

MNIST Sign Language Modeling with CNN and SVC

Magnus Aghe
Elijah C Walker
ANA675

Purpose

- The project aims to explore, preprocess, and analyze the Sign Language MNIST dataset.
- Develop a Neural Networking and Machine Learning model to accurately classify American Sign Language (ASL) letters.
- Compare the performance of Convolutional Neural Networks (CNN) and Support Vector Classifier (SVC) in classifying the ASL letters.
- A robust visual recognition algorithm like this could help the deaf and hard-of-hearing better communicate using computer vision applications.



Dataset

- The dataset used is the Sign Language MNIST dataset.
 - <https://www.kaggle.com/datasets/datamunge/sign-language-mnist/data>
- It consists of images of hand gestures representing the letters in the ASL.
- Each image is labeled with the corresponding letter that the hand gesture represents.
- The dataset was inspired by the Fashion-MNIST 2 and the machine learning pipeline for gestures.



Exploratory Data Analysis

- The dataset consists of two downloads from Kaggle.com
 - sign_mnist_train.csv
 - 27,455 Observations
 - sign_mnist_test.csv
 - 7,172 Observations
 - Both datasets contain 785 features (pixel values)
 - 24 labels, one for each letter of the alphabet with 0 being A, 23 being Y
 - Letters J and Z are excluded from the dataset because they require motion
 - Largely pre-processed and well-maintained (typical of MNIST datasets)
 - A few NaN values, otherwise no missing or inconsistent data

Baseline Class Images



Dataset Image Preview

```
1 index = np.random.choice(np.arange(len(X_train)), 24, replace=False) # 24 indices
2 figure, axes = plt.subplots(nrows=4, ncols=6, figsize=(12, 7))
3
4 for item in zip(axes.ravel(), X_train[index], train_label[index]):
5     axes, image, target = item
6     axes.imshow(image, cmap=plt.cm.gray_r)
7     axes.set_xticks([]) # remove x-axis tick marks
8     axes.set_yticks([]) # remove y-axis tick marks
9     axes.set_title(target)
10
11 plt.tight_layout()
```



Class Distribution (Training Data)

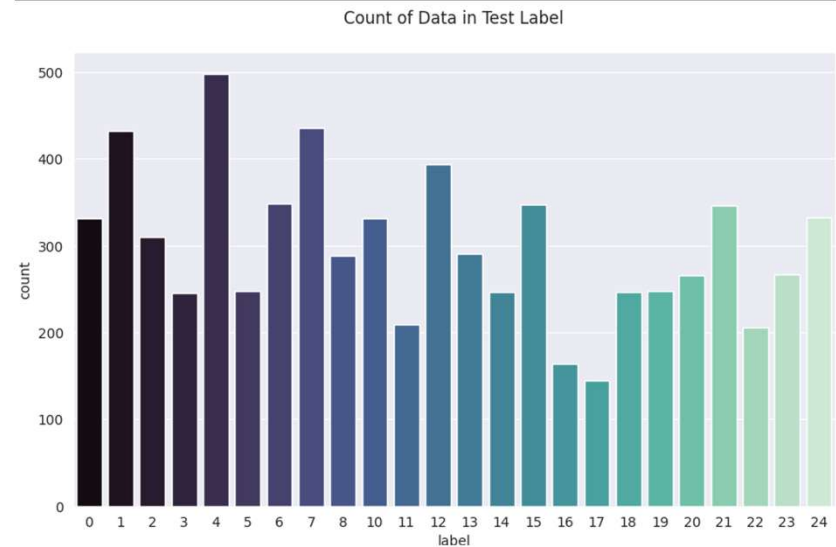
- Relatively normal distribution, no class dominance apparent




Class Distribution (Test Data)

