PUNJAB ENGINEERING COLLEGE

(DEEMED TO BE UNIVERSITY)

DEPARTMENT OF ELECTRONICS AND COMMUNICATION

Minor Project

ANUVADAK

Sign Language to Text Convertor for Dumb and Deaf

SUBMITTED BY-

Ritik Gupta	18105106
Aditya Mehta	18105107
Arjun Narula	18105114
Dhaval Gupta	18105115

UNDER THE SUPERVISION OF-

Dr. Neena Gupta

DECLARATION

We, hereby, declare that the project work entitled "**ANUVADAK: Sign Language to Text Convertor for Dumb and Deaf**" is an authentic record of our own work carried out as a requirement of Minor Project for the award of degree of B.Tech. Electronics and Communication Engineering, Punjab Engineering College, Chandigarh, under the guidance of **Dr. Neena Gupta**, from August 2020 to December, 2020.

18105106	Ritik Gupta
18105107	Aditya Mehta
18105114	Arjun Narula
18105115	Dhaval Gupta

Certified that the above statement made by the students is correct to the best of my knowledge and belief.

ACKNOWLEDGEMENT

We express our sincere gratitude to **Prof. Neena Gupta** for her valuable guidance and timely suggestions as a great mentor during the entire duration of our dissertation work, without which this work would not have been possible. We would also like to convey our deep regards to all other faculty members of our esteemed college, Punjab Engineering College, who have bestowed their great effort and guidance at appropriate times without which it would have been very difficult on our part to finish this work. Finally, we would also like to thank our friends for their advice and pointing out our mistakes during the review period of our sample model.

CONTENTS

S.No	**Chapter**	**Topic**	**P. No**
1		*List of Figures and Tables*	5
2		*Abstract*	6
3	Chapter 1	Introduction	7
4	Chapter 2	Literature Survey	11
5	Chapter 3	Artificial Neural Network – A Review	16
6	Chapter 4	Methodology	24
7	Chapter 5	Results and Applications	56
8	Chapter 6	Conclusions and Future Scope	60
9		*References*	62
10		*Appendix*	63

LIST OF FIGURES

Figure 1.1 Sign conventions of alphabets in ASL

Figure 3.1 Artificial Neural Network

Figure 3.2 Unsupervised learning

Figure 3.3 Supervised learning

Figure 3.4 Reinforcement learning

Figure 3.5 Convolution Neural Network

Figure 3.6 Average Pooling and Max Pooling

Figure 3.7 Fully converted Layer

Figure 3.8 Output Layer

Figure 3.9 Tensorflow

Figure 3.10 Keras

Figure 3.11 OpenCV

Figure 4.1 Capturing Image for a dataset

Figure 4.2 RGB image of alphabet “A”

Figure 4.3 Grayscale image of alphabet “A”

Figure 4.4 Image obtained of alphabet “A” after applying Gaussian filter

Figure 4.5 The CNN model used in the project

Figure 4.6 Solving the problem of overfitting using dropout

Figure 5.1 *Anuvadak*: Sign Language to Text Converter

Figure 5.2 Accuracy achieved in our model is 96.5% as shown above

Figure 5.3 Suggestions given based on the current word

Figure A.1 Ubuntu

Figure A.2 Python

Figure A.3 CNN

LIST OF TABLES

Table 1.1 Components of visual sign language

ABSTRACT

People affected by speech impairment can't communicate using hearing and speech, they rely on sign language for communication. Sign language is used among everybody who is speech impaired, but they find a hard time communicating with people which are nonsigners (people aren’t proficient in sign language). So requirement of a sign language interpreter is a must for speech impaired people. This makes their informal and formal communication difficult. There has been favorable progress in the field of gesture recognition and motion recognition with current advancements in deep learning. The proposed system tries to do a real time translation of hand gestures into equivalent English text. This system takes hand gestures as input through video and translates it text which could be understood by a non-signer. There were similar researches done earlier with most of them focussing on just sign translation of English alphabets or just numbers. There will be use of CNN for classification of hand gestures. By deploying this system, the communication gap between signers and non-signers. This will make communication speech impaired people less cumbersome. There were many researches which helped us to establish the idea of Artificial Neural Networks for this project. Many models were available that detected only characters with an accuracy of around 86%. We also explored the Linear discriminant analysis(LDA) technique but didn’t use it due its drawback to express complex data. The hardware implementation of the project was also read about. It has a cost and maintenence factor involved which we tried to eliminate in our project. We have achieved high acccuracy of 96.5% in our model, with the feature of suggestions of words and formation of sentences, an idea which was not found in any of the researches.

CHAPTER-1

INTRODUCTION

American sign language is a predominant sign language Since the only disability D&M people have is 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 dumb(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:

Finger Spelling	**Word Level 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.

Table 1.1 Components of visual sign language

In our project we basically focus on producing a model which can recognise Finger spelling 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.

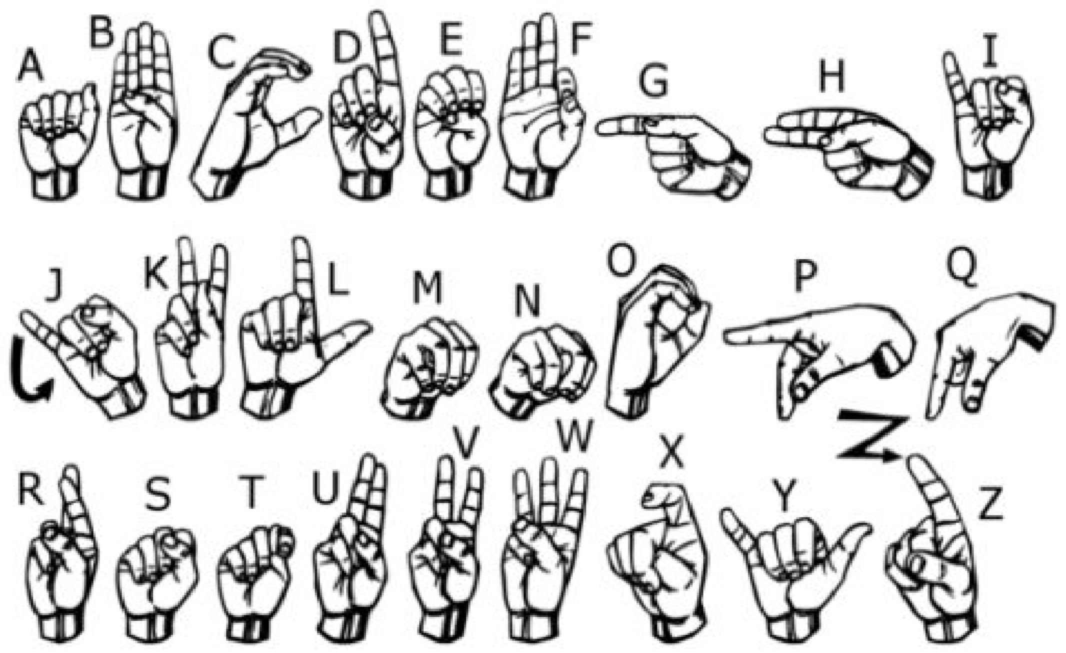


Figure 1.1 Sign conventions of alphabets in ASL

1.1 MOTIVATION

The main motivation is to translate sign language to text. The framework provides a helping-hand for speech-impaired to communicate with the rest of the world using sign language. This leads to the elimination of the middle person who generally acts as a medium of translation. This would contain a user-friendly environment for the user by providing text output for a sign gesture input.

There are many researches on recognition of sign languages or finger spellings. In [1] they have proposed a system that recognizes dynamic hand gestures for English numbers (0-9) in real-time using Hidden Markov Model. HMM is highly dependent on the probability of the hidden states, hence there are more number of parameters to learn which is time consuming. Our prime motivation was to devise a system which is time efficient for each and every alphabet of American Sign Language.

Pradumn Kumar and Upasana Dugal [2] proposed an algorithm using Tensorflow based on Advanced Convolutional Neural Networks for identification of plants. This research paper motivated us to use Convolutional Neural Networks for identification of American Sign Language symbols.

In [3], [4] and [5] we got an overview of Convolutional Neural Network(CNN), various deep learning approaches, their applications and their recent developments. We also gathered details about how overfitting in Neural Networks can be reduced by a technique called Dropout which is dropping off some neurons in the Neural Nets.

One method is based on Support Vector Machine (SVM) [7]. They extract signs from video sequences using skin colour segmentation, and distinguish static and dynamic gestures. However, the accuracy received in this method is 86%. Our aim is to achieve higher accuracy than this model.

Representing high-volume and high-order data is an essential problem, especially in machine learning field. This [8] research paper developed a model using Linear discriminant analysis (LDA), the classical LDA, however, demands that input data should be represented by vector. Such a constraint is a significant drawback to express complex data**.** To solve the above limitation for high-volume and high-order data nonlinear dimensionality reduction, we used Convolutional Neural Networks.

Another method is a controller gesture recognition system that extracts hand gestures using Flex sensors [9] for sensing the hand movements. The output from the microcontroller is the recognized text which is fed as input to the speech synthesizer. Arduino microcontroller processes the data for each particular gesture made. The system is trained for different voltage values for each letter. However, this method is not very economical and requires high maintenance from time to time.

We propose to develop a user friendly human computer interface (HCI) where the computer understands the human sign language and is accurate, effective and economical.

1.2 OBJECTIVES

1 To develop an application interface that interprets American Sign Language to Text in real time to assist deaf and dumb for communicating with them effectively eliminating the requirement of a translating individual.

2 To devise a model to achieve the highest possible accuracy and least time consumption for prediction of symbols as compared to already existing models.

3 To reduce the cost and develop an economical and user friendly graphical user interface (GUI) application that requires minimal maintenance for conversion of sign to its corresponding text.

4 To provide suggestions based on current word to eliminate the need of translating the full word, thereby improving accuracy and reducing time for sign to text conversion.

5 To reduce the chances of spelling mistakes by suggesting correct spellings from English dictionary words close to the current word.

6 To formulate characters, words and sentences with interpretation of symbols in American Sign Language in a portable and real time application.

CHAPTER-2

LITERATURE SURVEY

There are many researches on recognition of sign languages or finger spellings. We introspected various reasearch papers that were available and came up with an interface that converts sign language to text, provides the feature of adding a word and suggestions based on the word being translated.

In [1], they have proposed a system that recognizes dynamic hand gestures for English numbers (0-9) in real-time using Hidden Markov Model. HMM is highly dependent on the probability of the hidden states, hence there are more number of parameters to learn which is time consuming. The system contains two stages: Preprocessing for hand tracking and Classification to recognize gestures. Hidden Markov Model is used for the isolated and dynamic gesture recognition whose average recognition rates are 99.167% and 93.84% respectively. Hidden Markov Models (HMM) is used for the classification of the gestures. This model deals with dynamic aspects of gestures.Gestures are extracted from a sequence of video images by tracking the skin-colour blobs corresponding to the hand into a body– face space centered on the face of the user. The goal is to recognize two classes of gestures: deictic and symbolic.The image is filtered using a fast look–up indexing table. After filtering, skin colour pixels are gathered into blobs.

Pradumn Kumar and Upasana Dugal [2] proposed an algorithm using Tensorflow based on Advanced Convolutional Neural Networks for identification of plants. This research paper motivated us to use Convolutional Neural Networks for identification of American Sign Language symbols. Specially using CNN is a very trending procedure for Deep learning in computer point of view. ImageNet have produced a lot of expectation by giving exciting results. Here CNN takes the most challenging task for identification of plants by using their complete picture or any parts of that plants while others tackles one by one process like firstly they take any specific organisms (flowers, leaves and bark etc.) then whole picture of organisms. In CNN there are some limitations like it

is not better with very large sets of images or lack of explanatory power. So Advanced CNN will replace CNN because in Advanced CNN is small in size as compare to CNN for recognizing images. Here large models can be easily scale up and these models are small enough to train fast, by this we will get out new ideas and have a good chance for experiment on other methods also. The architecture of Advanced CNN is multi-layer consisting of alternate use of Convolution layers and nonlinearities. All these layers are followed by fully connected layers leading into a softmax classifier. This model gives a good accuracy results with in few time when we run on a GPU.

