HandTalk: Translating Sign Language to Text

Aakash, Parveen, Shubham Sharma, Pourav Surya
Indraprastha Institute of Information Technology Delhi
Delhi, India
{aakash21002, parveen21079, shubham21099, pourav21271}@iiitd.ac.in

Abstract

The motivation behind developing a machine learning-based sign language detection system stems from the need to bridge communication gaps between deaf or hard-of-hearing individuals and the hearing population. This underrepresentation leads to social and professional barriers for those who rely on sign language as their primary means of communication.

1. Introduction

Sign Language serves as a important means of communication for millions of people across the globe who are either completely deaf or hardly hears. However, the communication barrier between sign language users and those who don't understand sign language remains a significant challenge in today's society. This disconnect affects various aspects of daily life, from educational and professional opportunities to healthcare access and social interactions.

To address this problem, we have developed a robust sign language detection model that takes an image of a hand gesture as input and outputs the corresponding alphabet associated with that sign. Several machine learning models, including **Logistic Regression, Decision Tree, Random Forest, Perceptron,** and **Multi-Layer Perceptron**, were trained for the sign language classification task. Among these, the model with the best performance in terms of accuracy and loss will be selected for the final sign language detection task.

2. Literature Review

A New Benchmark on American Sign Language Recognition using Convolutional Neural Network: This study proposes a novel convolutional neural network (CNN) model to enhance American Sign Language (ASL) recognition accuracy. Evaluated on four publicly available ASL datasets, the model, applied to alphabet and numeral images, achieves a 9% improvement in accuracy over existing methods [1].

Real-Time Sign Language Detection Using CNN : To detect real-time sign language, a dataset is prepared on which a customized CNN model is trained. In the findings, it was observed that the customized CNN model can achieve the highest 98.6% accuracy [2].

Deep convolutional neural networks for sign language recognition : To address the lack of mobile selfie sign language datasets, they created one with five subjects performing 200 signs from five angles and various backgrounds, capturing 60 frames per sign. Their approach achieved a 92.88% recognition rate [3].

3. Dataset Used

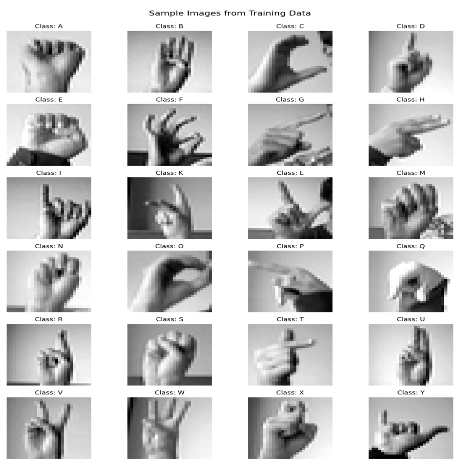

Figure 1. Sample Images

The Dataset used for the training and testing purpose is Sign Language MNIST. This dataset contains the American Sign Language data. 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). There is a total of 784 pixel which represent a single 28x28 pixel image with grayscale values between 0-255.

4. Methodology

The major methodolgy part consists of applying different model on the trained data, calculate and analyze their accuracy, hyperparameter tuning wherever needed and finding the model that performs best in terms of accuracy or loss.

4.1. Data Visualization and Processing

Below is the class distribution of each classes for training data. There are total of 24 class each representing a particular alphabet.

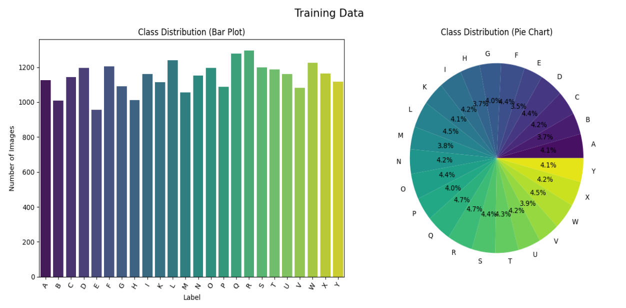

Figure 2. Class Distribution of Training data

All the grayscale pixel values of the images are normalized to scale them to a range of [0, 1].