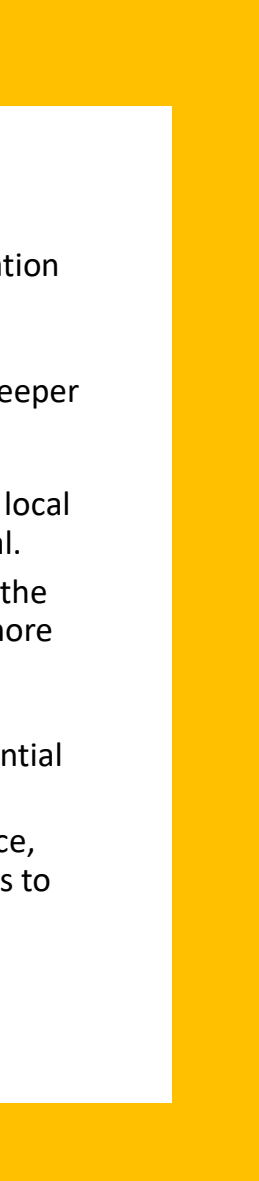
- Much less normally distributed than the training data. This could be more apparent or scaled differently due to less observations being present in the testing dataset.

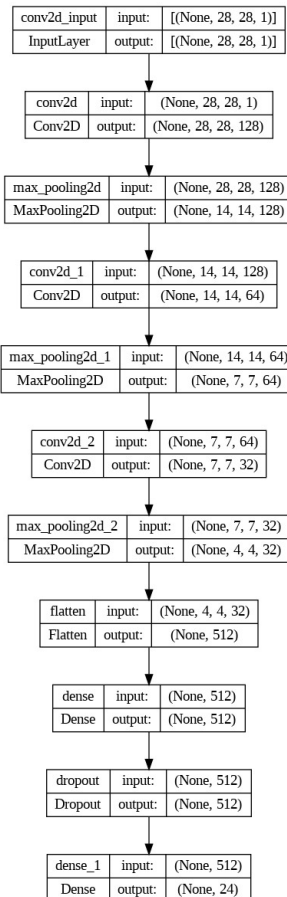
```
1 plt.figure(figsize = (10,6))
2 plt.title("Count of Data in Test Label\n", size = 12)
3
4 sns.countplot(x = test_label, palette='mako');
```





Convolutional Neural Network

- CNNs are particularly effective for image classification tasks for several key reasons:
 - Hierarchical Feature Learning – Lower layers capture basic textures and patterns, while deeper layers can recognize semantics and complex structures. Using convolutional layers, edge detection, textures, and awareness of other local features improves their recognition potential.
 - Parameter sharing – Shared weights reduce the number of parameters, making the model more generalized.
 - Pooling Layers – reduction of the spatial dimensions of the input. Only the most essential information is retained during analysis.
 - Several other reasons – Translation Invariance, Adaptability, End to End Training, Robustness to Variability
- 



```

[26] model=Sequential()
model.add(Conv2D(128,kernel_size=(5,5),
                strides=1,padding='same',activation='relu',input_shape=(28,28,1)))
model.add(MaxPooling2D(pool_size=(3,3),strides=2,padding='same'))
model.add(Conv2D(64,kernel_size=(2,2),
                strides=1,activation='relu',padding='same'))
model.add(MaxPooling2D((2,2),2,padding='same'))
model.add(Conv2D(32,kernel_size=(2,2),
                strides=1,activation='relu',padding='same'))
model.add(MaxPooling2D((2,2),2,padding='same'))
model.add(Flatten())

[27] model.add(Dense(units=512,activation='relu'))
model.add(Dropout(rate=0.25))
model.add(Dense(units=24,activation='softmax'))
model.summary()

```

Model: "sequential"

Layer (type)	Output Shape	Param #
conv2d (Conv2D)	(None, 28, 28, 128)	3328
max_pooling2d (MaxPooling2D)	(None, 14, 14, 128)	0
conv2d_1 (Conv2D)	(None, 14, 14, 64)	32832
max_pooling2d_1 (MaxPooling2D)	(None, 7, 7, 64)	0
conv2d_2 (Conv2D)	(None, 7, 7, 32)	8224
max_pooling2d_2 (MaxPooling2D)	(None, 4, 4, 32)	0
flatten (Flatten)	(None, 512)	0
dense (Dense)	(None, 512)	262656
dropout (Dropout)	(None, 512)	0
dense_1 (Dense)	(None, 24)	12312

Total params: 319352 (1.22 MB)
 Trainable params: 319352 (1.22 MB)
 Non-trainable params: 0 (0.00 Byte)

