Computer Vision to Translate Sign Language

Introduction

There are over 500 thousand people who live in the United States in which their native language is that of American Sign Language. The issue however is that people who use ASL as there only way to communicate struggle with holding conversations with people who do not possess the understanding of Sign Language. This project I plan to provide an overview of data from alphabetical signals in sign language, including data preparation, exploratory data analysis, modeling, and evaluation to build a computer vision model to translate sign language with the highest accuracy possible.

Dataset:

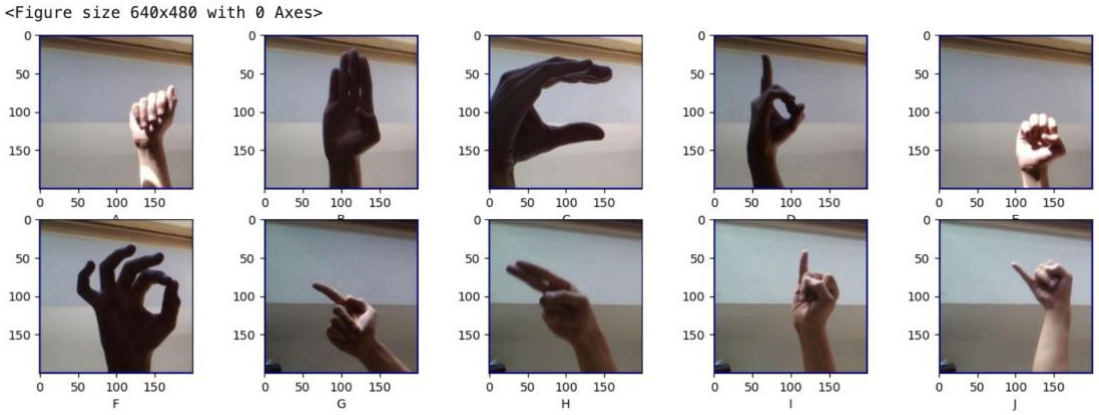
The ASL dataset is available on Kaggle at https://www.kaggle.com/datasets/grassknoted/asl-alphabet and consists of thousands of images in .jpg format for each alphabetical letter in American Sign Language. There was a modification to this dataset in which only ten letters were used A-J each containing 3000 images in a separate folder named the corresponding letter this is used so that training time will be lowered however due to the even distribution of the data these models could be trained on all letters of the alphabet and should maintain the same level of accuracy.

Data Preparation:

The first step in data mining on the ASL dataset is to prepare the data. This involves cleaning and transforming the data to ensure that it is in a format that can be easily analyzed. Once the dataset has been downloaded, it is important to check for any wrong images being labeled so that are data is clean and consistent for analysis to do this I ran code to check what each letter were in the dataset and the number of images for each letter as shown in the figure below.

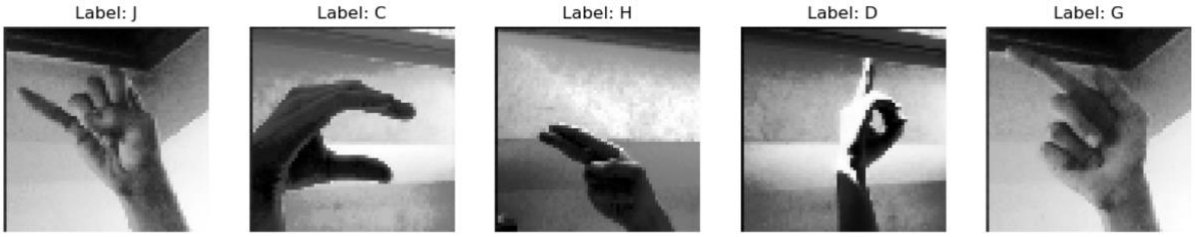
```
There are 10 ASL letters in this dataset.
The letters are that are in the dataset are: ['A', 'B', 'C', 'D', 'E', 'F', 'G', 'H', 'I', 'J'] .

No. of images present in A: 3000
No. of images present in B: 3000
No. of images present in C: 3000
No. of images present in D: 3000
No. of images present in E: 3000
No. of images present in F: 3000
No. of images present in G: 3000
No. of images present in H: 3000
No. of images present in I: 3000
No. of images present in J: 3000
```

