A Report of Minor Project on

Sign Language Recognition

Submitted by

Vikas Sharma (1904086)
Kamlesh Kumar(1904084)
Md Kashif Raza(1904103)

in partial fulfillment for the award of the degree

of

BACHELOR OF TECHNOLOGY

Under the Supervision
Of
Dr. BHARAT GUPTA

Department of Electronics and Communication Engineering
NATIONAL INSTITUTE OF TECHNOLOGY PATNA
NOVEMBER 2022

Contents

ABSTRACT

It becomes very difficult to converse with deaf and mute people, so we need a language structure to understand what they are trying to say. Language is a barrier in between normal people and mute people. Our aim is to create an interface that convert sign language to the text by which everyone can understand the sign language of the peoples who are unable to speak.

For this purpose, we will train a model to recognize hand gestures. The model can be trained using KNN, support vector machine, Logistic regression and CNN. But CNN is more accurate and effective in case of image recognition, that’s why we have train our model using CNN algorithm. After training the model on the American sign language dataset we can predict the text with good accuracy, but the model fails when we use OpenCV to recognize hand gestures due inaccuracy of dataset which does not provide the enough information to the model to predict real world objects.

Even though the proposed model gives 99 % of accuracy with the dataset from Kaggle.

Introduction

Communication can be done in several ways, one of which is by using spoken language. However, not all humans can communicate using well spoken language, in other words, the human is one of the people with disabilities, namely deaf. Deafness is a medical term that describes a condition with limited oral speech. According to the World Health Organization in 2020, there are 466 million deaf people, and it is estimated that by 2050 people with hearing impairment will increase to 900 million people. In communicating, deaf people use Sign language.

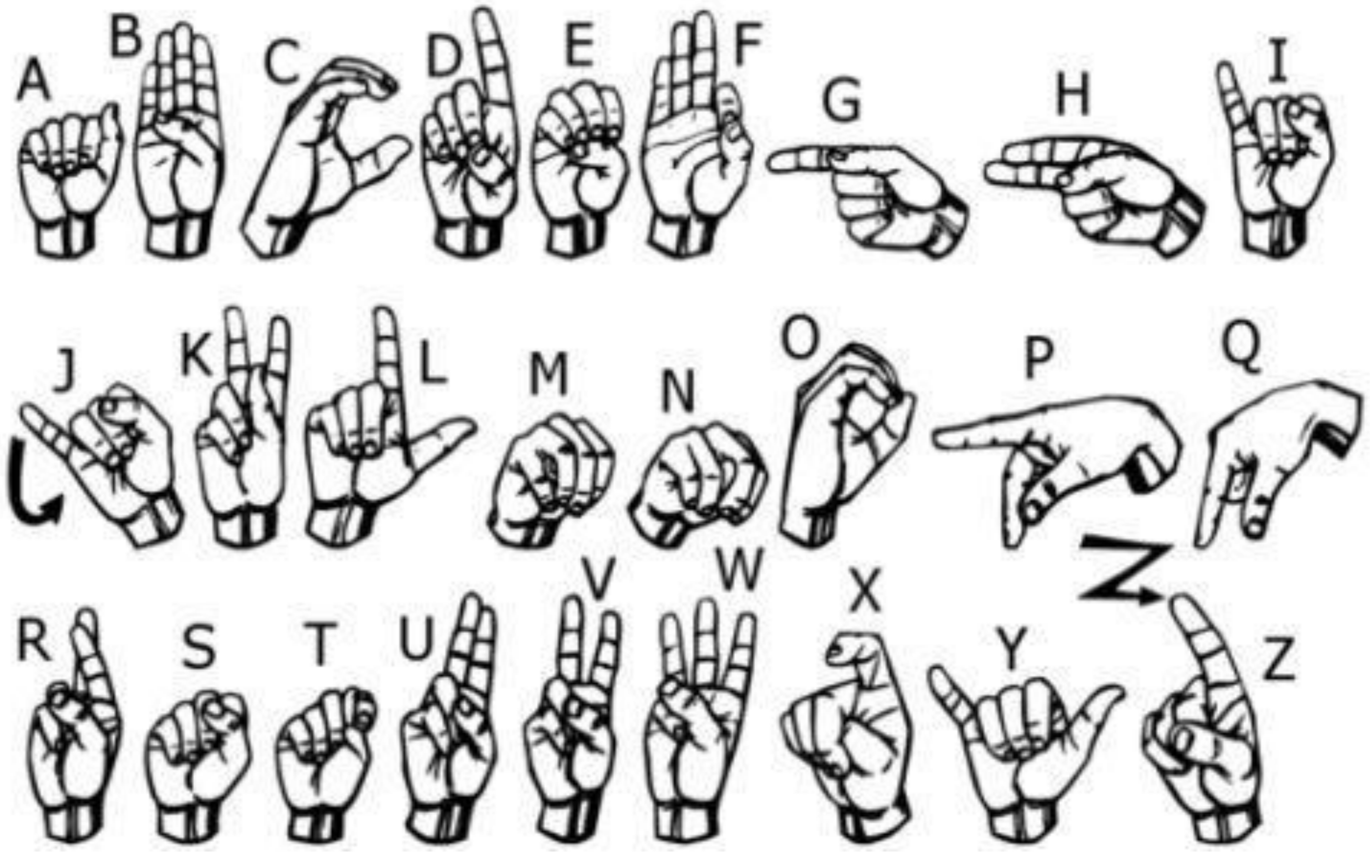
Image processing can translate images into text, where the data used in image processing can be either static or dynamic images. Image processing implementation can use several libraries, one of which is the hand key point library. Image processing cannot stand alone as a sign language translator but requires an algorithm that functions as a detection and classification tool. Several studies were conducted regarding the application and comparison of deep learning methods. Deep learning is a type of artificial neural network that uses metadata as input and processes it using several hidden layer non-linear transformations from the input data to calculate the output value. One application of the deep learning method is the classification process. The classification algorithm can be implemented in various types of datasets, the results of this implementation state that each algorithm has a different ability to detect objects.

One of the classification algorithms is Convolutional Neural Network (CNN).
CNN is proven to have advantages in image processing, besides that CNN also can work automatically to learn the representation of the features that exist in the visual input effectively.

Community of people who are not able to hear and use sign language for communication or who can't speak as well as hear, are known as deaf-mute community. This term is also used for someone who can't hear but can speak a little bit. They face many challenges in their life that restrict them to do even simple task sometimes.

Being not able to share what they feel to the people around them, makes them less social and introverts. There are various communication channels available for them to communicate like-lip reading, sign language, notes writing, helper pages etc. but still they face many problems. This problem is not limited to speech or hearing impairment but lack of awareness too. People are not aware enough about their problem and available solutions.

Many people want deaf-mute child to learn oral way of learning; means teach them reading lips and speaking rather than making them learn sign language. There has been debate over this topic since a long time in America as well as in India. Many parents are afraid that telling others about their child's disability will affect their reputation in the society. And this is also the reason many of them do not want their child to learn sign language as it will be visually clear that they have some kind of disability.


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
Z

Literature Review

In recent years there have been great strides in building classifiers for image detection and recognition on various datasets using various machine learning algorithms. Deep learning, in particular, has shown improvement in accuracy on various datasets. Some of the works have been described below:

Norhidayu binti Abdul Hamid et al. evaluated the performance on MNIST datasets using 3 different classifiers: SVM (support vector machines), KNN (K-nearest Neighbor) and CNN (convolutional neural networks). The Multilayer perceptron didn't perform well on that platform as it didn't reach the global minimum rather remained stuck in the local optimal and couldn't recognize digit 9 and 6 accurately. Other classifiers, performed correctly and it was concluded that performance on CNN can be improved by implementing the model on Keras platform. Mahmoud M. Abu Gosh et al.

Zafar Ahmed Ansari and Gaurav Harit have carried out extensive research in the field of correctly categorising Indian Sign Language gestures. They have classified 140 classes which include finger-spelling numbers, alphabets, and other common phrases as well. For the dataset, a Kinect sensor is used. RGB images of resolution 640 × 480 are captured along with their depth data. Depth values of each pixel are captured in the depth data of the image. Each pixel of the image corresponds to an value in the depth data file.

The recognition of sign symbols by Divya Deora and Nikesh Bajaj is done with PCA (Principal Component analysis). The paper also proposes recognition with neural networks. The data they acquired was using a 3 mega pixels camera and hence the quality was poor. For each sign they took 15 images and stored it in their database. The small dataset was one of the reasons why their results were not satisfactory. They performed segmentation and separated the RGB into components and performed a simple boundary pixel analysis. They have mentioned that even though combining finger tip algorithm with PCA provided a good result, a better output can be obtained using neural networks.

Sign language recognition carried out by Lionel et aluses Convolutional Neural Network (CNN). To automate the process of sign language recognition, two steps were carried out - feature extraction and classifications of the actions.

The first step is performed using CNN and the next step using an Artificial Neural Network. The dataset includes a total of 20 distinct Italian gestures that have been signed by 27 people in different surroundings. Videos are recorded using Microsoft Kinect. A total of 6600 images were used for development purposes out of which 4600 were used for training and the rest for validation. The images were preprocessed to crop the higher part of the hand and torso. For all gestures, only one side is required to be learnt by the model. Max pooling is used for feature extraction and classification. It consists of 2 CNNs, the output of which enters an ANN. The ANN combines the output of the two CNNs. A CPU was used for augmentation of the data and a GPU for training the model. Zooming rotation and transformation was done as a part of data distortion.

