SIGNIFY

12-08-2021 | DMPA MINI PROJECT | CCE A | BATCH 1

Anushree Bhat	190953043
Sanjana Chitturi	190953033
B. Yamini	190953086

Abstract:

Sign languages are developed primarily for the hearing and speech impaired. They use a concurrent and specific combination of hand movements, hand shapes, and orientation to convey information. One such set of language is the Indian Sign Language (ISL) system

The hearing-impaired people become neglected by society because the normal people never try to learn ISL nor try to interact with the hearing-impaired people. This becomes a curse for them and so they mostly remain uneducated and isolated. Thus, recognition of sign language was introduced which has not only been important from an engineering point of view but also for the impact on society.

Introduction:

This project aims to bridge the gap between us and the hearing-impaired people by introducing an inexpensive Sign Language Recognition technique which will allow the user to understand the meaning of the sign without the help of an expert translator. Computers are used in communication which helps in capturing the signs, processing them, and finally recognizing the sign. The certain aspect that distinguishes ISL from other sign languages is that ISL is devoid of any temporal inflections in its finger spelling chart and the usage of both hands.

A large number of previous works have been done for the same using videos, we aim to use static images as data as static images have better clarity and are better to train the model.

Literature Survey-

Below are the links linking to the research papers that we have referred to in this project.

Research Papers-
https://www.researchgate.net/publication/237054175_Recognition_of_Indian_Sign_Language_in_Live_Video

The Indian Sign Language recognition approach was implemented using MATLAB and the process uses classification

https://ieeexplore.ieee.org/document/7916786

Dataset-

We used this Kaggle dataset which was made through different datasets found across Kaggle and GitHub. It has only alphanumeric datasets.

https://www.kaggle.com/prathumarikeri/indian-sign-language-isl

Concepts-

In this project, we are making use of this research paper as it uses static images, ("https://ieeexplore.ieee.org/document/8537248"). We have used the same preprocessing techniques and used a CNN model to train and test the data instead of the Inception V3 model.

The major concepts used in the project include-

TensorFlow/Keras -

https://www.tensorflow.org/api_docs/python/tf

https://keras.io/

TensorFlow is an end-to-end open-source platform for machine learning. It's a the comprehensive and flexible ecosystem of tools, libraries, and other resources that provide workflows with high-level APIs.

Keras is a high-level neural networks library that is running on the top of TensorFlow. Using Keras in deep learning allows for easy and fast prototyping as well as running seamlessly on CPU and GPU.

Keras is being used for preprocessing and the CNN model in our project.

Open CV-

https://learnopencv.com/getting-started-with-opencv/

OpenCV is the huge open-source library for computer vision, machine learning, and image processing for real-time operations. Processing images and videos to identify objects, faces, or even the handwriting of a human can be done using OpenCV.

We use OpenCV for image processing and feature extraction. (The latest interface version of Open CV is written as cv2, hence while importing the module name is used as cv2)

