Sign Language Recognition

Authors: Congkai Sun, Yiyang Jiang, Jinxuan Zhang

Introduction

Sign languages are crucial for enabling communication among the deaf and hard-of-hearing community. They consist of a series of hand gestures, facial expressions, and body movements that convey meaning. However, the communication gap between the hearing and deaf communities persists due to the limited knowledge of sign languages among the general population. The objective of this project is to develop a system that can recognize and translate sign language into text or speech, thus bridging the communication gap and making the world more inclusive for deaf individuals.

We collected a dataset of images showcasing various hand gestures corresponding to the American Sign Language (ASL) alphabet and trained each model on this data. The Sign Language MNIST dataset only includes static images, which means that letters J and Z are excluded because they require motion to be accurately represented. Analyzing and recognizing sign language gestures in real-world applications may require considering motion information. Also, the dataset has undergone several preprocessing steps, including cropping, grayscaling, resizing, and creating multiple variations with different filters and rotations. These modifications can introduce variations in the dataset, which could affect the performance of models trained on it. Additionally, the small size of the images (28x28 pixels) might make it challenging to extract fine-grained features necessary for accurate classification.

In this project, we analyzed and compared the performance of three machine learning algorithms - K-Nearest Neighbors, Convolutional Neural Networks, and Logistic Regression - to recognize and translate sign language into text.

Methods

1. Data Preprocessing
2. Model Implementation (KNN, CNN, Logistic Regression)
3. Model Evaluation (confusion matrix, precision, recall, f1-score)

In our analysis, we employed three machine learning algorithms - K-Nearest Neighbors (KNN), Convolutional Neural Networks (CNN), and Logistic Regression - to recognize and translate sign language. The methodology can be summarized as follows:

Data Collection and Preprocessing:

We collected a dataset of images representing hand gestures for the American Sign Language (ASL) alphabet. The Dataset includes 27,455 training sets and 7172 test sets, each containing 784 pixels (28x28), with grayscale values ranging from 0-255. The original hand gesture image data represented multiple users repeating the gesture against different backgrounds.

The labels from the dataset are 0-24 except J and Z which are corresponding to 9 and 25. So first we created a letter's label for each numerical label in the dataset and visualize them.

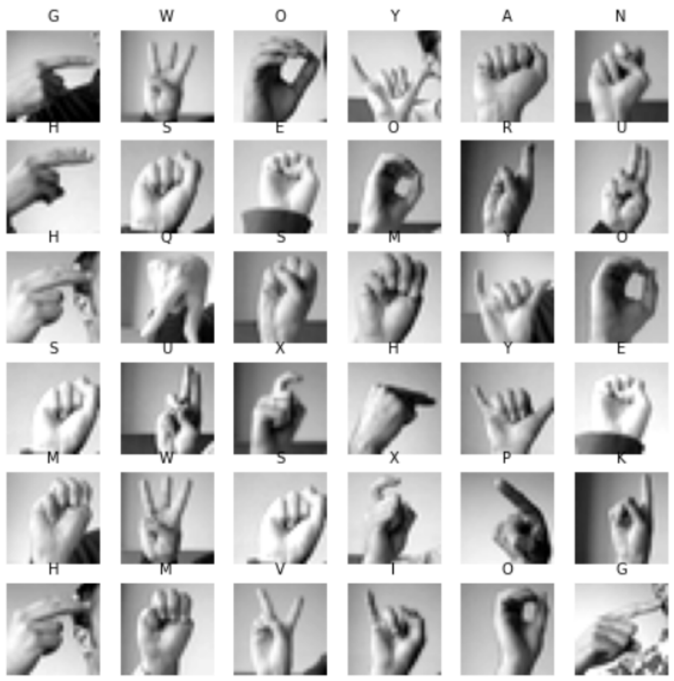


Figure 1: Gestures corresponding to letters

The training set is split into a separate validation set to evaluate the performance of the model during training, without using the test set. The validation set contains 2500 samples and the stratified splitting ensures that the distribution of classes in the validation set is similar to that of the original training set.

Then we scaled the training set features which can help improve the performance and convergence speed of these algorithms using StandardScaler. And applied the same scaling transformation as computed on the training set to the validation and test sets. The StandardScaler is a popular preprocessing technique that scales the input features by transforming them to have zero mean and unit variance (standard deviation of 1). This is done by subtracting the mean and dividing by the standard deviation for each feature.

Also, the dataset is preprocessed by one-hot encoding when implementing CNN. This can help the model better understand the distinct classes, as each class is represented by a separate dimension in the vector space. And the loss function we used is categorical cross-entropy, One-hot encoding ensures that the true probability distribution is represented in the appropriate format.

Model Implementation:

K-Nearest Neighbors (KNN):

KNN can adapt to changes in the dataset by simply updating the reference data points. This can be beneficial when dealing with datasets that have complex or unknown data distributions, such as sign language images with diverse hand shapes, orientations, and backgrounds.

The KNN algorithm was applied using the following formula:

$d(x, y) = \text{sqrt}(\Sigma (x_i - y_i)^2)$

where $d(x, y)$ represents the Euclidean distance between data points x and y, and x_i and y_i are their corresponding feature values. The algorithm classifies a new input based on the majority vote of its k nearest neighbors.

We trained the KNN model on the preprocessed dataset, experimented with different values of k, and chose the one that resulted in the highest classification accuracy.

Convolutional Neural Networks (CNN):

Convolutional Neural Networks (CNNs) are a type of artificial neural network commonly used for image recognition and computer vision tasks. CNNs are designed to identify spatial patterns in images and can learn features from data through a process of hierarchical feature extraction. This is particularly useful for sign language recognition, where complex patterns and shapes need to be identified and classified.

