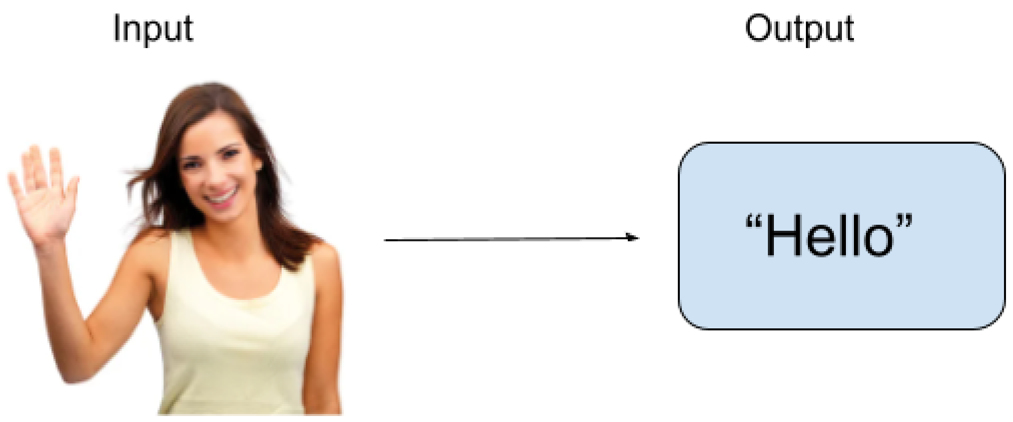
OpenCV based Sign Language Translator



Github Repository: https://github.com/dhyutin/sign-language-translator.git

Team Name: Visionaries

Team Members:

N. Sree Dhyuti - ced19i027@iiitdm.ac.in

A 4th year Computer Science Dual Degree student with a knack for CV

Projects:

Social Distancing Measure: A program that detects humans and the distance between them using user-defined Machine Learning algorithms and estimates if they are following social distancing protocols or not.

Tools used : Python

Leaf-based Plant Disease Classification:

A Machine Learning Program that uses CNNs to detect 9 tomato plant diseases that can be identified from the anomalies depicted on its leaf texture using various image segmentation techniques in Computer Vision and Machine Learning

Python Modules used: cv2, tensorflow, matplotlib, os

Vishal Akula - ced19i025@iiitdm.ac.in

An enthusiastic 4rth year Computer Science Dual Degree student.

Projects:

Emotion Detector : Used CNN to detect live emotion of a human face.

Stock price prediction : Used ML algorithms and descriptive statistics on past data to predict future stock price.

Problem Description:

Over 5% of our global population, i.e, 432 million adults and 34 million children have hearing disability. One of the major difficulties faced by deaf people is their ***means of communication.*** Not everyone around the world is aware of sign-language. Although many services are deaf-people-friendly, like provision of running subtitles in cinemas, providing written instructions near shops, etc, there are not many services available that put in efforts to understand what the deaf people want to convey.
In order to cater to the needs of deaf people and help them communicate better, there is a surging need for a mobile-sign language translation service which is handy.

Approach:

Our main aim is to facilitate the process of sign language translation by use of various ML algorithms and models.
Our Input is a*n image of a hand gesture of a person communicating in sign language.*
Our Output will be *English Text* corresponding to the translation of what the person said.
There are 4 major steps to facilitate the following:

1. Image Preprocessing
   a. Extract frames from the video and run the frames through certain preprocessing filters to prune unwanted data
2. Identification of human hand
   a. Using either available modules like *mediapipe* or building our own functions
3. Train a CNN to identify hand gestures
   a. The CNN architecture will be designed based on various logical and experimental iterations
   b. Aim here will be to attain maximum accuracy

Dataset:

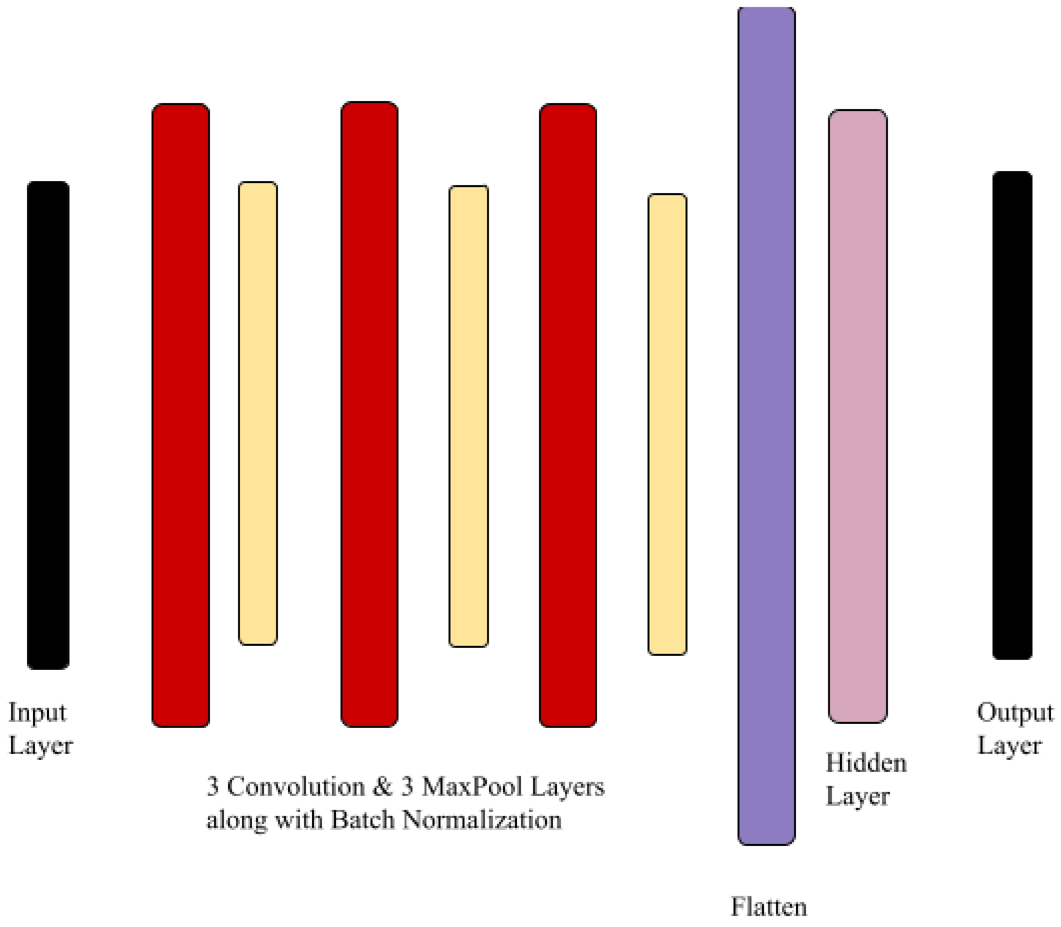
We have chosen the Standard MINST Sign Language Dataset for testing and training purposes.
Each training and test case represents a label (0-25) as a one-to-one map for each alphabetic letter A-Z (and no cases for 9=J or 25=Z because of gesture motions). The training data (27,455 cases) and test data (7172 cases) are approximately half the size of the standard MNIST but otherwise similar with a header row of label, pixel1,pixel2….pixel784 which represent a single 28x28 pixel image with grayscale

values between 0-255. The original hand gesture image data represented multiple users repeating the gesture against different backgrounds

---

Model:

To train our model, we have used a Convolutional Neural Network of the following architecture:


Input Layer
3 Convolution & 3 MaxPool Layers along with Batch Normalization
Flatten
Hidden Layer
Output Layer

1. An input layer
2. Convolution
3. Batch Normalization
4. Max Pooling
5. Convolution
6. Batch Normalization
7. Max Pooling
8. Convolution
9. Batch Normalization
10. Max Pooling
11. Flatten
12. Hidden Layer
13. Output

The model is a result of trial and error based experimentation and application of basic Computer Vision backed intuition.

