

# **SIGN LANGUAGE RECOGNITION USING CNN**

## **A PROJECT REPORT**

*Submitted by*

**SWATHI M                      2017504046**

**DHARINI M                    2017504521**

**HARINI K                      2017504531**

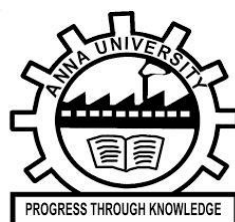
*In partial fulfilment for the award of the degree*

*of*

**BACHELOR OF ENGINEERING**

**IN**

**ELECTRONICS AND COMMUNICATION ENGINEERING**



**MADRAS INSTITUTE OF TECHNOLOGY**

**ANNA UNIVERSITY : CHENNAI 600 044**

**NOVEMBER 2020**

# **ANNA UNIVERSITY : CHENNAI 600 044**

## **BONAFIDE CERTIFICATE**

Certified that this project report titled, “**SIGN LANGUAGE RECOGNITION USING CNN**” is the bonafide work of

**SWATHI M      2017504046**

**DHARINI M      2017504521**

**HARINI K      2017504531**

who carried out the project work under my supervision.

**SIGNATURE**

**Dr.M.GANESH MADHAN**

**HEAD OF THE DEPARTMENT**

Professor,

Department of Electronics Engg,

Madras Institute of Technology,

Anna University,

Chennai – 600 044

**SIGNATURE**

**Dr.P.PRAKASH**

**SUPERVISOR**

Assistant Professor,

Department of Electronics Engg,

Madras Institute of Technology,

Anna University,

Chennai – 600 044

## ACKNOWLEDGEMENT

We consider it as our privilege and our primary duty to express our gratitude and respect to all those who guided and inspired us in the successful completion of the project.

We owe solemn gratitude to **Dr.T.THYAGARAJAN**, Dean, Madras Institute of Technology, for having given consent to carry out the project work at MIT Campus, Anna University.

We wish to express our sincere appreciation and gratitude to **Dr.M.GANESH MADHAN**, Professor and Head of the Department of Electronics Engineering, MIT Campus, Anna University.

We sincerely thank all our project co-ordinators **Dr.P.PRAKASH** and **Miss.P.KASTHURI** for providing us valuable information and for their help and support towards the successful completion of this project. We also thank all the teaching and non-teaching staff members of the Department of Electronics Engineering for their support in all aspects.

**SWATHI M      2017504046**

**DHARINI M      2017504521**

**HARINI K      2017504531**

## **ABSTRACT**

In the realm of multimodal communication, sign language is, and continues to be, one of the most understudied areas. In line with recent advances in the field of deep learning, there are far reaching implications and applications that neural networks can have for sign language interpretation. This paper deals with robust modelling of static signs in the context of sign language recognition using deep learning-based convolutional neural networks (CNN). In this paper, we present a method for using deep convolutional networks to classify images of total 14,500 sign images of 29 static signs of digits and some special characters in American Sign Language. The images were fed into the model called the Convolutional Neural Network (CNN) for classification of images. Keras was used for training of images. Provided with proper lighting condition and a uniform background, the system acquired an average testing accuracy of 99.587%. Also the performance has been compared with that of other CNNs in recent literatures.

# **TABLE OF CONTENTS**

<b>CHAPTER NO</b>	<b>TITLE</b>	<b>PAGE NO</b>
	<b>ABSTRACT</b>	4
	<b>LIST OF TABLES</b>	8
	<b>LIST OF ABBREVIATIONS</b>	8
	<b>LIST OF FIGURES</b>	9
<b>1.</b>	<b>INTRODUCTION</b>	10
	1.1 MOTIVATION	10
	1.2 OVERVIEW	10
	1.3 DEEP LEARNING TECHNOLOGY	12
	1.4 DEEP LEARNING CONCEPTS	13
	1.4.1 ARTIFICIAL NEURAL NETWORK	13
	1.4.2 DEEP NEURAL NETWORK	14
<b>2.</b>	<b>LITERATURE SURVEY</b>	15
	2.1 EXISTING SYSTEM	15
	2.2 PROBLEM STATEMENT	17
	2.3 PROPOSED SYSTEM	18
<b>3.</b>	<b>ARCHITECTURE AND SYSTEM DESIGN</b>	19
	3.1 ARCHITECTURE	19

# TABLE OF CONTENTS

CHAPTER NO	TITLE	PAGE NO
	3.2 DESCRIPTION OF MODULES	23
	3.2.1 DATA ACQUISITION	23
	3.2.2 DATA PREPROCESSING	24
	3.2.3 MODEL TRAINING	24
	3.2.4 TESTING	26
	3.3 FLOW CHART	26
<b>4.</b>	<b>SYSTEM IMPLEMENTATION</b>	27
	4.1 SIMULATION	27
	4.1.1 DATA ACQUISITION	27
	4.1.2 DATA PREPROCESSING	29
	4.1.3 MODEL TRAINING	31
	4.1.4 TESTING THE MODEL	33
	4.2 SOFTWARE REQUIREMENTS	35
	4.2.1 GOOGLE COLAB	35
	4.2.2 TENSORFLOW	36
	4.2.3 KERAS	37
	4.2.4 OPEN CV	37
	4.2.5 NUMPY	38
	4.2.6 PANDAS	38

## TABLE OF CONTENTS

CHAPTER NO	TITLE	PAGE NO
	4.2.7 SKLEARN	39
<b>5.</b>	<b>RESULTS AND DISCUSSION</b>	40
	5.1 RESULTS	40
	5.2 EVALUATION	43
	5.2.1 EVALUATION ACCURACY	44
	5.2.2 LOSS FUNCTION	45
	5.2.3 CONFUSION MATRIX	46
	5.2.4 MODEL COMPARISON	48
<b>6.</b>	<b>CONCLUSION AND FUTURE WORK</b>	50
	6.1 CONCLUSION	50
	6.2 FUTURE ENHANCEMENT	51
<b>7</b>	<b>REFERENCES</b>	52

## **LIST OF TABLES**

<b>TABLE NO</b>	<b>TITLE</b>	<b>PAGE NO</b>
3.3	PROPOSED SYSTEM ARCHITECTURE	25
5.6	MODEL COMPARISON	49

## **LIST OF ABBREVIATIONS**

<b>TERMS</b>	<b>EXPLANATION</b>
SLR	Sign Language Recognition
ASL	American Sign Language
CNN	Convolution Neural Network
ANN	Artificial Neural Network
DNN	Deep Neural Network
CV	Computer Vision
Adam	Adaptive Moment Estimation
ReLU	Rectified Linear Unit



## LIST OF FIGURES

FIGURE NO	TITLE	PAGE NO
3.1	SYSTEM ARCHITECTURE	22
3.2	THE CONVOLUTION OPERATION	23
3.4	FLOW CHART OF THE PROPOSED SYSTEM	26
4.1	PLOTTING ONE IMAGE FROM EACH CLASS	29
4.2	GOOGLE COLAB PLATFORM	36
5.1	PREDICTION MADE BY THE MODEL ON GIVEN TEST DATA SET	41
5.2	PREDICTION MADE BY THE MODEL ON REAL WORLD TEST IMAGES	42
5.3a	ACCURACY PLOT – TRAIN VS TEST __ GIVEN TEST DATA SET	44
5.3b	ACCURACY PLOT – TRAIN VS TEST __ REAL WORLD TEST DATA SET	44
5.4a	LOSS PLOT – TRAINING VS VALIDATION __ GIVEN TEST DATA SET	45
5.4b	LOSS PLOT – TRAINING VS VALIDATION __ REAL WORLD TEST DATA	46
5.5a	CONFUSION MATRIX __ GIVEN TEST DATA SET	47
5.5b	CONFUSION MATRIX __ REAL WORLD TEST DATA	48

# **CHAPTER 1**

## **INTRODUCTION**

### **1.1MOTIVATION**

While automatic speech recognition has now advanced to the point of being commercially available, automatic SLR is still in its infancy. Currently all commercial translation services are human based, and therefore expensive, due to the experienced personnel required.

SLR aims to develop algorithms and methods to correctly identify a sequence of produced signs and to understand their meaning. Many approaches to SLR incorrectly treat the problem as Gesture Recognition (GR). So research has thus far focused on identifying optimal features and classification methods to correctly label a given sign from a set of possible signs. However, sign language is far more than just a collection of well specified gestures.

### **1.2 OVERVIEW**

Sign language is a computer vision-based complete convoluted language that engrosses signs shaped by the movements of hands in combination with facial expressions. It is a natural language used by people with low or no hearing sense for communication. A sign language can be used for communication of letters, words or sentences using different signs of the hands. Hand signs are especially useful to express any word or feeling to communicate. Therefore, people around the world use signals from hand constantly to express despite the formulation of writing conventions. This type of communication makes it easier for hearing-impaired people to express their views and also help in bridging the communication gap between hearing-impaired people and other

person. In recent times, much research has been ongoing in developing systems that are able to classify signs of different sign languages into the given class. The automatic recognition of human signs is a complex multidisciplinary problem that has not yet been completely solved. In the past years, a number of approaches were used which involve the use of machine learning techniques for sign language recognition. Since the advent of deep learning techniques, there have been attempts to recognize human signs. Networks which are based on deep learning paradigms deal with the architectures and learning algorithms that are biologically inspired, in distinction to conventional networks.

