A

PROJECT REPORT

ON

“Sign Language to Text Conversion”

Submitted in partial fulfillment of the

Requirements for the award of the degree

of

BACHELOR OF TECHNOLOGY

in

Information Technology

by

ABHAY KUMAR (1711001)

ADARSH KUMAR (1711002)

AMIT ORAON (1711003)

ANIBRATO ADHIKARI (1711004)

RANJAN KUMAR CHOUBEY (1711025)

TANVEER AZAM ANSARI (1711038)

DEPARTMENT OF INFORMATION TECHNOLOGY

B.I.T. SINDRI

SINDRI – 828123 (INDIA)

July, 2021

CANDIDATE'S DECLARATION

I hereby declare that the work carried out in this report titled "**Sign Language to Text Conversion"** is presented on behalf of partial fulfilment of the requirements for the award of the degree of **Bachelor of Technology** with specialization in **Information Technology** submitted to the department of Information Technology**, B.I.T. Sindri, India**, under the supervision and guidance of **Mr**. **Sachin Kumar Agrawal**, Assistant Professor, Information Technology Dept., B.I.T. Sindri, India.

I have not submitted the matter embodied in this report for the award of any other degree or diploma.

Date: -

Place: **Sindri**

Student Name:

ABHAY KUMAR (1711001)

ADARSH KUMAR (1711002)

AMIT ORAON (1711003)

ANIBRATO ADHIKARI (1711004)

RANJAN KUMAR CHOUBEY (1711025)

TANVEER AZAM ANSARI (1711038)

B.I.T. SINDRI

(DEPARTMENT OF SCIENCE & TECHNOLOGY, GOVT. OF JHARKHAND, RANCHI)
P.O. SINDRI INSTITUTE, DHANBAD-828123

DECLARATION CERTIFICATE

This is to certify that the work presented in this project entitled **"Sign Language to Text Conversion "** submitted by ABHAY KUMAR, ADARSH KUMAR, AMIT ORAON, ANIBRATO ADHIKARI, RANJAN KUMAR CHOUBEY, and TANVEER AZAM ANSARI under supervision and guidance of **Mr. Sachin Kumar Agrawal**, Assistant Professor, Dept. of Information Technology, B.I.T. Sindri, India.

To the best of my knowledge, the content of this dissertation does not form a basis of the award of any previous degree to anyone else.

Mr. Sachin Kumar Agrawal,
Assistant Professor,
Dept. of Information Technology

Dr. F Ansari,
Head of Department,
Dept. of Information Technology

B.I.T. SINDRI

CERTIFICATE OF APPROVAL

The project entitled " **Sign Language to Text Conversion** " is here by approved as a creditable study of project topic carried out and has been presented in satisfactory manner to warrant its acceptance as prerequisite to the degree for which it has been submitted.

It is understood that by this approval, the undersigned do not necessarily endorse any conclusion drawn or opinion expressed there in, but approve the project for the purpose for which it is submitted.

Prof (Dr.) Md. F Ansari
Head of Department
Dept. of Information Technology
B.I.T. Sindri, Dhanbad

(External Examiner)

ACKNOWLEDGEMENT

Project work in final year is not only a part of study rather it has provided us to think beyond what we have learnt in the class room. We take this opportunity to acknowledge the efforts of all the people associated with this project, without which this project would not have been a reality.

We would like to pay our sincere thanks to our project guide Mr. Sachin Kumar Agrawal who provided invaluable time and all the technical guidance needed to complete this project. He also gave us all the liberty to do the study related to the project which led to the completion of this project in all regards. He was very supportive in all action and inspired us to think over the problems related to this project.

We are also grateful to our head of department Prof (Dr.) Md. F. Ansari for his invaluable suggestions.

ABSTRACT

Sign language is one of the oldest and most natural form of language for communication. It has been around in our world since ancient times when humans had not yet developed vocal languages. Sign language is a visual language band now mostly used by especially abled individuals who cannot hear or speak to convey their thoughts. But there is always a need of an interpreter to be present when it comes to them speaking with the masses and interpreters are very difficult to come by.

We have come up with a real-time method using neural networks for recognition and display of fingerspelling based American sign language. In this method, the image of the hand is captured frame by frame from live video feed. The image is first passed through a filter and after the filter is applied the image is passed through a classifier which predicts the class of the hand gestures.

CONTENTS:

Chapter	Title	Page no.
Chapter 1	**Introduction**	**1-4**
1.1	Problem Statement	2
1.2	Keywords and Definitions	3
Chapter 2	**Literary Survey**	**5**
Chapter 3	**Methodology**	**6-16**
3.1	Solution Overview	6
3.2	Data Acquisition	7
3.3	Data Pre-processing	8
3.4	Model Creation	9
Chapter 4	**Implementation**	**17-20**
4.1	Training and Testing	17
4.2	User Interface	19
Chapter 5	**Result**	**21**
Chapter 6	**Conclusion**	**22**
References		**23**

LIST OF FIGURES:

Figure No.	Figure Title	Page No.
1.1	American Sign Language	1
1.2	Neural Network	3
3.1	Solution Overview	6
3.2	Data Collection	7
3.3	Collected image Samples	7
3.4	CNN Model Flowchart	10
3.5	Convolutional Neural Network	11
3.6	Dense Layer	13
3.7	Activation Function (ReLu) Graph	15
3.8	Model Summary	16
4.1	Data Collection for {M, N, S, T}	18
4.2	Data Collection for {D, R, U}	18
4.3	Model Training (Initial and final epoch)	19
4.4	User Interface	20
5.1	Model Heatmap	21

CHAPTER 1: INTRODUCTION

American sign language is a predominant sign language Since the only disability Deaf & Mute people have been communication related and they cannot use spoken languages hence the only way for them to communicate is through sign language. Communication is the process of exchange of thoughts and messages in various ways such as speech, signals, behaviour and visuals. Deaf and Mute (D&M) people make use of their hands to express different gestures to express their ideas with other people. Gestures are the nonverbally exchanged messages and these gestures are understood with vision. This nonverbal communication of deaf and dumb people is called sign language.

Sign language is a visual language and consists of 3 major components:

Fingerspelling	**Word level sign vocabulary**	**Non-manual features**
Used to spell words letter by letter	Used for the majority of communication	Facial expressions and tongue, mouth and body position.

In our project we basically focus on producing a model which can recognise Fingerspelling based hand gestures in order to form a complete word by combining each gesture. The gestures we aim to train are as given in the image below:

