Final Report for Group 34
Translating Sign Language to English

Sagnik Chatterjee
IIIT-Delhi
Delhi-110020
sagnikc@iiitd.ac.in

Pramil Panjawani
IIIT-Delhi
Delhi-110020
pramilp@iiitd.ac.in

Abstract

Our project focuses on experimenting with various machine learning techniques to see which technique gives us a sign language recognition model with the best bang for its buck. We chose the publicly available Sign Language MNIST dataset on Kaggle for the first phase of our project. For the second phase of our project we built our own custom dataset and used that to provide real time translation from Sign Language to English.

1. Introduction

Deaf and mute people have always been a very marginalized section of our society. We firmly believe that the first step to inclusion is communication. If we manage to come up with a model that can provide a decent translation, then the battle related to communication is already won. This in a nutshell is our motivation for picking this particular project topic.

2. Related Work

Most of the related work in this field uses their custom built dataset. On Kaggle we found that the highest accuracy obtained on the test set was 94%[1] using CNN.

3. Dataset and Evaluation

For the first phase of our project we benchmarked our models on the American Sign Language MNIST dataset as available on Kaggle. A batch of 16 images selected at random from the training dataset is shown in Figure 1. For the latter half of our project we have built our own custom dataset which we have used to train our model.

[1] https://www.kaggle.com/pchoruzy/sign-language-recognition-with-keras-acc-0-94

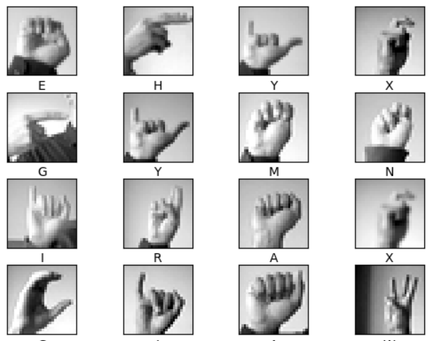

Figure 1. 16 random images (normalized to Gray-scale) from the Sign Language MNIST dataset

3.1. Feature Extraction

3.1.1 Static Data

We have used a pipeline of StandardScaler() and Principal Component Analysis for feature extraction. The PCA model was set to extract features such that 95% of the variance in the original data would be preserved (shown in Figure 2). We also used Histogram of Oriented Gradients as feature descriptors for training some of our models. Unfortunately we were not able to get a good performance on test datasets using HOG descriptors possibly because of the small size of the images.

3.1.2 Real Time

For images captured via webcam we have followed a tried and tested[2][3] pipeline.

[2] https://github.com/Evilport2/Sign-Language
[3] https://github.com/harshbg/Sign-Language-Interpreter-using-Deep-Learning

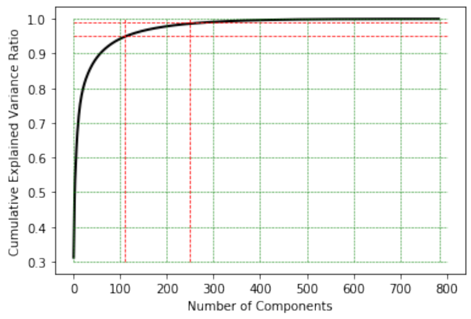

Figure 2. Cumulative Explained Variance Ratio vs Number of Components

1. Create a histogram of a hand in current lighting conditions.
2. Change colorspace of target image to Hue Saturation value model.
3. Back-project the target image to fit our histogram.
4. Based on the back projection, we apply an elliptical filter to detect the edges better.
5. Perform smoothing on the target image by applying Gaussian blur and Median blur in succession.
6. Perform Otsu's thresholding on the image.
7. Change image to 3 channels and crop the required part of the target image.

3.2. Evaluation Metrics

3.2.1 Static Data

Our evaluation metrics were Accuracy Score, F1-Score and Training Time.

3.2.2 Real Time

Our evaluation metrics are Loss, Accuracy and F1-Score for the CNN model.

4. Methodology

We had trained 6 different models on the static MNIST American Sign language dataset. The evaluation metrics are given in Table 1. Since we found that CNN gives the best test accuracy and F-1 score of all the six models, we went ahead with using CNN as our classifier for the real time data. The details of our CNN model is given in the next section. Since our CNN uses only two hidden layers, there is very low probability of overfitting the dataset.