Generally, the training of deep networks occurs in a layer-wise manner and depends on more distributed features as present in the human visual cortex. In this, the abstract features from the collected signs in the first layer are grouped into primary features in the second layer, which further combined into more defined features present in the next layer. These features are then further combined together into more engrossing features in the following layers, which help in the better recognition of different signs. Most of the research work in sign language recognition based on deep learning technique. Of recent, this area is gaining popularity among research experts. The earliest reported work on sign language recognition is mainly based on machine learning techniques. These methods result in low accuracy as it does not extract features automatically. The main goal of deep learning techniques is automatic feature engineering. The idea behind this is to automatically learn a set of features from raw data that can be useful in sign language recognition. In this manner, it avoids the manual process of handcrafted feature engineering by learning as a set of features automatically. The output of the sign language will be displayed in the text form in real time. This makes the system more efficient and hence communication of the hearing and speech impaired people more easy. The images captured through web cam are compared and the result of comparison is displayed at the same time.

## 1.3 DEEP LEARNING TECHNOLOGY

**Deep learning** (also known as **deep structured learning**) is part of a broader family of machine learning methods based on artificial neural networks with representation learning. Learning can be supervised, semi-supervised or unsupervised.

Deep-learning architectures such as deep neural networks, deep belief networks, recurrent neural networks and convolutional neural networks have been applied to fields including computer vision, machine vision, speech recognition, natural language processing, audio recognition, social network filtering, machine translation, bioinformatics, drug design, medical image analysis, material inspection and board game programs, where they have produced results comparable to and in some cases surpassing human expert performance.

The adjective "deep" in deep learning comes from the use of multiple layers in the network. Early work showed that a linear perceptron cannot be a universal classifier, and then that a network with a non polynomial activation function with one hidden layer of unbounded width can on the other hand so be. Deep learning is a modern variation which is concerned with an unbounded number of layers of bounded size, which permits practical application and optimized implementation, while retaining theoretical universality under mild conditions. In deep learning the layers are also permitted to be heterogeneous and to deviate widely from biologically informed connectionist models, for the sake of efficiency, trainability and understandability, whence the "structured" part.

## 1.4 DEEP LEARNING CONCEPTS

### 1.4.1 ARTIFICIAL NEURAL NETWORKS

**Artificial neural networks (ANNs)** or **connectionist systems** are computing systems inspired by the biological neural networks that constitute animal brains. Such systems learn (progressively improve their ability) to do tasks by considering examples, generally without task-specific programming. They have found most use in applications difficult to express with a traditional computer algorithm using rule-based programming.

An ANN is based on a collection of connected units called artificial neurons, (analogous to biological neurons in a biological brain). Each connection between neurons can transmit a signal to another neuron. The receiving neuron can process the signal(s) and then signal downstream neurons connected to it. Neurons may have state, generally represented by real numbers, typically between 0 and 1. Neurons and synapses may also have a weight that varies as learning proceeds, which can increase or decrease the strength of the signal that it sends downstream.

Typically, neurons are organized in layers. Different layers may perform different kinds of transformations on their inputs. Signals travel from the first (input), to the last (output) layer, possibly after traversing the layers multiple times.

The original goal of the neural network approach was to solve problems in the same way that a human brain would. Over time, attention focused on matching specific mental abilities, leading to deviations from biology such as back propagation, or passing information in the reverse direction and adjusting the network to reflect that information.

Neural networks have been used on a variety of tasks, including computer vision, speech recognition, machine translation, social network filtering, playing board and video games and medical diagnosis.

### **1.3.3 DEEP NEURAL NETWORKS**

A deep neural network (DNN) is an artificial neural network (ANN) with multiple layers between the input and output layers. There are different types of neural networks but they always consist of the same components: neurons, synapses, weights, biases, and functions. These components functioning similar to the human brains and can be trained like any other ML algorithm.

The user can review the results and select which probabilities the network should display (above a certain threshold, etc.) and return the proposed label. Each mathematical manipulation as such is considered a layer, and complex DNN have many layers, hence the name "deep" networks. DNNs can model complex non-linear relationships. DNN architectures generate compositional models where the object is expressed as a layered composition of primitives. The extra layers enable composition of features from lower layers, potentially modeling complex data with fewer units than a similarly performing shallow network.

DNNs are typically feed forward networks in which data flows from the input layer to the output layer without looping back. At first, the DNN creates a map of virtual neurons and assigns random numerical values, or "weights", to connections between them. The weights and inputs are multiplied and return an output between 0 and 1. If the network did not accurately recognize a particular pattern, an algorithm would adjust the weights. That way the algorithm can make certain parameters more influential, until it determines the correct mathematical manipulation to fully process the data.

## **CHAPTER 2**

### **LITERATURE SURVEY**

#### **2.1 EXISTING SYSTEM**

For the past decades, research on Sign Language Recognition (SLR) used sensor-based devices such as SignSpeak. These device used different sensors such as flex and contact sensors, accelerometers and gyros, motion sensors, Kinect sensors , leap motion controller or their combinations for capturing the finger and hand movements; then, by the Principal of Component Analysis, gloves fitted with these sensors were trained to recognize different gestures, and each gesture was then classified into alphabets in real time and displayed using android phone via Bluetooth. SignSpeak was found to have 92% accuracy. Although these sensors provide accurate parameters in measurement of data, they also have limitations; first is their high cost and they need high-end computers with powerful specifications that are compatible with these sensors. The user can encounter difficulties in setting up the device as the sensors are attached to the fingers and palms of a user.

Therefore, many researchers jumped from sensor-based to visual-based SLR.

#### **FEATURES OF PAPER[1] :**

Dataset had 35,000 images with 350 images for each of the static signs. CNN architecture included 4 groups of 2 convolutional layers followed by a max-pool layer, a dropout layer and a flatten layer, and two fully connected layer and one final output layer. The training accuracy is obtained as 99.17% and validation accuracy is 98.80%.

### **MERITS OF PAPER[1] :**

- The proposed system is robust enough to learn 100 different static manual signs with lower error rates, as in contrast to other recognition systems described in other works in which few hand signs are considered for recognition.
- The system will be extended to recognize dynamic signs which require the collection and development of a video-based dataset and the system is tested using CNN architecture by dividing the videos into frames.
- The work will also be extended to develop a mobile-based application for the recognition of different signs in real time.

### **DEMERITS OF PAPER[1] :**

- The results and process were severely affected and hindered by skin color and lighting variations in our self-generated data which led us to resort to a pre-made professionally constructed dataset.
- For future work, there is a need to collect more datasets to refine the recognition method.

### **FEATURES OF PAPER[8] :**

The model used self-generated dataset with 25 images from 5 people for each alphabet and the digits from 1 to 9. They also used a premade dataset to compare their self-generated dataset's performance with preprocessing which was done using background-subtraction techniques. CNN architecture included 3 groups of 2 convolutional layers followed by a max-pool layer and a dropout layer, and two groups of fully connected layer followed by a dropout layer and one final output layer. The objective was that of classification using deep convolutional neural networks to classify every letter and the digits, 0-9, in ASL.



The inputs were fixed size high-pixel images, 200 by 200 or 400 by 400, being padded and resized to 200 by 200. They observed 82.5% accuracy on the alphabet gestures, and 97% validation set accuracy on digits, when using the NZ ASL dataset. 67% accuracy on letters of the alphabet, and 70% accuracy on the digits when using their self-generated dataset.

#### **MERITS OF PAPER[8] :**

- Our method shows to have potential in solving this problem using a simple camera, if enough substantial training data is provided.
- More people have access to simple camera technologies, this could contribute to a scalable solution.

#### **DEMERITS OF PAPER[8] :**

- Our results and process were severely affected and hindered by skin color and lighting variations in our self-generated data which led us to resort to a pre-made professionally constructed dataset.
- With a camera like Microsoft's Kinect that has a depth sensor, this problem is easy to solve. However, such cameras and technology are not widely accessible, and can be costly.

## **2.2 PROBLEM STATEMENT**

To design a translator which can detect sign language performed by a disabled person and then that sign will be fed to a machine-learning algorithm involving a neural network which is then detect, translate and display the alphabet so that a normal person can understand what the sign conveys.

## **2.3 PROPOSED SYSTEM**

The proposed algorithm performs the recognition of the input hand gestures using image processing techniques. The system consists of four well-defined phases, namely Data Gathering , Creating Model , Training Model and Testing Model.

A separate directory is created with a collection of various sub-directories labelled as A, B, C, D,E, F, G, H, I, J, K, L, M, N, O, P, Q, R, S, T, U, V, W, X, Y, Z. Each sub-directory contains various images of the hand gesture of a user displaying a sign language for the corresponding label. A model is created which will recognize the input hand signs and will display the corresponding letter of the sign language. The model is being trained with some samples of data from the database of the images of the words from each word. After the model is trained it can be tested to observe the actual working of the system.

## CHAPTER 3

# ARCHITECTURE AND SYSTEM DESIGN

### 3.1 ARCHITECTURE

#### CNN ARCHITECTURE COMPONENTS:

The objective of CNN is to learn the features present in the data with higher order using convolutions. The CNN architecture works well for the recognition of objects which includes images. They can recognize individuals, faces, street signs and other facets of visual data. There exist a number of CNN variations, but each of them is based on the pattern of layers present, as shown in Fig. 3.1 CNN architecture consists of different components which include different types of layers and activation functions.

**Convolutional layer :** The core building blocks of CNN architecture are the convolutional layer. Convolutional layers (Conv) modify the input data with the help of a patch of neurons connected locally from the previous layer. The dot product will be computed by the layer between the region of the neurons present in the input layer and the weights to which they are locally connected present in the output layer. A convolution is a mathematical operation that describes the rule for merging two sets of information. The convolution operation takes input, applies a convolution filter or kernel, and returns a feature map as an output as shown in Fig. 3.2. This operation demonstrates the sliding of the kernel across the input data which produces the convoluted output data. At each step, the input data values are multiplied by the kernel within its boundaries and a single value in the output feature map is created.

