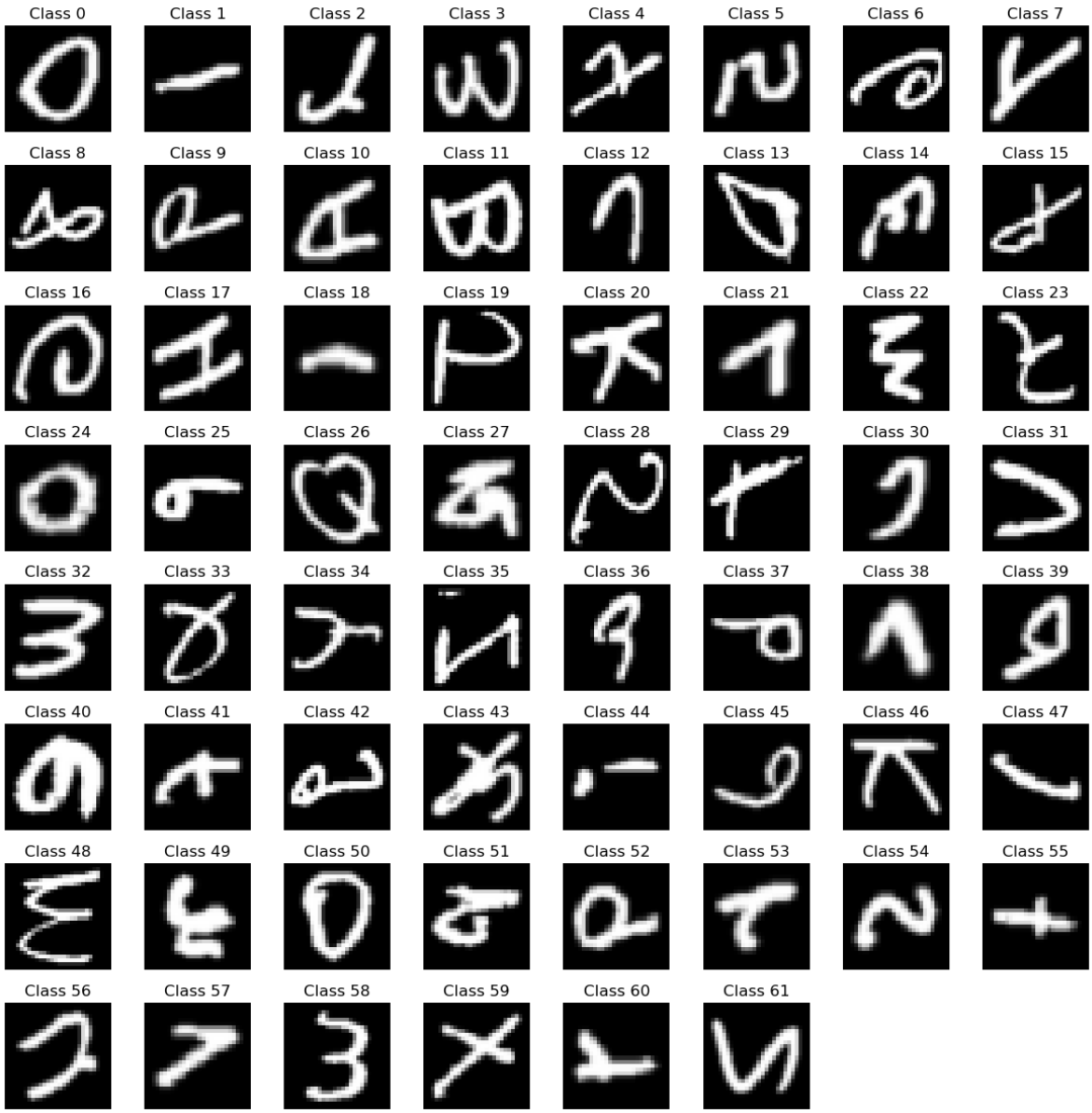
Assignment 2

In [8]:

```
import numpy as np
import pickle
import matplotlib.pyplot as plt
import time
from sklearn.decomposition import PCA
from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import train_test_split
from sklearn.svm import SVC
from sklearn.metrics import accuracy_score, precision_score, recall_score, confusic
from sklearn.model_selection import GridSearchCV

# Load the emnist_train.pkl file
with open('emnist_train.pkl', 'rb') as f:
    train_dict = pickle.load(f, encoding='bytes')

# Extract the data and labels arrays from the dictionary
train_data = train_dict['data']
train_labels = train_dict['labels']


# Load the emnist_test.pkl file
with open('emnist_test.pkl', 'rb') as f:
    test_dict = pickle.load(f, encoding='bytes')

# Extract the data and labels arrays from the dictionary
test_data = test_dict['data']
test_labels = test_dict['labels']

# Convert the data and labels arrays to NumPy arrays
train_data = np.array(train_data, dtype=np.float32)
train_labels = np.array(train_labels, dtype=np.int32)
test_data = np.array(test_data, dtype=np.float32)
test_labels = np.array(test_labels, dtype=np.int32)

# Define class names
# Create class names for all 62 classes
class_names = ['Class {}'.format(i) for i in range(62)]
#Plot a grid of EMNIST examples of a specified size."""

def plot_examples(data, pred, target, n_rows=5, n_cols=5):
    """Plot a grid of EMNIST examples of a specified size."""
    # Size figure depending on the size of the grid
    plt.figure(figsize=(n_cols * 1.2, n_rows * 2.0))

    for row in range(n_rows):
        for col in range(n_cols):
            # Get the next index of the image
            index = n_cols * row + col

            # Reshape the flattened image to its original shape
            image = data[index].reshape((28, 28))

            # Plot the image at the appropriate place in the grid
            plt.subplot(n_rows, n_cols, index + 1)
            plt.imshow(image, cmap="binary")
            plt.axis('off')
            plt.title(class_names[target[index]] + '\n' + '>>' + class_names[pred[ir
```

```
    plt.tight_layout()
    plt.show()

# Plot examples from different classes
fig, axes = plt.subplots(8, 8, figsize=(12, 12))
for i, ax in enumerate(axes.flat):
    if i < len(class_names):
        class_index = i  # Compute the class index
        # Get samples of the current class
        class_samples = train_data[train_labels == class_index]
        # Randomly select a sample from the class
        sample_index = np.random.randint(class_samples.shape[0])
        ax.imshow(class_samples[sample_index].reshape((28, 28)), cmap='gray')
        ax.axis('off')
        # Set the title as the class name
        ax.set_title(class_names[class_index])
    else:
        ax.axis('off')

plt.tight_layout()
plt.show()
```



MLP without Hyperparameter tuning

In [9]:

```
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, Flatten
from tensorflow.keras.utils import to_categorical
import time
import numpy as np
from sklearn.metrics import classification_report, confusion_matrix
from sklearn.metrics import accuracy_score, precision_score, recall_score

# Reshape the data to 2D for MLP
train_data = train_data.reshape((-1, 28*28))
test_data = test_data.reshape((-1, 28*28))

# Normalize the data
train_data = train_data / 255.0
test_data = test_data / 255.0

# Convert the labels to one-hot encoding
train_labels = to_categorical(train_labels)
test_labels = to_categorical(test_labels)

# Create the MLP model
model = Sequential([
    Dense(128, activation='relu', input_shape=(28*28,)),
    Dense(64, activation='relu'),
    Dense(len(train_labels[0]), activation='softmax')  # Number of output classes
])

# Compile the model
model.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['accuracy'

# Record the start time
start_time = time.time()

# Train the model
model.fit(train_data, train_labels, epochs=10, batch_size=128, validation_split=0.1)

# Record the end time
end_time = time.time()

# Calculate and print the training time
train_time = end_time - start_time
print('Training time:', train_time)

# Evaluate the model on test data
test_loss, test_acc = model.evaluate(test_data, test_labels)
print('Test accuracy:', test_acc)

# Generate predictions
preds = model.predict(test_data)
preds_classes = np.argmax(preds, axis=1)

# Convert back to integer format
test_labels_int = np.argmax(test_labels, axis=1)

# Generate and print the classification report
classification_report_results = classification_report(test_labels_int, preds_classes
print("Classification Report:")
print(classification_report_results)

# Generate and print the confusion matrix
confusion_matrix_results = confusion_matrix(test_labels_int, preds_classes)
print("Confusion Matrix:")
print(confusion_matrix_results)
```

```
accuracy = accuracy_score(test_labels_int, preds_classes)
precision = precision_score(test_labels_int, preds_classes, average='macro')
recall = recall_score(test_labels_int, preds_classes, average='macro')

print('Accuracy:', accuracy)
print('Precision:', precision)
print('Recall:', recall)
```

```
Epoch 1/10
704/704 [==============================] - 3s 3ms/step - loss: 1.3098 - accuracy: 0.
6502 - val_loss: 0.9197 - val_accuracy: 0.7303
Epoch 2/10
704/704 [==============================] - 2s 3ms/step - loss: 0.7866 - accuracy: 0.
7592 - val_loss: 0.7544 - val_accuracy: 0.7701
Epoch 3/10
704/704 [==============================] - 2s 3ms/step - loss: 0.6760 - accuracy: 0.
7859 - val_loss: 0.6960 - val_accuracy: 0.7813
Epoch 4/10
704/704 [==============================] - 2s 3ms/step - loss: 0.6141 - accuracy: 0.
8003 - val_loss: 0.6556 - val_accuracy: 0.7929
Epoch 5/10
704/704 [==============================] - 2s 3ms/step - loss: 0.5753 - accuracy: 0.
8100 - val_loss: 0.6289 - val_accuracy: 0.8002
Epoch 6/10
704/704 [==============================] - 3s 4ms/step - loss: 0.5444 - accuracy: 0.
8176 - val_loss: 0.6218 - val_accuracy: 0.7970
Epoch 7/10
704/704 [==============================] - 3s 4ms/step - loss: 0.5191 - accuracy: 0.
8245 - val_loss: 0.6010 - val_accuracy: 0.8048
Epoch 8/10
704/704 [==============================] - 2s 3ms/step - loss: 0.5027 - accuracy: 0.
8281 - val_loss: 0.6022 - val_accuracy: 0.8059
Epoch 9/10
704/704 [==============================] - 2s 3ms/step - loss: 0.4850 - accuracy: 0.
8336 - val_loss: 0.5930 - val_accuracy: 0.8042
Epoch 10/10
704/704 [==============================] - 2s 3ms/step - loss: 0.4680 - accuracy: 0.
8367 - val_loss: 0.5933 - val_accuracy: 0.8078
Training time: 23.027072191238403
625/625 [==============================] - 1s 2ms/step - loss: 0.5916 - accuracy: 0.
8069
Test accuracy: 0.8069499731063843
625/625 [==============================] - 1s 788us/step
Classification Report:
              precision    recall  f1-score   support

           0       0.64      0.81      0.72       976
           1       0.67      0.90      0.77      1023
           2       0.89      0.96      0.92      1003
           3       0.94      0.97      0.96      1035
           4       0.90      0.95      0.92       903
           5       0.80      0.92      0.85       928
           6       0.92      0.96      0.94       959
           7       0.97      0.97      0.97      1098
           8       0.93      0.93      0.93       941
           9       0.94      0.89      0.91       929
          10       0.84      0.89      0.87       170
          11       0.70      0.79      0.74       118
          12       0.73      0.77      0.75       316
          13       0.79      0.69      0.74       128
          14       0.90      0.80      0.85       162
          15       0.84      0.80      0.82       261
          16       0.71      0.53      0.61        60
          17       0.81      0.85      0.83        74
          18       0.62      0.39      0.48       350
          19       0.76      0.68      0.72        95
          20       0.65      0.56      0.60        71
          21       0.74      0.92      0.82       141
          22       0.74      0.86      0.80       273
          23       0.85      0.88      0.86       249
          24       0.65      0.49      0.56       741
          25       0.78      0.88      0.83       277
```

```
              26       0.58      0.72      0.64        67
              27       0.79      0.82      0.80       160
              28       0.81      0.68      0.74       624
              29       0.86      0.89      0.87       249
              30       0.76      0.74      0.75       342
              31       0.57      0.67      0.61       126
              32       0.69      0.83      0.76       125
              33       0.57      0.78      0.66        85
              34       0.88      0.58      0.70       141
              35       0.58      0.40      0.48        62
              36       0.80      0.77      0.78       278
              37       0.76      0.79      0.77       170
              38       0.32      0.20      0.25        90
              39       0.85      0.94      0.89       297
              40       0.91      0.95      0.93       709
              41       0.41      0.19      0.26        57
              42       0.29      0.36      0.32        97
              43       0.89      0.84      0.86       269
              44       0.59      0.36      0.44        76
              45       0.70      0.50      0.58        60
              46       0.57      0.60      0.58        83
              47       0.48      0.28      0.35       421
              48       0.42      0.15      0.22        75
              49       0.92      0.87      0.89       332
              50       0.00      0.00      0.00        79
              51       0.49      0.31      0.38        65
              52       0.39      0.21      0.27        68
              53       0.92      0.91      0.92       411
              54       0.40      0.03      0.05        79
              55       0.90      0.88      0.89       502
              56       0.33      0.34      0.34        89
              57       0.48      0.36      0.41        88
              58       0.73      0.66      0.69        93
              59       0.73      0.42      0.53       107
              60       0.41      0.43      0.42        60
              61       0.59      0.33      0.42        83

    accuracy                           0.81     20000
   macro avg       0.69      0.66      0.66     20000
weighted avg       0.80      0.81      0.80     20000

Confusion Matrix:
[[795   0   3 ...   0   0   0]
 [  0 919   4 ...   0   0   0]
 [  1   0 959 ...   1   0   2]
 ...
 [  0   2   0 ...  45   1   2]
 [  0   0   0 ...   1  26   0]
 [  0   0  28 ...   1   0  27]]
Accuracy: 0.80695
Precision: 0.6939762387610965
Recall: 0.6580846601742629
```

```
C:\Users\Haley\anaconda3\lib\site-packages\sklearn\metrics\_classification.py:1318:
UndefinedMetricWarning: Precision and F-score are ill-defined and being set to 0.0 i
n labels with no predicted samples. Use `zero_division` parameter to control this be
havior.
  _warn_prf(average, modifier, msg_start, len(result))
C:\Users\Haley\anaconda3\lib\site-packages\sklearn\metrics\_classification.py:1318:
UndefinedMetricWarning: Precision and F-score are ill-defined and being set to 0.0 i
n labels with no predicted samples. Use `zero_division` parameter to control this be
havior.
  _warn_prf(average, modifier, msg_start, len(result))
C:\Users\Haley\anaconda3\lib\site-packages\sklearn\metrics\_classification.py:1318:
UndefinedMetricWarning: Precision and F-score are ill-defined and being set to 0.0 i
n labels with no predicted samples. Use `zero_division` parameter to control this be
havior.
  _warn_prf(average, modifier, msg_start, len(result))
C:\Users\Haley\anaconda3\lib\site-packages\sklearn\metrics\_classification.py:1318:
UndefinedMetricWarning: Precision is ill-defined and being set to 0.0 in labels with
no predicted samples. Use `zero_division` parameter to control this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
```

MLP with Hyperparameter tuning

In [11]:

```
from sklearn.model_selection import GridSearchCV
from tensorflow.keras.wrappers.scikit_learn import KerasClassifier
# Define the model building function required for KerasClassifier
def create_model(n_hidden_layers=2, n_hidden_neurons=50, activation_function='relu'
    model = Sequential()
    model.add(Flatten(input_shape=(28*28,)))
    for _ in range(n_hidden_layers):
        model.add(Dense(n_hidden_neurons, activation=activation_function))
    model.add(Dense(len(train_labels[0]), activation='softmax'))  # Number of output
    model.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['accur
    return model

# Instantiate KerasClassifier with the new model
keras_classifier = KerasClassifier(build_fn=create_model, epochs=10, batch_size=128,
# Define the parameter grid
param_grid = {
    "n_hidden_neurons": [50, 100, 200],
    "activation_function": ["relu", "sigmoid", "tanh"]
}

# Instantiate GridSearchCV
grid = GridSearchCV(estimator=keras_classifier, param_grid=param_grid, cv=3)

# Perform hyperparameter tuning / model selection
grid_result = grid.fit(train_data, train_labels)

# Print the best score and best parameters
print(grid_result.best_score_, grid_result.best_params_)
```

```
C:\Users\Haley\AppData\Local\Temp\ipykernel_10312\3810704986.py:14: DeprecationWarni
ng: KerasClassifier is deprecated, use Sci-Keras (https://github.com/adriangb/sciker
as) instead. See https://www.adriangb.com/scikeras/stable/migration.html for help mi
grating.
  keras_classifier = KerasClassifier(build_fn=create_model, epochs=10, batch_size=12
8, verbose=0)
0.8090599973996481 {'activation_function': 'relu', 'n_hidden_neurons': 200}
```

In [7]:

```
# Evaluate the model on test data
test_loss, test_acc = grid_result.best_estimator_.model.evaluate(test_data, test_lab
```

```
print('Test accuracy:', test_acc)

# Generate predictions
preds = grid_result.best_estimator_.model.predict(test_data)
preds_classes = np.argmax(preds, axis=1)

# Convert back to integer format
test_labels_int = np.argmax(test_labels, axis=1)

# Generate and print the classification report
classification_report_results = classification_report(test_labels_int, preds_classes
print("Classification Report:")
print(classification_report_results)

# Generate and print the confusion matrix
confusion_matrix_results = confusion_matrix(test_labels_int, preds_classes)
print("Confusion Matrix:")
print(confusion_matrix_results)

accuracy = accuracy_score(test_labels_int, preds_classes)
precision = precision_score(test_labels_int, preds_classes, average='macro')
recall = recall_score(test_labels_int, preds_classes, average='macro')

print('Accuracy:', accuracy)
print('Precision:', precision)
print('Recall:', recall)
```

```
625/625 [==============================] - 1s 971us/step - loss: 0.5851 - accuracy:
0.8152
Test accuracy: 0.8151500225067139
625/625 [==============================] - 1s 946us/step
Classification Report:
              precision    recall  f1-score   support

           0       0.67      0.76      0.72       976
           1       0.70      0.82      0.76      1023
           2       0.96      0.90      0.92      1003
           3       0.94      0.98      0.96      1035
           4       0.90      0.96      0.93       903
           5       0.86      0.90      0.88       928
           6       0.92      0.97      0.94       959
           7       0.98      0.97      0.97      1098
           8       0.96      0.93      0.94       941
           9       0.89      0.98      0.93       929
          10       0.85      0.90      0.88       170
          11       0.67      0.81      0.73       118
          12       0.71      0.88      0.79       316
          13       0.87      0.76      0.81       128
          14       0.91      0.88      0.90       162
          15       0.81      0.79      0.80       261
          16       0.65      0.87      0.74        60
          17       0.64      0.96      0.77        74
          18       0.65      0.36      0.46       350
          19       0.76      0.75      0.75        95
          20       0.52      0.82      0.63        71
          21       0.83      0.93      0.88       141
          22       0.76      0.86      0.81       273
          23       0.92      0.82      0.87       249
          24       0.61      0.59      0.60       741
          25       0.76      0.87      0.81       277
          26       0.76      0.75      0.75        67
          27       0.93      0.78      0.85       160
          28       0.79      0.81      0.80       624
          29       0.94      0.83      0.88       249
          30       0.74      0.92      0.82       342
          31       0.62      0.65      0.63       126
          32       0.85      0.58      0.69       125
          33       0.66      0.60      0.63        85
          34       0.76      0.64      0.69       141
          35       0.38      0.84      0.52        62
          36       0.88      0.75      0.81       278
          37       0.93      0.61      0.73       170
          38       0.40      0.11      0.17        90
          39       0.91      0.92      0.92       297
          40       0.92      0.96      0.94       709
          41       0.34      0.32      0.33        57
          42       0.54      0.21      0.30        97
          43       0.91      0.85      0.88       269
          44       1.00      0.22      0.37        76
          45       0.53      0.67      0.59        60
          46       0.51      0.25      0.34        83
          47       0.38      0.38      0.38       421
          48       0.29      0.12      0.17        75
          49       0.91      0.89      0.90       332
          50       0.00      0.00      0.00        79
          51       0.45      0.26      0.33        65
          52       0.44      0.22      0.29        68
          53       0.92      0.91      0.92       411
          54       0.00      0.00      0.00        79
          55       0.79      0.96      0.86       502
          56       0.47      0.10      0.17        89
```

```
           57       0.52      0.34      0.41        88
           58       0.59      0.86      0.70        93
           59       0.67      0.58      0.62       107
           60       0.38      0.25      0.30        60
           61       0.54      0.36      0.43        83

    accuracy                           0.82     20000
   macro avg       0.70      0.67      0.67     20000
weighted avg       0.81      0.82      0.80     20000

Confusion Matrix:
[[744   0   0 ...   0   0   0]
 [  0 840   2 ...   0   0   0]
 [  1   0 899 ...   0   0  16]
 ...
 [  0   1   0 ...  62   1   4]
 [  0   0   0 ...   0  15   0]
 [  0   0  11 ...   1   0  30]]
Accuracy: 0.81515
Precision: 0.6992260869018739
Recall: 0.6688337843850689
```

