements

1.7.1 Hardware Requirements

- Windows 10
- Intel Core i5
- 8 GB RAM
- 500 GB Internal Storage

1.7.2 Software Requirements

Libraries -

- TensorFlow,Keras,gTTS
- Numpy,Hunspell,Tkinter
- OpenCV

Environment and Framework

- Jupyter IDE

Tensor Flow -

TensorFlow is an open-source software library for dataflow programming across a range of tasks. It is a symbolic math library, and is also used for machine learning applications such as neural networks. It is used for both research and production at Google. TensorFlow was developed by the Google brain team for internal Google use. It was released under the Apache 2.0 open-source library on November 9, 2015.

TensorFlow is Google Brain's second-generation system. Version 1.0.0 was released on February 11, 2017. While the reference implementation runs on single devices, TensorFlow can run on multiple CPUs and GPUs (with optional CUDA and SYCL extensions for general-purpose computing on graphics processing units). TensorFlow is available on 64-bit Linux, macOS, Windows, and mobile computing platforms including Android and iOS. Its flexible architecture allows for the easy deployment of computation across a variety of platforms (CPUs, GPUs, TPUs), and from desktops to clusters of servers to mobile and edge devices.

Keras -

Keras is an open-source software library that provides a Python interface for artificial neural networks. Keras acts as an interface for the TensorFlowlibrary.

Keras contains numerous implementations of commonly used neural-network building blocks such as layers, objectives, activation functions, optimizers, and a host of tools to make working with image and text data easier to simplify the coding necessary for writing deep neural network code. The code is hosted on GitHub, and community support forums include the GitHub issues page, and a Slack channel.

In addition to standard neural networks, Keras has support for convolutional and recurrent neural networks. It supports other common utility layers like dropout, batch normalization, and pooling.

Keras allows users to productize deep models on smartphones (iOS and Android), on the web, or on the Java Virtual Machine. It also allows use of distributed training of deep-learning models on clusters of Graphics processing units (GPU) and tensor processing units (TPU).

OpenCV -

OpenCV (Open-Source Computer Vision Library) is released under a BSD license and hence it’s free for both academic and commercial use. It has C++, Python and Java interfaces and supports Windows, Linux, Mac OS, iOS and Android. OpenCV was designed for computational efficiency and with a strong focus on real-time applications. Written in optimized C/C++, the library can take advantage of multi-core processing. Enabled with OpenCL, it can take advantage of the hardware acceleration of the underlying heterogeneous compute platform.

Adopted all around the world, OpenCV has more than 47 thousand users and an estimated number of downloads exceeding 14 million. Usage ranges from interactive art, to mine inspection, stitching maps on the web or through advanced robotics.

GTTS - (Google Text to Speech)

There are several APIs available to convert text to speech in Python. One of such APIs is the Google Text to Speech API commonly known as the gTTS API. gTTS is a very easy to use tool which converts the text entered, into audio which can be saved as a mp3 file.

The gTTS API supports several languages including English, Hindi, Tamil, French, German and many more. The speech can be delivered in any one of the two available audio speeds, fast or slow. However, as of the latest update, it is not possible to change the voice of the generated audio.

Hunspell -

Hunspell is a spell checker and morphological analyser designed for languages with rich morphology and complex word compounding and character encoding, originally designed for the Hungarian language.

Hunspell is based on MySpell and is backward-compatible with MySpell dictionaries. While MySpell uses a single-byte character encoding, Hunspell can use Unicode UTF-8-encoded dictionaries.It is a set of Python bindings for the Hunspell spell checker engine. It lets developers load Hunspell dictionaries, check words, get suggestions, add new words, etc. It also provides some basic morphological analysis related methods.

Tkinter -

Python has a lot of GUI frameworks, but Tkinter is the only framework that's built into the Python standard library. Tkinter has several strengths. It's cross-platform, so the same code works on Windows, macOS, and Linux. Visual elements are rendered using native operating system elements, so applications built with Tkinter look like they belong on the platform where they're run.

Tkinter is lightweight and relatively painless to use compared to other frameworks. This makes it a compelling choice for building GUI applications in Python, especially for applications where a modern sheen is unnecessary, and the top priority is to quickly build something that's functional and cross-platform.
The foundational element of a Tkinter GUI is the window. Windows are the containers in which all other GUI elements live. These other GUI elements, such as text boxes, labels, and buttons, are known as widgets. Widgets are contained inside of windows.

Fig 1.2: Tkinter window Example

Jupyter Notebook -

Jupyter like the planet; some users pronounce "py" is a project and community whose goal is to "develop open-source software, open-standards, and services for interactive computing across dozens of programming languages".It was spun off from IPython in 2014 by Fernando Pérez and Brian Granger. Project Jupiter's name is a reference to the three core programming languages supported by Jupyter, which are Julia, Python and R, and also a homage to Galileo's notebooks recording the discovery of the moons of Jupiter. Project Jupyter has developed and supported the interactive computing products Jupyter Notebook, JupyterHub, and JupyterLab. Jupyter is financially sponsored by NumFOCUS

Convolution Neural network

CNNs use a variation of Multilayer Perceptron is designed to require minimal preprocessing. They are also known as shift invariant or space invariant artificial neural networks (SIANN), based on their shared-weights architecture and translation invariance characteristics.

Convolutional networks were inspired by biological processes in that the connectivity pattern between neurons resembles the organization of the animal visual cortex. Individual cortical neurons respond to stimuli only in a restricted region of the visual field known as the receptive field. The receptive fields of different neurons partially overlap such that they cover the entire visual field.

CNNs use relatively little pre-processing compared to other image classification algorithms. This means that the network learns the filters that in traditional algorithms were hand-engineered. This independence from prior knowledge and human effort in feature design is a major advantage. They have applications in image and video recognition, recommender systems, image classification, medical image analysis, and natural language processing.

Challenges Faced:

There were many challenges faced by us during the project. The very first issue we faced was the dataset. We wanted to deal with raw images and that too square images as CNN in Keras as it was a lot more convenient working with only square images. We couldn't find any existing dataset for that hence we decided to make our own dataset. Second issue was to select a filter which we could apply on our images so that proper features of the images could be obtained and hence then we could provide that image as input for CNN model. We tried various filters including binary threshold, canny edge detection, gaussian blur etc. but finally we settled with a gaussian blur filter. More issues were faced relating to the accuracy of the model we trained in earlier phases which we eventually improved by increasing the input image size and also by improving the dataset.

CHAPTER 2

LITERATURE SURVEY

2.1 Existing System

Sign Language is the most natural and expressive way for the hearing impaired people.Many new techniques have been developed recently in this area.But the existing system represented in a IEEE research paper by author Mahesh Kumar NB uses Linear Discriminant Analysis (LDA) algorithm was used for gesture recognition and recognized gesture is converted into text.LDA mainly used in statistics, pattern recognition and machine learning. It is used to find a linear combination of features that characterizes or separates two or more classes of objects or events.

The current system has collected around 10 images for each sign. These images are included in the training and testing database. The captured image at a distance is adjusted by the signer to get the required image clarity.Due to fewer images given for training, the accuracy of such a system drops to 85-88%. The image components are extracted by Morphological Filtering tools which are useful for representation and description of shape.The features extracted from the segmentation operation used for gesture recognition. The smooth contour is obtained by removing the noise from the images with Morphological filtering techniques.

Ankit Ojha and Ayush Pandey uses a system based on Spanish speaking language which converts the basic words into the Spanish language which is good for Spanish deaf people as it will provide them a stance to understand the sign language at pace as it'll be converted in a Spanish language rather than in English which is popularly used as ASL. The CNN model used for training consists of 2 CNN layers, followed by 2 dense layers and a dropout layer in between. Each class has a corresponding probability of prediction allocated to it. The class with the maximum probability is displayed as the predicted gesture.

To discover bounding packing containers of various objects, we used the Gaussian historical past subtraction which used a technique to version each history pixel with the resource of a mixture of K Gaussian set distributions (k varies from 3 to 5). The possibly historical past colorations are those that stay longer and are greater static. On those fluctuating pixels, we design a square bounding field. After Obtaining all the gesture and heritage, a Convolutional NN model has been designed using those photos to separate the gesture symptoms and signs from the historical beyond. These function maps explain that the CNN can understand the common unexposed structures of some of the gesture indicators within the training set, and is therefore able to distinguish between a gesture and the past. This system though gives a good accuracy of 95% but is limited to only Spanish language and cannot be used by people all over the world.

Archana S. Ghotkar and Dr. Gajanan K. Kharate proposes a HGR system.Hand tracking and segmentation are to be done on captured video and feature extraction is to be done on segmented hand image which is further given to classification and recognition phase.They have used Indian Sign Language system limited to the Indian population,where there is a lot of deviation in skin tone.They used HSV and YCbCr color model for skin color detection.Supervised as well as Unsupervised Learning Model such as Bayesian classifier can be used for skin color segmentation. For pattern recognition they have used supervised classification technique i.e Bayesian classifier and Hidden Morkov model was used for construction of sentence followed by sequence of signs.This proposed system produced an accuracy of 83% with tracking failed for rapid movement of hand.

Dr.Rohini Deshpande and Y Divya Gnana Preetham proposed the braille pad which comprises of electromagnets is controlled through Arduino UNO, each braille cell is replaced by an electromagnet, and the functionality of electromagnets is controlled by the Arduino according to the text.For processing the image, they have used canny edge detection algorithm.Canny edge detection is a technique used to extract the useful information and reduce the amount of data to be processed.Tracking of edges was conducted using Hysteresis where few weak pixels are converted into strong ones. If there is at least one strong pixel present around the weak pixel it reconstructs the weak pixel into a strong one.

The output here is in the form of braille code which can be understood only by the blind people, so the idea of easy communication between normal and D&M people cannot be achieved in this case.Also, the use of arduino and other hardware increases the complexity of the system and slows it down.