Let us suppose the frame size of an input image  $W \in R^{w \times h}$ . The convolutional filter with size F is used for convolution with a stride of S and P padding for input

image boundary. The size of the output of the convolution layer is presented by Eq. (1).

$$\text{Output} = \frac{W - F + 2P}{S} + 1 \quad (1)$$

For example, there is one neuron with a receptive field size of  $F = 3$ , the input size is  $W = 128$ , and there is zero padding of  $P = 1$ . The neuron stride across the input in stride of  $S = 1$ , giving output of size  $(128 - 3 + 2)/1 + 1 = 128$ . The output of a convolutional layer is denoted with Standardized Eq.2

$$a_j^n = f \left( \sum_{i \in C_j} y_i^{n-1} * k_{ij}^n + b_j^n \right) \quad (2)$$

where  $*$  is the convolution operation,  $n$  represents the  $n$ th layer,  $a_j^n$  is the  $j$ th output map,  $y_i^{n-1}$  represents the  $i$ th input map in the  $n-1$ th layer, the convolutional kernel is represented by  $k_{ij}$ ,  $b_j$  represents bias,  $C_j$  is used for representing input maps and  $f$  is an activation function. For example, suppose that the input volume has size  $[128 * 128 * 3]$ . If the filter size is  $3 * 3$ , then each neuron in the convolution layer will have weights to a  $[3 * 3 * 3]$  region in the input volume, for a total of  $3*3*3 = 27$  weights and  $+1$  bias parameter. The main objective of other feature extraction layers is to reduce the dimensions of the output generated by convolutional layers. After convolution, the max-method will be used over a region with some specific size for subsampling of feature map. This operation is given by Eq. (3).

$$a_j^n = s(a_i^{n-1}), \quad \forall i \in V_j \quad (3)$$

where  $s$  is the subsampling operation and  $V_j$  is the  $j$ th region of subsampling in the  $n$ th input map [10].

**Pooling layer** : Pooling layers help in reducing the representation of data gradually over the network and control over-fitting. The pooling layer operates in an independent manner on every depth slice of the input. The max () operation used by the pooling layer helps in the resizing of the input data spatially (width, height). This operation is called as max-pooling. The down-sampling in this layer has been performed using filters on the input data. For example, the input volume of size [126 \* 126 \* 16] is pooled with filter size 2, stride 2 into output volume of size [63 \* 63 \* 16].

**ReLU layer** : ReLU stands for Rectified Linear Units. The ReLU layer helps in applying an element-wise activation function over the input data thresholding, for example,  $\max(0, x)$  at zero, giving the same dimension output as the input to the layer. The usage of ReLU layers does not affect the receptive field of the convolution layer and at the same time provides nonlinearity to the network. This nonlinear property of the function helps in the better generalization of the classifier. The nonlinear function  $f(x)$  used in the ReLU layer is shown in Eq. (4).

$$f(x) = \max(0, x) \quad (4)$$

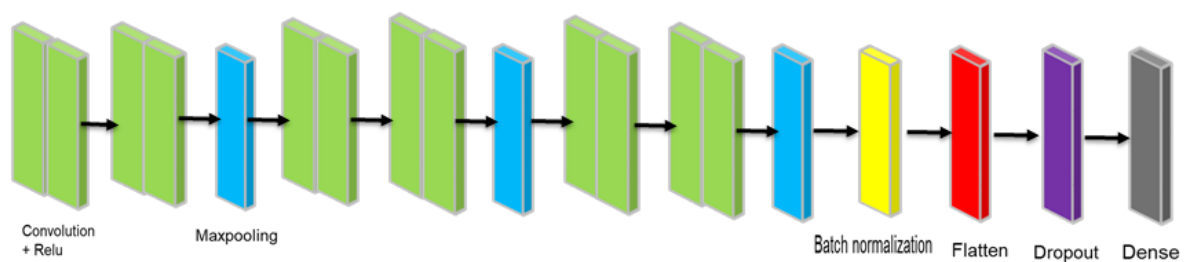
The sigmoid function and hyperbolic tangent are some other activation functions that can also be used to influence nonlinearity in the network. The usage of ReLU is preferred because the derivative of the function helps backpropagation work considerably faster without making any noticeable difference to generalization accuracy.

**Fully connected layer/output layer** : Fully connected layer is used to compute scores of different features for classification. The dimensions of the output volume are  $[1 * 1 * N]$ , where  $N$  represents the number of output classes to be evaluated. Each output neuron is connected with all other neurons in the previous layer with different sets of weights. Furthermore, the fully connected layer is a

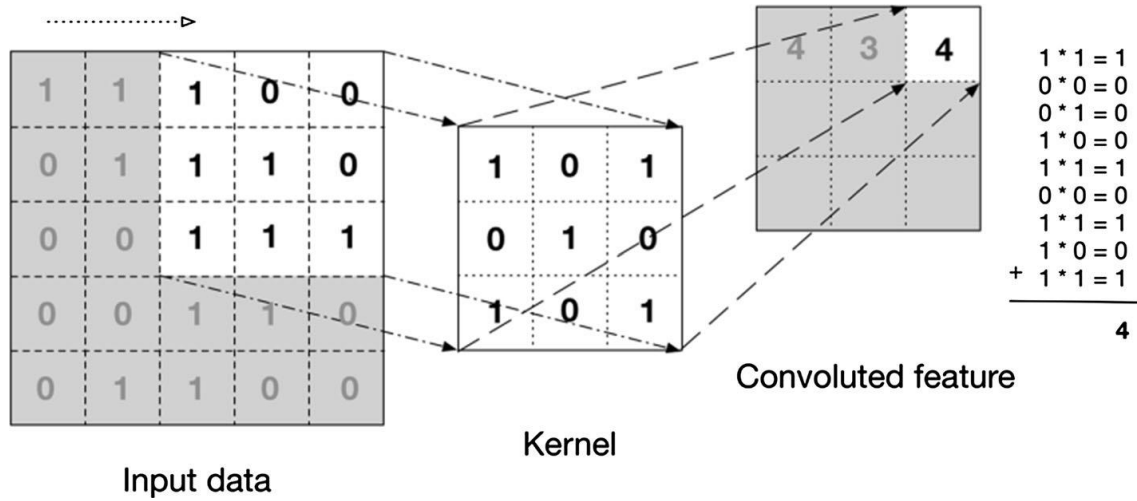
set of convolutions in which each feature map is connected with every field of the consecutive layer and filters consist of the same size as that of the input image .For example, a fully connected layer with  $[63 * 63 * 16]$  volume and a convolutional layer use filter size 16, giving output volume  $[1 * 1 * 63,504]$ .

The final and last layer is the classification layer. As this sign language recognition is a multi-classification problem, softmax function is used in the output layer for classification.

Generally, the CNN architecture consists of four main layers that are a convolutional layer, the pooling layer, the ReLU layer and the fully connected or output layer. The proposed CNN architecture include 3 groups of 2 convolutional layers followed by a max-pool layer, a dropout layer and a flatten layer, and two fully connected layer and one final output layer. To enhance the effectiveness of the results, dropout layer, is added in the proposed approach, which is a regularization technique used to ignore randomly selected neurons at the time of training and it helps in reducing the chances of over-fitting.



***Fig 3.1 CNN architecture***



*Fig. 3.2 The convolution operation*

## 3.2 DESCRIPTION OF MODULES

The proposed sign language recognition system includes four major phases that are data acquisition, image preprocessing, training and testing of the CNN classifier. The first phase is the data acquisition phase, in which the RGB data of static signs get collected using a camera. The collected sign images are then preprocessed using image resizing and normalization. These normalized images are stored in the data store for future use. In the next phase, the proposed system gets trained using CNN classifier and then the trained model is used to perform testing. The last phase is the testing phase in which the CNN architecture parameters are fine-tuned until the results match the desired accuracy.

### 3.2.1 DATA ACQUISITION

The training data set from kaggle contains 87,000 images which are 200x200 pixels. There are 29 classes, of which 26 are for the letters A-Z and 3 classes for *SPACE*, *DELETE* and *NOTHING*. The three-channel image frames (RGB) are retrieved from the camera, and then these images are passed to the image preprocessing module. The dataset consists of the collection of the RGB

images for different static signs. The dataset comprises 14,500 images which include 500 images for each of the static signs. There are 29 distinct sign classes that include 26 alphabets of English and SPACE, DELETE and NOTHING. The dataset consists of static sign images with various sizes, colors and taken under different environmental conditions to assist in the better generalization of the classifier.

### **3.2.2 DATA PREPROCESSING**

The data preprocessing is the application of different morphological operations that are used to remove noise from the data. In this phase, the sign images are preprocessed using two methods that are image resizing and normalization. In image resizing, the image is resized to 64 \* 64. These images are then normalized to change the range of pixel intensity values which results in mean 0 and variance 1.

### **3.2.3 MODEL TRAINING**

The proposed model is trained on the dataset of different ASL signs. The classifier takes the preprocessed sign images and classifies it into the corresponding category. The dataset is shuffled and divided into training and validation set by splitting original data into 95% of training data and 5% testing data. Shuffling the dataset is very significant in terms of adding randomness to the process of neural network training which prevents the network from being biased toward certain parameters. The configuration of the CNN architecture used in the proposed system is described in Table 3.3.