The basic building block of a CNN is a convolutional layer, which applies a set of filters to the input image. Each filter extracts a specific feature, such as edges or corners, from the image by computing a convolution operation between the filter and the image pixels. The output of the convolutional layer is a set of feature maps that represent the activations of each filter at different locations in the input image.

After the convolutional layer, a pooling layer is often applied to downsample the feature maps and reduce the dimensionality of the data. The most commonly used pooling operation is max pooling, which takes the maximum value within a small rectangular region of the feature map.

The output of the pooling layer is then fed into one or more fully connected layers, which perform classification based on the learned features. The fully connected layers consist of neurons that are connected to all the neurons in the previous layer. The weights of these neurons are adjusted during training to minimize the classification error.

Parameter tuning is an important part of training a CNN. The most important parameters include the number of filters in each convolutional layer, the size of the filters, the number of neurons in each fully connected layer, and the learning rate. These parameters can be tuned using techniques such as grid search or randomized search to find the optimal combination of hyperparameters.

The structure of CNN we used is as follows, this CNN structure refers to the online script, we adjusted the learning rate, filters, pool size, and image shape. Also, added a new layer to make this model more suitable for our dataset [9]:

Conv2D layer: A 2D convolution layer with 32 filters, each with a kernel size of 5x5, 'Same' padding (input and output have the same dimensions), and ReLU activation function. This is the input layer, and it expects input images of shape 28x28x1 (grayscale images).
Conv2D layer: Another 2D convolution layer with 32 filters, each with a kernel size of 5x5, 'Same' padding, and ReLU activation function.
MaxPool2D layer: A max-pooling layer with a pool size of 2x2, which reduces the spatial dimensions of the input by half.
BatchNormalization layer: This layer normalizes the activations of the previous layer, which can help improve the training speed and model performance.
Dropout layer: A dropout layer with a rate of 0.25, which helps prevent overfitting by randomly setting a fraction of the input units to 0 during training.
Conv2D layer: A 2D convolution layer with 64 filters, each with a kernel size of 3x3, 'Same' padding, and ReLU activation function.
Conv2D layer: Another 2D convolution layer with 64 filters, each with a kernel size of 3x3, 'Same' padding, and ReLU activation function.
MaxPool2D layer: A max-pooling layer with a pool size of 2x2 and stride of 2, which further reduces the spatial dimensions of the input.
BatchNormalization layer: Another batch normalization layer.
Dropout layer: Another dropout layer with a rate of 0.25.
Conv2D layer: A 2D convolution layer with 128 filters, each with a kernel size of 3x3, 'Same' padding, and ReLU activation function.

Conv2D layer: Another 2D convolution layer with 128 filters, each with a kernel size of 3x3, 'Same' padding, and ReLU activation function.

BatchNormalization layer: Another batch normalization layer.
MaxPool2D layer: A max-pooling layer with a pool size of 2x2, stride of 2, and 'same' padding.
Dropout layer: Another dropout layer with a rate of 0.25.
Flatten layer: This layer flattens the input, converting the 2D feature maps into a 1D vector, which can be used as input for the following dense layers.
Dense (fully connected) layer: A fully connected layer with 512 neurons and a ReLU activation function.
Dropout layer: Another dropout layer with a rate of 0.3.
Dense (output) layer: The output layer with 25 neurons (corresponding to the 25 classes) and a softmax activation function, which outputs the probability distribution of the input image belonging to each class.

Logistic Regression:

Logistic Regression is a statistical method for classification, generalized for multi-class problems using one-vs-all or one-vs-one approaches. Logistic regression can provide insight into the importance of different features by examining the learned weights. This can be helpful for understanding which features are most relevant for sign language recognition and potentially guiding the design of more sophisticated models. The model predicts the probability of a given class using the logistic/sigmoid function:

$p(y=1|x) = 1 / (1 + \exp(-z))$

where $z = w^T * x + b$, w represents the weight vector, x is the input feature vector, and b is the bias term.

We trained the Logistic Regression model on the preprocessed dataset, iteratively updating the weights to minimize the cross-entropy loss function.

Model Evaluation:

In the context of our project, a confusion matrix is a valuable tool for evaluating the performance of the Logistic Regression and K-Nearest Neighbors (KNN) algorithms in sign language recognition. A confusion matrix is a table that displays the true positive (TP), false positive (FP), true negative (TN), and false negative (FN) predictions made by a classification model. For multi-class problems like sign language recognition, the confusion matrix is extended to show the predictions and actual labels for each class.

For both Logistic Regression and KNN models, we can create a confusion matrix as follows:

- Create an n x n matrix (where n is the number of classes), with rows representing the actual labels and columns representing the predicted labels.
- Fill in the matrix by counting the occurrences of each combination of actual and predicted labels.
- The diagonal elements of the matrix represent the correct predictions (true positives), while the off-diagonal elements indicate the misclassifications (false positives and false negatives).
- Using the confusion matrix, we can calculate various performance metrics for each model, such as precision, recall, and F1-score, to gain insights into their strengths and weaknesses.

Specifically:

Precision (per class): TP / (TP + FP)
Recall (per class): TP / (TP + FN)
F1-score (per class): 2 * (Precision * Recall) / (Precision + Recall)

These metrics can be averaged across all classes to provide a comprehensive understanding of the model's performance. By comparing the confusion matrices and related metrics for both Logistic

Regression and KNN models, we can evaluate their suitability for sign language recognition and identify areas for improvement.

Datasets

1. Dataset Identification:

Sign Language MNIST[1]: This dataset is an extension of the original MNIST dataset for handwritten digits, but it features images of hand gestures representing the ASL alphabet. Researchers have used this dataset to train various deep learning models, including Convolutional Neural Networks (CNNs) and Long Short-Term Memory (LSTM) networks, for sign language recognition.