4.2. Models details

The models used for classification purpose were Logistic Regression, Decision Tree, Random Forest, Perceptron, Multi Layer perceptron. Each model is trained on the best parameter found using grid search cross validation with 3 fold cross validation, and accuracy, precesion, recall and F1 score is calculated and analyzed.

4.2.1 Logistic Regression

Logistic Regression is trained on the training data for the classification purpose. We have used various combinations of L1 and L2 regularization for the penalty using grid search cross-validation and found the best model out of it.

- **penalty**: Specifies the type of regularization (`l1`, `l2`, `elasticnet`, or `None`) to control overfitting.
- **c**: Controls the inverse regularization strength. Lower values indicate stronger regularization.
- **solver**: The optimization algorithm used for fitting the model. `saga` supports `l1`, `l2`, and `elasticnet` penalties.
- **max_iter**: Defines the maximum number of iterations for the solver to converge. Higher values ensure better convergence.

4.2.2 Decision Tree

The Decision Tree classifier was implemented with extensive hyperparameter tuning through grid search cross-validation. These are the hyperparameters from which we find the best parameters:

- **criterion**: The function to measure the quality of a split (`gini` or `entropy`).
- **splitter**: Strategy to choose the split at each node (`best` or `random`).
- **max_depth**: The maximum depth of the tree. Limits growth to prevent overfitting.
- **min_samples_split**: The minimum number of samples required to split an internal node.
- **min_samples_leaf**: The minimum number of samples required to be at a leaf node.

4.2.3 Random Forest

The Random Forest classifier was implemented with hyperparameter tuning using grid search cross-validation. These are the hyperparameters from which we find the best parameters:

- **n_estimators**: The number of trees in the forest. More trees generally improve performance but increase computation time.
- **criterion**: Function to measure the quality of a split (`gini` or `entropy`).
- **max_depth**: Maximum depth of the tree. Controls model complexity.
- **min_samples_split**: Minimum samples required to split a node.
- **min_samples_leaf**: Minimum samples required at a leaf node.

4.2.4 Perceptron

The Perceptron is inherently a binary classifier, but when scikit-learn's Perceptron gets multiclass dataset, it uses the one-vs-rest strategy by default. These are the hyperparameters from which we find the best parameters:

- **penalty**: Specifies the type of regularization (`l1`, `l2`, `elasticnet`, or `None`).
- **alpha**: Constant that multiplies the regularization term to control overfitting.
- **max_iter**: Maximum number of iterations over the training data.
- **tol**: The stopping criterion. Training stops when the change in the loss is below this threshold.

4.2.5 Multi-Layer Perceptron

The MLP (Multilayer Perceptron) is a class of feedforward artificial neural networks. It consists of at least three layers : an input layer, one or more hidden layers, and an output layer. Each layer is fully connected to the next layer. These are the hyperparameters from which we find the best parameters:

- **hidden_layer_sizes**: The number of neurons in the hidden layers, controlling network complexity.
- **activation**: Activation function for the hidden layers (`relu` or `tanh`).
- **solver**: The algorithm for weight optimization (`adam` or `sgd`).
- **alpha**: L2 penalty (regularization term) to prevent overfitting.
- **learning_rate**: Learning rate schedule for weight updates (`constant` or `adaptive`).
- **max_iter**: Maximum number of iterations for training.

5. Results and Analysis

After performing the grid search on the parameters described for the specific model, the best parameters for the following models were found.