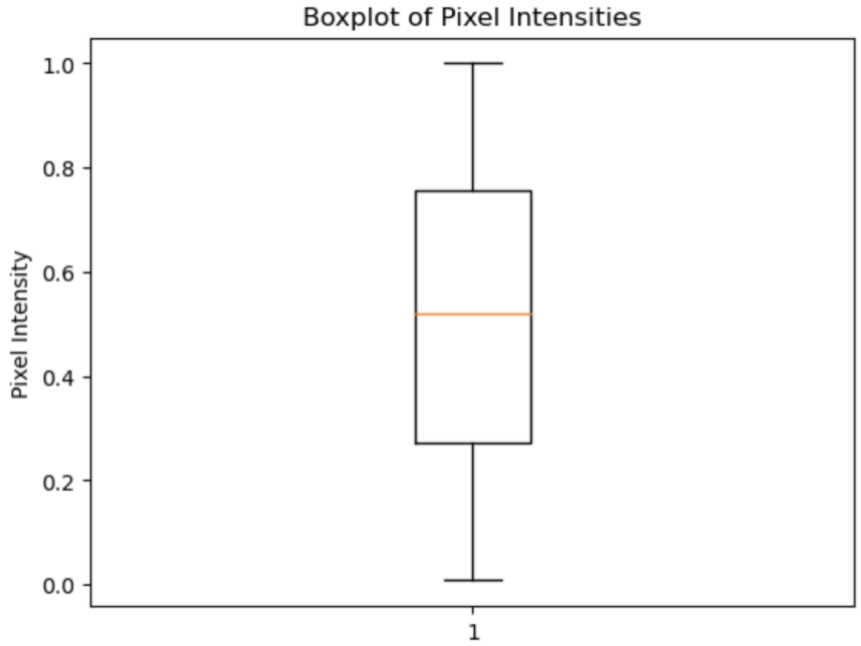
I also printed out sample images to verify the images were loading properly.

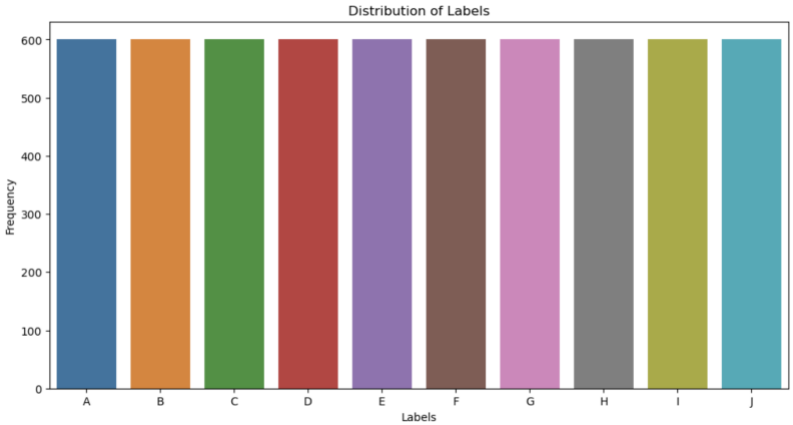


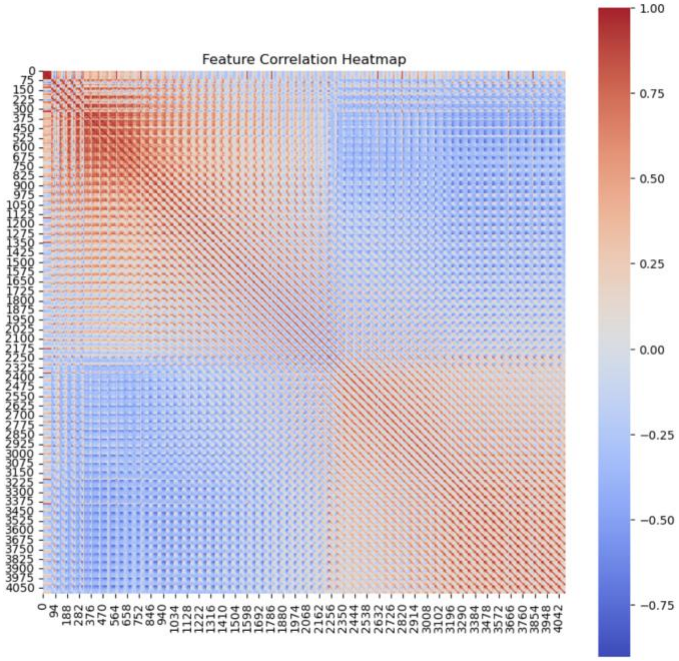
Exploratory Data Analysis:

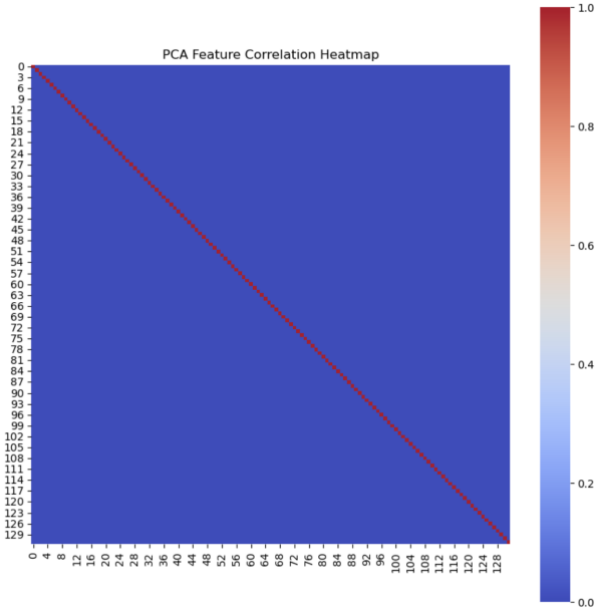
After the data is prepared, the next step is to perform exploratory data analysis (EDA). EDA involves visualizing, summarizing, and augmenting the data to gain insights into the relationships between the variables and prepare it for the models. For the ASL dataset, I augmented the images into grayscale, created a distribution between each class of images, and lastly observed the number of pixels in the augmented images. This is so that I can help identify any patterns or relationships that may exist in the data and guide the selection of appropriate modeling techniques. I also chose to use PCA to remove the high correlation that I had in my data. Below contains images of the augmented images, a boxplot containing the pixel intensities and the distribution of the labels and the feature correlation heatmap for both original and PCA version of the dataset.






Distribution of Labels
Frequency
Labels
A
B
C
D
E
F
G
H
I
J


Feature Correlation Heatmap


PCA Feature Correlation Heatmap

Modeling and Training

The next step is to select a modeling technique. I planned to originally use three algorithms for the ASL dataset, including k-means, neural networks, and support vector machines. This goal however changed due to bad evaluation of k-means, so I included an additional algorithm Random Forest to train upon. Each of these models were given intense tuning of their hyperparameters listed below are the models are hyperparameters tuned to gain the best results possible for each model below are each of the hyperparameters for each of the algorithms that I tuned.

K-Means Hyperparameters

- Num_clusters - hyperparameter represents the number of clusters to be formed, as well as the number of centroids to be generated.
- random state: The random state is the seed used by the random number generator in the K-Means algorithm. It affects the initialization of the centroids, which can impact the final clustering results.

Neural Networks

- Epochs - number of complete passes through the entire dataset during the training process of a neural network
- Dropout - a regularization technique used to prevent overfitting in neural network.
- Batch size - is a hyperparameter that determines the number of samples used for each update during training.

SVM Hyperparameters

- C (Cost or Regularization parameter): This parameter controls the trade-off between maximizing the margin and minimizing the classification error. A smaller value of C creates a wider margin, allowing some misclassifications, which may result in a more

generalized model. A larger value of C, on the other hand, aims for a smaller margin with fewer or no misclassifications, potentially leading to overfitting.

- Kernel: The kernel function is used to transform the input data into a higher-dimensional space, enabling SVM to handle non-linear classification problems. Common kernel functions include linear, polynomial, radial basis function (RBF), and sigmoid. Choosing the right kernel function is crucial, as it can significantly impact the performance of the SVM model.
- Degree (for polynomial kernel): This parameter determines the degree of the polynomial kernel function. A higher degree increases the complexity of the model, making it more prone to overfitting, while a lower degree may result in underfitting.
- Gamma (for RBF and polynomial kernels): Gamma controls the shape of the decision boundary in the transformed feature space. A small value of gamma leads to a more flexible decision boundary, while a large value results in a more rigid decision boundary. The optimal value of gamma depends on the specific problem and dataset.
- Coefficient (for polynomial and sigmoid kernels): This parameter, often denoted as 'coef0', controls the independent term in the kernel function. It can influence the flexibility of the decision boundary, especially for non-linear kernels.

Random Forest Hyperparameters

- n_estimators - refers to the number of decision trees to be constructed in the forest. Increasing the number of trees can improve accuracy but may lead to longer training times and increased memory usage.
- max_depth - is the maximum depth of the decision trees in the forest. A higher value may increase the complexity of the model, which can lead to overfitting.