**Model: "sequential"**

Layer (type)	Output Shape	Param #
conv2d (Conv2D)	(None, 64, 64, 16)	448
conv2d_1 (Conv2D)	(None, 64, 64, 32)	4640
max_pooling2d (MaxPooling2D)	(None, 21, 21, 32)	0
conv2d_2 (Conv2D)	(None, 21, 21, 32)	9248
conv2d_3 (Conv2D)	(None, 21, 21, 64)	18496
max_pooling2d_1 (MaxPooling2D)	(None, 7, 7, 64)	0
conv2d_4 (Conv2D)	(None, 7, 7, 128)	73856
conv2d_5 (Conv2D)	(None, 7, 7, 256)	295168
max_pooling2d_2 (MaxPooling2D)	(None, 2, 2, 256)	0
batch_normalization (Batch Normalization)	(None, 2, 2, 256)	1024
flatten (Flatten)	(None, 1024)	0
dropout (Dropout)	(None, 1024)	0
dense (Dense)	(None, 512)	524800
dense_1 (Dense)	(None, 29)	14877

***Table 3.3 Proposed System Architecture***

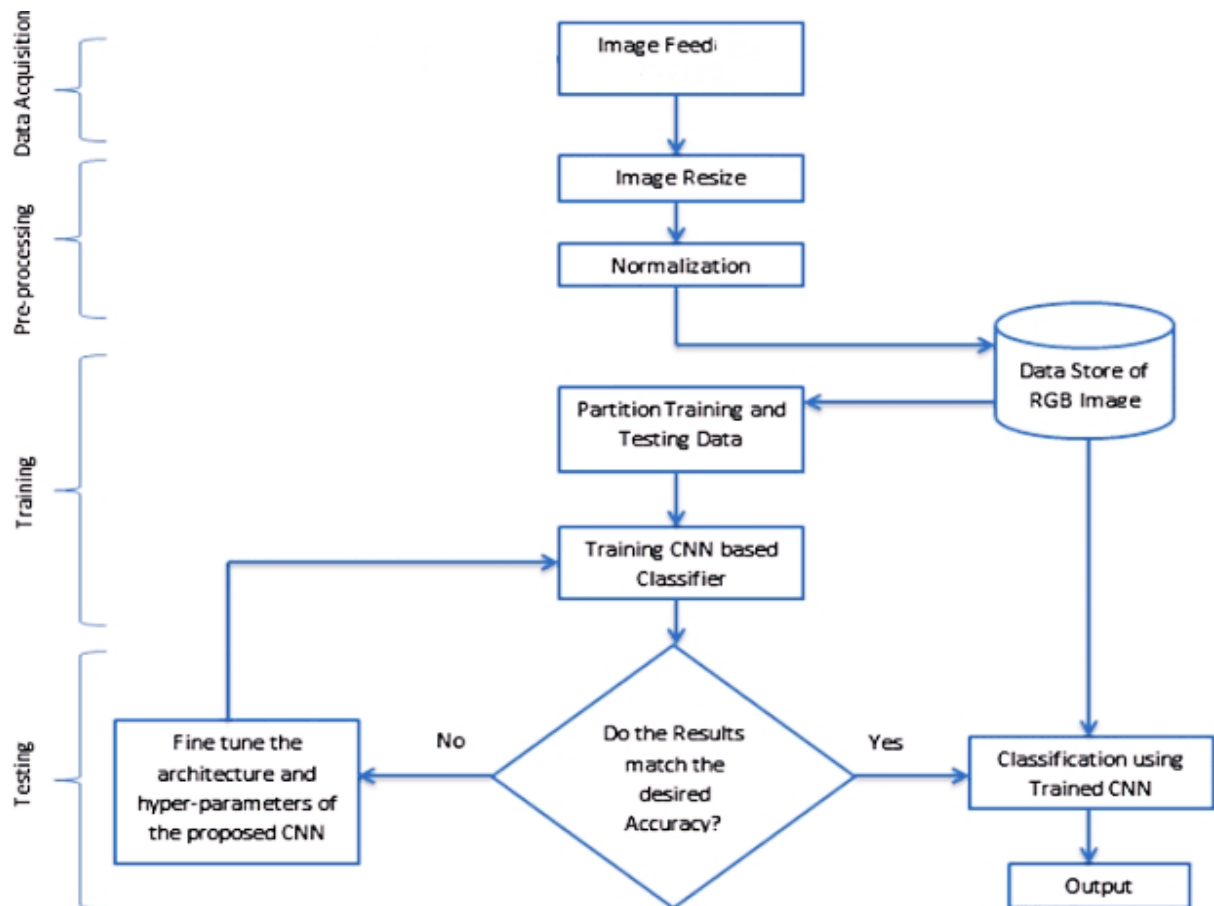
First convolutional layer accepts image input. Then, add a max pooling layer. Pooling passes a (3, 3) moving window over the image and downscales the image by outputting the maximum value within window. Add further convolutional layers : deeper models with more convolutional layers,are better able to learn features from images. Flatten flattens the output into a one-dimensional feature vector which can be passed into the following fully connected layers. Dropout prevents the model from perfectly remembering each image, by randomly setting a percentage of the input units to 0 at each update during training. Then,the model is fitted.

### **3.2.4 TESTING**

To check model performance, the model performance metrics Accuracy plot - train vs test and loss plot -training vs validation are plotted. The trained model on the test data split using `model.evaluate()` function is evaluated. The confusion matrix and evaluate the model performance is plotted. Then the test data is loaded and predictions are made using `model.predict_classes()` function. The test data along with prediction made by the model is plotted.

### **3.3 FLOW CHART**

The general flow diagram of system has been shown in Fig. 3.4



*Fig 3.4 Flow chart of the proposed system*

## CHAPTER 4

### SYSTEM IMPLEMENTATION

#### 4.1 SIMULATION

##### 4.1.1 DATA ACQUISITION

###### Import Libraries

*# Utils*

`import numpy as np`

`import pandas as pd`

`import os`

*# Visualization*

```

import keras
import matplotlib.pyplot as plt
import seaborn as sns
# Imports for CNN model
from keras.layers import Conv2D, MaxPool2D, Flatten, Dense, Dropout, Batch
Normalization
from keras.preprocessing.image import ImageDataGenerator
from keras.models import Sequential
from keras import regularizers
from sklearn.model_selection import train_test_split
# Imports to view data
import cv2
print(os.listdir('/content/drive/My Drive/Dataset'))

```

### **Plotting one image from each class**

```

def load_unique():
    size_img = 64,64
    images_for_plot = []
    labels_for_plot = []
    for folder in os.listdir(train_dir):
        for file in os.listdir(train_dir + '/' + folder):
            filepath = train_dir + '/' + folder + '/' + file
            image = cv2.imread(filepath)
            final_img = cv2.resize(image, size_img)
            final_img = cv2.cvtColor(final_img, cv2.COLOR_BGR2RGB)
            images_for_plot.append(final_img)
            labels_for_plot.append(folder)
        break
    return images_for_plot, labels_for_plot
images_for_plot, labels_for_plot = load_unique()

```

```

print("unique_labels = ", labels_for_plot)

fig = plt.figure(figsize = (15,15))
def plot_images(fig, image, label, row, col, index):
    fig.add_subplot(row, col, index)
    plt.axis('off')
    plt.imshow(image)
    plt.title(label)
    return
image_index = 0
row = 5
col = 6
for i in range(1,(row*col)-1):
    plot_images(fig, images_for_plot[image_index], labels_for_plot[image_index], row, col, i)
    image_index = image_index + 1
plt.show()
unique_labels = ['B', 'A', 'C', 'D', 'E', 'F', 'G', 'H', 'I', 'del', 'Q', 'R', 'O', 'N', 'M', 'L', 'K', 'nothing', 'J', 'P', 'Z', 'Y', 'X', 'W', 'V', 'U', 'T', 'space', 'S']

```



*Fig 4.1 Plotting one image from each class*

## 4.1.2 DATA PREPROCESSING

### Loading data

1. Defining a dictionary which contains labels and its mapping to a number which acts as class label.

```
labels_dict = {'A':0,'B':1,'C':2,'D':3,'E':4,'F':5,'G':6,'H':7,'I':8,'J':9,'K':10,'L':11,'M':12,'N':13,'O':14,'P':15,'Q':16,'R':17,'S':18,'T':19,'U':20,'V':21,'W':22,'X':23,'Y':24,'Z':25,'space':26,'del':27,'nothing':28}
```

2. Loading image data and labels and preprocessing ,then mapping those labels to the dictionary defined before.

```
def load_data():
    images = []
    labels = []
    size = 64,64
    print("LOADING DATA FROM : ",end = "")
    for folder in os.listdir(train_dir):
        print(folder, end = ' | ')
        for image in os.listdir(train_dir + "/" + folder):
            temp_img = cv2.imread(train_dir + "/" + folder + "/" + image)
            temp_img = cv2.resize(temp_img, size)
            images.append(temp_img)
            labels.append(labels_dict[folder])
```

3. Normalizing image data

```
images = np.array(images)
```

```
images = images.astype('float32')/255.0
```

4. Converting labels to one hot format.

```
labels = keras.utils.to_categorical(labels)
```

5. Creating training and test data by splitting original data into 95% of training data and 5% testing data.

```
X_train, X_test, Y_train, Y_test = train_test_split(images, labels, test_size = 0.05)
```

```
print()
```

```
print('Loaded', len(X_train), 'images for training', 'Train data shape =', X_train.shape)
```

```
print('Loaded', len(X_test), 'images for testing', 'Test data shape =', X_test.shape)
```

```
return X_train, X_test, Y_train, Y_test
```

