Machine Learning Challenge - Group 8

Ayse Yanmaz, Orçun Özdemir, Raúl Zamora Zunino, Shifrah Goldblatt

Workload distribution:
AY: Loading, checking and splitting data for Task 1. Report on the hardest/easiest letters to classify for both models (RFC and CNN), Top-5-accuracy code and file generation for Task 2.
OÖ: RFC HP tuning using randomized search CV, reshaping of data for CNN, coding of CNN model, labeling, filtering, and predicting data for Task 2.
RZZ: Coding of the RFC initial model. RFC and CNN performance analysis on testing data, comparison of RFC and CNN final models.
SG: CNN HP tuning, coding of the updated RFC model after HP tuning, plotting, and reporting validation accuracy for CNN model, appendix.

Task 1 - Model comparison (RFC and CNN)

1) Learning models and algorithms used

A Random Forest Classifier (RFC) was chosen for the first model as it is a favorable model for obtaining fast results in the first stage, before hyperparameter tuning. Additionally, RFC consists of multiple decision trees, which allows training more than one model simultaneously, ultimately choosing a final robust outcome based on majority vote, which cannot be achieved via a simple decision tree model.

The second model used for this task was a Convolutional Neural Network (CNN); one of the most popular deep learning models for image analysis. With complex data, it is usually more convenient to start an image classification task with CNN owing to its arrangement of convolutional layers that allow identification of the structure underlying a set of pixels.

2) Features used and input to the classifier

Train and test datasets as well as train and test labels were provided for this task. The datasets consist of 748-dimension vectors which can be represented as an image of size 28x28 by reshaping. The training data also contains 27,455 images and labels while the test data consists of 7,172 vectors and labels. The dataset was large enough that image augmentation would not be necessary.

When examining the whole format of the data, it is seen that all the inputs are structurally homogeneous. While performing checks for missing data, no missing data was found in any dataset. Additionally, it was verified that labels J=9 and Z=25 were not present in the original data, thus being omitted during model construction.

Given all aspects regarding the size, quality and structure of the data, as well as the analysis expected to be performed, there was no need to apply any feature engineering such as extraction or transformation. Hence, no feature engineering was implemented for the first task.

Leading on, the train dataset was divided into a training and validation dataset via an 80/20 split using the sklearn.model_selection package, providing both a large enough dataset to train the model and an adequately sized validation dataset for hyperparameter tuning.

3) Hyperparameter tuning

Regarding hyperparameter tuning, the depth of the tree, the maximum features it can take, and the minimum proportion of split to prevent overfitting were tuned for RFC. Randomized search was

applied instead of grid search as it does not assess all possible hyperparameter combinations to fit the data, making it more computationally efficient.

The original unsplit training dataset was used here since randomized search itself splits the data into train/validation (80/20) by default, so using the originally split data would result in tuning on a smaller sample.

The performance of the RFC was tested on the previously designated validation dataset using the optimal hyperparameters found by the randomized search (max depth = 10, max features = 4, min sample split = 7). However, this saw a decrease in F1 score, from 0.98 to 0.97 (see appendices A and B). Considering this, it can be argued that the hyperparameters selected by default in the initial model perform better, therefore the original model was implemented. Moreover, when tuning, not all hyperparameters were randomized which impeded finding the optimal set of hyperparameters.

For the CNN model, tuning was manually conducted on the number of epochs and batch size. Epochs were initially set to 1, resulting in an accuracy of 0.72. To find the optimal number, epochs were then set to 30 with a graph illustrating how validation accuracy improved with each iteration. The graph clearly demonstrated that validation accuracy began to plateau at around 15 epochs. Although more iterations led to a slightly higher accuracy, it was computationally expensive and training time was long, so 15 epochs were chosen; producing an accuracy of 0.95 (see appendix C).

Additionally, tuning of batch size was explored in an attempt to improve the time taken to train the model (273 seconds). A smaller batch size of 100 was tested, reducing training time to 208 seconds as well as maintaining an accuracy of 0.95, while also reducing loss from 0.25 to 0.17. When testing a larger batch size of 1,000, training time was minimized even further to 172 seconds, but at the expense of the model's accuracy which dropped to 0.9 (see appendix D). Ultimately, a batch size of 100 was implemented.

4) Final models specs

Since the RFC model did not improve in accuracy after hyperparameter tuning, the original model with default specs was chosen. The specs were set by the default parameters from the scikit library with max depth set to 10.

The specs of the final CNN model include 15 epochs and a batch size of 100. Additionally, ReLU was selected as the activation function. ReLU is the classically implemented activation function for neural networks due to its nonlinear nature as well as being more computationally efficient than other popular activation functions. Softmax was used for the inverse logit activation function of the final layer instead of sigmoid as softmax is suitable for multi-classification, whereas sigmoid handles binary classification. The Adam optimiser was selected due to requiring less memory and being useful for large datasets. It also builds upon classic gradient descent by updating weights iteratively, reaching the minimum faster. This was paired with categorical cross-entropy as a loss function since this is the most suitable method for a multi-classification problem.

5) Performance results for final models

The F1 scores clearly illustrate CNN as the superior model for this classification task. The RFC model has a test score of 0.74 while the CNN model boasts 0.95 accuracy on unseen data (see appendix E).

Additionally, when inspecting F1 scores for accurately identifying individual letters, CNN is once again seen to have higher scores for both easiest and hardest letters to classify. For CNN, the letters "a", "l", "s", "p", and "m" were found to be the easiest to classify with all letters obtaining a perfect F1 score of

1. On the other hand, letters “r”, “t”, “v”, “g”, and “x” were the most difficult to classify (F1 = 0.76, 0.82, 0.87, 0.89, 0.89) (see appendix F).