- min_samples_split - is the minimum number of samples required to split an internal node. Increasing this parameter can prevent overfitting but may lead to underfitting if the value is set too high.
- min_samples_leaf - is the minimum number of samples required to be at a leaf node. A higher value can help prevent overfitting, but it may result in a simpler model that may not capture the full complexity of the data.
- max_features - is the maximum number of features to consider when looking for the best split. Reducing this parameter can help prevent overfitting, but it may also reduce the model's ability to capture the full complexity of the data.

Evaluation and Results

SVM

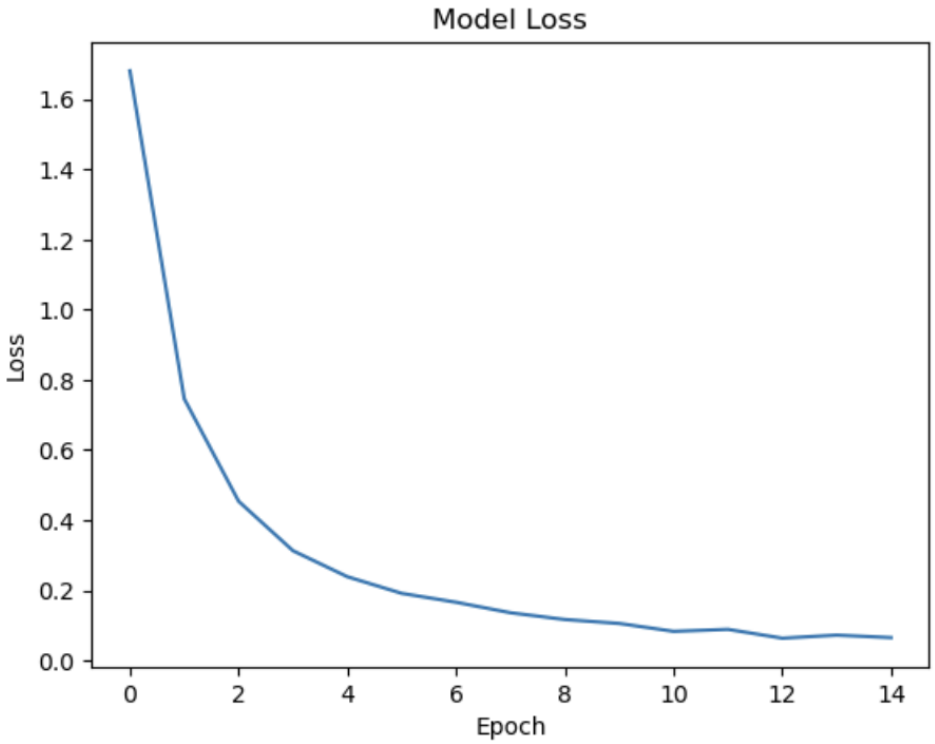
For my dataset the most optimal Support Vector Machine Model uses the data that has gone through PCA and has the following hyperparamters of either C=10 with a kernel of rbf, a gamma score of 0.01 or C=10 with a kernel of poly, degree of 3. To get the highest accuracy score of 96%

K-Means

For my dataset the most optimal K-Means Model uses the data that has gone through PCA and has the following hyperparameters, clusters = 8, and random state = 456. To get the highest accuracy score of 19%

CNN

For my dataset the most optimal Neural Network Model uses the data that has not gone through PCA and has the following num_filters = (64, 128), epochs = 15, dropout_rate = 0.6, dense_units = 256, batch_size = 32To get the highest accuracy score of 96%. Attached below is loss graph



Random Forest

For my dataset the most optimal Random Forest Model uses the data that has not gone through PCA and has the following hyperparamters n_estimators = 500, min_samples_split = 2,

min_samples_leaf = 1, max_features = 'auto', max_depth = 20To get the highest accuracy score of 93%

FINAL CONCLUSION:

The outcome of my models and evaluations shows that the best two models were SVM on PCA data and CNN on base data. SVM does train a lot faster than my other data however I will note that Neural Networks can always be improved upon so with more time and tuning there is a chance that CNN could perform even better while SVM is unlikely to gain that much better of a performance.

REFLECTION:

If I were to redo this project from scratch the first thing that I would have done would have completely scrapped k-means as I spent a lot of time trying to make it work with my data to no avail as shown by the accuracy of the best model I made. I do wish to implement this on the full-scale dataset as well as use real time object detection to create a live translation through video feed.