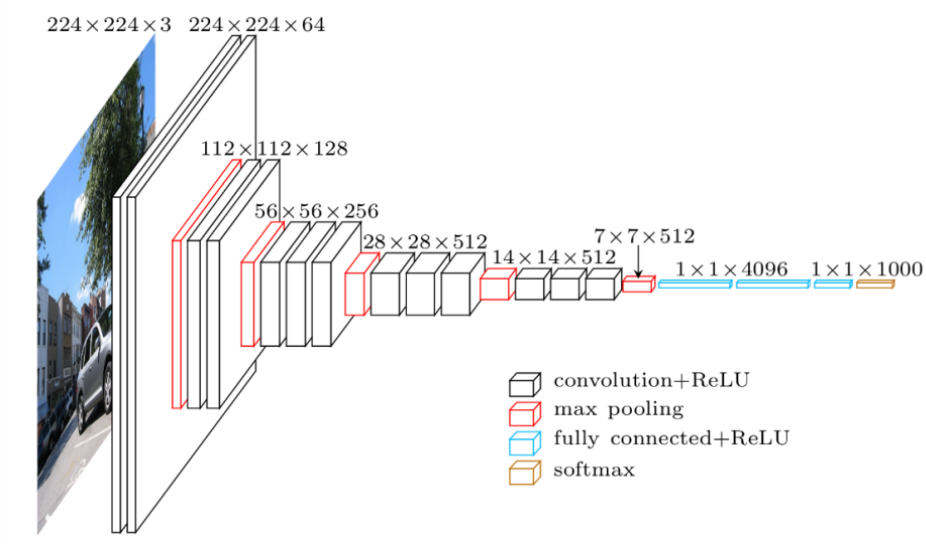
CNN-

https://towardsdatascience.com/covolutional-neural-network-cb0883dd6529

Convolutional neural networks power image recognition and computer vision tasks. Computer vision is a field of artificial intelligence (AI) that enables computers and systems to derive meaningful information from digital images, videos, and other visual inputs, and based on those inputs, it can take action. This ability to provide recommendations distinguishes it from image recognition tasks.

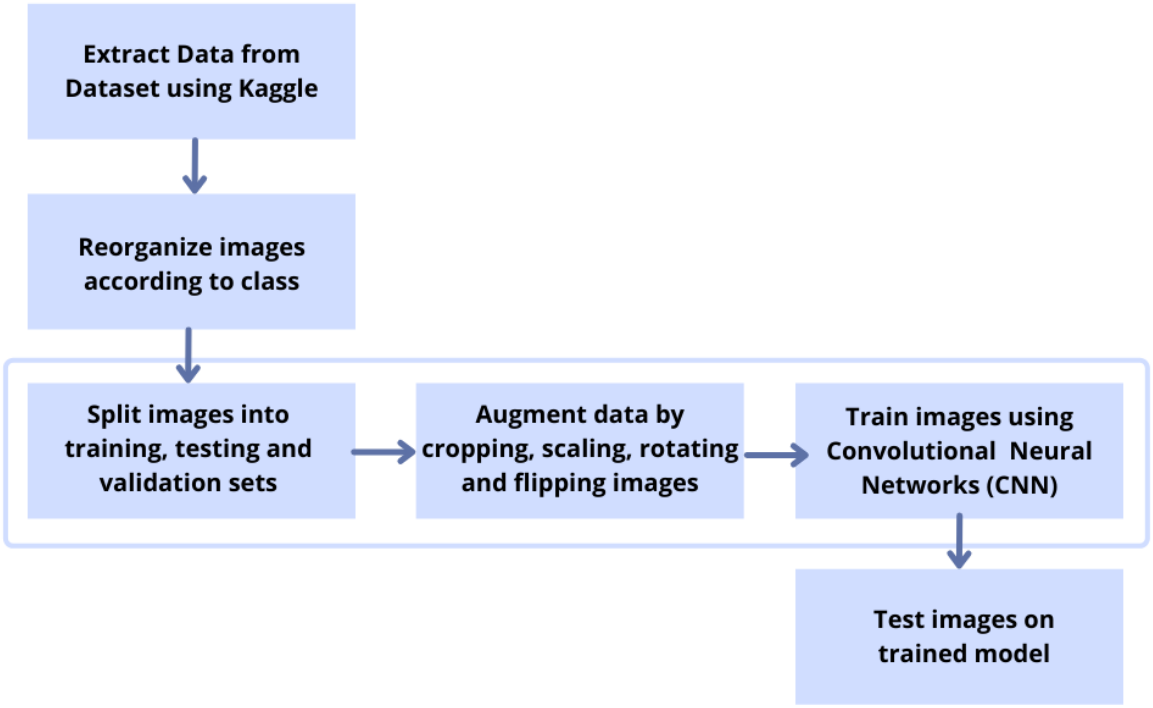
There are five different layers in CNN which are used here for the training and prediction of data-

- Input layer
- Convo layer (Convo + ReLU)
- Pooling layer
- Fully connected (FC) layer
- Softmax/logistic layer
- Output layer



