Table of Contents

INTRODUCTION

American Sign Language (ASL) is a natural language that is the primary sign language of Deaf communities in the United States and most of Anglophone Canada. ASL is a complete and structured visual language that is communicated by facial expression as well as movements and motions with the hands.

The National Institute on Deafness and other Communications Disorders (NIDCD) indicates that the 200-year-old ASL is a complex language, yet it is the leading minority language in the United States.

A software company, XConnect.ai, wants to build a robust visual recognition algorithm to help the deaf and hard-of-hearing better communicate using computer vision applications. The company would like to implement computer vision in an inexpensive board computer to enable improved and automated translation of applications using their algorithm.

PROBLEM STATEMENT

How can XConnect.ai classify ASL images with a minimum accuracy rate of 90% using a drop-in replacement for MNIST dataset for hand gesture recognition task?

DATASETS

We will be using the open dataset given at Kaggle called Sign Language MNIST – Drop-In Replacement for MNIST for Hand Gesture Recognition Task.

The dataset formatting is similar to the classic MNIST. Each training and test case are represented by 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 are 27,455 cases for training and 7,172 cases for testing. Cases are provided 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 was created by multiple users repeating the gesture against different backgrounds. The Sign Language MNIST data is obtained by extending a small number (only 1704) of the color images included as not cropped around the hand region of interest. An image pipeline based on ImageMagick was used to create a new data. Cropping to hands-only, gray-scaling, and resizing were also used in the process. To enlarge the quantity, more than 50 variations were created. For example: the modification and expansion strategy filters ('Mitchell', 'Robidoux', 'Catrom', 'Spline', 'Hermite'), along with 5% random pixelation, +/- 15% brightness/contrast, and finally 3 degrees rotation.

The following are cropped image montage panels of various users and backgrounds for ASL letters:

Figure 1

DATA WRANGLING AND EDA

In this step, we transformed data from one raw data form into a different format to make it more appropriate and valuable for our analysis. We use summary statistics and visual representations. We do the following:

- Check whether classes are balanced.
- Convert labels into categorical data type.
- Reshape the images.
- Use image augmentation technique to avoid overfitting problem.

CLASSES ARE BALANCED

In our training data, we have all the integers 0 through 25, except 9 and 25. The following histogram of the labels in the training data shows that the classes are distributed almost equally.