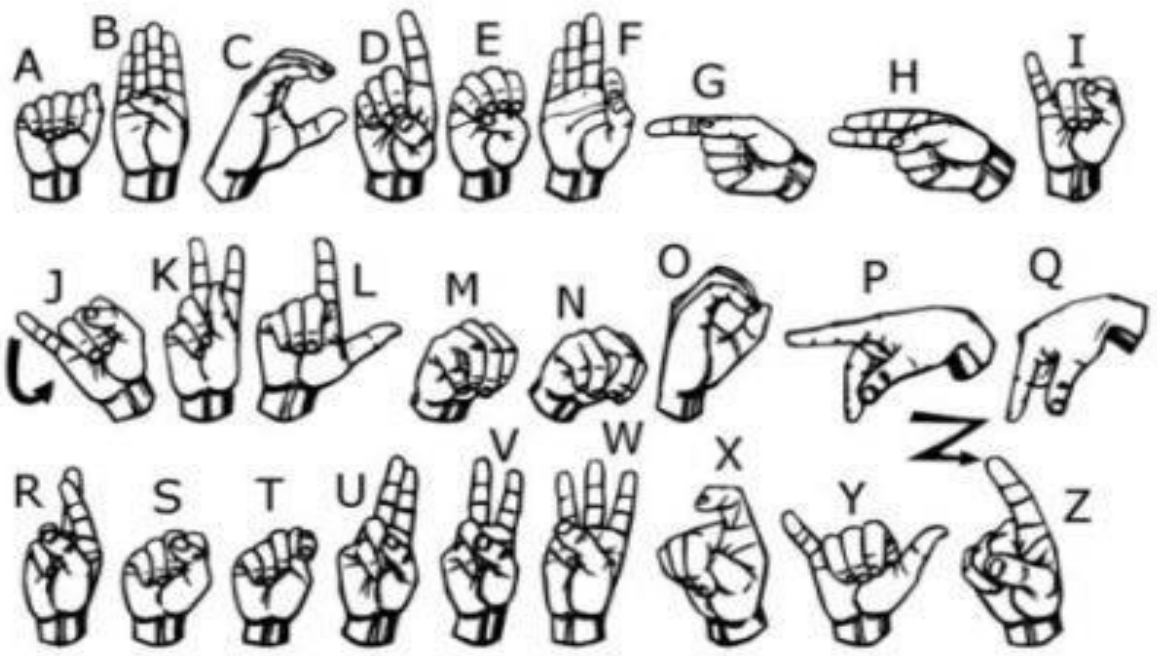

Fig. 1.1: American sign Language

1.1 Problem Statement

For interaction between normal people and the specially-abled Deaf & Mute people a language barrier is created as the sign language structure 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. Also, translators are not available for every person to go on and communicate in their day to day lives, let alone the fact that they are not a cost-efficient alternative also. Similarly, in our ongoing digital era transition that we are going into, humans are getting online to face masses from the security of their homes and this problem not being able to communicate without a third person (translator) can hinder the progress of the specially-abled class of the society. [1] Deaf and dumb people find it difficult as they can't find a well-experienced and educated translator at all the time to recognize convey their messages. The only efficient way through which they can communicate is through sign 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. For the base design of our solution, we are planning to design an ASL to English converted which can be used by the common masses without going through the hassle of high-tech gesture devices.

1.2 Keywords and Definitions

1.2.1 Artificial Neural Networks:

Artificial Neural Network is a connection of neurons, replicating the structure of human brain. Each connection of neuron transfers information to another neuron. Inputs are fed into first layer of neurons which process it and transfers to another layer of neurons called as hidden layers. After processing of information through multiple layers of hidden layers, information is passed to final output layer.

There are capable of learning and they have to be trained. There are different learning strategies:
1. Unsupervised Learning
2. Supervised Learning
3. Reinforcement Learning

1.2.2 Convolution Neural Network:

Unlike regular Neural Networks, in the layers of CNN, the neurons are arranged in 3 dimensions: width, height, depth. The neurons in a layer will only be connected to a small region of the layer (window size) before it, instead of all of the neurons in a fully-connected manner. Moreover, the final output layer would have dimensions (Number of classes), because by the end of the CNN architecture we will reduce the full image into a single vector of class scores.

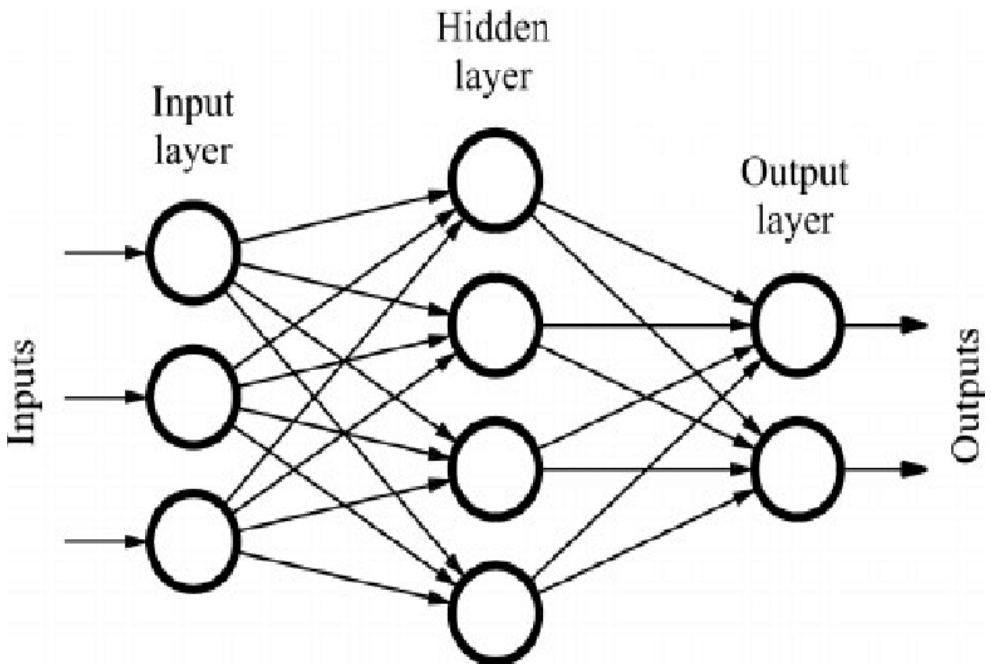

Fig. 1.2: Neural Network

1.2.3 TensorFlow:

TensorFlow is an open-source software library for numerical computation. First, we define the nodes of the computation graph, then inside a session, the actual computation takes place. TensorFlow is widely used in Machine Learning.

1.2.4 Keras:

Keras is a high-level neural networks library written in python that works as a wrapper to TensorFlow. It is used in cases where we want to quickly build and test the neural network with minimal lines of code. It contains implementations of commonly used neural network elements like layers, objective, activation functions, optimizers, and tools to make working with images and text data easier.

1.2.5 OpenCV:

OpenCV (Open-Source Computer Vision) is an open-source library of programming functions used for real-time computer-vision. It is mainly used for image processing, video capture and analysis for features like face and object recognition. It is written in C++ which is its primary interface, however bindings are available for Python, Java, MATLAB/OCTAVE.

1.2.6 Pandas

Pandas is mainly used for data analysis. Pandas allows importing data from various file formats such as comma-separated values, JSON, SQL, Microsoft Excel. Pandas allows various data manipulation operations such as merging, reshaping, selecting, as well as data cleaning, and data wrangling features.

1.2.7 NumPy

NumPy is a Python library used for working with arrays. It also has functions for working in domain of linear algebra, Fourier transform, and matrices. NumPy was created in 2005 by Travis Oliphant. It is an open-source project and you can use it freely.

1.2.8 Tkinter

Tkinter is a Python binding to the Tk GUI toolkit. It is the standard Python interface to the Tk GUI toolkit, and is Python's de facto standard GUI. Tkinter is included with standard Linux, Microsoft Windows and Mac OS X installs of Python.

CHAPTER 2: LITERARY SURVEY

Findings during Literary survey:

In [4], Natural, modern and innovative way of non-verbal communication can be achieved by using a hand gesture recognition system. The main aim of this paper is to discuss about the novel approach of the hand gesture recognition which is based on detecting the features of the shapes. The system setup comprises of a camera which is used to capture the gesture given by the user and take the image formed as the input to the proposed algorithm. The algorithm is divided into four steps, and they are segmentation, orientation detection, feature extraction and classification.