2. Literature Related

2.1 RWTH-PHOENIX Weather 2014 (53GB) Continuous Sign Language Benchmark data set with paper "Continuous sign language recognition: Towards large vocabulary statistical recognition systems handling multiple signers" [2]

The article discusses the use of statistical language models (LMs) in automatic sign language recognition (ASLR) systems. The authors propose using class LMs, which group similar signs or glosses together to increase the frequency of their occurrence in the training data, leading to better performance and they evaluate their approach on two publicly available sign language datasets and show improvements over previous best results. They show that this leads to improvements in ASLR performance compared to previous best results. The method could be considered better than other methods that do not use class LMs because it helps to overcome the issue of low frequency of vocabulary entries in SL sentences, thereby improving the consistency and frequency in which specific glosses are seen in context.

2.2 LSA64: A Dataset for Argentinian Sign Language with paper "Sign Language Recognition Based on 3D Convolutional Neural Networks" [3]

This paper proposes a 3D CNN architecture for recognizing 64 classes of gestures from Argentinian Sign Language. The method outperforms traditional methods based on hand-crafted features and other deep learning-based work, achieving 93.9% accuracy. It appears that the proposed 3D CNN architecture performs better than traditional methods and other deep learning-based approaches, achieving higher accuracy in sign language recognition.

2.3 LSA64: A Dataset for Argentinian Sign Language with paper "Sign Language Recognition based on hand and body skeletal data" [4]

The content describes a novel deep learning-based methodology for sign language recognition that relies on hand and body skeletal features extracted from RGB videos without the need for additional equipment such as data gloves. The proposed system uses a four-stream deep neural network consisting of stacked LSTM layers to produce descriptive temporal information from the spatial features. The authors claim that their methodology is superior to other state-of-the-art approaches relying solely on RGB features. They also employ joint-line distances as an alternative spatial skeleton representation, which models the relationship between joints, to significantly improve SLR results. The method is better than traditional RGB-based approaches since it provides highly discriminative skeletal data without additional equipment, thus overcoming the limitations of other approaches that may restrict signer's movements.

2.4 NUS hand posture dataset and American fingerspelling A dataset with paper "A Deep Convolutional Neural Network Approach for Static Hand Gesture Recognition" [5]

The paper proposes a deep learning approach using a convolutional neural network (CNN) architecture to recognize static hand gestures in sign language. The CNN architecture consists of three

convolutional layers, each with a ReLu activation function and maxpooling layer for feature extraction, followed by a softmax output layer and a final fully connected output layer for classification. The paper argues that this method is better than traditional pattern recognition approaches because it automates feature extraction and achieves higher recognition accuracies.

2.5 Face Recognition Using Laplacianfaces [6]

The authors propose a face recognition method called the Laplacianface approach that uses Locality Preserving Projections (LPP) to map face images into a face subspace for analysis. The method aims to preserve local information and detect the essential face manifold structure. The authors compare the Laplacianface approach with Eigenface and Fisherface methods and report that the Laplacianface approach achieves lower error rates in face recognition. The Laplacianface approach has been shown to perform well in various experiments and is an effective method for face recognition in certain scenarios.

3. Related Implementation

3.1 Sign Language MNIST with ANN [7]

This project implemented ANN on sign language recognition and the accuracy of model is 87%. The dataset in this project is the same as we used. Compare to our model, CNN can be more complex to implement and require more computational resources compared to simpler ANNs. However, their ability to exploit spatial structure, learn hierarchical features, and handle variations in input data makes them well-suited for sign language recognition tasks, such as the Sign Language MNIST datase.

3.2 Machine Learning Logistic Regression[8]

This project implemented Logistic Rgression on this topic and oiptimized algorithm with gradient descent, the accuracy of the test set is 93%. The dataset is 27 Class Sign Language Dataset, which has more gestures with different backgrounds. Compare to our model, CNNs are specifically designed to handle images and can exploit the spatial structure of the input data. By using convolutional layers, CNNs can detect local patterns and features in the images, which is beneficial for sign language recognition tasks. In contrast, Logistic Regression treats input features independently, losing spatial information when processing image data. Also, Logistic Regression relies on hand-engineered features or simple pixel values, which might not capture the complexity of the data as effectively.

Results

Logistic Regression:

Precision	0.7278
Recall	0.7002

F1 Score	0.7048

The confusion matrix and classification report are shown in appendix, it can be seen that the model has performed well, with most signs having a precision and recall above 0.70. The F1-score, which is the harmonic mean of precision and recall, is also above 0.70 for most signs.

K-Nearest Neighbors (KNN):

Precision	0.8272
Recall	0.8074
F1 Score	0.8081

We used KNN algorithm for sign language recognition and achieved an precision of 83% on the test dataset. The confusion matrix and classification report are shown in appendix.

Convolutional Neural Network

Precision	0.9768
Recall	0.9745
F1 Score	0.9738

Based on our experiments, the architecture of the model consisted of several layers including convolutional, max pooling, batch normalization, and dropout layers. The model was compiled using the Adam optimizer with a learning rate of 0.01 and a categorical cross-entropy loss function.

The model was trained for 30 epochs on a training dataset and validated on a validation dataset. Early stopping was used to prevent overfitting. The training and validation loss and accuracy were recorded and plotted. The loss curves for both training and validation data decreased while the accuracy curves increased, indicating that the model was learning effectively.

After training, the model was evaluated on a testing dataset. The precision, recall, and F1 score were calculated using the precision_recall_fscore_support function from the scikit-learn library. The precision was 0.9768, recall was 0.9745, and F1 score was 0.9738. These scores indicate that the model performed well in recognizing sign language gestures.

Overall, the results suggest that the CNN model with the architecture used in this study is a suitable approach for sign language recognition.