For RFC, the easiest letters to classify were, “p”, “a”, “c”, “b”, and “h” (F1 = 0.94, 0.93, 0.93, 0.90, 0.89) and the hardest letters to classify were “r”, “w”, “n”, “v”, “s” (F1 = 0.32, 0.51, 0.51, 0.53, 0.53) (see appendix G). The accuracy scores of all other letters and confusion matrices for both models are also available in appendices F and G.

Finally, when comparing the performance of the two developed models, although RFC is faster to train (32 seconds) (see appendix A), the accuracy of the model (0.74) on unseen data is much lower than CNN (0.95) (see appendix E). Therefore, CNN was selected as the appropriate model for the requirements of the second task.

Task 2 - Classification of new instances

Examination of the data illustrated the whole dataset had 10,000 observations, each composed of a 28x200 array. Initial visualization revealed the presence of noise alongside the real images of hand gestures. Further inspection indicated that every column of noise had a distinct mode of 200. In order to remove the noise from the data and leave only valid inputs of 28x28 images on each observation, every observation was transposed, and then the mode was calculated iteratively, row by row. A function was created to detect and filter out every row with a mode of 200.

To compensate for the fact that some valid images could also have rows with a mode of 200, a helper function was developed. The helper function inspected the dimensions of the cleaned data, and when an observation did not match the expected dimensions (28x28), a 28xN array made of ones was filled to the edge of the matrix.

When considering that this procedure could eliminate parts of the target images and eventually incorporate distortion into them, it is regarded that this issue was not intense enough to impair the following prediction task. The procedure was checked visually on approximately 20 observations. It was observed that only sign language images were remaining, all of them readable and visually identifiable after cropping and filling.

Finally, for the purpose of classifying the new instances after cropping, the final CNN model from Task 1 was used as it demonstrated the best performance. No other model or new model was trained nor used for this task.

The result of predicting using the CNN model produced the probability of each 28x28 image for each target class. To identify the 5 best predictions for each image within the 10,000 observations, the argsort in decreasing order function was used. For each image within the 10,000 observations, the Nth best predictions were concatenated. The results are displayed as a matrix where the rows represent the observations (one to five images), and each N column is the Nth best prediction for each letter of the observation (see appendix H).

References and resources

Original Dataset: Sign Language MNIST - https://www.kaggle.com/datamunge/sign-language-mnist

Scikit learn resources - https://scikit-learn.org/stable/about.html

Appendix

APPENDIX A: Random Forest Classifier Before Hyperparameter Tuning

i) Fitting the original RFC model

```
## Random Forest Classifier
print("Random Forest Classifier Part Initiated.")
print("\n")


# Training model


start = time()
clf = RandomForestClassifier(max_depth=10, random_state=54020)
clf.fit(train_data, train_label)

print(f'Time taken to run: {time() - start} seconds')
```

ii) F1 score of the original RFC model

```
# Running confusion matrix on validation dataset
print("Confusion Matrix on Validation Data")
print("\n")
print(confusion_matrix(val_label, clf.predict(val_data)))
print("\n")


# Testing accuracy using f1 score

print("Fit model using default hyperparameters")
print("F1 Score: ",f1_score(val_label, clf.predict(val_data), average='weighted'))
print("\n")
```

```
Fit model using default hyperparameters
F1 Score:  0.9777172212816997
```

iii) F1 score of the original RFC model on test data

```
# Testing the accuracy of the base model on unseen data
print("Testing the accuracy of the base model on unseen data")
print("F1 Score of Base Model for Test Data: ",f1_score(test_label, clf.predict(test_data), average='weighted'))

print("\n")
```

```
Testing the accuracy of the base model on unseen data
F1 Score of Base Model for Test Data:  0.7392929814733281
```

iv) Time taken to train the model

```
## Random Forest Classifier
print("Random Forest Classifier Part Initiated.")
print("\n")


# Training model


start = time()
clf = RandomForestClassifier(max_depth=10, random_state=54020)
clf.fit(train_data, train_label)

print(f'Time taken to run: {time() - start} seconds')
```

```
Random Forest Classifier Part Initiated.


Time taken to run: 32.39091753959656 seconds
```