In the recent years there has been tremendous research done on the hand gesture recognition. The basic steps in hand gesture recognition is: -

- Data acquisition
- Data pre-processing
- Feature extraction
- Gesture classification

In [2] The two types of approach for hand gesture recognition are vision-based approach and non-vision-based approach. Process such as segmentation, feature extraction and gesture recognition can be performed. High level segmentation is needed for posture recognition to improve the accuracy rate.

Use of sensory devices

It uses electromechanical devices to provide exact hand configuration, and position. Different glove-based approaches can be used to extract information. But it is expensive and not user friendly.

Vision based approach

In vision-based methods computer camera is the input device for observing the information of hands or fingers. The Vision Based methods require only a camera, thus realizing a natural interaction between humans and computers without the use of any extra devices. These systems tend to complement biological vision by describing 7artificial vision systems that are implemented in software and/or hardware.

The main challenge of vision-based hand detection is to cope with the large variability of human hand's appearance due to a huge number of hand movements, to different skin-colour possibilities as well as to the variations in viewpoints, scales, and speed of the camera capturing the scene.

CHAPTER 3: METHODOLOGY

3.1 Solution Overview

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.

Step 1: Data Acquisition

Step 2: Data Pre-processing

Step 3: Passing the data to layer 1 CNN model.

Step 4: Applying a filter to layer 1 output

Step 5: Passing the data to respective layer 2 model (if filter triggered).

Step 6: Printing the Output

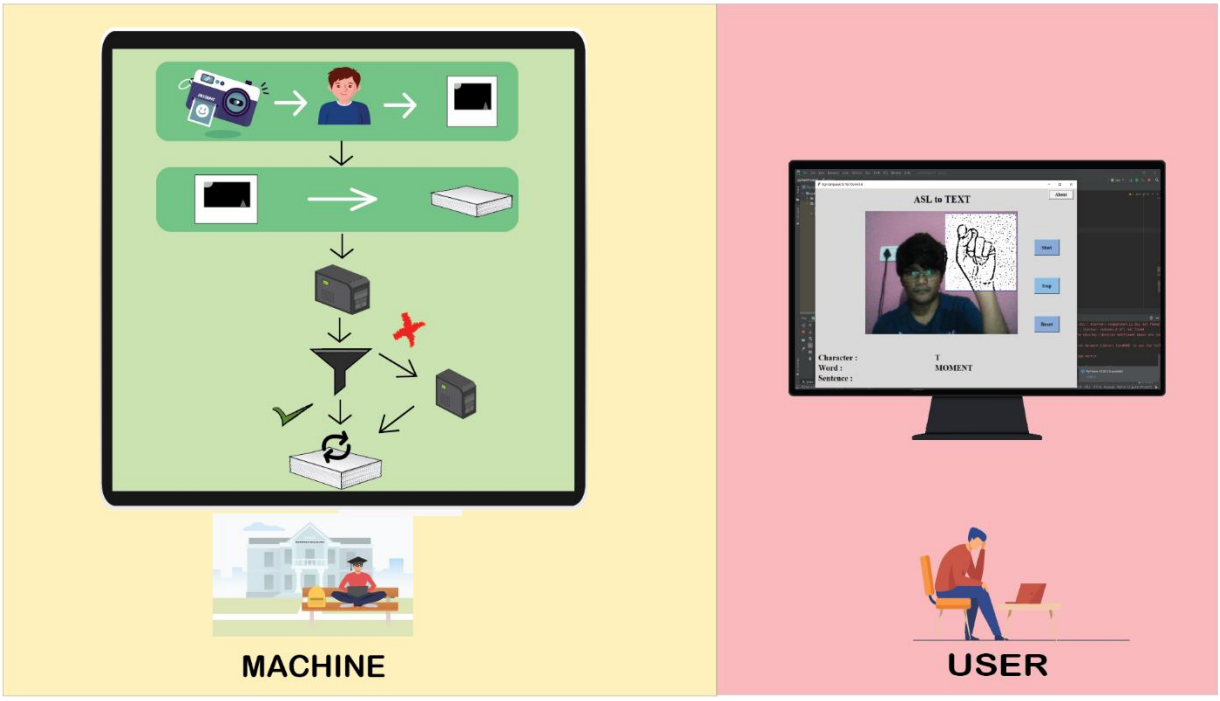

Fig. 3.1: Solution Overview

3.2 Data Acquisition

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 Open computer vision (OpenCV) library in order to produce our dataset. Firstly, we captured around 1,100 images of each of the symbol in ASL (American Sign Language) for training purposes and around 200 images per symbol for testing purpose. 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 as shown in the image below.

Fig. 3.2: Data Collection

From this whole image we extract our ROI which is RGB and convert it into grayscale image. Then, we apply our gaussian blur filter to our image which helps us extracting various features of our image. In our project we have made use of **Adaptive Threshold technique.** It is used to separate desirable foreground image objects from the background based on the difference in pixel intensities of each region. The image after applying the above looks like:

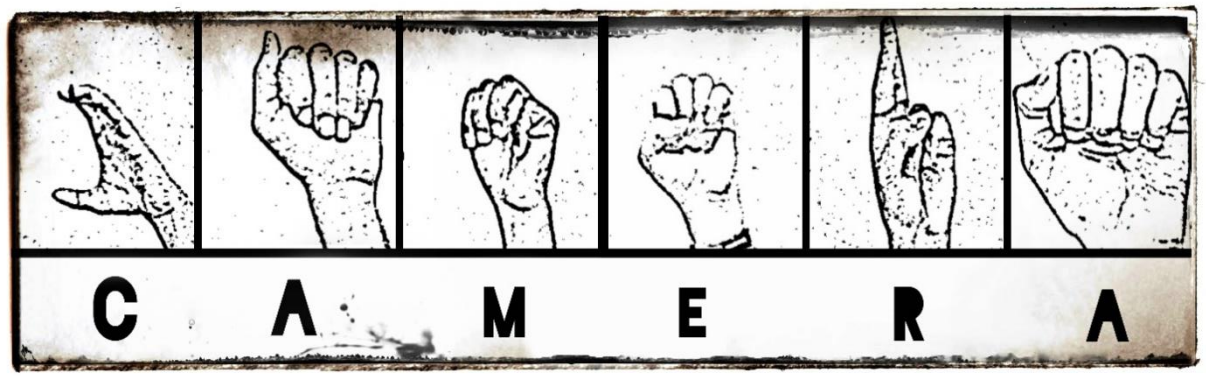

Fig. 3.3: Collected image samples

3.3 Data Pre-processing

3.3.1 Fetching the Data

After applying the above collection methods and filters we load the final archived data that has been stored in a storage drive to our project file for the further pre-processing and final dataset creation that will be fed to our model.

3.3.2 Size and dimensional analysis

We take one image at a time of an alphabet and resize it to a square dimension of 128 x 128 pixels using OpenCV so as to support our CNN model in faster processing.

We use the NumPy Python Library and convert the whole image dataset into a 4D NumPy Array. A linear array works as a list and when fed to a model, this data takes immense amounts of runtime and memory to execute, whereas the same data when fed to a model in a matrix format executes faster and with better efficiency. Also, NumPy arrays of this nature is heavily used in the domain of Machine Learning when it comes to model data feeding.

```
# Processing image
gray = cv2.cvtColor(roi, 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, image = cv2.threshold(th3, minValue, 255, cv2.THRESH_BINARY_INV + cv2.THRESH_OTSU)

cv2.imshow('processed image', image)
image = cv2.resize(image, (128, 128))D
```

Finally, the shape of the 4D NumPy array will look like:

```
Xtest.shape

(4268, 128, 128, 1)
```

The format of the above displayed NumPy array is:

Array [Number of images, length, breadth, channel] , where

- Number of images is the total number of images present in the pre-processed dataset.
- Length and Breadth are the dimensions of the images in pixels, i.e., 128 x 128.
- Channel is basically the number of primary colour schemes present per image. For example, in a RGB image there are 3 channels, one each for RED, GREEN, and BLUE. But, for a grayscale image the channel becomes 1 for we have to select only one Primary colour scheme.