### 4.1.3 MODEL TRAINING

**Compiling the model and fitting the training data**

```
model = create_model()
```

```
curr_model_hist = fit_model()
```

**Plotting the model performance metrics to check model performance.**

```
plt.plot(curr_model_hist.history['acc'])
```

```
plt.plot(curr_model_hist.history['val_acc'])
```

```
plt.legend(['train', 'test'], loc='lower right')
```

```
plt.title('accuracy plot - train vs test')
```

```
plt.xlabel('epoch')
```

```
plt.ylabel('accuracy')
```

```

plt.show()

plt.plot(curr_model_hist.history['loss'])

plt.plot(curr_model_hist.history['val_loss'])

plt.legend(['training loss', 'validation loss'], loc = 'upper right')

plt.title('loss plot - training vs vaidation')

plt.xlabel('epoch')

plt.ylabel('loss')

plt.show()

```

### **Evaluating the trained model on the test data split**

```

score = model.evaluate(x = X_test, y = Y_test, verbose = 0)

print('Accuracy for test images:', round(score[1]*100, 3), '%')

print("\nEvaluation loss = " , "{:.6f}".format(score[0]))

```

### **Confusion matrix**

```

cm = sklearn.metrics.confusion_matrix(ytest_conv, ypred)

cm

sklearn.metrics.accuracy_score(ytest_conv, ypred)

def plot_confusion_matrix(cm, target_names, title='Confusion matrix', cmap=plt.cm.Blues):

    plt.imshow(cm, interpolation='nearest', cmap=cmap)

    plt.title(title)

    plt.colorbar()

    tick_marks = np.arange(len(target_names))

    plt.xticks(tick_marks, target_names, rotation=45)

```



```

plt.yticks(tick_marks, target_names)

plt.tight_layout()
width, height = cm.shape
for x in range(width):
    for y in range(height):
        plt.annotate(str(cm[x][y]), xy=(y, x),
horizontalalignment='center',
                    verticalalignment='center')

plt.ylabel("True label")
plt.xlabel("Predicted label")
#cm = np.array([[13, 0, 0],[ 0, 10, 6],[ 0, 0, 9]])

plt.figure(figsize=(20,20))
plot_confusion_matrix(cm, set(ytest_conv))

```

#### 4.1.4 TESTING THE MODEL

##### Loading the test data

The test data contains images with the image name as the class, to which the image belongs to.

We will load the data and check if the prediction is correct for the image.

```

def load_test_data():
    images = []
    names = []
    size = 64,64
    for image in os.listdir(test_dir):
        temp = cv2.imread(test_dir + '/' + image)

```

```

temp = cv2.resize(temp, size)

images.append(temp)

names.append(image)

images = np.array(images)

images = images.astype('float32')/255.0

return images, names

test_images, test_img_names = load_test_data()

```

### **Make predictions on an image and append it to the list (predictions)**

```

predictions = [model.predict_classes(image.reshape(1,64,64,3))[0]

for image in test_images]

def get_labels_for_plot(predictions):

    predictions_labels = []

    for i in range(len(predictions)):

        for ins in labels_dict:

            if predictions[i] == labels_dict[ins]:

                predictions_labels.append(ins)

                break

    return predictions_labels

predictions_labels_plot = get_labels_for_plot(predictions)

```

### **Plotting the test data along with the prediction made by the model**

```

predfigure = plt.figure(figsize = (13,13))

def plot_image_1(fig, image, label, prediction, predictions_label, row, col, index):

fig.add_subplot(row, col, index)

```

```

plt.axis('off')

plt.imshow(image)

title = "prediction : [" + str(predictions_label) + "]" + "\n" + label

plt.title(title)

return

image_index = 0
row = 5
col = 6
for i in range(1,(row*col-1)):

    plot_image_1(predfigure, test_images[image_index], test_img_names[image_index], predictions[image_index], predictions_labels_plot[image_index], row, col, i)

    image_index = image_index + 1

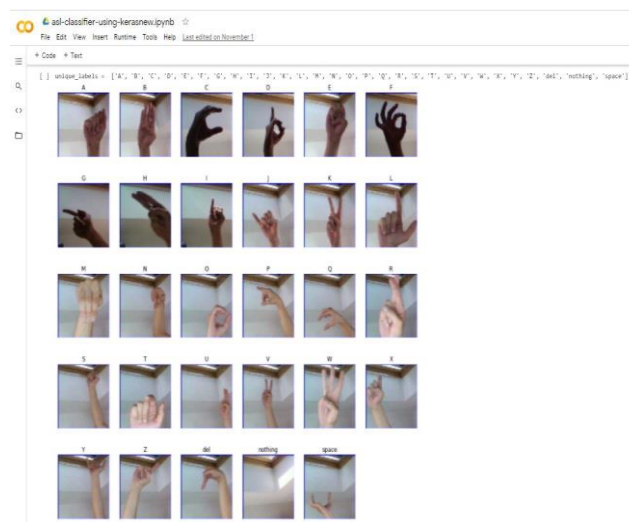
plt.show()

```

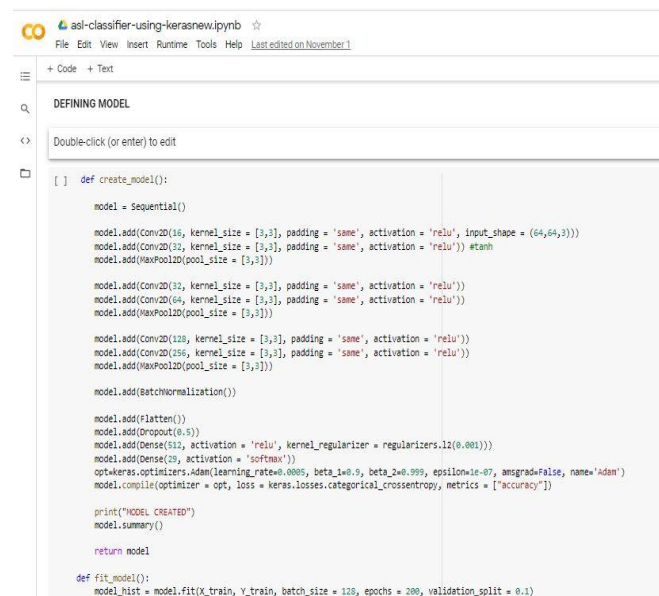
## 4.2 SOFTWARE REQUIREMENTS

### 4.2.1 GOOGLE COLAB

Colaboratory is a Google research project created to help disseminate machine learning education and research. Colaboratory, or “Colab”



for short, is a product from Google Research. Colab allows anybody to write and execute arbitrary python code through the browser, and is especially well suited to machine learning, data analysis and education. More technically, Colab is a hosted Jupyter notebook service that requires no setup to use, while providing free access to computing resources including GPUs. Colab notebooks are stored in Google Drive, or can be loaded from GitHub. Colab notebooks can be shared just as you would with Google Docs or Sheets. Simply click the Share button at the top right of any Colab notebook, or follow these Google Drive file sharing instructions. Code is executed in a virtual machine private to your account. The amount of memory available in Colab virtual machines varies over time. You may sometimes be automatically assigned a VM with extra memory when Colab detects that you are likely to need it. Users interested in having more memory available to them in Colab, and more reliably, may be interested in Colab Pro. Fig 4.2 shows Google Colab platform.



```
[ ] def create_model():  
    model = Sequential()  
  
    model.add(Conv2D(16, kernel_size = [3,3], padding = 'same', activation = 'relu', input_shape = (64,64,3)))  
    model.add(Conv2D(32, kernel_size = [3,3], padding = 'same', activation = 'relu')) #tanh  
    model.add(MaxPool2D(pool_size = [3,3]))  
  
    model.add(Conv2D(32, kernel_size = [3,3], padding = 'same', activation = 'relu'))  
    model.add(Conv2D(64, kernel_size = [3,3], padding = 'same', activation = 'relu'))  
    model.add(MaxPool2D(pool_size = [3,3]))  
  
    model.add(Conv2D(128, kernel_size = [3,3], padding = 'same', activation = 'relu'))  
    model.add(Conv2D(256, kernel_size = [3,3], padding = 'same', activation = 'relu'))  
    model.add(MaxPool2D(pool_size = [3,3]))  
  
    model.add(BatchNormalization())  
  
    model.add(Flatten())  
    model.add(Dropout(0.5))  
    model.add(Dense(512, activation = 'relu', kernel_regularizer = regularizers.l2(0.001)))  
    model.add(Dense(29, activation = 'softmax'))  
    opt=keras.optimizers.Adam(learning_rate=0.0005, beta_1=0.5, beta_2=0.999, epsilon=1e-07, amsgrad=False, name='Adam')  
    model.compile(optimizer = opt, loss = keras.losses.categorical_crossentropy, metrics = ['accuracy'])  
  
    print("MODEL CREATED")  
    model.summary()  
  
    return model  
  
def fit_model():  
    model_hist = model.fit(X_train, Y_train, batch_size = 128, epochs = 200, validation_split = 0.1)
```

***Fig 4.2 Google colab platform***

## **4.2.2 TENSORFLOW**

TensorFlow is a Python library for fast numerical computing created and released by Google. It is a foundation library that can be used to create Deep Learning models directly or by using wrapper libraries that simplify the process built on top of TensorFlow. It is an symbolic math library, and is also used for machine learning application such as neural networks, etc. It is mainly used for classification, perception, understanding, discovering, prediction and creation. It is Google Brain's second-generation system. 1st Version of tensorflow was released on February 11, 2017. While the reference implementation runs on single devices, Tensorflow can run on multiple CPU's and GPU. This is available on various platforms such as 64-bit Linux, macOS, Windows, Android and iOS. The architecture of tensorflow allows the easy deployment of computation across a variety of platforms (CPU's, GPU's, TPU's), and from desktops - clusters of servers to mobile and edge devices. Tensorflow computations are expressed as stateful dataflow graphs.