MLP Experiments with varying n_hidden_neurons_values and activation_function

In [12]:

```python
#vary n_hidden_neurons_values 10 trials with activation_function='relu'
n_hidden_neurons_values = np.linspace(50, 200, num=10, dtype=int)
n_iterations = 1

best_score = 0
best_params = {}

for n_hidden_neurons in n_hidden_neurons_values:
    for i in range(n_iterations):
        start_time = time.time()

        model = create_model(n_hidden_neurons=n_hidden_neurons, activation_function=
        history = model.fit(train_data, train_labels, epochs=10, batch_size=128, val

        # Generate predictions
        preds = model.predict(test_data)
        preds_classes = np.argmax(preds, axis=1)

        # Convert back to integer format
        test_labels_int = np.argmax(test_labels, axis=1)

        accuracy = accuracy_score(test_labels_int, preds_classes)
        precision = precision_score(test_labels_int, preds_classes, average='macro')
        recall = recall_score(test_labels_int, preds_classes, average='macro')

        end_time = time.time()
        elapsed_time = end_time - start_time

        print(f'Iteration {i+1}, n_hidden_neurons: {n_hidden_neurons}')
        print('Accuracy:', accuracy)
        print('Precision:', precision)
        print('Recall:', recall)
        print('Elapsed time:', elapsed_time, 'seconds')
```

```
        if accuracy > best_score:
            best_score = accuracy
            best_params = {'n_hidden_neurons': n_hidden_neurons}

print('Best parameters:', best_params, 'with highest accuracy:', best_score)
```

```
625/625 [==============================] - 0s 673us/step
Iteration 1, n_hidden_neurons: 50
Accuracy: 0.78675
Precision: 0.6636166882125001
Recall: 0.6185079213946566
Elapsed time: 12.830229997634888 seconds
C:\Users\Haley\anaconda3\lib\site-packages\sklearn\metrics\_classification.py:1318: UndefinedMetricWarning: Precision is ill-defined and being set to 0.0 in labels with no predicted samples. Use `zero_division` parameter to control this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
625/625 [==============================] - 1s 784us/step
Iteration 1, n_hidden_neurons: 66
Accuracy: 0.79215
Precision: 0.665171854438555
Recall: 0.634355040173728
Elapsed time: 13.400078296661377 seconds
C:\Users\Haley\anaconda3\lib\site-packages\sklearn\metrics\_classification.py:1318: UndefinedMetricWarning: Precision is ill-defined and being set to 0.0 in labels with no predicted samples. Use `zero_division` parameter to control this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
625/625 [==============================] - 1s 795us/step
Iteration 1, n_hidden_neurons: 83
Accuracy: 0.80295
Precision: 0.6812350108740798
Recall: 0.6428924862446574
Elapsed time: 13.961905241012573 seconds
625/625 [==============================] - 1s 956us/step
Iteration 1, n_hidden_neurons: 100
Accuracy: 0.80245
Precision: 0.6863373685561066
Recall: 0.6486132174422466
Elapsed time: 17.535341262817383 seconds
C:\Users\Haley\anaconda3\lib\site-packages\sklearn\metrics\_classification.py:1318: UndefinedMetricWarning: Precision is ill-defined and being set to 0.0 in labels with no predicted samples. Use `zero_division` parameter to control this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
625/625 [==============================] - 1s 746us/step
Iteration 1, n_hidden_neurons: 116
Accuracy: 0.80855
Precision: 0.693858153776578
Recall: 0.6528442668639657
Elapsed time: 18.015787363052368 seconds
C:\Users\Haley\anaconda3\lib\site-packages\sklearn\metrics\_classification.py:1318: UndefinedMetricWarning: Precision is ill-defined and being set to 0.0 in labels with no predicted samples. Use `zero_division` parameter to control this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
625/625 [==============================] - 1s 1ms/step
C:\Users\Haley\anaconda3\lib\site-packages\sklearn\metrics\_classification.py:1318: UndefinedMetricWarning: Precision is ill-defined and being set to 0.0 in labels with no predicted samples. Use `zero_division` parameter to control this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
```

```
Iteration 1, n_hidden_neurons: 133
Accuracy: 0.8125
Precision: 0.6981965852330315
Recall: 0.6641398118730842
Elapsed time: 28.224685192108154 seconds
625/625 [==============================] - 1s 1ms/step
Iteration 1, n_hidden_neurons: 150
Accuracy: 0.81245
Precision: 0.701345135716705
Recall: 0.6544264604049093
Elapsed time: 32.15014863014221 seconds
C:\Users\Haley\anaconda3\lib\site-packages\sklearn\metrics\_classification.py:1318:
UndefinedMetricWarning: Precision is ill-defined and being set to 0.0 in labels with
no predicted samples. Use `zero_division` parameter to control this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
625/625 [==============================] - 1s 1ms/step
Iteration 1, n_hidden_neurons: 166
Accuracy: 0.8147
Precision: 0.6929582315307035
Recall: 0.6727570945268735
Elapsed time: 25.518718242645264 seconds
C:\Users\Haley\anaconda3\lib\site-packages\sklearn\metrics\_classification.py:1318:
UndefinedMetricWarning: Precision is ill-defined and being set to 0.0 in labels with
no predicted samples. Use `zero_division` parameter to control this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
625/625 [==============================] - 1s 1ms/step
Iteration 1, n_hidden_neurons: 183
Accuracy: 0.8154
Precision: 0.6939548768308034
Recall: 0.6648246034446986
Elapsed time: 25.49815011024475 seconds
625/625 [==============================] - 1s 1ms/step
Iteration 1, n_hidden_neurons: 200
Accuracy: 0.8126
Precision: 0.7020516458116924
Recall: 0.6721542609509867
Elapsed time: 26.66808271408081 seconds
Best parameters: {'n_hidden_neurons': 183} with highest accuracy: 0.8154
C:\Users\Haley\anaconda3\lib\site-packages\sklearn\metrics\_classification.py:1318:
UndefinedMetricWarning: Precision is ill-defined and being set to 0.0 in labels with
no predicted samples. Use `zero_division` parameter to control this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
```

In [13]:

```
#vary n_hidden_neurons_values 10 trials with activation_function='sigmoid'
n_hidden_neurons_values = np.linspace(50, 200, num=10, dtype=int)
n_iterations = 1

best_score = 0
best_params = {}

for n_hidden_neurons in n_hidden_neurons_values:
    for i in range(n_iterations):
        start_time = time.time()

        model = create_model(n_hidden_neurons=n_hidden_neurons, activation_function=
        history = model.fit(train_data, train_labels, epochs=10, batch_size=128, val

        # Generate predictions
        preds = model.predict(test_data)
        preds_classes = np.argmax(preds, axis=1)

        # Convert back to integer format
        test_labels_int = np.argmax(test_labels, axis=1)
```

```
        accuracy = accuracy_score(test_labels_int, preds_classes)
        precision = precision_score(test_labels_int, preds_classes, average='macro')
        recall = recall_score(test_labels_int, preds_classes, average='macro')

        end_time = time.time()
        elapsed_time = end_time - start_time

        print(f'Iteration {i+1}, n_hidden_neurons: {n_hidden_neurons}')
        print('Accuracy:', accuracy)
        print('Precision:', precision)
        print('Recall:', recall)
        print('Elapsed time:', elapsed_time, 'seconds')

        if accuracy > best_score:
            best_score = accuracy
            best_params = {'n_hidden_neurons': n_hidden_neurons}

print('Best parameters:', best_params, 'with highest accuracy:', best_score)
```

```
625/625 [==============================] - 1s 850us/step
Iteration 1, n_hidden_neurons: 50
Accuracy: 0.75675
Precision: 0.6096183510782207
Recall: 0.5539012216941204
Elapsed time: 16.84760856628418 seconds
C:\Users\Haley\anaconda3\lib\site-packages\sklearn\metrics\_classification.py:1318: UndefinedMetricWarning: Precision is ill-defined and being set to 0.0 in labels with no predicted samples. Use `zero_division` parameter to control this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
625/625 [==============================] - 1s 760us/step
Iteration 1, n_hidden_neurons: 66
Accuracy: 0.76995
Precision: 0.6250326141981106
Recall: 0.5751443126918576
Elapsed time: 16.511566877365112 seconds
C:\Users\Haley\anaconda3\lib\site-packages\sklearn\metrics\_classification.py:1318: UndefinedMetricWarning: Precision is ill-defined and being set to 0.0 in labels with no predicted samples. Use `zero_division` parameter to control this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
625/625 [==============================] - 1s 747us/step
Iteration 1, n_hidden_neurons: 83
Accuracy: 0.78395
Precision: 0.6599860845206879
Recall: 0.599468787446257
Elapsed time: 16.601797342300415 seconds
C:\Users\Haley\anaconda3\lib\site-packages\sklearn\metrics\_classification.py:1318: UndefinedMetricWarning: Precision is ill-defined and being set to 0.0 in labels with no predicted samples. Use `zero_division` parameter to control this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
625/625 [==============================] - 1s 889us/step
Iteration 1, n_hidden_neurons: 100
Accuracy: 0.7901
Precision: 0.6757952152924077
Recall: 0.6182493251699354
Elapsed time: 18.88583779335022 seconds
C:\Users\Haley\anaconda3\lib\site-packages\sklearn\metrics\_classification.py:1318: UndefinedMetricWarning: Precision is ill-defined and being set to 0.0 in labels with no predicted samples. Use `zero_division` parameter to control this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
```

```
625/625 [==============================] - 1s 1ms/step
Iteration 1, n_hidden_neurons: 116
Accuracy: 0.79835
Precision: 0.6875708204894133
Recall: 0.6297269322283052
Elapsed time: 20.639202117919922 seconds
C:\Users\Haley\anaconda3\lib\site-packages\sklearn\metrics\_classification.py:1318: UndefinedMetricWarning: Precision is ill-defined and being set to 0.0 in labels with no predicted samples. Use `zero_division` parameter to control this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
625/625 [==============================] - 1s 818us/step
Iteration 1, n_hidden_neurons: 133
Accuracy: 0.80285
Precision: 0.6873893240335153
Recall: 0.6400336714468474
Elapsed time: 19.75173306465149 seconds
C:\Users\Haley\anaconda3\lib\site-packages\sklearn\metrics\_classification.py:1318: UndefinedMetricWarning: Precision is ill-defined and being set to 0.0 in labels with no predicted samples. Use `zero_division` parameter to control this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
625/625 [==============================] - 1s 1ms/step
Iteration 1, n_hidden_neurons: 150
Accuracy: 0.8053
Precision: 0.68870807018782
Recall: 0.6350585191371019
Elapsed time: 22.533874034881592 seconds
C:\Users\Haley\anaconda3\lib\site-packages\sklearn\metrics\_classification.py:1318: UndefinedMetricWarning: Precision is ill-defined and being set to 0.0 in labels with no predicted samples. Use `zero_division` parameter to control this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
625/625 [==============================] - 1s 979us/step
Iteration 1, n_hidden_neurons: 166
Accuracy: 0.81055
Precision: 0.702978613200429
Recall: 0.6487298083366403
Elapsed time: 19.90397357940674 seconds
C:\Users\Haley\anaconda3\lib\site-packages\sklearn\metrics\_classification.py:1318: UndefinedMetricWarning: Precision is ill-defined and being set to 0.0 in labels with no predicted samples. Use `zero_division` parameter to control this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
625/625 [==============================] - 1s 863us/step
Iteration 1, n_hidden_neurons: 183
Accuracy: 0.80925
Precision: 0.6879398439059695
Recall: 0.6429953671238094
Elapsed time: 19.76059055328369 seconds
C:\Users\Haley\anaconda3\lib\site-packages\sklearn\metrics\_classification.py:1318: UndefinedMetricWarning: Precision is ill-defined and being set to 0.0 in labels with no predicted samples. Use `zero_division` parameter to control this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
625/625 [==============================] - 1s 1ms/step
Iteration 1, n_hidden_neurons: 200
Accuracy: 0.81155
Precision: 0.699349335494755
Recall: 0.6540769711868644
Elapsed time: 31.357375860214233 seconds
Best parameters: {'n_hidden_neurons': 200} with highest accuracy: 0.81155
C:\Users\Haley\anaconda3\lib\site-packages\sklearn\metrics\_classification.py:1318: UndefinedMetricWarning: Precision is ill-defined and being set to 0.0 in labels with no predicted samples. Use `zero_division` parameter to control this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
```

In [117…

```
#vary n_hidden_neurons_values 10 trials with activation_function='tanh'
n_hidden_neurons_values = np.linspace(50, 200, num=10, dtype=int)
n_iterations = 1

best_score = 0
best_params = {}

for n_hidden_neurons in n_hidden_neurons_values:
    for i in range(n_iterations):
        start_time = time.time()

        model = create_model(n_hidden_neurons=n_hidden_neurons, activation_function=
        history = model.fit(train_data, train_labels, epochs=10, batch_size=128, val

        # Generate predictions
        preds = model.predict(test_data)
        preds_classes = np.argmax(preds, axis=1)

        # Convert back to integer format
        test_labels_int = np.argmax(test_labels, axis=1)

        accuracy = accuracy_score(test_labels_int, preds_classes)
        precision = precision_score(test_labels_int, preds_classes, average='macro')
        recall = recall_score(test_labels_int, preds_classes, average='macro')

        end_time = time.time()
        elapsed_time = end_time - start_time

        print(f'Iteration {i+1}, n_hidden_neurons: {n_hidden_neurons}')
        print('Accuracy:', accuracy)
        print('Precision:', precision)
        print('Recall:', recall)
        print('Elapsed time:', elapsed_time, 'seconds')

        if accuracy > best_score:
            best_score = accuracy
            best_params = {'n_hidden_neurons': n_hidden_neurons}

print('Best parameters:', best_params, 'with highest accuracy:', best_score)
```

```
625/625 [==============================] - 0s 726us/step
Iteration 1, n_hidden_neurons: 50
Accuracy: 0.78145
Precision: 0.6459470446428905
Recall: 0.6050419646758668
Elapsed time: 12.479187726974487 seconds
C:\Users\Haley\anaconda3\lib\site-packages\sklearn\metrics\_classification.py:1318:
UndefinedMetricWarning: Precision is ill-defined and being set to 0.0 in labels with
no predicted samples. Use `zero_division` parameter to control this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
625/625 [==============================] - 1s 763us/step
Iteration 1, n_hidden_neurons: 66
Accuracy: 0.79365
Precision: 0.6638872597386016
Recall: 0.625086623732739
Elapsed time: 13.616044521331787 seconds
C:\Users\Haley\anaconda3\lib\site-packages\sklearn\metrics\_classification.py:1318:
UndefinedMetricWarning: Precision is ill-defined and being set to 0.0 in labels with
no predicted samples. Use `zero_division` parameter to control this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
```

```
625/625 [==============================] - 1s 748us/step
Iteration 1, n_hidden_neurons: 83
Accuracy: 0.7999
Precision: 0.6921451765570633
Recall: 0.6361593499114829
Elapsed time: 16.139023542404175 seconds
```

```
C:\Users\Haley\anaconda3\lib\site-packages\sklearn\metrics\_classification.py:1318:
UndefinedMetricWarning: Precision is ill-defined and being set to 0.0 in labels with
no predicted samples. Use `zero_division` parameter to control this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
```

```
625/625 [==============================] - 1s 763us/step
Iteration 1, n_hidden_neurons: 100
Accuracy: 0.8033
Precision: 0.683118558830502
Recall: 0.6559164557710065
Elapsed time: 15.396340608596802 seconds
```

```
C:\Users\Haley\anaconda3\lib\site-packages\sklearn\metrics\_classification.py:1318:
UndefinedMetricWarning: Precision is ill-defined and being set to 0.0 in labels with
no predicted samples. Use `zero_division` parameter to control this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
```

```
625/625 [==============================] - 1s 812us/step
Iteration 1, n_hidden_neurons: 116
Accuracy: 0.8076
Precision: 0.6872544888282485
Recall: 0.6566628466836997
Elapsed time: 17.027933835983276 seconds
```

```
C:\Users\Haley\anaconda3\lib\site-packages\sklearn\metrics\_classification.py:1318:
UndefinedMetricWarning: Precision is ill-defined and being set to 0.0 in labels with
no predicted samples. Use `zero_division` parameter to control this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
```

```
625/625 [==============================] - 1s 747us/step
Iteration 1, n_hidden_neurons: 133
Accuracy: 0.815
Precision: 0.7000799085887366
Recall: 0.661848493854145
Elapsed time: 16.12827205657959 seconds
```

```
C:\Users\Haley\anaconda3\lib\site-packages\sklearn\metrics\_classification.py:1318:
UndefinedMetricWarning: Precision is ill-defined and being set to 0.0 in labels with
no predicted samples. Use `zero_division` parameter to control this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
```

```
625/625 [==============================] - 1s 816us/step
Iteration 1, n_hidden_neurons: 150
Accuracy: 0.80685
Precision: 0.6904438187115848
Recall: 0.6547353161267282
Elapsed time: 17.236314058303833 seconds
```

```
C:\Users\Haley\anaconda3\lib\site-packages\sklearn\metrics\_classification.py:1318:
UndefinedMetricWarning: Precision is ill-defined and being set to 0.0 in labels with
no predicted samples. Use `zero_division` parameter to control this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
```

```
625/625 [==============================] - 1s 817us/step
Iteration 1, n_hidden_neurons: 166
Accuracy: 0.81215
Precision: 0.6900095237354827
Recall: 0.6676738368377416
Elapsed time: 15.4945068359375 seconds
```

```
C:\Users\Haley\anaconda3\lib\site-packages\sklearn\metrics\_classification.py:1318:
UndefinedMetricWarning: Precision is ill-defined and being set to 0.0 in labels with
no predicted samples. Use `zero_division` parameter to control this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
```

```
625/625 [==============================] - 1s 1ms/step
Iteration 1, n_hidden_neurons: 183
Accuracy: 0.8155
Precision: 0.7003523276479232
Recall: 0.6663832058986104
Elapsed time: 24.31060528755188 seconds
625/625 [==============================] - 1s 1ms/step
Iteration 1, n_hidden_neurons: 200
Accuracy: 0.80835
Precision: 0.7131020251544583
Recall: 0.664274854221722
Elapsed time: 26.549508571624756 seconds
Best parameters: {'n_hidden_neurons': 183} with highest accuracy: 0.8155
C:\Users\Haley\anaconda3\lib\site-packages\sklearn\metrics\_classification.py:1318:
UndefinedMetricWarning: Precision is ill-defined and being set to 0.0 in labels with
no predicted samples. Use `zero_division` parameter to control this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
```