Training using this model resulted in an accuracy of 91.7%, and a mean Jaccard index of 0.789 was obtained using the model in ChaLearn 2014.

There has been various researches done in the area of recognition of hand gestures and after studying some of them it has been found that following are the basic steps to be followed in this process:-

A. Gathering Data

There are various ways to collect data. In this particular sign language detection topic, mainly these two approaches to collect data can be used-

Using devices with sensors: In this approach electromagnetic devices with sensors can be used to collect data for example wearable gloves with sensors. These sensors take out the necessary information but this technique is quite costly.

Vision based: In this approach camera is used to collect images for the model. So, it gives us a plus point that no external device is used & no extra expense made. All we need is a computer with webcam. Despite of advances, it also has to deal with some issues like skin color variation, difference in view angles, camera quality, scaling size of hand etc. One of its applications is shown in human-machine interface application for automotives like car.

B. Preprocessing Gathered Data & Feature Extraction

Preprocessing of data means converting raw data into usable form so that it can be fed to the model for training. Features are nothing but patterns present in your data that helps in extraction necessary information for further model training process. It removes unnecessary data hence simplify your data. For data collected from gloves, we don't need to do much thing as it already gives precise results. But for vision based approach preprocessing becomes crucial. A filter called Gaussian Blur can be applied which is present in OpenCV library on the raw images. And in this project image background is kept single colored so that there is no need to do segmentation on the basis of skin tones.

C. Classifying Gestures

Various techniques like Hidden Markov Model , Naïve bayes Classifier , Convolution Neural network etc can be used in classification of gestures.

Data Augmentation

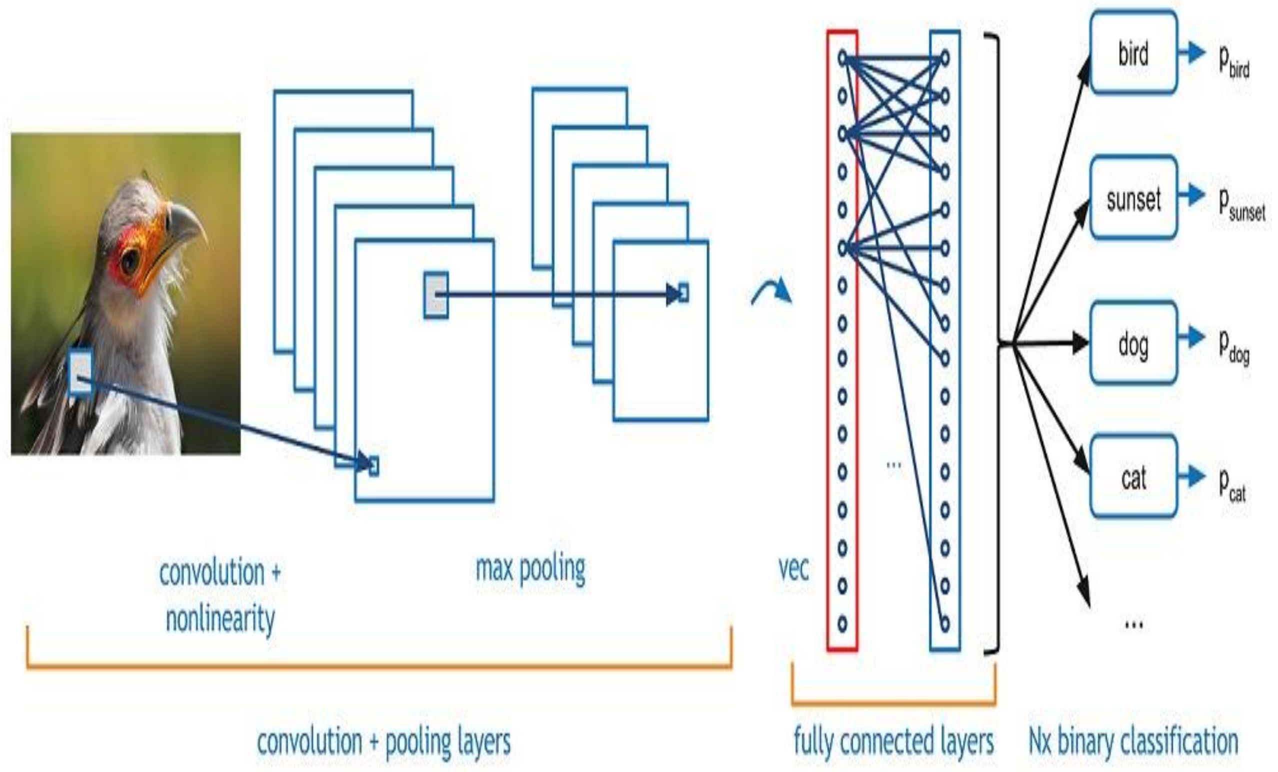
In Data augmentation from one image, we generate more images. Therefore, we can have a variety in our dataset, and it is also used to increase the size of the dataset to avoid overfitting. Our data augmentation includes random rotation ranging 10 degrees and random height and width shift in the range of 0.1, random zooming, and random flips. This, we implemented using Keras image pre-processing module.

Model Building

The model is developed by using the combination of convolution, max pooling, and flattening layers in Keras. We also used batch normalization and dropout layers to prevent the model from overfitting the data. A callback is used to determine whether the validation set accuracy did not increase even after 2 (patience level) consecutive epochs during the model fitting stage. Then, the learning rate will be altered by a factor of 0.5.

Convolutional Neural Network (CNN)

A Convolutional Neural Network is a special class of neural networks that are built with the ability to extract unique features from image data. For instance, they are used in face detection and recognition because they can identify complex features in image data.



How Do Convolutional Neural Networks Wor?

Like other types of neural networks, CNNs consume numerical data. Therefore, the images fed to these networks must be converted to a numerical representation. Since images are made up of pixels, they are converted into a numerical form that is passed to the CNN.

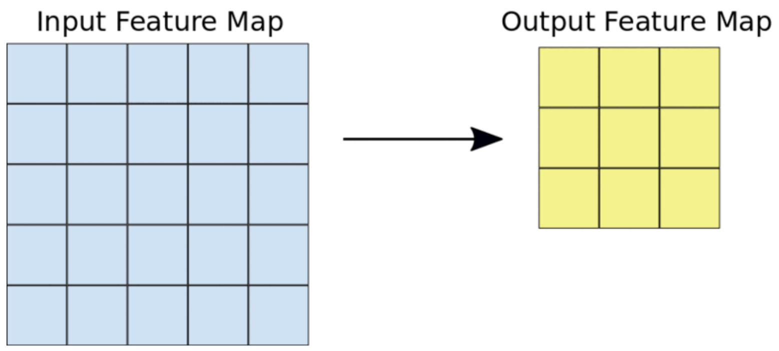
Convolution

Reducing the size of the numerical representation sent to the CNN is done via the convolution operation. This process is vital so that only features that are important in classifying an image are sent to the neural network. Apart from improving the accuracy of the network, this also ensures that minimal compute resources are used in training the network.

The result of the convolution operation is referred to as a feature map, convolved feature, or activation map. Applying a feature detector is what leads to a feature map. The feature detector is also known by other names such as kernel or filter.

The kernel is usually a 3 by 3 matrix.

Performing an element-wise multiplication of the kernel with the input image and summing the values, outputs the feature map. This is done by sliding the kernel on the input image. The sliding happens in steps known as strides. The strides and the size of the kernel can be set manually when creating the CNN.



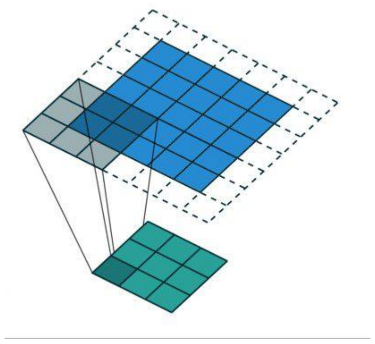
A 3 by 3 convolutions operation.

For example, given a 5 by 5 input, a kernel of 3 by 3 will output a 3 by 3 output feature map.

Padding

In the above operations, we have seen that the size of the feature map reduces as part of applying the convolution operation. What if you want the size of the feature map to be the same size as that of the input image? That is achieved through padding.

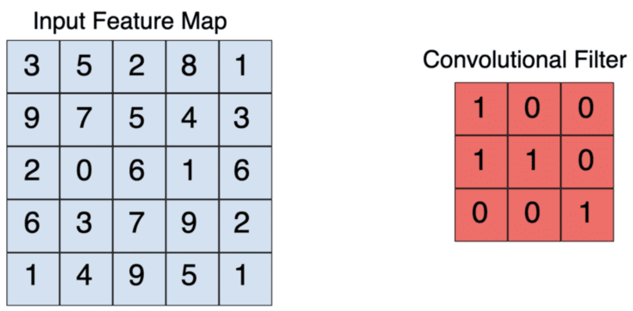
Padding involves increasing the size of the input image by “padding” the images with zeros. As a result, applying the filter to the image leads to a feature map of the same size as the input image.



The uncolored area represents the padded area.

Padding reduces the amount of information lost in the convolution operation. It also ensures that the edges of the images are factored more often in the convolution operation.

When building the CNN, you will have the option to define the type of padding you want or no padding at all. The common options here are valid or same. Valid means no padding will be applied while same means that padding will be applied so that the size of the feature map is the same as the size of the input image.