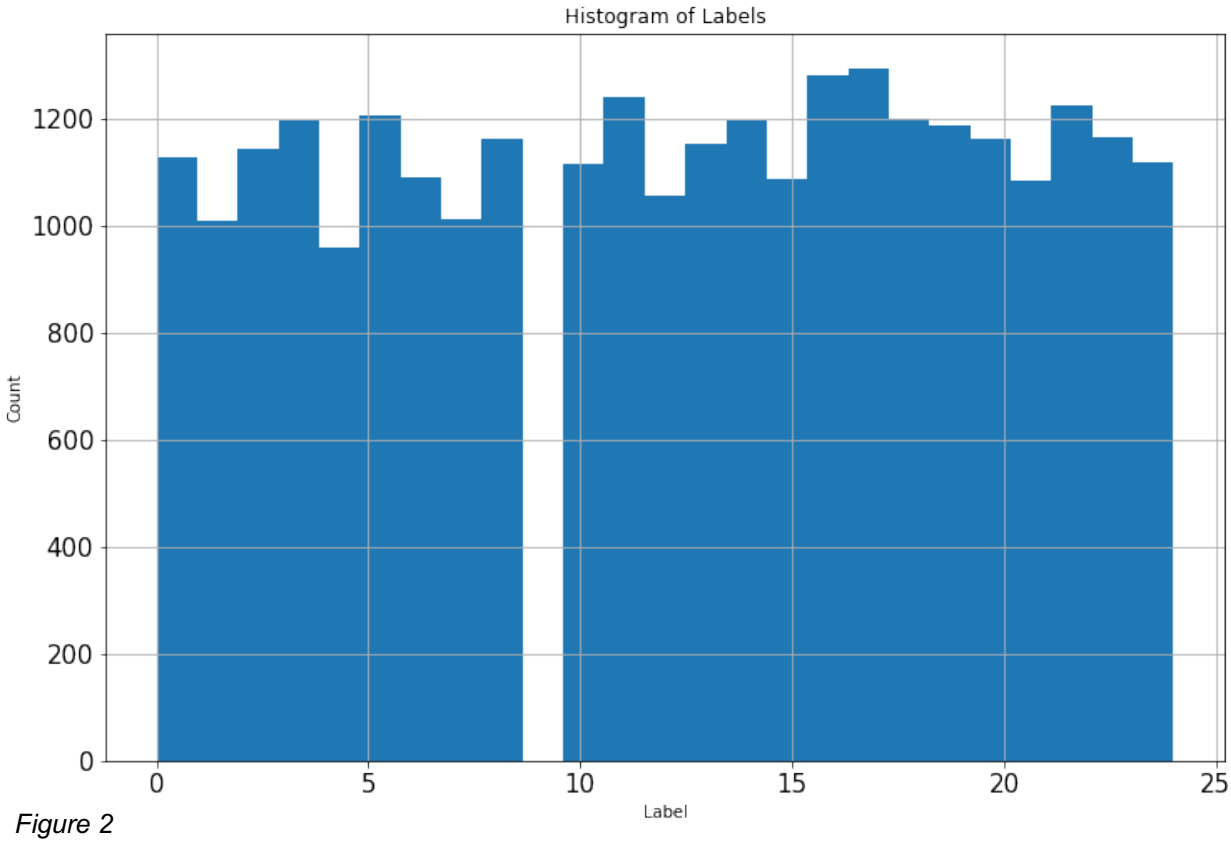

Figure 2

The following dataframe also indicates that the classes are balanced. The number of the cases in each class is approximately 4% of the total number of the cases.

	label	count	label_percent
0	17	1294	4.71
1	16	1279	4.66
2	11	1241	4.52
3	22	1225	4.46
4	5	1204	4.39
5	18	1199	4.37
6	14	1196	4.36
7	3	1196	4.36
8	19	1186	4.32
9	23	1164	4.24
10	8	1162	4.23
11	20	1161	4.23
12	13	1151	4.19
13	2	1144	4.17
14	0	1126	4.10
15	24	1118	4.07
16	10	1114	4.06
17	6	1090	3.97
18	15	1088	3.96
19	21	1082	3.94
20	12	1055	3.84
21	7	1013	3.69
22	1	1010	3.68
23	4	957	3.49

Figure 3

Convert Labels Into Categorical Data Type

We convert the labels into categorical data type using LabelBinarizer(). This process is one hot encoding for the labels.

Reshape The Images

To reduce the effect caused by illumination's differences and make our model faster, we perform a grayscale normalization. In this process, we divide each pixel by 255 which changes the range of pixel intensity values from (0, 255) to (0, 1).

We plot 20 images that are randomly chosen from the training data.

Figure 4

Average Image Of Each Label

We calculate the average pixel intensity value for each pixel of each label. This allows us to see the average image of each label. The following contains the average image of each label in the training data:

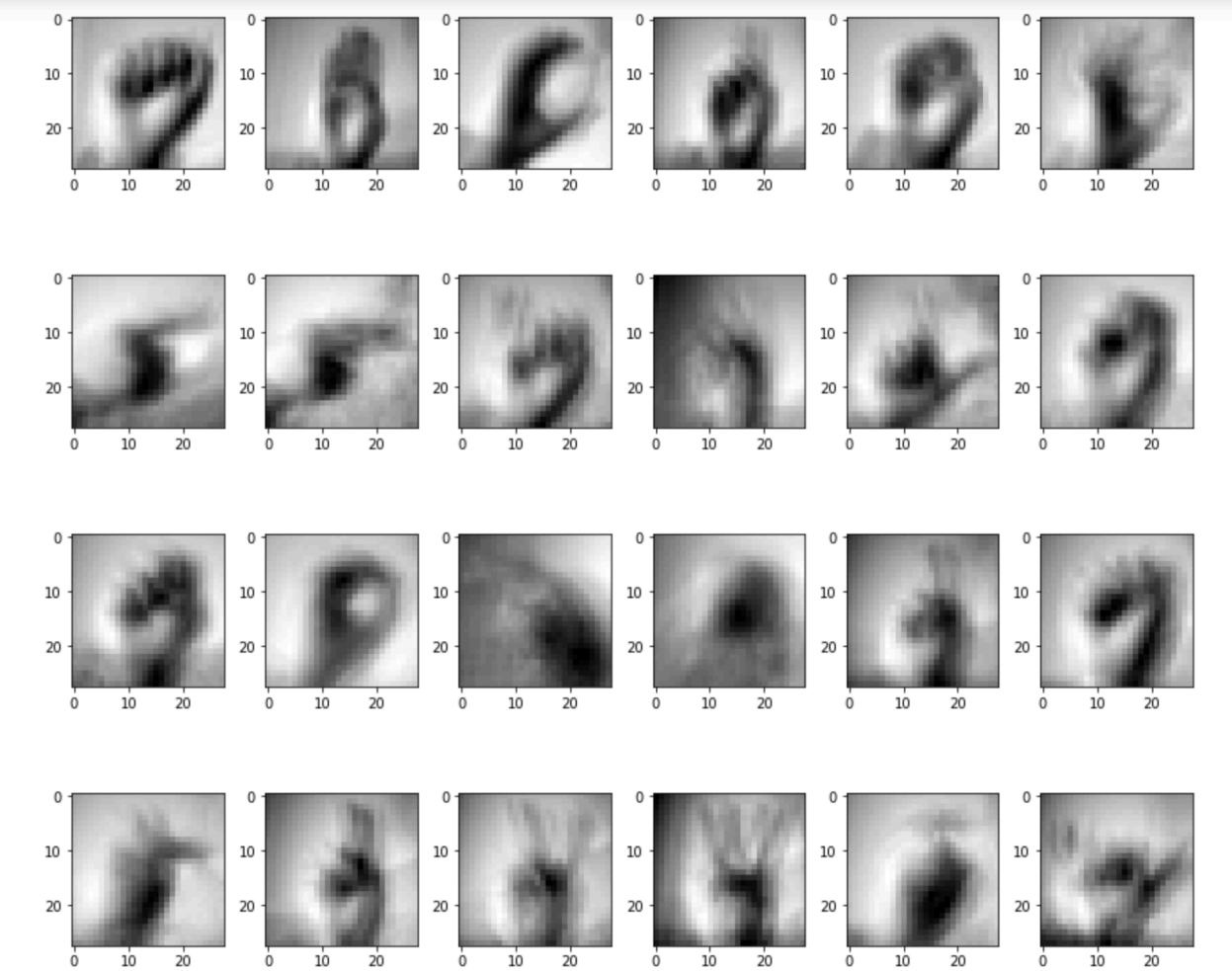

Figure 5

Maximum Variation Of Images

We examine the standard deviation of each pixel for each label in order to compare the variations in images. The images labelled as 16 and 24 have the highest and lowest variations, respectively.