Discussion

Sign language recognition has been a popular research topic in recent years, and several state-of-the-art (SOTA) methods have been proposed. Here are three of the most commonly used methods:

1. Convolutional Neural Networks (CNNs): CNNs have been widely used for sign language recognition due to their ability to capture spatial features in images. Researchers have used various

CNN architectures, such as ResNet, Inception, and DenseNet, to achieve high accuracy in sign language recognition.

2. Recurrent Neural Networks (RNNs): RNNs have been employed to capture temporal information in sign language videos. Long Short-Term Memory (LSTM) and Gated Recurrent Unit (GRU) are the most commonly used RNN architectures for sign language recognition.

3. 3D Convolutional Neural Networks (3D CNNs): 3D CNNs have been used to capture both spatial and temporal features in sign language videos. They are designed to work with 3D inputs, such as videos or image sequences, and have achieved high accuracy in sign language recognition.

In our project on sign language recognition, we employed three different machine learning algorithms: Convolutional Neural Networks (CNNs), K-Nearest Neighbors (KNN), and Logistic Regression. Upon analyzing the results, we found that CNNs outperformed KNN and Logistic Regression. One of the main reasons for this is that CNNs are designed to capture spatial patterns in image, which is crucial for recognizing sign language gestures. By using hierarchical feature extraction, CNNs can automatically learn relevant features from the data, whereas KNN and Logistic Regression rely on hand-engineered features that may not be optimal for sign language recognition.

Furthermore, CNNs can capture both local and global features, which is important for recognizing the intricate details of sign language gestures. In contrast, KNN and Logistic Regression are limited in their ability to identify these features. While KNN can be effective in some cases, it requires a large amount of data and can be computationally expensive. Logistic Regression, on the other hand, is a linear classifier that may not be able to capture the nonlinear relationships between the input and output in sign language recognition.

Overall, our analysis demonstrates that CNNs are the most effective method for sign language recognition among the algorithms we tested. We found that CNNs were able to achieve higher accuracy and performance compared to KNN and Logistic Regression. From this analysis, we learned the importance of using a machine learning algorithm that is specifically designed for the type of data being analyzed. In the case of sign language recognition, CNNs are well-suited for capturing the spatial and temporal patterns that are unique to this type of data.

Conclusion

In this project, we applied three different machine learning algorithms - Convolutional Neural Networks (CNN), K-Nearest Neighbors (KNN), and Logistic Regression - to classify hand gestures from the Sign Language MNIST dataset. Based on our experiments and analysis, CNN outperformed the other two algorithms in terms of accuracy and overall performance. Given these findings, we recommend using the CNN-based solution for sign language recognition tasks. The advantages of using a CNN include its ability to exploit the spatial structure of the input data, hierarchical feature learning, invariance to translation, robustness to variations, and state-of-the-art performance on image classification tasks.

If we had more data available, we could further improve CNN's performance by training the model on a larger and more diverse dataset. This would help the model to generalize better to new and unseen hand gestures, making it even more robust and accurate.

For a company seeking to run this solution at a high scale, we recommend optimizing the CNN architecture and hyperparameters to ensure the best possible performance while minimizing the computational requirements. This can be achieved through techniques such as architecture search, regularization, and early stopping. Also, Implement hardware acceleration, such as GPUs or dedicated AI chips, to speed up the training and inference processes. This will enable the solution to scale efficiently and handle large volumes of data in real time.

References

[1] TECPERSON. https://www.kaggle.com/datamunge/sign-language-mnist

[2] O. Koller, J. Forster, and H. Ney. Continuous sign language recognition: Towards large vocabulary statistical recognition systems handling multiple signers. Computer Vision and Image Understanding, volume 141, pages 108-125, December 2015.

[3] Neto, G.M.R., Junior, G.B., de Almeida, J.D.S., de Paiva, A.C. (2018). Sign Language Recognition Based on 3D Convolutional Neural Networks. In: Campilho, A., Karray, F., ter Haar Romeny, B. (eds) Image Analysis and Recognition. ICIAR 2018

[4] D. Konstantinidis, K. Dimitropoulos and P. Daras, "SIGN LANGUAGE RECOGNITION BASED ON HAND AND BODY SKELETAL DATA," 2018 - 3DTV-Conference: The True Vision - Capture, Transmission and Display of 3D Video (3DTV-CON), Helsinki, Finland, 2018

[5] Adithya V., Rajesh R., A Deep Convolutional Neural Network Approach for Static Hand Gesture Recognition, Procedia Computer Science, Volume 171, Pages 2353-2361, 2020