Input Feature Map

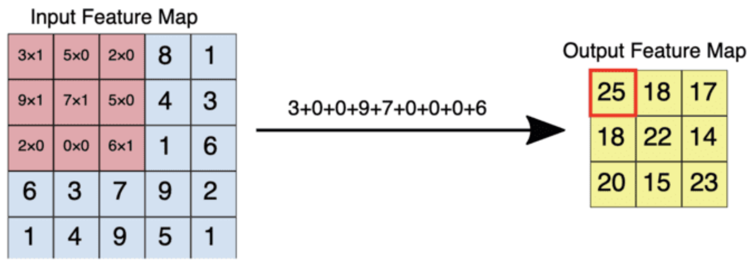
3	5	2	8	1
9	7	5	4	3
2	0	6	1	6
6	3	7	9	2
1	4	9	5	1

Convolutional Filter

1	0	0
1	1	0
0	0	1

A 3 by3 kernel reduces a 5 by 5 input to a 3 by 3 output

Here is what the element-wise multiplication of the above feature map and filter would look like.

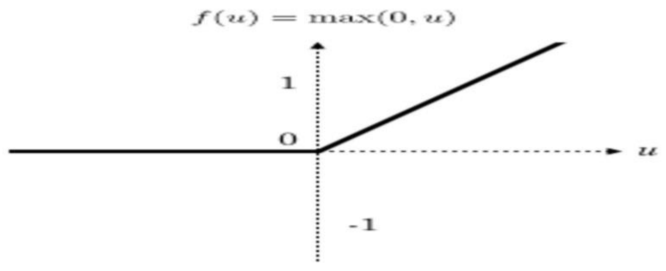


Element-wise multiplication of a 5 by input with a 3 by 3 filter.

Activation functions

A Rectified Linear Unit (ReLU) transformation is applied after every convolution operation to ensure non-linearity. ReLU is the most popular activation function but there are other activation functions to choose from.

After the transformation, all values below zero are returned as zero while the other values are returned as they are.

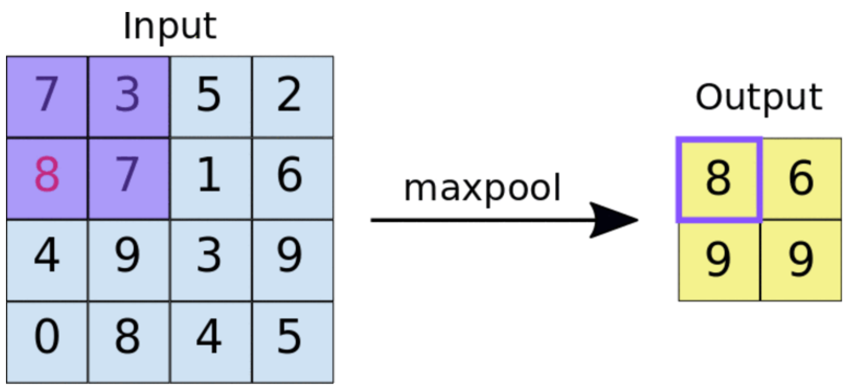


ReLu function plot

Pooling

 In this operation, the size of the feature map is reduced further. There are various pooling methods.

A common technique is max-pooling. The size of the pooling filter is usually a 2 by 2 matrix. In max-pooling, the 2 by 2 filter slides over the feature map and picks the largest value in a given box. This operation results in a pooled feature map.



Applying a 2 by 2 pooling filter to a 4 by 4 feature map.

Pooling forces the network to identify key features in the image irrespective of their location. The reduced image size also makes training the network faster.

Dropout Regularization

Applying Dropout Regularization is a common practice in CNNs. This involves randomly dropping some nodes in layers so that they are not updated during back-propagation. This prevents overfitting.

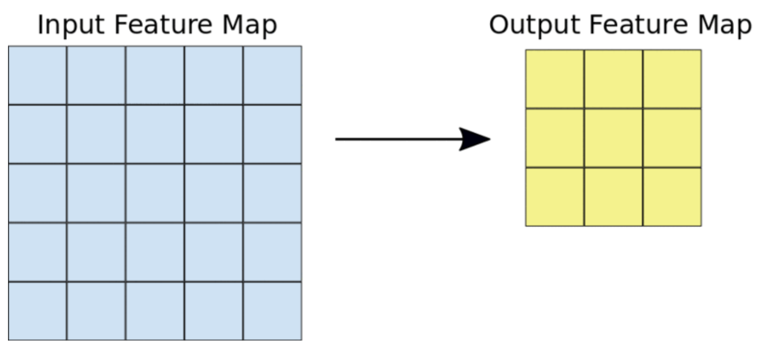
Flattening

Flattening involves transforming the pooled feature map into a single column that is passed to the fully connected layer. This is a common practice during the transition from convolutional layers to fully connected layers.

Fully connected layers

The flattened feature map is then passed to a fully connected layer. There might be several fully connected layers depending on the problem and the network. The last fully connected layer is responsible for outputting the prediction.

An activation function is used in the final layer depending on the type of problem. A sigmoid activation is used for binary classification, while a softmax activation function is used for multi-class image classification.



Fully connected convolutional neural network

Tools, modules, functions, and models used:

NumPy is a general-purpose array-processing package. It provides a high-performance multidimensional array object, and tools for working with these arrays. It is the fundamental package for scientific computing with Python.

Pandas is an open source Python package that is most widely used for data science/data analysis and machine learning tasks. It is built on top of another package named Numpy, which provides support for multi-dimensional arrays.

Matplotlib is a comprehensive library for creating static, animated, and interactive visualizations in Python. Matplotlib makes easy things easy and hard things possible. Create publication quality plots. Make interactive figures that can zoom, pan, update

TensorFlow is an end-to-end open source platform for machine learning. It has a comprehensive, flexible ecosystem of tools, libraries, and community resources that lets researchers push the state-of-the-art in ML and developers easily build and deploy ML-powered applications.

Keras is a high-level, deep learning API developed by Google for implementing neural networks. It is written in Python and is used to make the implementation of neural networks easy. It also supports multiple backend neural network computation.

Scikit-learn is probably the most useful library for machine learning in Python. The sklearn library contains a lot of efficient tools for machine learning and statistical modeling including classification, regression, clustering and dimensionality reduction.

CHALLENGES AND ISSUES

- There were many challenges faced by us during the project. The very first issue we faced was of selecting the right image classifier algorithm as there are many like KNN, Naive Bayes classifier, etc . As we are working with images we need more accuracy to predict the images

- The second issue was with the accuracy which is not up to the mark as we were expected.

- At last after saving the high accuracy model we tested on the real world data (hand signs) using openCv and found that the accuracy was very low.

- After analysing the whole project and the model we found that the dataset used for this project from Kaggle does not provide enough information so that the model recognise real hand sign.

Approach to the problem

- To solve the first issue we end up choosing CNN as it designed to automatically and adaptively learn spatial hierarchies of features through backpropagation by using multiple building blocks, such as convolution layers, pooling layers, and fully connected layers as well as fits our requirements.

- we used the data agumentation to solve our second issue, which is a process of artificially increasing the amount of data by generating new data points from existing data. This includes adding minor alterations to data or using machine learning models to generate new data points in the latent space of original data to amplify the dataset.

- The resolution to the last issue we faced is disussed in the future scope.

Note :

- KNN stands for K-Nearest neighbours. It is also an algorithm popularly used for multi-class classification.
- It is implemented in sklearn using KNeighborsClassifier class.
- Implementing a Naive Bayes classifier
- It is the most fundamental machine learning classifier, also abbreviated as NB. It works based on Bayes Theorem and has independent features.
- It is implemented in sklearn using the GaussianNB class.

Data Collections

we have used the American Sign Language (ASL) data set that is provided by MNIST and it is publicly available at Kaggle. This dataset contains 27455 training images and 7172 test images all with a shape of 28 x 28 pixels. These images belong to the 25 classes of English alphabet starting from A to Y (No class labels for Z because of gesture motions).

The dataset on Kaggle is available in the CSV format where training data has 27455 rows and 785 columns. The first column of the dataset represents the class label of the image and the remaining 784 columns represent the 28 x 28 pixels. The same paradigm is followed by the test data set.

The data collected in the MNSIT data is varied by having the same signs made by different subjects in different backgrounds. Moreover, the images were also cropped in various ways; however, the desired hand region was left unchanged.

After exploring the dataset, as shown in figure “Distribution of labels in the training set” , the data is observed to be evenly balanced across the values. The wide variety of training sample values are evenly distributed, as can be seen from the visualization.