APPENDIX B: Random Forest Classifier After Hyperparameter Tuning

i) Hyperparameter tuning of the RFC model using randomized search

```
# Hyperparameter tuning using randomised search cross validation
print("Hyperparameter tuning using randomised search cross validation")
print("\n")

start = time()
features_dict = {
        "max_depth": [3, 5, 7, 10],
        "max_features": ss.randint(1, 20),
        "min_samples_split": ss.randint(2, 20),
    }

print("for HP, use both original training and validation together, for efficiency of the data.")
```

```
Hyperparameter tuning using randomised search cross validation


for HP, use both original training and validation together, for efficiency of the data.
```

```
# for HP, use both original training and validation together, for efficiency of the data.

rs_p = RandomizedSearchCV(clf, features_dict, random_state=54020,verbose=2)
randomized_searched_model = rs_p.fit(train_data_unsplit, train_label_unsplit)
print(randomized_searched_model.best_params_)
print(f'Time taken to run: {time() - start} seconds')
print("\n")
```

```
Fitting 5 folds for each of 10 candidates, totalling 50 fits
[CV] END ..max_depth=10, max_features=4, min_samples_split=7; total time=   5.0s
[CV] END ..max_depth=10, max_features=4, min_samples_split=7; total time=   5.1s
[CV] END ..max_depth=10, max_features=4, min_samples_split=7; total time=   4.7s
[CV] END ..max_depth=10, max_features=4, min_samples_split=7; total time=   4.7s
[CV] END ..max_depth=10, max_features=4, min_samples_split=7; total time=   4.7s
[CV] END ..max_depth=7, max_features=7, min_samples_split=13; total time=   5.2s
[CV] END ..max_depth=7, max_features=7, min_samples_split=13; total time=   5.3s
[CV] END ..max_depth=7, max_features=7, min_samples_split=13; total time=   6.8s
[CV] END ..max_depth=7, max_features=7, min_samples_split=13; total time=   8.2s
[CV] END ..max_depth=7, max_features=7, min_samples_split=13; total time=   5.6s
[CV] END ..max_depth=5, max_features=5, min_samples_split=18; total time=   2.9s
[CV] END ..max_depth=5, max_features=5, min_samples_split=18; total time=   3.5s
[CV] END ..max_depth=5, max_features=5, min_samples_split=18; total time=   3.0s
[CV] END ..max_depth=5, max_features=5, min_samples_split=18; total time=   2.7s
[CV] END ..max_depth=5, max_features=5, min_samples_split=18; total time=   2.6s
[CV] END ..max_depth=3, max_features=16, min_samples_split=2; total time=   4.4s
[CV] END ..max_depth=3, max_features=16, min_samples_split=2; total time=   4.4s
[CV] END ..max_depth=3, max_features=16, min_samples_split=2; total time=   4.4s
[CV] END ..max_depth=3, max_features=16, min_samples_split=2; total time=   4.4s
[CV] END ..max_depth=3, max_features=16, min_samples_split=2; total time=   4.6s
[CV] END .max_depth=7, max_features=17, min_samples_split=15; total time=  11.7s
[CV] END .max_depth=7, max_features=17, min_samples_split=15; total time=  12.7s
[CV] END .max_depth=7, max_features=17, min_samples_split=15; total time=  11.8s
[CV] END .max_depth=7, max_features=17, min_samples_split=15; total time=  14.8s
[CV] END .max_depth=7, max_features=17, min_samples_split=15; total time=  10.6s
[CV] END .max_depth=7, max_features=14, min_samples_split=10; total time=   9.3s
[CV] END .max_depth=7, max_features=14, min_samples_split=10; total time=   9.7s
[CV] END .max_depth=7, max_features=14, min_samples_split=10; total time=   9.5s
[CV] END .max_depth=7, max_features=14, min_samples_split=10; total time=   9.4s
[CV] END .max_depth=7, max_features=14, min_samples_split=10; total time=   9.6s
[CV] END ...max_depth=7, max_features=2, min_samples_split=5; total time=   2.1s
[CV] END ...max_depth=7, max_features=2, min_samples_split=5; total time=   1.9s
[CV] END ...max_depth=7, max_features=2, min_samples_split=5; total time=   1.9s
[CV] END ...max_depth=7, max_features=2, min_samples_split=5; total time=   1.9s
[CV] END ...max_depth=7, max_features=2, min_samples_split=5; total time=   2.0s
[CV] END ..max_depth=5, max_features=8, min_samples_split=19; total time=   3.9s
[CV] END ..max_depth=5, max_features=8, min_samples_split=19; total time=   4.4s
[CV] END ..max_depth=5, max_features=8, min_samples_split=19; total time=   4.2s
[CV] END ..max_depth=5, max_features=8, min_samples_split=19; total time=   4.1s
[CV] END ..max_depth=5, max_features=8, min_samples_split=19; total time=   4.2s
```

```
[CV] END ..max_depth=10, max_features=1, min_samples_split=8; total time=   1.9s
[CV] END ..max_depth=10, max_features=1, min_samples_split=8; total time=   1.7s
[CV] END ..max_depth=10, max_features=1, min_samples_split=8; total time=   1.9s
[CV] END ..max_depth=10, max_features=1, min_samples_split=8; total time=   1.9s
[CV] END ..max_depth=10, max_features=1, min_samples_split=8; total time=   1.9s
[CV] END ..max_depth=3, max_features=7, min_samples_split=14; total time=   2.8s
[CV] END ..max_depth=3, max_features=7, min_samples_split=14; total time=   2.2s
[CV] END ..max_depth=3, max_features=7, min_samples_split=14; total time=   2.3s
[CV] END ..max_depth=3, max_features=7, min_samples_split=14; total time=   2.3s
[CV] END ..max_depth=3, max_features=7, min_samples_split=14; total time=   2.2s
{'max_depth': 10, 'max_features': 4, 'min_samples_split': 7}
Time taken to run: 264.2878420352936 seconds
```

ii) Fitting the updated model & F1 score

```
# Optimal hyperparameters = {'max_depth': 10, 'max_features': 4, 'min_samples_split': 7}
# Fit model using new optimal hyperparameters

print("Optimal hyperparameters = {'max_depth': 10, 'max_features': 4, 'min_samples_split': 7}")
print("Fit model using new optimal hyperparameters")

start = time()
hyperparameter_optimized_clf = RandomForestClassifier(max_depth=10,max_features=4 ,min_samples_split=7,random_state=54020)
hyperparameter_optimized_clf.fit(train_data, train_label)

print("F1 Score for Validation Data: ",f1_score(val_label, hyperparameter_optimized_clf.predict(val_data), average='weighted'))

print(f'Time taken to run: {time() - start} seconds')

print("\n")
```

```
Optimal hyperparameters = {'max_depth': 10, 'max_features': 4, 'min_samples_split': 7}
Fit model using new optimal hyperparameters
F1 Score for Validation Data:  0.964908364851211
```

iii) F1 score of the updated RFC model on test data

```
# Testing the accuracy of the updated model on unseen data

print("Testing the accuracy of the updated model on unseen data")
print("F1 Score of Updated Model for Test Data: ",f1_score(test_label, hyperparameter_optimized_clf.predict(test_data), average='weighted'))

print("\n")
```

```
Testing the accuracy of the updated model on unseen data
F1 Score of Updated Model for Test Data:  0.7090054537043667
```

