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

HandTalk uses machine learning to translate static hand gestures into text, bridging communication between sign language users and non-users. Initially, we tested traditional models like Logistic Regression, Decision Trees, Random Forest, and MLP. Advanced techniques, including CNNs, SVMs were later implemented. The CNN model achieved the highest accuracy of 93.52%, demonstrating its effectiveness. This report outlines our methodology, progress, and key findings.
Github Link

1. Introduction

Millions of individuals rely on sign language for communication, but its limited adoption creates barriers in education, healthcare, and workplaces. HandTalk addresses this by detecting hand gestures and mapping them to corresponding alphabets using image-based classification techniques.

Our system uses the Sign Language MNIST dataset to train multiple machine learning models, refining their performance through hyperparameter tuning. The final goal is to develop a real-time, deployable solution.

2. Literature Review

Several studies inspired our work:

- **Rahman et al. (2019)** developed CNN models for American Sign Language (ASL), achieving 9% better accuracy than existing benchmarks [1].
- **Saiful et al. (2022)** demonstrated a custom CNN for real-time ASL detection, achieving 95.6% accuracy [2].
- **Rao et al. (2018)** created a dataset for selfie-based ASL recognition, achieving 92.88% accuracy [3].

These works emphasize the efficacy of CNNs for gesture recognition, guiding our adoption of deep learning techniques.

3. Dataset

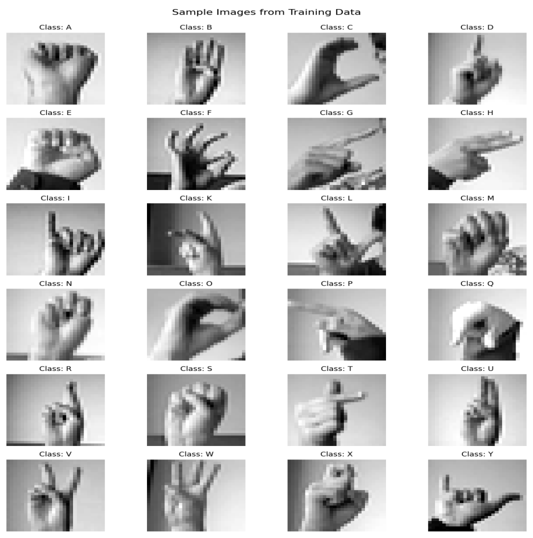

Figure 1. Sample Images

We used the Sign Language MNIST dataset, containing 784 pixe values of grayscale image of 28×28 representing 24 classes. Each class corresponds to a static alphabet gesture, excluding 'J' and 'Z'.

Preprocessing Steps:

- Normalized pixel values to [0, 1].
- Applied PCA to reduce features from 784 to 50 for faster training.

4. Methodology

The major methodolgy part consists of applying different model on the trained data, hyperparameter tuning wherever needed, finding the best parameter and again training on that best parametre and finally finding the model that performs best in terms of accuracy or loss.

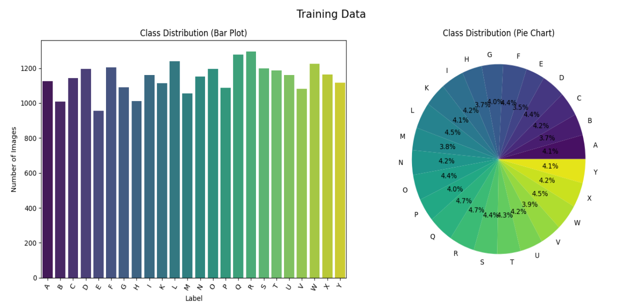

Figure 2. Class Distribution of Training data

4.1. Models details

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

4.1.1 Logistic Regression

Below are the various parameters that are used for training Logistic Regression

- **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.

4.1.2 Decision Tree

Below are the various parameters that are used for training Decision Tree.

- **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.

4.1.3 Random Forest

Below are the various parameters that are used for training Random Forest

- **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.

4.1.4 Perceptron

Below are the various parameters that are used for training Perceptron.

- **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.

4.1.5 Multi-Layer Perceptron

Below are the various parameters that are used for training Multi-layer Perceptron.

- **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.

4.1.6 Support Vector Machine (SVM)

Below are the various parameters that are used for training SVM.

- **C**: Balances training error and model complexity. Higher C increases flexibility but risks overfitting (`0.1`, `1`, `10`).
- **kernel**: The function used to map the data into a higher-dimensional space. Common kernel types include (`linear` or `rbf`).
- **gamma**: A parameter for the RBF kernel that controls the influence of individual training samples. (`scale` or `auto`)

4.1.7 Convolutional Neural Network (CNN)

Below are the various parameters that is used for training CNN.