Methodology/Implementation-

Flow Diagram-



Step 1- Preprocessing-

Import all the necessary packages.: -

Common packages

- Pandas
- NumPy
- Seaborn
- Matplotlib
- random

Path processes

- Path
- Glob

Image processing

- PIL
- Skimage

Scaler and transformation

- Sklearn

Optimizer

- from Keras. optimizers

Model Layers

- Sequential model from Keras from TensorFlow

A few packages have been shown below-

```
#GENERAL
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
import random
#PATH PROCESS
import os
import os.path
from pathlib import Path
import glob
#IMAGE PROCESS
from PIL import Image
from keras.preprocessing import image
from tensorflow.keras.preprocessing.image import ImageDataGenerator
import cv2
from keras.applications.vgg16 import preprocess_input, decode_predictions
from keras.preprocessing import image
```

Import the dataset through Kaggle using the 'path' function. Find the different jpg paths using a list. ".glob" finds the different paths with respect to a particular format (here, .jpg files)

Then, find out the class labels by creating a list with only the class each picture belongs to. (Class label e.g.- A, B, C…. Z & 1,2,3…9)

Series makes an array; two series are created. One for all the path names of the jpg files and another array for all the class labels.

MAIN PATH

```
[2]: Indian_Sign_Main_Path = Path("../input/indian-sign-language-isl/Indian")
```

JPG PATH

```
[3]: Sign_JPG = list(Indian_Sign_Main_Path.glob(r"*/*.jpg"))
```

LABELS

```
[4]: Sign_Labels = list(map(lambda x: os.path.split(os.path.split(x)[0])[1],Sign_JPG))
```

TO SERIES

```
[5]: Sign_JPG_Series = pd.Series(Sign_JPG,name="JPG").astype(str)
     Sign_Labels_Series = pd.Series(Sign_Labels,name="CATEGORY")
```

```
[6]: temp = Sign_Labels_Series.unique()
```

The call pd.concat makes an object of two Series to create a DataFrame (numPy array) with the JPG file path and JPG class labels as columns.

```
[10]: Main_Sign_Data = pd.concat([Sign_JPG_Series,Sign_Labels_Series],axis=1)
```

```
[11]: print(Main_Sign_Data.head(-1))
```

```
                                                  JPG CATEGORY
0      ../input/indian-sign-language-isl/Indian/N/623...        N
1      ../input/indian-sign-language-isl/Indian/N/764...        N
2      ../input/indian-sign-language-isl/Indian/N/107...        N
3      ../input/indian-sign-language-isl/Indian/N/771...        N
4      ../input/indian-sign-language-isl/Indian/N/208...        N
...                                                  ...      ...
42739  ../input/indian-sign-language-isl/Indian/J/90.jpg        J
42740  ../input/indian-sign-language-isl/Indian/J/599...        J
42741  ../input/indian-sign-language-isl/Indian/J/25.jpg        J
42742  ../input/indian-sign-language-isl/Indian/J/147...        J
42743  ../input/indian-sign-language-isl/Indian/J/921...        J

[42744 rows x 2 columns]
```

TO SHUFFLE

```
[12]: Main_Sign_Data = Main_Sign_Data.sample(frac=1).reset_index(drop=True)
```

Feature extraction -

We tried feature extraction of 3 types to see which produces the best output.

Simple vision uses HSV color space. HSV color space is mostly used for object tracking.

Threshold vision is used to convert the image to greyscale by setting all pixel values above 90 to 255.

Canny vision uses Canny which goes through the following steps

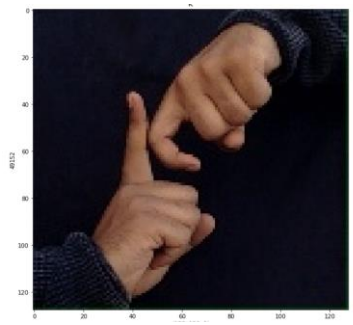
- Noise reduction using Gaussian filter
- Gradient calculation along the horizontal and vertical axis
- Non-Maximum suppression of false edges
- Double thresholding for segregating strong and weak edges
- Edge tracking by hysteresis

And thus, produces only the edges of the image.