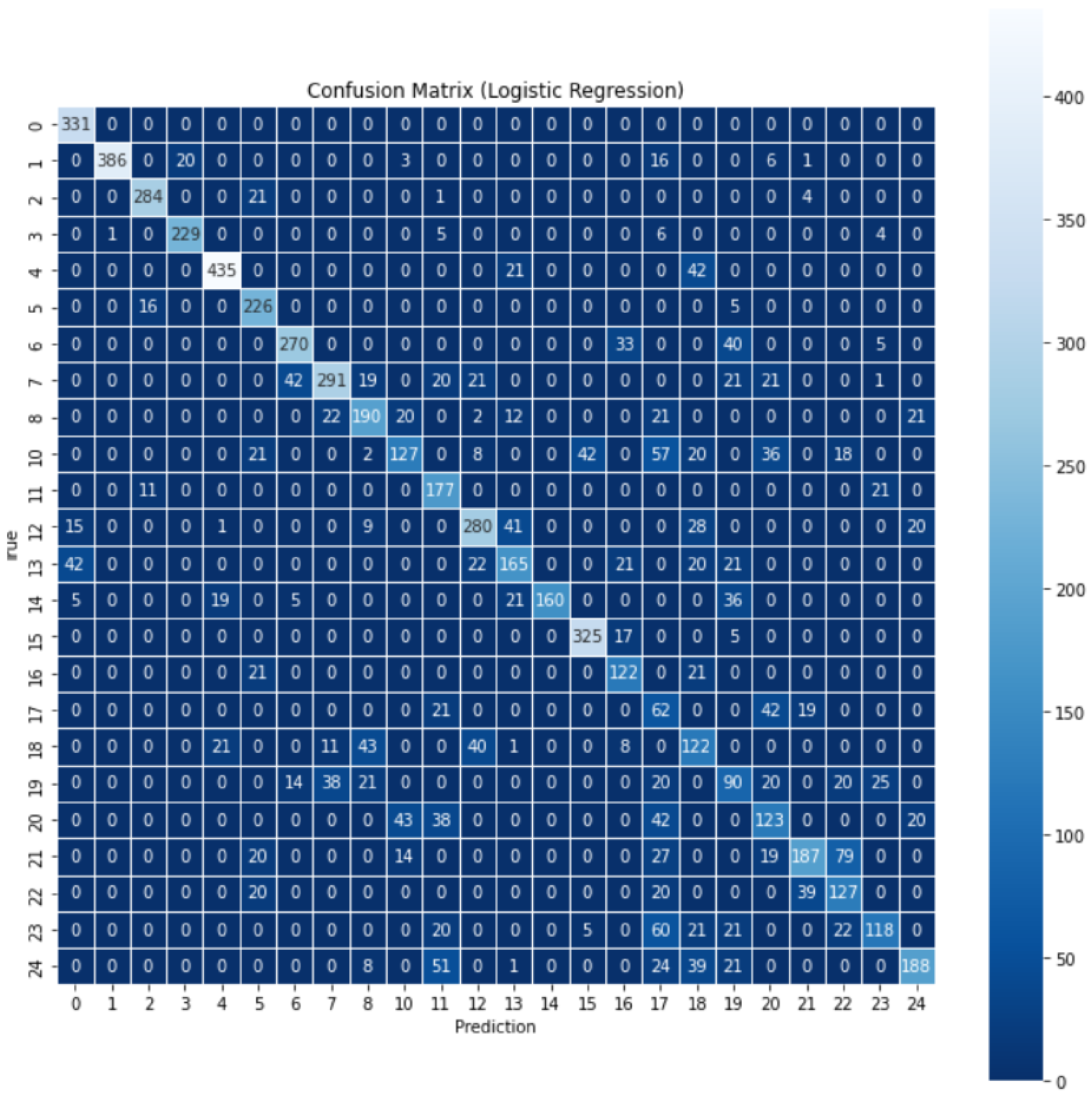
[6] S. Yan, Y. Hu, H. Zhang, P. Niyogi and X. He, "Face Recognition Using Laplacianfaces" in IEEE Transactions on Pattern Analysis & Machine Intelligence, vol. 27, no. 03, pp. 328-340, 2005.

[7] EMAM. https://www.kaggle.com/code/elemam/sign-language-mnist-with-ann

[8] SHIVAM BURNWAL.
https://www.kaggle.com/code/hasanemrebaryank/machine-learning-logistic-regression

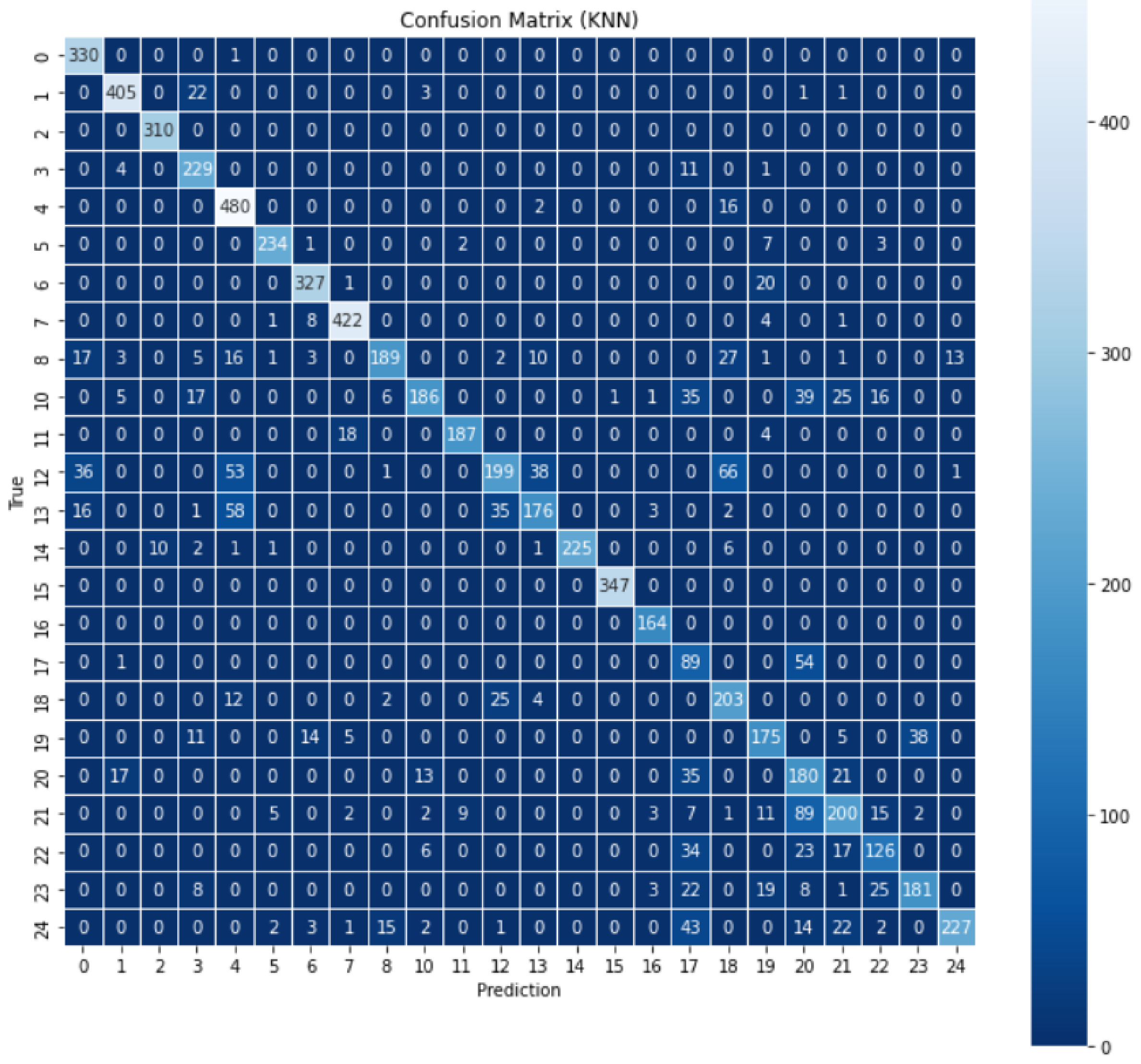
[9] NIKHL https://www.kaggle.com/code/nikhilthakur97/asl-detection-99-accuracy