#### **4.2.3 KERAS**

Keras is an open-source neural-network library written in Python. It is capable of running on top of TensorFlow, Microsoft Cognitive Toolkit, R, Theano, or PlaidML. It is designed to enable fast experimentation with deep neural networks, it focuses on being user-friendly, modular, and extensible. Keras is an API designed for human beings, not machines. Keras follows best practices for reducing cognitive load that is it offers consistent & simple APIs, and it provides clear & actionable error messages. It also has extensive documentation and developer guides. 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.

#### **4.2.4 OPEN CV**

OpenCV (Open Source Computer Vision Library) is an open source computer vision and machine learning software library. OpenCV is an library of programming functions mainly aimed on real time computer vision. originally developed by Intel, it is later supported by Willow Garage then Itseez.. It has C++, Python, Java and MATLAB interfaces and supports Windows, Linux, Android and Mac OS. OpenCV leans mostly towards real-time vision applications. OpenCV is written natively in C++ and has a templated interface that works seamlessly with STL containers. The library has more than 2500 optimized algorithms. These algorithms can be used to detect and recognize faces, identify objects, classify human actions in videos, track camera movements, track moving objects, stitch images together to produce a high resolution image of an entire scene, find similar images from an image database, follow eye movements, recognize scenery, etc.

#### **4.2.5 NUMPY**

NumPy is library of Python programming language, adding support for large, multi-dimensional array and matrice, along with large collection of high-level mathematical function to operate over these arrays. The ancestor of NumPy, Numeric, was originally created by Jim Hugunin with contributions from several developers. In 2005 Travis Olphant created NumPy by incorporating features of computing Numarray into Numeric, with extension modifications. NumPy is open-source software and has many contributors. NumPy targets the CPython reference implementation of Python, which is a non-optimizing bytecode interpreter. Using NumPy in Python gives functionality comparable to MATLAB since they are both interpreted and they both allow the user to write fast programs as long as most operations work on arrays or matrices

instead of scalars. The NumPy array as universal data structure in OpenCV for images, extracted feature points, filter kernels and many more vastly simplifies the programming workflow and debugging.

#### 4.2.6 PANDAS

In computer programming, **pandas** is a software library written for the Python programming language for data manipulation and analysis. In particular, it offers data structures and operations for manipulating numerical tables and time series. It is free software released under the three-clause BSD license. The name is derived from the term "panel data", an econometrics term for data sets that include observations over multiple time periods for the same individuals. 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. *Pandas* aims to be the fundamental high-level building block for doing practical, real world data analysis in Python.

#### 4.2.7 SKLEARN

**Scikit-learn** (formerly **scikits.learn** and also known as **sklearn**) is a free software machine learning library for the Python programming language. It features various classification, regression and clustering algorithms including support vector machines, random forests, gradient boosting, *k*-means and DBSCAN, and is designed to interoperate with the Python numerical and scientific libraries NumPy and SciPy. Scikit-learn is largely written in Python, and uses numpy extensively for high-performance linear algebra and array operations. Support vector machines are implemented by a Cython wrapper around LIBSVM; logistic regression and linear support vector machines by a

similar wrapper around LIBLINEAR. Scikit-learn integrates well with many other Python libraries, such as matplotlib and plotly for plotting, numpy for array vectorization, pandas dataframes, scipy, and many more.

## **CHAPTER 5**

### **RESULTS AND DISCUSSIONS**

#### **5.1 RESULTS**

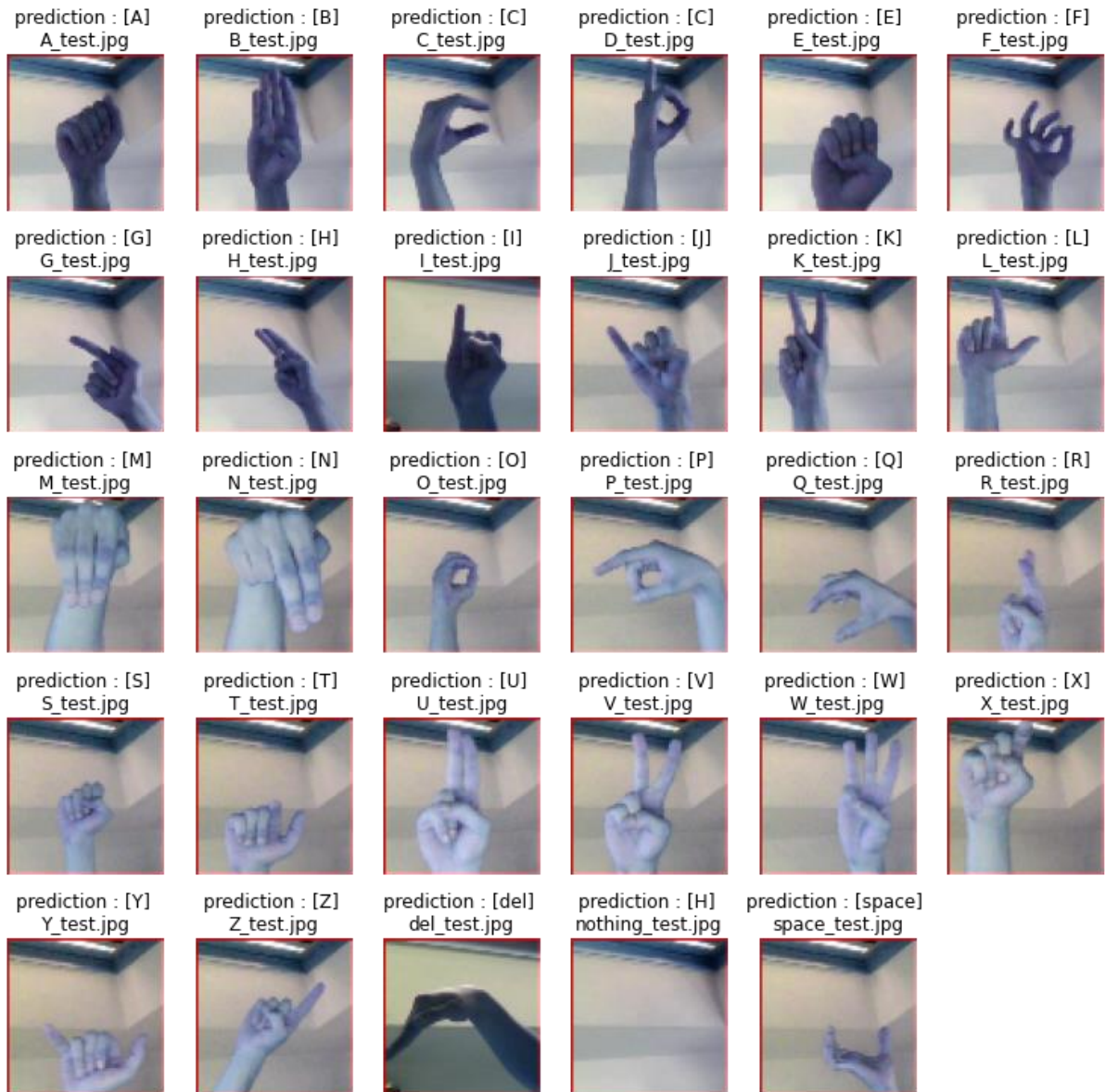
The trained sequential model can be used to detect the sign in the test image fed to the system. Our model exhibits best performance for 50 epochs. Our model was able to achieve an evaluation accuracy of 99.587% with evaluation loss of 0.029690 for 50 epochs. Fig 5.1 shows the prediction made by our model on test data set. As we can see from the output, the predictions are correct for almost all the images. Thus the model performs well on test data-set.

We have also tested our model against real world test data set. Our model was able to achieve an evaluation accuracy of 89.67% with evaluation loss of 0.627641 on real world test data. Fig 5.2 shows the prediction made by our model on real world images. From the figure, we see the model has predicted



some images wrongly. We infer that the model loses its efficiency when making predictions on real world images. This is mainly because the real world images are captured under uncontrolled lighting conditions and also because the model does not include skin colour thresholding block in the preprocessing part.

The real world images can be uploaded in drive and the model can read the images at run time or can also be captured using webcam during run time. We have included a block which will capture image using webcam and feed it to the system which will further predict the sign. This features real-time implementation of SLR system. Thus our system can operate as a real-time sign language translator.



***Fig 5.1 Prediction made by the model on given test data set***



***Fig 5.2 Prediction made by the model on real world test images***

## **RESULTS:**

### **Test Data set**

Evaluation Accuracy = 99.587%

Evaluation loss = 0.029690

### **Real world images dataset**

Evaluation Accuracy = 89.67%

Evaluation loss = 0.627641

## 5.2 EVALUATION

Deep CNN models are evaluated using accuracy plot, loss plot and confusion matrix. **Accuracy** is one metric for evaluating classification models. Informally, accuracy is the fraction of predictions our model got right. Formally, accuracy has the following definition:

$$\text{Accuracy} = \frac{\text{Number of correct predictions}}{\text{Total number of predictions}}$$

Typically, a model is said to have a good performance if its accuracy is greater than 90%

The **Loss Function** is one of the important components of Neural Networks. **Loss** is nothing but a prediction error of Neural Net and the method to calculate the loss is called Loss Function. In simple words, the Loss is used to calculate the gradients and gradients are used to update the weights of the Neural Net. This is how a Neural Net is trained. **Keras** and **Tensorflow** have various inbuilt loss functions for different objectives. We trained our model with a categorical cross entropy loss function.