3.4 Model Creation

3.4.1 Algorithm:

The approach which we used for this project is:

Our approach uses two layers of algorithm to predict the final symbol of the letter sign displayed by the user.

Algorithm Layer 1:

- Apply gaussian blur filter and threshold to the frame taken with OpenCV to get the processed image after feature extraction.
- This processed image is passed to the CNN model for prediction and if a letter is detected for more than 40 frames then the letter is printed and taken into consideration for forming the word.
- Space between the words is considered using the blank symbol.

Algorithm Layer 2:

- We once again detect various sets of symbols particularly which show similar results on getting detected in the first layer.
- We then classify between those sets using classifiers made for those sets only.

3.4.2 CNN Model Overview:

The CNN model that we have created for our project follows a certain flow of operation and layers to reach the final results.

First, we have grouped two Convolutional layers (using ReLu activation function) with each having one Batch Normalization function after each and then one Max Pooling Layer followed by a Dropout Layer. Now, this whole group is repeated in 4 times and flow is transferred to the Flatten Layer.

The Flatten layer now converts the whole input to a linear one-dimensional array from a multi-dimensional one. After this we enter the next group of Dense Layers.

In the next group we have combined a dense layer (using ReLu activation function) followed by a dropout function (to avoid overfitting of data). This group is executed in 3 times one after the other.

Finally, the program flow enters the output layer in which it passes through one final Dense layer which using the Softmax activation function.

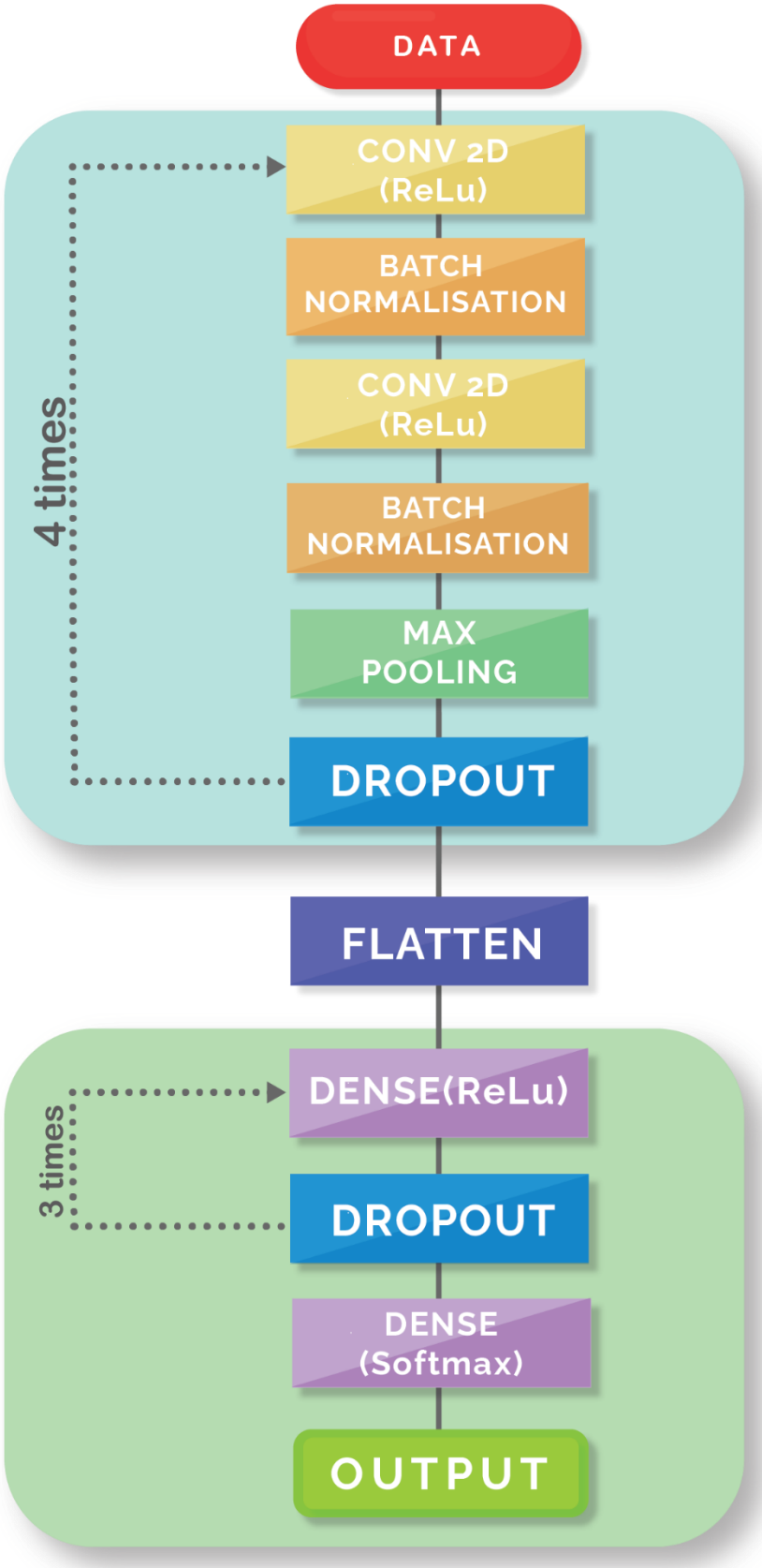

Fig3.4: CNN Model flowchart

3.4.3 Convolutional Layers

The NumPy array gets passed into the Convolution2D layer where we specify the number of filters as one of the hyperparameters. The set of filters (aka. Kernel) are unique with randomly generated weights. Each filter, receptive field, slides across the original image with shared weights to create a feature map. Convolution generates feature maps that represent how pixel values are enhanced, for example, edge and pattern detection. A feature map is created by applying filter across the entire image. Other filters are applied one after another creating a set of feature maps.

In our model, the Conv 2D layers are in pairs and there are 4 pairs in total executing one after the other (Sequential Model). The basic hyperparameters of each Pair of the Conv 2D layers is as stated:

Pair 1: Filter size 64, kernel size 3 x 3, ReLu Activation function used.

Pair 2: Filter size 128, kernel size 3 x 3, ReLu Activation function used.

Pair 3: Filter size 256, kernel size 3 x 3, ReLu Activation function used.

Pair 4: Filter size 512, kernel size 3 x 3, ReLu Activation function used.