S.No	Paper Title	Author's Name	Problem	Solution	Future work	Year
1.	Conversion of Sign Language into Text	Mahesh Kumar N B	Lower accuracy due to less images for training	Provide more images for training	Other preprocessing techniques can be evaluated	2016
2.	Sign Language to Text and Speech Translation in Real Time Using Convolutional Neural Network	Ankit Ojha ,Ayush Pandey	Use of spanish language signs	Use of a more universal sign language	Application on ASL signs	2017
3.	Study of vision based hand gesture recognition using Indian Sign language	Archana S. Ghotkar and Dr. Gajanan K. Kharate	Use of HSV and YCbCr color model for skin color detection.A supervised bayesian model for training	Should work on the performance	Include working with rapid hand movements	2019
4.	Conversion of Sign Language to Text Using Image Processing	Dr.Rohini Deshpande and Y Divya Gnana Preetham	Use of canny edge algorithm for preprocessing	Should use better preprocessing technique	Deploying into a web app	2020

Fig 2.1 : Table Representing the existing systems

2.2 Proposed System

Dactylology is the technique of communicating by signs made with the fingers, especially in the manual alphabets used by the deaf. For interaction between normal people and D&M people a language barrier is created as sign language structure which is different from normal text. So they depend on vision based communication for interaction. If there is a common interface that converts the sign language to text the gestures can be easily understood by the other people. So research has been made for a vision based interface system where D&M people can enjoy communication without really knowing each other's language.

American Sign Language is a predominant sign language and is widely used by people all over the world. ASL alphabets are quite different from the gestures, alphabets involve manual letters from A-Z with each letter shown to convey a message, while gestures convey an entire emotional together.Creating an human interactive interface that takes ASL as the input and converts it into speech can be used by people all over the world without any barriers. The proposed system with a live video input is subjected to the trained sign recognition model, trained victimization neural networks of deep learning, the model acknowledges the alphabet, combines recognised letters to words and finally combines words to sentences. A Hunspell spell checker is also included to suggest or correct words during the recognition process.The sentences are converted to speech/audio by using the gTTS library.

The planned system is tested with associated degreed experimented against an in-built camera, CNN is employed for Sign Language Prediction. A Convolutional Neural Networks may be a Deep Learning algorithmic program that absorbs associate degree input image.Then assigns importance to numerous aspects/objects within the image and be ready to differentiate one from the opposite.

The system is a vision-based approach. All the signs are represented with bare hands and so it eliminates the problem of using any artificial devices for interaction. For the project we tried to find already made datasets but we couldn’t find dataset in the form of raw images that matched our requirements. All we could find were the datasets in the form of RGB values. Hence, we decided to create our own data set. Steps we followed to create our data set are as follows. We used the Open computer vision (OpenCV) library in order to produce our dataset. Firstly, we captured around 800 images of each of the symbols in ASL for training purposes and around 200 images per symbol for testing purposes.

First, we capture each frame shown by the webcam of our machine. In each frame we define a region of interest (ROI) which is denoted by a blue bounded square.

From this whole image we extract our ROI which is RGB and convert it into gray scale Image as shown below.

Fig 2.2: Gray Scale Image

Finally, we apply our gaussian blur filter to our image which helps us extract various features of our image. The image after applying gaussian blur looks like below.

Fig 2.3: Gaussian Blur Image

Training in the proposed system:

Layer 1: CNN Model:

1. **1st Convolution Layer: The** input picture has a resolution of 128x128 pixels. It is first processed in the first convolutional layer using 32 filter weights (3x3 pixels each). This will result in a 126X126 pixel image, one for each Filter-weights.

2. **1st Pooling Layer:** The pictures are Downsampled using max pooling of 2x2 i.e we keep the highest

value in the 2x2 square of the array. Therefore, our picture is Downsampled to 63x63 pixels.

3. **2nd Convolution Layer:** Now, these 63 x 63 from the output of the first pooling layer is served as an input to the second convolutional layer. It is processed in the second convolutional layer using 32 filter weights (3x3 pixels each).This will result in a 60 x 60 pixel image.

4. **2nd Pooling Layer:** The resulting images are Downsampled again using a max pool of 2x2 and is reduced to 30 x 30 resolution of images.

5. **1st Densely Connected Layer:** Now these images are used as an input to a fully connected layer with 128 neurons and the output from the second convolutional layer is reshaped to an array of 30x30x32 =28800 values. The input to this layer is an array of 28800 values. The output of this layer is fed to the 2nd Densely Connected Layer. We are using a dropout layer of value 0.5 to avoid overfitting.

6. **2nd Densely Connected Layer:** Now the output from the 1st Densely Connected Layer is used as an input to a fully connected layer with 96 neurons.

7. **Final layer:** The output of the 2nd Densely Connected Layer serves as an input for the final layer which will have the number of neurons as the number of classes we are classifying (alphabets + blank symbol).

Activation Function:

We have used ReLu (Rectified Linear Unit) in each of the layers(convolutional as well as fully connected neurons). ReLu calculates max(x,0) for each input pixel. This adds nonlinearity to the formula and helps to learn more complicated features. It helps in removing the vanishing gradient problem and speeding up the training by reducing the computation time.

Pooling Layer:

We apply **Max** pooling to the input image with a pool size of (2, 2) with ReLu activation function. This reduces the amount of parameters thus lessening the computation cost and reduces overfitting.

Dropout Layers:

The problem of overfitting, where after training, the weights of the network are so tuned to the training examples they are given that the network doesn't perform well when given new examples. This layer "drops out" a random set of activations in that layer by setting them to zero. The network should be able to provide the right classification or output for a specific example even if some of the activations are dropped out.

Optimizer:

We have used Adam optimizer for updating the model in response to the output of the loss function. Adam combines the advantages of two extensions of two stochastic gradient descent algorithms namely adaptive gradient algorithm (ADA GRAD) and root mean square propagation (RMSProp)

Layer 2:

We are using two layers of algorithms to verify and predict symbols which are more similar to each other so that we can get as close as we can get to detect the symbol shown. In our testing we found that following symbols were not showing properly and were giving other symbols also:

1. ForD:RandU

2. ForU:DandR

3. ForI:T,D,KandI 4. ForS:MandN

So to handle above cases we made three different classifiers for classifying these sets:

1. {D,R,U} 2. {T,K,D,I} 3. {S,M,N}

Finger spelling sentence formation Implementation:

1. Whenever the count of a letter detected exceeds a specific value and no other letter is close to it by a threshold, we print the letter and add it to the current string (In our code we kept the value as 50 and difference threshold as 20).

2. Otherwise we clear the current dictionary which has the count of detections of present symbol to avoid the probability of a wrong letter getting predicted.

3. Whenever the count of a blank (plain background) detected exceeds a specific value and if the current buffer is empty no spaces are detected.

4. In other cases it predicts the end of word by printing a space and the current gets appended to the sentence below.

Autocorrect Feature:

A python library **Hunspell_suggest** is used to suggest correct alternatives for each (incorrect) input word and we display a set of words matching the current word in which the user can select a word to append it to the current sentence. This helps in reducing mistakes committed in spellings and predicting complex words.

Training and Testing:

We convert our input images (RGB) into grayscale and apply gaussian blur to remove unnecessary noise. We apply an adaptive threshold to extract our hand from the background and resize our images to 128 x 128. We feed the input images after preprocessing to our model for training and testing after applying all the operations mentioned above. The prediction layer estimates how likely the image will fall under one of the classes. So the output is normalized between 0 and 1 and such that the sum of each value in each class sums to 1. We have achieved this using the SoftMax function. At first the output of the prediction layer will be somewhat far from the actual value. To make it better we have trained the networks using labeled data. The cross-entropy is a performance measurement used in the classification. It is a continuous function which is positive at values which are not the same as labeled value and is zero exactly when it is equal to the labeled value. Therefore we optimized the cross-entropy by minimizing it as close to zero. To do this in our network layer we adjust the weights of our neural networks. TensorFlow has an inbuilt function to calculate the cross entropy. As we have found out the cross-entropy function, we have optimized it using Gradient Descent. In fact, the best gradient descent optimizer is called Adam Optimizer.

CHAPTER 3

PROPOSED ARCHITECTURE

The proposed architecture presents a model which captures images from the web camera, the image frame through which the desired input has to be taken can be manipulated from the software. The Image is then processed in the OpenCV platform to recognize the gestures through image processing algorithms. Spell checker package is included to correct the spellings in case of some erroneous inputs. The text is then passed to the gTTS library that converts the opened text into audio file.

Block Diagram:

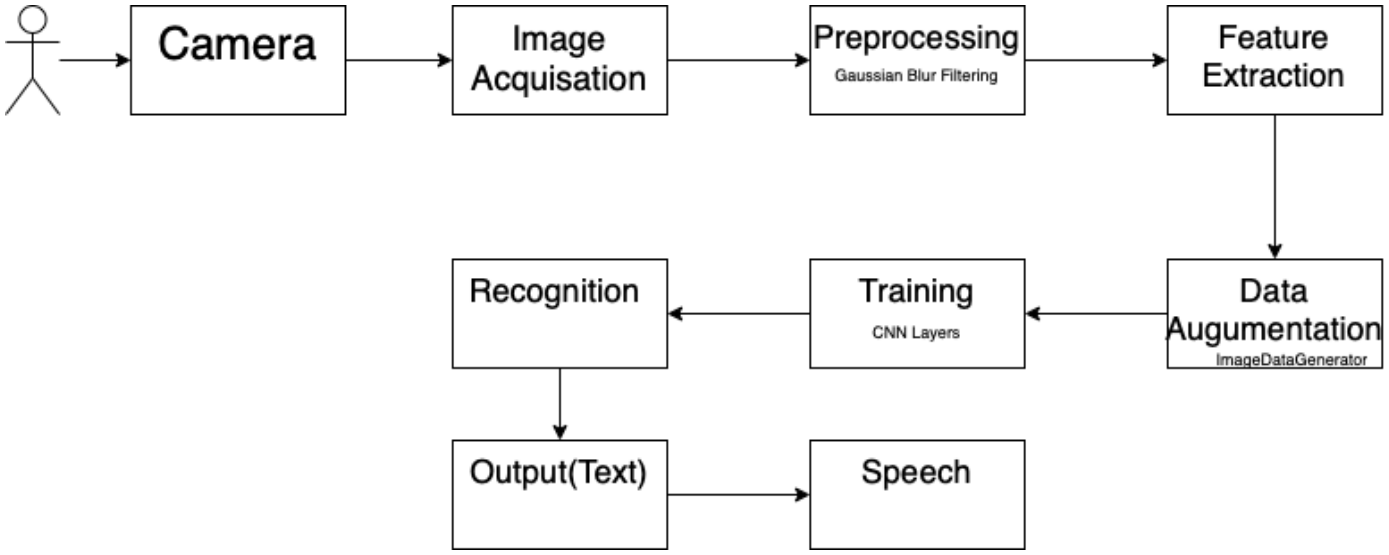


Fig 3.1: Block Diagram of Dactylology Recognition System

The user gives input from the video capture with the help of the in-built camera on the computer. This camera helps in capturing the required images for each alphabet. This involves the image acquisition phase where the input from the user is taken. These images acquired are then preprocessed. The preprocessing involves converting the images into gaussian blur images.