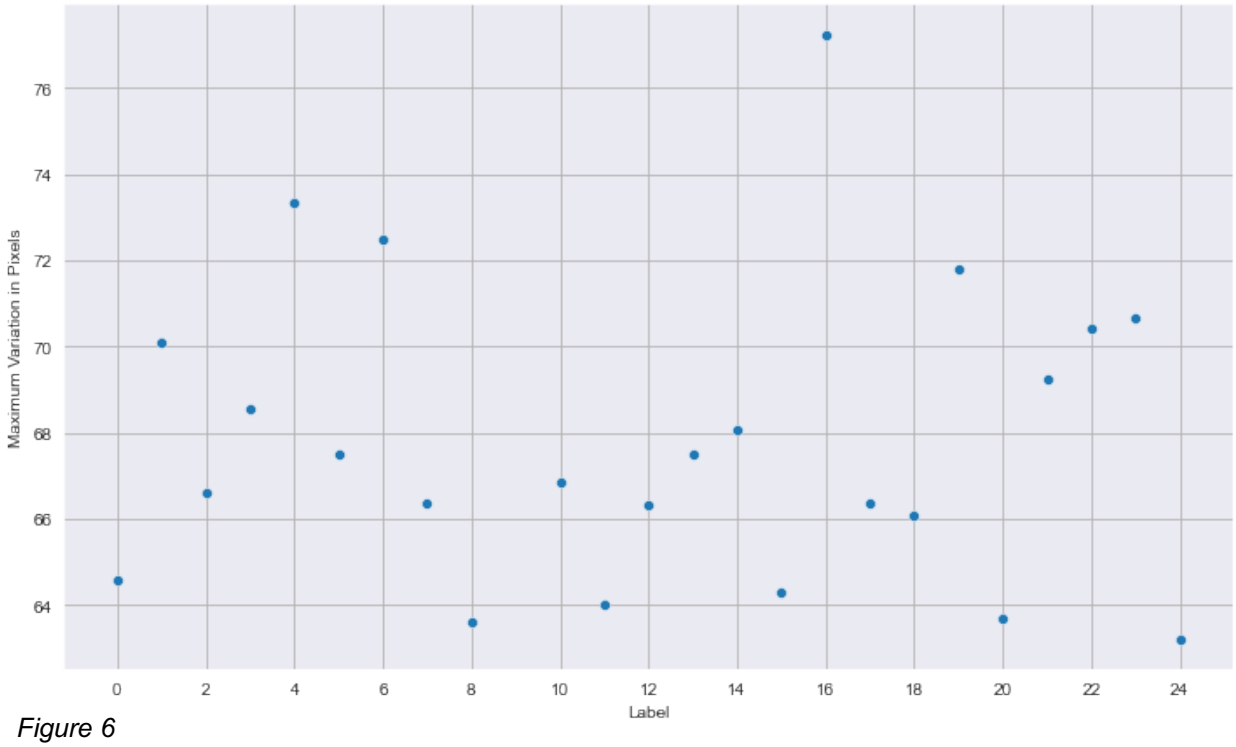

Figure 6

Data Augmentation

We need to avoid the overfitting problem at all cost to make sure that our model is robust. We will use an image augmentation technique to expand our training data set in order to increase the performance of our model by generalizing better, therefore reducing overfitting. We will modify the training data with small transformations to produce variations as we increase the size of the data.

We will use ImageDataGenerator() method from Tensorflow for the task. With this technique, we alter the training data by changing the array representation of the images while keeping the label the same. We will consider some of the well-known augmentations such as grayscales, horizontal flips, vertical flips, random crops, color jitters, translations, and rotations. We specifically used the following:

- Randomly rotate a subset of training images by 10 degrees.
- Randomly zoom by 10% some of the training images.
- Randomly shift images horizontally by 10% of the width.
- Randomly shift images vertically by 10% of the height.

PREPROCESSING AND MODELING

In this step we do the following:

- Data pre-processing.
- Build models with different optimizers.
- Compare the performance of the models.
- Pick the best of the models.

DATA PREPROCESSING

We take the following data pre-processing steps in our project:

- Converting arrays to tensors.
- Performing one hot encoding for the labels.
- Augmenting the data.

BUILD THE BASE MODEL

We build a CNN model to classify images. Our base model consists of the following:

- Conv2D layers with activation function "relu":
  - In the first and the last conv2D layer we set strides=1 and padding="same", which means the input and output will have the same size and the filter will shift by one.
- Batch normalization - to normalize the inputs of each layer in order to fight the internal covariate shift problem.
- Max Pooling with 2×2 pixels applied with a stride of 2 pixels - to calculate the maximum value for each patch of the feature map. This will also reduce the size of each feature map by a factor of 2.
- Dropout, a regularization technique - to prevent neural networks from overfitting.
- Flatten layer - to convert the input data into a long vector in order to pass the input through the artificial neural network to be processed further.
- Dense layers - to connect the neurons of the layer to every neuron of its preceding layer.

DEFINE A CALLBACK OBJECT

Models often benefit from reducing the learning rate by a factor of 5-10 once learning stagnates. We define a callback object that performs actions at various stages of training in order to monitor a quantity and if no improvement is seen for a 'patience' number of epochs, the learning rate is reduced.