Appendix



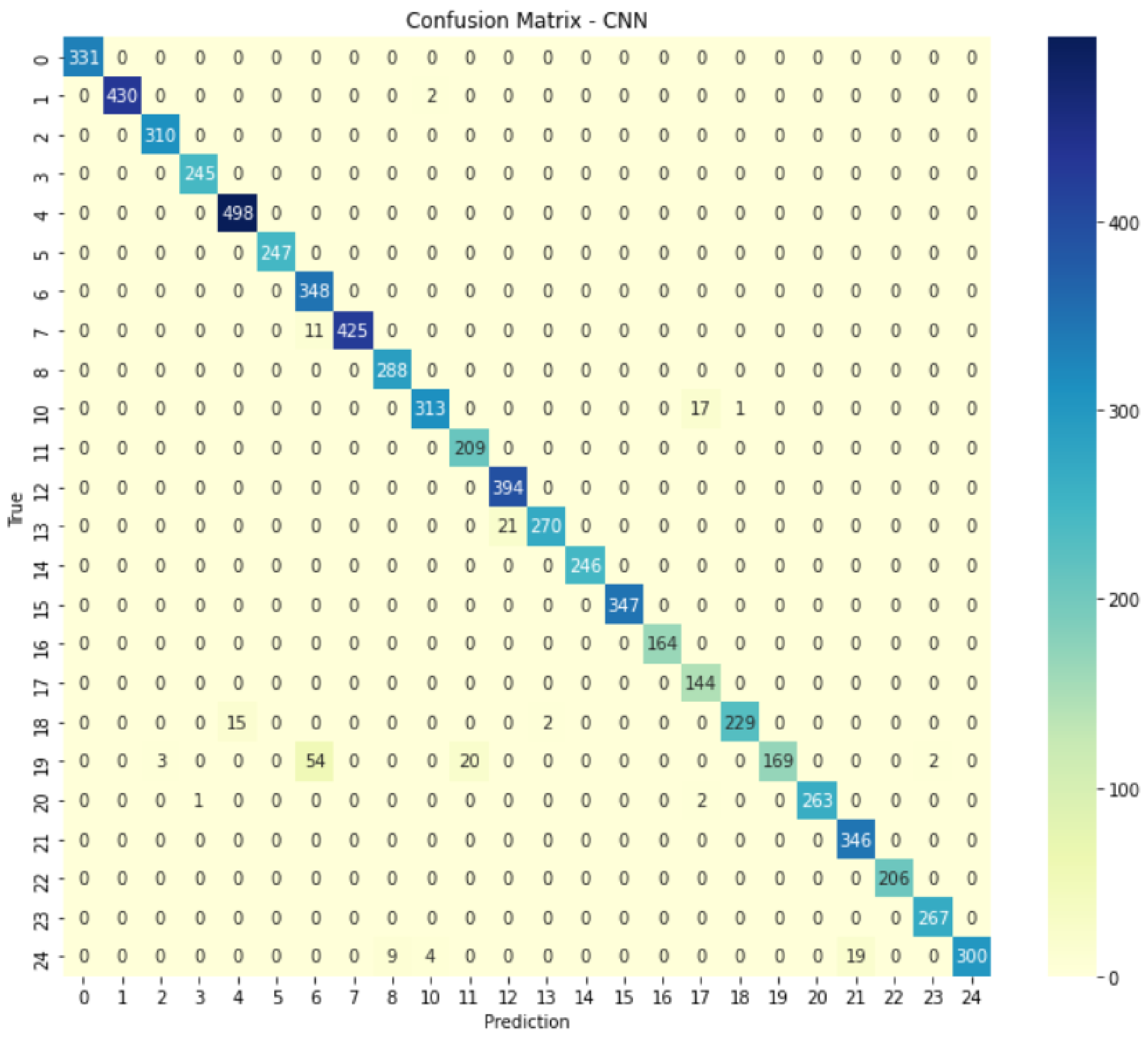
	0	1	2	3	4	5	6	7	8	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24
0	331	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
1	0	386	0	20	0	0	0	0	0	3	0	0	0	0	0	0	16	0	0	6	1	0	0	0
2	0	0	284	0	0	21	0	0	0	0	1	0	0	0	0	0	0	0	0	0	4	0	0	0
3	0	1	0	229	0	0	0	0	0	0	5	0	0	0	0	0	6	0	0	0	0	0	4	0
4	0	0	0	0	435	0	0	0	0	0	0	0	21	0	0	0	0	42	0	0	0	0	0	0
5	0	0	16	0	0	226	0	0	0	0	0	0	0	0	0	0	0	0	5	0	0	0	0	0
6	0	0	0	0	0	0	270	0	0	0	0	0	0	0	0	33	0	0	40	0	0	0	5	0
7	0	0	0	0	0	0	42	291	19	0	20	21	0	0	0	0	0	0	21	21	0	0	1	0
8	0	0	0	0	0	0	0	22	190	20	0	2	12	0	0	0	21	0	0	0	0	0	0	21
10	0	0	0	0	0	21	0	0	2	127	0	8	0	0	42	0	57	20	0	36	0	18	0	0
11	0	0	11	0	0	0	0	0	0	0	177	0	0	0	0	0	0	0	0	0	0	0	21	0
12	15	0	0	0	1	0	0	0	9	0	0	280	41	0	0	0	0	28	0	0	0	0	0	20
13	42	0	0	0	0	0	0	0	0	0	0	22	165	0	0	21	0	20	21	0	0	0	0	0
14	5	0	0	0	19	0	5	0	0	0	0	0	21	160	0	0	0	0	36	0	0	0	0	0
15	0	0	0	0	0	0	0	0	0	0	0	0	0	0	325	17	0	0	5	0	0	0	0	0
16	0	0	0	0	0	21	0	0	0	0	0	0	0	0	0	122	0	21	0	0	0	0	0	0
17	0	0	0	0	0	0	0	0	0	0	21	0	0	0	0	0	62	0	0	42	19	0	0	0
18	0	0	0	0	21	0	0	11	43	0	0	40	1	0	0	8	0	122	0	0	0	0	0	0
19	0	0	0	0	0	0	14	38	21	0	0	0	0	0	0	0	20	0	90	20	0	20	25	0
20	0	0	0	0	0	0	0	0	0	43	38	0	0	0	0	0	42	0	0	123	0	0	0	20
21	0	0	0	0	0	20	0	0	0	14	0	0	0	0	0	0	27	0	0	19	187	79	0	0
22	0	0	0	0	0	20	0	0	0	0	0	0	0	0	0	0	20	0	0	0	39	127	0	0
23	0	0	0	0	0	0	0	0	0	0	20	0	0	0	5	0	60	21	21	0	0	22	118	0
24	0	0	0	0	0	0	0	0	8	0	51	0	1	0	0	0	24	39	21	0	0	0	0	188