In image processing, a Gaussian blur (also known as Gaussian smoothing) is the result of blurring an image by a Gaussian function (named after mathematician and scientist Carl Friedrich Gauss). It is a widely used effect in graphics software, typically to reduce image noise and reduce detail. The visual effect of this blurring technique is a smooth blur resembling that of viewing the image through a translucent screen, distinctly different from the bokeh effect produced by an out-of-focus lens or the shadow of an object under usual illumination. Gaussian smoothing is also used as a pre-processing stage in computer vision algorithms in order to enhance image structures at different scales—see scale space representation and scale space implementation.

Gaussian blurring is highly effective in removing Gaussian noise from an image. Smoothens the image & reduces noise. Each pixel in an image gets multiplied by a Gaussian Kernel. The next step is the feature extraction. Feature Extraction aims to reduce the number of features in a dataset by creating new features from the existing ones (and then discarding the original features). These new reduced set of features should then be able to summarize most of the information contained in the original set of features. In this way, a summarized version of the original features can be created from a combination of the original set.

After extracting the important features from the acquired images, we have to perform data augmentation, this is done to create more images with differences in each. Data Augmentation performs 2D Transformations like translation, scaling, shearing for generalization of the model. Applying a (small) amount of the transformations to an input image will change its appearance slightly, but it does not change the class label — thereby making data augmentation a very natural, easy method to apply for computer vision tasks. Image data augmentation is a technique that can be used to artificially expand the size of a training dataset by creating modified versions of images in the dataset.
Training deep learning neural network models on more data can result in more skillful models, and the augmentation techniques can create variations of the images that can improve the ability of the fit models to generalize what they have learned to new images.

The Keras deep learning neural network library provides the capability to fit models using image data augmentation via the ImageDataGenerator class. The next step involves training the images. The architecture used for training can be shown below.

After training, the models are saved into. JSON and. Hadoop File(.h5) files with all the saved weights. These models are then used by creating a front end for the project. A Tkinter window helps in the creation of a graphical user interface in python. Python offers multiple options for developing GUI (Graphical User Interface). Out of all the GUI methods, tkinter is the most commonly used method. It is a standard Python interface to the Tk GUI toolkit shipped with Python. Python with tkinter is the fastest and easiest way to create GUI applications. Creating a GUI using tkinter is an easy task.

To create a tkinter app**:**

1. Importing the module – tkinter
2. Create the main window (container)
3. Add any number of widgets to the main window
4. Apply the event Trigger on the widgets.

The video runs on the Tkinter window, and the Hunspell library is used to help in spell checking, to correct the input sentences. Finally, the recognized text is converted to speech. This is done using the gTTS library. There are several APIs available to convert text to speech in Python. One of such APIs is the Google Text to Speech API commonly known as the gTTS API. gTTS is a very easy to use tool which converts the text entered into audio which can be saved as a mp3 file. The gTTS API supports several languages including English, Hindi, Tamil, French, German and many more. The speech can be delivered in any one of the two available audio speeds, fast or slow.

Data preprocessing and Feature extraction for vision-based approach:

- The approach for hand detection combines threshold-based color detection with background subtraction. We can use Adaboost face detector to differentiate between faces and hands as both involve similar skin-color.
- We can also extract the necessary image which is to be trained by applying a filter called Gaussian blur. The filter can be easily applied using open computer vision also known as OpenCV and is described in.
- For extracting the necessary image which is to be trained we can use instrumented gloves as mentioned in. This helps reduce computation time for preprocessing and can give us more concise and accurate data compared to applying filters on data received from video extraction.
- We tried doing the hand segmentation of an image using color segmentation techniques but as mentioned in the research paper skin color and tone is highly dependent on the lighting conditions due to which output, we got for the segmentation we tried to do were not so great. Moreover, we have a huge number of symbols to be trained for our project many of which look similar to each other like the gesture for symbol ‘V’ and digit ‘2’, hence we decided that in order to produce better accuracies for our large number of symbols, rather than

Training Architecture:

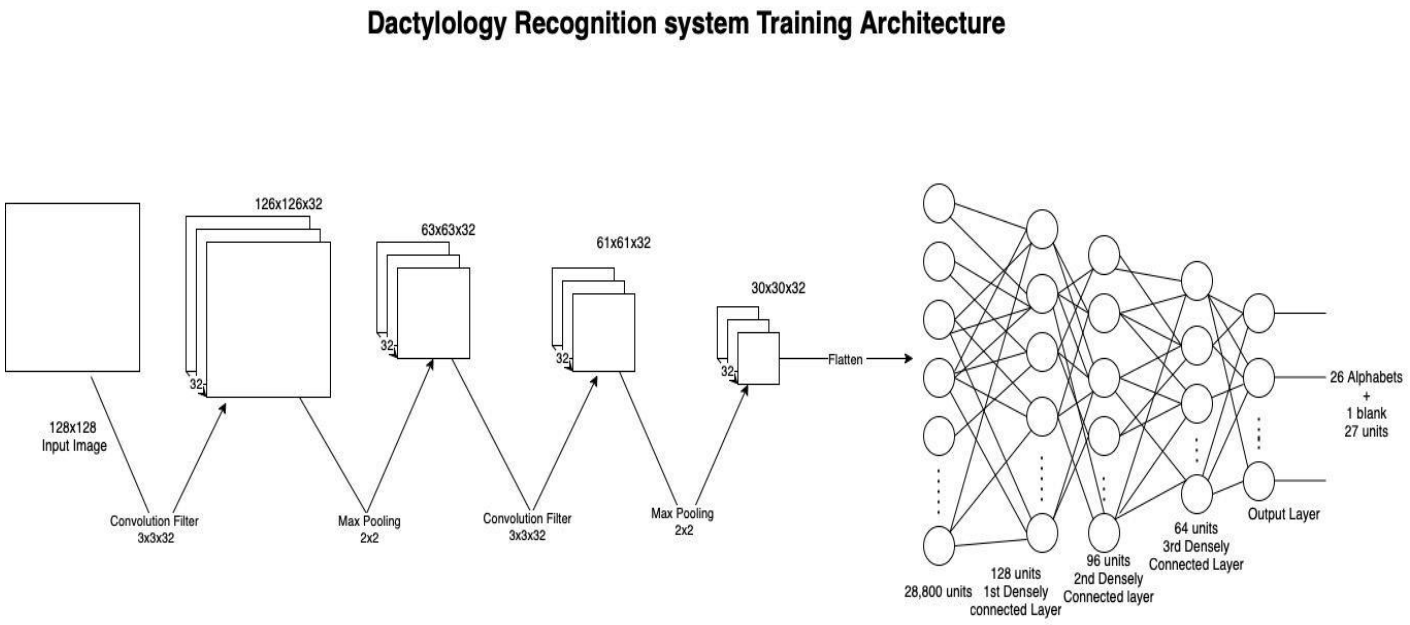


Fig 3.2: Training Architecture

- 1st Convolution Layer -Input 128*128 image, using 32 filters, output 126*126 image.
- 1st Pooling Layer - Downsampled using 2*2 max pooling to 63*63 pixels image
- 2nd Convolution Layer - Input 63*63 image, using 32 filters, output 60*60 images.
- 2nd Pooling Layer - Downsampled using 2*2 max pooling to 30*30 images.
- The outcome of the CNN layers is 28,800 units
- The 1st Fully Connected Dense layer consists of 128 units and this is followed by 2 fully connected layers of 96 units and 64 units respectively.
- Finally, there is the output layer which consists of 27 units, one for each alphabet and last one neuron for the blank space.

CHAPTER 4
IMPLEMENTATION

4.1 Data Implementation

4.1.1 Data Collection

We created a dataset which has 33 gestures and these gestures are made up of 26 alphabets, 7 numeric digits from 0-6. There are 100 images for each member in the 7 different folders Also there are 1500 images for each letter out of which 1480 are for training and the rest of the images are for testing

4.1.2 Preprocessing

In pre-processing the image data is improved which reduces unwanted deviation or enhances image features for further processing. Preprocessing is also referred to as an attempt to capture the important pattern which expresses the uniqueness in data without unwanted data which includes cropping, resizing and gray scaling. Cropping removes the unwanted parts of an image to improve framing.

4.1.3 Creating Gestures

In the dataset that we created all the 33 gestures are stored in numerical order and alphabetical order like 0 has digit 0, I have digit 1...6 has digit 5, a has alphabet A, b has B, z has Z.

4.1.4 Designing the CNN (Convolution Neural Networks)

This step is the most important part of the entire process as we design the CNN through which we will pass our features to train the model and eventually test it using the test features. We have used a combination of several different functions foleonstruct CNN which we will discuss one by one. Building and training a CNN in Keras:

Data Collection -

Data is collected in the form of images which are used to train the neural network. It consists of different numbers to test and train from 0-5. Folders for these numbers are created. Oper used to capture the images and save them to the corresponding folder.

Training -

Import the Keras libraries and packages Build the CNN, convolution layer and pooling, Preparing the train/test data and training the model.

Predict -

Loading the model, Drawing the ROI, Resizing the ROI so it can be fed to the model for prediction, Sorting based on top prediction, Displaying the predictions

• Sequential() - A sequential model is just a linear stack of layers which is putting layers on top of each other as we progress from the input layer to the output layer.

• classifier.add(Conv2D()) - This is a 2D Convolutional layer which performs the convolution operation as described at the beginning of this post. To quote Keras Documentation "This layer creates a convolution kernel that is convolved with the layer input to produce a tensor of outputs." Here we are using a 3x3 kernel size and Rectified Linear Unit (ReLU) as our activation function.

• classifier.add(MaxPooling2D()) This function performs the pooling operation on the data as explained in Literature Review. We are taking a pooling window of 2x2 with 2x2 strides in this model.

• classifier.add(Flatten())- This just flattens the input from ND to ID and does not affect the batch size.