Implementation of Sign Language Recognition

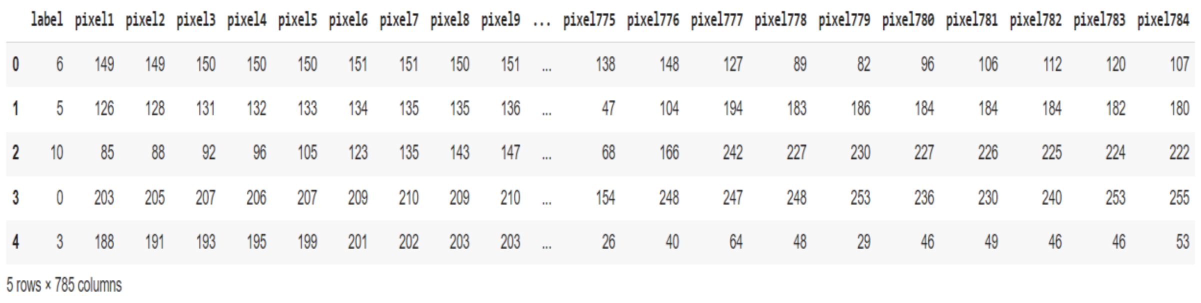
Important libraries will be uploaded to read the dataset, preprocessing and visualization.

```
import numpy as np # linear algebra
import pandas as pd # data processing, CSV file I/O (e.g. pd.read_csv)
from matplotlib import pyplot as plt
import seaborn as sns

import tensorflow as tf
from tensorflow import keras
from tensorflow.keras import layers
from tensorflow.keras.layers.experimental import preprocessing
from keras.preprocessing.image import ImageDataGenerator
from tensorflow.keras.utils import to_categorical
import tensorflow.keras.layers as tfl
from sklearn.model_selection import train_test_split
from sklearn.metrics import confusion_matrix
from sklearn.metrics import classification_report
```

We will read the training and test CSV files

```
training_data = pd.read_csv('/content/drive/MyDrive/SEM_7_MINOR_PROJECT/archive/sign_mnist_train.csv')
test_data = pd.read_csv('/content/drive/MyDrive/SEM_7_MINOR_PROJECT/archive/sign_mnist_test.csv')
```

Verfying the dataset



	label	pixel1	pixel2	pixel3	pixel4	pixel5	pixel6	pixel7	pixel8	pixel9	...	pixel775	pixel776	pixel777	pixel778	pixel779	pixel780	pixel781	pixel782	pixel783	pixel784
0	6	149	149	150	150	150	151	151	150	151	...	138	148	127	89	82	96	106	112	120	107
1	5	126	128	131	132	133	134	135	135	136	...	47	104	194	183	186	184	184	184	182	180
2	10	85	88	92	96	105	123	135	143	147	...	68	166	242	227	230	227	226	225	224	222
3	0	203	205	207	206	207	209	210	209	210	...	154	248	247	248	253	236	230	240	253	255
4	3	188	191	193	195	199	201	202	203	203	...	26	40	64	48	29	46	49	46	46	53

5 rows × 785 columns

We will check the shape of the training and test data that we have read above.

```
print(training_data.shape)
print(test_data.shape)
```

```
(27455, 785)
(7172, 785)
```

Dividing the dataset into x and y training values
X -> info. About image
Y -> label of the image

```
Y_train = training_data["label"]

X_train = training_data.drop(labels = ["label"],axis = 1)
```

Dividing the dataset into x and y test values
X -> info. About image
Y -> label of the image

```
Y_test = test_data["label"]

X_test = test_data.drop(labels = ["label"],axis = 1)
```

Scaling the **images**

We perform a grayscale normalization to reduce the effect of illumination's differences.moreover the cnn converges faster on [0..1] data than on [0..255].

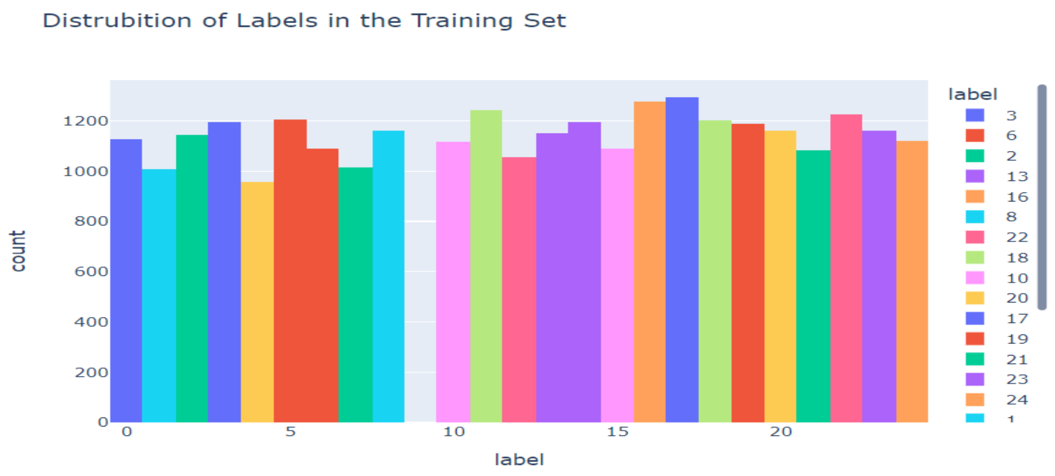
```
# Pixel values are between 0 - 255. Here we convert them to values between 0 - 1
X_train = X_train / 255.0

X_test = X_test / 255.0
```

```
# Reshaping the images to 28 X 28

X_train = X_train.values.reshape(-1,28,28,1)
X_test = X_test.values.reshape(-1,28,28,1)
print(X_train.shape)
print(X_test.shape)
```

```
(27455, 28, 28, 1)
(7172, 28, 28, 1)
```

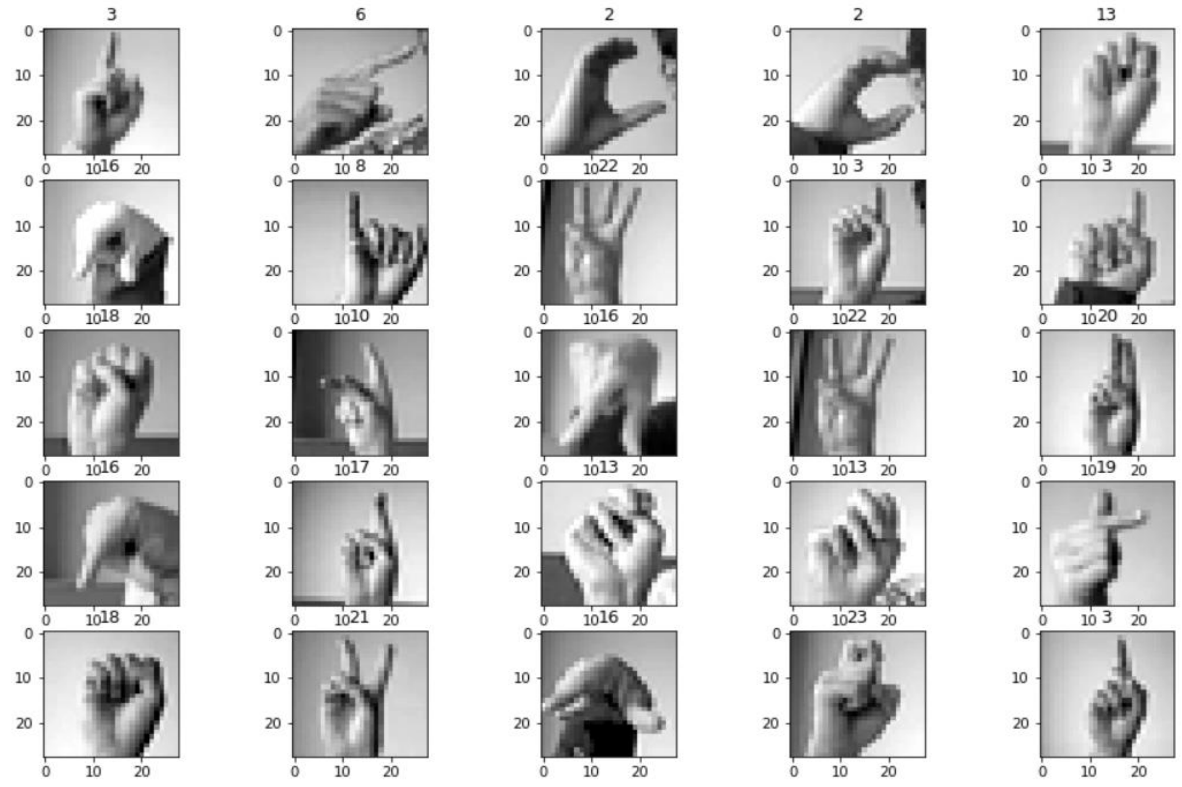
Plotting the Distribution of Labels

```
import matplotlib.pyplot as plt
import seaborn as sns
import plotly.express as px
from PIL import Image

fig = px.histogram(training_data,
                   x='label',
                   color = 'label',
                   title="Distrubition of Labels in the Training Set",
                   width=700, height=500)
fig.show()
```