[3] presents a comprehensive review of deep learning and develops a categorization scheme to analyze the existing deep learning literature. It divides the deep learning algorithms into four categories according to the basic model they derived from: Convolutional Neural Networks, Restricted Boltzmann Machines, Autoencoder and Sparse Coding. The state-of-the-art approaches of the four classes are discussed and analyzed in detail. For the applications in the computer vision domain, the paper mainly reports the advancements of CNN based schemes, as it is the most extensively utilized and most suitable for images. Most notably, some recent articles have reported inspiring advances showing that some CNN-based algorithms have already exceeded the accuracy of human raters.Despite the promising results reported so far, there is significant room for further advances. For example, the underlying theoretical foundation does not yet explain under what conditions they will perform well or outperform other approaches, and how to determine the optimal structure for a certain task. This paper describes these challenges and summarizes the new trends in designing and training deep neural networks, along with several directions that may be further explored in the future.

Convolution neural network has long been used in the field of digital image processing and speech recognition, and has achieved great success. Before the convolutional neural network was proposed, both image processing and speech recognition were done by traditional machine learning algorithms. Although great results were achieved, it was difficult to make further breakthroughs, so CNN came into being. Currently, CNN for image processing and speech recognition are

relatively mature. Both the theoretical research and the industrial application have been very successful, which has promoted CNN's leap-forward development. CNN's success of image processing and speech recognition has stimulated its research frenzy in natural language processing. The current CNN to handle natural language has been widely used, although some achievements have been made, the current effect is not very good.

The purpose of [4] is to give a clearer explanation of the structure of CNN. At the same time, give a brief summary and prospect of current CNN research in image processing, speech recognition and natural language processing.

Results observed in the comparative study with other traditional methods suggest that CNN gives better accuracy and boosts the performance of the system due to unique features like shared weights and local connectivity.

CNN is better than other deep learning methods in applications pertaining to computer vision and natural language processing because it mitigates most of the traditional problems.

To reduce the problem of overfitting, we used the Dropout Technique as suggested in [5]. Dropout is a technique for improving neural networks by reducing overfitting. Standard backpropagation learning builds up brittle co-adaptations that work for the training data but do not generalize to unseen data. Random dropout breaks up these co-adaptations by making the presence of any particular hidden unit unreliable. This technique was found to improve the performance of neural nets in a wide variety of application domains including object classification, digit recognition, speech recognition, document classification and analysis of computational biology data. This suggests that dropout is a general technique and is not specific to any domain. Dropout considerably improved the performance of standard neural nets on other data sets as well. This idea can be extended to Restricted Boltzmann Machines and other graphical models. The central idea of dropout is to take a large model that overfits easily and repeatedly sample and train smaller sub-models from it.

Ankit Ojha, Ayush Pandey, Shubham Maurya, Abhishek Thakur and Dr. Dayananda P in [6] developed a finger spelling sign language translator is obtained which has an accuracy of 95%. They created a desktop application that uses a computer’s webcam to capture a person signing gestures for ASL, and translate it into corresponding text and speech in real time. The translated sign language gesture will be acquired in text which is farther converted into audio. In this manner they are implementing a finger spelling sign language translator. To enable the detection of gestures, they used Convolutional neural network (CNN). This research paper provided us insight about the base model for our project.

In the following paper [7], Support Vector Machine (SVM) was used as the machine learning method. They proposed a recognition method of fingerspellings in Japanese sign language, which uses classification tree based on pattern recognition and machine learning. Fingerspellings of Japanese sign language are based on American alphabets, and some are added according to Japanese Character, gestures, numbers, and meanings. They constructed a classfication tree for easily recognized fingerspellings and also used machine learning for dificultly recognized ones. They achieved an accuracy of 86% using this model.

Representing high-volume and high-order data is an essential problem, especially in machine learning field. The [8] research paper developed a model using Linear discriminant analysis (LDA). The classical LDA, however, demands that input data should be represented by vector. Such a constraint is a significant drawback to express complex data. In this paper, they have proposed a convolutional 2D-LDA method for nonlinear dimensionality reduction. The difficult problem of optimization is solved by a clever equivalence of two objective functions. The proposed method employs a two stage end-to-end CNN to realize dimensionality reduction. Effectiveness of such structure has been proved with two different networks. The convolutional 2D LDA method out- performs the classical LDA in all experiment settings.

Another method is a hardware based controller gesture recognition system that extracts hand gestures using Flex sensors [9] for sensing the hand movements using a glove. The sensor glove design along with the tactile sensor helps in reducing the ambiguity in gestures and shows improved accuracy. The output from the microcontroller is the recognized text which is fed as input to the speech synthesizer. Arduino microcontroller processes the data for each particular gesture made. The system is trained for different voltage values for each letter. However, this method is not very economical and requires high maintenance from time to time.

CHAPTER-3
ARTIFICIAL NEURAL NETWORK (ANN) – A REVIEW

Feature Extraction and Representation : *The representation of an image as a 3D matrix having dimension as of height and width of the image and the value of each pixel as depth (1 in case of Grayscale and 3 in case of RGB). Further, these pixel values are used for extracting useful features using CNN.*

3.1 Artificial Neural Networks

Artificial Neural Network is a connections 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 processes 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.

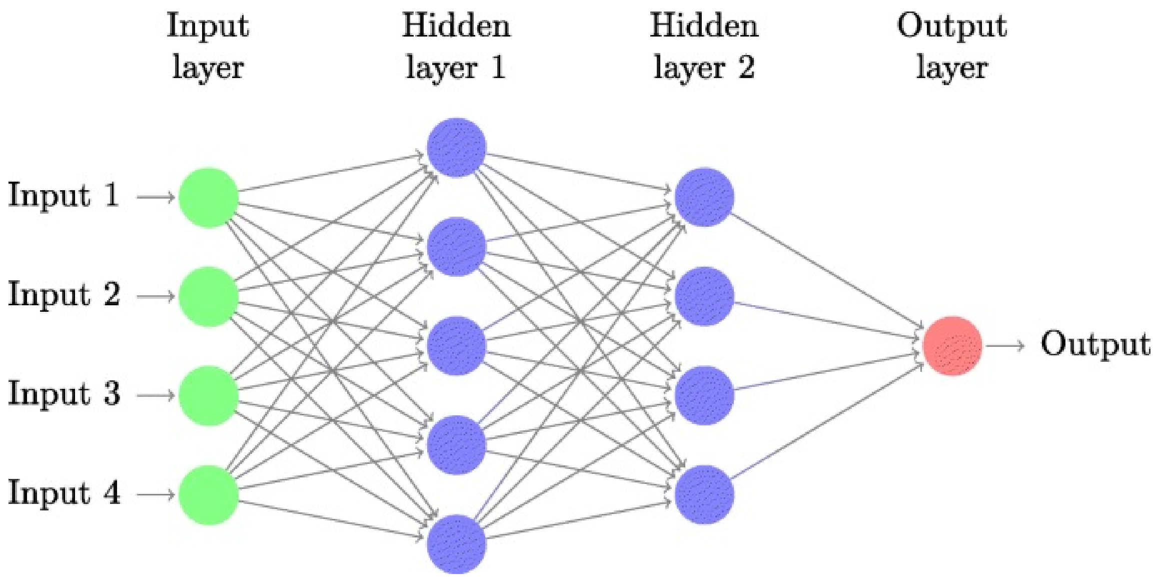


Figure 3.1 Artificial Neural Network

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

3.1.1. Unsupervised Learning:

Unsupervised learning is a type of machine learning that looks for previously undetected patterns in a data set with no pre-existing labels and with a minimum of human supervision. Two of the main methods used in unsupervised learning are principal component and cluster analysis.

The only requirement to be called an unsupervised learning strategy is to learn a new feature space that captures the characteristics of the original space by maximizing some objective function or minimising some loss function. Therefore, generating a covariance matrix is not unsupervised learning, but taking the eigenvectors of the covariance matrix is because the linear algebra eigendecomposition operation maximizes the variance; this is known as principal component analysis.

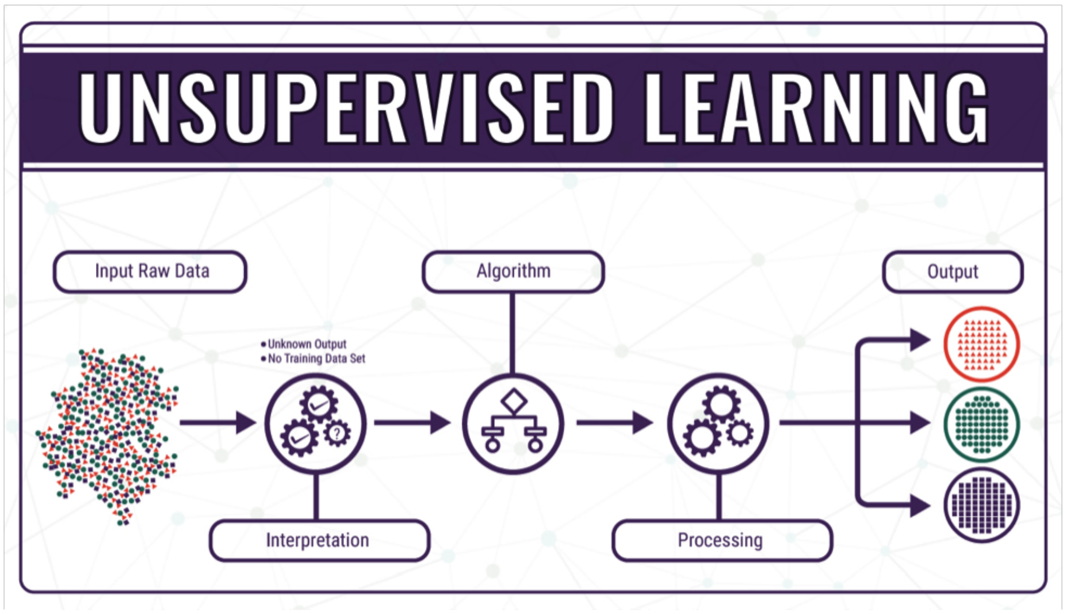


Figure 3.2 Unsupervised learning

3.1.2. Supervised Learning:

Supervised learning is the machine learning task of learning a function that maps an input to an output based on example input-output pairs. It infers a function from labeled training data consisting of a set of training examples. In supervised learning, each example is a pair consisting of an input object (typically a vector) and a desired output value (also called the supervisory signal). A supervised learning algorithm analyzes the training data and produces an inferred function, which can be used for mapping new examples. An optimal scenario will allow for the algorithm to correctly determine the class labels for unseen instances. This requires the learning algorithm to generalize from the training data to unseen situations in a "reasonable" way.