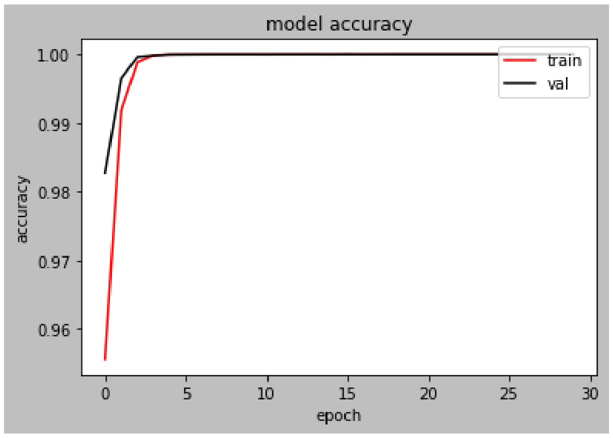

Figure 3. Accuracy vs Epoch on MNIST Sign Language Dataset

4.1. Details of our CNN model

- **Input Layer**: 32 feature extractors with a 3×3 convolutional kernel. The input shape is 50×50×3. This is followed by a Max Pooling Layer with a 2×2 kernel. The activation function is Recitified linear unit.
- **Hidden Layers**: The first hidden layer is again a 32 feature extraction layer with a 3×3 kernel. This is followed by a Max Pooling Layer with a 2×2 kernel. The activation function is Recitified linear unit. The Pooling Layer is followed by a Flattening Layer. The second hidden layer is a fully connected layer with 128 units. The activation function is Recitified linear unit.
- **Output layer**: Activation Function is Recitified linear unit. Classification is done using Softmax classifier.
- **Loss**: Categorical Cross-Entropy for multi-class classification, and Binary Cross-Entropy for binary classification.
- **Metrics**: Accuracy.
- **Optimizer**: Adam Optimizer.
- **Epoch**: We ran the CNN for 30 epochs using Stochastic Gradient Descent.

The Loss versus Epochs and Accuracy versus Epochs graphs are plotted in Figures 3 and 4.

5. Results

5.1. Static

The results are detailed in Table 1. By far the best performing model we get is unsurprisingly CNN. Logistic Regression serves as a good baseline model.
We see that our CNN model performs at par with the accuracy of the best kernel on Kaggle as mentioned in Section 2,

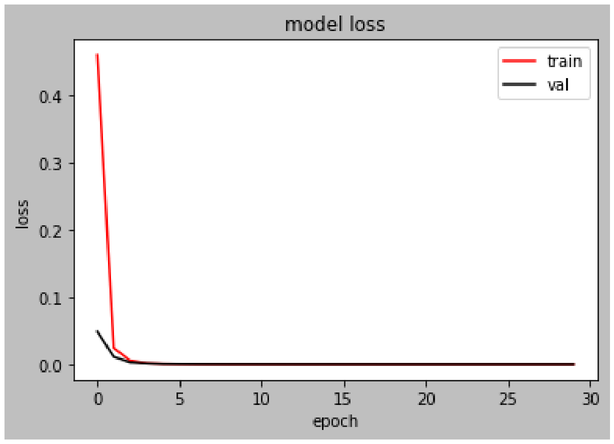

Figure 4. Loss vs Epoch on MNIST Sign Language Dataset

when the principal components of the images are passed as input and output. Therefore our feature extraction step with PCA was extremely necessary.

5.2. Real Time

Our CNN model achieves high train accuracy over the custom dataset within 6 epochs (see Figure 3). The test accuracy metric was not measured since the data is being predicted in real time, but it was extremely accurate given favourable backgrounds and lighting conditions. A few images of our program in action are given in Figures 5, 6 and 7.

6. Contributions

6.1. Deliverables

- **Sagnik**: Initially planned to build robust models to classify static data. Finally ended up training all non-CNN models on MNIST data, and wrote the scripts to perform Hand Gesture Recognition.
- **Pramil**: Responsible for real time data handling. Created the Convolution Neural Network to accurately classify the real time input.

6.2. References

The following resources were invaluable as a reference for the code.

- https://www.tensorflow.org/tutorials
- https://docs.opencv.org/2.4/doc/tutorials/tutorials.html
- https://docs.opencv.org/master/index.html
- https://medium.freecodecamp.org/weekend-projects-sign-language-and-static-gesture-recognition-using-scikit-learn-60813d600e79
- https://github.com/harshbg/Sign-Language-Interpreter-using-Deep-Learning
- https://github.com/Evilport2/Sign-Language

6.3. Individual Contributions

- **Sagnik**: Training of all non-CNN models on MNIST data, and wrote the scripts to perform Hand Gesture Recognition. The python files 'model_non_cnn.py', and 'train.py' were contributed by him. The methods set_hist_spot(), get_hist() and predict_images() in 'predict_hand_sign.py' were also contributed by him.
- **Pramil**: Responsible for real time data handling. Created the Convolution Neural Network to accurately classify the real time input. The methods accuracy(), buildNN() and load_data() in 'predict_hand_sign.py' were contributed by him.