CNN Model Summary and Layers

CNN Training and Testing

- The CNN Model went from 35% training and 76% validation accuracy on the first training cycle (epoch) to 94% training and 99% validation accuracy by the third epoch.
- Accuracy for training and validation achieved and loosely held 100% accuracy after 25 epochs. It ended at 99.97% training and 100% validation accuracy after 35 epochs.
- The final accuracy on the test data after 35 epochs, achieved a respectable 92.43% accuracy and 0.4970 loss.

```
Epoch 35/35  
110/110 [=====] - 1s 13ms/step - loss: 7.7902e-04 - accuracy: 0.9997 - val_loss: 1.8150e-04 - val_accuracy: 1.0000
```

CNN Evaluation and Results

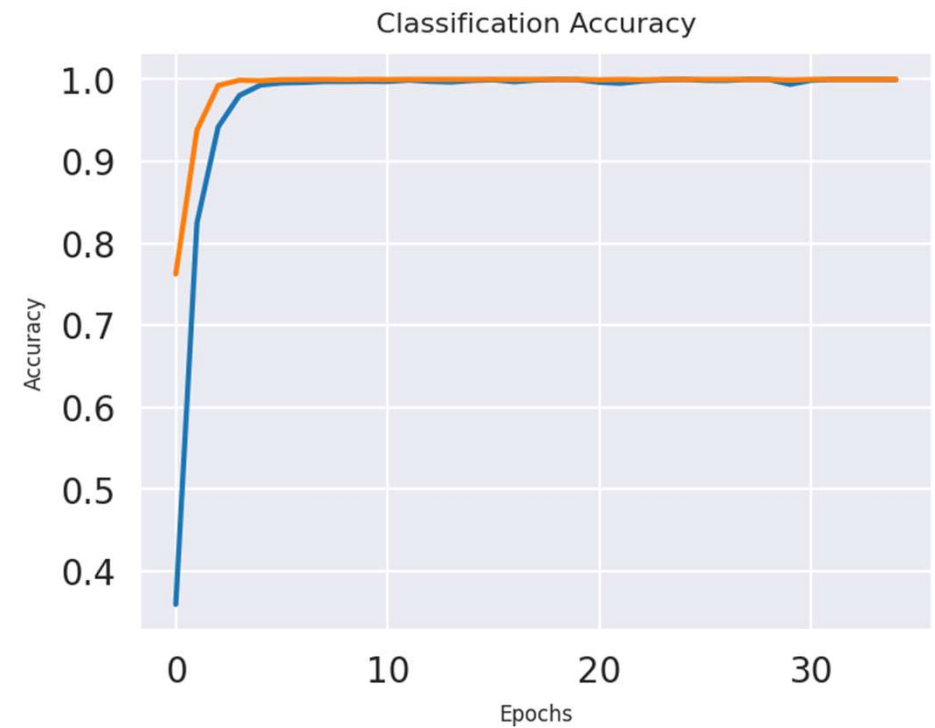
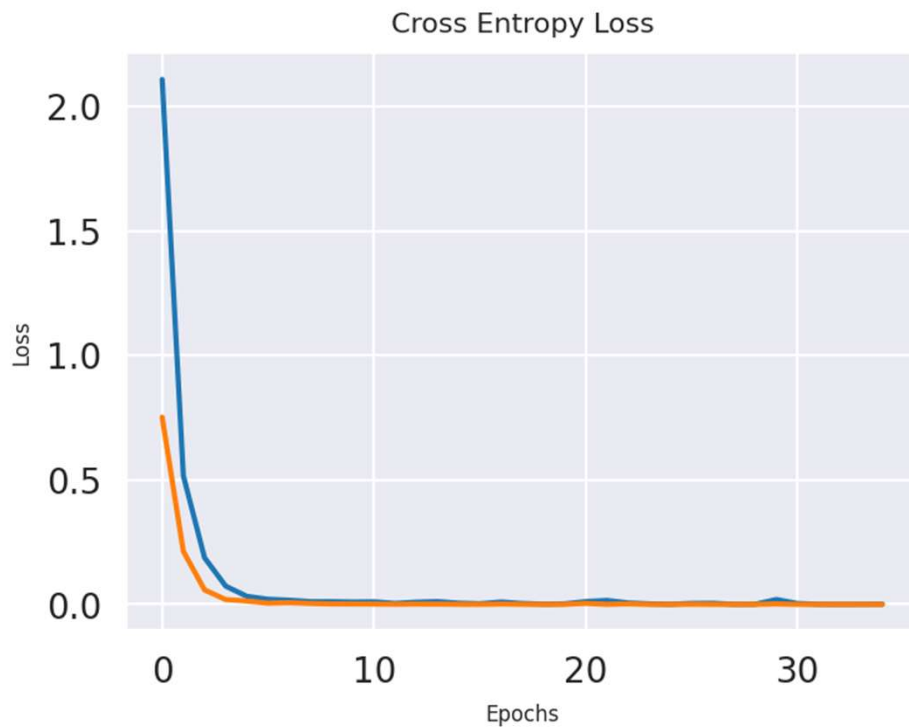
```
[31] (loss,acc)=model.evaluate(x=X_test,y=y_test)
```

```
225/225 [=====] - 1s 4ms/step - loss: 0.4970 - accuracy: 0.9243
```

```
[32] print('The accuracy of the model for testing data is:',acc*100)  
print('The Loss of the model for testing data is:',loss)
```

```
The accuracy of the model for testing data is: 92.42889285087585  
The Loss of the model for testing data is: 0.4970296025276184
```