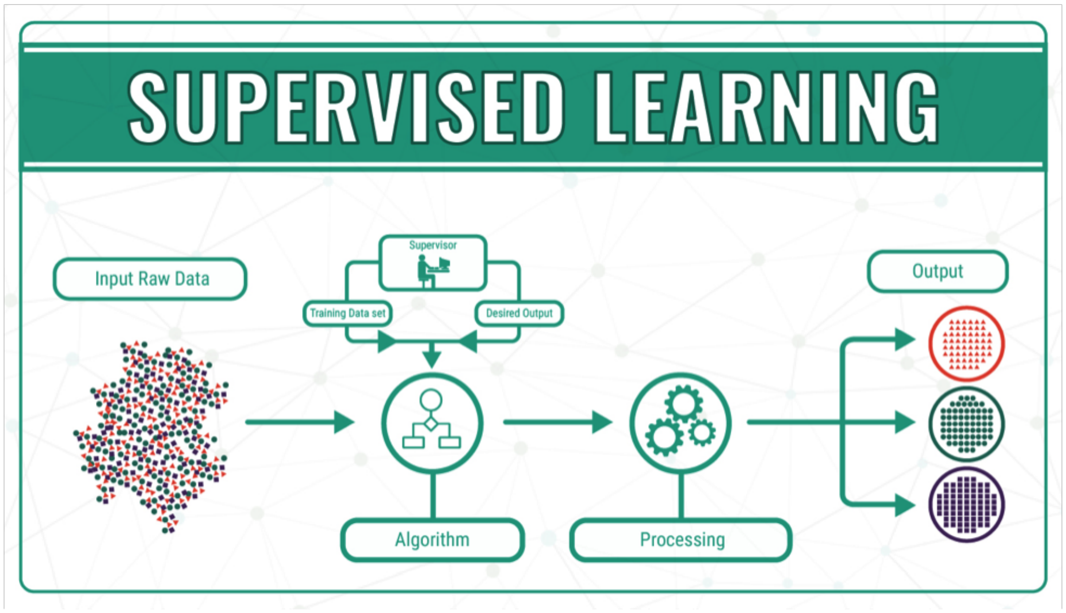


Figure 3.3 Supervised learning

3.1.3. Reinforcement Learning:

Reinforcement learning (RL) is an area of machine learning concerned with how software agents ought to take actions in an environment in order to maximize the notion of cumulative reward. Reinforcement learning is one of three basic machine learning paradigms, alongside supervised learning and unsupervised learning. Reinforcement learning differs from supervised learning in not needing labelled input/output pairs be presented, and in not needing sub-optimal actions to be explicitly corrected. Instead the focus is on finding a balance between exploration (of uncharted territory) and exploitation (of current knowledge).

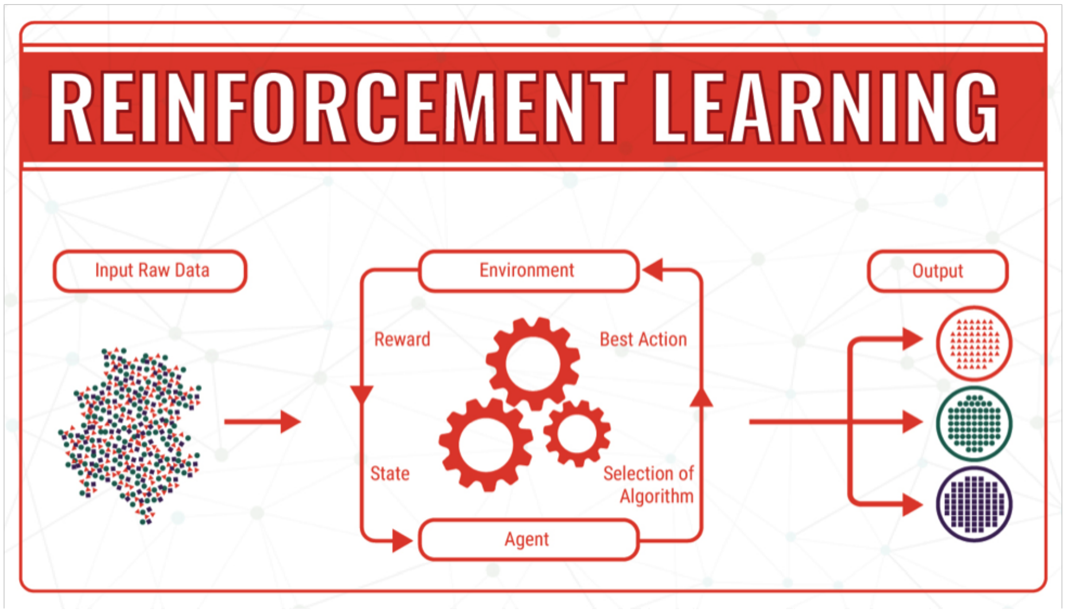


Figure 3.4 Reinforcement learning

3.2 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.

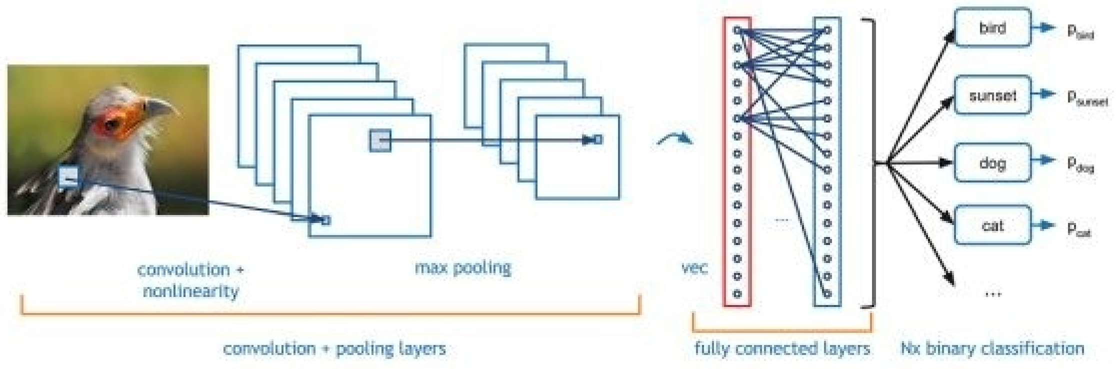


Figure 3.5 Convolution Neural Network

1 **Convolution Layer :**

In convolution layer we take a small window size [typically of length 5*5] that extends to the depth of the input matrix. The layer consist of learnable filters of window size. During every iteration we slid the window by stride size [typically 1], and compute the dot product of filter entries and input values at a given position. As we continue this process well create a 2-Dimensional activation matrix that gives the response of that matrix at every spatial position. That is, the network will learn filters that activate when they see some type of visual feature such as an edge of some orientation or a blotch of some color.

2 **Pooling Layer :**

We use pooling layer to decrease the size of activation matrix and ultimately reduce the learnable parameters. There are two type of pooling:

2.a **Max Pooling :** In max pooling we take a window size [for example window of size 2*2], and only take the maximum of 4 values. Well lid this window and continue this process, so well finally get a activation matrix half of its original Size.

2.b **Average Pooling :** In average pooling we take average of all values in a window.

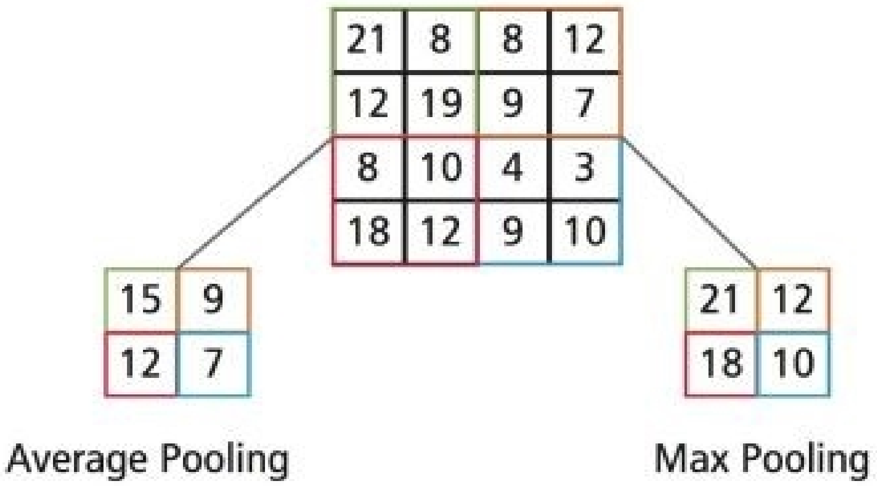


Figure 3.6 Average Pooling and Max Pooling

3 Fully Connected Layer :

In convolution layer neurons are connected only to a local region, while in a fully connected region, well connect the all the inputs to neurons.

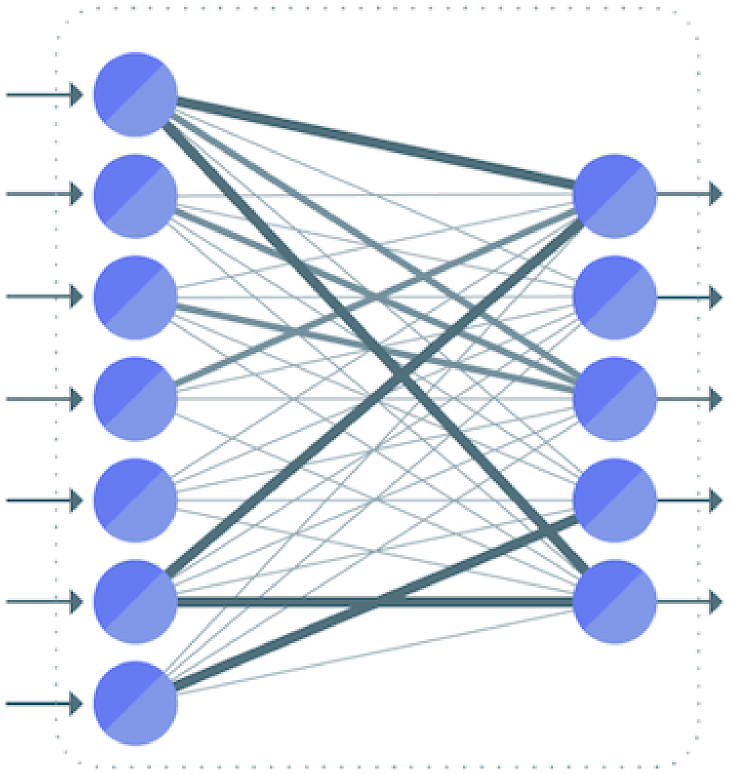


Figure 3.7 Fully converted Layer

4 Final Output Layer :

After getting values from fully connected layer, well connect them to final layer of neurons[having count equal to total number of classes], that will predict the probability of each image to be in different classes.

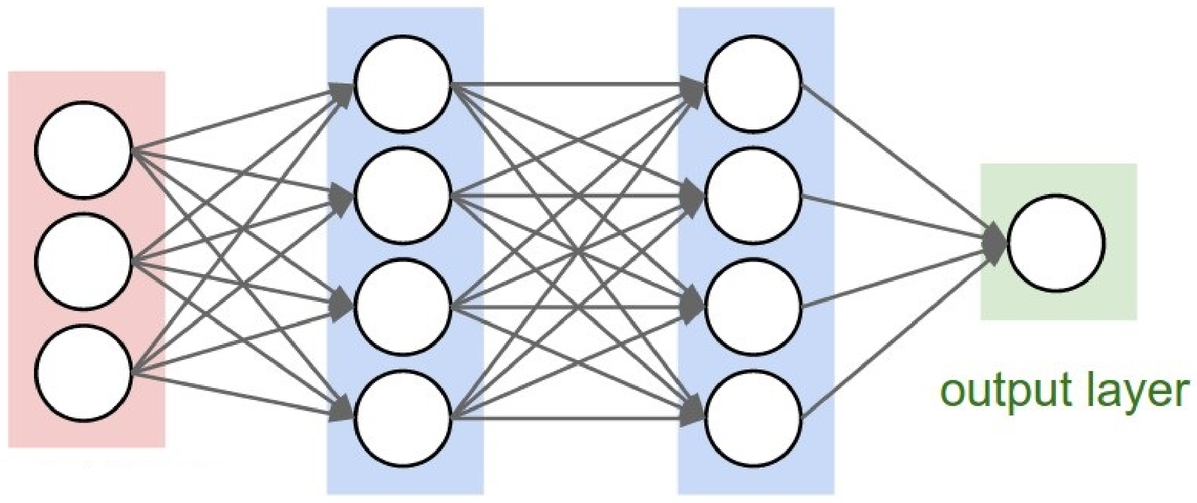


Figure 3.8 Output Layer

3.3 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.

Figure 3.9 Tensorflow

3.4 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.

Figure 3.10 Keras

3.5 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.