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #   
=================================================================
 conv2d (Conv2D)             (None, 28, 28, 75)        750       
                                                                 
 batch_normalization (BatchN  (None, 28, 28, 75)       300       
 ormalization)                                                   
                                                                 
 max_pooling2d (MaxPooling2D  (None, 14, 14, 75)       0         
 )                                                               
                                                                 
 conv2d_1 (Conv2D)           (None, 14, 14, 50)        33800     
                                                                 
 dropout (Dropout)           (None, 14, 14, 50)        0         
                                                                 
 batch_normalization_1 (Batc  (None, 14, 14, 50)       200       
 hNormalization)                                                 
                                                                 
 max_pooling2d_1 (MaxPooling  (None, 7, 7, 50)         0         
 2D)                                                             
                                                                 
 conv2d_2 (Conv2D)           (None, 7, 7, 25)          11275     
                                                                 
 batch_normalization_2 (Batc  (None, 7, 7, 25)         100       
 hNormalization)                                                 
 max_pooling2d_2 (MaxPooling  (None, 4, 4, 25)         0
 2D)

 flatten (Flatten)          (None, 400)               0

 dense (Dense)              (None, 512)               205312

 dropout_1 (Dropout)        (None, 512)               0

 dense_1 (Dense)            (None, 25)                12825
=================================================================
Total params: 264,562
Trainable params: 264,262
Non-trainable params: 300
```

Model Output:

```
Epoch 1/15
40/40 [==============================] - 13s 309ms/step - loss: 2.1389 - accuracy: 0.3880 -
val_loss: 2.2153 - val_accuracy: 0.3490

Epoch 2/15
40/40 [==============================] - 12s 310ms/step - loss: 0.6160 - accuracy: 0.7920 -
val_loss: 1.2334 - val_accuracy: 0.5988

Epoch 3/15
40/40 [==============================] - 13s 326ms/step - loss: 0.2351 - accuracy: 0.9294 -
val_loss: 0.5511 - val_accuracy: 0.8450

Epoch 4/15
40/40 [==============================] - 13s 323ms/step - loss: 0.1007 - accuracy: 0.9762 -
val_loss: 0.3712 - val_accuracy: 0.8942

Epoch 5/15
40/40 [==============================] - 12s 310ms/step - loss: 0.0758 - accuracy: 0.9824 -
val_loss: 0.3993 - val_accuracy: 0.8818

Epoch 6/15
40/40 [==============================] - 12s 307ms/step - loss: 0.0418 - accuracy: 0.9914 -
val_loss: 0.2352 - val_accuracy: 0.9238

Epoch 7/15
40/40 [==============================] - 13s 323ms/step - loss: 0.0220 - accuracy: 0.9958 -
val_loss: 0.2040 - val_accuracy: 0.9376

Epoch 8/15
```

```
40/40 [==============================] - 13s 327ms/step - loss: 0.0202 - accuracy: 0.9966 -
val_loss: 0.1795 - val_accuracy: 0.9434

Epoch 9/15
40/40 [==============================] - 13s 329ms/step - loss: 0.0151 - accuracy: 0.9978 -
val_loss: 0.2452 - val_accuracy: 0.9296

Epoch 10/15
40/40 [==============================] - 15s 385ms/step - loss: 0.0099 - accuracy: 0.9992 -
val_loss: 0.1619 - val_accuracy: 0.9510

Epoch 11/15
40/40 [==============================] - 15s 366ms/step - loss: 0.0091 - accuracy: 0.9988 -
val_loss: 0.1502 - val_accuracy: 0.9546

Epoch 12/15
40/40 [==============================] - 11s 288ms/step - loss: 0.0096 - accuracy: 0.9988 -
val_loss: 0.1649 - val_accuracy: 0.9584

Epoch 13/15
40/40 [==============================] - 13s 323ms/step - loss: 0.0057 - accuracy: 0.9998 -
val_loss: 0.1521 - val_accuracy: 0.9574

Epoch 14/15
40/40 [==============================] - 12s 305ms/step - loss: 0.0045 - accuracy: 0.9998 -
val_loss: 0.1309 - val_accuracy: 0.9588

Epoch 15/15
40/40 [==============================] - 12s 310ms/step - loss: 0.0064 - accuracy: 0.9990 -
val_loss: 0.1487 - val_accuracy: 0.9616
```

CNN Error: 3.84%
CNN Accuracy: **96.16%**

Conclusion:

In conclusion, we built a CNN model which takes 28x28 resolution images as input to predict hand-signs(in the image) with approximately 96% accuracy.
Modules like openCV, tensorflow, keras, matplotlib were used for the same.

Resources:

https://opencv.org/core-opencv/
https://github.com/Gogul09/gesture-recognition

https://www.youtube.com/watch?v=NZde8Xt78Iw&list=WL&index=210&t=28s
https://www.kaggle.com/datasets/datamunge/sign-language-mnist
https://www.youtube.com/watch?v=SgKOTgrGnXI

THE END