• classifier.add(Dense())- According to Keras Documentation, Dense implements the operation: output= activation(dot(input, kernel)where activation is the element-wise activation function passed as the activation argument, kernel is a weights matrix created by the layer. In simple words, it is the final nail in the coffin which uses the features learned using the layers and maps it to the label. During testing, this layer is responsible for creating the final label for the image being processed.

4.1.5 Recognition of Gestures

• To capture a video, a VideoCapture object is needed to be created.
• cam.read() is the most convenient method for reading video files or capturing
• cv2.flip(flips the image with OpenCV. cv2.resize() resizes the image with OpenCV in dimensions.

If a pixel value is greater than a threshold value, it is assigned one value (may be white), else it is assigned another value (may be black). The function used is ev2.threshold Thresholding is a technique in OpenCV, which is the assignment of pixel values in relation to the threshold value provided. In Thresholding, each pixel value is compared with the threshold value. If the pixel value is smaller than the threshold, it is set to 0, otherwise, it is set to a maximum value (generally 255). It is used for separating an object considered as a foreground from its background. If the pixel value is greater than a threshold value, it is assigned one value (may be white), else it is assigned another value (may be black). The function cv2.threshold is used.

4.2 Algorithm

Our approach uses two layers of algorithms to predict the final symbol of the user.

Algorithm Layer 1: Apply Gaussian blur filter to the frame to get the processed image after feature extraction. This processed image is passed to the CNN model for and if a letter is detected for more than 50 frames then the letter is printed and considered for forming the word. Space between the words is taken as blank symbol.

Algorithm Layer 2: The symbols which look alike of each other, special classifiers will be applied to them depending on the accuracy obtained.

CNN Layers in Algorithm:

- 1st Convolution Layer -Input 128*128 image, using 32 filters, output 126*126 image.
- 1st Pooling Layer - Downsampled using 2*2 max pooling to 63*63 pixels image
- 2nd Convolution Layer - Input 63*63 image, using 32 filters, output 60*60 images.
- 2nd Pooling Layer - Downsampled using 2*2 max pooling to 30*30 images.
- 2 Densely Connected Layers, followed by a final layer with 27 neurons (alphabets + blank symbol)

4.3 Code Implementation

Data Collection -

```
            }

        # Printing the count in each set to the screen
        cv2.putText(frame, "ZERO : "+str(count['zero']), (10, 70), cv2.FONT_HERSHEY_PLAIN, 1, (0,255,255), 1)
        cv2.putText(frame, "a : "+str(count['a']), (10, 100), cv2.FONT_HERSHEY_PLAIN, 1, (0,255,255), 1)
        cv2.putText(frame, "b : "+str(count['b']), (10, 110), cv2.FONT_HERSHEY_PLAIN, 1, (0,255,255), 1)
        cv2.putText(frame, "c : "+str(count['c']), (10, 120), cv2.FONT_HERSHEY_PLAIN, 1, (0,255,255), 1)
        cv2.putText(frame, "d : "+str(count['d']), (10, 130), cv2.FONT_HERSHEY_PLAIN, 1, (0,255,255), 1)
        cv2.putText(frame, "e : "+str(count['e']), (10, 140), cv2.FONT_HERSHEY_PLAIN, 1, (0,255,255), 1)
        cv2.putText(frame, "f : "+str(count['f']), (10, 150), cv2.FONT_HERSHEY_PLAIN, 1, (0,255,255), 1)
        cv2.putText(frame, "g : "+str(count['g']), (10, 160), cv2.FONT_HERSHEY_PLAIN, 1, (0,255,255), 1)
        cv2.putText(frame, "h : "+str(count['h']), (10, 170), cv2.FONT_HERSHEY_PLAIN, 1, (0,255,255), 1)
        cv2.putText(frame, "i : "+str(count['i']), (10, 180), cv2.FONT_HERSHEY_PLAIN, 1, (0,255,255), 1)
        cv2.putText(frame, "j : "+str(count['j']), (10, 180), cv2.FONT_HERSHEY_PLAIN, 1, (0,255,255), 1)
        cv2.putText(frame, "k : "+str(count['k']), (10, 190), cv2.FONT_HERSHEY_PLAIN, 1, (0,255,255), 1)
        cv2.putText(frame, "l : "+str(count['l']), (10, 200), cv2.FONT_HERSHEY_PLAIN, 1, (0,255,255), 1)
        cv2.putText(frame, "m : "+str(count['m']), (10, 210), cv2.FONT_HERSHEY_PLAIN, 1, (0,255,255), 1)
        cv2.putText(frame, "n : "+str(count['n']), (10, 220), cv2.FONT_HERSHEY_PLAIN, 1, (0,255,255), 1)
        cv2.putText(frame, "o : "+str(count['o']), (10, 230), cv2.FONT_HERSHEY_PLAIN, 1, (0,255,255), 1)
        cv2.putText(frame, "p : "+str(count['p']), (10, 240), cv2.FONT_HERSHEY_PLAIN, 1, (0,255,255), 1)
        cv2.putText(frame, "q : "+str(count['q']), (10, 250), cv2.FONT_HERSHEY_PLAIN, 1, (0,255,255), 1)
        cv2.putText(frame, "r : "+str(count['r']), (10, 260), cv2.FONT_HERSHEY_PLAIN, 1, (0,255,255), 1)
        cv2.putText(frame, "s : "+str(count['s']), (10, 270), cv2.FONT_HERSHEY_PLAIN, 1, (0,255,255), 1)
        cv2.putText(frame, "t : "+str(count['t']), (10, 280), cv2.FONT_HERSHEY_PLAIN, 1, (0,255,255), 1)
        cv2.putText(frame, "u : "+str(count['u']), (10, 290), cv2.FONT_HERSHEY_PLAIN, 1, (0,255,255), 1)
        cv2.putText(frame, "v : "+str(count['v']), (10, 300), cv2.FONT_HERSHEY_PLAIN, 1, (0,255,255), 1)
        cv2.putText(frame, "w : "+str(count['w']), (10, 310), cv2.FONT_HERSHEY_PLAIN, 1, (0,255,255), 1)
        cv2.putText(frame, "x : "+str(count['x']), (10, 320), cv2.FONT_HERSHEY_PLAIN, 1, (0,255,255), 1)
        cv2.putText(frame, "y : "+str(count['y']), (10, 330), cv2.FONT_HERSHEY_PLAIN, 1, (0,255,255), 1)
        cv2.putText(frame, "z : "+str(count['z']), (10, 340), cv2.FONT_HERSHEY_PLAIN, 1, (0,255,255), 1)
        # Coordinates of the ROI - Rectangular Region of interest
        x1 = int(0.1*frame.shape[1])
        y1 = 10
```

Fig 4.1: Data Collection Code

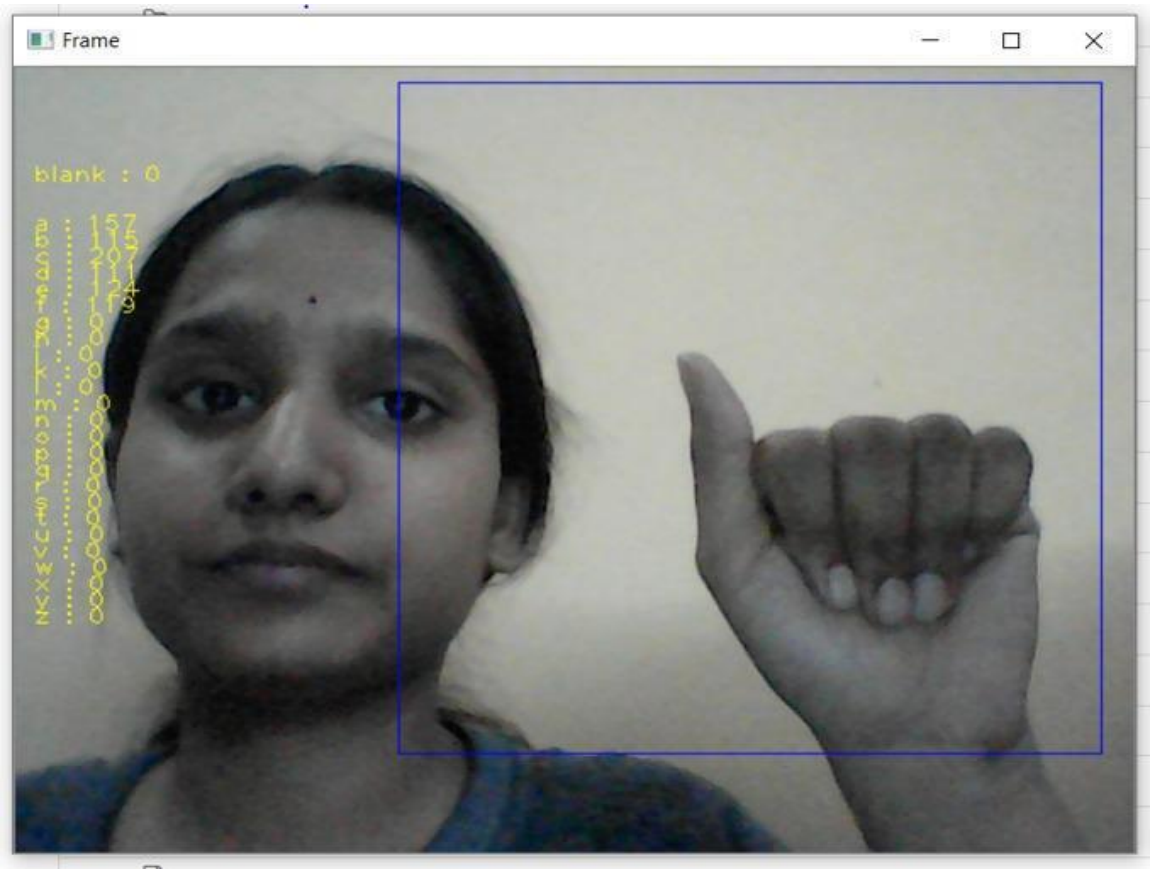


Fig 4.2: Data Collection Output (1)