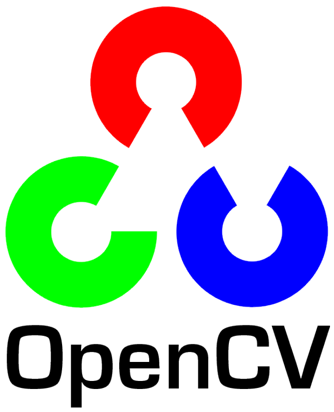


Figure 3.11 Keras

CHAPTER-4

METHODOLOGY

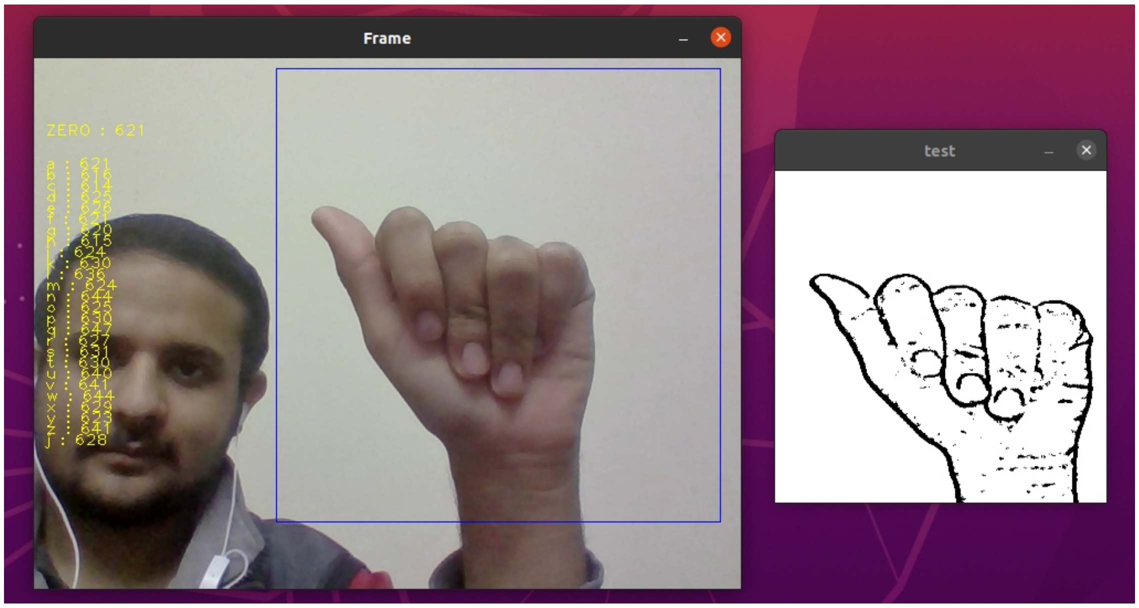
The system is a vision based approach. All the signs are represented with bare hands and so it eliminates the problem of using any artificial devices for interaction.

4.1 Data Set Generation

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:

Step 1:

We used Open computer vision(OpenCV) library in order to produce our dataset. Firstly we captured around 600 images of each of the symbol in ASL for training purposes and around 150 images per symbol for testing purpose.



```
ZERO : 621

a : 621
b : 616
c : 614
d : 625
e : 626
f : 621
g : 620
h : 615
i : 624
k : 630
l : 636
m : 624
n : 644
o : 625
p : 630
q : 647
r : 627
s : 631
t : 630
u : 640
v : 641
w : 644
x : 629
y : 623
z : 641
j : 628
```

Figure 4.1 Capturing Image for a dataset

Step 2:

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

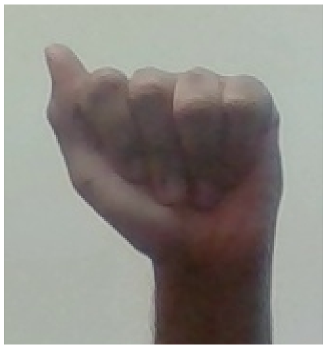


Figure 4.2 RGB image of alphabet "A"

Step 3:

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

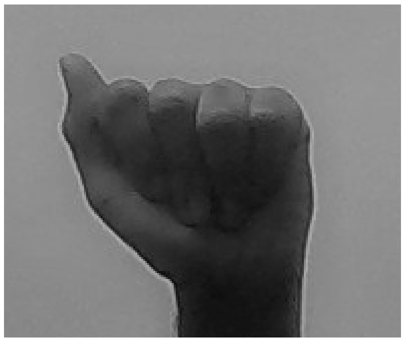


Figure 4.3 Grayscale image of alphabet "A"

Step 4:

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

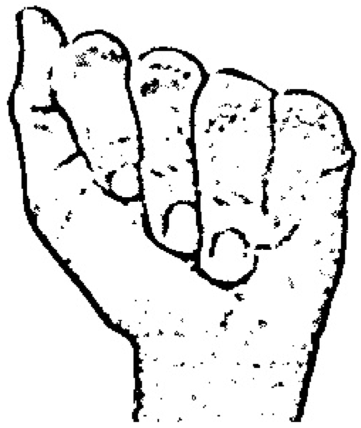


Figure 4.4 image obtained of alphabet “A” after applying Gaussian filter

4.2 GESTURE CLASSIFICATION

The approach which we used for this project is :
Our approach uses two layers of algorithm to predict the final symbol of the user.

4.2.1 Algorithm Layer :

1. Apply gaussian blur filter and threshold to the frame taken with opencv to get the processed image after feature extraction.
2. This processed image is passed to the CNN model for prediction and if a letter is detected for more than 60 frames then the letter is printed and taken into consideration for forming the word.
3. Space between the words are considered using the blank symbol.

4.2.2 CNN Model :

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

2 **1st Pooling Layer :** The pictures are downsampled using max pooling of 2x2 i.e we keep the highest value in the 2x2 square of array. Therefore, our picture is downsampled to 63x63 pixels.

3 **2nd Convolution Layer :** Now, these 63 x 63 from the output of the first pooling layer is served as an input to the second convolutional layer.It is processed in the second convolutional layer using 32 filter weights (3x3 pixels each).This will result in a 61 x 61 pixel image.

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

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

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

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

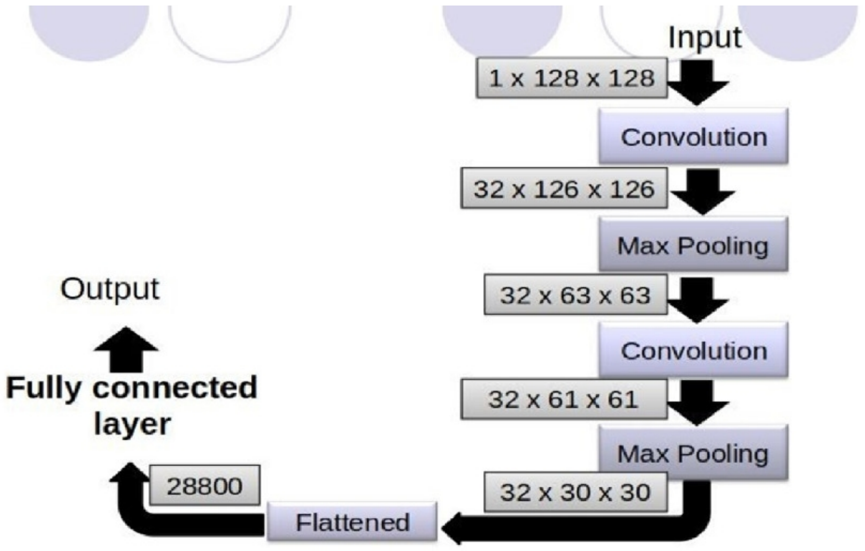


Figure 4.5 The CNN model used in the project

4.2.3 Activation Function :

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

4.2.4 Pooling Layer :

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

4.2.5 Dropout Layers:

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

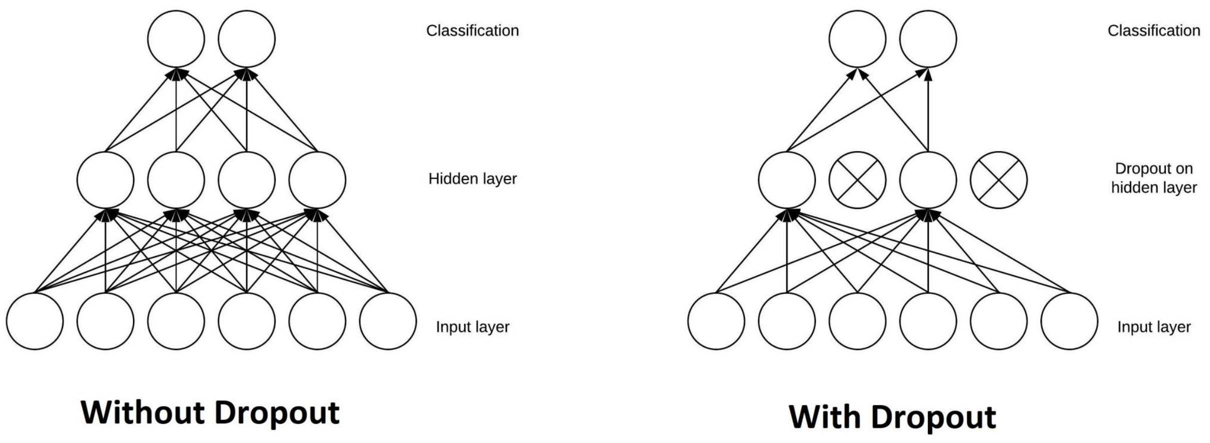


Figure 4.6 Solving the problem of overfitting using dropout

4.2.6 Optimizer :

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

4.3 Finger spelling sentence formation

4.3.1 Implementation

1 Whenever the count of a letter detected exceedoutput connected layers a specific value and no other letter is close to it by a threshold we print the letter and add it to the current string(In our code we kept the value as 60).

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

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

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

4.3.2 Autocorrect Feature :

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

4.4 TRAINING AND TESTING

- We convert our input images(RGB) into grayscale and apply gaussian blur to remove unnecessary noise.We apply adaptive threshold to extract our hand from the background and resize our images to 128 x 128.

- We feed the input images after preprocessing to our model for training and testing after applying all the operations mentioned above.

- The prediction layer estimates how likely the image will fall under one of the classes. So the output is normalized between 0 and 1 and such that the sum of each values in each class sums to 1. We have achieved this using softmax function.

- At first the output of the prediction layer will be somewhat far from the actual value. To make it better we have trained the networks using labeled data. The cross-entropy is a performance measurement used in the classification. It is a continuous function which is positive at values which is not same as labeled value and is zero exactly when it is equal to the labeled value.

- 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.

4.5 LIBRARIES USED

4.5.1. OpenCV

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

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

4.5.2 Tensorflow

TensorFlow is an open-source software library for dataflow programming across a range of tasks. It is a symbolic math library, and is also used for machine learning applications such as neural networks. It is used for both research and production at Google.

TensorFlow was developed by the Google brain team for internal Google use. It was released under the Apache 2.0 open source library on November 9, 2015.

TensorFlow is Google Brain's second-generation system. Version 1.0.0 was released on February 11, 2017. While the reference implementation runs on single devices, TensorFlow can run on multiple CPUs and GPUs (with optional CUDA and SYCL extensions for general-purpose computing on graphics processing units).

TensorFlow is available on 64-bit Linux, macOS, Windows, and mobile computing platforms including Android and iOS.

Its flexible architecture allows for the easy deployment of computation across a variety of platforms (CPUs, GPUs, TPUs), and from desktops to clusters of servers to mobile and edge devices.

4.5.3 Keras

Keras is one of the leading high-level neural networks APIs. It is written in Python and supports multiple back-end neural network computation engines.

Given that the TensorFlow project has adopted Keras as the high-level API for the upcoming TensorFlow 2.0 release, Keras looks to be *a* winner, if not necessarily *the* winner. In

The biggest reasons to use Keras stem from its guiding principles, primarily the one about being user friendly. Beyond ease of learning and ease of model building, Keras offers the advantages of broad adoption, support for a wide range of production deployment options, integration with at least five back-end engines (TensorFlow, CNTK, Theano, MXNet, and PlaidML), and strong support for multiple GPUs and distributed training.

4.5.4 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.NumPy stands for Numerical Python.

In Python we have lists that serve the purpose of arrays, but they are slow to process.NumPy aims to provide an array object that is up to 50x faster than traditional Python lists.The array object in NumPy is called ndarray, it provides a lot of supporting functions that make working with ndarray very easy.Arrays are very frequently used in data science, where speed and resources are very important.