CNN Loss and Accuracy Per Epoch



CNN Tuning

- Due to the exceptional accuracy of the first CNN model, tuning model parameters may show little to no improvements.
- The CNN accuracy levels off very quickly. Increasing the number of epochs from 35 to 48 produced virtually the same accuracy but increased test prediction accuracy to 93.35% and decreased loss from 0.4970 to 0.4784.

Epoch 48/48

110/110 [=====] - 1s 13ms/step - loss: 0.0017 - accuracy: 0.9995 - val_loss: 8.5035e-05 - val_accuracy: 1.0000

[35] #cnn_model2 evaluation and results

```
(loss2,acc2)=model.evaluate(x=X_test,y=y_test)
```

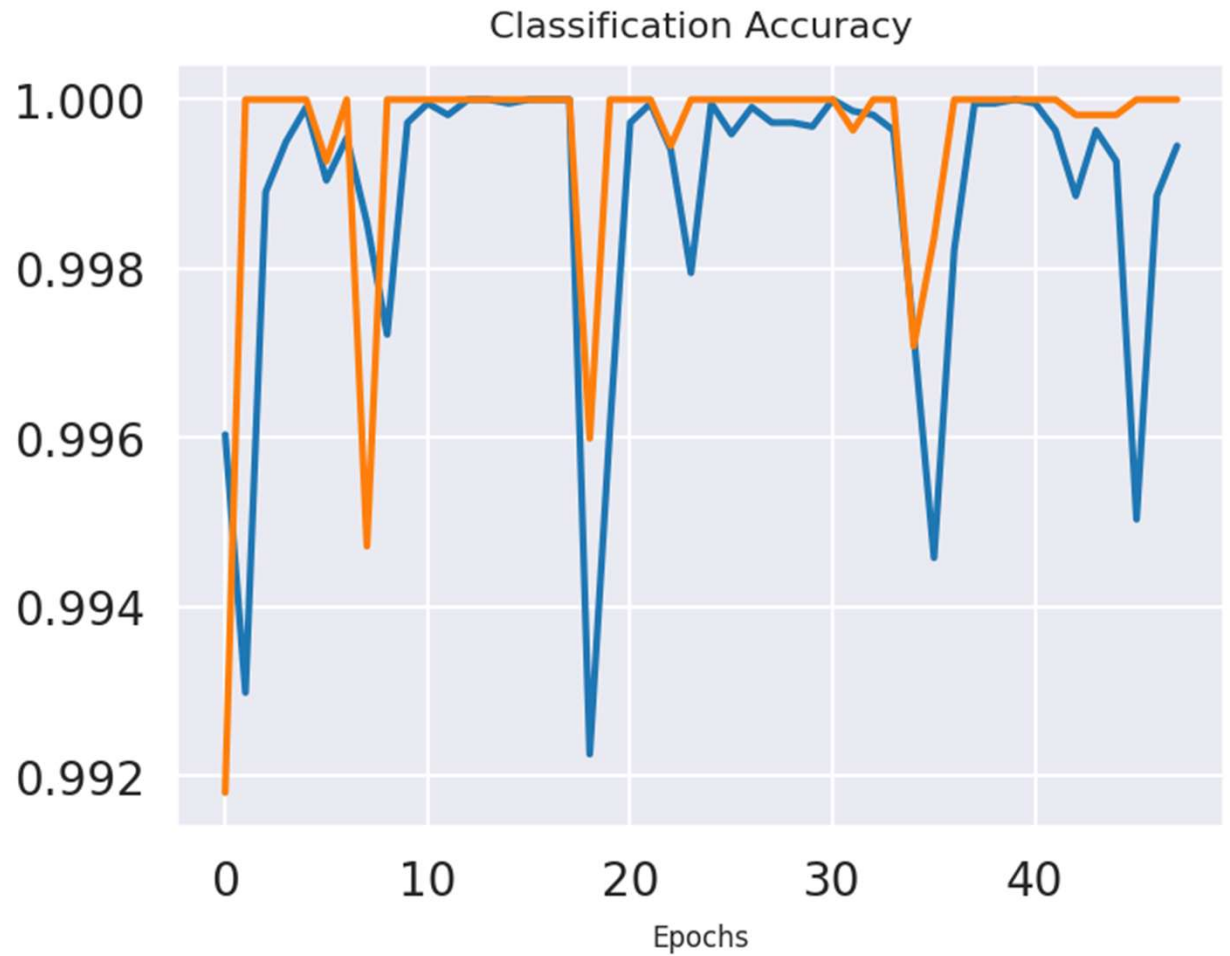
225/225 [=====] - 1s 3ms/step - loss: 0.4784 - accuracy: 0.9335

```
print('The accuracy of the model for testing data is:',acc2*100)
print('The Loss of the model for testing data is:',loss2)
```

The accuracy of the model for testing data is: 93.34913492202759
The Loss of the model for testing data is: 0.4784312844276428



CNN Classification Accuracy per Epoch



CNN Summary

- The developed CNN model has exhibited strong predictive capabilities with a significant training accuracy of around 99.95% and a test accuracy of 93.35%. The architecture, featuring a blend of convolutional layers, pooling layers, and dense layers, was instrumental in harnessing hierarchical feature representations crucial for ASL image classification.
- Despite its successes, the model shows signs of overfitting, as evidenced by the discrepancy between training and testing performances. This aspect necessitates further investigation and potential strategies such as data augmentation, regularization, or architectural adjustments to enhance generalization and robustness, ensuring that the model's predictions remain reliable and consistent across unseen or new data.
- In conclusion, the CNN model has demonstrated promising outcomes in classifying ASL images, paving the way for further optimizations and improvements to bolster its effectiveness and reliability in real-world applications.
- Adding additional epochs to the 2nd CNN greatly improved its loss metric during testing, and accuracy on test data.



Support Vector Classifier

- SVCs are also incredibly useful in certain image recognition scenarios because they create high-dimensional input spaces.
- SVCs utilize the “kernel trick” which transforms the input space so that hyperplanes can separate the classes. Non-linearities can be addressed using radial basis functions with SVC.
- Robust to overfitting, especially with proper tuning of the hyperparameters (C and gamma)
- The preprocessed nature of the data assisted in SVCs high accuracy (99%)