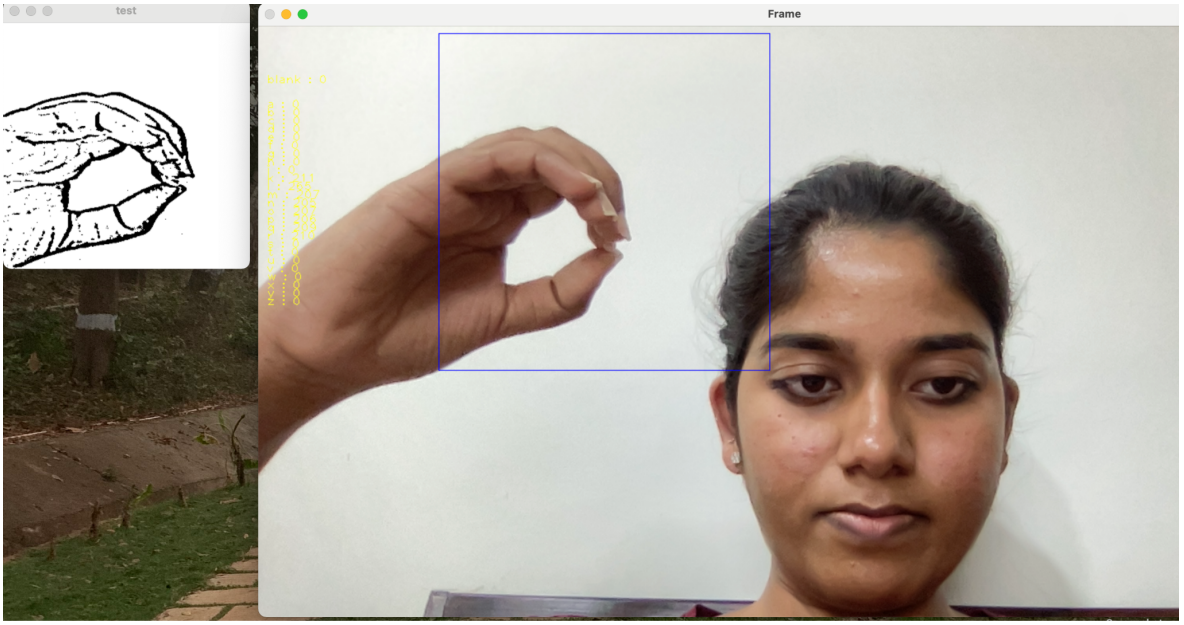


Fig 4.3: Data Collection Output(2)

Code explanation:

- This file consists of code that is used to capture images for all the alphabets from A to Z including blank space. This image capturing is done using the open cv video capture function.
- It creates directories for each alphabet and stores the images captured into its respected directories.
- There is also text on the screen showing the no. of images captured for each alphabet for simplicity.

Output:

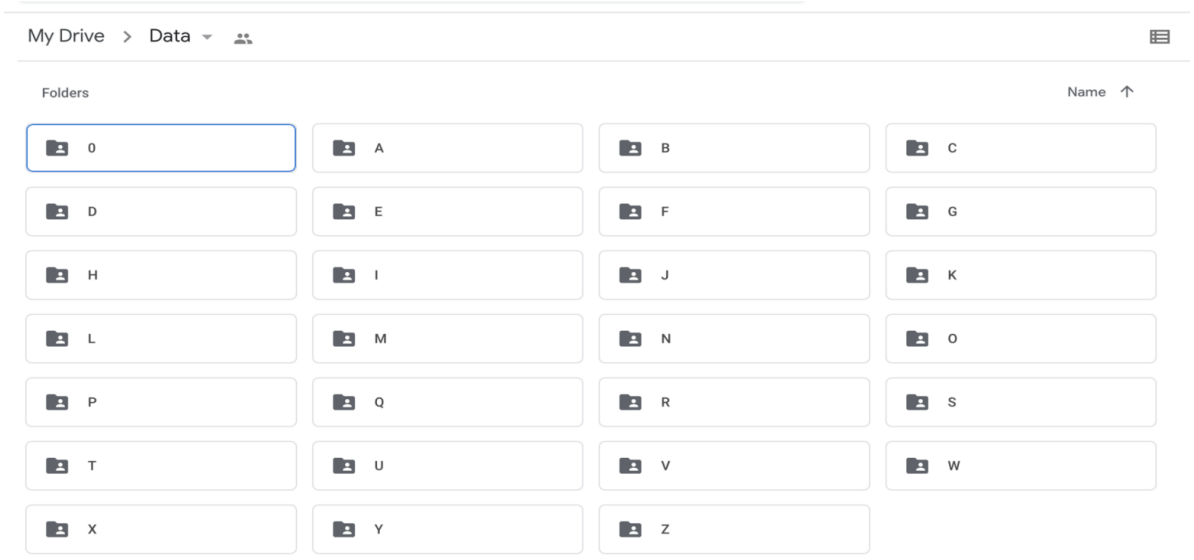
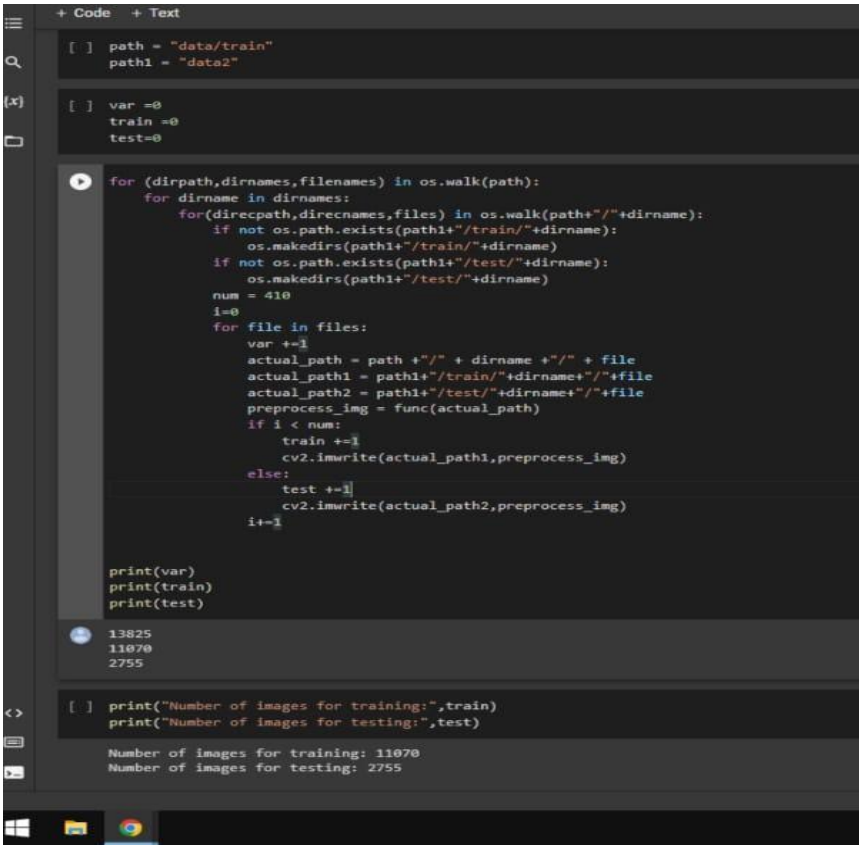


Fig 4.5: Data Collected Stored in directories

Preprocessing -



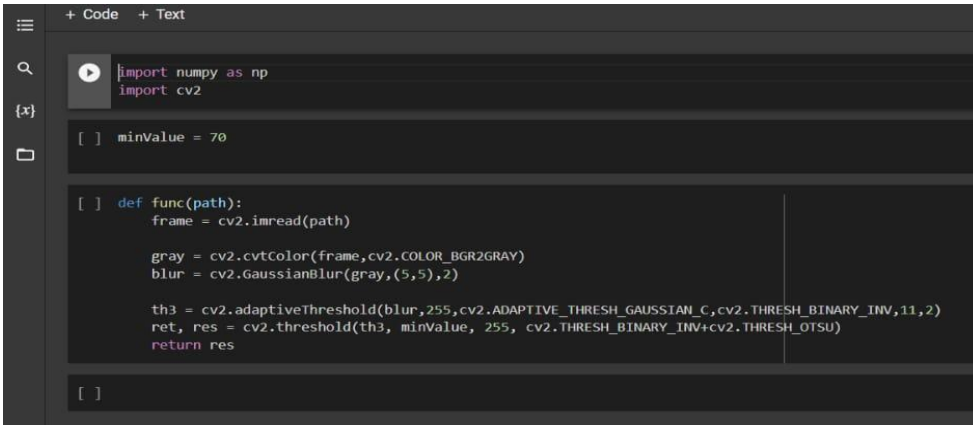
```
+ Code  + Text

[ ]  path = "data/train"
     path1 = "data2"

[ ]  var =0
     train =0
     test=0

for (dirpath,dirnames,filenames) in os.walk(path):
    for dirname in dirnames:
        for(direcpath,direcnames,files) in os.walk(path+"/"+dirname):
            if not os.path.exists(path1+"/train/"+dirname):
                os.makedirs(path1+"/train/"+dirname)
            if not os.path.exists(path1+"/test/"+dirname):
                os.makedirs(path1+"/test/"+dirname)
            num = 410
            i=0
            for file in files:
                var +=1
                actual_path = path +"/" + dirname +"/" + file
                actual_path1 = path1+"/train/"+dirname+"/"+file
                actual_path2 = path1+"/test/"+dirname+"/"+file
                preprocess_img = func(actual_path)
                if i < num:
                    train +=1
                    cv2.imwrite(actual_path1,preprocess_img)
                else:
                    test +=1
                    cv2.imwrite(actual_path2,preprocess_img)
                i+=1


print(var)
print(train)
print(test)

13825
11070
2755

[ ]  print("Number of images for training:",train)
     print("Number of images for testing:",test)

Number of images for training: 11070
Number of images for testing: 2755
```

Fig 4.6: Preprocessing the data

Code explanation:

This file walks through images captured in collect_data.py and splits the images into train and test categories with 80% images for training and 20% for testing

Image Processing -



```
+ Code  + Text

import numpy as np
import cv2

[ ]  minValue = 70

[ ]  def func(path):
         frame = cv2.imread(path)

         gray = cv2.cvtColor(frame,cv2.COLOR_BGR2GRAY)
         blur = cv2.GaussianBlur(gray,(5,5),2)

         th3 = cv2.adaptiveThreshold(blur,255,cv2.ADAPTIVE_THRESH_GAUSSIAN_C,cv2.THRESH_BINARY_INV,11,2)
         ret, res = cv2.threshold(th3, minValue, 255, cv2.THRESH_BINARY_INV+cv2.THRESH_OTSU)
         return res

[ ]
```

Fig 4.7: Image Preprocessing

Code explanation:

- Gaussian blurring is highly effective in removing Gaussian noise from an image. Smoothens the image & reduces noise.
- Collected images are converted to grayscale and then a gaussian blur filter is applied on it. Gaussian blurring is highly effective in removing noise from an image. It smoothens the image. Each pixel in an image gets multiplied by a Gaussian Kernel.
- Then a simple threshold is applied on the image. Thresholding is a popular segmentation technique used for separating an object considered as foreground from its background.
- Adaptive Thresholding is also applied to deal with the changing lighting for different regions.