NumPy arrays are stored at one continuous place in memory unlike lists, so processes can access and manipulate them very efficiently.This behavior is called locality of reference in computer science.This is the main reason why NumPy is faster than lists. Also it is optimized to work with latest CPU architectures.

4.5.5 Os

The OS module in python provides functions for interacting with the operating system. OS, comes under Python’s standard utility modules. This module provides a portable way of using operating system dependent functionality. The *os* and *os.path* modules include many functions to interact with the file system.

Python OS module provides the facility to establish the interaction between the user and the operating system. It offers many useful OS functions that are used to perform OS-based tasks and get related information about operating system.The OS comes under Python's standard utility modules. This module offers a portable way of using operating system dependent functionality.The Python OS module lets us work with the files and directories.

4.5.6 Hunspell (Autocorrect feature)

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

4.5.7 Tkinter

Tkinter is the standard GUI library for Python. Python when combined with Tkinter provides a fast and easy way to create GUI applications. Tkinter provides a powerful object-oriented interface to the Tk GUI toolkit.

Creating a GUI application using Tkinter is an easy task. All you need to do is perform the following steps −

1. Import the *Tkinter* module.
2. Create the GUI application main window.

3 Add one or more of the above-mentioned widgets to the GUI application.
4 Enter the main event loop to take action against each event triggered by the user.

4.5.8 PIL (Python Imaging Library)

Python Imaging Library (abbreviated as PIL) (in newer versions known as Pillow) is a free and open-source additional library for the Python programming language that adds support for opening, manipulating, and saving many different image file formats. It is available for Windows, Mac OS X and Linux. The latest version of PIL is 1.1.7, was released in September 2009 and supports Python 1.5.2–2.7, with Python 3 support to be released "later".

Development appears to be discontinued, with the last commit to the PIL repository coming in 2011. Consequently, a successor project called Pillow has forked the PIL repository and added Python 3.x support. This fork has been adopted as a replacement for the original PIL in Linux distributions including Debian and Ubuntu (since 13.04).

4.6 CODES

The dataset collection file, the image processing files, the model training file and the application interface files, all have been written in Python.

4.6.1 Collect-data.py

```
import cv2

import numpy as np

import os

import string

# Create the directory structure

if not os.path.exists("data3"):

    os.makedirs("data3")
```

```
if not os.path.exists("data3/train"):
    os.makedirs("data3/train")
if not os.path.exists("data3/test"):
    os.makedirs("data3/test")
if not os.path.exists("data3/train/" + "0"):
        os.makedirs("data3/train/"+"0")
for i in string.ascii_uppercase:
    if not os.path.exists("data3/train/" + i):
        os.makedirs("data3/train/"+i)
    if not os.path.exists("data3/test/" + i):
        os.makedirs("data3/test/"+i)
# Train or test
mode = 'train'
directory = 'data3/'+mode+'/'
minValue = 70
cap = cv2.VideoCapture(0)
interrupt = -1
while True:
    _, frame = cap.read()
    # Simulating mirror image
    frame = cv2.flip(frame, 1)
    # Getting count of existing images
    count = {
            'zero': len(os.listdir(directory+"/0")),
            'a': len(os.listdir(directory+"/A")),
            'b': len(os.listdir(directory+"/B")),
            'c': len(os.listdir(directory+"/C")),
        'd': len(os.listdir(directory+"/D")),
        'e': len(os.listdir(directory+"/E")),
        'f': len(os.listdir(directory+"/F")),
        'g': len(os.listdir(directory+"/G")),
        'h': len(os.listdir(directory+"/H")),
        'i': len(os.listdir(directory+"/I")),
        'j': len(os.listdir(directory+"/J")),
        'k': len(os.listdir(directory+"/K")),
        'l': len(os.listdir(directory+"/L")),
        'm': len(os.listdir(directory+"/M")),
        'n': len(os.listdir(directory+"/N")),
        'o': len(os.listdir(directory+"/O")),
        'p': len(os.listdir(directory+"/P")),
        'q': len(os.listdir(directory+"/Q")),
        'r': len(os.listdir(directory+"/R")),
        's': len(os.listdir(directory+"/S")),
        't': len(os.listdir(directory+"/T")),
        'u': len(os.listdir(directory+"/U")),
        'v': len(os.listdir(directory+"/V")),
        'w': len(os.listdir(directory+"/W")),
        'x': len(os.listdir(directory+"/X")),
        'y': len(os.listdir(directory+"/Y")),
        'z': len(os.listdir(directory+"/Z"))
        }

    # Printing the count in each set to the screen
    cv2.putText(frame, "ZERO : "+str(count['zero']), (10, 70),
cv2.FONT_HERSHEY_PLAIN, 1, (0,255,255), 1)

    cv2.putText(frame, "a : "+str(count['a']), (10, 100),
cv2.FONT_HERSHEY_PLAIN, 1, (0,255,255), 1)

    cv2.putText(frame, "b : "+str(count['b']), (10, 110),
cv2.FONT_HERSHEY_PLAIN, 1, (0,255,255), 1)

    cv2.putText(frame, "c : "+str(count['c']), (10, 120),
cv2.FONT_HERSHEY_PLAIN, 1, (0,255,255), 1)

    cv2.putText(frame, "d : "+str(count['d']), (10, 130),
cv2.FONT_HERSHEY_PLAIN, 1, (0,255,255), 1)

    cv2.putText(frame, "e : "+str(count['e']), (10, 140),
cv2.FONT_HERSHEY_PLAIN, 1, (0,255,255), 1)

    cv2.putText(frame, "f : "+str(count['f']), (10, 150),
cv2.FONT_HERSHEY_PLAIN, 1, (0,255,255), 1)

    cv2.putText(frame, "g : "+str(count['g']), (10, 160),
cv2.FONT_HERSHEY_PLAIN, 1, (0,255,255), 1)

    cv2.putText(frame, "h : "+str(count['h']), (10, 170),
cv2.FONT_HERSHEY_PLAIN, 1, (0,255,255), 1)

    cv2.putText(frame, "i : "+str(count['i']), (10, 180),
cv2.FONT_HERSHEY_PLAIN, 1, (0,255,255), 1)

    cv2.putText(frame, "k : "+str(count['k']), (10, 190),
cv2.FONT_HERSHEY_PLAIN, 1, (0,255,255), 1)

    cv2.putText(frame, "l : "+str(count['l']), (10, 200),
cv2.FONT_HERSHEY_PLAIN, 1, (0,255,255), 1)

    cv2.putText(frame, "m : "+str(count['m']), (10, 210),
cv2.FONT_HERSHEY_PLAIN, 1, (0,255,255), 1)

    cv2.putText(frame, "n : "+str(count['n']), (10, 220),
cv2.FONT_HERSHEY_PLAIN, 1, (0,255,255), 1)

    cv2.putText(frame, "o : "+str(count['o']), (10, 230),
cv2.FONT_HERSHEY_PLAIN, 1, (0,255,255), 1)

    cv2.putText(frame, "p : "+str(count['p']), (10, 240),
cv2.FONT_HERSHEY_PLAIN, 1, (0,255,255), 1)
```

```
    cv2.putText(frame, "q : "+str(count['q']), (10, 250),
cv2.FONT_HERSHEY_PLAIN, 1, (0,255,255), 1)
    cv2.putText(frame, "r : "+str(count['r']), (10, 260),
cv2.FONT_HERSHEY_PLAIN, 1, (0,255,255), 1)
    cv2.putText(frame, "s : "+str(count['s']), (10, 270),
cv2.FONT_HERSHEY_PLAIN, 1, (0,255,255), 1)
    cv2.putText(frame, "t : "+str(count['t']), (10, 280),
cv2.FONT_HERSHEY_PLAIN, 1, (0,255,255), 1)
    cv2.putText(frame, "u : "+str(count['u']), (10, 290),
cv2.FONT_HERSHEY_PLAIN, 1, (0,255,255), 1)
    cv2.putTe


xt(frame, "v : "+str(count['v']), (10, 300), cv2.FONT_HERSHEY_PLAIN, 1,
(0,255,255), 1)
    cv2.putText(frame, "w : "+str(count['w']), (10, 310),
cv2.FONT_HERSHEY_PLAIN, 1, (0,255,255), 1)
    cv2.putText(frame, "x : "+str(count['x']), (10, 320),
cv2.FONT_HERSHEY_PLAIN, 1, (0,255,255), 1)
    cv2.putText(frame, "y : "+str(count['y']), (10, 330),
cv2.FONT_HERSHEY_PLAIN, 1, (0,255,255), 1)
    cv2.putText(frame, "z : "+str(count['z']), (10, 340),
cv2.FONT_HERSHEY_PLAIN, 1, (0,255,255), 1)
    cv2.putText(frame, "j : "+str(count['j']), (10, 350),
cv2.FONT_HERSHEY_PLAIN, 1, (0,255,255), 1)
    # Coordinates of the ROI
    x1 = int(0.5*frame.shape[1])
    y1 = 10
    x2 = frame.shape[1]-10
    y2 = int(0.5*frame.shape[1])
    # Drawing the ROI
    # The increment/decrement by 1 is to compensate for the bounding box
    cv2.rectangle(frame, (220-1, 9), (620+1, 419), (255,0,255) ,1)
    # Extracting the ROI
    roi = frame[10:410, 220:520]
    cv2.imshow("Frame", frame)  #show bigger coloured frame
    gray = cv2.cvtColor(roi, cv2.COLOR_BGR2GRAY)
    blur = cv2.GaussianBlur(gray,(5,5),2)
th3=cv2.adaptiveThreshold(blur,255,cv2.ADAPTIVE_THRESH_GAUSSIA
N_C,cv2.THRESH_BINARY_INV,11,2)
    ret, test_image = cv2.threshold(th3, minValue, 255,
cv2.THRESH_BINARY_INV+cv2.THRESH_OTSU)
    test_image = cv2.resize(test_image, (300,300))
    cv2.imshow("test", test_image)  #show smaller black-white frame
    interrupt = cv2.waitKey(10)
    if interrupt & 0xFF == 27: # esc key
        break
    if interrupt & 0xFF == ord('0'):
        cv2.imwrite(directory+'0/'+str(count['zero'])+'.jpg', roi)
    if interrupt & 0xFF == ord('a'):
        cv2.imwrite(directory+'A/'+str(count['a'])+'.jpg', roi)
    if interrupt & 0xFF == ord('b'):
        cv2.imwrite(directory+'B/'+str(count['b'])+'.jpg', roi)
    if interrupt & 0xFF == ord('c'):
        cv2.imwrite(directory+'C/'+str(count['c'])+'.jpg', roi)
    if interrupt & 0xFF == ord('d'):
        cv2.imwrite(directory+'D/'+str(count['d'])+'.jpg', roi)
    if interrupt & 0xFF == ord('e'):
        cv2.imwrite(directory+'E/'+str(count['e'])+'.jpg', roi)
```

```
if interrupt & 0xFF == ord('f'):
    cv2.imwrite(directory+'F/'+str(count['f'])+'.jpg', roi)
if interrupt & 0xFF == ord('g'):
    cv2.imwrite(directory+'G/'+str(count['g'])+'.jpg', roi)
if interrupt & 0xFF == ord('h'):
    cv2.imwrite(directory+'H/'+str(count['h'])+'.jpg', roi)
if interrupt & 0xFF == ord('i'):
    cv2.imwrite(directory+'I/'+str(count['i'])+'.jpg', roi)
if interrupt & 0xFF == ord('j'):
    cv2.imwrite(directory+'J/'+str(count['j'])+'.jpg', roi)
if interrupt & 0xFF == ord('k'):
    cv2.imwrite(directory+'K/'+str(count['k'])+'.jpg', roi)
if interrupt & 0xFF == ord('l'):
    cv2.imwrite(directory+'L/'+str(count['l'])+'.jpg', roi)
if interrupt & 0xFF == ord('m'):
    cv2.imwrite(directory+'M/'+str(count['m'])+'.jpg', roi)
if interrupt & 0xFF == ord('n'):
    cv2.imwrite(directory+'N/'+str(count['n'])+'.jpg', roi)
if interrupt & 0xFF == ord('o'):
    cv2.imwrite(directory+'O/'+str(count['o'])+'.jpg', roi)
if interrupt & 0xFF == ord('p'):
    cv2.imwrite(directory+'P/'+str(count['p'])+'.jpg', roi)
if interrupt & 0xFF == ord('q'):
    cv2.imwrite(directory+'Q/'+str(count['q'])+'.jpg', roi)
if interrupt & 0xFF == ord('r'):
    cv2.imwrite(directory+'R/'+str(count['r'])+'.jpg', roi)
if interrupt & 0xFF == ord('s'):
        cv2.imwrite(directory+'S/'+str(count['s'])+'.jpg', roi)
    if interrupt & 0xFF == ord('t'):
        cv2.imwrite(directory+'T/'+str(count['t'])+'.jpg', roi)
    if interrupt & 0xFF == ord('u'):
        cv2.imwrite(directory+'U/'+str(count['u'])+'.jpg', roi)
    if interrupt & 0xFF == ord('v'):
        cv2.imwrite(directory+'V/'+str(count['v'])+'.jpg', roi)
    if interrupt & 0xFF == ord('w'):
        cv2.imwrite(directory+'W/'+str(count['w'])+'.jpg', roi)
    if interrupt & 0xFF == ord('x'):
        cv2.imwrite(directory+'X/'+str(count['x'])+'.jpg', roi)
    if interrupt & 0xFF == ord('y'):
        cv2.imwrite(directory+'Y/'+str(count['y'])+'.jpg', roi)
    if interrupt & 0xFF == ord('z'):
        cv2.imwrite(directory+'Z/'+str(count['z'])+'.jpg', roi)
cap.release()
cv2.destroyAllWindows()
```