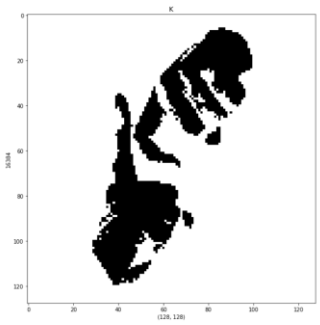
```
[ ]:
def simple_vision(img_path):
    Picking_Img = cv2.cvtColor(cv2.imread(img_path),cv2.COLOR_BGR2RGB)

    return Picking_Img
```

```
[ ]:
def threshold_vision(img_path):
    Picking_Img = simple_vision(img_path)
    Gray_Img = cv2.cvtColor(Picking_Img,cv2.COLOR_RGB2GRAY)
    _,threshold_Img = cv2.threshold(Gray_Img,90,255,cv2.THRESH_BINARY_INV)

    return threshold_Img
```

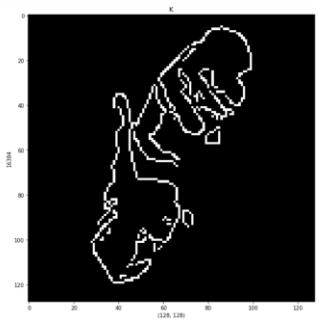
```
[ ]:
def canny_vision(img_path):
    Threshold_Img = threshold_vision(img_path)
    Canny_Img = cv2.Canny(Threshold_Img,10,100)

    return Canny_Img
```



Simple Vision



Threshold Vision



Canny vision

Furthermore, we split the data into two parts using ratios 0.9 and 0.1 for training and testing respectively.

SPLITTING DATA

```
[38]: X_Train,X_Test = train_test_split(Main_Sign_Data,train_size=0.9,random_state=123,shuffle=True)

[39]: print(X_Train.shape)
      print(X_Test.shape)

      (38470, 2)
      (4275, 2)

[40]: print(type(X_Train))
      print(type(X_Test))

      <class 'pandas.core.frame.DataFrame'>
      <class 'pandas.core.frame.DataFrame'>
```

As our dataset is quite small, we are generating more pictures through

- Zoom/Crop
- Scaling
- Shearing transformations
- Rotation
- Shift
- Flips

IMAGE DATA GENERATOR PROCESS

GENERATOR STRUCTURE

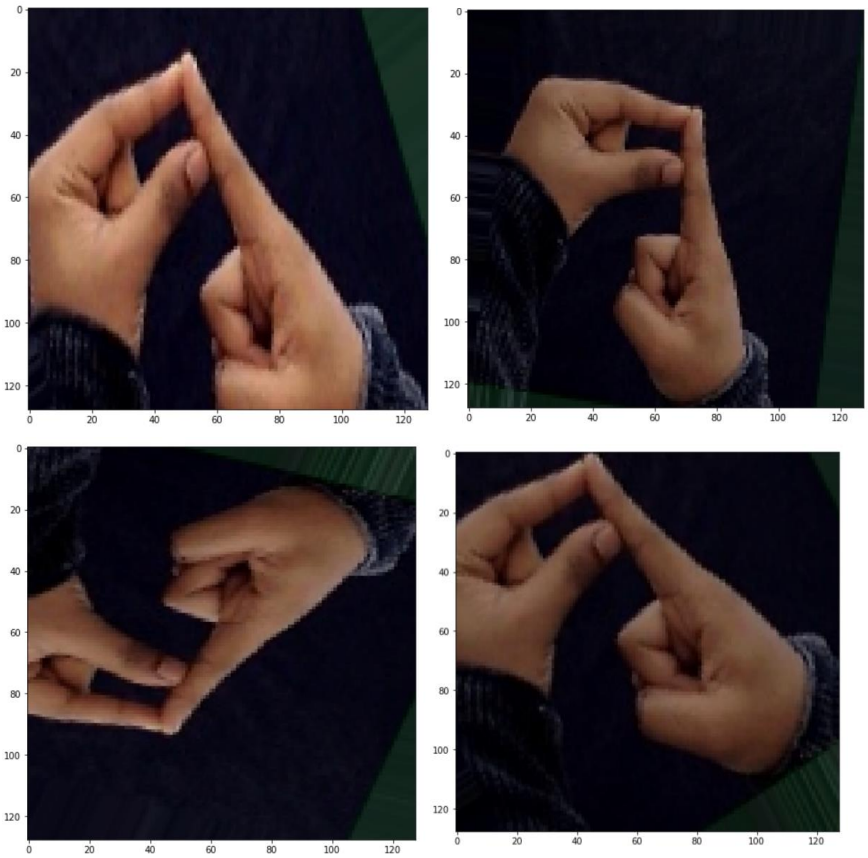
```
[41]: Train_IMG_Generator = ImageDataGenerator(rescale=1./255,
                                               zoom_range=0.5,
                                               shear_range=0.5,
                                               brightness_range=[0.6,1.0],
                                               rotation_range=35,
                                               width_shift_range=0.1,
                                               height_shift_range=0.1,
                                               vertical_flip=True,
                                                featurewise_std_normalization=False,
                                                samplewise_center=False,
                                                samplewise_std_normalization=False,
                                               fill_mode="nearest",
                                               validation_split=0.1)

[42]: Test_IMG_Generator = ImageDataGenerator(rescale=1./255)
```