CNN before tuning

rerun the first block and continue

In [17]:

```
import numpy as np
import pickle
import time
import tensorflow as tf
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Conv2D, MaxPooling2D, Dense, Flatten
from tensorflow.keras.utils import to_categorical

# Reshape the data for CNN
train_data = train_data.reshape((-1, 28, 28, 1))
test_data = test_data.reshape((-1, 28, 28, 1))

# Normalize the data
train_data = train_data / 255.0
test_data = test_data / 255.0

# Convert the labels to one-hot encoding
train_labels = to_categorical(train_labels)
test_labels = to_categorical(test_labels)


# Create the CNN model
model = Sequential([
    Conv2D(32, (3, 3), activation='relu', input_shape=(28, 28, 1)),
    MaxPooling2D((2, 2)),
    Conv2D(64, (3, 3), activation='relu'),
    MaxPooling2D((2, 2)),
    Flatten(),
    Dense(128, activation='relu'),
    Dense(len(train_labels[0]), activation='softmax')  # Number of output classes
])


# Compile the model
model.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['accuracy'

# Record the start time
start_time = time.time()
```

```
# Train the model
model.fit(train_data, train_labels, epochs=10, batch_size=128, validation_split=0.1)

# Record the end time
end_time = time.time()

# Calculate and print the training time
train_time = end_time - start_time
print('Training time:', train_time)

# Evaluate the model on test data
test_loss, test_acc = model.evaluate(test_data, test_labels)
print('Test accuracy:', test_acc)

# Generate predictions
preds = model.predict(test_data)
preds_classes = np.argmax(preds, axis=1)

# Convert back to integer format
test_labels_int = np.argmax(test_labels, axis=1)

# Generate and print the classification report
classification_report_results = classification_report(test_labels_int, preds_classes
print("Classification Report:")
print(classification_report_results)

# Generate and print the confusion matrix
confusion_matrix_results = confusion_matrix(test_labels_int, preds_classes)
print("Confusion Matrix:")
print(confusion_matrix_results)

accuracy = accuracy_score(test_labels_int, preds_classes)
precision = precision_score(test_labels_int, preds_classes, average='macro')
recall = recall_score(test_labels_int, preds_classes, average='macro')

print('Accuracy:', accuracy)
print('Precision:', precision)
print('Recall:', recall)
```

```
Epoch 1/10
704/704 [==============================] - 33s 46ms/step - loss: 0.9589 - accuracy:
0.7249 - val_loss: 0.6323 - val_accuracy: 0.7913
Epoch 2/10
704/704 [==============================] - 37s 52ms/step - loss: 0.5433 - accuracy:
0.8171 - val_loss: 0.5152 - val_accuracy: 0.8240
Epoch 3/10
704/704 [==============================] - 32s 45ms/step - loss: 0.4755 - accuracy:
0.8356 - val_loss: 0.4770 - val_accuracy: 0.8405
Epoch 4/10
704/704 [==============================] - 32s 45ms/step - loss: 0.4358 - accuracy:
0.8463 - val_loss: 0.4614 - val_accuracy: 0.8388
Epoch 5/10
704/704 [==============================] - 32s 45ms/step - loss: 0.4068 - accuracy:
0.8536 - val_loss: 0.4524 - val_accuracy: 0.8424
Epoch 6/10
704/704 [==============================] - 31s 45ms/step - loss: 0.3845 - accuracy:
0.8593 - val_loss: 0.4441 - val_accuracy: 0.8439
Epoch 7/10
704/704 [==============================] - 34s 48ms/step - loss: 0.3643 - accuracy:
0.8652 - val_loss: 0.4361 - val_accuracy: 0.8462
Epoch 8/10
704/704 [==============================] - 35s 49ms/step - loss: 0.3446 - accuracy:
0.8701 - val_loss: 0.4356 - val_accuracy: 0.8503
Epoch 9/10
704/704 [==============================] - 33s 47ms/step - loss: 0.3297 - accuracy:
0.8746 - val_loss: 0.4406 - val_accuracy: 0.8449
Epoch 10/10
704/704 [==============================] - 34s 48ms/step - loss: 0.3119 - accuracy:
0.8811 - val_loss: 0.4359 - val_accuracy: 0.8513
Training time: 332.20822501182556
625/625 [==============================] - 2s 4ms/step - loss: 0.4544 - accuracy: 0.
8459
Test accuracy: 0.8458999991416931
625/625 [==============================] - 2s 3ms/step
Classification Report:
              precision    recall  f1-score   support

           0       0.66      0.81      0.73       976
           1       0.68      0.92      0.78      1023
           2       0.92      0.97      0.95      1003
           3       0.99      0.99      0.99      1035
           4       0.96      0.95      0.96       903
           5       0.94      0.92      0.93       928
           6       0.97      0.97      0.97       959
           7       0.99      0.98      0.98      1098
           8       0.98      0.96      0.97       941
           9       0.93      0.96      0.95       929
          10       0.93      0.94      0.93       170
          11       0.83      0.90      0.87       118
          12       0.75      0.80      0.77       316
          13       0.85      0.88      0.87       128
          14       0.93      0.92      0.93       162
          15       0.81      0.90      0.86       261
          16       0.80      0.88      0.84        60
          17       0.86      0.97      0.91        74
          18       0.64      0.46      0.54       350
          19       0.83      0.73      0.78        95
          20       0.62      0.77      0.69        71
          21       0.86      0.93      0.89       141
          22       0.79      0.81      0.80       273
          23       0.92      0.97      0.95       249
          24       0.64      0.52      0.58       741
          25       0.82      0.87      0.85       277
```

```
           26       0.85      0.85      0.85        67
           27       0.92      0.89      0.91       160
           28       0.79      0.93      0.86       624
           29       0.90      0.91      0.90       249
           30       0.76      0.89      0.82       342
           31       0.64      0.65      0.64       126
           32       0.75      0.77      0.76       125
           33       0.64      0.69      0.67        85
           34       0.79      0.67      0.73       141
           35       0.62      0.45      0.52        62
           36       0.89      0.85      0.87       278
           37       0.87      0.84      0.85       170
           38       0.28      0.19      0.23        90
           39       0.97      0.96      0.97       297
           40       0.95      0.99      0.97       709
           41       0.40      0.07      0.12        57
           42       0.52      0.52      0.52        97
           43       0.92      0.91      0.91       269
           44       0.67      0.38      0.49        76
           45       0.61      0.57      0.59        60
           46       0.72      0.57      0.64        83
           47       0.54      0.25      0.34       421
           48       0.34      0.28      0.31        75
           49       0.89      0.93      0.91       332
           50       0.00      0.00      0.00        79
           51       0.50      0.37      0.42        65
           52       0.47      0.24      0.31        68
           53       0.95      0.94      0.94       411
           54       0.33      0.01      0.02        79
           55       0.93      0.93      0.93       502
           56       0.37      0.21      0.27        89
           57       0.47      0.40      0.43        88
           58       0.70      0.67      0.69        93
           59       0.70      0.60      0.64       107
           60       0.37      0.42      0.39        60
           61       0.53      0.29      0.38        83

    accuracy                           0.85     20000
   macro avg       0.73      0.71      0.71     20000
weighted avg       0.83      0.85      0.84     20000

Confusion Matrix:
[[793   0   0 ...   0   0   0]
 [  0 937   1 ...   0   0   0]
 [  0   0 976 ...   0   0   3]
 ...
 [  0   0   1 ...  64   0   0]
 [  0   0   0 ...   0  25   0]
 [  0   0  35 ...   1   0  24]]
Accuracy: 0.8459
Precision: 0.7332046578940103
Recall: 0.7061215043103424
```

CNN with Data Augmentation

In [18]:

```
import numpy as np
import pickle
import tensorflow as tf
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Conv2D, MaxPooling2D, Dense, Flatten, Dropout
from tensorflow.keras.utils import to_categorical
from tensorflow.keras.preprocessing.image import ImageDataGenerator
```

```
# Data Augmentation
datagen = ImageDataGenerator(
    rotation_range=10,  # randomly rotate images in the range (degrees, 0 to 180)
    zoom_range = 0.1, # Randomly zoom image
    width_shift_range=0.1,  # randomly shift images horizontally (fraction of total
    height_shift_range=0.1,  # randomly shift images vertically (fraction of total h
)
datagen.fit(train_data)

# Create the CNN model
model = Sequential([
    Conv2D(32, (3, 3), activation='relu', input_shape=(28, 28, 1)),
    MaxPooling2D((2, 2)),
    Dropout(0.25),  # Dropout regularization
    Conv2D(64, (3, 3), activation='relu'),
    MaxPooling2D((2, 2)),
    Dropout(0.25),  # Dropout regularization
    Flatten(),
    Dense(128, activation='relu'),
    Dropout(0.5),  # Dropout regularization
    Dense(len(train_labels[0]), activation='softmax')  # Number of output classes
])

# Compile the model
model.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['accuracy'


# Record the start time
start_time = time.time()

# Train the model
model.fit(datagen.flow(train_data, train_labels, batch_size=128), epochs=10, validat

# Record the end time
end_time = time.time()

# Calculate and print the training time
train_time = end_time - start_time
print('Training time:', train_time)


# Evaluate the model on test data
test_loss, test_acc = model.evaluate(test_data, test_labels)
print('Test accuracy:', test_acc)

# Generate predictions
preds = model.predict(test_data)
preds_classes = np.argmax(preds, axis=1)

# Convert back to integer format
test_labels_int = np.argmax(test_labels, axis=1)

# Generate and print the classification report
classification_report_results = classification_report(test_labels_int, preds_classes
print("Classification Report:")
print(classification_report_results)

# Generate and print the confusion matrix
confusion_matrix_results = confusion_matrix(test_labels_int, preds_classes)
print("Confusion Matrix:")
print(confusion_matrix_results)
```

```
accuracy = accuracy_score(test_labels_int, preds_classes)
precision = precision_score(test_labels_int, preds_classes, average='macro')
recall = recall_score(test_labels_int, preds_classes, average='macro')

print('Accuracy:', accuracy)
print('Precision:', precision)
print('Recall:', recall)
```

```
Epoch 1/10
782/782 [==============================] - 43s 54ms/step - loss: 1.8065 - accuracy:
0.5191 - val_loss: 0.6790 - val_accuracy: 0.7784
Epoch 2/10
782/782 [==============================] - 46s 58ms/step - loss: 1.0971 - accuracy:
0.6682 - val_loss: 0.5655 - val_accuracy: 0.8052
Epoch 3/10
782/782 [==============================] - 45s 58ms/step - loss: 0.9511 - accuracy:
0.7050 - val_loss: 0.5258 - val_accuracy: 0.8152
Epoch 4/10
782/782 [==============================] - 47s 60ms/step - loss: 0.8669 - accuracy:
0.7268 - val_loss: 0.4994 - val_accuracy: 0.8230
Epoch 5/10
782/782 [==============================] - 44s 57ms/step - loss: 0.8122 - accuracy:
0.7416 - val_loss: 0.4838 - val_accuracy: 0.8247
Epoch 6/10
782/782 [==============================] - 44s 56ms/step - loss: 0.7740 - accuracy:
0.7512 - val_loss: 0.4706 - val_accuracy: 0.8328
Epoch 7/10
782/782 [==============================] - 43s 55ms/step - loss: 0.7516 - accuracy:
0.7572 - val_loss: 0.4584 - val_accuracy: 0.8339
Epoch 8/10
782/782 [==============================] - 44s 56ms/step - loss: 0.7316 - accuracy:
0.7643 - val_loss: 0.4546 - val_accuracy: 0.8357
Epoch 9/10
782/782 [==============================] - 44s 56ms/step - loss: 0.7144 - accuracy:
0.7664 - val_loss: 0.4485 - val_accuracy: 0.8362
Epoch 10/10
782/782 [==============================] - 44s 57ms/step - loss: 0.6953 - accuracy:
0.7740 - val_loss: 0.4401 - val_accuracy: 0.8400
Training time: 444.8043382167816
625/625 [==============================] - 2s 4ms/step - loss: 0.4401 - accuracy: 0.
8400
Test accuracy: 0.8400499820709229
625/625 [==============================] - 2s 3ms/step
Classification Report:
              precision    recall  f1-score   support

           0       0.61      0.87      0.72       976
           1       0.59      0.98      0.74      1023
           2       0.90      0.98      0.94      1003
           3       0.99      0.99      0.99      1035
           4       0.94      0.98      0.96       903
           5       0.88      0.94      0.91       928
           6       0.95      0.97      0.96       959
           7       0.98      0.99      0.99      1098
           8       0.95      0.98      0.97       941
           9       0.90      0.97      0.93       929
          10       0.94      0.94      0.94       170
          11       0.90      0.91      0.90       118
          12       0.75      0.94      0.84       316
          13       0.85      0.81      0.83       128
          14       0.94      0.92      0.93       162
          15       0.83      0.94      0.88       261
          16       0.90      0.75      0.82        60
          17       0.83      0.96      0.89        74
          18       0.74      0.27      0.40       350
          19       0.82      0.84      0.83        95
          20       0.65      0.65      0.65        71
          21       0.89      0.94      0.91       141
          22       0.76      0.97      0.85       273
          23       0.91      0.98      0.94       249
          24       0.68      0.40      0.51       741
          25       0.80      0.95      0.87       277
```

```
           26       0.95      0.88      0.91        67
           27       0.92      0.95      0.94       160
           28       0.80      0.84      0.82       624
           29       0.90      0.93      0.91       249
           30       0.73      0.93      0.82       342
           31       0.56      0.90      0.69       126
           32       0.57      0.94      0.71       125
           33       0.55      0.87      0.67        85
           34       0.74      0.82      0.77       141
           35       0.57      0.50      0.53        62
           36       0.90      0.83      0.87       278
           37       0.89      0.85      0.87       170
           38       0.00      0.00      0.00        90
           39       0.94      0.96      0.95       297
           40       0.96      0.98      0.97       709
           41       0.00      0.00      0.00        57
           42       0.56      0.21      0.30        97
           43       0.96      0.92      0.94       269
           44       0.83      0.33      0.47        76
           45       0.79      0.52      0.63        60
           46       0.70      0.64      0.67        83
           47       0.00      0.00      0.00       421
           48       0.00      0.00      0.00        75
           49       0.95      0.94      0.94       332
           50       0.00      0.00      0.00        79
           51       0.75      0.05      0.09        65
           52       0.80      0.06      0.11        68
           53       0.96      0.94      0.95       411
           54       0.00      0.00      0.00        79
           55       0.93      0.94      0.94       502
           56       0.00      0.00      0.00        89
           57       0.75      0.03      0.07        88
           58       0.76      0.14      0.24        93
           59       0.82      0.37      0.51       107
           60       0.80      0.07      0.12        60
           61       0.75      0.22      0.34        83

    accuracy                           0.84     20000
   macro avg       0.73      0.67      0.66     20000
weighted avg       0.81      0.84      0.81     20000

Confusion Matrix:
[[ 851    0    0 ...    0    0    0]
 [   0 1006    4 ...    0    0    0]
 [   1    0  987 ...    0    0    0]
 ...
 [   0    0    1 ...   40    0    0]
 [   0    0    0 ...    0    4    0]
 [   0    0   39 ...    0    0   18]]
Accuracy: 0.84005
Precision: 0.72577282007204
Recall: 0.667475164631716
```

```
C:\Users\Haley\anaconda3\lib\site-packages\sklearn\metrics\_classification.py:1318:
UndefinedMetricWarning: Precision and F-score are ill-defined and being set to 0.0 i
n labels with no predicted samples. Use `zero_division` parameter to control this be
havior.
  _warn_prf(average, modifier, msg_start, len(result))
C:\Users\Haley\anaconda3\lib\site-packages\sklearn\metrics\_classification.py:1318:
UndefinedMetricWarning: Precision and F-score are ill-defined and being set to 0.0 i
n labels with no predicted samples. Use `zero_division` parameter to control this be
havior.
  _warn_prf(average, modifier, msg_start, len(result))
C:\Users\Haley\anaconda3\lib\site-packages\sklearn\metrics\_classification.py:1318:
UndefinedMetricWarning: Precision and F-score are ill-defined and being set to 0.0 i
n labels with no predicted samples. Use `zero_division` parameter to control this be
havior.
  _warn_prf(average, modifier, msg_start, len(result))
C:\Users\Haley\anaconda3\lib\site-packages\sklearn\metrics\_classification.py:1318:
UndefinedMetricWarning: Precision is ill-defined and being set to 0.0 in labels with
no predicted samples. Use `zero_division` parameter to control this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
```

Added more Dropout layers and BatchNormalization layers, EarlyStopping callback, and narrowed the search range of learning rate

rerun the first block and continue

In [67]:

```
import numpy as np
import time
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Conv2D, MaxPooling2D, Dense, Flatten, Dropout,
from tensorflow.keras.utils import to_categorical
from tensorflow.keras.callbacks import EarlyStopping, ModelCheckpoint
from tensorflow.keras.optimizers import Adam
from sklearn.metrics import classification_report, confusion_matrix, accuracy_score

# Reshape and normalize the data
train_data = train_data.reshape((-1, 28, 28, 1)) / 255.0
test_data = test_data.reshape((-1, 28, 28, 1)) / 255.0

# Convert the labels to one-hot encoding
train_labels = to_categorical(train_labels)
test_labels = to_categorical(test_labels)

# Create the CNN model
model = Sequential([
    Conv2D(32, (3, 3), activation='relu', input_shape=(28, 28, 1)),
    BatchNormalization(),
    MaxPooling2D((2, 2)),
    Dropout(0.2),

    Conv2D(64, (3, 3), activation='relu'),
    BatchNormalization(),
    MaxPooling2D((2, 2)),
    Dropout(0.3),

    Flatten(),

    Dense(128, activation='relu'),
```

```
    BatchNormalization(),
    Dropout(0.4),

    Dense(len(train_labels[0]), activation='softmax')  # Number of output classes
])

# Set learning rate range
optimizer = Adam(learning_rate=0.001)

# Compile the model
model.compile(optimizer=optimizer, loss='categorical_crossentropy', metrics=['accura

# Create callbacks
early_stopping = EarlyStopping(monitor='val_loss', patience=3)
model_checkpoint = ModelCheckpoint('model.h5', save_best_only=True)

# Record the start time
start_time = time.time()

# Train the model
model.fit(train_data, train_labels, epochs=10, batch_size=128, validation_split=0.1,
          callbacks=[early_stopping, model_checkpoint])

# Record the end time
end_time = time.time()

# Calculate and print the training time
train_time = end_time - start_time
print('Training time:', train_time)

# Evaluate the model on test data
test_loss, test_acc = model.evaluate(test_data, test_labels)
print('Test accuracy:', test_acc)

# Generate predictions
preds = model.predict(test_data)
preds_classes = np.argmax(preds, axis=1)

# Convert back to integer format
test_labels_int = np.argmax(test_labels, axis=1)

# Generate and print the classification report
classification_report_results = classification_report(test_labels_int, preds_classes
print("Classification Report:")
print(classification_report_results)

# Generate and print the confusion matrix
confusion_matrix_results = confusion_matrix(test_labels_int, preds_classes)
print("Confusion Matrix:")
print(confusion_matrix_results)

accuracy = accuracy_score(test_labels_int, preds_classes)
precision = precision_score(test_labels_int, preds_classes, average='macro')
recall = recall_score(test_labels_int, preds_classes, average='macro')

print('Accuracy:', accuracy)
print('Precision:', precision)
print('Recall:', recall)
```

```
Epoch 1/10
704/704 [==============================] - 58s 80ms/step - loss: 1.0874 - accuracy:
0.6948 - val_loss: 0.6971 - val_accuracy: 0.7704
Epoch 2/10
704/704 [==============================] - 60s 86ms/step - loss: 0.6273 - accuracy:
0.7952 - val_loss: 0.4978 - val_accuracy: 0.8263
Epoch 3/10
704/704 [==============================] - 62s 88ms/step - loss: 0.5587 - accuracy:
0.8116 - val_loss: 0.4623 - val_accuracy: 0.8423
Epoch 4/10
704/704 [==============================] - 60s 86ms/step - loss: 0.5230 - accuracy:
0.8214 - val_loss: 0.4397 - val_accuracy: 0.8476
Epoch 5/10
704/704 [==============================] - 53s 76ms/step - loss: 0.5014 - accuracy:
0.8271 - val_loss: 0.4288 - val_accuracy: 0.8486
Epoch 6/10
704/704 [==============================] - 57s 81ms/step - loss: 0.4840 - accuracy:
0.8303 - val_loss: 0.4183 - val_accuracy: 0.8508
Epoch 7/10
704/704 [==============================] - 58s 82ms/step - loss: 0.4727 - accuracy:
0.8348 - val_loss: 0.4154 - val_accuracy: 0.8510
Epoch 8/10
704/704 [==============================] - 58s 83ms/step - loss: 0.4623 - accuracy:
0.8358 - val_loss: 0.4123 - val_accuracy: 0.8531
Epoch 9/10
704/704 [==============================] - 60s 85ms/step - loss: 0.4526 - accuracy:
0.8392 - val_loss: 0.4106 - val_accuracy: 0.8556
Epoch 10/10
704/704 [==============================] - 57s 81ms/step - loss: 0.4447 - accuracy:
0.8411 - val_loss: 0.4087 - val_accuracy: 0.8551
Training time: 583.6247837543488
625/625 [==============================] - 4s 6ms/step - loss: 0.4151 - accuracy: 0.
8532
Test accuracy: 0.8532000184059143
625/625 [==============================] - 3s 5ms/step
Classification Report:
              precision    recall  f1-score   support

           0       0.64      0.85      0.73       976
           1       0.64      0.96      0.77      1023
           2       0.95      0.97      0.96      1003
           3       0.99      0.99      0.99      1035
           4       0.96      0.97      0.96       903
           5       0.93      0.91      0.92       928
           6       0.97      0.97      0.97       959
           7       0.97      0.99      0.98      1098
           8       0.97      0.98      0.97       941
           9       0.90      0.98      0.94       929
          10       0.93      0.94      0.94       170
          11       0.92      0.92      0.92       118
          12       0.74      0.96      0.84       316
          13       0.87      0.86      0.86       128
          14       0.93      0.96      0.95       162
          15       0.83      0.93      0.87       261
          16       0.83      0.90      0.86        60
          17       0.87      1.00      0.93        74
          18       0.64      0.43      0.51       350
          19       0.81      0.78      0.80        95
          20       0.62      0.85      0.71        71
          21       0.80      0.94      0.87       141
          22       0.77      0.96      0.85       273
          23       0.91      0.98      0.95       249
          24       0.67      0.47      0.55       741
          25       0.82      0.95      0.88       277
```

```
              26       0.97      0.87      0.91        67
              27       0.92      0.94      0.93       160
              28       0.79      0.92      0.85       624
              29       0.89      0.94      0.91       249
              30       0.75      0.94      0.84       342
              31       0.64      0.83      0.72       126
              32       0.78      0.82      0.80       125
              33       0.85      0.46      0.60        85
              34       0.73      0.84      0.79       141
              35       0.63      0.42      0.50        62
              36       0.87      0.88      0.88       278
              37       0.90      0.86      0.88       170
              38       0.00      0.00      0.00        90
              39       0.96      0.97      0.96       297
              40       0.97      0.98      0.98       709
              41       0.50      0.02      0.03        57
              42       0.84      0.27      0.41        97
              43       0.95      0.91      0.93       269
              44       0.85      0.38      0.53        76
              45       0.67      0.58      0.63        60
              46       0.81      0.55      0.66        83
              47       0.51      0.08      0.14       421
              48       0.40      0.03      0.05        75
              49       0.94      0.92      0.93       332
              50       0.00      0.00      0.00        79
              51       0.77      0.15      0.26        65
              52       0.75      0.22      0.34        68
              53       0.93      0.95      0.94       411
              54       0.00      0.00      0.00        79
              55       0.93      0.93      0.93       502
              56       0.60      0.07      0.12        89
              57       0.73      0.27      0.40        88
              58       0.80      0.71      0.75        93
              59       0.65      0.83      0.73       107
              60       0.38      0.10      0.16        60
              61       0.57      0.52      0.54        83

    accuracy                           0.85     20000
   macro avg       0.76      0.70      0.70     20000
weighted avg       0.84      0.85      0.83     20000

Confusion Matrix:
[[829   0   0 ...   0   0   0]
 [  0 979   0 ...   0   0   1]
 [  1   0 971 ...   1   0   8]
 ...
 [  0   1   0 ...  89   0   1]
 [  0   0   0 ...   0   6   0]
 [  0   0  17 ...   1   0  43]]
Accuracy: 0.8532
Precision: 0.7603490179470113
Recall: 0.7026956944244633
```

```
C:\Users\Haley\anaconda3\lib\site-packages\sklearn\metrics\_classification.py:1318:
UndefinedMetricWarning: Precision and F-score are ill-defined and being set to 0.0 i
n labels with no predicted samples. Use `zero_division` parameter to control this be
havior.
  _warn_prf(average, modifier, msg_start, len(result))
C:\Users\Haley\anaconda3\lib\site-packages\sklearn\metrics\_classification.py:1318:
UndefinedMetricWarning: Precision and F-score are ill-defined and being set to 0.0 i
n labels with no predicted samples. Use `zero_division` parameter to control this be
havior.
  _warn_prf(average, modifier, msg_start, len(result))
C:\Users\Haley\anaconda3\lib\site-packages\sklearn\metrics\_classification.py:1318:
UndefinedMetricWarning: Precision and F-score are ill-defined and being set to 0.0 i
n labels with no predicted samples. Use `zero_division` parameter to control this be
havior.
  _warn_prf(average, modifier, msg_start, len(result))
C:\Users\Haley\anaconda3\lib\site-packages\sklearn\metrics\_classification.py:1318:
UndefinedMetricWarning: Precision is ill-defined and being set to 0.0 in labels with
no predicted samples. Use `zero_division` parameter to control this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
```