4.6.2 image-processing.py

```
import numpy as np
import cv2
minValue = 70
def func(path):
    frame = cv2.imread(path)
    gray = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY)
    blur = cv2.GaussianBlur(gray,(5,5),2)
```

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

4.6.3 pre-processing.py

```
import cv2
import os
from image_processing import func
if not os.path.exists("temp"):
    os.makedirs("temp")
if not os.path.exists("temp/train"):
    os.makedirs("temp/train")
if not os.path.exists("temp/test"):
    os.makedirs("temp/test")
path="data3/train"  #coloured images here
path1 = "temp"  #black and white images stored here
label=0 #number of characters
var = 0 #total number of images
c1 = 0 #total images in train
c2 = 0#number images in test
for (dirpath,dirnames,filenames) in os.walk(path):
    for dirname in dirnames:
        print(dirname)
        for(direcpath,direcnames,files) in os.walk(path+"/"+dirname):
            if not os.path.exists(path1+"/train/"+dirname):
             os.makedirs(path1+"/train/"+dirname)
         if not os.path.exists(path1+"/test/"+dirname):
             os.makedirs(path1+"/test/"+dirname)
         num=0.8*len(files)
         i=0
         for file in files:
             var+=1
             actual_path=path+"/"+dirname+"/"+file
             actual_path1=path1+"/"+"train/"+dirname+"/"+file
             actual_path2=path1+"/"+"test/"+dirname+"/"+file
             img = cv2.imread(actual_path, 0)
             bw_image = func(actual_path)
             if i<num:
                 c1 += 1
                 cv2.imwrite(actual_path1 , bw_image)
             else:
                 c2 += 1
                 cv2.imwrite(actual_path2 , bw_image)
             i=i+1
        label=label+1
print(var)
print(c1)
print(c2)
print(label)
```

4.6.4 train.py

```
# Importing the Keras libraries and packages
from keras.models import Sequential
from keras.layers import Convolution2D

from keras.layers import MaxPooling2D

from keras.layers import Flatten

from keras.layers import Dense , Dropout

import os

os.environ["CUDA_VISIBLE_DEVICES"] = "1"

sz = 128

# Step 1 - Building the CNN

# Initializing the CNN

classifier = Sequential()

# First convolution layer and pooling

classifier.add(Convolution2D(32, (3, 3), input_shape=(sz, sz, 1),
activation='relu'))

classifier.add(MaxPooling2D(pool_size=(2, 2)))

# Second convolution layer and pooling

classifier.add(Convolution2D(32, (3, 3), activation='relu'))

# input_shape is going to be the pooled feature maps from the previous
convolution layer

classifier.add(MaxPooling2D(pool_size=(2, 2)))

# Flattening the layers

classifier.add(Flatten())

# Adding a fully connected layer

classifier.add(Dense(units=128, activation='relu'))

classifier.add(Dropout(0.40))

classifier.add(Dense(units=96, activation='relu'))

classifier.add(Dropout(0.40))

classifier.add(Dense(units=64, activation='relu'))
classifier.add(Dense(units=27, activation='softmax')) # softmax for more
than 2

# Compiling the CNN

classifier.compile(optimizer='adam', loss='categorical_crossentropy',
metrics=['accuracy']) # categorical_crossentropy for more than 2

# Step 2 - Preparing the train/test data and training the model

classifier.summary()

from keras.preprocessing.image import ImageDataGenerator

train_datagen = ImageDataGenerator(

        rescale=1./255,

        shear_range=0.2,

        zoom_range=0.2,

        horizontal_flip=True)

test_datagen = ImageDataGenerator(rescale=1./255)

training_set = train_datagen.flow_from_directory('data2/train',

                                                 target_size=(sz, sz),

                                                 batch_size=10,


                                                 color_mode='grayscale',

                                                 class_mode='categorical')

test_set = test_datagen.flow_from_directory('data2/test',

                                            target_size=(sz , sz),

                                            batch_size=10,

                                            color_mode='grayscale',

                                            class_mode='categorical')

classifier.fit(

        training_set,

        steps_per_epoch=1540, # No of images in training set
        epochs=20,
        validation_data=test_set,
        validation_steps=380) # No of images in test set
# Saving the model
model_json = classifier.to_json()
with open("model-bw.json", "w") as json_file:
    json_file.write(model_json)
print('Model Saved')
classifier.save_weights('model-bw.h5')
print('Weights saved')
```

4.6.4 app.py

```
from PIL import Image, ImageTk
import tkinter as tk
import cv2
import numpy as np
import os
from keras.models import model_from_json
import operator
import hunspell
from string import ascii_uppercase


class Application:
    def __init__(self):
        self.directory = 'model'
        self.hs = hunspell.HunSpell('/usr/share/hunspell/en_US.dic', '/usr/share/
hunspell/en_US.aff')
```

```
        self.vs = cv2.VideoCapture(0)
        self.current_image = None
        self.current_image2 = None
        self.json_file = open("model/model-bw.json", "r")
        self.model_json = self.json_file.read()
        self.json_file.close()
        self.loaded_model = model_from_json(self.model_json)
        self.loaded_model.load_weights("model/model-bw.h5")

        self.ct = {}
        self.ct['blank'] = 0
        self.blank_flag = 0
        for i in ascii_uppercase:
          self.ct[i] = 0
        print("Loaded model from disk")
        self.root = tk.Tk()
        self.root.title("Anuvadak: Sign language to Text Converter")
        self.root.protocol('WM_DELETE_WINDOW', self.destructor)
        self.root.geometry("900x1100")
        self.panel = tk.Label(self.root)
        self.panel.place(x = 135, y = 10, width = 640, height = 640)
        self.panel2 = tk.Label(self.root) # initialize image panel
        self.panel2.place(x = 460, y = 95, width = 310, height = 310)
        self.T = tk.Label(self.root)
        self.T.place(x=31,y = 17)
        self.T.config(text = "Anuvadak: \nSign Language to Text
Converter",font=("courier",40,"bold"))
```

```
        self.panel3 = tk.Label(self.root) # Current SYmbol
        self.panel3.place(x = 500,y=640)
        self.T1 = tk.Label(self.root)
        self.T1.place(x = 10,y = 640)
        self.T1.config(text="Character :",font=("Courier",40,"bold"))
        self.panel4 = tk.Label(self.root) # Word
        self.panel4.place(x = 220,y=700)
        self.T2 = tk.Label(self.root)
        self.T2.place(x = 10,y = 700)
        self.T2.config(text ="Word :",font=("Courier",40,"bold"))
        self.panel5 = tk.Label(self.root) # Sentence
        self.panel5.place(x = 350,y=760)
        self.T3 = tk.Label(self.root)
        self.T3.place(x = 10,y = 760)
        self.T3.config(text ="Sentence :",font=("Courier",40,"bold"))
        self.T4 = tk.Label(self.root)
        self.T4.place(x = 250,y = 820)
        self.T4.config(text = "Suggestions",fg="red",font = 
("Courier",40,"bold"))


        self.bt1=tk.Button(self.root, command=self.action1,height = 0,width = 
0)
        self.bt1.place(x = 26,y=890)
        #self.bt1.grid(padx = 10, pady = 10)
        self.bt2=tk.Button(self.root, command=self.action2,height = 0,width = 
0)
        self.bt2.place(x = 325,y=890)
        #self.panel3.place(x = 10,y=660)
        # self.bt2.grid(row = 4, column = 1, columnspan = 1, padx = 10, pady =
10, sticky = tk.NW)

        self.bt3=tk.Button(self.root, command=self.action3,height = 0,width =
0)

        self.bt3.place(x = 625,y=890)

        # self.bt3.grid(row = 4, column = 2, columnspan = 1, padx = 10, pady =
10, sticky = tk.NW)

        self.bt4=tk.Button(self.root, command=self.action4,height = 0,width =
0)

        self.bt4.place(x = 125,y=950)

        # self.bt4.grid(row = bt1, column = 0, columnspan = 1, padx = 10, pady
= 10, sticky = tk.N)

        self.bt5=tk.Button(self.root, command=self.action5,height = 0,width =
0)

        self.bt5.place(x = 425,y=950)

        # self.bt5.grid(row = 5, column = 1, columnspan = 1, padx = 10, pady =
10, sticky = tk.N)

        self.str=""

        self.word=""

        self.current_symbol="Empty"

        self.photo="Empty"

        self.video_loop()


    def video_loop(self):

        ok, frame = self.vs.read()

        if ok:

            cv2image = cv2.flip(frame, 1)

            x1 = int(0.5*frame.shape[1])

            y1 = 10
            x2 = frame.shape[1]-10
            y2 = int(0.5*frame.shape[1])
            cv2.rectangle(frame, (x1-1, y1-1), (x2+1, y2+1), (255,0,0) ,1)
            cv2image = cv2.cvtColor(cv2image, cv2.COLOR_BGR2RGBA)
            self.current_image = Image.fromarray(cv2image)
            imgtk = ImageTk.PhotoImage(image=self.current_image)
            self.panel.imgtk = imgtk
            self.panel.config(image=imgtk)
            cv2image = cv2image[y1:y2, x1:x2]
            gray = cv2.cvtColor(cv2image, cv2.COLOR_BGR2GRAY)
            blur = cv2.GaussianBlur(gray,(5,5),2)
            th3 =
cv2.adaptiveThreshold(blur,255,cv2.ADAPTIVE_THRESH_GAUSSIAN_
C,cv2.THRESH_BINARY_INV,11,2)
            ret, res = cv2.threshold(th3, 70, 255,
cv2.THRESH_BINARY_INV+cv2.THRESH_OTSU)
            self.predict(res)
            self.current_image2 = Image.fromarray(res)
            imgtk = ImageTk.PhotoImage(image=self.current_image2)
            self.panel2.imgtk = imgtk
            self.panel2.config(image=imgtk)
            self.panel3.config(text=self.current_symbol,font=("Courier",50))
            self.panel4.config(text=self.word,font=("Courier",40))
            self.panel5.config(text=self.str,font=("Courier",40))
            predicts=self.hs.suggest(self.word)
            if(len(predicts) > 0):
               self.bt1.config(text=predicts[0],font = ("Courier",20))
            else:
                self.bt1.config(text="")
            if(len(predicts) > 1):
                self.bt2.config(text=predicts[1],font = ("Courier",20))
            else:
                self.bt2.config(text="")
            if(len(predicts) > 2):
                self.bt3.config(text=predicts[2],font = ("Courier",20))
            else:
                self.bt3.config(text="")
            if(len(predicts) > 3):
                self.bt4.config(text=predicts[3],font = ("Courier",20))
            else:
                self.bt4.config(text="")
            if(len(predicts) > 4):
                self.bt5.config(text=predicts[4],font = ("Courier",20))
            else:
                self.bt5.config(text="")
        self.root.after(30, self.video_loop)
def predict(self,test_image):
        test_image = cv2.resize(test_image, (128,128))
        result = self.loaded_model.predict(test_image.reshape(1, 128, 128, 1))
        prediction={}
        prediction['blank'] = result[0][0]
        inde = 1
        for i in ascii_uppercase:
            prediction[i] = result[0][inde]
            inde += 1
        #LAYER 1
        prediction = sorted(prediction.items(), key=operator.itemgetter(1), 
reverse=True)
        self.current_symbol = prediction[0][0]
        if(self.current_symbol == 'blank'):
            for i in ascii_uppercase:
                self.ct[i] = 0
        self.ct[self.current_symbol] += 1
        if(self.ct[self.current_symbol] > 60):
            for i in ascii_uppercase:
                if i == self.current_symbol:
                    continue
                tmp = self.ct[self.current_symbol] - self.ct[i]
                if tmp < 0:
                    tmp *= -1
                if tmp <= 20:
                    self.ct['blank'] = 0
                    for i in ascii_uppercase:
                        self.ct[i] = 0
                    return
            self.ct['blank'] = 0
            for i in ascii_uppercase:
                self.ct[i] = 0
            if self.current_symbol == 'blank':
                if self.blank_flag == 0:
                    self.blank_flag = 1
                    if len(self.str) > 0:
                        self.str += " "
                    self.str += self.word
                    self.word = ""
            else:
                if(len(self.str) > 16):
                    self.str = ""
                self.blank_flag = 0
                self.word += self.current_symbol
def action1(self):
      predicts=self.hs.suggest(self.word)
      if(len(predicts) > 0):
            self.word=""
            self.str+=" "
            self.str+=predicts[0]
def action2(self):
      predicts=self.hs.suggest(self.word)
      if(len(predicts) > 1):
            self.word=""
            self.str+=" "
            self.str+=predicts[1]
def action3(self):
      predicts=self.hs.suggest(self.word)
      if(len(predicts) > 2):
            self.word=""
            self.str+=" "
            self.str+=predicts[2]
def action4(self):
```

```
            predicts=self.hs.suggest(self.word)
            if(len(predicts) > 3):
              self.word=""
              self.str+=" "
              self.str+=predicts[3]
    def action5(self):
            predicts=self.hs.suggest(self.word)
            if(len(predicts) > 4):
              self.word=""
              self.str+=" "
              self.str+=predicts[4]
    def destructor(self):
        print("Closing Application...")
        self.root.destroy()
        self.vs.release()
        cv2.destroyAllWindows()
    def destructor1(self):
        print("Closing Application...")
        self.root1.destroy()
print("Starting Application...")
pba = Application()
pba.root.mainloop()
```