Output:

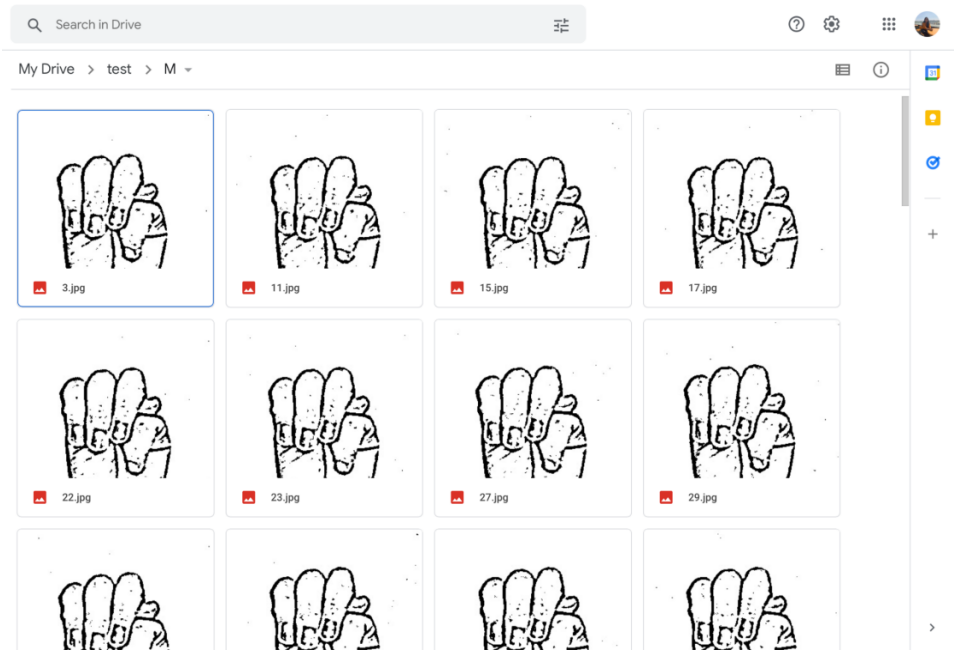


Fig 4.8: Data after preprocessing

Training -

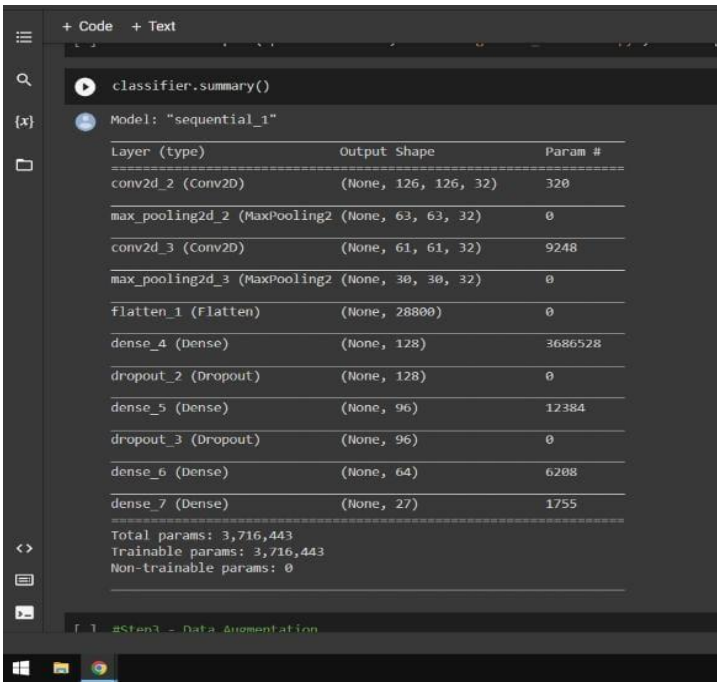


Fig 4.9: CNN Model Summary

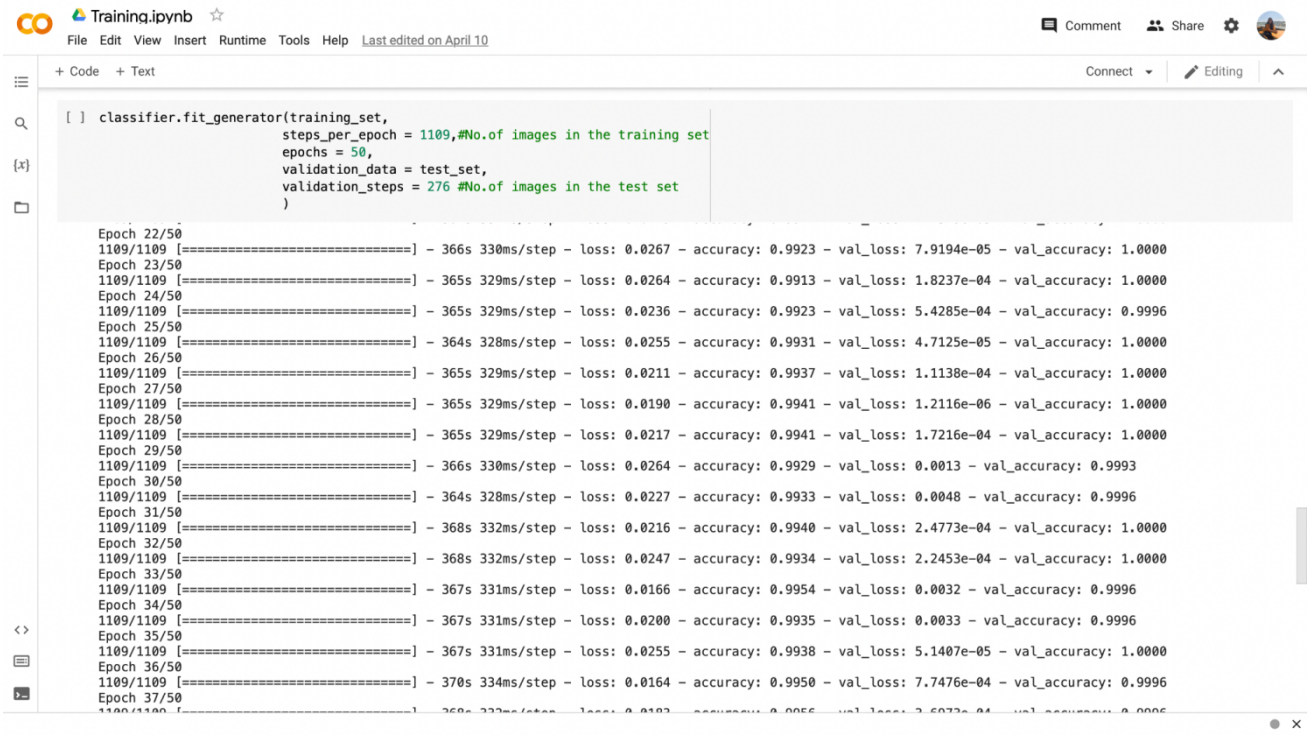


Fig 4.10: CNN Model Training

Code explanation:

- Import all the required Libraries and Packages including TensorFlow.
- Have a look at random images from the Dataset.
- Look at the number of Images in Training and Testing Datasets.
- Create a Neural Network using 2 Convolutional Layers and 3 Fully Connected dense Layers.
- Initializing CNN.
- In 1st Convolution Layer, there are 32 (3,3) filters with "same" Padding and Shape of the Input_Image is (128,128,1)
- Normalizing to speed up learning.
- Applying nonlinear Activation Function "ReLu"
- Adding a Max Pool Layer of size (2,2)
- In 2nd Convolution layer, there are 32 (3,3) filters with "same" Padding
- Normalizing to speed up learning.
- Applying nonlinear Activation Function "ReLu".
- Adding a Max Pool Layer of size (2,2).
- Flattening
- Fully connected layer with 128 neurons.
- Normalizing to speed up learning.
- Applying nonlinear Activation Function "ReLu"
- Dropout layer with 0.40 fraction of the input units to drop
- Fully connected layer with 96 neurons.
- Normalizing to speed up learning.
- Applying nonlinear Activation Function "|ReLu"
- Dropout layer with 0.40 fraction of the input units to drop
- Fully connected layer with 64 neurons.
- Normalizing to speed up learning.
- Applying non linear Activation Function "ReLu".
- Adding a final Dense Layer with 27 outputs corresponding to 7 different emotions with a "SoftMax" Activation Function.
- Use Adam Optimizer
- Compile defines the loss function categorical_crossentropy, the optimizer (Adam) and the metrics(accuracy).
- Perform Data Augmentation - performs 2D Transformations like translation, scaling, shearing for generalization of the model.

- ImageDataGenerator () is used for data augmentation.
- Let us train the Model for 50 epochs using fit_generator() with 1109 steps per each epoch
- Converting the model into JSON format and storing it in “model-bw.json" file and save it’s Hadoop file(.h5).
- Training for layer 2
- Repeating the same for D,R,U letters and save it in “model-bw_dru.json” file and save it’s Hadoop file(.h5)
- Performing training for S,M,N letters and save it in “model-bw_smn.json” file and save it’s .h5 file.
- Similarly. Perform training for T,K,D,I and save it in “model-bw_tkdi.json” file and save it’s .h5 file.