CNN with tuning and solve zero_division

In [77]:

```
from kerastuner.tuners import RandomSearch
from kerastuner.engine.hyperparameters import HyperParameters

# Define the hyperparameter search space
hyperparameters = HyperParameters()
hyperparameters.Int('conv_1_filters', 32, 128, step=32)
hyperparameters.Float('dropout_1', 0.2, 0.5, step=0.1)
hyperparameters.Int('conv_2_filters', 64, 256, step=32)
hyperparameters.Float('dropout_2', 0.2, 0.5, step=0.1)
hyperparameters.Int('dense_units', 128, 512, step=64)
hyperparameters.Float('dropout_3', 0.2, 0.5, step=0.1)
hyperparameters.Choice('learning_rate', values=[1e-2, 1e-3, 1e-4])

# Define the function to build the model
def build_model(hp):
    model = Sequential()
    model.add(Conv2D(hp.Int('conv_1_filters', 32, 128, step=32), (3, 3), activation=
    model.add(BatchNormalization())
    model.add(MaxPooling2D((2, 2)))
    model.add(Dropout(hp.Float('dropout_1', 0.2, 0.5, step=0.1)))

    model.add(Conv2D(hp.Int('conv_2_filters', 64, 256, step=32), (3, 3), activation=
    model.add(BatchNormalization())
    model.add(MaxPooling2D((2, 2)))
    model.add(Dropout(hp.Float('dropout_2', 0.2, 0.5, step=0.1)))

    model.add(Flatten())

    model.add(Dense(hp.Int('dense_units', 128, 512, step=64), activation='relu'))
    model.add(BatchNormalization())
    model.add(Dropout(hp.Float('dropout_3', 0.2, 0.5, step=0.1)))

    model.add(Dense(len(train_labels[0]), activation='softmax'))

    optimizer = Adam(learning_rate=hp.Choice('learning_rate', values=[1e-2, 1e-3, 1e
    model.compile(optimizer=optimizer, loss='categorical_crossentropy', metrics=['ac

    return model
```

```
# Create the tuner with reduced max_trials
tuner = RandomSearch(
    build_model,
    objective='val_accuracy',
    max_trials=3,
    executions_per_trial=1,
    directory='tuner_dir',
    project_name='my_model'
)

# Perform hyperparameter tuning with reduced epochs
tuner.search(train_data, train_labels, epochs=3, validation_split=0.1)

# Get the best hyperparameters
best_hps = tuner.get_best_hyperparameters()[0]

# Print the best hyperparameters
print("Best Hyperparameters:")
print(best_hps)
```

```
Trial 2 Complete [00h 08m 54s]
val_accuracy: 0.8285999894142151

Best val_accuracy So Far: 0.8468999862670898
Total elapsed time: 00h 08m 54s
INFO:tensorflow:Oracle triggered exit
Best Hyperparameters:
<keras_tuner.engine.hyperparameters.hyperparameters.HyperParameters object at 0x0000
0206C8542D00>
```

In [94]:

```
import numpy as np
import time
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Conv2D, MaxPooling2D, Dense, Flatten, Dropout,
from tensorflow.keras.utils import to_categorical
from tensorflow.keras.callbacks import EarlyStopping, ModelCheckpoint
from tensorflow.keras.optimizers import Adam
from sklearn.metrics import classification_report, confusion_matrix, accuracy_score

# Import Keras Tuner related functions
from kerastuner.tuners import RandomSearch
from kerastuner.engine.hyperparameters import HyperParameters

# Define the hyperparameter search space
hyperparameters = HyperParameters()
hyperparameters.Int('conv_1_filters', 32, 128, step=32)
hyperparameters.Float('dropout_1', 0.2, 0.5, step=0.1)
hyperparameters.Int('conv_2_filters', 64, 256, step=32)
hyperparameters.Float('dropout_2', 0.2, 0.5, step=0.1)
hyperparameters.Int('dense_units', 128, 512, step=64)
hyperparameters.Float('dropout_3', 0.2, 0.5, step=0.1)
hyperparameters.Choice('learning_rate', values=[1e-2, 1e-3, 1e-4])

# Assuming train_data, test_data, train_labels, and test_labels are defined

# Reshape and normalize the data
train_data = train_data.reshape((-1, 28, 28, 1)) / 255.0
test_data = test_data.reshape((-1, 28, 28, 1)) / 255.0

# Convert the labels to one-hot encoding
train_labels = to_categorical(train_labels)
test_labels = to_categorical(test_labels)

# Define the function to build the model
```

```
def build_model(hp):
    model = Sequential()
    model.add(Conv2D(hp.Int('conv_1_filters', 32, 128, step=32), (3, 3), activation=
    model.add(BatchNormalization())
    model.add(MaxPooling2D((2, 2)))
    model.add(Dropout(hp.Float('dropout_1', 0.2, 0.5, step=0.1)))

    model.add(Conv2D(hp.Int('conv_2_filters', 64, 256, step=32), (3, 3), activation=
    model.add(BatchNormalization())
    model.add(MaxPooling2D((2, 2)))
    model.add(Dropout(hp.Float('dropout_2', 0.2, 0.5, step=0.1)))

    model.add(Flatten())

    model.add(Dense(hp.Int('dense_units', 128, 512, step=64), activation='relu'))
    model.add(BatchNormalization())
    model.add(Dropout(hp.Float('dropout_3', 0.2, 0.5, step=0.1)))

    model.add(Dense(len(train_labels[0]), activation='softmax'))

    optimizer = Adam(learning_rate=hp.Choice('learning_rate', values=[1e-2, 1e-3, 1e
    model.compile(optimizer=optimizer, loss='categorical_crossentropy',
    metrics=['accuracy'])

    return model

# Create the tuner
tuner = RandomSearch(
    build_model,
    objective='val_accuracy',
    max_trials=3,
    executions_per_trial=1,
    directory='tuner_dir',
    project_name='my_model'
)

# Create callbacks
early_stopping = EarlyStopping(monitor='val_loss', patience=3)
model_checkpoint = ModelCheckpoint('model.h5', save_best_only=True)

# Perform hyperparameter tuning
tuner.search(train_data, train_labels, epochs=10, validation_split=0.1,
             callbacks=[early_stopping, model_checkpoint])

# Get the best hyperparameters
best_hps = tuner.get_best_hyperparameters()[0]

# Print the best hyperparameters
print("Best Hyperparameters:")
print(best_hps)

# Build the model with the best hyperparameters
model = build_model(best_hps)

# Train the model
model.fit(train_data, train_labels, epochs=10, batch_size=128, validation_split=0.1,
          callbacks=[early_stopping, model_checkpoint])

# Evaluate the model on test data
test_loss, test_acc = model.evaluate(test_data, test_labels)
print('Test accuracy:', test_acc)

# Generate predictions
preds = model.predict(test_data)
```

```
preds_classes = np.argmax(preds, axis=1)

# Convert back to integer format
test_labels_int = np.argmax(test_labels, axis=1)

# Generate and print the classification report
classification_report_results = classification_report(test_labels_int, preds_classes
print("Classification Report:")
print(classification_report_results)

# Generate and print the confusion matrix
confusion_matrix_results = confusion_matrix(test_labels_int, preds_classes)
print("Confusion Matrix:")
print(confusion_matrix_results)

# Calculate and print the accuracy, precision, and recall scores
accuracy = accuracy_score(test_labels_int, preds_classes)
precision = precision_score(test_labels_int, preds_classes, average='macro', zero_di
recall = recall_score(test_labels_int, preds_classes, average='macro', zero_division

print('Accuracy:', accuracy)
print('Precision:', precision)
print('Recall:', recall)
```

```
INFO:tensorflow:Reloading Tuner from tuner_dir\my_model\tuner0.json
INFO:tensorflow:Oracle triggered exit
Best Hyperparameters:
<keras_tuner.engine.hyperparameters.hyperparameters.HyperParameters object at 0x0000
0206D8FB7DF0>
Epoch 1/10
704/704 [==============================] - 136s 192ms/step - loss: 1.5989 - accurac
y: 0.6012 - val_loss: 1.2708 - val_accuracy: 0.6839
Epoch 2/10
704/704 [==============================] - 137s 195ms/step - loss: 0.9053 - accurac
y: 0.7439 - val_loss: 0.7315 - val_accuracy: 0.7908
Epoch 3/10
704/704 [==============================] - 149s 212ms/step - loss: 0.7368 - accurac
y: 0.7795 - val_loss: 0.6266 - val_accuracy: 0.8081
Epoch 4/10
704/704 [==============================] - 175s 248ms/step - loss: 0.6520 - accurac
y: 0.7966 - val_loss: 0.5678 - val_accuracy: 0.8220
Epoch 5/10
704/704 [==============================] - 190s 269ms/step - loss: 0.5976 - accurac
y: 0.8089 - val_loss: 0.5351 - val_accuracy: 0.8247
Epoch 6/10
704/704 [==============================] - 189s 268ms/step - loss: 0.5585 - accurac
y: 0.8166 - val_loss: 0.5039 - val_accuracy: 0.8338
Epoch 7/10
704/704 [==============================] - 191s 272ms/step - loss: 0.5290 - accurac
y: 0.8247 - val_loss: 0.4858 - val_accuracy: 0.8355
Epoch 8/10
704/704 [==============================] - 202s 287ms/step - loss: 0.5039 - accurac
y: 0.8304 - val_loss: 0.4605 - val_accuracy: 0.8426
Epoch 9/10
704/704 [==============================] - 189s 269ms/step - loss: 0.4848 - accurac
y: 0.8359 - val_loss: 0.4487 - val_accuracy: 0.8483
Epoch 10/10
704/704 [==============================] - 160s 228ms/step - loss: 0.4697 - accurac
y: 0.8377 - val_loss: 0.4366 - val_accuracy: 0.8515
625/625 [==============================] - 7s 12ms/step - loss: 0.4460 - accuracy:
0.8469
Test accuracy: 0.8469499945640564
625/625 [==============================] - 7s 11ms/step
Classification Report:
              precision    recall  f1-score   support

           0       0.68      0.73      0.70       976
           1       0.66      0.93      0.77      1023
           2       0.93      0.97      0.95      1003
           3       0.99      0.99      0.99      1035
           4       0.96      0.96      0.96       903
           5       0.93      0.91      0.92       928
           6       0.94      0.98      0.96       959
           7       0.98      0.98      0.98      1098
           8       0.93      0.99      0.96       941
           9       0.91      0.97      0.94       929
          10       0.90      0.95      0.92       170
          11       0.88      0.85      0.87       118
          12       0.75      0.91      0.82       316
          13       0.85      0.84      0.85       128
          14       0.94      0.89      0.91       162
          15       0.83      0.91      0.86       261
          16       0.96      0.80      0.87        60
          17       0.91      0.92      0.91        74
          18       0.57      0.49      0.53       350
          19       0.93      0.67      0.78        95
          20       0.63      0.80      0.70        71
          21       0.84      0.94      0.89       141
```

```
           22       0.77      0.95      0.85       273
           23       0.89      0.96      0.92       249
           24       0.60      0.63      0.62       741
           25       0.85      0.86      0.85       277
           26       0.90      0.85      0.88        67
           27       0.90      0.92      0.91       160
           28       0.79      0.92      0.85       624
           29       0.91      0.89      0.90       249
           30       0.77      0.90      0.83       342
           31       0.61      0.75      0.67       126
           32       0.80      0.78      0.79       125
           33       0.67      0.71      0.69        85
           34       0.73      0.79      0.76       141
           35       0.65      0.21      0.32        62
           36       0.86      0.86      0.86       278
           37       0.91      0.81      0.86       170
           38       0.00      0.00      0.00        90
           39       0.94      0.97      0.95       297
           40       0.94      0.99      0.96       709
           41       0.43      0.05      0.09        57
           42       0.69      0.26      0.38        97
           43       0.94      0.90      0.92       269
           44       0.77      0.36      0.49        76
           45       0.73      0.68      0.71        60
           46       0.75      0.58      0.65        83
           47       0.56      0.15      0.23       421
           48       0.43      0.04      0.07        75
           49       0.92      0.93      0.92       332
           50       1.00      0.00      0.00        79
           51       0.47      0.46      0.47        65
           52       0.74      0.21      0.32        68
           53       0.90      0.95      0.93       411
           54       1.00      0.00      0.00        79
           55       0.93      0.93      0.93       502
           56       0.50      0.20      0.29        89
           57       0.64      0.31      0.42        88
           58       0.78      0.74      0.76        93
           59       0.77      0.64      0.70       107
           60       0.43      0.20      0.27        60
           61       0.65      0.48      0.55        83

    accuracy                           0.85     20000
   macro avg       0.78      0.70      0.70     20000
weighted avg       0.84      0.85      0.83     20000

Confusion Matrix:
[[712   0   1 ...   0   0   0]
 [  0 949   2 ...   0   0   0]
 [  0   0 973 ...   1   0   2]
 ...
 [  0   1   1 ...  69   0   0]
 [  0   0   0 ...   0  12   0]
 [  0   0  25 ...   1   0  40]]
Accuracy: 0.84695
Precision: 0.7808151860624953
Recall: 0.6966060166276296
```