iv) Time taken to train the model

```
# Optimal hyperparameters = {'max_depth': 10, 'max_features': 4, 'min_samples_split': 7}
# Fit model using new optimal hyperparameters

print("Optimal hyperparameters = {'max_depth': 10, 'max_features': 4, 'min_samples_split': 7}")
print("Fit model using new optimal hyperparameters")

start = time()
hyperparameter_optimized_clf = RandomForestClassifier(max_depth=10,max_features=4 ,min_samples_split=7,random_state=54020)
hyperparameter_optimized_clf.fit(train_data, train_label)

print("F1 Score for Validation Data: ",f1_score(val_label, hyperparameter_optimized_clf.predict(val_data), average='weighted'))

print(f'Time taken to run: {time() - start} seconds')

print("\n")
```

```
Optimal hyperparameters = {'max_depth': 10, 'max_features': 4, 'min_samples_split': 7}
Fit model using new optimal hyperparameters
F1 Score for Validation Data:  0.964908364851211
Time taken to run: 4.439097881317139 seconds
```

APPENDIX C: Convolutional Neural Network hyperparameter tuning (Epochs)

i) Hyperparameter tuning: 1 epoch

```
# Hyperparameter tuning

print("Hyperparameter Tuning")
print("\n")

learning_rate_reduction = tf.keras.callbacks.ReduceLROnPlateau(
    monitor='val_accuracy', patience = 2, verbose=1,factor=0.5, min_lr=0.00001
)

model.compile(optimizer='adam',
              loss=tf.keras.losses.SparseCategoricalCrossentropy(from_logits=True),
              metrics=['accuracy'])

start = time()
history = model.fit(reshaped_train_data, train_label, epochs=1,
                    validation_data=(reshaped_val_data, val_label),callbacks=[
                                        tf.keras.callbacks.EarlyStopping(
                                            monitor='val_loss',
                                            patience=3,
                                            restore_best_weights=True
                                            ),learning_rate_reduction])

print(f'Time taken to run: {time() - start} seconds')
```

```
Test Loss and Test Accuracy:
225/225 - 2s - loss: 0.9212 - accuracy: 0.7202 - 2s/epoch - 8ms/step
0.9211756587028503 0.7201617360115051
```

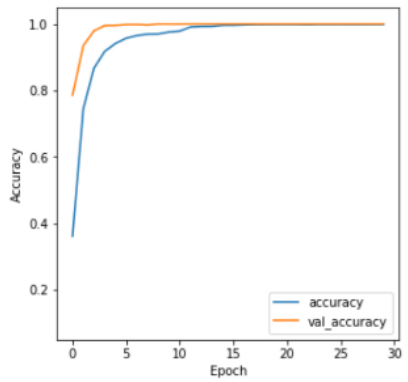
```
# Accuracy of model on unseen data
print("Accuracy of model on unseen data")
print(f1_score(test_label, predicted_test_labels, average='weighted'))

import pandas as pd
from sklearn.metrics import confusion_matrix, ConfusionMatrixDisplay, classification_report, precision_recall_fscore_support
import matplotlib.pyplot as plt
```

```
Accuracy of model on unseen data
0.7171114164385964
```

ii) Hyperparameter tuning: 30 epochs

```
# Hyperparameter tuning

print("Hyperparameter Tuning")
print("\n")

learning_rate_reduction = tf.keras.callbacks.ReduceLROnPlateau(
    monitor='val_accuracy', patience = 2, verbose=1,factor=0.5, min_lr=0.00001
)

model.compile(optimizer='adam',
              loss=tf.keras.losses.SparseCategoricalCrossentropy(from_logits=True),
              metrics=['accuracy'])

start = time()
history = model.fit(reshaped_train_data, train_label, epochs=30,
                    validation_data=(reshaped_val_data, val_label),callbacks=[
                                        tf.keras.callbacks.EarlyStopping(
                                            monitor='val_loss',
                                            patience=3,
                                            restore_best_weights=True
                                            ),learning_rate_reduction])

print(f'Time taken to run: {time() - start} seconds')
```

```
Test Loss and Test Accuracy:
225/225 - 2s - loss: 0.2692 - accuracy: 0.9610 - 2s/epoch - 8ms/step
0.2691827714443207 0.9609593152999878
```



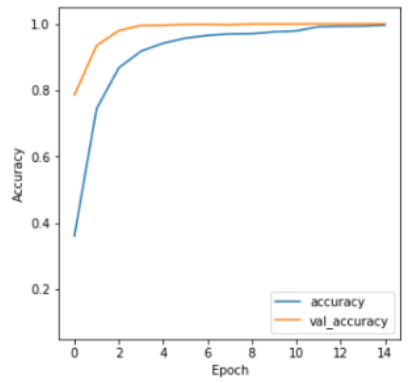
```
# Accuracy of model on unseen data
print("Accuracy of model on unseen data")
print(f1_score(test_label, predicted_test_labels, average='weighted'))

import pandas as pd
from sklearn.metrics import confusion_matrix, ConfusionMatrixDisplay, classification_report, precision_recall_fscore_support
import matplotlib.pyplot as plt
```

```
Accuracy of model on unseen data
0.9610156817071468
```