4.7 CHALLENGES FACED

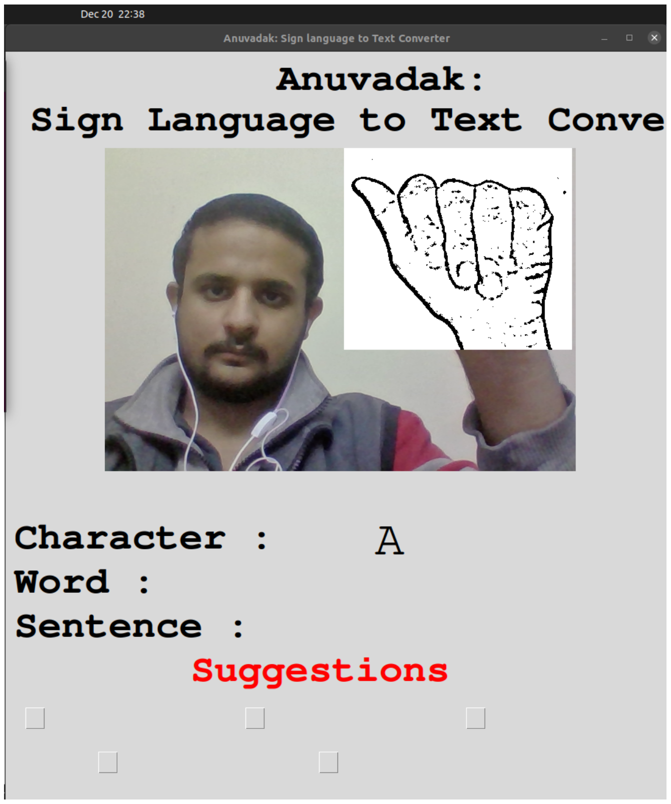
There were many challenges faced by us during the making of this project:

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

CHAPTER-5
RESULTS AND APPLICATIONS

5.1 Results

1 An application interface that interprets American Sign Language to Text in real time to assist deaf and dumb for communicating with them effectively eliminating the requirement of a translating individual has been developed successfully.



Anuvadak:
Sign Language to Text Conve

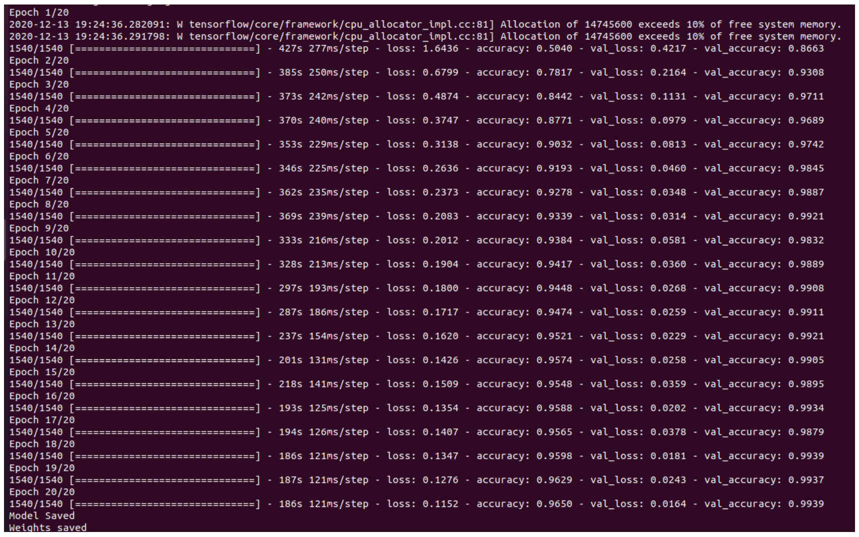
Character : A
Word :
Sentence :
Suggestions

Figure 5.1 Anuvadak: Sign Language to Text Converter

2 The model devised achieved an accuracy of 96.5% which is greater than the many of the models that we researched. The sign had to be kept stable for at least 60 frames in front of the camera.

• Other frame capturing values like 40 and 80 were also tried for prediction. For 40 frames, the predicted values were flickering much and for 80 frames the time taken to predict the correct alphabet was high.

• Hence frame value 60 proved to be the best optimised value for the sign detection.



```
Epoch 1/20
2020-12-13 19:24:36.282091: W tensorflow/core/framework/cpu_allocator_impl.cc:81] Allocation of 14745600 exceeds 10% of free system memory.
2020-12-13 19:24:36.291798: W tensorflow/core/framework/cpu_allocator_impl.cc:81] Allocation of 14745600 exceeds 10% of free system memory.
1540/1540 [==============================] - 427s 277ms/step - loss: 1.6436 - accuracy: 0.5040 - val_loss: 0.4217 - val_accuracy: 0.8663
Epoch 2/20
1540/1540 [==============================] - 385s 250ms/step - loss: 0.6799 - accuracy: 0.7817 - val_loss: 0.2164 - val_accuracy: 0.9308
Epoch 3/20
1540/1540 [==============================] - 373s 242ms/step - loss: 0.4874 - accuracy: 0.8442 - val_loss: 0.1131 - val_accuracy: 0.9711
Epoch 4/20
1540/1540 [==============================] - 370s 240ms/step - loss: 0.3747 - accuracy: 0.8771 - val_loss: 0.0979 - val_accuracy: 0.9689
Epoch 5/20
1540/1540 [==============================] - 353s 229ms/step - loss: 0.3138 - accuracy: 0.9032 - val_loss: 0.0813 - val_accuracy: 0.9742
Epoch 6/20
1540/1540 [==============================] - 346s 225ms/step - loss: 0.2636 - accuracy: 0.9193 - val_loss: 0.0460 - val_accuracy: 0.9845
Epoch 7/20
1540/1540 [==============================] - 362s 235ms/step - loss: 0.2373 - accuracy: 0.9278 - val_loss: 0.0348 - val_accuracy: 0.9887
Epoch 8/20
1540/1540 [==============================] - 369s 239ms/step - loss: 0.2083 - accuracy: 0.9339 - val_loss: 0.0314 - val_accuracy: 0.9921
Epoch 9/20
1540/1540 [==============================] - 333s 216ms/step - loss: 0.2012 - accuracy: 0.9384 - val_loss: 0.0581 - val_accuracy: 0.9832
Epoch 10/20
1540/1540 [==============================] - 328s 213ms/step - loss: 0.1904 - accuracy: 0.9417 - val_loss: 0.0360 - val_accuracy: 0.9889
Epoch 11/20
1540/1540 [==============================] - 297s 193ms/step - loss: 0.1800 - accuracy: 0.9448 - val_loss: 0.0268 - val_accuracy: 0.9908
Epoch 12/20
1540/1540 [==============================] - 287s 186ms/step - loss: 0.1717 - accuracy: 0.9474 - val_loss: 0.0259 - val_accuracy: 0.9911
Epoch 13/20
1540/1540 [==============================] - 237s 154ms/step - loss: 0.1620 - accuracy: 0.9521 - val_loss: 0.0229 - val_accuracy: 0.9921
Epoch 14/20
1540/1540 [==============================] - 201s 131ms/step - loss: 0.1426 - accuracy: 0.9574 - val_loss: 0.0258 - val_accuracy: 0.9905
Epoch 15/20
1540/1540 [==============================] - 218s 141ms/step - loss: 0.1509 - accuracy: 0.9548 - val_loss: 0.0359 - val_accuracy: 0.9895
Epoch 16/20
1540/1540 [==============================] - 193s 125ms/step - loss: 0.1354 - accuracy: 0.9588 - val_loss: 0.0202 - val_accuracy: 0.9934
Epoch 17/20
1540/1540 [==============================] - 194s 126ms/step - loss: 0.1407 - accuracy: 0.9565 - val_loss: 0.0378 - val_accuracy: 0.9879
Epoch 18/20
1540/1540 [==============================] - 186s 121ms/step - loss: 0.1347 - accuracy: 0.9598 - val_loss: 0.0181 - val_accuracy: 0.9939
Epoch 19/20
1540/1540 [==============================] - 187s 121ms/step - loss: 0.1276 - accuracy: 0.9629 - val_loss: 0.0243 - val_accuracy: 0.9937
Epoch 20/20
1540/1540 [==============================] - 186s 121ms/step - loss: 0.1152 - accuracy: 0.9650 - val_loss: 0.0164 - val_accuracy: 0.9939
Model Saved
Weights saved
```

Figure 5.2 Accuracy achieved in our model is 96.5% as shown above

3 Our project is implemented using deep learning model that is implemented in python. The project requires only a webcam of a laptop and a computer system to be deployed. This is much cost effective and efficient than the hardware implemented system in [9] using flex sensors. The maintenance cost of our project is very less. We only need a functional webcam to detect the sign.

4. Our model provides suggestions based on the current word being translated from the word in the US dictionary.

- For eg: If a speech or hearing impaired person wants to say “ELECTRON” to a normal person in sign language with the help of our project ; all they have to do is start conversing in sign language for the initial alphabets(here ‘E’,’L’,’E’,’C’,’T’) and our project would provide 5 suggestions (like ‘ELECTOR’, ‘ELECTRON’, ‘ELECTRA’, ‘ELECTROCUTE’) accordingly that matches with the initials written so far.