Figure 2: Confusion Matrix of Logistic Rgression



True \ Prediction	0	1	2	3	4	5	6	7	8	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24
0	330	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
1	0	405	0	22	0	0	0	0	0	3	0	0	0	0	0	0	0	0	0	1	1	0	0	0
2	0	0	310	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
3	0	4	0	229	0	0	0	0	0	0	0	0	0	0	0	0	11	0	1	0	0	0	0	0
4	0	0	0	0	480	0	0	0	0	0	0	0	2	0	0	0	0	16	0	0	0	0	0	0
5	0	0	0	0	0	234	1	0	0	0	2	0	0	0	0	0	0	0	7	0	0	3	0	0
6	0	0	0	0	0	0	327	1	0	0	0	0	0	0	0	0	0	0	20	0	0	0	0	0
7	0	0	0	0	0	1	8	422	0	0	0	0	0	0	0	0	0	0	4	0	1	0	0	0
8	17	3	0	5	16	1	3	0	189	0	0	2	10	0	0	0	0	27	1	0	1	0	0	13
10	0	5	0	17	0	0	0	0	6	186	0	0	0	0	1	1	35	0	0	39	25	16	0	0
11	0	0	0	0	0	0	0	18	0	0	187	0	0	0	0	0	0	0	4	0	0	0	0	0
12	36	0	0	0	53	0	0	0	1	0	0	199	38	0	0	0	0	66	0	0	0	0	0	1
13	16	0	0	1	58	0	0	0	0	0	0	35	176	0	0	3	0	2	0	0	0	0	0	0
14	0	0	10	2	1	1	0	0	0	0	0	0	1	225	0	0	0	6	0	0	0	0	0	0
15	0	0	0	0	0	0	0	0	0	0	0	0	0	0	347	0	0	0	0	0	0	0	0	0
16	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	164	0	0	0	0	0	0	0	0
17	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	89	0	0	54	0	0	0	0
18	0	0	0	0	12	0	0	0	2	0	0	25	4	0	0	0	0	203	0	0	0	0	0	0
19	0	0	0	11	0	0	14	5	0	0	0	0	0	0	0	0	0	0	175	0	5	0	38	0
20	0	17	0	0	0	0	0	0	0	13	0	0	0	0	0	0	35	0	0	180	21	0	0	0
21	0	0	0	0	0	5	0	2	0	2	9	0	0	0	0	3	7	1	11	89	200	15	2	0
22	0	0	0	0	0	0	0	0	0	6	0	0	0	0	0	0	34	0	0	23	17	126	0	0
23	0	0	0	8	0	0	0	0	0	0	0	0	0	0	0	3	22	0	19	8	1	25	181	0
24	0	0	0	0	0	2	3	1	15	2	0	1	0	0	0	0	43	0	0	14	22	2	0	227