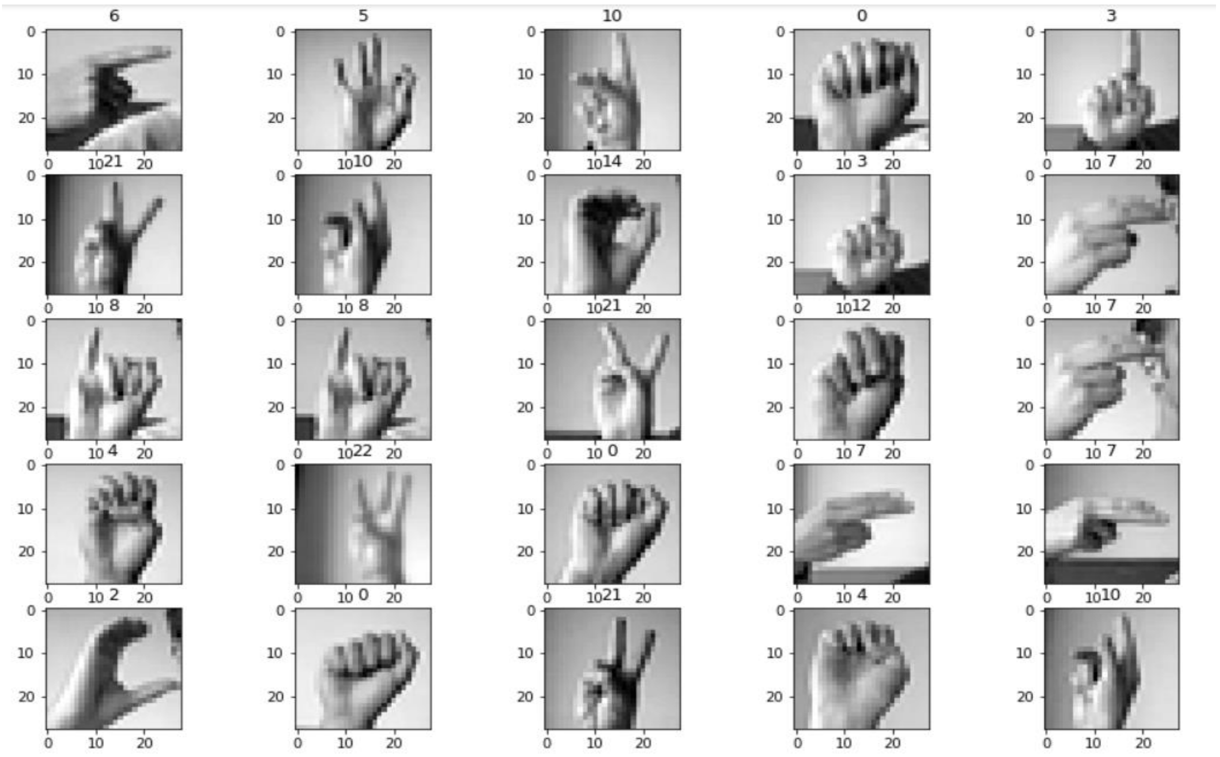
Viewing the **images**
Training images

```
#creating a 5x5 grid of the first 25 photos in the training images
fig, axes = plt.subplots(nrows=5, ncols=5, figsize=(15, 10),
                         subplot_kw={'xticks': [], 'yticks': []})

for i in range(25):
    plt.subplot(5,5,i+1)
    plt.imshow(np.squeeze(X_train[i]), cmap='gray')
    plt.title(Y_train[i])
plt.show()
```



Test images

```
#creating a 5x5 grid of the first 25 photos in the test images
fig, axes = plt.subplots(nrows=5, ncols=5, figsize=(15, 10),
                        subplot_kw={'xticks': [], 'yticks': []})

for i in range(25):
    plt.subplot(5,5,i+1)
    plt.imshow(np.squeeze(X_test[i]), cmap='gray')
    plt.title(Y_test[i])
plt.show()
```



Data Augmentation

In order to avoid overfitting problem, we need to expand artificially our dataset. We can make your existing dataset even larger. The idea is to alter the training data with small transformations to reproduce the variations.

Approaches that alter the training data in ways that change the array representation while keeping the label the same are known as data augmentation techniques. Some popular augmentations people use are grayscales, horizontal flips, vertical flips, random crops, color jitters, translations, rotations, and much more.

By applying just a couple of these transformations to our training data, we can easily double or triple the number of training examples and create a very robust model.

```
datagen = ImageDataGenerator(
        featurewise_center=False,  # set input mean to 0 over the dataset
        samplewise_center=False,  # set each sample mean to 0
        featurewise_std_normalization=False,  # divide inputs by std of the dataset
        samplewise_std_normalization=False,  # divide each input by its std
        zca_whitening=False,  # apply ZCA whitening
        rotation_range=10,  # randomly rotate images in the range (degrees, 0 to 180)
        zoom_range = 0.1, # Randomly zoom image
        width_shift_range=0.1,  # randomly shift images horizontally (fraction of total width)
        height_shift_range=0.1,  # randomly shift images vertically (fraction of total height)
        horizontal_flip=False,  # randomly flip images
        vertical_flip=False)  # randomly flip images


datagen.fit(X_train)
```

Splitting the training data

```
#spliting training images into the images we will use for training the model and validating the model
X_train, X_val, Y_train, Y_val = train_test_split(X_train, Y_train, test_size = 0.3, random_state=7)
```

```
#showing the shapes of our train, validate, and test images
print(X_train.shape)
print(Y_train.shape)
print(X_val.shape)
print(Y_val.shape)
print(X_test.shape)
print(Y_test.shape)
```

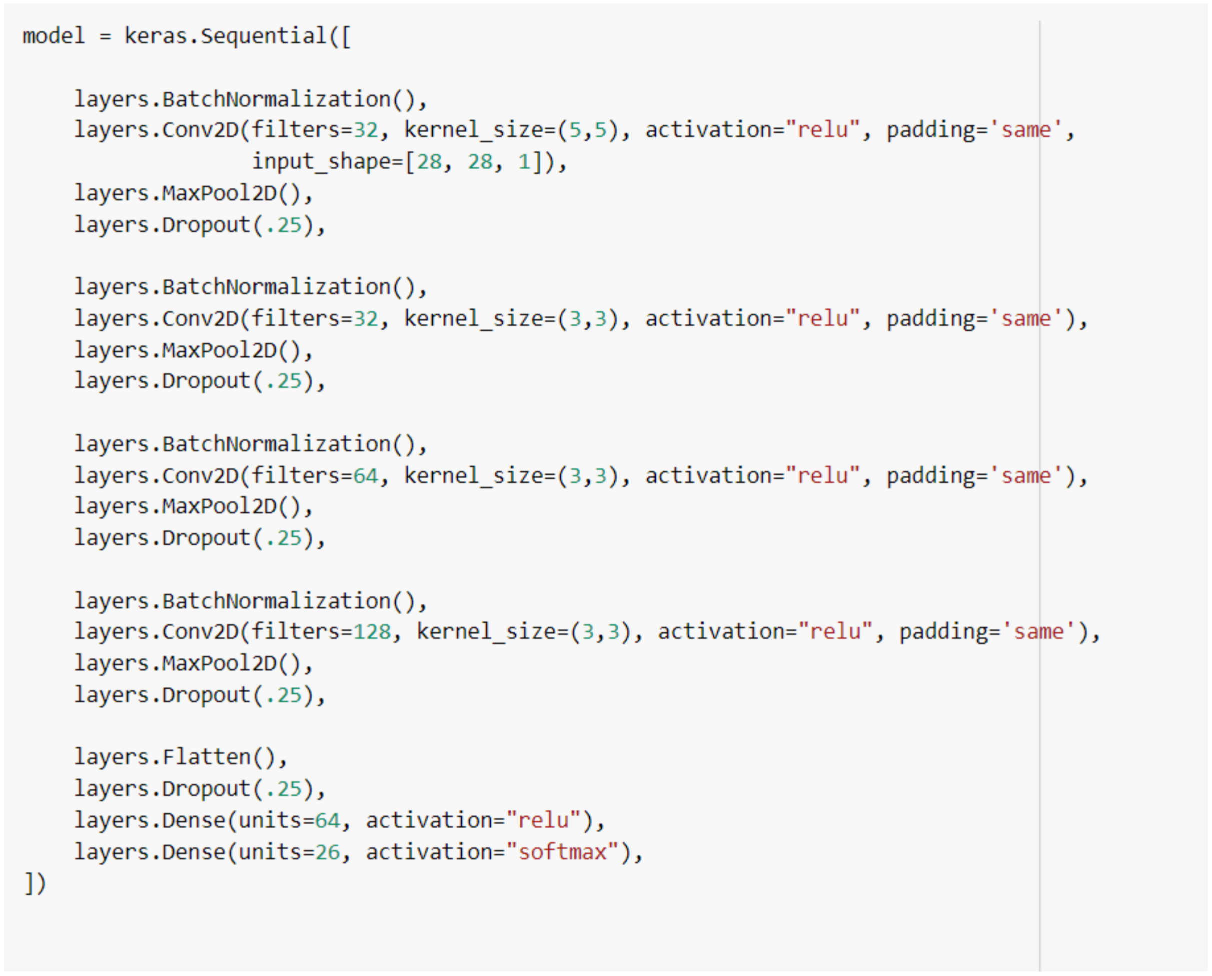
```
(13452, 28, 28, 1)
(13452,)
(5766, 28, 28, 1)
(5766,)
(7172, 28, 28, 1)
(7172,)
```

CNN MODEL CREATION

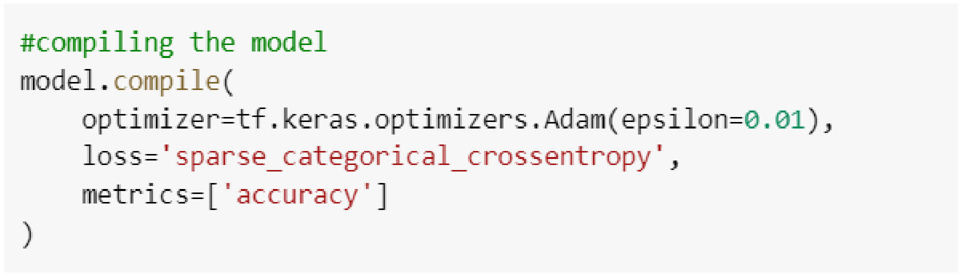
A convolutional neural network is a special type of an artificial intelligence implementation which uses a special mathematical matrix manipulation called the convolution operation to process data from the images.