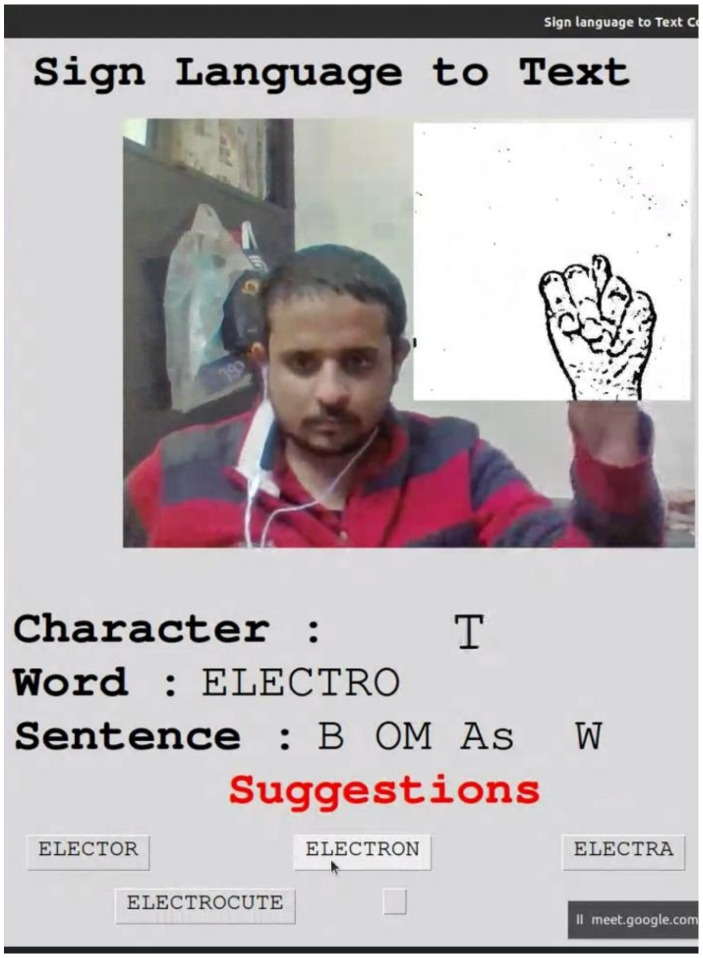


Figure 5.3 Suggestions given based on the current word

5. The suggestions given by the hunspell library helps to detect the correct spelling mistakes that may be introduced due to wrong detections of an alphabet or wrong knowledge of spellings of a word.

6. We can make words by combining multiple characters and hence can also formulate sentences with the help of generated words.This is the highlighting feature of our project which is not available in any of the models that we researched.

5.2 APPLICATIONS

- According to the National Deaf Association (NAD), 18 million people are estimated to be deaf in India.Hence a project like this could cater a huge percentage of such impaired community by providing a helping hand to communicate with the rest of the world using sign language.This leads to the elimination of the middle person who generally acts as a medium of translation.Due to the simplicity of the model, it can also be implemented in mobile application and is regarded as our future plan to do so.

- Deaf people do not have that many options for communicating with a hearing person, and all of the alternatives do have major flaws. Interpreters aren't usually available, and also could be expensive.Our project as mentioned before is quite economical and requires minimal amount of management.Hence is quite advantageous in terms of cost reduction.
- Pen and paper approach is just a bad idea: it's very uncomfortable, messy, time-consuming for both deaf and hearing person.Our translator “ANUVADAK” thereby solves the problem by removing the need of any written approach for communication.
- This project idea can even be implemented for deaf and dumb community in various places like schools and colleges(for teaching purposes),airports(for security check or communicating during on-boarding or in flight),community service agencies and courts(to adjudicate legal disputes between parties),doctor offices(to get to know about the illness properly)

CHAPTER-6
CONCLUSIONS AND FUTURE SCOPE

6.1 CONCLUSIONS

- With the continuous breakthrough of neural network in artificial intelligence, computer vision and other related fields, neural network has brought dynamic new methods to the study of sign language recognition based on vision.
- In this report, we have proposed a recognition method of finger-spellings in American Sign Language, which uses computer vision based on pattern recognition and Convolutional Neural Network(a Deep Learning algorithm).
- We have trained our model for a total of 27 symbols(26 English alphabets & a ‘blank’ symbol for spacing in between the sentences). We were able to achieve an accuracy of 96.5% for our CNN classifier.
- A basic GUI application is created to test our classifier in this system.The application allows users to form characters, words and sentences according to their needs.Moreover by providing multiple suggestions for the corresponding word being formed further helps in communication.
- The main objective has been achieved, that is, the need for an interpreter has been eliminated.

6.2 FUTURE SCOPE

In this report, a functional real time vision based american sign language recognition for D&M people have been developed for asl alphabets.We achieved final accuracy of 95% on our dataset. We are able to improve our prediction after implementing two layers of algorithms in which we verify and predict symbols which are more similar to each other. This way we are able to detect almost all the symbols provided that they are shown properly, there is no noise in the background and lighting is adequate. Our project aims to create a computer application and train a model which when shown a real time video of hand gestures of American Sign Language shows the output for that particular sign in text format on the screen.

- This project did not focus on Facial Expressions although it is well known that facial expressions convey important part of sign languages. This system can be implemented in many application areas examples include accessing government websites whereby no video clip for deaf and mute is available or filling out forms online whereby no interpreter may be present to help.
- This technology could mean a new way for about 70 million people with hearing and speech impairment to verbally communicate and connect to people around them. This project aims to lower the communication gap between the deaf or mute community and the normal world.
- In future work, proposed system can be developed and implemented using Raspberry Pi. Image Processing part should be improved so that System would be able to communicate in both directions i.e.it should be capable of converting normal language to sign language and vice versa.
- We will try to recognize signs which include motion. Moreover we will focus on converting the sequence of gestures into text i.e. word and sentences and then converting it into the speech which can be heard.
- Although this project has achieved high accuracy, the data set is limited in scope and does not include word-level sign language.With broad application and development space, sign language recognition still has much room for improvement .
- At the same time, most methods of sign language recognition now only consider the accuracy of the algorithm. However, for the application of sign language recognition in real scene, real-time performance is another important index. Therefore, it is also a direction worthy of breakthrough about how to improve the speed of hand locating and recognition of sign language words.

REFERENCES

[1] M.M.Gharasuie, H.Seyedarabi, “Real-time Dynamic Hand Gesture Recognition using Hidden Markov Models”, 8th Iranian Conference on Machine Vision and Image Processing (MVIP), 2013.

[2] Pradumn Kumar, Upasana Dugal, “Tensorflow Based Image Classification using Advanced Convolutional Neural Network”, International Journal of Recent Technology and Engineering (IJRTE) ISSN: 2277-3878, Volume-8 Issue-6, March 2020, March, 2020.

[3] Xin Jia, “Image Recognition Method Based on Deep Learning”, CCDC, DOI: 978-1-5090-4657-7/17, 2017.

[4] Jiudong Yang, Jianping Li, “Application Of Deep Convolution Neural Network”, IEEE, DOI: 978-1-5386-1010-7/17, 2017.

[5] Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, Ruslan Salakhutdinov, “Dropout: A Simple Way to Prevent Neural Networks from Overfitting”, Journal of Machine Learning Research 15 (2014) 1929-1958, 2014.

[6] Ankit Ojha, Ayush Pandey, Shubham Maurya, Abhishek Thakur, Dr. Dayananda P, “Sign Language to Text and Speech Translation in Real Time Using Convolutional Neural Network”, International Journal of Engineering Research & Technology (IJERT), ISSN: 2278-0181, NCAIT - 2020 Conference Proceedings.

[7] Nobuhiko MUKAI, Naoto HARADA, Youngha CHANG, “Japanese Fingerspelling Recognition based on Classification Tree and Machine Learning”, NICOGRAPH International, 2017.

[8] Qi Wang, Zequn Qin, Feiping Nie, Yuan Yuan, “Convolutional 2D LDA for Nonlinear Dimensionality Reduction”, Proceedings of the Twenty-Sixth International Joint Conference on Artificial Intelligence (IJCAI-17).

[9] Aarthi M, Vijayalakshmi P, “ Sign Language To Speech Conversion”, Fifth International Conference On Recent Trends In Information Technology, 2016.

APPENDIX

Operating System Used:

Ubuntu LTS 20.04.

Ubuntu is a Linux distribution based on Debian and mostly composed of free and open-source software. Ubuntu is officially released in three editions: Desktop,Server, and Core for Internet of things devices and robots.All the editions can run on the computer alone, or in a virtual machine.Ubuntu is a popular operating system for cloud computing, with support for OpenStack. Ubuntu's default desktop has been GNOME, since version 17.10.

Ubuntu is released every six months, with long-term support (LTS) releases every two years. As of 22 October 2020, the most recent long-term support release is 20.04 ("Focal Fossa"), which is supported until 2025 under public support and until 2030 as a paid option. The latest standard release is 20.10 ("Groovy Gorilla"), which is supported for nine months.

Figure A.1 Ubuntu

Ubuntu is developed by Canonical, and a community of other developers, under a meritocratic governance model. Canonical provides security updates and support for each Ubuntu release, starting from the release date and until the release reaches its designated end-of-life (EOL) date.Canonical generates revenue through the sale of premium services related to Ubuntu.

Python

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

Figure A.2 Python

Often, programmers fall in love with Python because of the increased productivity it provides. Since there is no compilation step, the edit-test-debug cycle is incredibly fast. Debugging Python programs is easy: a bug or bad input will never cause a segmentation fault. Instead, when the interpreter discovers an error, it raises an exception. When the program doesn't catch the exception, the interpreter prints a stack trace. A source level debugger allows inspection of local and global variables, evaluation of arbitrary expressions, setting breakpoints, stepping through the code a line at a time, and so on. The debugger is written in Python itself, testifying to Python's introspective power. On the other hand, often the quickest way to debug a program is to add a few print statements to the source: the fast edit-test-debug cycle makes this simple approach very effective.

Convolution Neural network

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

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

CNNs use relatively little pre-processing compared to other image classification algorithms. This means that the network learns the filters that in traditional algorithms were hand-engineered. This independence from prior knowledge and human effort in feature design is a major advantage.

They have applications in image and video recognition, recommender systems, image classification, medical image analysis, and natural language processing.

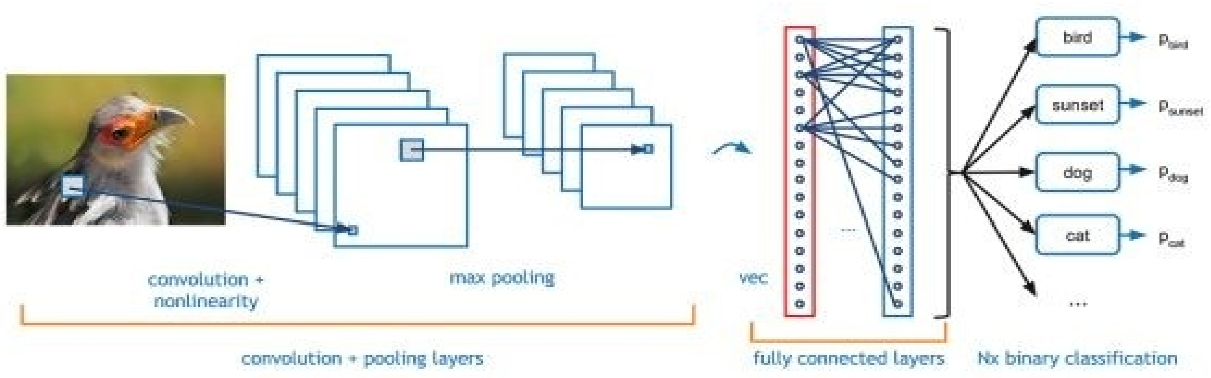


Figure A.3 CNN