SVM Experiments with varying C and Kernels

In [6]:

```
from sklearn.svm import SVC
from sklearn.metrics import accuracy_score, precision_score, recall_score, confusic
from sklearn.model_selection import cross_val_score, train_test_split
import numpy as np
import time

# Define parameter grid
C_values = np.logspace(-3, 2, num=5)

# Define kernel types
kernels = ['rbf', 'poly']

# Split your data so that you only use 50% of it for training
sample_X_train, _, sample_y_train, _ = train_test_split(X_train, y_train, test_size=

best_score = 0
best_params = {}

# Perform grid search on the training data for each kernel type and 'C' value
for kernel in kernels:
    for C in C_values:
        start = time.time()

        # Create an instance of the SVM model with current kernel
        svm_model_param_grid = SVC(kernel=kernel, C=C)

        # Perform cross-validation on the training data
        scores = cross_val_score(svm_model_param_grid, sample_X_train, sample_y_trai

        # Compute the mean accuracy and check if it's the best score so far
        mean_score = np.mean(scores)
        if mean_score > best_score:
            best_score = mean_score
            best_params = {'kernel': kernel, 'C': C}

        # Fit the model with the 'C' parameter
        svm_model_param_grid.fit(sample_X_train, sample_y_train)

        # Make predictions on the test set using the model
        predictions_SVMBestparam = svm_model_param_grid.predict(X_test)

        # Calculate evaluation metrics
        accuracy = accuracy_score(y_test, predictions_SVMBestparam)
        precision = precision_score(y_test, predictions_SVMBestparam, average='macro
        recall = recall_score(y_test, predictions_SVMBestparam, average='macro')
        confusion_mat = confusion_matrix(y_test, predictions_SVMBestparam)

        end = time.time()

        print("Parameters: Kernel=", kernel, "C=", C)
        print("Mean cross-validation accuracy:", mean_score)
        print("Accuracy:", accuracy)
        print("Precision:", precision)
        print("Recall:", recall)
        print("Elapsed Time:", end - start)
        print()

print("Best Parameters:", best_params, "with highest mean cross-validation accuracy:
```

```
C:\Users\Haley\anaconda3\lib\site-packages\sklearn\metrics\_classification.py:1318:
UndefinedMetricWarning: Precision is ill-defined and being set to 0.0 in labels with
no predicted samples. Use `zero_division` parameter to control this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
```

```
Parameters: Kernel= rbf C= 0.001
Mean cross-validation accuracy: 0.05535000006918577
Accuracy: 0.05465
Precision: 0.007791782423387355
Recall: 0.016175291432800838
Elapsed Time: 5008.530836582184

C:\Users\Haley\anaconda3\lib\site-packages\sklearn\metrics\_classification.py:1318: UndefinedMetricWarning: Precision is ill-defined and being set to 0.0 in labels with no predicted samples. Use `zero_division` parameter to control this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
Parameters: Kernel= rbf C= 0.01778279410038923
Mean cross-validation accuracy: 0.4880500087348441
Accuracy: 0.533
Precision: 0.22705525025965226
Recall: 0.20633334240237858
Elapsed Time: 22268.483466863632

C:\Users\Haley\anaconda3\lib\site-packages\sklearn\metrics\_classification.py:1318: UndefinedMetricWarning: Precision is ill-defined and being set to 0.0 in labels with no predicted samples. Use `zero_division` parameter to control this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
Parameters: Kernel= rbf C= 0.31622776601683794
Mean cross-validation accuracy: 0.7150249908940088
Accuracy: 0.73145
Precision: 0.5956326855218941
Recall: 0.47159356019462234
Elapsed Time: 1928.0591492652893

C:\Users\Haley\anaconda3\lib\site-packages\sklearn\metrics\_classification.py:1318: UndefinedMetricWarning: Precision is ill-defined and being set to 0.0 in labels with no predicted samples. Use `zero_division` parameter to control this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
Parameters: Kernel= rbf C= 5.623413251903491
Mean cross-validation accuracy: 0.7782250147224131
Accuracy: 0.78765
Precision: 0.6843231429789959
Recall: 0.5996393949271969
Elapsed Time: 1553.9620718955994

Parameters: Kernel= rbf C= 100.0
Mean cross-validation accuracy: 0.7664500203325542
Accuracy: 0.77735
Precision: 0.6745260369351598
Recall: 0.601002881319926
Elapsed Time: 1818.9180636405945

C:\Users\Haley\anaconda3\lib\site-packages\sklearn\metrics\_classification.py:1318: UndefinedMetricWarning: Precision is ill-defined and being set to 0.0 in labels with no predicted samples. Use `zero_division` parameter to control this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
Parameters: Kernel= poly C= 0.001
Mean cross-validation accuracy: 0.05562499944454513
Accuracy: 0.0551
Precision: 0.047407157440332165
Recall: 0.017502732863971263
Elapsed Time: 8902.655396938324

C:\Users\Haley\anaconda3\lib\site-packages\sklearn\metrics\_classification.py:1318: UndefinedMetricWarning: Precision is ill-defined and being set to 0.0 in labels with no predicted samples. Use `zero_division` parameter to control this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
```

```
Parameters: Kernel= poly C= 0.01778279410038923
Mean cross-validation accuracy: 0.1416752476709023
Accuracy: 0.26
Precision: 0.36394742064784563
Recall: 0.10906894234203014
Elapsed Time: 3895.9232897758484

C:\Users\Haley\anaconda3\lib\site-packages\sklearn\metrics\_classification.py:1318: UndefinedMetricWarning: Precision is ill-defined and being set to 0.0 in labels with no predicted samples. Use `zero_division` parameter to control this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
Parameters: Kernel= poly C= 0.31622776601683794
Mean cross-validation accuracy: 0.6480001283067924
Accuracy: 0.6809
Precision: 0.5754524787241456
Recall: 0.3982451637096467
Elapsed Time: 2705.840261220932

Parameters: Kernel= poly C= 5.623413251903491
Mean cross-validation accuracy: 0.7730000715894604
Accuracy: 0.7869
Precision: 0.6780583818063879
Recall: 0.5922134056978122
Elapsed Time: 1857.6277301311493

Parameters: Kernel= poly C= 100.0
Mean cross-validation accuracy: 0.7689250353352729
Accuracy: 0.78105
Precision: 0.6726165518699603
Recall: 0.6062399787837118
Elapsed Time: 1790.865995168686

Best Parameters: {'kernel': 'rbf', 'C': 5.623413251903491} with highest mean cross-validation accuracy: 0.7782250147224131
```

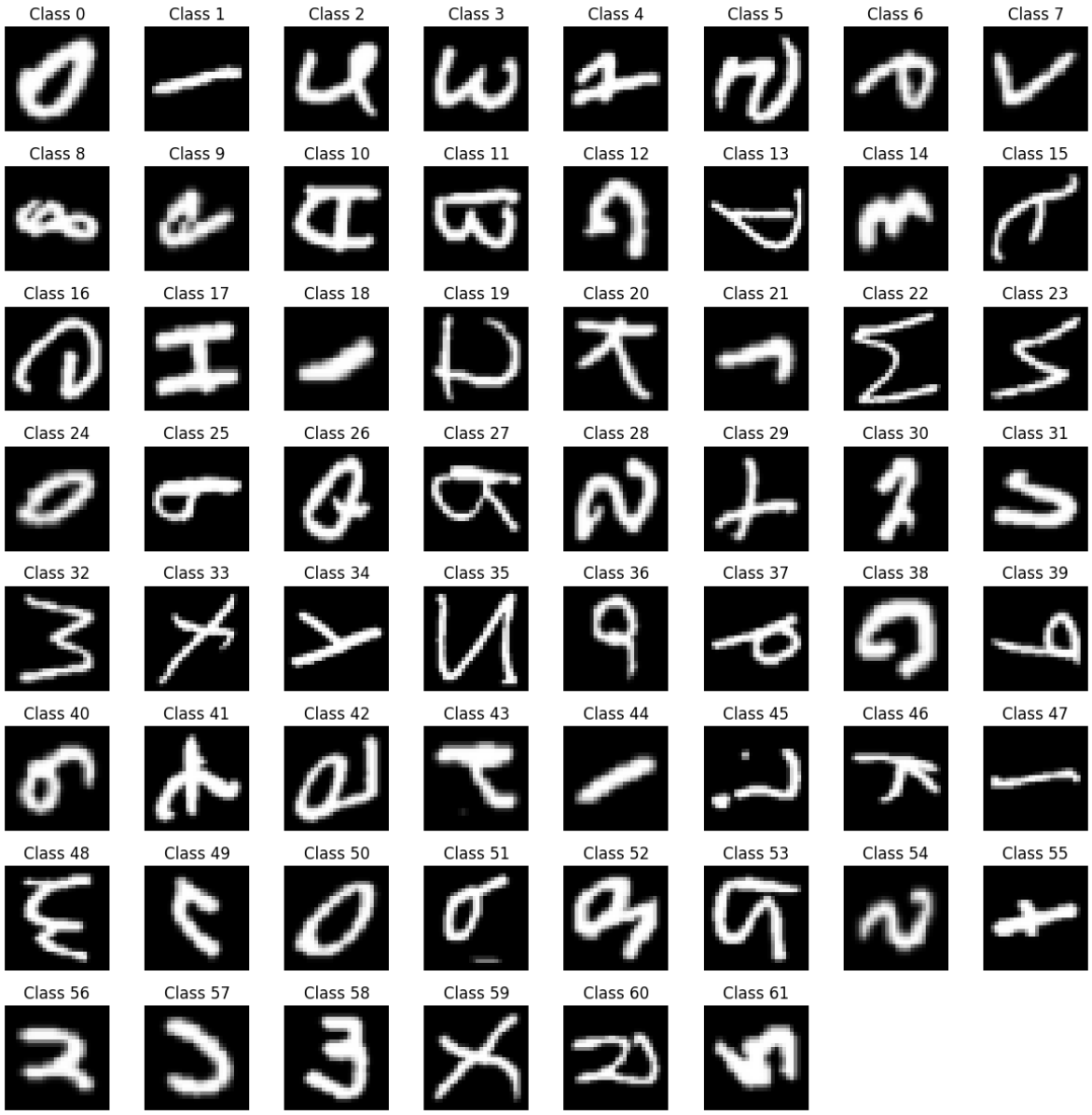
In [ ]:

Linear model

In [1]:
```
#Import all neccessary libraries
import numpy as np
import pickle
import matplotlib.pyplot as plt
import time
from sklearn.decomposition import PCA
from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import train_test_split
from sklearn.svm import SVC
from sklearn.metrics import accuracy_score, precision_score, recall_score, confusic
from sklearn.model_selection import GridSearchCV



# Load the emnist_train.pkl file
with open('/kaggle/input/emnist-trainpkl/emnist_train.pkl', 'rb') as f:
    train_dict = pickle.load(f, encoding='bytes')

# Extract the data and labels arrays from the dictionary
train_data = train_dict['data']
train_labels = train_dict['labels']


# Load the emnist_test.pkl file
with open('/kaggle/input/emnist-testpkl/emnist_test.pkl', 'rb') as f:
    test_dict = pickle.load(f, encoding='bytes')

# Extract the data and labels arrays from the dictionary
test_data = test_dict['data']
test_labels = test_dict['labels']

# Convert the data and labels arrays to NumPy arrays
train_data = np.array(train_data, dtype=np.float32)
train_labels = np.array(train_labels, dtype=np.int32)
test_data = np.array(test_data, dtype=np.float32)
test_labels = np.array(test_labels, dtype=np.int32)

# Define class names
# Create class names for all 62 classes
class_names = ['Class {}'.format(i) for i in range(62)]


#Plot a grid of EMNIST examples of a specified size."""

def plot_examples(data, pred, target, n_rows=3, n_cols=7):
    """Plot a grid of EMNIST examples of a specified size."""
    # Size figure depending on the size of the grid
    plt.figure(figsize=(n_cols * 1.2, n_rows * 2.0))

    for row in range(n_rows):
        for col in range(n_cols):
            # Get the next index of the image
            index = n_cols * row + col

            # Reshape the flattened image to its original shape
            image = data[index].reshape((28, 28))

            # Plot the image at the appropriate place in the grid
```

```
            plt.subplot(n_rows, n_cols, index + 1)
            plt.imshow(image, cmap="binary")
            plt.axis('off')
            plt.title(class_names[target[index]] + '\n' + '>>' + class_names[pred[ir

    plt.tight_layout()
    plt.show()

# Plot examples from different classes
fig, axes = plt.subplots(8, 8, figsize=(12, 12))
for i, ax in enumerate(axes.flat):
    if i < len(class_names):
        class_index = i  # Compute the class index
        # Get samples of the current class
        class_samples = train_data[train_labels == class_index]
        # Randomly select a sample from the class
        sample_index = np.random.randint(class_samples.shape[0])
        ax.imshow(class_samples[sample_index].reshape((28, 28)), cmap='gray')
        ax.axis('off')
        # Set the title as the class name
        ax.set_title(class_names[class_index])
    else:
        ax.axis('off')

plt.tight_layout()
plt.show()
```

```
/opt/conda/lib/python3.10/site-packages/scipy/__init__.py:146: UserWarning: A NumPy
version >=1.16.5 and <1.23.0 is required for this version of SciPy (detected version
1.23.5
  warnings.warn(f"A NumPy version >={np_minversion} and <{np_maxversion}"
```



In [2]:
```
#Data preprocessing
#Perform simple Normalise and reshape data
train_images = train_data / 255.0
test_images = test_data / 255.0

# Reshape the data from (28, 28) to (784,)
train_images = train_data.reshape(-1, 784)
test_images = test_data.reshape(-1, 784)

# Normalize the data by standardization
scaler = StandardScaler()
train_images = scaler.fit_transform(train_images)
test_images = scaler.transform(test_images)
```

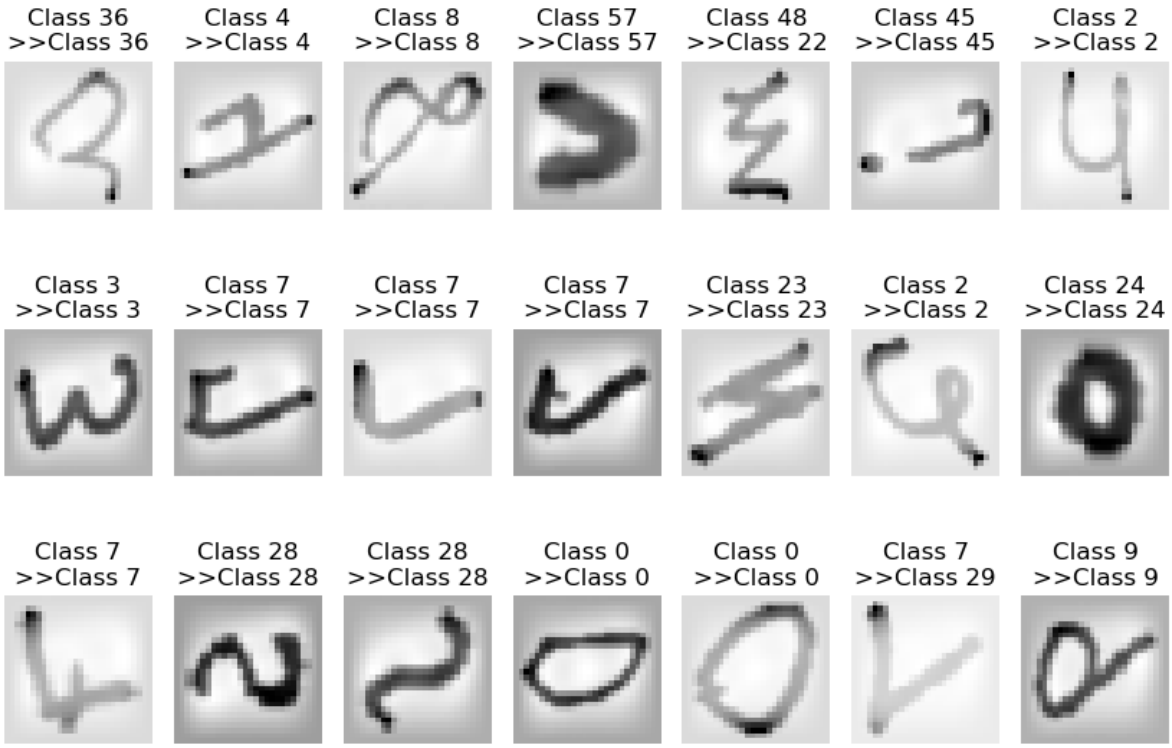
In [3]:
```
# Split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(train_images, train_labels, test
```

SVM with Linear kernel

In [ ]:
```
# Create an instance of the linear SVM model
svm_model = SVC(kernel='linear')

start = time.time()
```

```
# Train the model
svm_model.fit(X_train, y_train)

end = time.time()
print(end - start)

# Make predictions on the test set
predictions = svm_model.predict(X_test)

start = time.time()

# Calculate evaluation metrics
accuracy = accuracy_score(y_test, predictions)
precision = precision_score(y_test, predictions, average='macro')
recall = recall_score(y_test, predictions, average='macro')
confusion_mat = confusion_matrix(y_test, predictions)

end = time.time()
print(end - start)

# Print the evaluation metrics and confusion matrix
print("Accuracy_Linear:", accuracy)
print("Precision_Linear:", precision)
print("Recall_Linear:", recall)
print("Confusion Matrix_Linear:")
print(confusion_mat)

# Plot examples of linear SVM
plot_examples(X_test, predictions, y_test, n_rows=3, n_cols=7)
```

```
978.5045220851898

0.02817225456237793

Accuracy_Linear: 0.7264

Precision_Linear: 0.581932100433202

Recall_Linear: 0.5626831399061308

Confusion Matrix_Linear:

[[ 731    0    0 ...    0    0    0]

 [   0 1005    2 ...    0    0    0]

 [   2    3  837 ...    0    0   20]

 ...

 [   0    3    0 ...   39    2    0]

 [   0    3    1 ...    1   21    0]

 [   0    0   35 ...    0    1   24]]
```



PCA compression on Linear model

In [ ]:

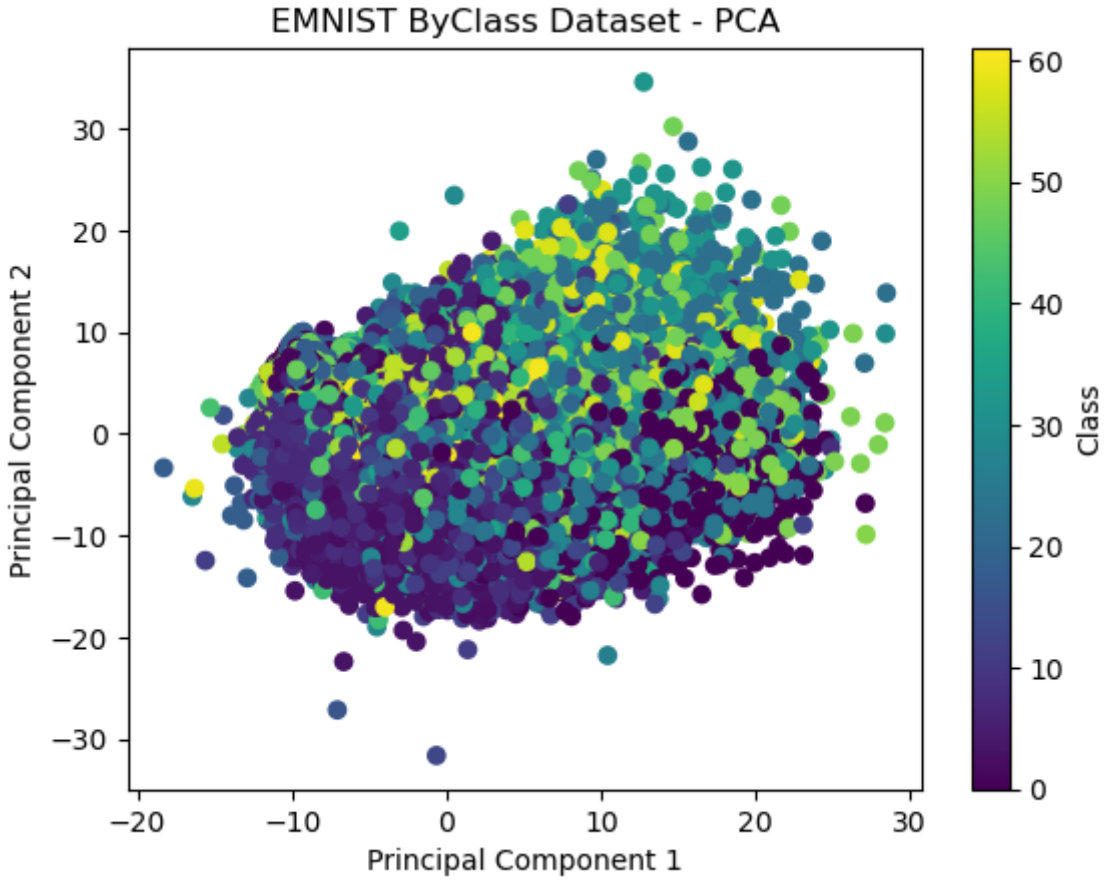
```
# apply PCA without reducing dimensionality, then compute the min number of dimensio
pca = PCA()
pca.fit(X_train)
cumsum = np.cumsum(pca.explained_variance_ratio_)
d = np.argmax(cumsum >= 0.95) + 1


# Apply PCA for dimensionality reduction

pca = PCA(n_components=d)
X_train_pca = pca.fit_transform(X_train)
# Transform the test set
X_test_pca = pca.transform(X_test)

# Fit the SVM model on the transformed training set
svm_model = SVC(kernel= 'linear')

start = time.time()

svm_model.fit(X_train_pca, y_train)

end = time.time()
print(end - start)

# Make predictions on the transformed test set
predictions_pca = svm_model.predict(X_test_pca)

start = time.time()
# Calculate accuracy
accuracy = accuracy_score(y_test, predictions_pca)

end = time.time()
print(end - start)

print("Reduced data Accuracy:", accuracy)
```