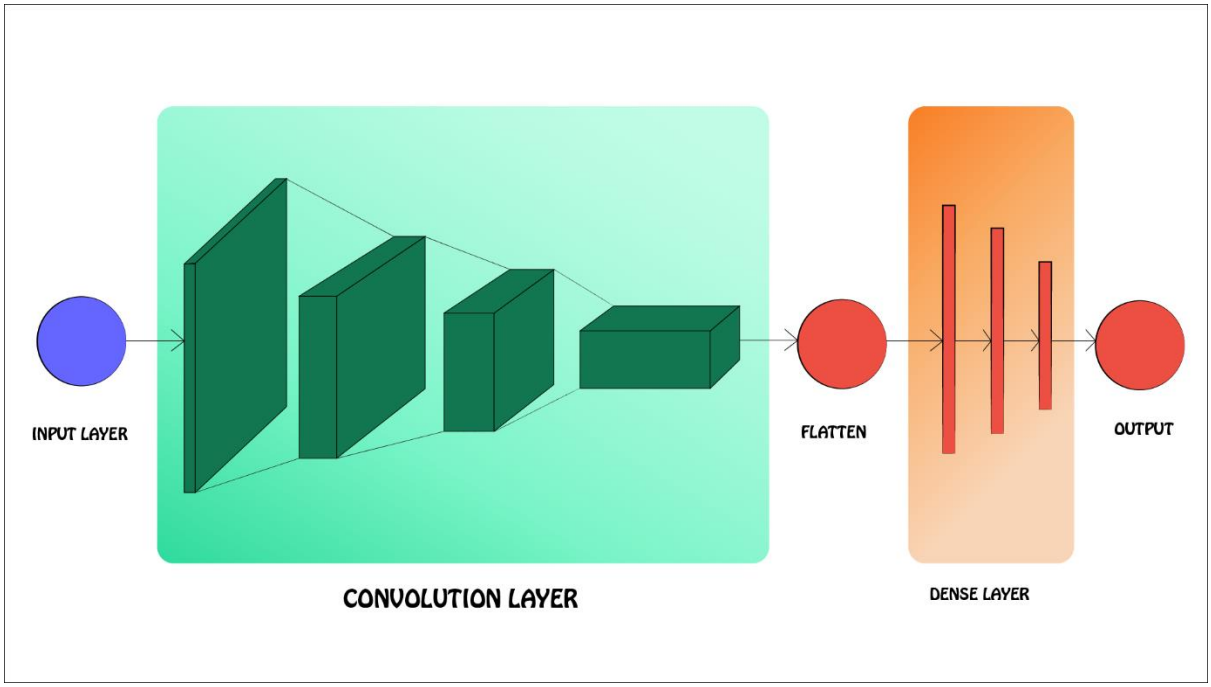

Fig. 3.5: Convolution Neural Network

3.4.4 Batch Normalization

Batch Normalization is a technique that mitigates the effect of unstable gradients within deep neural networks. BN introduces an additional layer to the neural network that performs operations on the inputs from the previous layer.

Normalization is typically carried out on the input data, but it would make sense that the flow of internal data within the network should remain normalized. BN is the internal enforcer of normalization within the input values passed between the layer of a neural network. Internal normalization limits the covariate shift that usually occurs to the activations within the layers.

Input: Values of x over a mini-batch: $\mathcal{B} = \{x_{1...m}\}$;
Parameters to be learned: γ, β
Output: $\{y_i = \mathrm{BN}_{\gamma,\beta}(x_i)\}$

$$\mu_{\mathcal{B}} \leftarrow \frac{1}{m}\sum_{i=1}^{m} x_i \qquad \text{// mini-batch mean}$$

$$\sigma_{\mathcal{B}}^2 \leftarrow \frac{1}{m}\sum_{i=1}^{m}(x_i - \mu_{\mathcal{B}})^2 \qquad \text{// mini-batch variance}$$

$$\widehat{x}_i \leftarrow \frac{x_i - \mu_{\mathcal{B}}}{\sqrt{\sigma_{\mathcal{B}}^2 + \epsilon}} \qquad \text{// normalize}$$

$$y_i \leftarrow \gamma\widehat{x}_i + \beta \equiv \mathrm{BN}_{\gamma,\beta}(x_i) \qquad \text{// scale and shift}$$

where,

‘m’ refers to the number of inputs in the mini-batch.

‘μ’ refers to the mean.

‘B’ is a subscript that refers to the current batch.

‘xi’ is an instance of the input data.

The mean(‘μ’) of a batch(‘B’) is calculated by the sum of the several input instances of the batch and dividing it by the total number of inputs(‘m’).

3.4.5 MaxPooling

Pooling is a dimension reduction technique usually applied after one or several convolutional layers. It is an important step when building CNNs as adding more convolutional layers can greatly affect computational time. We used a popular pooling method called MaxPooling2D that uses (2, 2) windows across the feature map only keeping the maximum pixel value. The pooled pixels form an image with dimensions reduced by 4.

We have used one MaxPooling layer after every Conv2D pair in our Model.

3.4.6 Dropout

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.

3.4.7 Flatten

Flattening is converting the data into a 1-dimensional array for inputting it to the next layer. We flatten the output of the convolutional layers to create a single long feature vector. And it is connected to the final classification model, which is called a fully-connected layer. In other words, we put all the pixel data in one line and make connections with the final layer.

3.4.8 Dense Layer

The dense layer is a neural network layer that is connected deeply, which means each neuron in the dense layer receives input from all neurons of its previous layer. The dense layer is found to be the most commonly used layer in the models.

In the background, the dense layer performs a matrix-vector multiplication. The values used in the matrix are actually parameters that can be trained and updated with the help of backpropagation.

The output generated by the dense layer is an 'm' dimensional vector. Thus, dense layer is basically used for changing the dimensions of the vector. Dense layers also apply operations like rotation, scaling, translation on the vector.

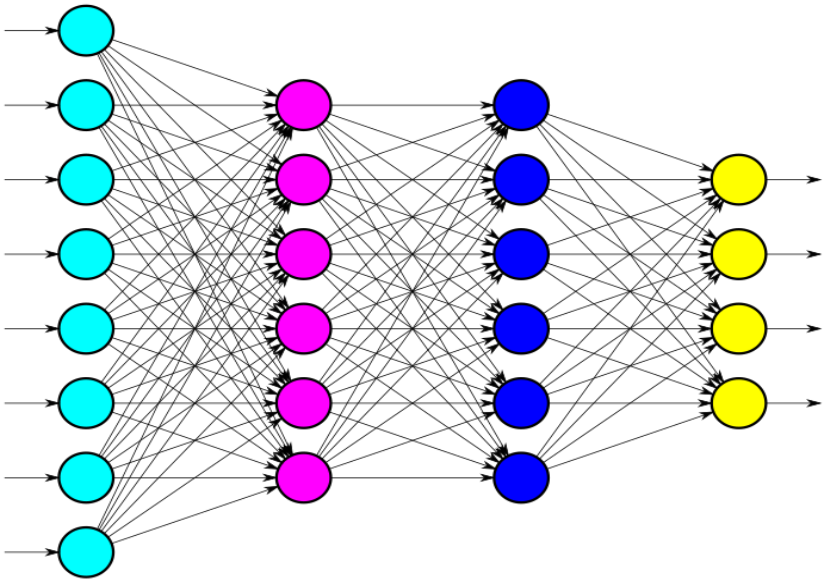

Fig. 3.6: Dense Layer

3.4.9 Activation Function:

Activation functions are functions used in neural networks to computes the weighted sum of input and biases, of which is used to decide if a neuron can be fired or not. It manipulates the presented data through some gradient processing usually gradient descent and afterwards produce an output for the neural network, that contains the parameters in the data.

In our project, we have made use of particularly two Activation Functions namely Softmax Activation Function and ReLu Activation.

Softmax Activation Function: The Softmax function is another type of activation function used in neural computing. It is used to compute probability distribution from a vector of real numbers. The Softmax function produces an output which is a range of values between 0 and 1, with the sum of the probabilities been equal to 1. The Softmax function [5] is computed using the relationship:

$$f(x_i) = \frac{exp\ (x_i)}{\sum_j exp\ (x_j)}$$

The Softmax function is used in multi-class models where it returns probabilities of each class, with the target class having the highest probability.