To show the images we look at the generated images:

HOW TO LOOK BY GENERATOR

[43]:
```
Example_Img = simple_vision(X_Train.JPG[3])
Example_Img = Example_Img.reshape((1,) + Example_Img.shape)

i = 0

for batch in Train_IMG_Generator.flow(Example_Img,batch_size=32):

    figure = plt.figure(figsize=(8,8))
    plt.imshow(image.img_to_array(batch[0]))

    i += 1
    if i % 4 == 0:
        break

plt.show()
```

Generated images after applying various functions on the dataset:



Step 2. Training -

Training 90% of the dataset using the CNN model.

Convolutional Neural Network:

Layers of the CNN model -

- Input layer
- Convo layer (Convo + ReLU)
- Pooling layer
- Fully connected (FC) layer
- Softmax/logistic layer
- Output layer

[50]:
```
Model = Sequential()

Model.add(Conv2D(24,(3,3),activation="relu",input_shape=INPUT_DIM))
Model.add(BatchNormalization())
Model.add(MaxPooling2D((2,2),strides=2))

Model.add(Conv2D(64,(3,3),activation="relu",padding="same"))
Model.add(Dropout(0.3))
Model.add(MaxPooling2D((2,2),strides=2))

Model.add(Conv2D(64,(3,3),activation="relu",padding="same"))
Model.add(Dropout(0.3))
Model.add(MaxPooling2D((2,2),strides=2))

Model.add(Conv2D(128,(3,3),activation="relu",padding="same"))
Model.add(Conv2D(128,(3,3),activation="relu",padding="same"))
Model.add(Dropout(0.3))
Model.add(MaxPooling2D((2,2),strides=2))

Model.add(Conv2D(256,(3,3),activation="relu",padding="same"))
Model.add(Dropout(0.3))
Model.add(MaxPooling2D((2,2),strides=2))

Model.add(Flatten())
Model.add(Dense(2352,activation="relu"))
Model.add(Dropout(0.5))
Model.add(Dense(OUTPUT_DIM,activation="softmax"))
```

[51]:
```
Model.compile(optimizer=COMPILE_OPTIMIZER,loss=COMPILE_LOSS,metrics=COMPILE_METRICS)
Model.summary()
```