5.1. Logistic Regression

Hyperparameter	Values (Grid Search)	Best Value
penalty	{'l1', 'l2', 'elasticnet', None}	'l1'
C	{0.01, 0.1, 1, 10}	0.1
solver	{'saga'}	'saga'
max_iter	{100, 200}	200

Table 1. Logistic Regression Hyperparameter Grid and Best Parameters from Grid Search

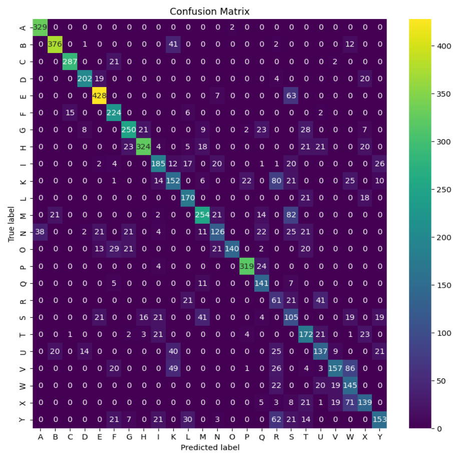

Figure 3. Confusion Matrix of Logistic Regression

5.2. Decision Tree

Hyperparameter	Values (Grid Search)	Best Value
criterion	{'gini', 'entropy'}	'entropy'
splitter	{'best', 'random'}	'best'
max_depth	{None, 10, 20, 30}	None
min_samples_split	{2, 10, 20}	2
min_samples_leaf	{1, 5, 10}	1

Table 2. Decision Tree Hyperparameter Grid and Best Parameters

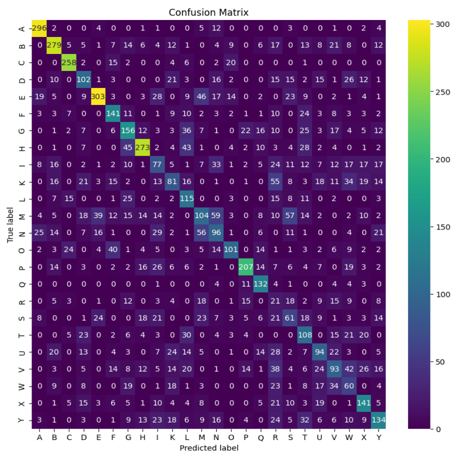

Figure 4. Confusion Matrix of Decision Tree

5.3. Random Forest

Hyperparameter	Values (Grid Search)	Best Value
n_estimators	{50, 100, 200}	200
criterion	{'gini', 'entropy'}	'gini'
max_depth	{None, 10, 20}	20
min_samples_split	{2, 10}	2
min_samples_leaf	{1, 5}	1

Table 3. Random Forest Hyperparameter Grid and Best Parameters

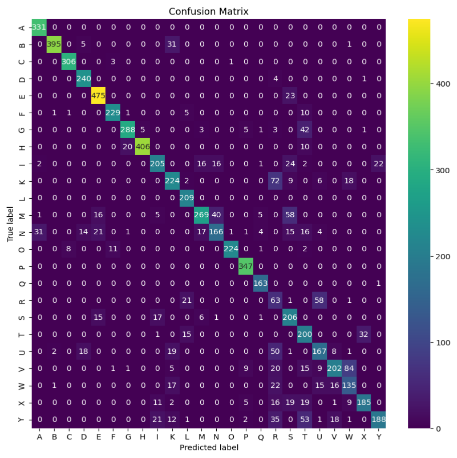

Figure 5. Confusion Matrix of Random Forest

5.4. Perceptron

Hyperparameter	Values (Grid Search)	Best Value
penalty	{'l1', 'l2', 'elasticnet', None}	'l1'
alpha	{0.0001, 0.001, 0.01}	0.001
max_iter	{1000}	1000
tol	{1e-3}	0.001