iii) Hyperparameter tuning: 15 epochs

```
# Hyperparameter tuning

print("Hyperparameter Tuning")
print("\n")

learning_rate_reduction = tf.keras.callbacks.ReduceLROnPlateau(
    monitor='val_accuracy', patience = 2, verbose=1,factor=0.5, min_lr=0.00001
)

model.compile(optimizer='adam',
              loss=tf.keras.losses.SparseCategoricalCrossentropy(from_logits=True),
              metrics=['accuracy'])

start = time()
history = model.fit(reshaped_train_data, train_label, epochs=15,
                    validation_data=(reshaped_val_data, val_label),callbacks=[
                                            tf.keras.callbacks.EarlyStopping(
                                                monitor='val_loss',
                                                patience=3,
                                                restore_best_weights=True
                                                ),learning_rate_reduction])

print(f'Time taken to run: {time() - start} seconds')
```

```
Test Loss and Test Accuracy:
225/225 - 2s - loss: 0.2473 - accuracy: 0.9513 - 2s/epoch - 7ms/step
0.24732373654842377 0.951338529586792
```




```
# Accuracy of model on unseen data
print("Accuracy of model on unseen data")
print(f1_score(test_label, predicted_test_labels, average='weighted'))

import pandas as pd
from sklearn.metrics import confusion_matrix, ConfusionMatrixDisplay, classification_report, precision_recall_fscore_support
import matplotlib.pyplot as plt
```

```
Accuracy of model on unseen data
0.951795871808639
```

APPENDIX D: Convolutional Neural Network hyperparameter tuning (Batch size)

i) Run time, test accuracy, and test lost of CNN model (15 epochs, no batch size)

```
Epoch 1/15
687/687 [==============================] - 17s 22ms/step - loss: 2.2950 - accuracy: 0.3612 - val_loss: 0.7012 -
val_accuracy: 0.7858 - lr: 0.0010
Epoch 2/15
687/687 [==============================] - 18s 27ms/step - loss: 0.7504 - accuracy: 0.7438 - val_loss: 0.2113 -
val_accuracy: 0.9344 - lr: 0.0010
Epoch 3/15
687/687 [==============================] - 19s 27ms/step - loss: 0.3959 - accuracy: 0.8671 - val_loss: 0.0744 -
val_accuracy: 0.9794 - lr: 0.0010
Epoch 4/15
687/687 [==============================] - 22s 32ms/step - loss: 0.2473 - accuracy: 0.9175 - val_loss: 0.0233 -
val_accuracy: 0.9953 - lr: 0.0010
Epoch 5/15
687/687 [==============================] - 25s 37ms/step - loss: 0.1798 - accuracy: 0.9414 - val_loss: 0.0188 -
val_accuracy: 0.9956 - lr: 0.0010
Epoch 6/15
687/687 [==============================] - 19s 28ms/step - loss: 0.1330 - accuracy: 0.9570 - val_loss: 0.0130 -
val_accuracy: 0.9982 - lr: 0.0010
Epoch 7/15
687/687 [==============================] - 21s 30ms/step - loss: 0.1111 - accuracy: 0.9651 - val_loss: 0.0074 -
val_accuracy: 0.9984 - lr: 0.0010
Epoch 8/15
687/687 [==============================] - 19s 28ms/step - loss: 0.0987 - accuracy: 0.9695 - val_loss: 0.0082 -
val_accuracy: 0.9969 - lr: 0.0010
Epoch 9/15
687/687 [==============================] - 15s 22ms/step - loss: 0.0941 - accuracy: 0.9701 - val_loss: 0.0015 -
val_accuracy: 0.9998 - lr: 0.0010
Epoch 10/15
687/687 [==============================] - 15s 22ms/step - loss: 0.0753 - accuracy: 0.9758 - val_loss: 0.0019 -
val_accuracy: 0.9996 - lr: 0.0010
Epoch 11/15
686/687 [==============================>.] - ETA: 0s - loss: 0.0759 - accuracy: 0.9785
Epoch 11: ReduceLROnPlateau reducing learning rate to 0.0005000000237487257.
687/687 [==============================] - 15s 21ms/step - loss: 0.0759 - accuracy: 0.9786 - val_loss: 0.0019 -
val_accuracy: 0.9996 - lr: 0.0010
Epoch 12/15
687/687 [==============================] - 14s 21ms/step - loss: 0.0305 - accuracy: 0.9911 - val_loss:
5.2925e-04 - val_accuracy: 1.0000 - lr: 5.0000e-04
Epoch 13/15
687/687 [==============================] - 19s 28ms/step - loss: 0.0221 - accuracy: 0.9927 - val_loss:
3.7179e-04 - val_accuracy: 1.0000 - lr: 5.0000e-04
Epoch 14/15
686/687 [==============================>.] - ETA: 0s - loss: 0.0240 - accuracy: 0.9930
Epoch 14: ReduceLROnPlateau reducing learning rate to 0.0002500000118743628.
687/687 [==============================] - 21s 30ms/step - loss: 0.0240 - accuracy: 0.9930 - val_loss:
3.3057e-05 - val_accuracy: 1.0000 - lr: 5.0000e-04
Epoch 15/15
687/687 [==============================] - 15s 21ms/step - loss: 0.0140 - accuracy: 0.9961 - val_loss:
6.5598e-05 - val_accuracy: 1.0000 - lr: 2.5000e-04
Time taken to run: 273.92393589019775 seconds


Test Loss and Test Accuracy:
225/225 - 1s - loss: 0.2473 - accuracy: 0.9513 - 1s/epoch - 6ms/step
0.24732373654842377 0.951338529586792
```