A convolution does this by multiplying two matrices and yielding a third, smaller matrix.
The network takes an input image, and uses a filter (or kernel) to create a feature map describing the image.

In the convolution operation, we take a filter (usually 2x2 or 3x3 matrix) and slide it over the image matrix. The coresponding numbers in both matrices are multiplied and and added to yield a single number describing that input space.



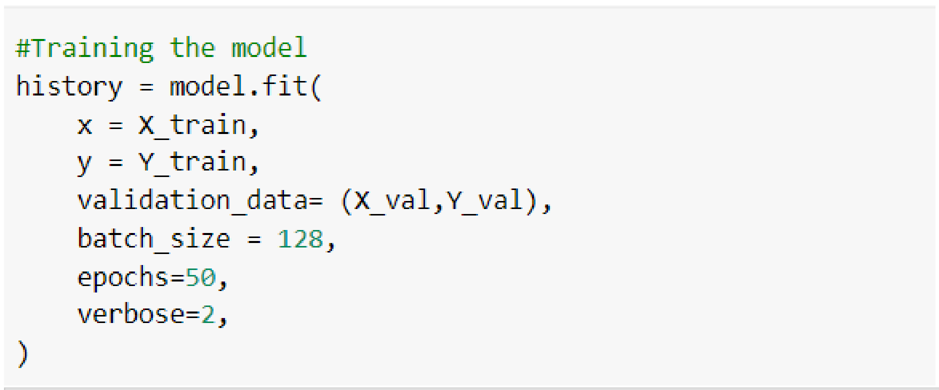
```
model = keras.Sequential([

    layers.BatchNormalization(),
    layers.Conv2D(filters=32, kernel_size=(5,5), activation="relu", padding='same',
                  input_shape=[28, 28, 1]),
    layers.MaxPool2D(),
    layers.Dropout(.25),

    layers.BatchNormalization(),
    layers.Conv2D(filters=32, kernel_size=(3,3), activation="relu", padding='same'),
    layers.MaxPool2D(),
    layers.Dropout(.25),

    layers.BatchNormalization(),
    layers.Conv2D(filters=64, kernel_size=(3,3), activation="relu", padding='same'),
    layers.MaxPool2D(),
    layers.Dropout(.25),

    layers.BatchNormalization(),
    layers.Conv2D(filters=128, kernel_size=(3,3), activation="relu", padding='same'),
    layers.MaxPool2D(),
    layers.Dropout(.25),

    layers.Flatten(),
    layers.Dropout(.25),
    layers.Dense(units=64, activation="relu"),
    layers.Dense(units=26, activation="softmax"),
])
```



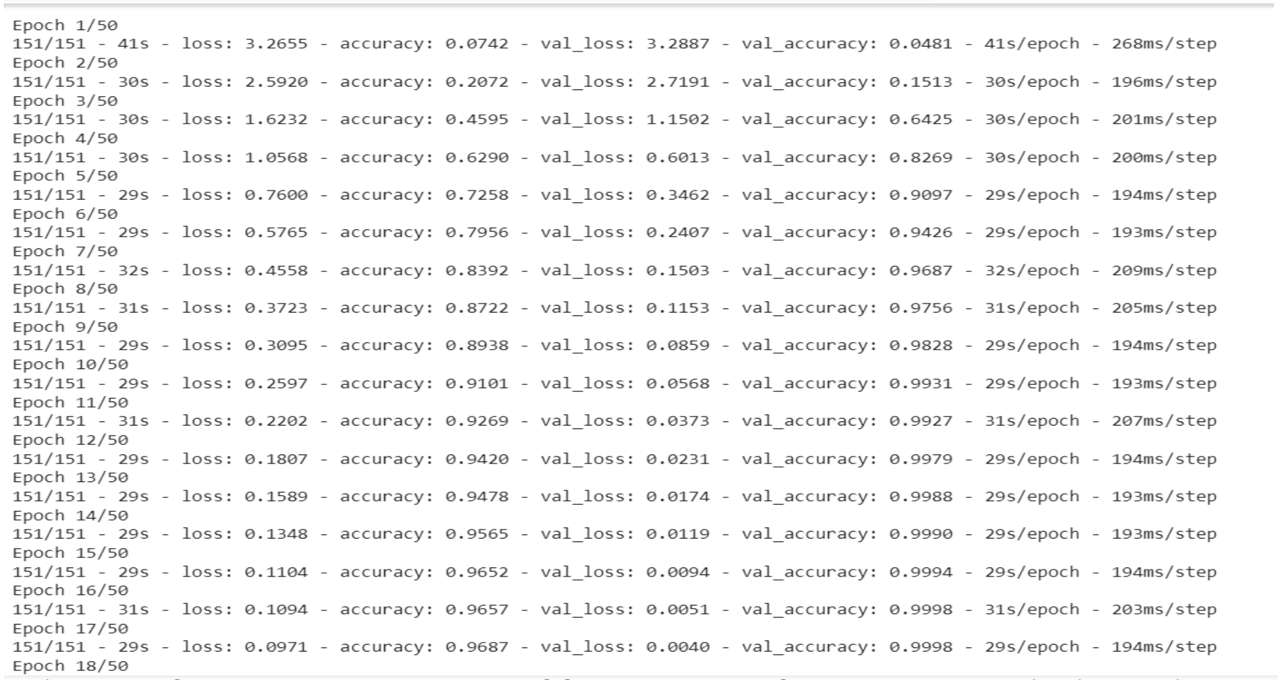
```
#compiling the model
model.compile(
    optimizer=tf.keras.optimizers.Adam(epsilon=0.01),
    loss='sparse_categorical_crossentropy',
    metrics=['accuracy']
)
```

Training the model

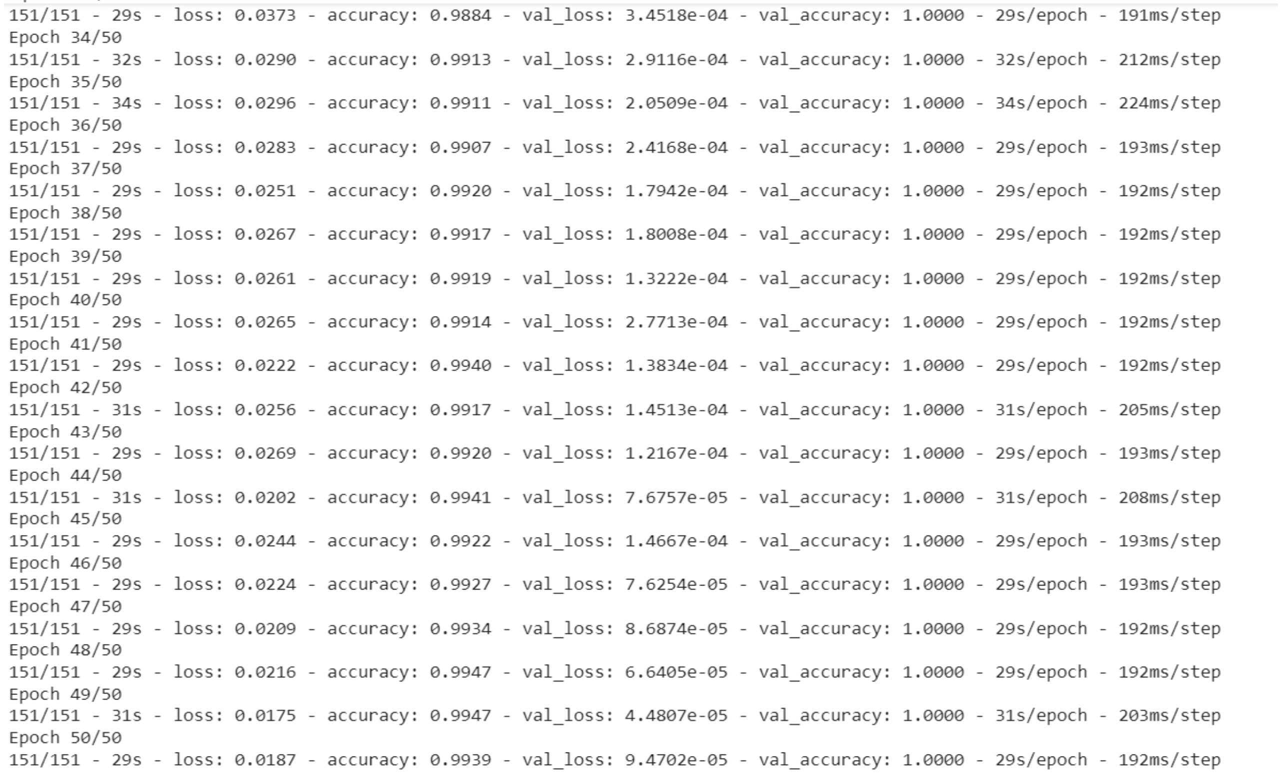
First train on only the actual dataset
Then after our model has learned, we will train the model on the dataset with our image transformations. We do this so to speed up trainning and so the model becomes more rebust to real world data.



```
#Training the model
history = model.fit(
    x = X_train,
    y = Y_train,
    validation_data= (X_val,Y_val),
    batch_size = 128,
    epochs=50,
    verbose=2,
)
```