Table 4. Perceptron Hyperparameter Grid and Best Parameters

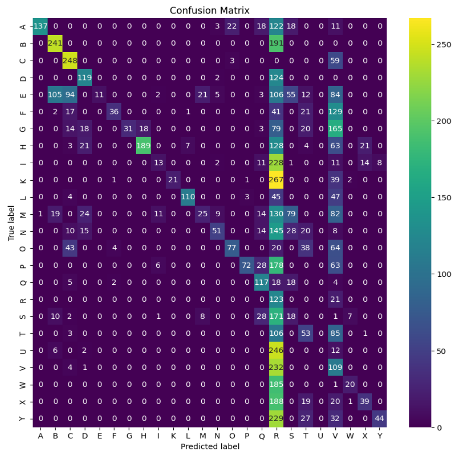

Figure 6. Confusion Matrix of Perceptron

5.5. Multi-Layer Perceptron

Hyperparameter	Values (Grid Search)	Best Value
hidden_layer_sizes	{(128,), (128, 64), (256, 128)}	(256, 128)
activation	{'relu', 'tanh'}	'relu'
solver	{'adam', 'sgd'}	'adam'
alpha	{0.0001, 0.001}	0.001
learning_rate	{'constant', 'adaptive'}	'constant'
max_iter	{500}	500

Table 5. MLPClassifier Hyperparameter Grid and Best Parameters

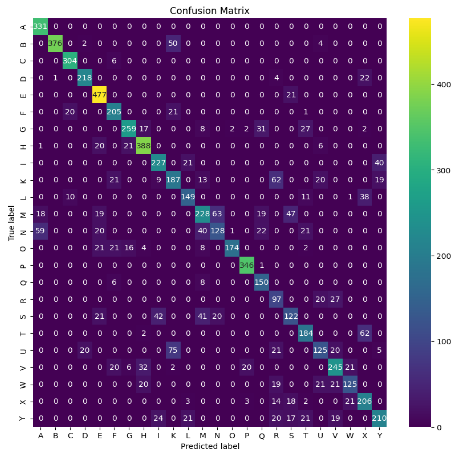

Figure 7. Confusion Matrix of Multi Layer Perceptron Classifier

5.6. Performance Metrics

All the models with their best parameters are trained on the train dataset and then were test on the testing dataset. For all the models accuracy, precision, recall and F1-score are calculated on the test dataset and summarized in the below table.

Model	Accuracy	Precision	Recall	F1-Score
Logistic Regression	0.693809	0.724626	0.693809	0.698777
Decision Tree Classifier	0.478667	0.498118	0.478667	0.483469
Random Forest Classifier	0.811907	0.835269	0.811907	0.815962
Perceptron	0.265477	0.651270	0.265477	0.298552
MLP Classifier	0.761433	0.766137	0.761433	0.758345

Table 6. Model Performance Metrics

6. Conclusions

From table 6, it can be inferred that Random Forest Classifier is the best model, outperforming other models in terms of accuracy, precision, recall and F1 score. This indicates that the Random Forest model is the most effective at correctly classifying the instances, maintaining a good balance between precision and recall.

7. Future Work

Future works include training CNN and SVM, improving the accuracy further of all the model, testing on video frames by processing each frames and predicting the alphabet corresponding a specific sign language.

8. Work Distribution

- **Aakash**: Decision Tree, Hyperparameter tuning, Data processing, Performance metric
- **Parveen**: Logistic Regression, Perceptron, Hyperparameter tuning, Data processing
- **Shubham Sharma**: EDA, Random Forest, Hyperparameter tuning, Data visualization and processing
- **Pourav Surya**: MLP classifier, Confusion matrix, Testing on test dataset

References

[1] M. M. Rahman, M. S. Islam, M. H. Rahman, R. Sassi, M. W. Rivolta, and M. Aktaruzzaman, "A New Benchmark on American Sign Language Recognition using Convolutional Neural Network," in *2019 International Conference on Sustainable Technologies for Industry 4.0 (STI)*, pp. 1–6, 2019.

[2] M. N. Saiful, A. A. Isam, H. A. Moon, R. T. Jaman, M. Das, M. R. Alam, and A. Rahman, "Real-Time Sign Language Detection Using CNN," in *2022 International Conference on Data Analytics for Business and Industry (ICDABI)*, pp. 697–701, 2022.

[3] G. A. Rao, K. Syamala, P. V. V. Kishore, and A. S. C. S. Sastry, “Deep convolutional neural networks for sign language recognition,” in *2018 Conference on Signal Processing And Communication Engineering Systems (SPACES)*, pp. 194–197, 2018.