App.py -

```
from PIL import Image,ImageTk
import tkinter as tk
import cv2
import os
import numpy as np
from gtts import gTTS
from tkinter import Button
pip install gTTs

from tensorflow.keras.models import model_from_json

import operator
import time
from string import ascii_uppercase
from hunspell import Hunspell

class Application:
    def _init_(self):
        self.directory = 'model/'
        self.hs = Hunspell('en_US')
        self.vs = cv2.VideoCapture(0)
        self.current_image = None
        self.current_image2 = None
        self.json_file = open(self.directory+"model-bw.json", "r")
```

```
self.model_json = self.json_file.read()
self.json_file.close()
self.loaded_model = model_from_json(self.model_json)
self.loaded_model.load_weights(self.directory+"model-bw.h5")

self.json_file_dru = open(self.directory+"model-bw_dru.json" , "r")
self.model_json_dru = self.json_file_dru.read()
self.json_file_dru.close()
self.loaded_model_dru = model_from_json(self.model_json_dru)
self.loaded_model_dru.load_weights(self.directory+"model-bw_dru.h5")

self.json_file_tkdi = open(self.directory+"model-bw_tkdi.json" , "r")
self.model_json_tkdi = self.json_file_tkdi.read()
self.json_file_tkdi.close()
self.loaded_model_tkdi = model_from_json(self.model_json_tkdi)
self.loaded_model_tkdi.load_weights(self.directory+"model-bw_tkdi.h5")

self.json_file_smn = open(self.directory+"model-bw_smn.json" , "r")
self.model_json_smn = self.json_file_smn.read()
self.json_file_smn.close()
self.loaded_model_smn = model_from_json(self.model_json_smn)
self.loaded_model_smn.load_weights(self.directory+"model-bw_smn.h5")

self.ct = {}
self.ct['blank'] = 0
self.blank_flag = 0
    for i in ascii_uppercase:
    self.ct[i] = 0
print("Loaded model from disk")
self.root = tk.Tk()
self.root.title("Dactylology Recognition System")
self.root.protocol('WM_DELETE_WINDOW', self.destructor)
self.root.geometry("900x1100")
self.panel = tk.Label(self.root)
self.panel.place(x = 135, y = 10, width = 640, height = 640)
```

```
    self.panel2 = tk.Label(self.root) # initialize image panel
    self.panel2.place(x = 460, y = 95, width = 310, height = 310)

    self.T = tk.Label(self.root)
    self.T.place(x=250,y = 17)
    self.T.config(text = "Dactylology RecognitionSystem",font=("courier",40,"bold"))
    self.panel3 = tk.Label(self.root) # Current SYmbol
    self.panel3.place(x = 500,y=600)
    self.T1 = tk.Label(self.root)
    self.T1.place(x = 10,y = 600)
    self.T1.config(text="Character :",font=("Courier",40,"bold"))
    self.panel4 = tk.Label(self.root) # Word
    self.panel4.place(x = 220,y=650)
    self.T2 = tk.Label(self.root)
    self.T2.place(x = 10,y = 650)
    self.T2.config(text ="Word :",font=("Courier",40,"bold"))
    self.panel5 = tk.Label(self.root) # Sentence
    self.panel5.place(x = 350,y=700)
    self.T3 = tk.Label(self.root)
    self.T3.place(x = 10,y = 700)
    self.T3.config(text ="Sentence :",font=("Courier",40,"bold")
    self.str=""
    self.word=""
    self.current_symbol="Empty"
    self.photo="Empty"
    self.video_loop()
def video_loop(self):
    ok, frame = self.vs.read()
    if ok:
        cv2image = cv2.flip(frame, 1)
        x1 = int(0.5*frame.shape[1])
        y1 = 10
        x2 = frame.shape[1]-10
        y2 = int(0.5*frame.shape[1])
        cv2.rectangle(frame, (x1-1, y1-1), (x2+1, y2+1), (255,0,0) ,1)
```

```
            cv2image = cv2.cvtColor(cv2image, cv2.COLOR_BGR2RGBA)
        self.current_image = Image.fromarray(cv2image)
        imgtk = ImageTk.PhotoImage(image=self.current_image)
        self.panel.imgtk = imgtk
        self.panel.config(image=imgtk)
        cv2image = cv2image[y1:y2, x1:x2]
        gray = cv2.cvtColor(cv2image, cv2.COLOR_BGR2GRAY)
        blur = cv2.GaussianBlur(gray,(5,5),2)
        th3 =
cv2.adaptiveThreshold(blur,255,cv2.ADAPTIVE_THRESH_GAUSSIAN_C,cv2.THRESH_BINAR
Y_INV,11,2)
        ret, res = cv2.threshold(th3, 70, 255, cv2.THRESH_BINARY_INV+cv2.THRESH_OTSU)
        self.predict(res)
        self.current_image2 = Image.fromarray(res)
        imgtk = ImageTk.PhotoImage(image=self.current_image2)
        self.panel2.imgtk = imgtk
        self.panel2.config(image=imgtk)
        self.panel3.config(text=self.current_symbol,font=("Courier",50))
        self.panel4.config(text=self.word,font=("Courier",40))
        self.panel5.config(text=self.str,font=("Courier",40))
        predicts=self.hs.suggest(self.word)
        self.root.after(30, self.video_loop)

    def predict(self,test_image):
        test_image = cv2.resize(test_image, (128,128))
        result = self.loaded_model.predict(test_image.reshape(1, 128, 128, 1))
        result_dru = self.loaded_model_dru.predict(test_image.reshape(1 , 128 , 128 , 1))
        result_tkdi = self.loaded_model_tkdi.predict(test_image.reshape(1 , 128 , 128 , 1))
        result_smn = self.loaded_model_smn.predict(test_image.reshape(1 , 128 , 128 ,1))
        prediction={}
        prediction['blank'] = result[0][0]
        inde = 1
        for i in ascii_uppercase:
```

```
            prediction[i] = result[0][inde]
            inde += 1
        #LAYER 1
        prediction = sorted(prediction.items(), key=operator.itemgetter(1), reverse=True)
        self.current_symbol = prediction[0][0]
        #LAYER 2
        if(self.current_symbol == 'D' or self.current_symbol == 'R' or self.current_symbol == 'U'):
            prediction = {}
            prediction['D'] = result_dru[0][0]
            prediction['R'] = result_dru[0][1]
            prediction['U'] = result_dru[0][2]
            prediction = sorted(prediction.items(), key=operator.itemgetter(1), reverse=True)
            self.current_symbol = prediction[0][0]

        if(self.current_symbol == 'D' or self.current_symbol == 'I' or self.current_symbol == 'K' or
self.current_symbol == 'T'):
            prediction = {}
            prediction['D'] = result_tkdi[0][0]
            prediction['I'] = result_tkdi[0][1]
            prediction['K'] = result_tkdi[0][2]
            prediction['T'] = result_tkdi[0][3]
            prediction = sorted(prediction.items(), key=operator.itemgetter(1), reverse=True)
            self.current_symbol = prediction[0][0]
if(self.current_symbol == 'M' or self.current_symbol == 'N' or self.current_symbol == 'S'):
            prediction1 = {}
            prediction1['M'] = result_smn[0][0]
            prediction1['N'] = result_smn[0][1]
            prediction1['S'] = result_smn[0][2]
            prediction1 = sorted(prediction1.items(), key=operator.itemgetter(1), reverse=True)
            if(prediction1[0][0] == 'S'):
                    self.current_symbol = prediction1[0][0]
            else:
                    self.current_symbol = prediction[0][0]
        if(self.current_symbol == 'blank'):
            for i in ascii_uppercase:
```

```
        self.ct[i] = 0
self.ct[self.current_symbol] += 1
if(self.ct[self.current_symbol] > 60):
    for i in ascii_uppercase:
        if i == self.current_symbol:
            continue
        tmp = self.ct[self.current_symbol] - self.ct[i]
        if tmp < 0:
            tmp *= -1
        if tmp <= 20:
            self.ct['blank'] = 0
            for i in ascii_uppercase:
                self.ct[i] = 0
            return
    self.ct['blank'] = 0
    for i in ascii_uppercase:
        self.ct[i] = 0
    if self.current_symbol == 'blank':
        if self.blank_flag == 0:
            self.blank_flag = 1
            if len(self.str) > 0:
                self.str += " "
            self.str += self.word
            self.word = ""
    else:
        if(len(self.str) > 16):
            self.str = ""
        self.blank_flag = 0
        self.word += self.current_symbol
    btn = Button(self.root,text = "click to convert into speech ",width = "30", pady = 10,font =
    "bold, 15",command = self.play, bg='yellow')
btn.place(x = 1000,y = 400)
```

```
def play(self):
    language ='en'
    myobj=gTTS(text=self.str,lang=language,slow=False)

    myobj.save("convert.wav")
    os.system("convert.wav")

def action1(self):
        predicts=self.hs.suggest(self.word)
        if(len(predicts) > 0):
        self.word=""
        self.str+=" "
        self.str+=predicts[0]
def action2(self):
        predicts=self.hs.suggest(self.word)
        if(len(predicts) > 1):
        self.word=""
        self.str+=" "
        self.str+=predicts[1]
def action3(self):
        predicts=self.hs.suggest(self.word)
        if(len(predicts) > 2):
        self.word=""
        self.str+=" "
        self.str+=predicts[2]
def action4(self):
        predicts=self.hs.suggest(self.word)
        if(len(predicts) > 3):
        self.word=""
        self.str+=" "
        self.str+=predicts[3]
def action5(self):
        predicts=self.hs.suggest(self.word)
        if(len(predicts) > 4):
        self.word=""
```

```
            self.str+=" "
            self.str+=predicts[4]
    def destructor(self):
        print("Closing Application...")
        self.root.destroy()
        self.vs.release()
        cv2.destroyAllWindows()

    def destructor1(self):
        print("Closing Application...")
        self.root1.destroy()

print("Starting Application...")
pba = Application()
pba.root.mainloop()
```

Code explanation:

- This file creates a Tkinter window.
- A video capture is opened which takes input from the user and displays the recognized letter in the form of character. This character is then passed to the word.
- On receiving blank as an input, the obtained word is passed to the sentence.
- On clicking the click to convert into speech button, the sentence is converted to speech.