SVC Model Development

- More sensitive to NaN values which are removed for the model
- Using the 'rbf' kernel assisted in non linear problems such as image recognition
- Due to extremely high accuracy with minimal adjusting, cross validation is performed to check for overfitting

```
2 nan_rows_train = np.any(np.isnan(X_train.reshape(X_train.shape[0], -1)), axis=1)
3 nan_rows_test = np.any(np.isnan(X_test.reshape(X_test.shape[0], -1)), axis=1)
4
5
6 X_train_clean = X_train[~nan_rows_train]
7 y_train_clean = y_train[~nan_rows_train]
8
9 X_test_clean = X_test[~nan_rows_test]
10 y_test_clean = y_test[~nan_rows_test]
11
12 |
13 X_train_flatten = X_train_clean.reshape(X_train_clean.shape[0], -1)
14 X_test_flatten = X_test_clean.reshape(X_test_clean.shape[0], -1)
15
16 # Dimensionality issues
17 y_train_single_labels = np.argmax(y_train_clean, axis=1) if y_train_clean.ndim > 1 else y_train_clean
18 y_test_single_labels = np.argmax(y_test_clean, axis=1) if y_test_clean.ndim > 1 else y_test_clean
19
20
21 X_train_flatten = X_train_flatten / 255.0
22 X_test_flatten = X_test_flatten / 255.0
23
24
25 1 svc_model = SVC(kernel='rbf', random_state=31)
26
27
28 1 cross_val_scores = cross_val_score(svc_model, X_train_flatten, y_train_single_labels, cv=5)
29 2
30 3 print(f"Cross-Validation Accuracy Scores: {cross_val_scores}")
31 4 print(f"Mean Cross-Validation Score: {np.mean(cross_val_scores)}")
32 5
33
34 Cross-Validation Accuracy Scores: [1.          1.          0.99981788 0.99981788 1.          ]
35 Mean Cross-Validation Score: 0.9999271535239483
```