ReLu Activation Function: The ReLU represents a nearly linear function and therefore preserves the properties of linear models that made them easy to optimize, with gradient-descent methods [5]. The ReLU activation function performs a threshold operation to each input element where values less than zero are set to zero thus the ReLU is given by:

$$f(x) = max\ (0, x) = \begin{cases} x_i, & if\ x_i \geq 0 \\ 0, & if\ x_i < 0 \end{cases}$$

The main advantage of using the rectified linear units in computation is that, they guarantee faster computation since it does not compute exponentials and divisions. Another property of the ReLU is that it introduces sparsity in the hidden units as it squishes the values between zero to maximum.

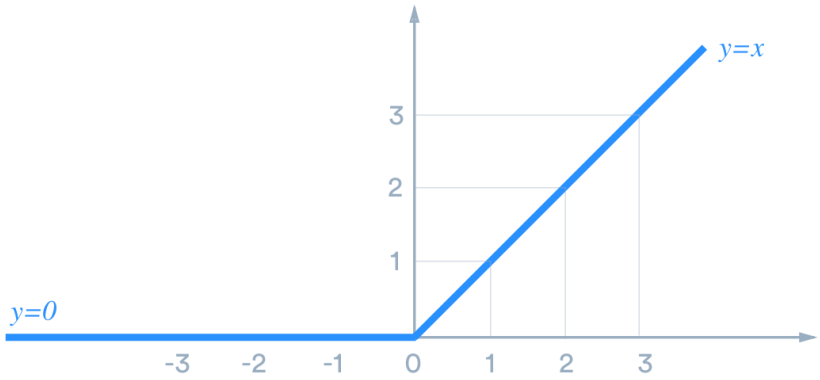

Fig.3.7: Activation function (ReLu) Graph

3.4.10 Optimizer:

Optimizers are algorithms or methods used to minimize an error function (loss function) or to maximize the efficiency of production. Optimizers are mathematical functions which are dependent on model's learnable parameters i.e., Weights & Biases. Optimizers help to know how to change weights and learning rate of neural network to reduce the losses.

In our model, we have used a reputed optimizer named Adam Optimizer.

Adam Optimizer: Adam optimizer is one of the most popular and famous gradient descent optimization algorithms. It is a method that computes adaptive learning rates for each parameter. It stores both the decaying average of the past gradients, similar to momentum and also the decaying average of the past squared gradients, similar to RMS-Prop and AdaDelta. Thus, it combines the advantages of both the methods.

$$w_t = w_{t-1} - \frac{\eta}{\sqrt{S_{dw_t} - \varepsilon}} * V_{dw_t}$$

$$b_t = b_{t-1} - \frac{\eta}{\sqrt{S_{db_t} - \varepsilon}} * V_{db_t}$$

```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d (Conv2D)              (None, 126, 126, 64)      640
_________________________________________________________________
batch_normalization (BatchNo (None, 126, 126, 64)      256
_________________________________________________________________
conv2d_1 (Conv2D)            (None, 124, 124, 64)      36928
_________________________________________________________________
batch_normalization_1 (Batch (None, 124, 124, 64)      256
_________________________________________________________________
max_pooling2d (MaxPooling2D) (None, 62, 62, 64)        0
_________________________________________________________________
dropout (Dropout)            (None, 62, 62, 64)        0
_________________________________________________________________
conv2d_2 (Conv2D)            (None, 60, 60, 128)       73856
_________________________________________________________________
batch_normalization_2 (Batch (None, 60, 60, 128)       512
_________________________________________________________________
conv2d_3 (Conv2D)            (None, 58, 58, 128)       147584
_________________________________________________________________
batch_normalization_3 (Batch (None, 58, 58, 128)       512
_________________________________________________________________
max_pooling2d_1 (MaxPooling2 (None, 29, 29, 128)       0
_________________________________________________________________
dropout_1 (Dropout)          (None, 29, 29, 128)       0
_________________________________________________________________
conv2d_4 (Conv2D)            (None, 27, 27, 256)       295168
_________________________________________________________________
batch_normalization_4 (Batch (None, 27, 27, 256)       1024
_________________________________________________________________
conv2d_5 (Conv2D)            (None, 25, 25, 256)       590080
_________________________________________________________________
batch_normalization_5 (Batch (None, 25, 25, 256)       1024
_________________________________________________________________
max_pooling2d_2 (MaxPooling2 (None, 12, 12, 256)       0
_________________________________________________________________
dropout_2 (Dropout)          (None, 12, 12, 256)       0
_________________________________________________________________
conv2d_6 (Conv2D)            (None, 10, 10, 512)       1180160
_________________________________________________________________
batch_normalization_6 (Batch (None, 10, 10, 512)       2048
_________________________________________________________________
conv2d_7 (Conv2D)            (None, 8, 8, 512)         2359808
_________________________________________________________________
batch_normalization_7 (Batch (None, 8, 8, 512)         2048
_________________________________________________________________
max_pooling2d_3 (MaxPooling2 (None, 4, 4, 512)         0
_________________________________________________________________
dropout_3 (Dropout)          (None, 4, 4, 512)         0
_________________________________________________________________
flatten (Flatten)            (None, 8192)              0
_________________________________________________________________
dense (Dense)                (None, 256)               2097408
_________________________________________________________________
dropout_4 (Dropout)          (None, 256)               0
_________________________________________________________________
dense_1 (Dense)              (None, 128)               32896
_________________________________________________________________
dropout_5 (Dropout)          (None, 128)               0
_________________________________________________________________
dense_2 (Dense)              (None, 64)                8256
_________________________________________________________________
dropout_6 (Dropout)          (None, 64)                0
_________________________________________________________________
dense_3 (Dense)              (None, 27)                1755
=================================================================
Total params: 6,832,219
Trainable params: 6,828,379
Non-trainable params: 3,840
```

Fig. 3.8: Model Summary

CHAPTER 4: IMPLEMENTATION

4.1 Training and Testing

4.1.1 Phase 1:

Training:

For the Phase 1 Training, we used our CNN model design as described earlier to implement a Classifier and then fed the collected dataset (pre-processed image files) into the model. The prediction layer estimates how likely the image will fall under one of the classes. 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 softmax function.

Testing:

For the Phase 1 Testing, we had some parameters set as:

- If during detection, a character is detected 25 times it is probably that character.
- Also, the most detected character must have a Count difference of 10 from the second most detected character.

After completing the testing for every hand sign corresponding to the alphabets in ASL, we found that certain signs were not being detected accurately. On further analysis, we recognized that there are certain sets of signs which look similar to the camera and thus produce an uncertain prediction.

- For D: R and U
- For S: M, T, and N
- For M: S and N
- U: D and R

These signs were then segregated into the following sets:

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

4.1.2 Phase 2:

Training:

As we saw the sets formed from incorrect detection of some signs in Phase 1, we made particular classifiers for both the sets {D, R, U} and {S, M, T, N}.

Then, we trained these classifiers again with these signs only and made sure that the hyperparameters setting this time works to detect them accurately even if the classifier detects other signs incorrectly.