CHAPTER 5

RESULTS

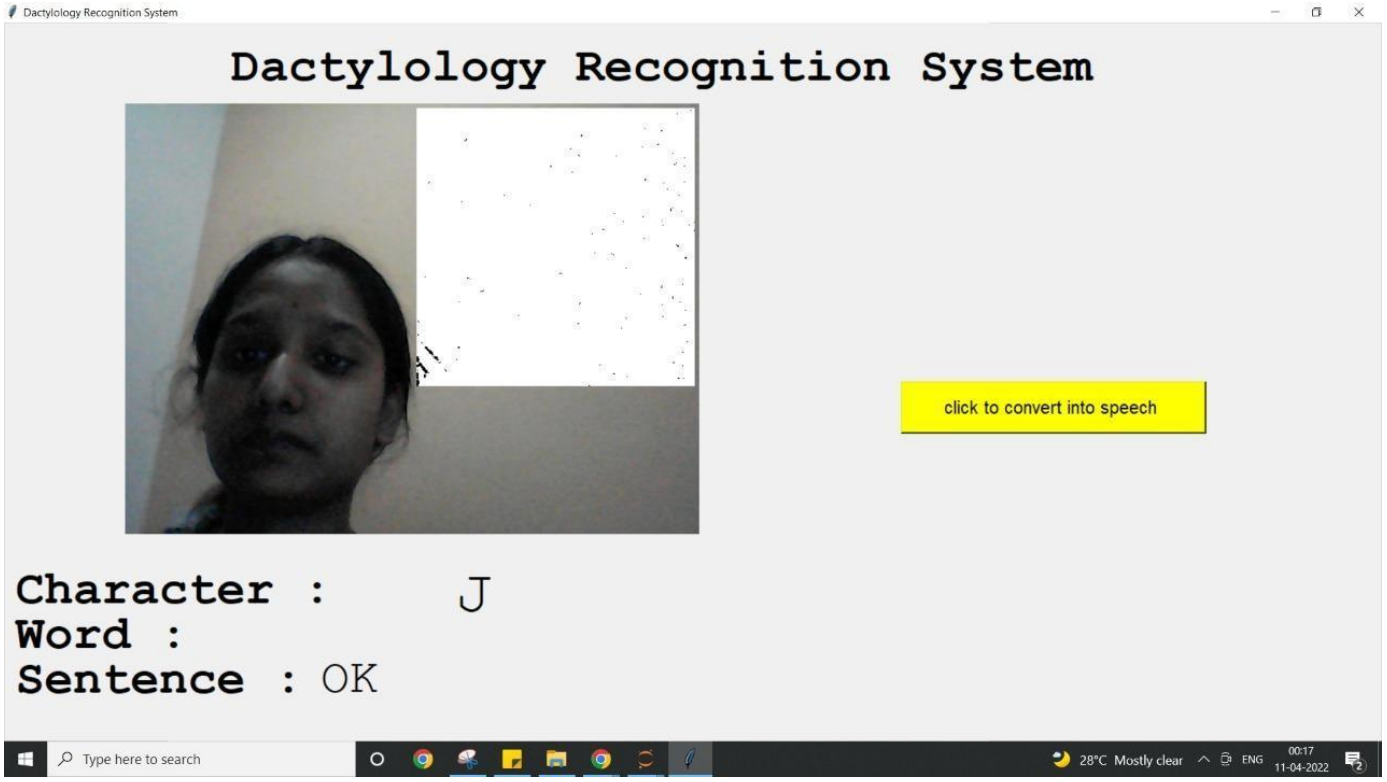


Fig 5.1: Output for the given Sign in the form of sentence

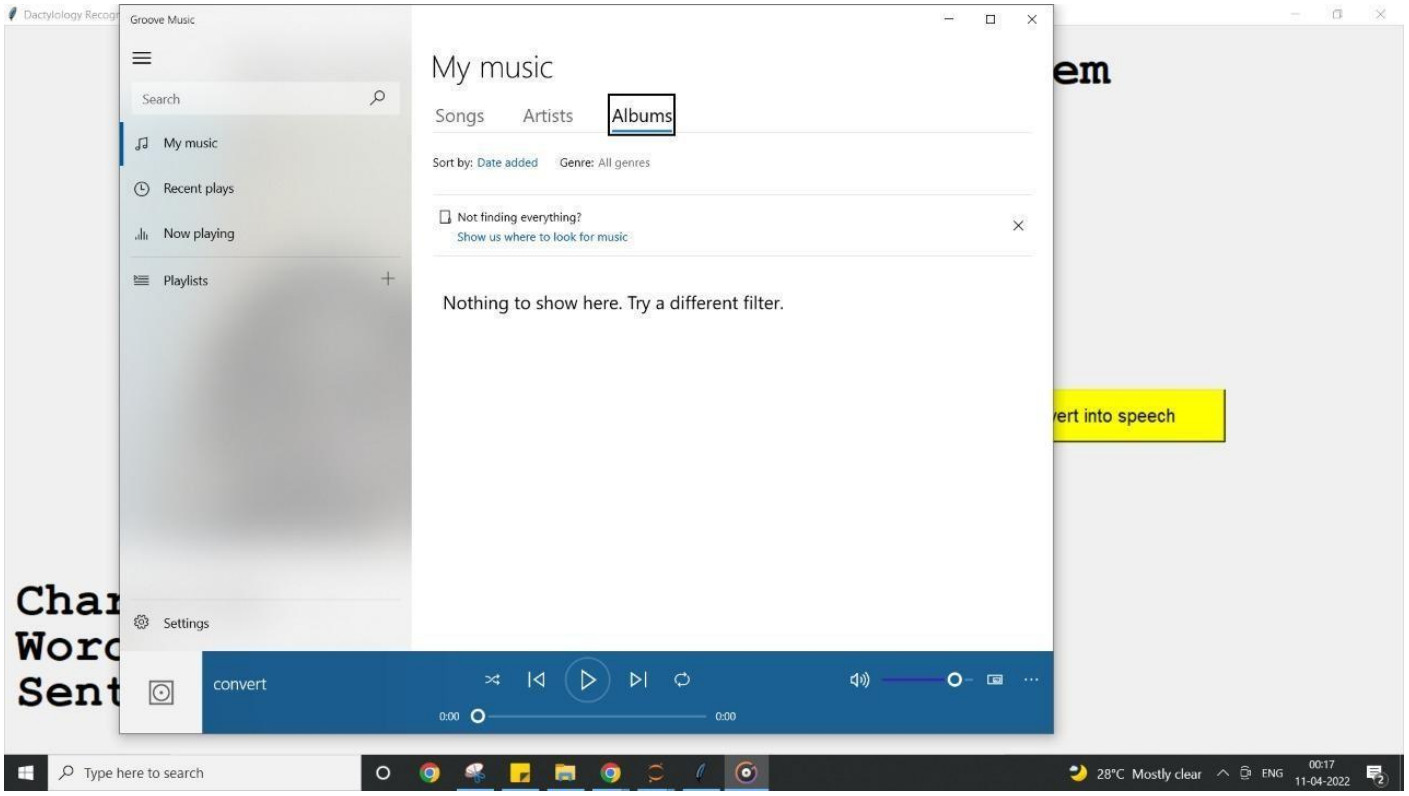


Fig 5.2: Sign Language to Speech Conversion

CHAPTER 6

CONCLUSION

In this project, a functional real time vision based American sign language recognition for Deaf and Dumb people have been developed using ASL alphabets. We achieved final accuracy of 95% on our dataset. We are able to improve our prediction after implementing two layers of algorithms in which we verify and predict symbols which are more similar to each other.

We started from collecting the data by using OpenCV video capturing. This followed preprocessing the data and smoothing the images by using Gaussian Blur Filtering. The cleaned data was then trained using 2 CNN layers and 3 Densely connected layers. Training was done in 2 steps, one for all the alphabets and second step for alphabets looking similar to avoid confusion.

A front-end GUI was then developed using the Tkinter window to capture inputs. The output is a sentence which is converted to audio, on click of a button present on the window. This way we are able to detect almost all the symbols provided that they are shown properly and there is no noise in the background and lighting is adequate.

CHAPTER 7

FUTURE SCOPE

We are planning to achieve higher accuracy even in case of complex backgrounds by trying out various background subtraction algorithms.

We are also thinking of improving the preprocessing to predict text in low light conditions with a higher accuracy. Currently, our project uses manual alphabets where each letter is shown to combine it to a word and then into a sentence, we plan to add gestures in the future which would convey emotions like Thank you, I Love you, Hello etc. We plan to add dactylology recognition to online meetings as well using which deaf and dumb people and communicate, an algorithm can be applied to online meetings where a gesture shown is converted to text and shown to the others in the meeting.

Finally, we would like to improve the time efficiency of our system in order to make it appropriate to use in different applications.

CHAPTER 8

REFERENCES

1. T. Yang, Y. Xu, and “A. , Hidden Markov Model for Gesture Recognition”, CMU-RI-TR-94 10, Robotics Institute, Carnegie Mellon Univ.,Pittsburgh,PA, May 1994.

2. Pujan Ziaie, Thomas M¨uller , Mary Ellen Foster , and Alois Knoll“A Na¨ive Bayes Munich,Dept. of Informatics VI, Robotics and Embedded Systems,Boltzmannstr. 3, DE-85748 Garching, Germany.

3. https://docs.opencv.org/2.4/doc/tutorials/imgproc/gausian_median_blur_b
ilateral_filter/gausian_median_blur_bilateral_filter.html

4. Mohammed Waleed Kalous, Machine recognition of Auslan signs using PowerGloves: Towards large-lexicon recognition of sign language.

5. aeshpande3.github.io/A-Beginner%27s-Guide-To-Understanding-Convol
utional-Neural-Networks-Part-2/

6. http://www-i6.informatik.rwth-aachen.de/~dreuw/database.php

7. Pigou L., Dieleman S., Kindermans PJ., Schrauwen B. (2015) Sign Language Recognition Using Convolutional Neural Networks. In: Agapito L., Bronstein M., Rother C. (eds) Computer Vision - ECCV 2014 Workshops. ECCV 2014. Lecture Notes in Computer Science, vol 8925. Springer, Cham.

8. Zaki, M.M., Shaheen, S.I.: Sign language recognition using a combination of new vision based features. Pattern Recognition Letters 32(4),

9. N. Mukai, N. Harada and Y. Chang, "Japanese Fingerspelling Recognition Based on Classification Tree and Machine Learning," *2017 Nicograph International (NicoInt)*, Kyoto, Japan, 2017, pp. 19-24. doi:10.1109/NICOInt.2017.9

10. Byeongkeun Kang , Subarna Tripathi , Truong Q. Nguyen ”Real-time sign language fingerspelling recognition using convolutional neural networks from depth map” 2015 3rd IAPR Asian Conference on Pattern Recognition (ACPR)

11. https://opencv.org/

12. https://en.wikipedia.org/wiki/TensorFlow

13. https://en.wikipedia.org/wiki/Convolutional_neural_network