Model With Adam Optimizer

We use our base model with Adam optimizer. We tune some of the hyperparameters using keras-tuner. We set the following:

- Batch_size=128
- epochs=10
- objective=val_accuracy
- max_trials=5

We extract the best model from tuner_search object. We visualize the performance of the best model with respect to each epoch:

Figure 7

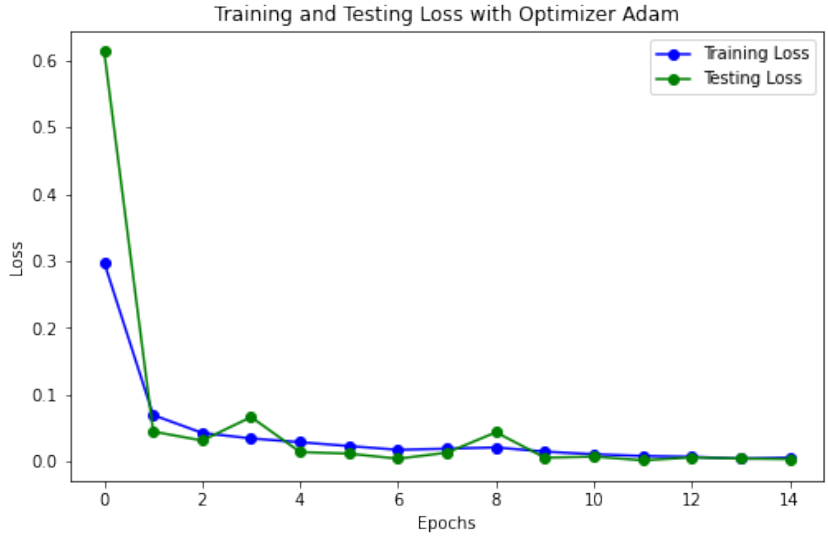

Figure 8

The model performs as follows:

```
------------------------------------------------------------
Accuracy_test: 0.9993028443948689
------------------------------------------------------------
              precision    recall  f1-score   support

     Class 0       1.00      1.00      1.00       331
     Class 1       1.00      1.00      1.00       432
     Class 2       1.00      1.00      1.00       310
     Class 3       1.00      1.00      1.00       245
     Class 4       1.00      1.00      1.00       498
     Class 5       1.00      1.00      1.00       247
     Class 6       1.00      0.99      0.99       348
     Class 7       0.99      1.00      0.99       436
     Class 8       1.00      1.00      1.00       288
    Class 10       1.00      1.00      1.00       331
    Class 11       1.00      1.00      1.00       209
    Class 12       1.00      1.00      1.00       394
    Class 13       1.00      1.00      1.00       291
    Class 14       1.00      1.00      1.00       246
    Class 15       1.00      1.00      1.00       347
    Class 16       1.00      1.00      1.00       164
    Class 17       1.00      1.00      1.00       144
    Class 18       1.00      1.00      1.00       246
    Class 19       1.00      1.00      1.00       248
    Class 20       1.00      1.00      1.00       266
    Class 21       1.00      1.00      1.00       346
    Class 22       1.00      1.00      1.00       206
    Class 23       1.00      1.00      1.00       267
    Class 24       1.00      1.00      1.00       332

    accuracy                           1.00      7172
   macro avg       1.00      1.00      1.00      7172
weighted avg       1.00      1.00      1.00      7172
```

Figure 9

Model With The SGD Optimizer

We use our base model with the SGD optimizer. We tune some of the hyperparameters using keras-tuner. The hyperparameters are same as in the model with Adam optimizer. We extract the best model from tuner_search object. The learning curves of the best model are given below:

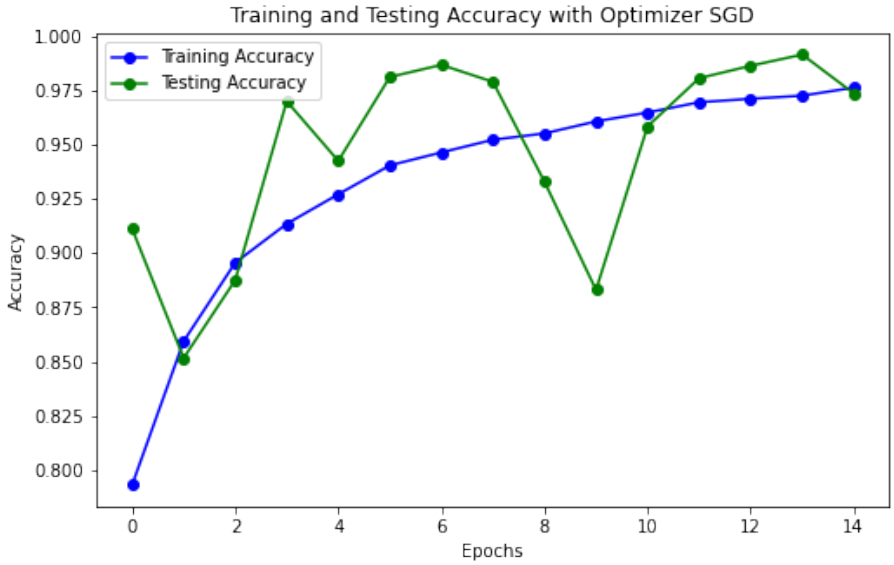

Figure 10

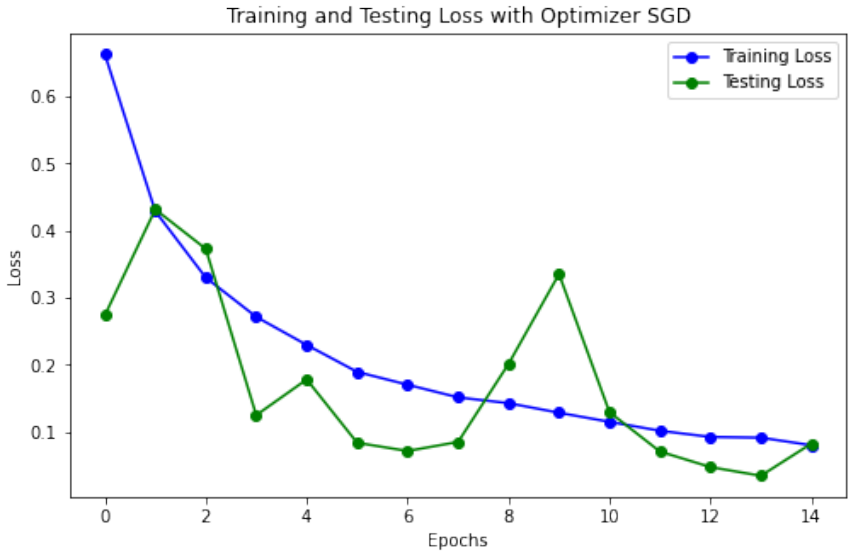

Figure 11

The performance of the model is given below:

```
------------------------------------------------------------
Accuracy_test: 0.9735080870050196
------------------------------------------------------------
              precision    recall  f1-score   support

     Class 0       0.99      1.00      1.00       331
     Class 1       1.00      1.00      1.00       432
     Class 2       1.00      1.00      1.00       310
     Class 3       1.00      0.97      0.98       245
     Class 4       0.96      0.92      0.94       498
     Class 5       1.00      1.00      1.00       247
     Class 6       1.00      0.92      0.96       348
     Class 7       1.00      1.00      1.00       436
     Class 8       1.00      1.00      1.00       288
    Class 10       0.97      1.00      0.99       331
    Class 11       1.00      1.00      1.00       209
    Class 12       0.94      0.85      0.89       394
    Class 13       0.83      1.00      0.91       291
    Class 14       1.00      0.97      0.98       246
    Class 15       0.97      1.00      0.99       347
    Class 16       1.00      1.00      1.00       164
    Class 17       0.94      0.92      0.93       144
    Class 18       0.88      1.00      0.93       246
    Class 19       1.00      1.00      1.00       248
    Class 20       1.00      1.00      1.00       266
    Class 21       0.94      1.00      0.97       346
    Class 22       1.00      1.00      1.00       206
    Class 23       1.00      0.99      1.00       267
    Class 24       1.00      0.91      0.95       332

    accuracy                           0.97      7172
   macro avg       0.98      0.98      0.98      7172
weighted avg       0.98      0.97      0.97      7172
```

Figure 12

Model With The AdaGrad Optimizer

We use our base model with the AdaGrad optimizer. We tune some of the hyperparameters using keras-tuner. The hyperparameters are the same as in the model with Adam optimizer. We extract the best model from tuner_search object. The learning curves of the best model are given below:

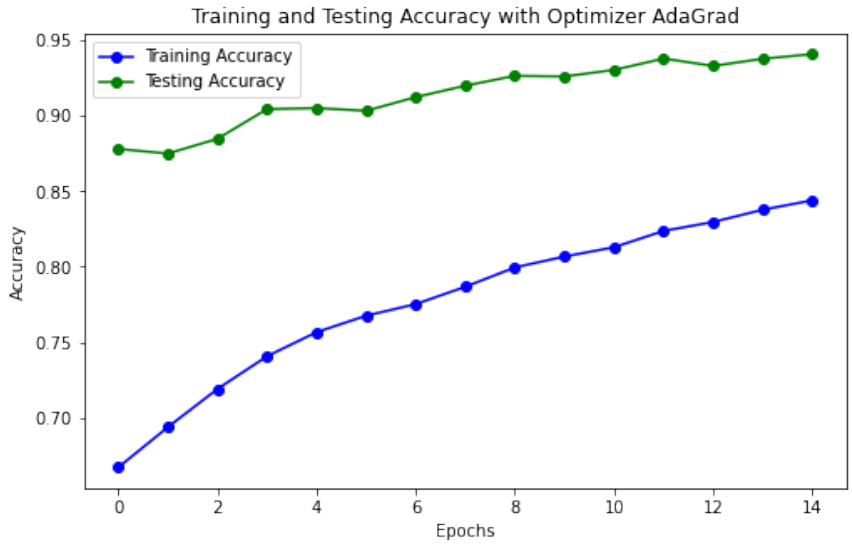

Figure 13

Figure 14

The performance of the model is given below:

```
------------------------------------------------------------
Accuracy_test: 0.9407417735638595
------------------------------------------------------------
              precision    recall  f1-score   support

     Class 0       0.98      1.00      0.99       331
     Class 1       1.00      1.00      1.00       432
     Class 2       1.00      0.98      0.99       310
     Class 3       0.90      1.00      0.95       245
     Class 4       0.90      0.97      0.93       498
     Class 5       0.98      0.97      0.97       247
     Class 6       0.86      0.95      0.90       348
     Class 7       0.95      0.93      0.94       436
     Class 8       0.94      1.00      0.97       288
    Class 10       1.00      0.84      0.91       331
    Class 11       0.92      1.00      0.96       209
    Class 12       0.94      0.90      0.92       394
    Class 13       0.84      0.90      0.87       291
    Class 14       1.00      0.93      0.96       246
    Class 15       1.00      1.00      1.00       347
    Class 16       0.99      1.00      1.00       164
    Class 17       0.83      1.00      0.91       144
    Class 18       0.91      0.80      0.85       246
    Class 19       1.00      0.74      0.85       248
    Class 20       0.91      1.00      0.95       266
    Class 21       0.92      0.90      0.91       346
    Class 22       0.88      0.92      0.90       206
    Class 23       0.93      0.99      0.96       267
    Class 24       1.00      0.88      0.94       332

    accuracy                           0.94      7172
   macro avg       0.94      0.94      0.94      7172
weighted avg       0.94      0.94      0.94      7172
```

Figure 12

Figure 15

FINAL MODEL SELECTION

After analyzing multiple models, we observe the following:

- The model with AdaGrad optimizer has approximately 94% accuracy score. This is the lowest score among the 3 models we tried.
- The model with SGD optimizer has approximately 97% accuracy score. This is the second-best score.
- The model with Adam optimizer has approximately 99% accuracy score, which is the best. This model also has the highest score for precision, recall, and f1 across all classes.

CRITIQUES OF THE MODEL

WHAT IS GOOD ABOUT THE MODEL?

The accuracy score is very high in general for classification models. The model is also doing well with respect to other metrics such as precision, recall, and f1 score (see previous section on Models).

WHAT IS BAD ABOUT THE MODEL?

The dataset contains images of a very small size 28x28 gray level image with the same hand direction and illumination. Therefore, it may not be generalizable and practical for applications.

FUTURE PROJECTS

We would like to increase the number of images gradually; we want to double, triple, and quadruple the number of images and compare the performance of the model with our project at each case. This will allow us to see how much our model depends on the number of images and whether the model is viable on a larger scale.

We would also like to consider other sign languages as well. This will allow us to see whether our model is generalizable.