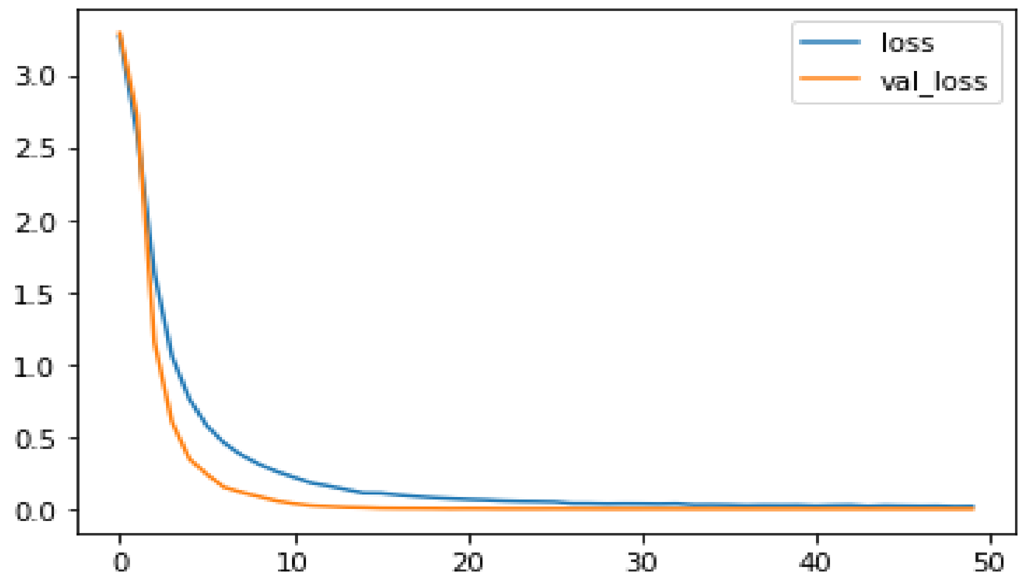
```
Epoch 1/50
151/151 - 41s - loss: 3.2655 - accuracy: 0.0742 - val_loss: 3.2887 - val_accuracy: 0.0481 - 41s/epoch - 268ms/step
Epoch 2/50
151/151 - 30s - loss: 2.5920 - accuracy: 0.2072 - val_loss: 2.7191 - val_accuracy: 0.1513 - 30s/epoch - 196ms/step
Epoch 3/50
151/151 - 30s - loss: 1.6232 - accuracy: 0.4595 - val_loss: 1.1502 - val_accuracy: 0.6425 - 30s/epoch - 201ms/step
Epoch 4/50
151/151 - 30s - loss: 1.0568 - accuracy: 0.6290 - val_loss: 0.6013 - val_accuracy: 0.8269 - 30s/epoch - 200ms/step
Epoch 5/50
151/151 - 29s - loss: 0.7600 - accuracy: 0.7258 - val_loss: 0.3462 - val_accuracy: 0.9097 - 29s/epoch - 194ms/step
Epoch 6/50
151/151 - 29s - loss: 0.5765 - accuracy: 0.7956 - val_loss: 0.2407 - val_accuracy: 0.9426 - 29s/epoch - 193ms/step
Epoch 7/50
151/151 - 32s - loss: 0.4558 - accuracy: 0.8392 - val_loss: 0.1503 - val_accuracy: 0.9687 - 32s/epoch - 209ms/step
Epoch 8/50
151/151 - 31s - loss: 0.3723 - accuracy: 0.8722 - val_loss: 0.1153 - val_accuracy: 0.9756 - 31s/epoch - 205ms/step
Epoch 9/50
151/151 - 29s - loss: 0.3095 - accuracy: 0.8938 - val_loss: 0.0859 - val_accuracy: 0.9828 - 29s/epoch - 194ms/step
Epoch 10/50
151/151 - 29s - loss: 0.2597 - accuracy: 0.9101 - val_loss: 0.0568 - val_accuracy: 0.9931 - 29s/epoch - 193ms/step
Epoch 11/50
151/151 - 31s - loss: 0.2202 - accuracy: 0.9269 - val_loss: 0.0373 - val_accuracy: 0.9927 - 31s/epoch - 207ms/step
Epoch 12/50
151/151 - 29s - loss: 0.1807 - accuracy: 0.9420 - val_loss: 0.0231 - val_accuracy: 0.9979 - 29s/epoch - 194ms/step
Epoch 13/50
151/151 - 29s - loss: 0.1589 - accuracy: 0.9478 - val_loss: 0.0174 - val_accuracy: 0.9988 - 29s/epoch - 193ms/step
Epoch 14/50
151/151 - 29s - loss: 0.1348 - accuracy: 0.9565 - val_loss: 0.0119 - val_accuracy: 0.9990 - 29s/epoch - 193ms/step
Epoch 15/50
151/151 - 29s - loss: 0.1104 - accuracy: 0.9652 - val_loss: 0.0094 - val_accuracy: 0.9994 - 29s/epoch - 194ms/step
Epoch 16/50
151/151 - 31s - loss: 0.1094 - accuracy: 0.9657 - val_loss: 0.0051 - val_accuracy: 0.9998 - 31s/epoch - 203ms/step
Epoch 17/50
151/151 - 29s - loss: 0.0971 - accuracy: 0.9687 - val_loss: 0.0040 - val_accuracy: 0.9998 - 29s/epoch - 194ms/step
Epoch 18/50
```

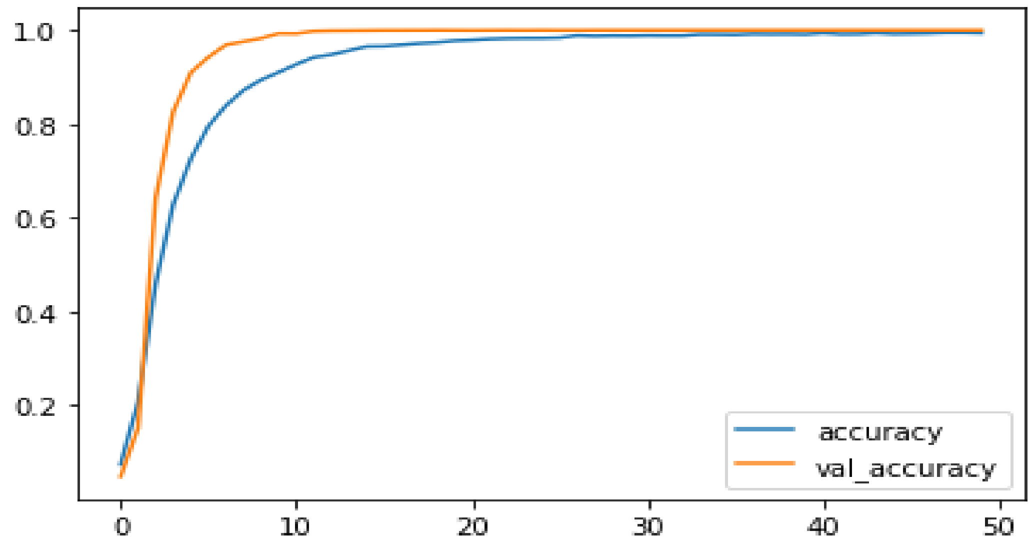


```
151/151 - 29s - loss: 0.0373 - accuracy: 0.9884 - val_loss: 3.4518e-04 - val_accuracy: 1.0000 - 29s/epoch - 191ms/step
Epoch 34/50
151/151 - 32s - loss: 0.0290 - accuracy: 0.9913 - val_loss: 2.9116e-04 - val_accuracy: 1.0000 - 32s/epoch - 212ms/step
Epoch 35/50
151/151 - 34s - loss: 0.0296 - accuracy: 0.9911 - val_loss: 2.0509e-04 - val_accuracy: 1.0000 - 34s/epoch - 224ms/step
Epoch 36/50
151/151 - 29s - loss: 0.0283 - accuracy: 0.9907 - val_loss: 2.4168e-04 - val_accuracy: 1.0000 - 29s/epoch - 193ms/step
Epoch 37/50
151/151 - 29s - loss: 0.0251 - accuracy: 0.9920 - val_loss: 1.7942e-04 - val_accuracy: 1.0000 - 29s/epoch - 192ms/step
Epoch 38/50
151/151 - 29s - loss: 0.0267 - accuracy: 0.9917 - val_loss: 1.8008e-04 - val_accuracy: 1.0000 - 29s/epoch - 192ms/step
Epoch 39/50
151/151 - 29s - loss: 0.0261 - accuracy: 0.9919 - val_loss: 1.3222e-04 - val_accuracy: 1.0000 - 29s/epoch - 192ms/step
Epoch 40/50
151/151 - 29s - loss: 0.0265 - accuracy: 0.9914 - val_loss: 2.7713e-04 - val_accuracy: 1.0000 - 29s/epoch - 192ms/step
Epoch 41/50
151/151 - 29s - loss: 0.0222 - accuracy: 0.9940 - val_loss: 1.3834e-04 - val_accuracy: 1.0000 - 29s/epoch - 192ms/step
Epoch 42/50
151/151 - 31s - loss: 0.0256 - accuracy: 0.9917 - val_loss: 1.4513e-04 - val_accuracy: 1.0000 - 31s/epoch - 205ms/step
Epoch 43/50
151/151 - 29s - loss: 0.0269 - accuracy: 0.9920 - val_loss: 1.2167e-04 - val_accuracy: 1.0000 - 29s/epoch - 193ms/step
Epoch 44/50
151/151 - 31s - loss: 0.0202 - accuracy: 0.9941 - val_loss: 7.6757e-05 - val_accuracy: 1.0000 - 31s/epoch - 208ms/step
Epoch 45/50
151/151 - 29s - loss: 0.0244 - accuracy: 0.9922 - val_loss: 1.4667e-04 - val_accuracy: 1.0000 - 29s/epoch - 193ms/step
Epoch 46/50
151/151 - 29s - loss: 0.0224 - accuracy: 0.9927 - val_loss: 7.6254e-05 - val_accuracy: 1.0000 - 29s/epoch - 193ms/step
Epoch 47/50
151/151 - 29s - loss: 0.0209 - accuracy: 0.9934 - val_loss: 8.6874e-05 - val_accuracy: 1.0000 - 29s/epoch - 192ms/step
Epoch 48/50
151/151 - 29s - loss: 0.0216 - accuracy: 0.9947 - val_loss: 6.6405e-05 - val_accuracy: 1.0000 - 29s/epoch - 192ms/step
Epoch 49/50
151/151 - 31s - loss: 0.0175 - accuracy: 0.9947 - val_loss: 4.4807e-05 - val_accuracy: 1.0000 - 31s/epoch - 203ms/step
Epoch 50/50
151/151 - 29s - loss: 0.0187 - accuracy: 0.9939 - val_loss: 9.4702e-05 - val_accuracy: 1.0000 - 29s/epoch - 192ms/step
```