ii) Run time, test accuracy, and test loss of CNN model (15 epochs, batch size = 100)

```
Epoch 1/15
220/220 [==============================] - 12s 51ms/step - loss: 3.0511 - accuracy: 0.2328 - val_loss: 1.2230 -
val_accuracy: 0.6383 - lr: 0.0010
Epoch 2/15
220/220 [==============================] - 12s 53ms/step - loss: 1.1082 - accuracy: 0.6286 - val_loss: 0.4058 -
val_accuracy: 0.9015 - lr: 0.0010
Epoch 3/15
220/220 [==============================] - 15s 69ms/step - loss: 0.6063 - accuracy: 0.7947 - val_loss: 0.1753 -
val_accuracy: 0.9517 - lr: 0.0010
Epoch 4/15
220/220 [==============================] - 14s 62ms/step - loss: 0.3670 - accuracy: 0.8775 - val_loss: 0.0667 -
val_accuracy: 0.9851 - lr: 0.0010
Epoch 5/15
220/220 [==============================] - 15s 68ms/step - loss: 0.2334 - accuracy: 0.9213 - val_loss: 0.0339 -
val_accuracy: 0.9936 - lr: 0.0010
Epoch 6/15
220/220 [==============================] - 14s 64ms/step - loss: 0.1677 - accuracy: 0.9455 - val_loss: 0.0168 -
val_accuracy: 0.9969 - lr: 0.0010
Epoch 7/15
220/220 [==============================] - 18s 83ms/step - loss: 0.1305 - accuracy: 0.9579 - val_loss: 0.0098 -
val_accuracy: 0.9996 - lr: 0.0010
Epoch 8/15
220/220 [==============================] - 12s 56ms/step - loss: 0.1034 - accuracy: 0.9663 - val_loss: 0.0054 -
val_accuracy: 0.9995 - lr: 0.0010
Epoch 9/15
219/220 [============================>.] - ETA: 0s - loss: 0.0940 - accuracy: 0.9693
Epoch 9: ReduceLROnPlateau reducing learning rate to 0.0005000000237487257.
220/220 [==============================] - 13s 58ms/step - loss: 0.0940 - accuracy: 0.9693 - val_loss: 0.0069 -
val_accuracy: 0.9993 - lr: 0.0010
Epoch 10/15
220/220 [==============================] - 14s 63ms/step - loss: 0.0499 - accuracy: 0.9842 - val_loss: 0.0021 -
val_accuracy: 1.0000 - lr: 5.0000e-04
Epoch 11/15
220/220 [==============================] - 14s 66ms/step - loss: 0.0435 - accuracy: 0.9875 - val_loss: 0.0013 -
val_accuracy: 1.0000 - lr: 5.0000e-04
Epoch 12/15
220/220 [==============================] - ETA: 0s - loss: 0.0418 - accuracy: 0.9865
Epoch 12: ReduceLROnPlateau reducing learning rate to 0.0002500000118743628.
220/220 [==============================] - 15s 66ms/step - loss: 0.0418 - accuracy: 0.9865 - val_loss: 0.0023 -
val_accuracy: 0.9998 - lr: 5.0000e-04
Epoch 13/15
220/220 [==============================] - 13s 57ms/step - loss: 0.0292 - accuracy: 0.9903 - val_loss: 0.0010 -
val_accuracy: 1.0000 - lr: 2.5000e-04
Epoch 14/15
220/220 [==============================] - ETA: 0s - loss: 0.0291 - accuracy: 0.9911
Epoch 14: ReduceLROnPlateau reducing learning rate to 0.0001250000059371814.
220/220 [==============================] - 14s 66ms/step - loss: 0.0291 - accuracy: 0.9911 - val_loss:
7.0294e-04 - val_accuracy: 1.0000 - lr: 2.5000e-04
Epoch 15/15
220/220 [==============================] - 13s 61ms/step - loss: 0.0236 - accuracy: 0.9930 - val_loss:
6.0919e-04 - val_accuracy: 1.0000 - lr: 1.2500e-04
Time taken to run: 208.05656361579895 seconds


Test Loss and Test Accuracy:
225/225 - 1s - loss: 0.1681 - accuracy: 0.9516 - 1s/epoch - 5ms/step
0.1680556684732437 0.951617419719696
```

iii) Run time, test accuracy, and test loss of CNN model (15 epochs, batch size = 1000)

```
Epoch 1/15
22/22 [==============================] - 13s 535ms/step - loss: 8.1041 - accuracy: 0.0455 - val_loss: 3.1838 - val_accuracy: 0.0479 - lr: 0.0010
Epoch 2/15
22/22 [==============================] - 11s 514ms/step - loss: 3.0810 - accuracy: 0.0860 - val_loss: 2.8859 - val_accuracy: 0.1688 - lr: 0.0010
Epoch 3/15
22/22 [==============================] - 11s 485ms/step - loss: 2.6834 - accuracy: 0.2001 - val_loss: 2.2122 - val_accuracy: 0.3866 - lr: 0.0010
Epoch 4/15
22/22 [==============================] - 10s 476ms/step - loss: 2.0735 - accuracy: 0.3605 - val_loss: 1.3809 - val_accuracy: 0.6197 - lr: 0.0010
Epoch 5/15
22/22 [==============================] - 10s 479ms/step - loss: 1.5878 - accuracy: 0.4902 - val_loss: 0.9619 - val_accuracy: 0.7301 - lr: 0.0010
Epoch 6/15
22/22 [==============================] - 11s 483ms/step - loss: 1.1941 - accuracy: 0.6048 - val_loss: 0.6484 - val_accuracy: 0.8221 - lr: 0.0010
Epoch 7/15
22/22 [==============================] - 11s 507ms/step - loss: 0.9549 - accuracy: 0.6853 - val_loss: 0.4540 - val_accuracy: 0.8869 - lr: 0.0010
Epoch 8/15
22/22 [==============================] - 12s 551ms/step - loss: 0.7627 - accuracy: 0.7453 - val_loss: 0.3200 - val_accuracy: 0.9261 - lr: 0.0010
Epoch 9/15
22/22 [==============================] - 11s 513ms/step - loss: 0.6284 - accuracy: 0.7875 - val_loss: 0.2327 - val_accuracy: 0.9426 - lr: 0.0010
Epoch 10/15
22/22 [==============================] - 11s 519ms/step - loss: 0.5134 - accuracy: 0.8266 - val_loss: 0.1783 - val_accuracy: 0.9614 - lr: 0.0010
Epoch 11/15
22/22 [==============================] - 12s 545ms/step - loss: 0.4283 - accuracy: 0.8539 - val_loss: 0.1431 - val_accuracy: 0.9636 - lr: 0.0010
Epoch 12/15
22/22 [==============================] - 13s 574ms/step - loss: 0.3578 - accuracy: 0.8842 - val_loss: 0.1004 - val_accuracy: 0.9796 - lr: 0.0010
Epoch 13/15
22/22 [==============================] - 12s 554ms/step - loss: 0.3075 - accuracy: 0.9005 - val_loss: 0.0730 - val_accuracy: 0.9869 - lr: 0.0010
Epoch 14/15
22/22 [==============================] - 12s 530ms/step - loss: 0.2776 - accuracy: 0.9071 - val_loss: 0.0645 - val_accuracy: 0.9905 - lr: 0.0010
Epoch 15/15
22/22 [==============================] - 11s 514ms/step - loss: 0.2334 - accuracy: 0.9216 - val_loss: 0.0446 - val_accuracy: 0.9949 - lr: 0.0010
Time taken to run: 171.67975687980652 seconds

Test Loss and Test Accuracy:
225/225 - 2s - loss: 0.2789 - accuracy: 0.8999 - 2s/epoch - 9ms/step
0.2789001762866974 0.8998884558677673
```