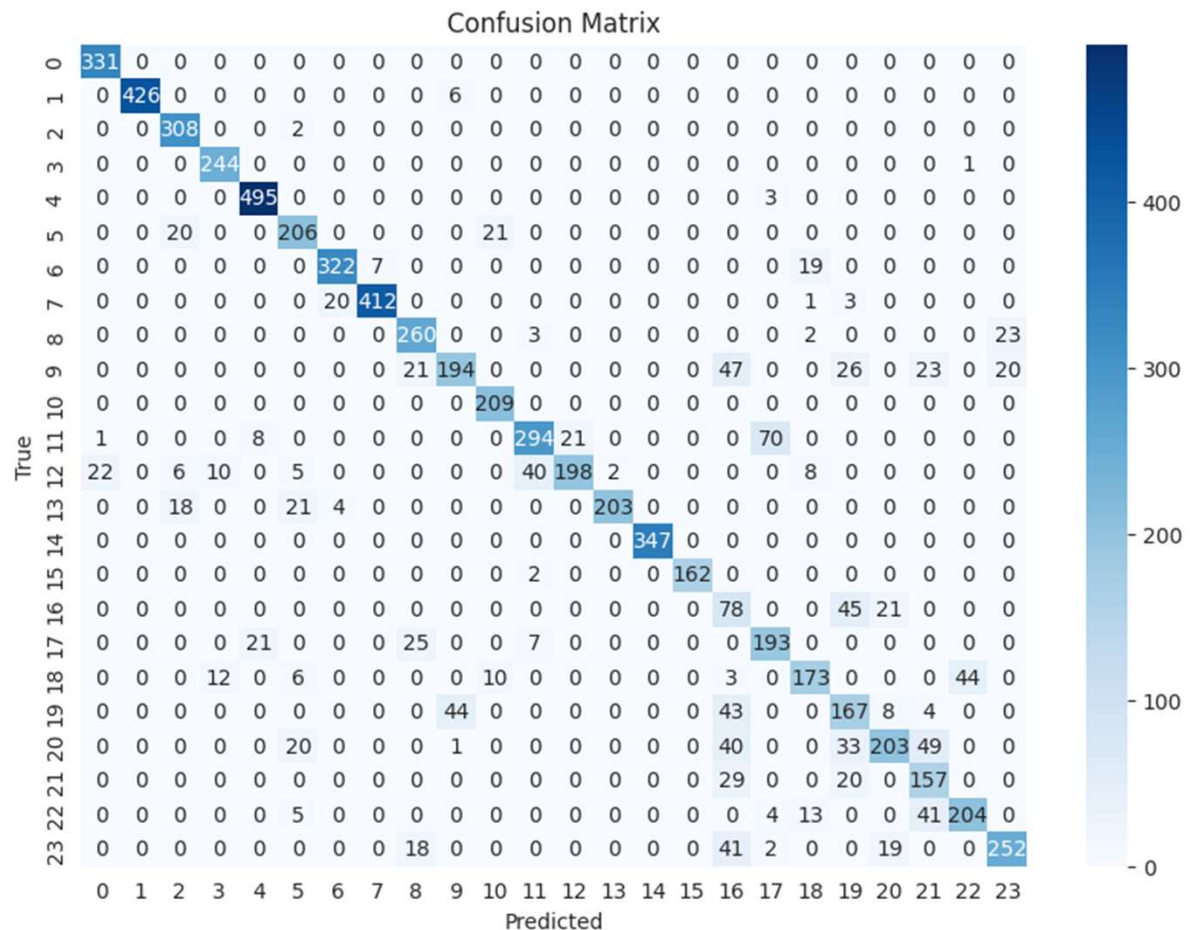


SVC Results

- The model achieved a mean cross-validation (CV) score of 0.99, demonstrating its robust performance during the training phase and its ability to handle various subsets of the dataset effectively.
- The SVC model attained a test accuracy of 0.84. This robust performance illustrates the model's capability to generalize well to unseen data, despite some degree of disparity from the training phase.
- The SVC model presents a strong, albeit varied, performance across different classes in the ASL dataset, proving its efficacy as a classification tool but also highlighting areas necessitating further refinement for enhanced accuracy and reliability.

SVC Confusion Matrix

- Misclassifications happen in reasonable areas. For example, ASL "M" and "S" are extremely similar in appearance when using ASL, and had the highest misclassification amount (70)



SVC Confusion Matrix Analysis

- **Precision, Recall, and F1-Score:**
 - Class 0 has high precision (0.94) and recall (1.00), resulting in an F1-score of 0.97, showing excellent classification performance.
 - However, certain classes like Class 16 exhibit lower precision (0.28) and a moderate recall (0.54), leading to a lesser F1-score of 0.37, indicating room for improvement.
- **Confusion Matrix Analysis:**
 - The confusion matrix reveals that the model has varying degrees of success in classifying different letters correctly.
 - Some letters are identified with high accuracy and precision, while others show mixed results, indicating potential areas where model fine-tuning and data preprocessing might improve performance.

SVC Model Tuning

- Using grid search CV, automated hyperparameter testing is applied to a 2nd SVC model
- Using feature scaling, imputation, and parameter grid searching in an attempt to improve the model all created a drastic decline in model performance.