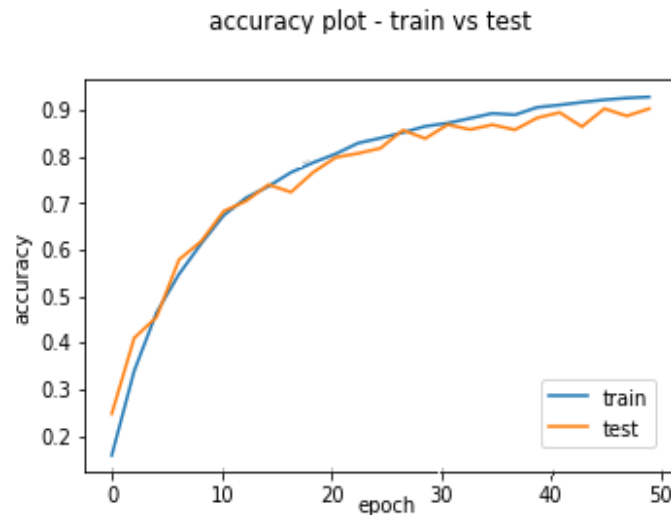
$$H(p, q) = - \sum_x p(x) \log(q(x))$$

When we have a multi-class classification task, one of the loss function you can go ahead is this one. If you are using categorical cross entropy loss function, there must be the same number of output nodes as the classes and the final layer output should be passed through a *softmax* activation so that each node output a probability value between (0–1).

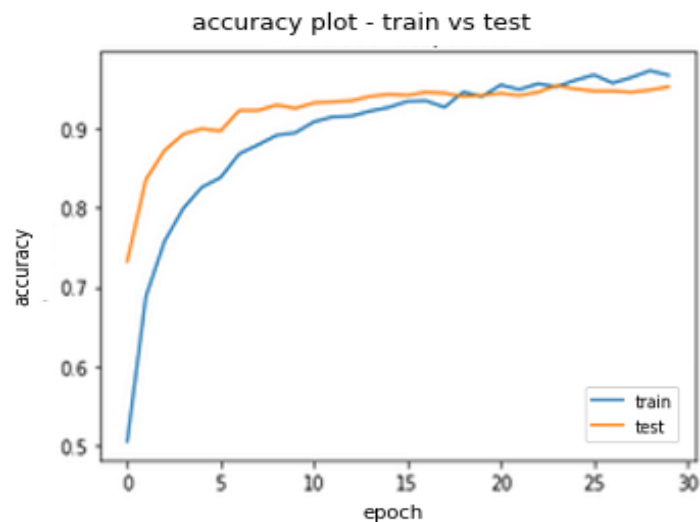
Initially, we observed low accuracy measures when testing on the validation data set but as we increased the epoch our model was able to perform with a higher accuracy. By decreasing the learning rate, we were able to avoid model over-fitting.

### 5.2.1 EVALUATION ACCURACY

During the training of the network the evaluation accuracy value for different epochs are plotted as shown in the graph. Fig 5.3a shows the accuracy plot - train vs test for given data set. Fig 5.3b shows the accuracy plot - train vs test for real world test images.



***Fig 5.3a Accuracy plot – train vs test \_\_ given test data set***



***Fig 5.3b Accuracy plot – train vs test \_\_ real world test data***

It can be observed from the graph that as the number of epochs are increased the evaluation accuracy value increases and thus the performance of the model increases.

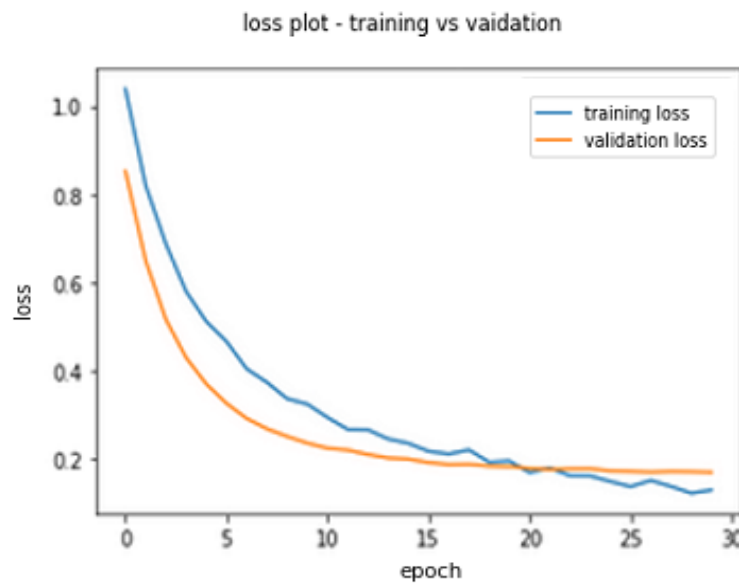
### 5.2.2 LOSS FUNCTION

During the training of the network the loss function value for different epochs are plotted as shown in the graph. Fig 5.4a shows the loss plot - training vs validation for given data set. Fig 5.4b loss plot - training vs validation for real world test images.

It can be observed from the graph that as the number of epochs are increased the loss function value decreases and thus the performance of the model increases.



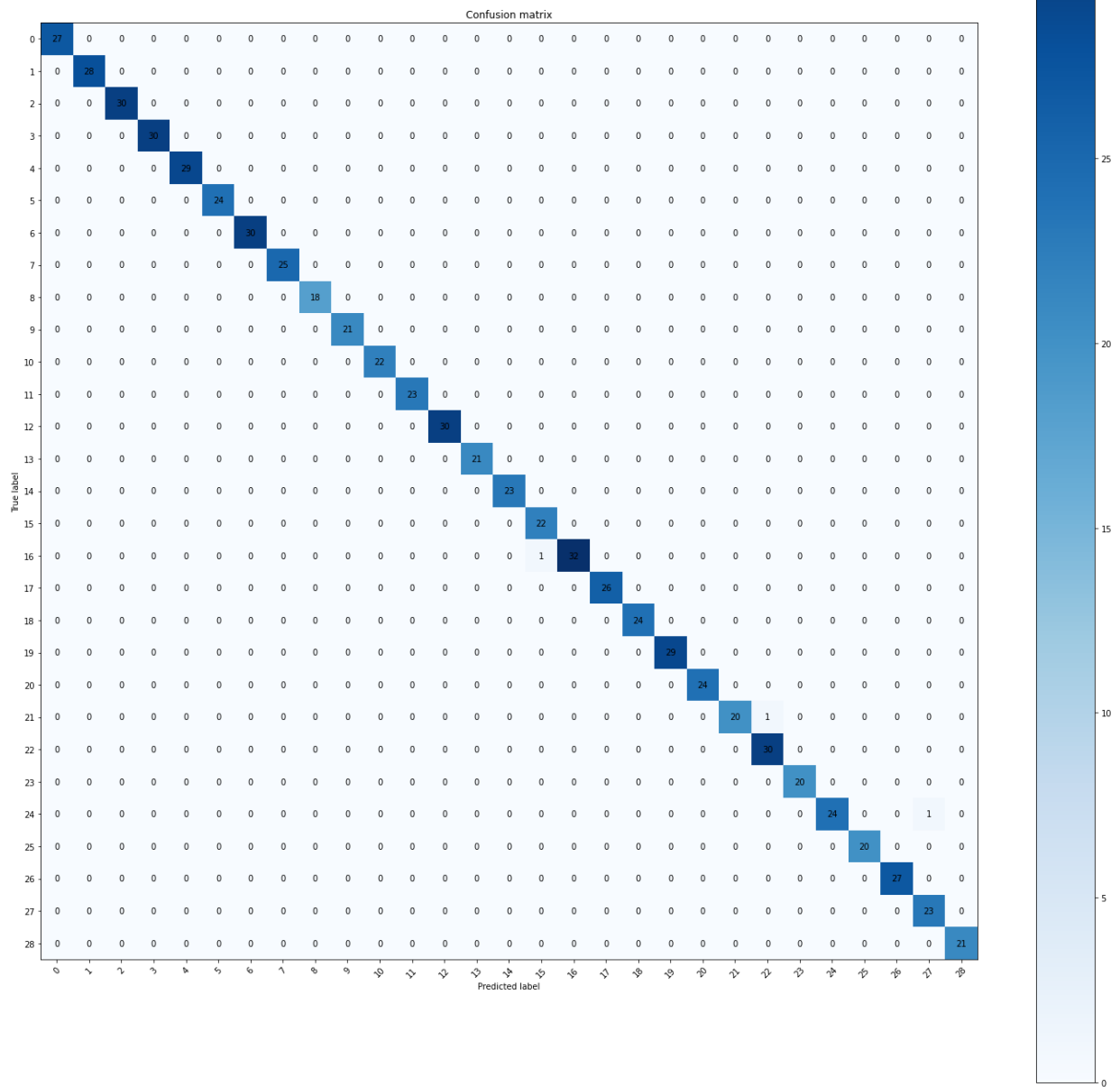
***Fig 5.4a Loss plot – training vs validation \_\_ given test data set***