Layer Type	Parameters
Conv2D (1)	Filters: 32, Kernel: (3, 3), ReLU, Input (28,28,1)
MaxPool2D (1)	Pool size: (2, 2), Strides: 2
Conv2D (2)	Filters: 128, Kernel: (3, 3), Activation: ReLU
MaxPool2D (2)	Pool size: (2, 2), Strides: 2
Conv2D (3)	Filters: 256, Kernel: (3, 3), Activation: Tanh
MaxPool2D (3)	Pool size: (2, 2), Strides: 2
Dense (1)	Units: 64, Activation: ReLU
Dense (2)	Units: 128, Activation: Tanh
Dense (3)	Units: 128, Activation: Tanh
Output Layer	Units: 25, Activation: Softmax
Optimizer	Adam, Learning Rate: 0.0005
Loss Function	Sparse Categorical Crossentropy
Metrics	Accuracy

Table 1. CNN Model Parameters

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 2. Logistic Regression Hyperparameter Grid and Best Parameters from Grid Search

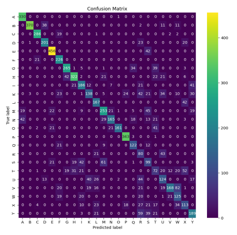
Figure 3. Confusion Matrix of Logistic Regression

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 3. Decision Tree Hyperparameter Grid and Best Parameters

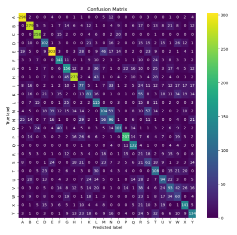
Figure 4. Confusion Matrix of Decision Tree

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 4. Random Forest Hyperparameter Grid and Best Parameters

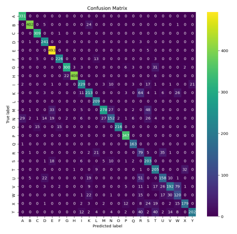
Figure 5. Confusion Matrix of Random Forest

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 5. Perceptron Hyperparameter Grid and Best Parameters

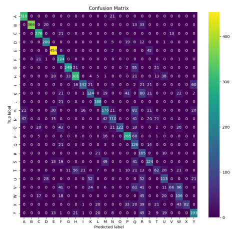

Figure 6. Confusion Matrix of Perceptron

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 6. MLPClassifier Hyperparameter Grid and Best Parameters

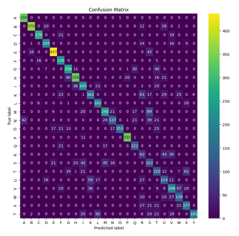

Figure 7. Confusion Matrix of Multi Layer Perceptron Classifier

5.6. Support Vector Machine (SVM)

Hyperparameter	Values (Grid Search)	Best Value
C	{0.1, 1, 10}	0.1
kernel	{'linear', 'rbf'}	'linear'
gamma	{'scale', 'auto'}	'scale'

Table 7. SVM Hyperparameter Grid and Best Parameters

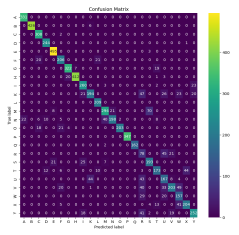

Figure 8. Confusion Matrix of SVM

5.7. Convolutional Neural Network (CNN)

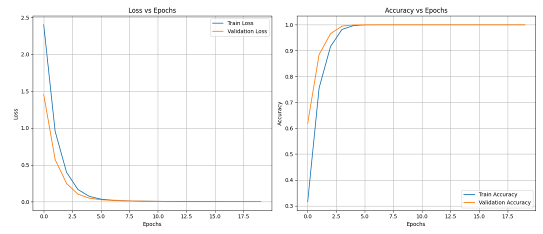

Figure 9. Accuracy vs Epochs and Loss vs Epochs in CNN

5.8. 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
SVM	0.8419	0.8568	0.8419	0.8444
CNN (Sequential)	0.9352	0.9376	0.9352	0.9347

Table 8. Model Performance Metrics

6. Conclusions

From table 8, it can be inferred that CNN is the best model, outperforming other models in terms of accuracy, precision, recall and F1 score. This highlights the effectiveness of CNNs for image-based classification, the model demonstrate strong potential for real-time sign language recognition.

Key Findings: CNN demonstrated superior performance due to its ability to capture spatial hierarchies.

7. Work Distribution

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

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.