The output shape becomes single-dimensional after the CNN layering is added as shown below.

```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d (Conv2D)              (None, 254, 254, 24)      240
_________________________________________________________________
batch_normalization (BatchNo (None, 254, 254, 24)      96
_________________________________________________________________
max_pooling2d (MaxPooling2D) (None, 127, 127, 24)      0
_________________________________________________________________
conv2d_1 (Conv2D)            (None, 127, 127, 64)      13888
_________________________________________________________________
dropout (Dropout)            (None, 127, 127, 64)      0
_________________________________________________________________
max_pooling2d_1 (MaxPooling2 (None, 63, 63, 64)        0
_________________________________________________________________
conv2d_2 (Conv2D)            (None, 63, 63, 64)        36928
_________________________________________________________________
dropout_1 (Dropout)          (None, 63, 63, 64)        0
_________________________________________________________________
max_pooling2d_2 (MaxPooling2 (None, 31, 31, 64)        0
_________________________________________________________________
conv2d_3 (Conv2D)            (None, 31, 31, 128)       73856
_________________________________________________________________
conv2d_4 (Conv2D)            (None, 31, 31, 128)       147584
_________________________________________________________________
dropout_2 (Dropout)          (None, 31, 31, 128)       0
_________________________________________________________________
max_pooling2d_3 (MaxPooling2 (None, 15, 15, 128)       0
_________________________________________________________________
conv2d_5 (Conv2D)            (None, 15, 15, 256)       295168
_________________________________________________________________
dropout_3 (Dropout)          (None, 15, 15, 256)       0
_________________________________________________________________
max_pooling2d_4 (MaxPooling2 (None, 7, 7, 256)         0
_________________________________________________________________
flatten (Flatten)            (None, 12544)             0
_________________________________________________________________
dense (Dense)                (None, 2352)              29505840
_________________________________________________________________
dropout_4 (Dropout)          (None, 2352)              0
_________________________________________________________________
dense_1 (Dense)              (None, 35)                82355
=================================================================
Total params: 30,155,955
Trainable params: 30,155,907
Non-trainable params: 48
_________________________________________________________________
```

We trained the set for 6 epochs so as to gain maximum accuracy.

[52]:
```
CNN_Model = Model.fit(Train_Set,
                      validation_data=Validation_Set,
                      callbacks=[Early_Stopper,Checkpoint_Model],
                      epochs=6)
```

```
Epoch 1/6
1082/1082 [==============================] - 449s 410ms/step - loss: 1.6410 - accuracy: 0.5474 - val_loss: 0.6189 - val_accuracy: 0.9444
Epoch 2/6
1082/1082 [==============================] - 321s 296ms/step - loss: 0.2307 - accuracy: 0.9263 - val_loss: 0.4368 - val_accuracy: 0.8867
Epoch 3/6
1082/1082 [==============================] - 323s 298ms/step - loss: 0.1290 - accuracy: 0.9600 - val_loss: 0.2336 - val_accuracy: 0.9719
Epoch 4/6
1082/1082 [==============================] - 322s 297ms/step - loss: 0.1006 - accuracy: 0.9682 - val_loss: 0.3364 - val_accuracy: 0.9550
Epoch 5/6
1082/1082 [==============================] - 328s 303ms/step - loss: 0.0802 - accuracy: 0.9741 - val_loss: 0.0754 - val_accuracy: 0.9888
Epoch 6/6
1082/1082 [==============================] - 328s 303ms/step - loss: 0.0810 - accuracy: 0.9741 - val_loss: 0.1107 - val_accuracy: 0.9805
```

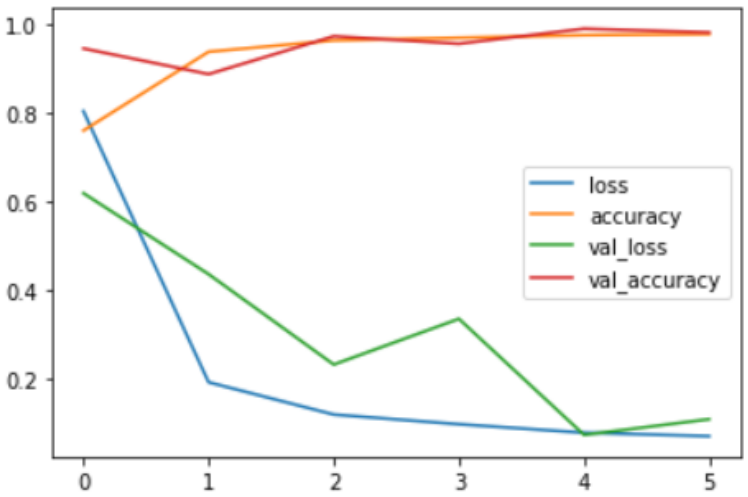
RESULTS-

Step 3. Visualizations-

A graph to represent the increase in accuracy with each iteration.