After successful training, we will visualize the training performance of the CNN model.

```
#Viewing the training results
history_frame = pd.DataFrame(history.history)
history_frame.loc[:, ['loss', 'val_loss']].plot()
history_frame.loc[:, ['accuracy', 'val_accuracy']].plot();
```





CLASSIFICATION REPORT

```
#creating our predictions using the test pixel values
predictions = model.predict(X_test)
predictions = np.argmax(predictions,axis = 1)

#creating a report that show how our predictions compare with actual values
print(classification_report(Y_test, predictions))
```

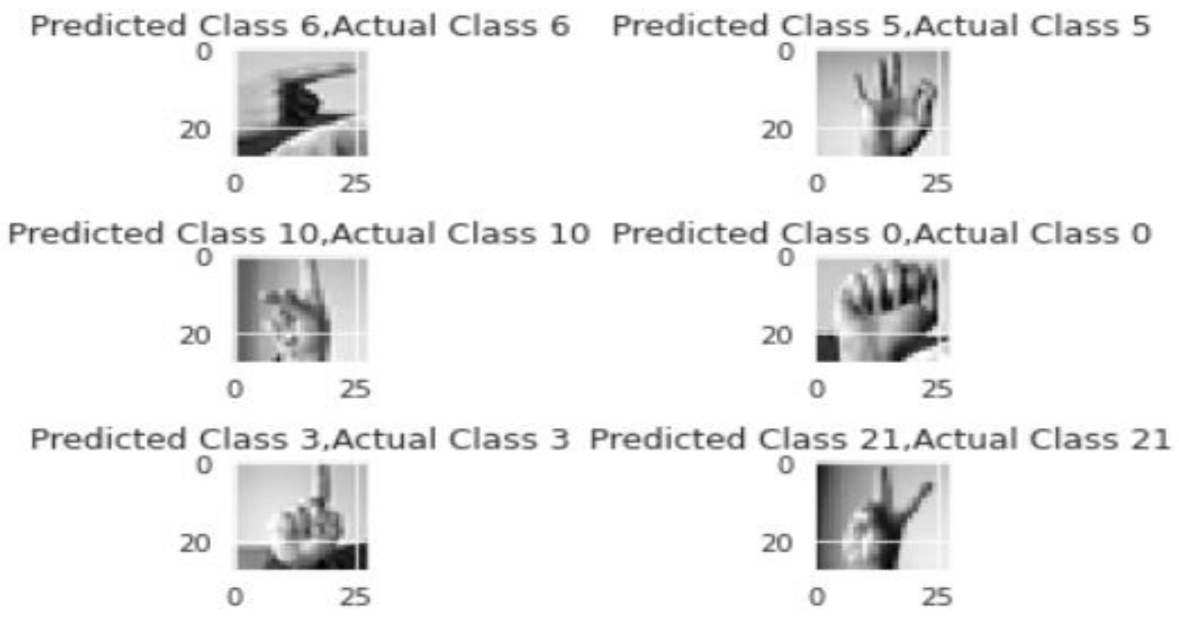
```
225/225 [==============================] - 3s 13ms/step
              precision    recall  f1-score   support

           0       1.00      1.00      1.00       331
           1       1.00      1.00      1.00       432
           2       1.00      1.00      1.00       310
           3       1.00      1.00      1.00       245
           4       0.98      1.00      0.99       498
           5       1.00      1.00      1.00       247
           6       0.99      0.99      0.99       348
           7       0.99      1.00      0.99       436
           8       0.99      1.00      0.99       288
          10       1.00      1.00      1.00       331
          11       1.00      1.00      1.00       209
          12       1.00      1.00      1.00       394
          13       1.00      1.00      1.00       291
          14       1.00      1.00      1.00       246
          15       1.00      1.00      1.00       347
          16       1.00      1.00      1.00       164
          17       1.00      0.85      0.92       144
          18       1.00      0.96      0.98       246
          19       1.00      1.00      1.00       248
          20       1.00      1.00      1.00       266
          21       0.94      1.00      0.97       346
          22       1.00      1.00      1.00       206
          23       1.00      1.00      1.00       267
          24       1.00      0.98      0.99       332

    accuracy                           0.99      7172
   macro avg       1.00      0.99      0.99      7172
weighted avg       0.99      0.99      0.99      7172
```

```
correct = np.nonzero(predictions == y)[0]
```

```
/opt/conda/lib/python3.6/site-packages/numpy/core/fromnumeric.py:61: FutureWarning: Seri
es.nonzero() is deprecated and will be removed in a future version.Use Series.to_numpy
().nonzero() instead
  return bound(*args, **kwds)
```

Some of the Correctly Predicted Classes

```
i = 0
for c in correct[:6]:
    plt.subplot(3,2,i+1)
    plt.imshow(x_test[c].reshape(28,28), cmap="gray", interpolation='none')
    plt.title("Predicted Class {},Actual Class {}".format(predictions[c], y[c]))
    plt.tight_layout()
    i += 1
```



As we can see in the above visualization, the CNN model has predicted the correct class labels for almost all the images.

Here, we can conclude that the Convolutional Neural Network has given an outstanding performance in the classification of sign language symbol images. The average accuracy score of the model is more than 94% and it can further be improved by tuning the hyperparameters.

We have trained our model in 50 epochs and the accuracy may be improved if we have more epochs of training. The accuracy increased to 99%.

Conclusion and Future Work:

The proposed CNN model has the ability to achieve an accuracy of 99.63% on the proposed test dataset. Initially, when only one convolutional layer is utilized and there was overfitting in the dataset. To overcome this issue of overfitting, two more layers are added so that some parameters are dropped and still preserve the essential features of images. Moreover, the dropout layer, Batch Normalization, and data augmentation are also used to overcome the same and achieve a better performance of our network.

The letters like ‘J’ and ‘Z’ were not included in the classification due to the hand movement. So, the video frames are required to classify them. The study may be carried out in the proposed work to accept the video frames and classify the letters like ‘J’ and ‘Z’ as well. Moreover, a large dataset of Sign Language is required as the available dataset is quite constrained and most of the researchers are carried out on the same by making it more difficult to compare the work of different researchers.

References

1. T. Yang, Y. Xu, and "A. , Hidden Markov Model for Gesture Recognition", CMU-RI-TR-94 10, Robotics Institute, Carnegie Mellon Univ.,Pittsburgh,PA, May 1994.
2. Pujan Ziaie, Thomas M ¨uller , Mary Ellen Foster , and Alois Knoll"A Na ¨ıve Bayes Munich,Dept. of Informatics VI, Robotics and Embedde Systems, Boltzmannstr. 3, DE-85748 Garching, Germany.
3. https://docs.opencv.org/2.4/doc/tutorials/imgproc/gausian_median_blur_bilateral_filter/gausian_median_blur_bilateral_filter.html
4. Mohammed Waleed Kalous, Machine recognition of Auslan signs using PowerGloves: Towards large-lexicon recognition of sign language.
5. http://www-i6.informatik.rwth-aachen.de/~dreuw/database.php
6. Pigou L., Dieleman S., Kindermans PJ., Schrauwen B. (2015) Sign Language Recognition Using Convolutional Neural Networks. In: Agapito L., Bronstein M., Rother C. (eds) Computer Vision - ECCV 2014 Workshops. ECCV 2014. Lecture Notes in Computer Science, vol 8925. Springer, Cham
7. Zaki, M.M., Shaheen, S.I.: Sign language recognition using a combination of new vision based features. Pattern Recognition Letters 32(4), 572–577 (2011)
8. N. Mukai, N. Harada and Y. Chang, "Japanese Fingerspelling Recognition Based on Classification Tree and Machine Learning," *2017 Nicograph International (NicoInt)*,Kyoto, Japan, 2017, pp. 19-24. doi:10.1109/NICOInt.2017.9
9. Byeongkeun Kang , Subarna Tripathi , Truong Q. Nguyen "Real-time sign language fingerspelling recognition using convolutional neural networks from depth map" 2015 3rd IAPR Asian Conference on Pattern Recognition (ACPR)
10. https://opencv.org/