- The original model outperformed the tuned version.
- The 'optimal' parameters chosen by GridSearchCV were:
 - $C = 1$
 - $\text{Gamma} = 1$
 - Kernel – 'RBF'



SVC Summary

- For a somewhat simpler model, SVC still performed quite well
- It appeared to be far more computationally expensive and much slower to cross-validate and produce quality results.
- Produced moderate-high accuracy on the testing data (~83%)
- The 'rbf' parameter allowed compensation for non-linearity, linear kernels did not perform well during tuning
- Parameter searching to find the ideal hyperparameters resulted in the default parameters being the best performing.
- Imputing the model and scaling features for tuning purposes greatly decreased model accuracy on the testing data (~43%)



Model Comparison Results

CNN is more computationally efficient and accurate, outperforming SVC in nearly every critical feat. CNN did not require many layers to correctly classify the ASL images and is arguably the simplest model compared to SVC in this experiment.

SVC could be considered simpler and more interpretable, but it did not demonstrate these features on this dataset. It required additional pre-processing, took much longer to run than the CNN, and tested at a lower accuracy rate despite having 100% accuracy on the cross-validated training approach.



Conclusion

AI models can automate the process of recognizing and interpreting sign language, making communication more accessible for the deaf and hard-of-hearing community.

It aids in creating systems or applications that can facilitate real-time interpretation of sign language.

CNN demonstrated much better performance and speed over SVC on this type of problem.

References

- Tecperson (2017). Sign Language MNIST. Drop-in Replacement for MNIST for Hand Gesture Recognition Tasks.
<https://www.kaggle.com/datasets/datamunge/sign-language-mnist/data>