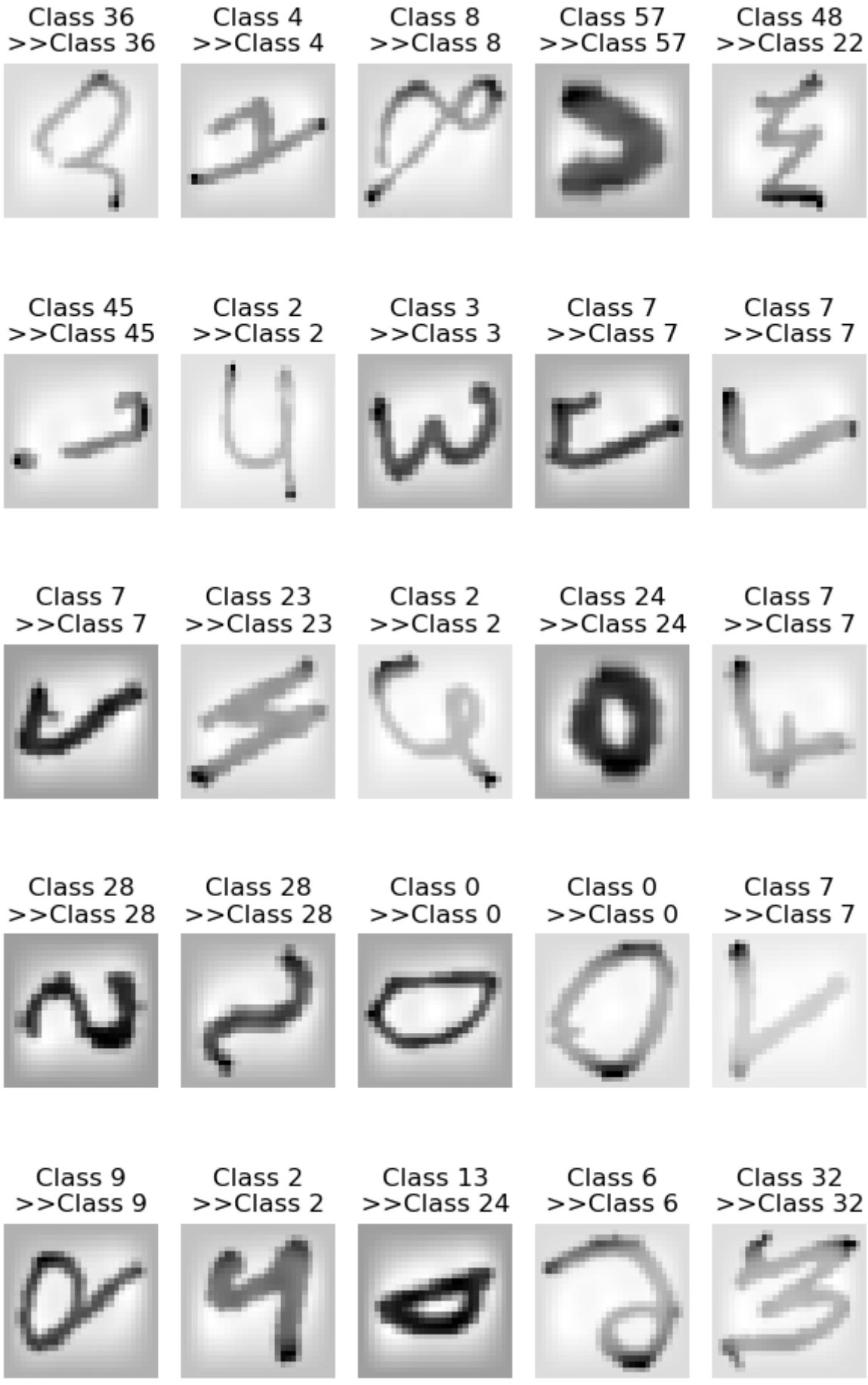
```
# Plotting the reduced data to find if they form uniform clusters

plt.scatter(X_train_pca[:, 0], X_train_pca[:, 1], c=y_train)
plt.xlabel('Principal Component 1')
plt.ylabel('Principal Component 2')
plt.title('EMNIST ByClass Dataset - PCA')
plt.colorbar(label='Class')
plt.savefig('pca.pdf')
plt.show()

# Plot examples of PCA compressed SVM results
plot_examples(X_test, predictions_pca, y_test, n_rows=5, n_cols=5)
```

843.0075702667236

0.0025217533111572266

Reduced data Accuracy: 0.7494





SVM with default paramters

In [ ]:

```
# Create an instance of the SVM model with default hyperparameters
svm_model = SVC(kernel='poly', C=1.0, gamma='scale')

start = time.time()
# Train the model
svm_model.fit(X_train, y_train)
end = time.time()
print(end - start)

# Make predictions on the test set
predictions_SVMD = svm_model.predict(X_test)

start = time.time()

# Calculate evaluation metrics
accuracy = accuracy_score(y_test, predictions_SVMD)
precision = precision_score(y_test, predictions_SVMD, average='macro')
recall = recall_score(y_test, predictions_SVMD, average='macro')
confusion_mat = confusion_matrix(y_test, predictions_SVMD)

end = time.time()
print(end - start)

# Print the evaluation metrics and confusion matrix
print("Accuracy_SVMD:", accuracy)
print("Precision_SVMD:", precision)
print("Recall_SVMD:", recall)
print("Confusion Matrix_SVMD:")
print(confusion_mat)
```

```
1925.7620568275452

0.03128767013549805

Accuracy_SVMD: 0.78095

Precision_SVMD: 0.6770057099086038

Recall_SVMD: 0.5627796728995796

Confusion Matrix_SVMD:

[[ 832    0    1 ...    0    0    0]

 [   0 1051    1 ...    0    0    0]

 [   1    0  928 ...    0    0    7]

 ...

 [   0    0    2 ...   45    0    0]

 [   0    3    1 ...    0    1    0]

 [   0    0   50 ...    0    0   21]]
```

```
/suphys/mlan9395/miniconda3/lib/python3.10/site-packages/sklearn/metrics/_classification.py:1344: UndefinedMetricWarning: Precision is ill-defined and being set to 0.0 in labels with no predicted samples. Use `zero_division` parameter to control this behavior.

  _warn_prf(average, modifier, msg_start, len(result))
```

SVM with hyperparamter tuning

In [7]:
```
# Define the parameter grid
param_grid = {
    'C': [ 0.1, 1.0, 10, 100],
    'gamma': ['scale', 'auto']
}
# Create an instance of the SVM model
svm_model_param_grid = SVC(kernel='rbf')


# Split the data into training and testing sets for gridsearch
X_train_Grid, X_test_Grid, y_train_Grid, y_test_Grid = train_test_split(train_images

#Create an instance of the GridSearchCV with the SVM model and parameter grid
grid_search = GridSearchCV(svm_model_param_grid, param_grid, cv=5, scoring='accuracy

start = time.time()
# Perform grid search on the training data
grid_search.fit(X_train_Grid, y_train_Grid)
end = time.time()
print(end - start)

# Get the best parameters and best model
best_params = grid_search.best_params_
best_model = grid_search.best_estimator_

# Make predictions on the test set using the best model
predictions_SVMBestparam = best_model.predict(X_test_Grid)

start = time.time()
# Calculate evaluation metrics
accuracy = accuracy_score(y_test_Grid, predictions_SVMBestparam)
precision = precision_score(y_test_Grid, predictions_SVMBestparam, average='macro')
recall = recall_score(y_test_Grid, predictions_SVMBestparam, average='macro')
confusion_mat = confusion_matrix(y_test_Grid, predictions_SVMBestparam)

end = time.time()
print(end - start)

# Print the evaluation metrics and confusion matrix
print("Best Parameters:", best_params)
print("Accuracy_SVMBestparam:", accuracy)
print("Precision_SVMBestparam:", precision)
print("Recall_SVMBestparam:", recall)
print("Confusion Matrix_SVMBestparam:")
print(confusion_mat)
```

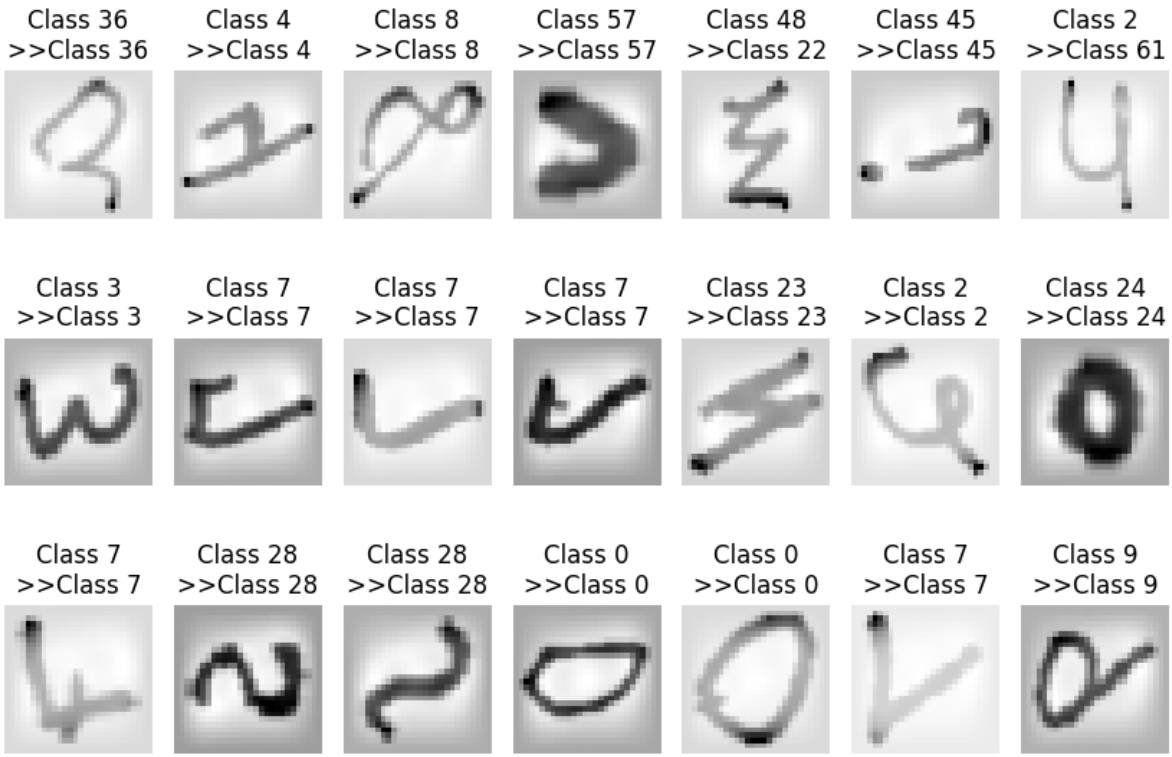
```
Fitting 5 folds for each of 8 candidates, totalling 40 fits
```

```
/opt/conda/lib/python3.10/site-packages/scipy/__init__.py:146: UserWarning: A NumPy version >=1.16.5 and <1.23.0 is required for this version of SciPy (detected version 1.23.5
  warnings.warn(f"A NumPy version >={np_minversion} and <{np_maxversion}"
/opt/conda/lib/python3.10/site-packages/scipy/__init__.py:146: UserWarning: A NumPy version >=1.16.5 and <1.23.0 is required for this version of SciPy (detected version 1.23.5
  warnings.warn(f"A NumPy version >={np_minversion} and <{np_maxversion}"
/opt/conda/lib/python3.10/site-packages/scipy/__init__.py:146: UserWarning: A NumPy version >=1.16.5 and <1.23.0 is required for this version of SciPy (detected version 1.23.5
  warnings.warn(f"A NumPy version >={np_minversion} and <{np_maxversion}"
/opt/conda/lib/python3.10/site-packages/scipy/__init__.py:146: UserWarning: A NumPy version >=1.16.5 and <1.23.0 is required for this version of SciPy (detected version 1.23.5
  warnings.warn(f"A NumPy version >={np_minversion} and <{np_maxversion}"
2376.185211658478
0.14995598793029785
Best Parameters: {'C': 10, 'gamma': 'auto'}
Accuracy_SVMBestparam: 0.766675
Precision_SVMBestparam: 0.651683821326552
Recall_SVMBestparam: 0.5713692727767514
Confusion Matrix_SVMBestparam:
[[2920    1    1 ...    0    0    0]
 [   0 4166    9 ...    0    0    0]
 [  12    4 3624 ...    2    0   32]
 ...
 [   0    1    5 ...  190    1    0]
 [   0    5    2 ...    1   38    0]
 [   0    0  146 ...    0    0   92]]
```

SVM with best parameters

In [9]:
```
# Create an instance of the SVM model with the best hyperparameters
svm_model = SVC(kernel='rbf', C=10, gamma='auto')

start = time.time()
svm_model.fit(X_train, y_train)
end = time.time()
print(end - start)

# Make predictions
predictions_SVMNew = svm_model.predict(X_test)

start = time.time()
# Calculate evaluation metrics
accuracy = accuracy_score(y_test, predictions_SVMNew)
precision = precision_score(y_test, predictions_SVMNew, average='macro')
recall = recall_score(y_test, predictions_SVMNew, average='macro')
confusion_mat = confusion_matrix(y_test, predictions_SVMNew)
end = time.time()
print(end - start)

# Print the evaluation metrics and confusion matrix
print("Accuracy_SVMNew:", accuracy)
print("Precision_SVMNew:", precision)
print("Recall_SVMNew:", recall)
print("Confusion Matrix_SVMNew:")
print(confusion_mat)

# Plot examples of best paramterized SVM
plot_examples(X_test, predictions_SVMNew, y_test, n_rows=3, n_cols=7)
```

```
1378.204790353775
0.0479891300201416
Accuracy_SVMNew: 0.8042
Precision_SVMNew: 0.7041671297346184
Recall_SVMNew: 0.6339506453348212
Confusion Matrix_SVMNew:
[[ 777    0    1 ...    0    0    0]
 [   0 1029    0 ...    0    0    0]
 [   2    2  913 ...    0    0    8]
 ...
 [   0    0    1 ...   53    1    0]
 [   0    0    1 ...    0   16    0]
 [   0    0   29 ...    0    0   39]]
```



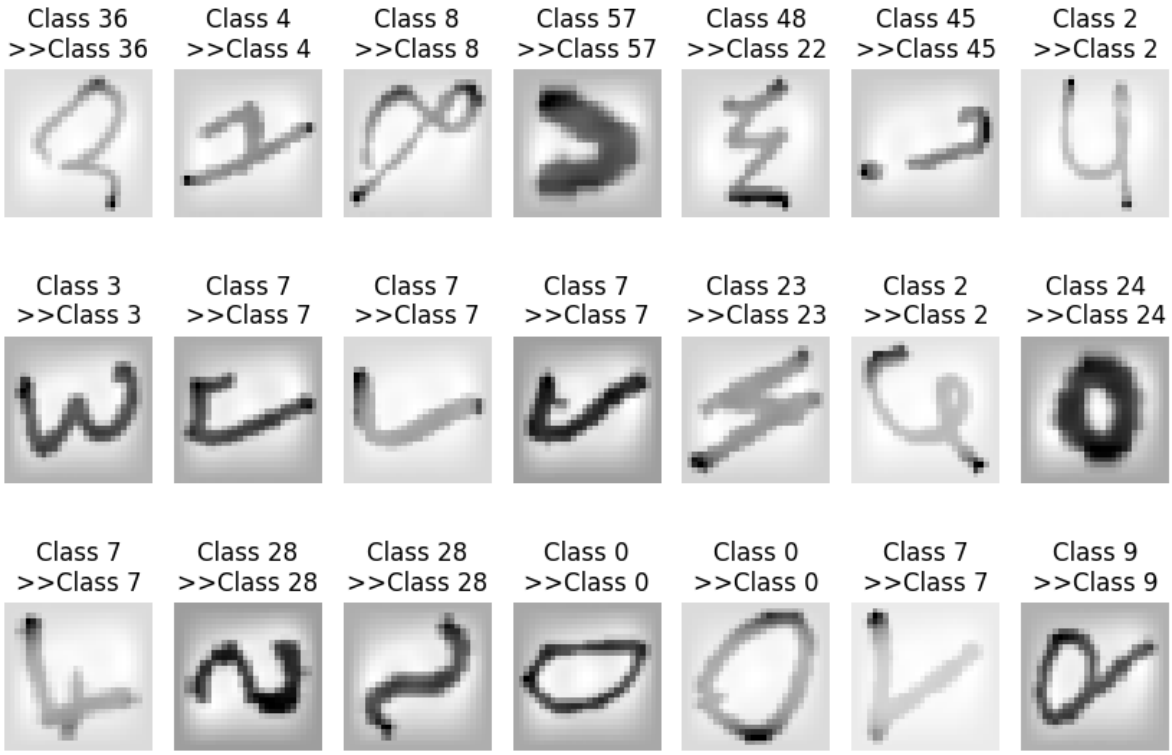
SVM with Regularization

In [10]:

```
# Access the value of the C parameter
best_C = best_params['C']

# Apply regularization by reducing C value
regularized_C = best_C * 0.1  # Example: reduce C by 10 times

# Update the C parameter in the best_params dictionary
best_params['C'] = regularized_C

# Initialize a new SVM classifier with the regularized C value
regularized_svm = SVC(**best_params)

start = time.time()

# Fit the regularized SVM on the training data
regularized_svm.fit(X_train, y_train)

end = time.time()
print(end - start)

# Make predictions on the test set using the regularized model
predictions_Regularized = regularized_svm.predict(X_test)
```

```
start = time.time()
# Calculate evaluation metrics
accuracy = accuracy_score(y_test, predictions_Regularized)
precision = precision_score(y_test, predictions_Regularized, average='macro')
recall = recall_score(y_test, predictions_Regularized, average='macro')
confusion_mat = confusion_matrix(y_test, predictions_Regularized)

end = time.time()
print(end - start)

# Print the evaluation metrics and confusion matrix
print("Regularized Parameters:", best_params)
print("Accuracy_Regularized:", accuracy)
print("Precision_Regularized:", precision)
print("Recall_Regularized:", recall)
print("Confusion Matrix_Regularized:")
print(confusion_mat)

# Plot examples of Regularized SVM
plot_examples(X_test, predictions_Regularized, y_test, n_rows=3, n_cols=7)
```

```
1454.2729988098145
/opt/conda/lib/python3.10/site-packages/sklearn/metrics/_classification.py:1344: UndefinedMetricWarning: Precision is ill-defined and being set to 0.0 in labels with no predicted samples. Use `zero_division` parameter to control this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
0.04742550849914551
Regularized Parameters: {'C': 1.0, 'gamma': 'auto'}
Accuracy_Regularized: 0.79505
Precision_Regularized: 0.6775295893378837
Recall_Regularized: 0.6022630503219472
Confusion Matrix_Regularized:
[[ 830    0    0 ...    0    0    0]
 [   0 1036    0 ...    0    0    0]
 [   3    2  919 ...    0    0    7]
 ...
 [   0    0    1 ...   54    1    0]
 [   0    3    1 ...    1    8    0]
 [   0    0   42 ...    0    0   26]]
```