Figure 3: Confusion Matrix of KNN



	0	1	2	3	4	5	6	7	8	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24
0	331	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
1	0	430	0	0	0	0	0	0	0	2	0	0	0	0	0	0	0	0	0	0	0	0	0	0
2	0	0	310	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
3	0	0	0	245	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
4	0	0	0	0	498	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
5	0	0	0	0	0	247	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
6	0	0	0	0	0	0	348	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
7	0	0	0	0	0	0	11	425	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
8	0	0	0	0	0	0	0	0	288	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
10	0	0	0	0	0	0	0	0	0	313	0	0	0	0	0	0	17	1	0	0	0	0	0	0
11	0	0	0	0	0	0	0	0	0	0	209	0	0	0	0	0	0	0	0	0	0	0	0	0
12	0	0	0	0	0	0	0	0	0	0	0	394	0	0	0	0	0	0	0	0	0	0	0	0
13	0	0	0	0	0	0	0	0	0	0	0	21	270	0	0	0	0	0	0	0	0	0	0	0
14	0	0	0	0	0	0	0	0	0	0	0	0	0	246	0	0	0	0	0	0	0	0	0	0
15	0	0	0	0	0	0	0	0	0	0	0	0	0	0	347	0	0	0	0	0	0	0	0	0
16	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	164	0	0	0	0	0	0	0	0
17	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	144	0	0	0	0	0	0	0
18	0	0	0	0	15	0	0	0	0	0	0	0	2	0	0	0	0	229	0	0	0	0	0	0
19	0	0	3	0	0	0	54	0	0	0	20	0	0	0	0	0	0	0	169	0	0	0	2	0
20	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	2	0	0	263	0	0	0	0
21	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	346	0	0	0
22	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	206	0	0
23	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	267	0
24	0	0	0	0	0	0	0	0	9	4	0	0	0	0	0	0	0	0	0	0	19	0	0	300

Figure 4: Confusion Matrix of CNN

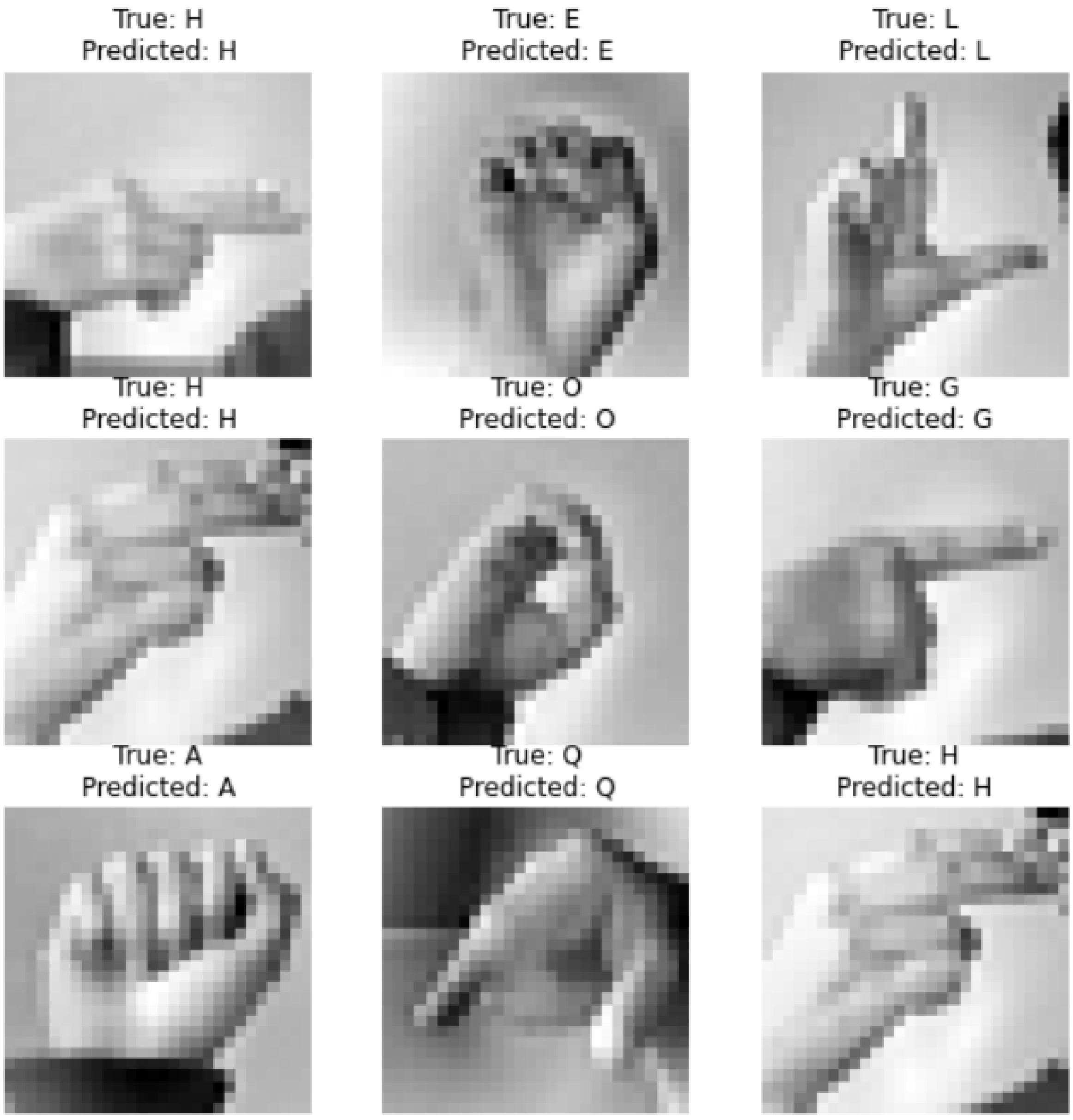


Figure 5: Prediction Example

Github Link: https://github.com/CERKO12/Sign-Language-Recognition.git

Contribution

All group members contributed equally in this project

Yiyang Jiang: write the introduction part, introduce the three methods we used in our model and describe how data collection and preprocessing work, list the model implementation and the model evaluation.

Congkai Sun: Report (Conclusion, Appendix, Revision), Coding

Jinxuan Zhang: Analyzing the dataset and researching related work from papers and Kaggle implementations for the datasets section. Additionally, I worked on writing the result and discussion parts of the report and ensured that all references were properly cited.