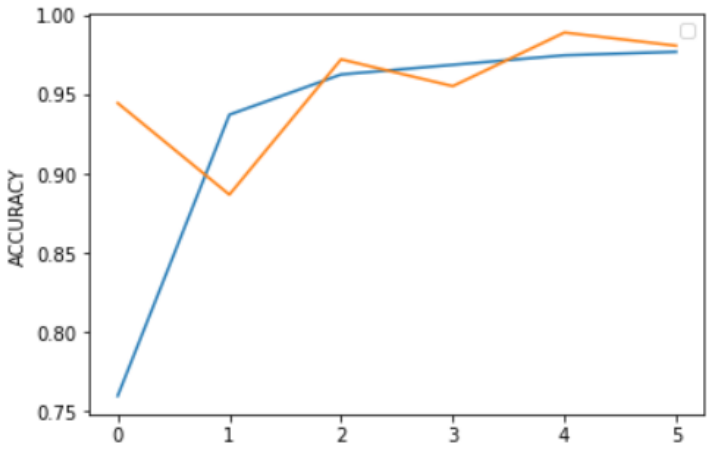
```
Grap_Data = pd.DataFrame(CNN_Model.history)
Grap_Data.plot()
```

```
<AxesSubplot:>
```



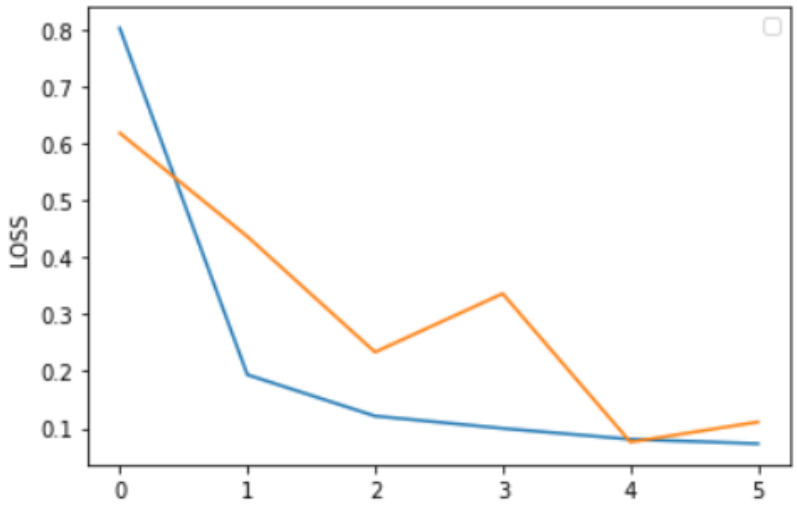
Graph representing the validation(test) accuracy and trained accuracy-

```
plt.plot(CNN_Model.history["accuracy"])
plt.plot(CNN_Model.history["val_accuracy"])
plt.ylabel("ACCURACY")
plt.legend()
plt.show()
```



Graph representing the validation(test) loss and trained loss-

```
plt.plot(CNN_Model.history["loss"])
plt.plot(CNN_Model.history["val_loss"])
plt.ylabel("LOSS")
plt.legend()
plt.show()
```



Step 4- Testing

Using the 10% testing dataset we get the predicted results. As shown below, the prediction is shown above an image and the actual value below it. For e.g., the first image shows the value S in Sign language and the predicted value is S as well.

Predictions of the test data set with a 0.98 accuracy.

Conclusion-

We were successfully able to use convolutional neural networks for correctly recognizing images of static sign language gestures. The results obtained showed a consistently high accuracy rate of 98% this goes on to show that given a proper dataset, and correctly cropped images, is an apt model for static sign language gesture recognition.

References-

- https://www.researchgate.net/publication/237054175_Recognition_of_Indian_Sign_Language_in_Live_Video
- https://ieeexplore.ieee.org/document/7916786
- https://www.kaggle.com/prathumarikeri/indian-sign-language-isl
- https://ieeexplore.ieee.org/document/8537248
- https://www.tensorflow.org/api_docs/python/tf
- https://keras.io/
- https://learnopencv.com/getting-started-with-opencv/
- https://towardsdatascience.com/covolutional-neural-network-cb0883dd6529