***Fig 5.4b Loss plot – training vs validation \_\_ real world test data***

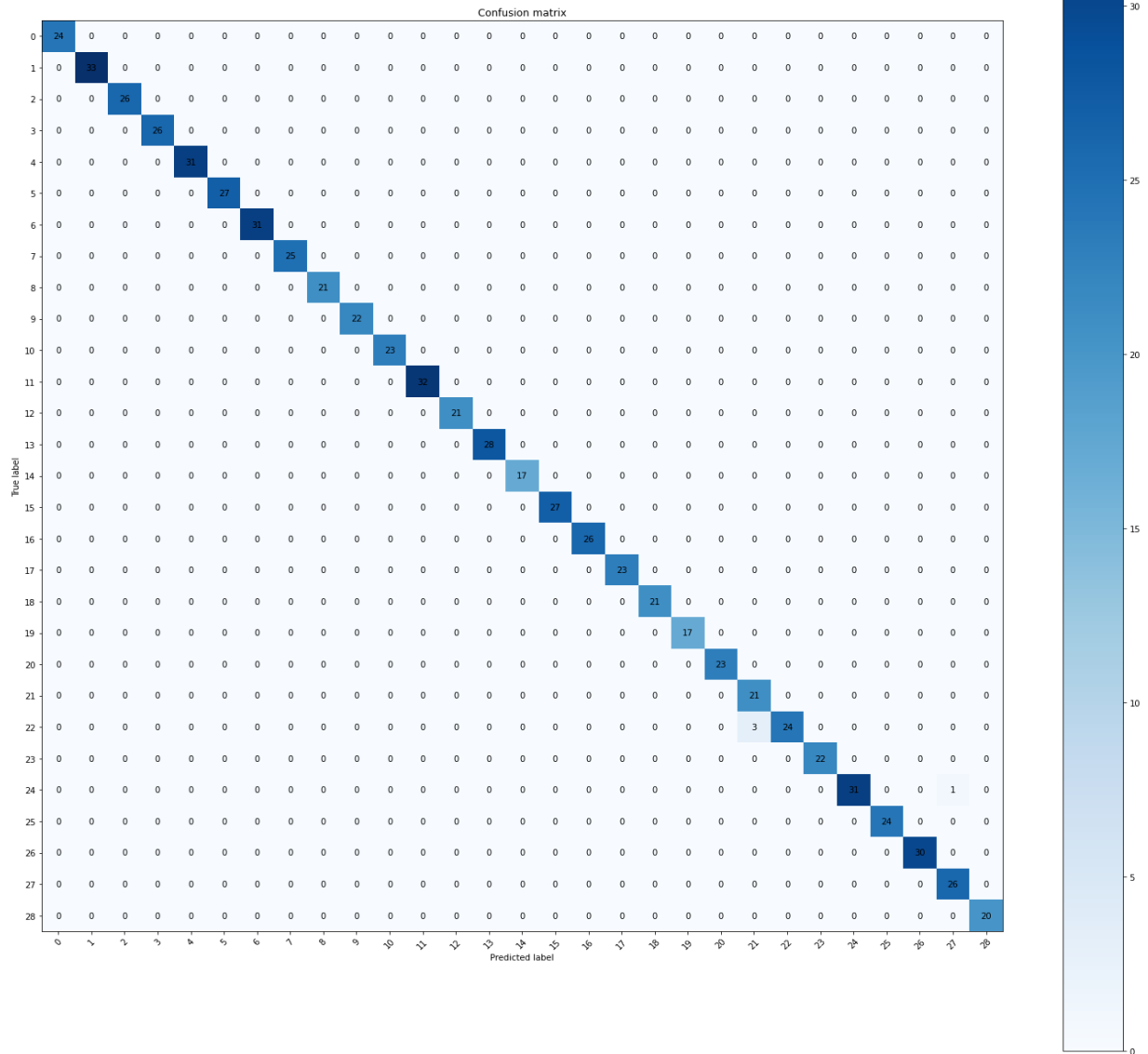
### 5.2.3 CONFUSION MATRIX

Confusion matrix is a matrix or table used to describe the performance of a classification model on a set of test data for which the true values are already known i.e., the validation split set. The confusion matrix shows the number of correct predictions and incorrect predictions made by a classification model. The columns of a confusion matrix represent actual values or true labels and the rows of the confusion matrix represent predicted labels. A model is said to perform well only when the confusion matrix has values only where row index is equal to the column index i.e., along the diagonal elements. Fig 5.5a shows the confusion matrix obtained for given test data set. Fig 5.5b shows the confusion matrix obtained for real world test data. From the obtained confusion matrix we can infer that our model has good performance since values are present only along the diagonal.



**Fig 5.5a Confusion Matrix \_\_ given test data set**





*Fig 5.5b Confusion Matrix \_\_ real world test data*

## 5.2.4 MODEL COMPARISON

Table 5.6 shows the different CNN models used by different sign language recognition systems and the accuracy achieved.

MODEL	ACCURACY
[1] CNN model with 4 groups of 2 convolutional layers followed by a max-pool layer, a dropout layer and a flatten layer, and two fully connected layer and one final output layer.	99.17%
[2] CNN architecture included 3 groups of 2 convolutional layers followed by a max-pool layer and a dropout layer, and two groups of fully connected layer followed by a dropout layer and one final output layer.	82.5% in alphabet recognition 97% in digit recognition
[3] CNN architecture included convolutional layer with 16 filters, each of which has a $2 \times 2$ kernel. Then, a $2 \times 2$ Max Pooling filter followed by Dropout(0.2) functions. Then, a flattening layer and dense layer with rectified linear activation and with SoftMax classifier.	90.04% in letter recognition, 93.44% in number and 97.52% in static word recognition
[4] Selfie-based sign language recognition system using Deep CNN, adopted mean pooling, max-pooling and stochastic pooling strategies on CNN	92.88%
[5] Two stream CNN architecture, which takes two color-coded images the joint distance topographic descriptor (JDTD) and joint angle topographical descriptor (JATD) as input	92.14%
[6] Max-pooling CNN for vision-based hand gesture recognition with color segmentation to retrieve hand contour and morphological image processing to remove noisy edges	96%
[7] CNN network and Stacked Denoising Auto encoders (SDAE) network	accuracy of 91.33% and 92.83% on testing data
[8] Real-sense-based deep CNN system	98.9%
Proposed CNN model	accuracy of 99.587% on testing and 89.67% on real world test images

***Table 5.6 Model comparison***

## **CHAPTER 6**

### **CONCLUSION AND FUTURE WORKS**

#### **6.1 CONCLUSION**

The main objective of the project was to develop a system that can translate static sign language into its corresponding letters. We have described a deep learning approach for the classification algorithm of American Sign Language. The proposed CNN architecture is designed with convolutional layers, followed by ReLU and max-pooling layers. A dataset consisting of 500 images (of size 200x200) per every 29 directory (26 for alphabets, 3 for special characters such as delete, nothing and space) was used to train the model. The proposed architecture has been tested on different learning models using different optimizers. It has been found that the model performs well on Adam optimiser. After testing with various models and optimizers, we chose this model which was able to achieve a higher evaluation accuracy.

Reaching the training phase of the development, one of the objectives of the project was to match or even exceed the accuracy of the studies presented using deep learning. Our system was able to achieve 99.587% training accuracy for 50 epochs in letter recognition. Our system have an advantage over the existing sensor based SLR systems that it can perform recognition without the aid of sensors, gloves, high end computers or hand markings and there are no difficulties in setting up the device.

## **6.2 FUTURE ENHANCEMENT**

In future enhancement, the model can be made efficient enough to work with real time applications by achieving high accuracy with low losses. The model can also be improved by training it to predict ASL words and numbers. Our results and process were severely affected and hindered by skin color and lighting variations in our self-generated data which led us to resort to a pre-made professionally constructed dataset. This problem can be solved by preprocessing the self-generated data set with skin color thresholding and enhancing the test image before feeding it to the model. However, including such methods will increase the complexity and decrease the computational speed.

## CHAPTER 7

### REFERENCES

- [1] Ankita Wadhawan<sup>1</sup>, Parteek Kumar<sup>1</sup> (2020) Deep learning-based sign language recognition system for static signs. Springer-Verlag London Ltd., part of Springer Nature 2020
- [2] Huang J, Zhou W, Li H, Li W (2015) Sign language recognition using real-sense. In: IEEE China summit and international conference on signal and information processing (ChinaSIP), pp 166–170
- [3] Kumar EK, Kishore PVV, Kiran Kumar MT (2019) 3D sign language recognition with joint distance and angular coded color topographical descriptor on a 2—stream CNN. *Neurocomput* 372:40–54
- [4] Lean Karlo S. Tolentino, Ronnie O. Serfa Juan, August C. Thio-ac, Maria Abigail B. Pamahoy, Joni Rose R. Forteza, and Xavier Jet O. Garcia (2019) Static Sign Language Recognition Using Deep Learning. *International Journal of Machine Learning and Computing*, Vol. 9, No.6, December 2019
- [5] Nagi J, Ducatelle F, Di Caro GA, Cires<sub>an</sub> D, Meier U, Giusti A, Gambardella LM (2011) Max-pooling convolutional neural networks for vision-based hand gesture recognition. In: IEEE international conference on signal and image processing applications (ICSIPA), pp 342–347

- [6] Rao GA, Syamala K, Kishore PVV, Sastry ASCS (2018) Deep convolutional neural networks for sign language recognition. In: IEEE conference on signal processing and communication engineering systems (SPACES), pp 194–197
- [7] Tushar AK, Ashiquzzaman A, Islam MR (2017) Faster convergence and reduction of overfitting in numerical hand sign recognition using DCNN. In: Humanitarian technology conference (R10-HTC), IEEE Region 10, pp 638–641
- [8] Vivek Bheda and N. Dianna Radpour (2017) Using Deep Convolutional Networks for Gesture Recognition in American Sign Language. CoRR, vol.abs/1710.06836, 2017