CNN Experiments with varying learning_rates and dense_units_options

In [3]:
```
import numpy as np
import time
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Conv2D, MaxPooling2D, Dense, Flatten, Dropout,
from tensorflow.keras.utils import to_categorical
from tensorflow.keras.callbacks import EarlyStopping, ModelCheckpoint
from tensorflow.keras.optimizers import Adam
from sklearn.metrics import classification_report, confusion_matrix, accuracy_score

# Reshape and normalize the data
train_data = train_data.reshape((-1, 28, 28, 1)) / 255.0
test_data = test_data.reshape((-1, 28, 28, 1)) / 255.0

# Convert the labels to one-hot encoding
train_labels = to_categorical(train_labels)
test_labels = to_categorical(test_labels)

# Define the function to build the model
def build_model(units, learning_rate):
    model = Sequential()
    model.add(Conv2D(64, (3, 3), activation='relu', input_shape=(28, 28, 1)))
    model.add(BatchNormalization())
    model.add(MaxPooling2D((2, 2)))
    model.add(Dropout(0.3))

    model.add(Conv2D(128, (3, 3), activation='relu'))
    model.add(BatchNormalization())
    model.add(MaxPooling2D((2, 2)))
    model.add(Dropout(0.3))

    model.add(Flatten())

    model.add(Dense(units, activation='relu'))
    model.add(BatchNormalization())
    model.add(Dropout(0.3))

    model.add(Dense(len(train_labels[0]), activation='softmax'))

    optimizer = Adam(learning_rate=learning_rate)
    model.compile(optimizer=optimizer, loss='categorical_crossentropy', metrics=['ac

    return model

# Define the parameter grid
dense_units_options = list(range(128, 512 + 1, 128))  # 10 values from 128 to 512
learning_rates = [1e-2, 1e-3, 1e-4, 1e-5]  # 4 values for learning rate

# Create callbacks
early_stopping = EarlyStopping(monitor='val_loss', patience=3)
model_checkpoint = ModelCheckpoint('model.h5', save_best_only=True)

# Record results
results = []

# Perform manual hyperparameter tuning
for units in dense_units_options:
```

```
    for learning_rate in learning_rates:
        start_time = time.time()

        model = build_model(units, learning_rate)

        model.fit(train_data, train_labels, epochs=10, batch_size=128, validation_sp
                  callbacks=[early_stopping, model_checkpoint])

        test_loss, test_acc = model.evaluate(test_data, test_labels)
        end_time = time.time()

        # Generate predictions
        preds = model.predict(test_data)
        preds_classes = np.argmax(preds, axis=1)

        # Convert back to integer format
        test_labels_int = np.argmax(test_labels, axis=1)

        accuracy = accuracy_score(test_labels_int, preds_classes)
        precision = precision_score(test_labels_int, preds_classes, average='macro',
        recall = recall_score(test_labels_int, preds_classes, average='macro', zero_

        results.append({
            'learning_rate': learning_rate,
            'units': units,
            'accuracy': accuracy,
            'precision': precision,
            'recall': recall,
            'runtime': end_time - start_time
        })

        print("Learning rate:", learning_rate)
        print("Units:", units)
        print("Accuracy:", accuracy)
        print("Precision:", precision)
        print("Recall:", recall)
        print("Runtime:", end_time - start_time)
        print()

# Print final results
for result in results:
    print("Learning rate:", result['learning_rate'])
    print("Units:", result['units'])
    print("Accuracy:", result['accuracy'])
    print("Precision:", result['precision'])
    print("Recall:", result['recall'])
    print("Runtime:", result['runtime'])
    print()
```

```
Epoch 1/10
704/704 [==============================] - 94s 131ms/step - loss: 0.7751 - accuracy:
0.7564 - val_loss: 0.5234 - val_accuracy: 0.8188
Epoch 2/10
704/704 [==============================] - 90s 128ms/step - loss: 0.5641 - accuracy:
0.8077 - val_loss: 0.4837 - val_accuracy: 0.8300
Epoch 3/10
704/704 [==============================] - 90s 128ms/step - loss: 0.5172 - accuracy:
0.8206 - val_loss: 0.4549 - val_accuracy: 0.8425
Epoch 4/10
704/704 [==============================] - 91s 129ms/step - loss: 0.4919 - accuracy:
0.8268 - val_loss: 0.4275 - val_accuracy: 0.8488
Epoch 5/10
704/704 [==============================] - 92s 131ms/step - loss: 0.4751 - accuracy:
0.8324 - val_loss: 0.4386 - val_accuracy: 0.8447
Epoch 6/10
704/704 [==============================] - 91s 130ms/step - loss: 0.4601 - accuracy:
0.8361 - val_loss: 0.4170 - val_accuracy: 0.8531
Epoch 7/10
704/704 [==============================] - 92s 130ms/step - loss: 0.4534 - accuracy:
0.8368 - val_loss: 0.4127 - val_accuracy: 0.8506
Epoch 8/10
704/704 [==============================] - 92s 131ms/step - loss: 0.4396 - accuracy:
0.8410 - val_loss: 0.4221 - val_accuracy: 0.8507
Epoch 9/10
704/704 [==============================] - 90s 128ms/step - loss: 0.4317 - accuracy:
0.8431 - val_loss: 0.4083 - val_accuracy: 0.8535
Epoch 10/10
704/704 [==============================] - 89s 127ms/step - loss: 0.4262 - accuracy:
0.8460 - val_loss: 0.4129 - val_accuracy: 0.8529
625/625 [==============================] - 7s 11ms/step - loss: 0.4304 - accuracy:
0.8477
625/625 [==============================] - 7s 11ms/step
Learning rate: 0.01
Units: 128
Accuracy: 0.8477
Precision: 0.8043123489373756
Recall: 0.6988400193387927
Runtime: 919.8214240074158

Epoch 1/10
704/704 [==============================] - 92s 129ms/step - loss: 0.9372 - accuracy:
0.7319 - val_loss: 0.6508 - val_accuracy: 0.7973
Epoch 2/10
704/704 [==============================] - 91s 129ms/step - loss: 0.5620 - accuracy:
0.8119 - val_loss: 0.4706 - val_accuracy: 0.8392
Epoch 3/10
704/704 [==============================] - 90s 128ms/step - loss: 0.5042 - accuracy:
0.8271 - val_loss: 0.4387 - val_accuracy: 0.8479
Epoch 4/10
704/704 [==============================] - 91s 129ms/step - loss: 0.4696 - accuracy:
0.8368 - val_loss: 0.4292 - val_accuracy: 0.8491
Epoch 5/10
704/704 [==============================] - 91s 129ms/step - loss: 0.4490 - accuracy:
0.8414 - val_loss: 0.4260 - val_accuracy: 0.8506
Epoch 6/10
704/704 [==============================] - 91s 130ms/step - loss: 0.4352 - accuracy:
0.8447 - val_loss: 0.4094 - val_accuracy: 0.8575
Epoch 7/10
704/704 [==============================] - 91s 129ms/step - loss: 0.4191 - accuracy:
0.8489 - val_loss: 0.4011 - val_accuracy: 0.8548
Epoch 8/10
704/704 [==============================] - 92s 130ms/step - loss: 0.4114 - accuracy:
0.8498 - val_loss: 0.4013 - val_accuracy: 0.8601
```

```
Epoch 9/10
704/704 [==============================] - 90s 128ms/step - loss: 0.4024 - accuracy:
0.8539 - val_loss: 0.3975 - val_accuracy: 0.8568
Epoch 10/10
704/704 [==============================] - 89s 127ms/step - loss: 0.3928 - accuracy:
0.8559 - val_loss: 0.4042 - val_accuracy: 0.8557
625/625 [==============================] - 7s 11ms/step - loss: 0.4190 - accuracy:
0.8526
625/625 [==============================] - 7s 11ms/step
Learning rate: 0.001
Units: 128
Accuracy: 0.85255
Precision: 0.7867328400171182
Recall: 0.707327582008939
Runtime: 915.5359442234039

Epoch 1/10
704/704 [==============================] - 90s 126ms/step - loss: 1.8558 - accuracy:
0.5451 - val_loss: 1.3337 - val_accuracy: 0.6647
Epoch 2/10
704/704 [==============================] - 90s 127ms/step - loss: 1.0449 - accuracy:
0.7092 - val_loss: 0.7781 - val_accuracy: 0.7750
Epoch 3/10
704/704 [==============================] - 89s 126ms/step - loss: 0.8387 - accuracy:
0.7549 - val_loss: 0.6669 - val_accuracy: 0.8006
Epoch 4/10
704/704 [==============================] - 90s 127ms/step - loss: 0.7355 - accuracy:
0.7750 - val_loss: 0.5861 - val_accuracy: 0.8153
Epoch 5/10
704/704 [==============================] - 90s 127ms/step - loss: 0.6693 - accuracy:
0.7913 - val_loss: 0.5480 - val_accuracy: 0.8231
Epoch 6/10
704/704 [==============================] - 91s 129ms/step - loss: 0.6227 - accuracy:
0.8007 - val_loss: 0.5232 - val_accuracy: 0.8296
Epoch 7/10
704/704 [==============================] - 90s 128ms/step - loss: 0.5872 - accuracy:
0.8103 - val_loss: 0.5001 - val_accuracy: 0.8354
Epoch 8/10
704/704 [==============================] - 90s 127ms/step - loss: 0.5607 - accuracy:
0.8161 - val_loss: 0.4794 - val_accuracy: 0.8405
Epoch 9/10
704/704 [==============================] - 88s 126ms/step - loss: 0.5394 - accuracy:
0.8214 - val_loss: 0.4703 - val_accuracy: 0.8414
Epoch 10/10
704/704 [==============================] - 89s 126ms/step - loss: 0.5197 - accuracy:
0.8265 - val_loss: 0.4558 - val_accuracy: 0.8434
625/625 [==============================] - 7s 11ms/step - loss: 0.4596 - accuracy:
0.8430
625/625 [==============================] - 7s 11ms/step
Learning rate: 0.0001
Units: 128
Accuracy: 0.843
Precision: 0.786125959157098
Recall: 0.6869673416696267
Runtime: 901.746609210968

Epoch 1/10
704/704 [==============================] - 89s 125ms/step - loss: 3.5610 - accuracy:
0.2011 - val_loss: 2.5119 - val_accuracy: 0.4362
Epoch 2/10
704/704 [==============================] - 86s 123ms/step - loss: 2.3527 - accuracy:
0.4388 - val_loss: 1.8504 - val_accuracy: 0.5708
Epoch 3/10
704/704 [==============================] - 87s 124ms/step - loss: 1.9694 - accuracy:
```

```
0.5211 - val_loss: 1.6330 - val_accuracy: 0.6141
Epoch 4/10
704/704 [==============================] - 89s 126ms/step - loss: 1.7571 - accuracy:
0.5642 - val_loss: 1.4801 - val_accuracy: 0.6429
Epoch 5/10
704/704 [==============================] - 87s 124ms/step - loss: 1.5993 - accuracy:
0.5968 - val_loss: 1.3559 - val_accuracy: 0.6660
Epoch 6/10
704/704 [==============================] - 88s 125ms/step - loss: 1.4837 - accuracy:
0.6179 - val_loss: 1.2523 - val_accuracy: 0.6847
Epoch 7/10
704/704 [==============================] - 88s 125ms/step - loss: 1.3779 - accuracy:
0.6399 - val_loss: 1.1683 - val_accuracy: 0.7006
Epoch 8/10
704/704 [==============================] - 89s 126ms/step - loss: 1.2928 - accuracy:
0.6607 - val_loss: 1.0958 - val_accuracy: 0.7164
Epoch 9/10
704/704 [==============================] - 88s 126ms/step - loss: 1.2194 - accuracy:
0.6743 - val_loss: 1.0361 - val_accuracy: 0.7265
Epoch 10/10
704/704 [==============================] - 87s 124ms/step - loss: 1.1573 - accuracy:
0.6891 - val_loss: 0.9822 - val_accuracy: 0.7369
625/625 [==============================] - 6s 10ms/step - loss: 0.9743 - accuracy:
0.7347
625/625 [==============================] - 7s 10ms/step
Learning rate: 1e-05
Units: 128
Accuracy: 0.73465
Precision: 0.7162887549989162
Recall: 0.47558125347175484
Runtime: 885.5641613006592

Epoch 1/10
704/704 [==============================] - 94s 132ms/step - loss: 0.7820 - accuracy:
0.7578 - val_loss: 0.5630 - val_accuracy: 0.8104
Epoch 2/10
704/704 [==============================] - 91s 130ms/step - loss: 0.5575 - accuracy:
0.8092 - val_loss: 0.4729 - val_accuracy: 0.8359
Epoch 3/10
704/704 [==============================] - 92s 130ms/step - loss: 0.5061 - accuracy:
0.8226 - val_loss: 0.4778 - val_accuracy: 0.8396
Epoch 4/10
704/704 [==============================] - 96s 136ms/step - loss: 0.4787 - accuracy:
0.8310 - val_loss: 0.4482 - val_accuracy: 0.8374
Epoch 5/10
704/704 [==============================] - 94s 133ms/step - loss: 0.4574 - accuracy:
0.8370 - val_loss: 0.4365 - val_accuracy: 0.8489
Epoch 6/10
704/704 [==============================] - 92s 130ms/step - loss: 0.4473 - accuracy:
0.8396 - val_loss: 0.4465 - val_accuracy: 0.8459
Epoch 7/10
704/704 [==============================] - 93s 132ms/step - loss: 0.4364 - accuracy:
0.8432 - val_loss: 0.4151 - val_accuracy: 0.8531
Epoch 8/10
704/704 [==============================] - 92s 131ms/step - loss: 0.4266 - accuracy:
0.8447 - val_loss: 0.4369 - val_accuracy: 0.8388
Epoch 9/10
704/704 [==============================] - 91s 130ms/step - loss: 0.4209 - accuracy:
0.8456 - val_loss: 0.4305 - val_accuracy: 0.8474
Epoch 10/10
704/704 [==============================] - 91s 129ms/step - loss: 0.4103 - accuracy:
0.8486 - val_loss: 0.4248 - val_accuracy: 0.8481
625/625 [==============================] - 7s 11ms/step - loss: 0.4298 - accuracy:
0.8505
```

```
625/625 [==============================] - 7s 11ms/step
Learning rate: 0.01
Units: 256
Accuracy: 0.85055
Precision: 0.8134135377605697
Recall: 0.7079299811099503
Runtime: 990.635219335556

Epoch 1/10
704/704 [==============================] - 92s 129ms/step - loss: 0.8462 - accuracy:
0.7467 - val_loss: 0.5616 - val_accuracy: 0.8180
Epoch 2/10
704/704 [==============================] - 89s 127ms/step - loss: 0.5305 - accuracy:
0.8183 - val_loss: 0.4596 - val_accuracy: 0.8417
Epoch 3/10
704/704 [==============================] - 90s 128ms/step - loss: 0.4730 - accuracy:
0.8350 - val_loss: 0.4297 - val_accuracy: 0.8502
Epoch 4/10
704/704 [==============================] - 91s 129ms/step - loss: 0.4441 - accuracy:
0.8429 - val_loss: 0.4255 - val_accuracy: 0.8490
Epoch 5/10
704/704 [==============================] - 91s 129ms/step - loss: 0.4249 - accuracy:
0.8463 - val_loss: 0.4127 - val_accuracy: 0.8448
Epoch 6/10
704/704 [==============================] - 91s 130ms/step - loss: 0.4065 - accuracy:
0.8515 - val_loss: 0.3999 - val_accuracy: 0.8518
Epoch 7/10
704/704 [==============================] - 91s 129ms/step - loss: 0.3911 - accuracy:
0.8558 - val_loss: 0.3967 - val_accuracy: 0.8589
Epoch 8/10
704/704 [==============================] - 90s 128ms/step - loss: 0.3816 - accuracy:
0.8582 - val_loss: 0.4018 - val_accuracy: 0.8547
Epoch 9/10
704/704 [==============================] - 89s 126ms/step - loss: 0.3693 - accuracy:
0.8619 - val_loss: 0.3934 - val_accuracy: 0.8603
Epoch 10/10
704/704 [==============================] - 89s 126ms/step - loss: 0.3607 - accuracy:
0.8642 - val_loss: 0.3876 - val_accuracy: 0.8601
625/625 [==============================] - 7s 11ms/step - loss: 0.4048 - accuracy:
0.8572
625/625 [==============================] - 7s 11ms/step
Learning rate: 0.001
Units: 256
Accuracy: 0.85725
Precision: 0.7893143274662201
Recall: 0.7183029748170656
Runtime: 909.7056584358215

Epoch 1/10
704/704 [==============================] - 92s 129ms/step - loss: 1.6217 - accuracy:
0.5851 - val_loss: 1.0392 - val_accuracy: 0.7184
Epoch 2/10
704/704 [==============================] - 91s 129ms/step - loss: 0.8919 - accuracy:
0.7355 - val_loss: 0.6605 - val_accuracy: 0.7935
Epoch 3/10
704/704 [==============================] - 90s 128ms/step - loss: 0.7346 - accuracy:
0.7727 - val_loss: 0.5750 - val_accuracy: 0.8150
Epoch 4/10
704/704 [==============================] - 90s 128ms/step - loss: 0.6565 - accuracy:
0.7895 - val_loss: 0.5346 - val_accuracy: 0.8231
Epoch 5/10
704/704 [==============================] - 91s 129ms/step - loss: 0.6066 - accuracy:
0.8029 - val_loss: 0.5024 - val_accuracy: 0.8307
Epoch 6/10
```

```
704/704 [==============================] - 90s 129ms/step - loss: 0.5661 - accuracy:
0.8131 - val_loss: 0.4779 - val_accuracy: 0.8378
Epoch 7/10
704/704 [==============================] - 90s 128ms/step - loss: 0.5386 - accuracy:
0.8196 - val_loss: 0.4647 - val_accuracy: 0.8417
Epoch 8/10
704/704 [==============================] - 90s 127ms/step - loss: 0.5176 - accuracy:
0.8254 - val_loss: 0.4527 - val_accuracy: 0.8441
Epoch 9/10
704/704 [==============================] - 90s 127ms/step - loss: 0.4966 - accuracy:
0.8299 - val_loss: 0.4425 - val_accuracy: 0.8454
Epoch 10/10
704/704 [==============================] - 90s 128ms/step - loss: 0.4800 - accuracy:
0.8344 - val_loss: 0.4327 - val_accuracy: 0.8488
625/625 [==============================] - 7s 11ms/step - loss: 0.4431 - accuracy:
0.8464
625/625 [==============================] - 7s 11ms/step
Learning rate: 0.0001
Units: 256
Accuracy: 0.8464
Precision: 0.7865026115391711
Recall: 0.6944006943451094
Runtime: 911.2518379688263

Epoch 1/10
704/704 [==============================] - 93s 129ms/step - loss: 3.4790 - accuracy:
0.2348 - val_loss: 2.2663 - val_accuracy: 0.4781
Epoch 2/10
704/704 [==============================] - 92s 130ms/step - loss: 2.1198 - accuracy:
0.4841 - val_loss: 1.5505 - val_accuracy: 0.6142
Epoch 3/10
704/704 [==============================] - 92s 131ms/step - loss: 1.7303 - accuracy:
0.5613 - val_loss: 1.3260 - val_accuracy: 0.6611
Epoch 4/10
704/704 [==============================] - 92s 130ms/step - loss: 1.5093 - accuracy:
0.6047 - val_loss: 1.1733 - val_accuracy: 0.6919
Epoch 5/10
704/704 [==============================] - 93s 132ms/step - loss: 1.3563 - accuracy:
0.6343 - val_loss: 1.0655 - val_accuracy: 0.7136
Epoch 6/10
704/704 [==============================] - 102s 144ms/step - loss: 1.2381 - accurac
y: 0.6615 - val_loss: 0.9814 - val_accuracy: 0.7302
Epoch 7/10
704/704 [==============================] - 93s 131ms/step - loss: 1.1483 - accuracy:
0.6790 - val_loss: 0.9133 - val_accuracy: 0.7439
Epoch 8/10
704/704 [==============================] - 94s 134ms/step - loss: 1.0787 - accuracy:
0.6953 - val_loss: 0.8616 - val_accuracy: 0.7541
Epoch 9/10
704/704 [==============================] - 96s 136ms/step - loss: 1.0179 - accuracy:
0.7081 - val_loss: 0.8148 - val_accuracy: 0.7647
Epoch 10/10
704/704 [==============================] - 96s 136ms/step - loss: 0.9673 - accuracy:
0.7183 - val_loss: 0.7792 - val_accuracy: 0.7728
625/625 [==============================] - 8s 12ms/step - loss: 0.7850 - accuracy:
0.7670
625/625 [==============================] - 7s 12ms/step
Learning rate: 1e-05
Units: 256
Accuracy: 0.76695
Precision: 0.6844942561186504
Recall: 0.5549426168871804
Runtime: 991.9178223609924
```

```
Epoch 1/10
704/704 [==============================] - 98s 137ms/step - loss: 0.7771 - accuracy:
0.7574 - val_loss: 0.5857 - val_accuracy: 0.7930
Epoch 2/10
704/704 [==============================] - 96s 136ms/step - loss: 0.5504 - accuracy:
0.8120 - val_loss: 0.7374 - val_accuracy: 0.8183
Epoch 3/10
704/704 [==============================] - 95s 135ms/step - loss: 0.4989 - accuracy:
0.8262 - val_loss: 0.4714 - val_accuracy: 0.8364
Epoch 4/10
704/704 [==============================] - 96s 136ms/step - loss: 0.4784 - accuracy:
0.8307 - val_loss: 0.4543 - val_accuracy: 0.8425
Epoch 5/10
704/704 [==============================] - 95s 136ms/step - loss: 0.4600 - accuracy:
0.8362 - val_loss: 0.4374 - val_accuracy: 0.8482
Epoch 6/10
704/704 [==============================] - 95s 134ms/step - loss: 0.4468 - accuracy:
0.8390 - val_loss: 0.4356 - val_accuracy: 0.8438
Epoch 7/10
704/704 [==============================] - 94s 134ms/step - loss: 0.4355 - accuracy:
0.8420 - val_loss: 0.4349 - val_accuracy: 0.8498
Epoch 8/10
704/704 [==============================] - 94s 134ms/step - loss: 0.4244 - accuracy:
0.8453 - val_loss: 0.4450 - val_accuracy: 0.8421
Epoch 9/10
704/704 [==============================] - 95s 135ms/step - loss: 0.4175 - accuracy:
0.8481 - val_loss: 0.4446 - val_accuracy: 0.8392
Epoch 10/10
704/704 [==============================] - 95s 135ms/step - loss: 0.4109 - accuracy:
0.8473 - val_loss: 0.4226 - val_accuracy: 0.8521
625/625 [==============================] - 8s 13ms/step - loss: 0.4446 - accuracy:
0.8467
625/625 [==============================] - 8s 12ms/step
Learning rate: 0.01
Units: 384
Accuracy: 0.84665
Precision: 0.7905247884057616
Recall: 0.7130944499335667
Runtime: 991.4726612567902

Epoch 1/10
704/704 [==============================] - 97s 136ms/step - loss: 0.8197 - accuracy:
0.7496 - val_loss: 0.5699 - val_accuracy: 0.8108
Epoch 2/10
704/704 [==============================] - 95s 135ms/step - loss: 0.5246 - accuracy:
0.8209 - val_loss: 0.4502 - val_accuracy: 0.8384
Epoch 3/10
704/704 [==============================] - 96s 136ms/step - loss: 0.4687 - accuracy:
0.8347 - val_loss: 0.4225 - val_accuracy: 0.8515
Epoch 4/10
704/704 [==============================] - 96s 136ms/step - loss: 0.4366 - accuracy:
0.8433 - val_loss: 0.4177 - val_accuracy: 0.8516
Epoch 5/10
704/704 [==============================] - 97s 138ms/step - loss: 0.4172 - accuracy:
0.8486 - val_loss: 0.4084 - val_accuracy: 0.8535
Epoch 6/10
704/704 [==============================] - 96s 137ms/step - loss: 0.4008 - accuracy:
0.8532 - val_loss: 0.4102 - val_accuracy: 0.8524
Epoch 7/10
704/704 [==============================] - 97s 137ms/step - loss: 0.3893 - accuracy:
0.8546 - val_loss: 0.4039 - val_accuracy: 0.8553
Epoch 8/10
704/704 [==============================] - 96s 136ms/step - loss: 0.3742 - accuracy:
0.8595 - val_loss: 0.3915 - val_accuracy: 0.8603
```

```
Epoch 9/10
704/704 [==============================] - 97s 138ms/step - loss: 0.3593 - accuracy:
0.8649 - val_loss: 0.4111 - val_accuracy: 0.8559
Epoch 10/10
704/704 [==============================] - 106s 151ms/step - loss: 0.3541 - accurac
y: 0.8663 - val_loss: 0.4045 - val_accuracy: 0.8574
625/625 [==============================] - 9s 15ms/step - loss: 0.4177 - accuracy:
0.8541
625/625 [==============================] - 9s 14ms/step
Learning rate: 0.001
Units: 384
Accuracy: 0.8541
Precision: 0.7530003563189905
Recall: 0.7165125146157162
Runtime: 984.3839774131775

Epoch 1/10
704/704 [==============================] - 114s 160ms/step - loss: 1.5209 - accurac
y: 0.6029 - val_loss: 0.9681 - val_accuracy: 0.7277
Epoch 2/10
704/704 [==============================] - 100s 142ms/step - loss: 0.8376 - accurac
y: 0.7451 - val_loss: 0.6143 - val_accuracy: 0.8062
Epoch 3/10
704/704 [==============================] - 99s 141ms/step - loss: 0.6872 - accuracy:
0.7821 - val_loss: 0.5401 - val_accuracy: 0.8230
Epoch 4/10
704/704 [==============================] - 100s 142ms/step - loss: 0.6162 - accurac
y: 0.7989 - val_loss: 0.5034 - val_accuracy: 0.8319
Epoch 5/10
704/704 [==============================] - 99s 140ms/step - loss: 0.5718 - accuracy:
0.8111 - val_loss: 0.4765 - val_accuracy: 0.8386
Epoch 6/10
704/704 [==============================] - 99s 140ms/step - loss: 0.5344 - accuracy:
0.8194 - val_loss: 0.4610 - val_accuracy: 0.8420
Epoch 7/10
704/704 [==============================] - 99s 140ms/step - loss: 0.5117 - accuracy:
0.8252 - val_loss: 0.4497 - val_accuracy: 0.8454
Epoch 8/10
704/704 [==============================] - 96s 137ms/step - loss: 0.4883 - accuracy:
0.8315 - val_loss: 0.4391 - val_accuracy: 0.8487
Epoch 9/10
704/704 [==============================] - 97s 137ms/step - loss: 0.4730 - accuracy:
0.8366 - val_loss: 0.4291 - val_accuracy: 0.8523
Epoch 10/10
704/704 [==============================] - 104s 148ms/step - loss: 0.4580 - accurac
y: 0.8405 - val_loss: 0.4310 - val_accuracy: 0.8452
625/625 [==============================] - 9s 14ms/step - loss: 0.4393 - accuracy:
0.8433
625/625 [==============================] - 8s 13ms/step
Learning rate: 0.0001
Units: 384
Accuracy: 0.8433
Precision: 0.7891048209621495
Recall: 0.7014753587907294
Runtime: 1054.4654290676117

Epoch 1/10
704/704 [==============================] - 113s 158ms/step - loss: 3.1625 - accurac
y: 0.2915 - val_loss: 2.0095 - val_accuracy: 0.5307
Epoch 2/10
704/704 [==============================] - 105s 149ms/step - loss: 1.9249 - accurac
y: 0.5191 - val_loss: 1.3929 - val_accuracy: 0.6357
Epoch 3/10
704/704 [==============================] - 104s 147ms/step - loss: 1.5803 - accurac
```

```
y: 0.5829 - val_loss: 1.1975 - val_accuracy: 0.6724
Epoch 4/10
704/704 [==============================] - 105s 149ms/step - loss: 1.3831 - accurac
y: 0.6237 - val_loss: 1.0631 - val_accuracy: 0.7012
Epoch 5/10
704/704 [==============================] - 104s 148ms/step - loss: 1.2509 - accurac
y: 0.6508 - val_loss: 0.9703 - val_accuracy: 0.7195
Epoch 6/10
704/704 [==============================] - 107s 153ms/step - loss: 1.1451 - accurac
y: 0.6761 - val_loss: 0.8917 - val_accuracy: 0.7382
Epoch 7/10
704/704 [==============================] - 105s 149ms/step - loss: 1.0663 - accurac
y: 0.6927 - val_loss: 0.8434 - val_accuracy: 0.7503
Epoch 8/10
704/704 [==============================] - 106s 150ms/step - loss: 1.0006 - accurac
y: 0.7074 - val_loss: 0.7892 - val_accuracy: 0.7617
Epoch 9/10
704/704 [==============================] - 104s 148ms/step - loss: 0.9516 - accurac
y: 0.7181 - val_loss: 0.7535 - val_accuracy: 0.7695
Epoch 10/10
704/704 [==============================] - 103s 147ms/step - loss: 0.9097 - accurac
y: 0.7285 - val_loss: 0.7191 - val_accuracy: 0.7776
625/625 [==============================] - 8s 14ms/step - loss: 0.7206 - accuracy:
0.7772
625/625 [==============================] - 8s 13ms/step
Learning rate: 1e-05
Units: 384
Accuracy: 0.77725
Precision: 0.690018898958171
Recall: 0.5786048612733493
Runtime: 1114.2524206638336

Epoch 1/10
704/704 [==============================] - 111s 156ms/step - loss: 0.8052 - accurac
y: 0.7523 - val_loss: 0.5326 - val_accuracy: 0.8195
Epoch 2/10
704/704 [==============================] - 108s 153ms/step - loss: 0.5536 - accurac
y: 0.8114 - val_loss: 0.5141 - val_accuracy: 0.8290
Epoch 3/10
704/704 [==============================] - 109s 155ms/step - loss: 0.5092 - accurac
y: 0.8212 - val_loss: 0.5487 - val_accuracy: 0.8251
Epoch 4/10
704/704 [==============================] - 109s 154ms/step - loss: 0.4825 - accurac
y: 0.8295 - val_loss: 0.4652 - val_accuracy: 0.8374
Epoch 5/10
704/704 [==============================] - 109s 155ms/step - loss: 0.4670 - accurac
y: 0.8330 - val_loss: 0.4528 - val_accuracy: 0.8490
Epoch 6/10
704/704 [==============================] - 107s 153ms/step - loss: 0.4557 - accurac
y: 0.8369 - val_loss: 0.4435 - val_accuracy: 0.8449
Epoch 7/10
704/704 [==============================] - 108s 153ms/step - loss: 0.4455 - accurac
y: 0.8388 - val_loss: 0.4561 - val_accuracy: 0.8357
Epoch 8/10
704/704 [==============================] - 110s 156ms/step - loss: 0.4346 - accurac
y: 0.8416 - val_loss: 0.4329 - val_accuracy: 0.8476
Epoch 9/10
704/704 [==============================] - 108s 153ms/step - loss: 0.4245 - accurac
y: 0.8450 - val_loss: 0.4492 - val_accuracy: 0.8448
Epoch 10/10
704/704 [==============================] - 108s 154ms/step - loss: 0.4203 - accurac
y: 0.8458 - val_loss: 0.4303 - val_accuracy: 0.8485
625/625 [==============================] - 9s 15ms/step - loss: 0.4465 - accuracy:
0.8453
```

```
625/625 [==============================] - 9s 14ms/step
Learning rate: 0.01
Units: 512
Accuracy: 0.84525
Precision: 0.7942784310891914
Recall: 0.7128416009952518
Runtime: 1098.0348672866821

Epoch 1/10
704/704 [==============================] - 113s 158ms/step - loss: 0.8159 - accurac
y: 0.7522 - val_loss: 0.6532 - val_accuracy: 0.7742
Epoch 2/10
704/704 [==============================] - 109s 155ms/step - loss: 0.5232 - accurac
y: 0.8205 - val_loss: 0.4610 - val_accuracy: 0.8402
Epoch 3/10
704/704 [==============================] - 109s 154ms/step - loss: 0.4649 - accurac
y: 0.8348 - val_loss: 0.4390 - val_accuracy: 0.8397
Epoch 4/10
704/704 [==============================] - 108s 154ms/step - loss: 0.4367 - accurac
y: 0.8422 - val_loss: 0.4354 - val_accuracy: 0.8441
Epoch 5/10
704/704 [==============================] - 111s 158ms/step - loss: 0.4122 - accurac
y: 0.8492 - val_loss: 0.4147 - val_accuracy: 0.8561
Epoch 6/10
704/704 [==============================] - 107s 152ms/step - loss: 0.3977 - accurac
y: 0.8527 - val_loss: 0.4080 - val_accuracy: 0.8558
Epoch 7/10
704/704 [==============================] - 107s 152ms/step - loss: 0.3819 - accurac
y: 0.8573 - val_loss: 0.4029 - val_accuracy: 0.8512
Epoch 8/10
704/704 [==============================] - 109s 154ms/step - loss: 0.3658 - accurac
y: 0.8619 - val_loss: 0.3970 - val_accuracy: 0.8589
Epoch 9/10
704/704 [==============================] - 107s 151ms/step - loss: 0.3554 - accurac
y: 0.8648 - val_loss: 0.3986 - val_accuracy: 0.8603
Epoch 10/10
704/704 [==============================] - 105s 150ms/step - loss: 0.3428 - accurac
y: 0.8687 - val_loss: 0.4071 - val_accuracy: 0.8555
625/625 [==============================] - 9s 14ms/step - loss: 0.4220 - accuracy:
0.8522
625/625 [==============================] - 9s 14ms/step
Learning rate: 0.001
Units: 512
Accuracy: 0.85215
Precision: 0.7637378673224909
Recall: 0.7207188433737077
Runtime: 1113.2304723262787

Epoch 1/10
704/704 [==============================] - 108s 152ms/step - loss: 1.4339 - accurac
y: 0.6185 - val_loss: 0.8721 - val_accuracy: 0.7425
Epoch 2/10
704/704 [==============================] - 105s 150ms/step - loss: 0.7927 - accurac
y: 0.7543 - val_loss: 0.5985 - val_accuracy: 0.8063
Epoch 3/10
704/704 [==============================] - 107s 152ms/step - loss: 0.6607 - accurac
y: 0.7871 - val_loss: 0.5356 - val_accuracy: 0.8193
Epoch 4/10
704/704 [==============================] - 105s 150ms/step - loss: 0.5994 - accurac
y: 0.8028 - val_loss: 0.4987 - val_accuracy: 0.8321
Epoch 5/10
704/704 [==============================] - 107s 152ms/step - loss: 0.5549 - accurac
y: 0.8135 - val_loss: 0.4756 - val_accuracy: 0.8380
Epoch 6/10
```

```
704/704 [==============================] - 105s 149ms/step - loss: 0.5003 - accurac
y: 0.8285 - val_loss: 0.4470 - val_accuracy: 0.8462
Epoch 8/10
704/704 [==============================] - 105s 149ms/step - loss: 0.4789 - accurac
y: 0.8331 - val_loss: 0.4308 - val_accuracy: 0.8495
Epoch 9/10
704/704 [==============================] - 104s 148ms/step - loss: 0.4611 - accurac
y: 0.8390 - val_loss: 0.4276 - val_accuracy: 0.8513
Epoch 10/10
704/704 [==============================] - 105s 149ms/step - loss: 0.4436 - accurac
y: 0.8428 - val_loss: 0.4179 - val_accuracy: 0.8547
625/625 [==============================] - 9s 14ms/step - loss: 0.4295 - accuracy:
0.8498
625/625 [==============================] - 9s 14ms/step
Learning rate: 0.0001
Units: 512
Accuracy: 0.8498
Precision: 0.7788123184105319
Recall: 0.6996241869554957
Runtime: 1066.905986070633

Epoch 1/10
 23/704 [..............................] - ETA: 1:43 - loss: 5.1569 - accuracy: 0.02
45
```

```
IOPub message rate exceeded.
The notebook server will temporarily stop sending output
to the client in order to avoid crashing it.
To change this limit, set the config variable
`--NotebookApp.iopub_msg_rate_limit`.

Current values:
NotebookApp.iopub_msg_rate_limit=1000.0 (msgs/sec)
NotebookApp.rate_limit_window=3.0 (secs)
```

```
704/704 [==============================] - 108s 153ms/step - loss: 1.1737 - accurac
y: 0.6679 - val_loss: 0.8864 - val_accuracy: 0.7388
Epoch 6/10
704/704 [==============================] - 108s 153ms/step - loss: 1.0775 - accurac
y: 0.6886 - val_loss: 0.8173 - val_accuracy: 0.7539
Epoch 7/10
704/704 [==============================] - 108s 153ms/step - loss: 1.0016 - accurac
y: 0.7065 - val_loss: 0.7691 - val_accuracy: 0.7651
Epoch 8/10
704/704 [==============================] - 108s 154ms/step - loss: 0.9438 - accurac
y: 0.7199 - val_loss: 0.7270 - val_accuracy: 0.7756
Epoch 9/10
704/704 [==============================] - 110s 156ms/step - loss: 0.8899 - accurac
y: 0.7318 - val_loss: 0.6949 - val_accuracy: 0.7844
Epoch 10/10
704/704 [==============================] - 108s 153ms/step - loss: 0.8533 - accurac
y: 0.7414 - val_loss: 0.6677 - val_accuracy: 0.7923
625/625 [==============================] - 9s 15ms/step - loss: 0.6708 - accuracy:
0.7898
625/625 [==============================] - 9s 14ms/step
Learning rate: 1e-05
Units: 512
Accuracy: 0.78985
Precision: 0.7265378257011776
Recall: 0.5978188654641512
Runtime: 1097.3781945705414

Learning rate: 0.01
Units: 128
Accuracy: 0.8477
Precision: 0.8043123489373756
Recall: 0.6988400193387927
Runtime: 919.8214240074158

Learning rate: 0.001
Units: 128
Accuracy: 0.85255
Precision: 0.7867328400171182
Recall: 0.707327582008939
Runtime: 915.5359442234039

Learning rate: 0.0001
Units: 128
Accuracy: 0.843
Precision: 0.786125959157098
Recall: 0.6869673416696267
Runtime: 901.746609210968

Learning rate: 1e-05
Units: 128
Accuracy: 0.73465
Precision: 0.7162887549989162
Recall: 0.47558125347175484
Runtime: 885.5641613006592

Learning rate: 0.01
Units: 256
Accuracy: 0.85055
Precision: 0.8134135377605697
Recall: 0.7079299811099503
Runtime: 990.635219335556

Learning rate: 0.001
Units: 256
```