APPENDIX E: F1 Scores of the Final RFC and CNN Models on Test Data

i) F1 score of RFC on test data

```
# Testing the accuracy of the base model on unseen data
print("Testing the accuracy of the base model on unseen data")
print("F1 Score of Base Model for Test Data: ",f1_score(test_label, clf.predict(test_data), average='weighted'))

print("\n")
```

```
Testing the accuracy of the base model on unseen data
F1 Score of Base Model for Test Data:  0.7392929814733281
```

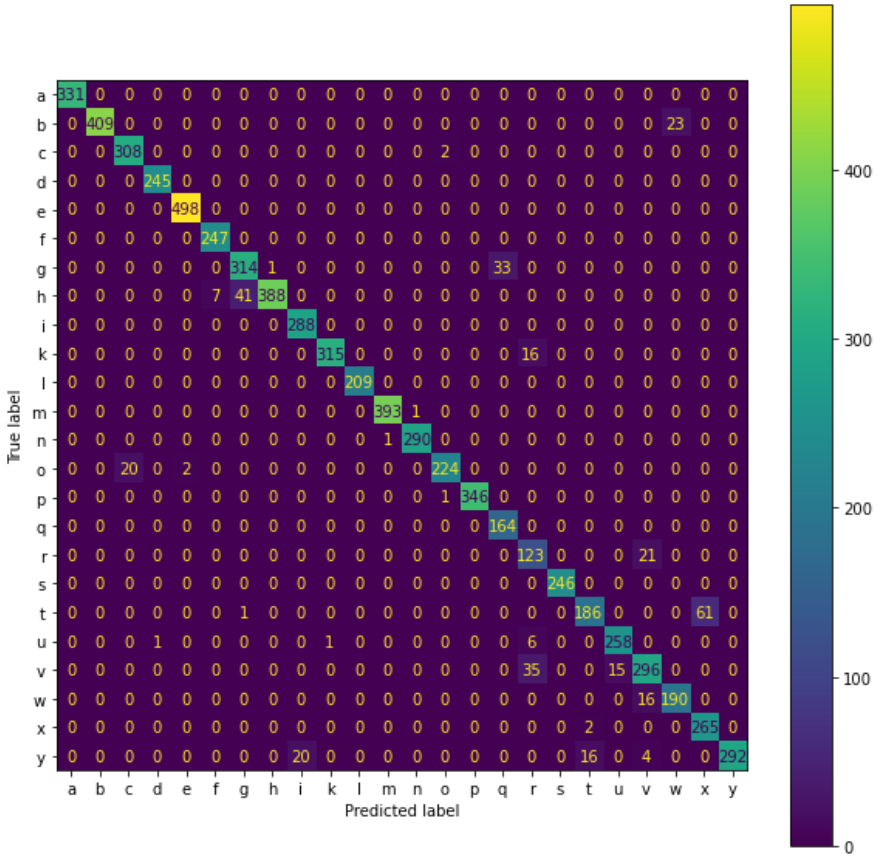
ii) F1 score of CNN on test data

```
# Accuracy of model on unseen data
print("Accuracy of model on unseen data")
print(f1_score(test_label, predicted_test_labels, average='weighted'))

import pandas as pd
from sklearn.metrics import confusion_matrix, ConfusionMatrixDisplay, classification_report, precision_recall_fscore_support
import matplotlib.pyplot as plt
```

```
Accuracy of model on unseen data
0.951951326196079
```

APPENDIX F: Results from Fitting the CNN Model on Test Data

i) Confusion matrix from fitting the CNN model on test data



ii) F1 score for each letter

	precision	recall	f1-score	support
a	1.00	1.00	1.00	331
b	1.00	0.95	0.97	432
c	0.94	0.99	0.97	310
d	1.00	1.00	1.00	245
e	1.00	1.00	1.00	498
f	0.97	1.00	0.99	247
g	0.88	0.90	0.89	348
h	1.00	0.89	0.94	436
i	0.94	1.00	0.97	288
k	1.00	0.95	0.97	331
l	1.00	1.00	1.00	209
m	1.00	1.00	1.00	394
n	1.00	1.00	1.00	291
o	0.99	0.91	0.95	246
p	1.00	1.00	1.00	347
q	0.83	1.00	0.91	164
r	0.68	0.85	0.76	144
s	1.00	1.00	1.00	246
t	0.91	0.75	0.82	248
u	0.95	0.97	0.96	266
v	0.88	0.86	0.87	346
w	0.89	0.92	0.91	206
x	0.81	0.99	0.89	267
y	1.00	0.88	0.94	332
accuracy			0.95	7172
macro avg	0.94	0.95	0.95	7172
weighted avg	0.96	0.95	0.95	7172

iii) Easiest letters to classify

Easiest letters to classify (by bests F1 score).