Model	Time to train (secs)	Train Accuracy Score	Train F1-Score	Test Accuracy Score	Test F1-Score
GNB	0.4724	0.4600	0.4554	0.3899	0.3904
GNB(PCA)	0.3466	0.7922	0.7975	0.6087	0.6240
MNB	0.1176	0.5460	0.5463	0.4629	0.4598
CNB	0.1190	0.3825	0.3514	0.3864	0.3448
Logistic Regression	664.461	1.0	1.0	0.6264	0.6322
Logistic Regression (PCA)	198.405	0.9681	0.9681	0.5891	0.5874
Decision Tree	18.654	0.8579	0.8631	0.4095	0.4252
Decision Tree (PCA)	5.98	0.8728	0.8760	0.4546	0.4608
CNN	363	1.0	1.0	0.9244	0.9385
CNN (PCA)	354	1.0	1.0	0.9393	0.9609

Table 1. Evaluating the various models we have explored.

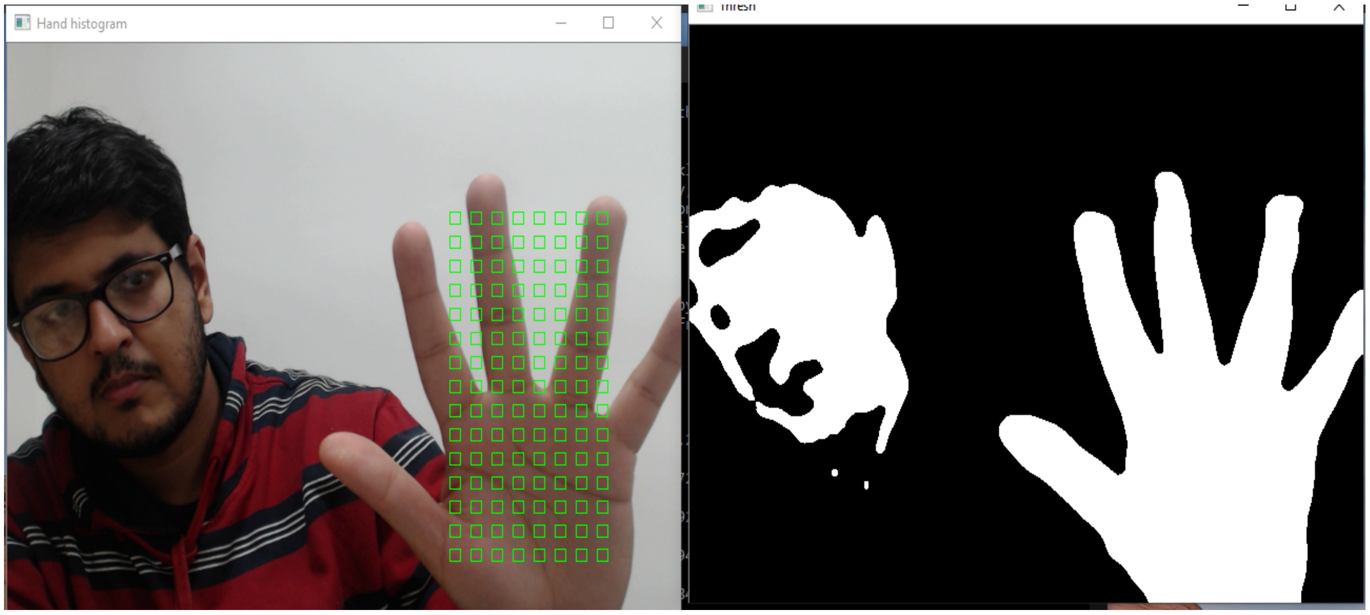

Figure 5. Capturing the Histogram

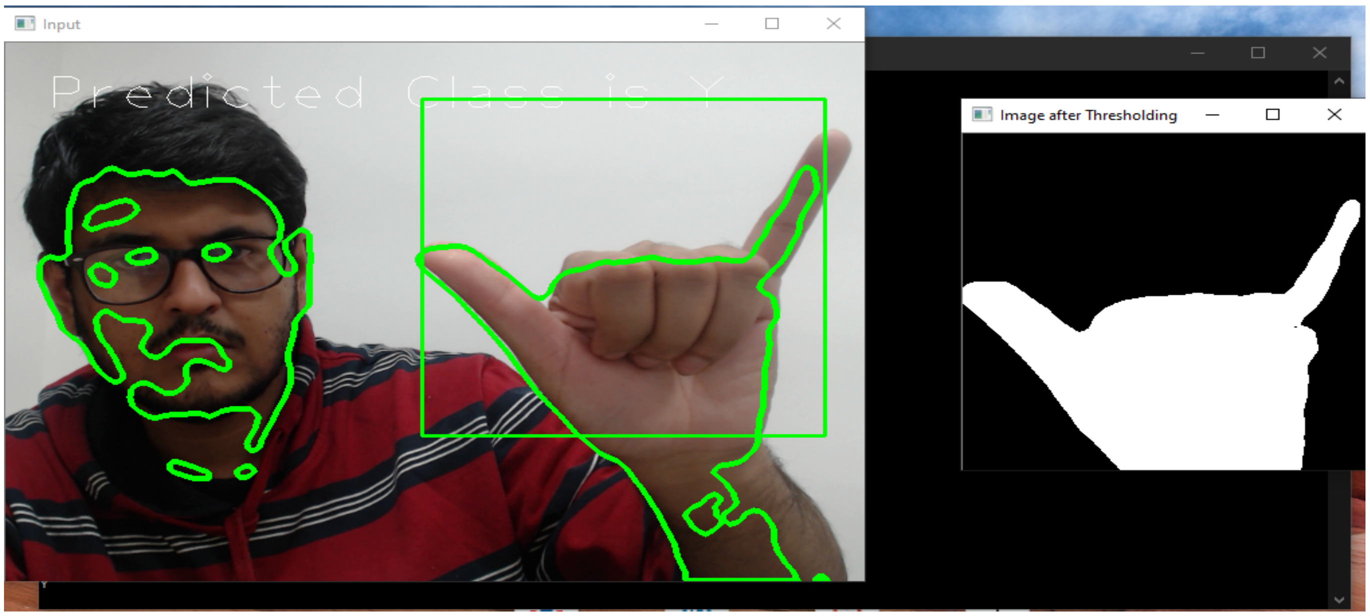

Figure 6. Our program correctly predicting that Sign corresponds to the alphabet Y

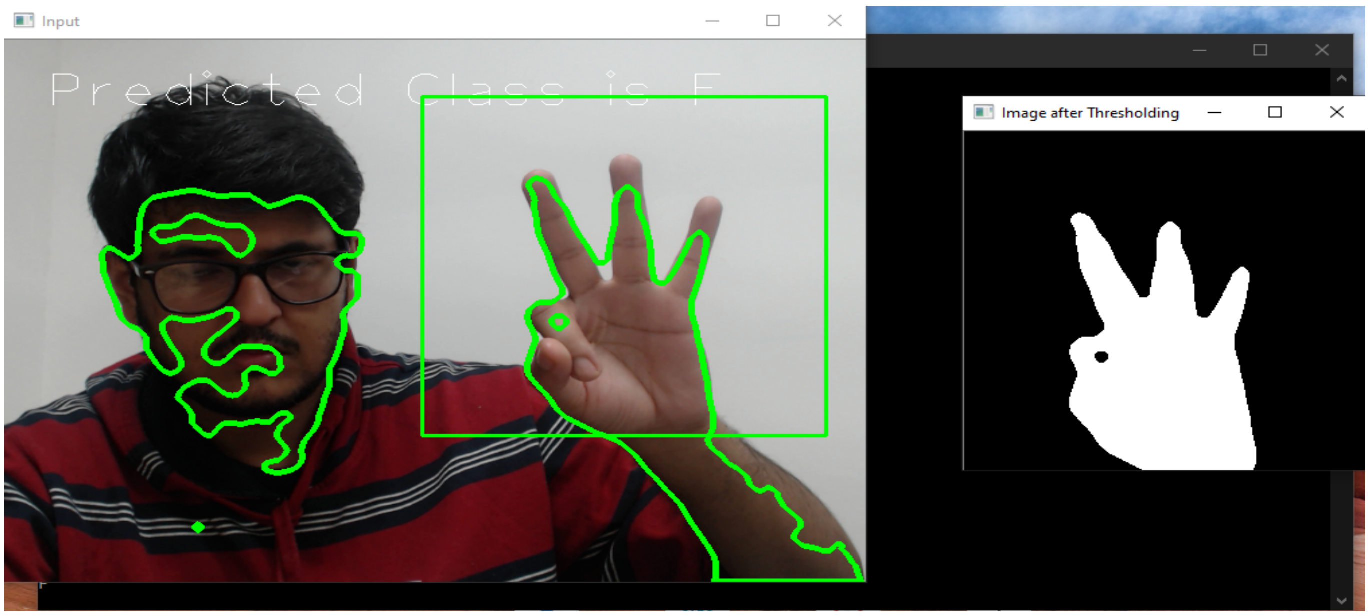

Figure 7. Our program correctly predicting that Sign corresponds to the alphabet F