Lastly here is how we modified our final CNN model:

- If during detection we encounter signs belonging to any character other than {D, R, U} and {S, M, T, N}, then we continue to use the phase 1 classifier.
- If we detect any signs among {D, R, U} and {S, M, T, N}, then they go through their separate set classifier once again and then their final output is shown to the user.

```
Xtrain = [cv2.resize(np.asarray(cv2.imread('train/' + symbol + '/' + file, cv2.IMREAD_GRAYSCALE)), (128, 128),
                     interpolation=cv2.INTER_CUBIC) for symbol in ['M', 'N', 'S', 'T'] for file in
          os.listdir('train/' + symbol)]
ytrain = [symbol for symbol in ['M', 'N', 'S', 'T'] for _ in range(len(os.listdir('train/' + symbol)))]

Xtest = [cv2.resize(np.asarray(cv2.imread('test/' + symbol + '/' + file, cv2.IMREAD_GRAYSCALE)), (128, 128),
                    interpolation=cv2.INTER_CUBIC) for symbol in ['M', 'N', 'S', 'T'] for file in
         os.listdir('test/' + symbol)]
ytest = [symbol for symbol in ['M', 'N', 'S', 'T'] for _ in range(len(os.listdir('test/' + symbol)))]
```

Fig. 4.1 Data Collection for {M, N, S, T}

```
Xtrain = [cv2.resize(np.asarray(cv2.imread('train/' + symbol + '/' + file, cv2.IMREAD_GRAYSCALE)), (128, 128),
                     interpolation=cv2.INTER_CUBIC) for symbol in ['D', 'R', 'U'] for file in
          os.listdir('train/' + symbol)]
ytrain = [symbol for symbol in ['D', 'R', 'U'] for _ in range(len(os.listdir('train/' + symbol)))]

Xtest = [cv2.resize(np.asarray(cv2.imread('test/' + symbol + '/' + file, cv2.IMREAD_GRAYSCALE)), (128, 128),
                    interpolation=cv2.INTER_CUBIC) for symbol in ['D', 'R', 'U'] for file in
         os.listdir('test/' + symbol)]
ytest = [symbol for symbol in ['D', 'R', 'U'] for _ in range(len(os.listdir('test/' + symbol)))]
```

Fig. 4.2: Data Collection for {D, R, U}

Testing:

During the testing of Phase 2 model, we saw that all the signs were being recognized 83 percent of the times and it went up to 92 percent when lighting conditions and contrast with background was improved to reduce surrounding noise in the live feed.

To make the model perform 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 is not 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 with the best gradient descent optimizer is called Adam Optimizer.

```
model.fit(x = Xtrain,
                y = ytrain,
                epochs=50,
                batch_size = 64,
                validation_data=(Xtest, ytest))
```

```
Epoch 1/50
488/488 [==============================] - 145s 223ms/step - loss: 2.3904 - accuracy: 0.3134 - val_loss: 2.7897 - val_accuracy: 0.3292
Epoch 2/50
488/488 [==============================] - 105s 215ms/step - loss: 1.0032 - accuracy: 0.6877 - val_loss: 3.5789 - val_accuracy: 0.3278
Epoch 3/50
488/488 [==============================] - 105s 215ms/step - loss: 0.6723 - accuracy: 0.7896 - val_loss: 3.8343 - val_accuracy: 0.3575
Epoch 4/50
488/488 [==============================] - 105s 215ms/step - loss: 0.5085 - accuracy: 0.8359 - val_loss: 3.6838 - val_accuracy: 0.3859
Epoch 5/50
488/488 [==============================] - 105s 215ms/step - loss: 0.4290 - accuracy: 0.8621 - val_loss: 3.5799 - val_accuracy: 0.4381
Epoch 6/50
488/488 [==============================] - 105s 214ms/step - loss: 0.3757 - accuracy: 0.8791 - val_loss: 4.2640 - val_accuracy: 0.3236


488/488 [==============================] - 105s 215ms/step - loss: 0.0840 - accuracy: 0.9734 - val_loss: 6.3496 - val_accuracy: 0.4147
Epoch 45/50
488/488 [==============================] - 105s 215ms/step - loss: 0.0785 - accuracy: 0.9740 - val_loss: 8.7244 - val_accuracy: 0.3440
Epoch 46/50
488/488 [==============================] - 105s 214ms/step - loss: 0.0853 - accuracy: 0.9719 - val_loss: 5.2288 - val_accuracy: 0.4574
Epoch 47/50
488/488 [==============================] - 104s 214ms/step - loss: 0.0744 - accuracy: 0.9748 - val_loss: 4.9279 - val_accuracy: 0.4339
Epoch 48/50
488/488 [==============================] - 104s 214ms/step - loss: 0.0735 - accuracy: 0.9755 - val_loss: 7.4412 - val_accuracy: 0.4377
Epoch 49/50
488/488 [==============================] - 104s 214ms/step - loss: 0.0803 - accuracy: 0.9726 - val_loss: 11.5128 - val_accuracy: 0.3069
Epoch 50/50
488/488 [==============================] - 104s 214ms/step - loss: 0.0741 - accuracy: 0.9736 - val_loss: 8.3663 - val_accuracy: 0.3208
<tensorflow.python.keras.callbacks.History at 0x7f2c9065b4d0>
```

Fig. 4.3: Model Training (Initial and final epoch)

4.2 User Interface:

To design our user interface, we have made use of Tkinter. It is the standard Python interface to the Tk GUI toolkit, and is Python's de facto standard GUI.

Firstly, we created a window(frame) and then used Tkinter labels(containers) to design button, textboxes, images, camera live feed etc. Then, we added functionality to the buttons by writing separate codes for each functional button.

In our UI, we have three FOUR buttons as follows:

- START Button: When we click this button, the backend code is triggered and the live feed starts detecting the signs appearing on the shown part of the feed.
- STOP Button: When we click this button, the backend code is stopped and hence any further signs are not detected even if the live feed is ongoing.
- RESET Button: When we click this button, all previously detected texts vanish and the STOP functionality as described earlier is also simultaneously triggered. It is just like a restart of the UI.
- ABOUT Button: This button displays Project developer details when clicked.