Letter	Precision	Recall	F1 score
a	1.0	1.0	1.0
l	1.0	1.0	1.0
s	1.0	1.0	1.0
p	1.0	1.0	1.0
m	1.0	1.0	1.0

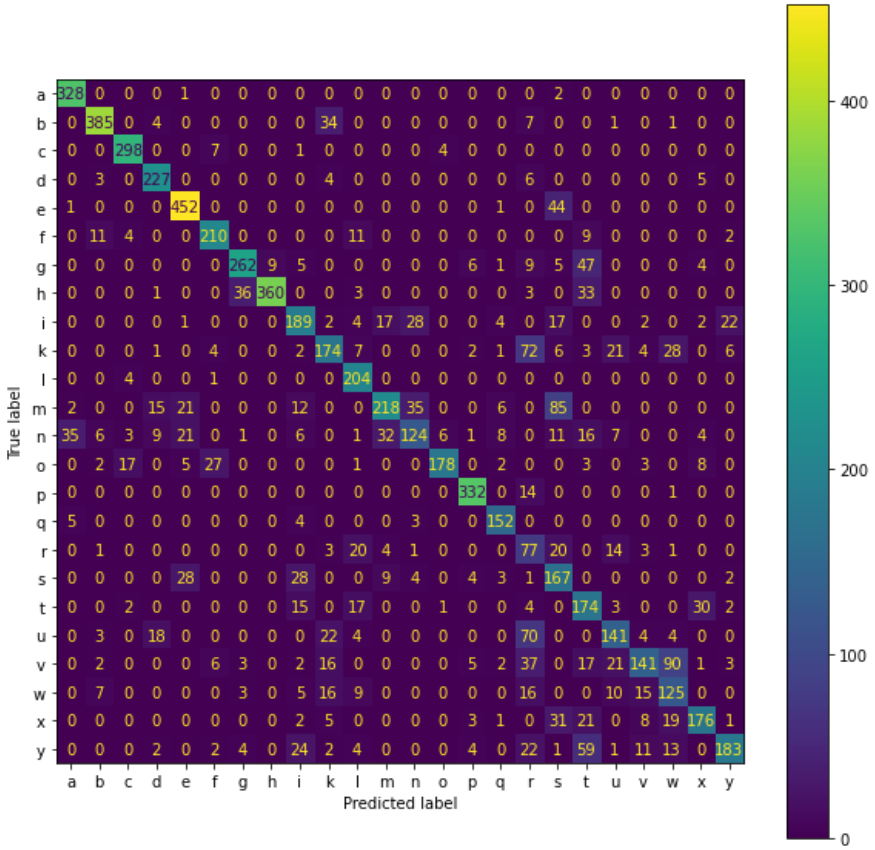
iv) Hardest letters to classify

Hardest letters to classify (by worsts F1 score).

Letter	Precision	Recall	F1 score
r	0.68	0.85	0.76
t	0.91	0.75	0.82
v	0.88	0.86	0.87
g	0.88	0.90	0.89
x	0.81	0.99	0.89

APPENDIX G: Results from Fitting the RFC Model on Test Data

i) Confusion matrix from fitting the RFC model on test data



ii) F1 score for each letter

	precision	recall	f1-score	support
a	0.88	0.99	0.93	331
b	0.92	0.89	0.90	432
c	0.91	0.96	0.93	310
d	0.82	0.93	0.87	245
e	0.85	0.91	0.88	498
f	0.82	0.85	0.83	247
g	0.85	0.75	0.80	348
h	0.98	0.83	0.89	436
i	0.64	0.66	0.65	288
k	0.63	0.53	0.57	331
l	0.72	0.98	0.83	209
m	0.78	0.55	0.65	394
n	0.64	0.43	0.51	291
o	0.94	0.72	0.82	246
p	0.93	0.96	0.94	347
q	0.84	0.93	0.88	164
r	0.23	0.53	0.32	144
s	0.43	0.68	0.53	246
t	0.46	0.70	0.55	248
u	0.64	0.53	0.58	266
v	0.74	0.41	0.53	346
w	0.44	0.61	0.51	206
x	0.77	0.66	0.71	267
y	0.83	0.55	0.66	332
accuracy			0.74	7172
macro avg	0.74	0.73	0.72	7172
weighted avg	0.77	0.74	0.74	7172

iii) Easiest letters to classify

```
Easiest letters to classify (by F1 score).
        Precision  Recall  F1 score
Letter
p            0.93    0.96      0.94
a            0.88    0.99      0.93
c            0.91    0.96      0.93
b            0.92    0.89      0.90
h            0.98    0.83      0.89
```

iv) Hardest letters to classify

```
Hardest letters to classify (by F1 score).
         Precision  Recall  F1 score
Letter
r             0.23    0.53      0.32
w             0.44    0.61      0.51
n             0.64    0.43      0.51
v             0.74    0.41      0.53
s             0.43    0.68      0.53
```

APPENDIX H: Top 5 Predictions

First Five Rows of Result

	0	1	2	3	4
0	19181012	21001713	17122400	11041504	03032118
1	2423142413	0816050812	1103021018	1006041310	1502111517
2	03201617	17171511	11211421	21112320	20011810
3	2203180819	2118122408	0504040521	1013130011	0112002200
4	210214	221405	171302	110404	201916