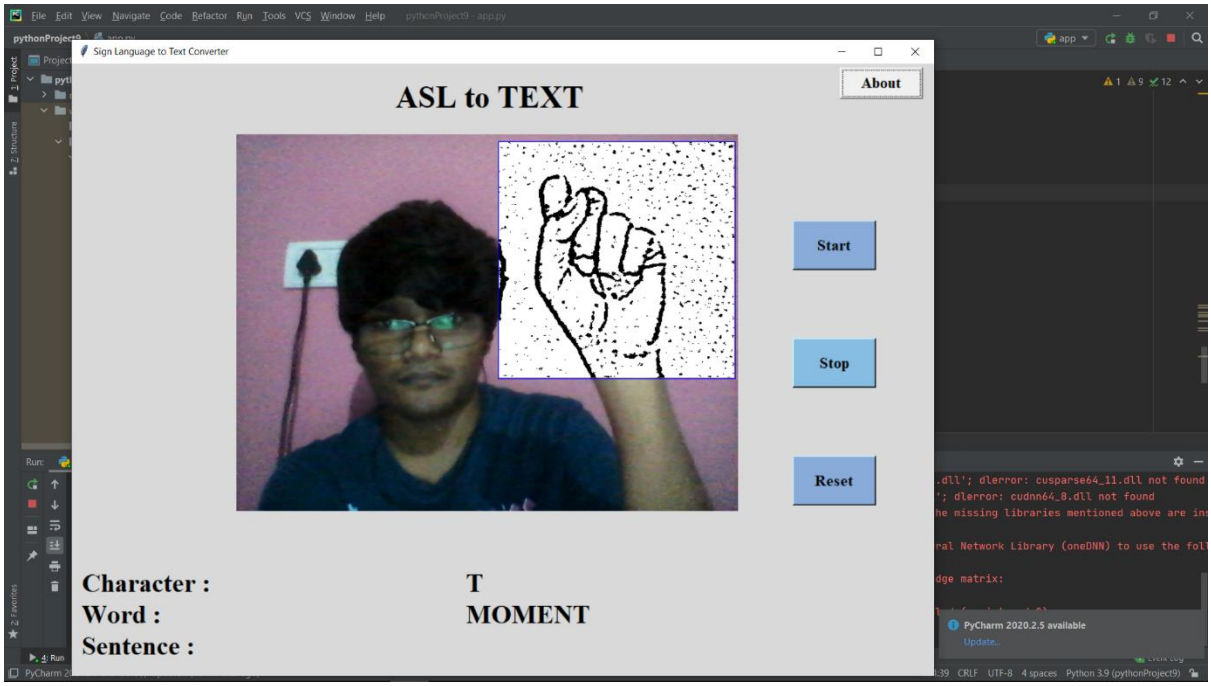

Fig. 4.4: User Interface

CHAPTER 5: RESULT

We have achieved an accuracy of 92.37% in our model using the combination of layer 1 and layer 2. Most of the research papers focus on using devices like Kinect [6] for hand detection. They also used CNN for their recognition system. One thing should be noted that our model doesn't uses any background subtraction algorithm while some of the models present before do that. So, once we try to implement background subtraction in our project the accuracies may vary.

Our main aim was to create a project which can be used with readily available resources for a naïve user wanting to access this technology without spending a lot on hardware. A sensor not only isn't readily available but also is expensive. Thus, our model uses a normal webcam of the laptop.

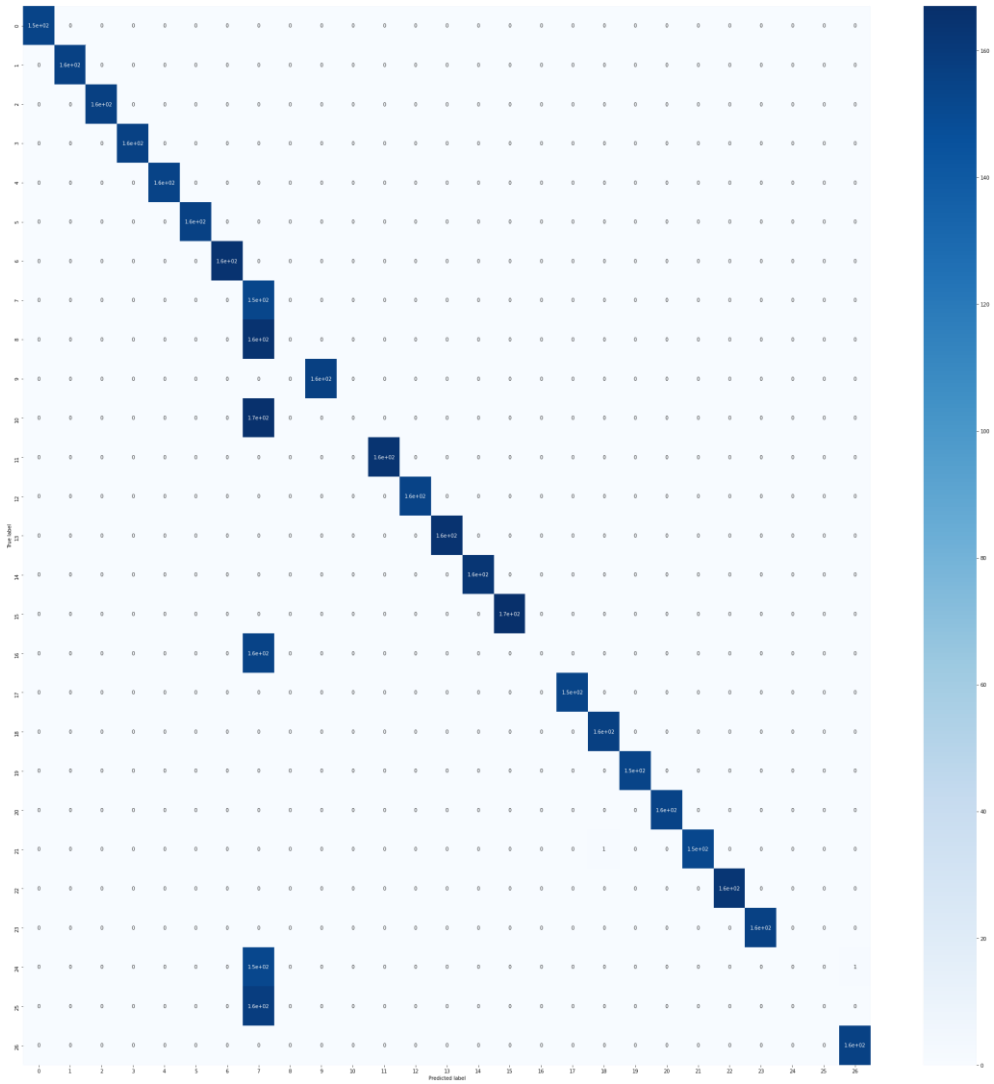

Fig: 5.1 Model Heatmap

CHAPTER 6: CONCLUSION

Based on the present work, the following conclusions can be drawn:

a) A finger spelling sign language translator is obtained which has an overall **accuracy of 92.37%**. The project can be extended to other sign languages by building the corresponding dataset and training the CNN

b) The project is a simple demonstration of how CNN can be used to solve computer vision problems with an extremely high degree of accuracy

c) The main objective has been achieved, that is, the need for an interpreter has been eliminated.

d) Bad lighting conditions can lead to errors in detection of signs by the algorithm due to surrounding noise. Also, a lighter background contrasting with the hand being detected would yield better outputs as experienced by us.

e) The other issue that people might face is regarding their proficiency in knowing the ASL gestures. Bad gesture postures will not yield correct predictions.

f) Future prospects: This project can be used as a web or mobile application in various fields like education, entertainment, and defence. One can use it to predict already existing sign languages like ASL (already done here) and also create their own secret code language (can be used by the army).

References

[1] J. S. Sonkusare, N. B. Chopade, R. Sor, and S. L. Tade (2017), "A Review on Hand Gesture Recognition System" Issue: pp. 790- 794.

[2] H.Y.LaiandH.J.Lai (2017),"Real Time Dynamic Hand Gesture Recognition" IEEEInt.Symp.Comput.Consum.Control, Issue :01, pp. 658-661

[3] Sergey Ioffe, Christian Szegedy, “Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift” (arxiv.org)

[4] A. Thorat, V. Satpute, A. ehe, T.AtreY.Ngargoje (2016),"Indian Sign Language Recognition System for Deaf People," Int.J.Adv.compt.comm.engg.IJARCCE,Issue: pp.5319-5321,2016.

[5] I. Goodfellow, Y. Bengio, and A. Courville, “Deep learning.” MIT Press, 2016. [Online]. Available:[http://www.deeplearningbook.org]

[6] https://www.researchgate.net/publication/260349786_Robust_Part Based_Hand_Gesture_Recognition_Using_Kinect_Sensor

[7] Neural Networks and Deep Learning on Coursera.org [https://www.coursera.org/learn/neural-networks-deep-learning]

[8] Convolution neural Network [https://www.youtube.com/watch?v=Jy9-aGMB_TE]