Accuracy: 0.85725
Precision: 0.7893143274662201
Recall: 0.7183029748170656
Runtime: 909.7056584358215

Learning rate: 0.0001
Units: 256
Accuracy: 0.8464
Precision: 0.7865026115391711
Recall: 0.6944006943451094
Runtime: 911.2518379688263

Learning rate: 1e-05
Units: 256
Accuracy: 0.76695
Precision: 0.6844942561186504
Recall: 0.5549426168871804
Runtime: 991.9178223609924

Learning rate: 0.01
Units: 384
Accuracy: 0.84665
Precision: 0.7905247884057616
Recall: 0.7130944499335667
Runtime: 991.4726612567902

Learning rate: 0.001
Units: 384
Accuracy: 0.8541
Precision: 0.7530003563189905
Recall: 0.7165125146157162
Runtime: 984.3839774131775

Learning rate: 0.0001
Units: 384
Accuracy: 0.8433
Precision: 0.7891048209621495
Recall: 0.7014753587907294
Runtime: 1054.4654290676117

Learning rate: 1e-05
Units: 384
Accuracy: 0.77725
Precision: 0.690018898958171
Recall: 0.5786048612733493
Runtime: 1114.2524206638336

Learning rate: 0.01
Units: 512
Accuracy: 0.84525
Precision: 0.7942784310891914
Recall: 0.7128416009952518
Runtime: 1098.0348672866821

Learning rate: 0.001
Units: 512
Accuracy: 0.85215
Precision: 0.7637378673224909
Recall: 0.7207188433737077
Runtime: 1113.2304723262787

Learning rate: 0.0001
Units: 512
Accuracy: 0.8498

Precision: 0.7788123184105319
Recall: 0.6996241869554957
Runtime: 1066.905986070633

Learning rate: 1e-05
Units: 512
Accuracy: 0.78985
Precision: 0.7265378257011776
Recall: 0.5978188654641512
Runtime: 1097.3781945705414

Graph process

In [1]:

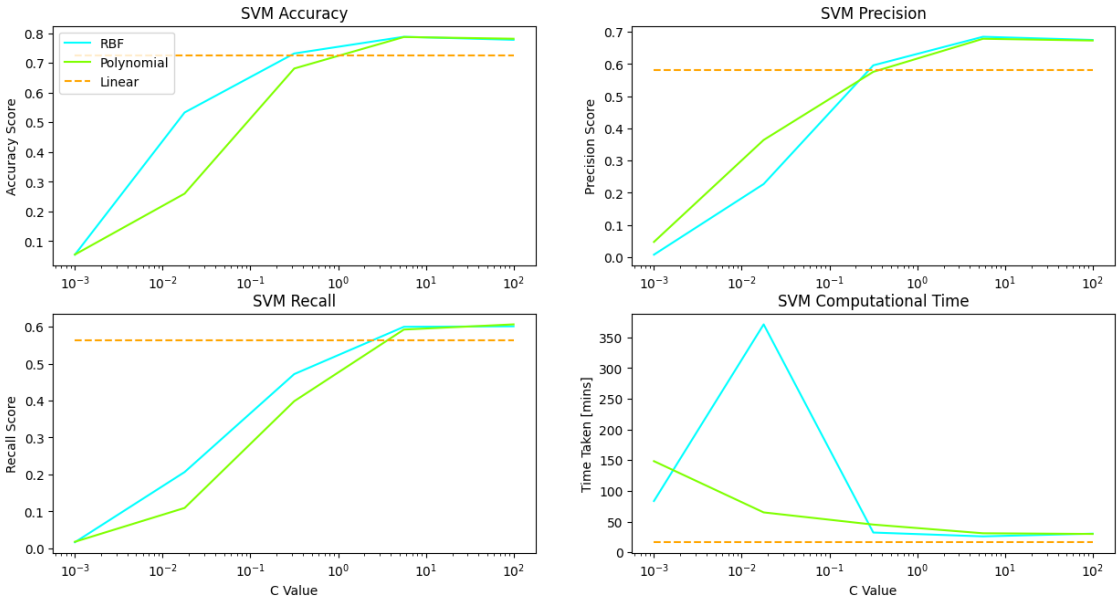
```
# Import packages
import numpy as np
import matplotlib.pyplot as plt
```

In [2]:

```
## SVM RESULTS

# Linear
l_c   = [0.0010, 100.00]

l_acc = [0.7264, 0.7264]
l_pre = [0.5819, 0.5819]
l_rec = [0.5627, 0.5627]
l_tim = [16.308, 16.308] # in mins

# non-linear
c     = [0.0010, 0.0178, 0.3162, 5.6234, 100.00]

# rbf
r_acc = [0.0547, 0.5330, 0.7315, 0.7877, 0.7774]
r_pre = [0.0078, 0.2271, 0.5956, 0.6843, 0.6745]
r_rec = [0.0162, 0.2063, 0.4716, 0.5996, 0.6010]
r_tim = [83.476, 371.14, 32.134, 25.899, 30.315] # in mins

# poly
p_acc = [0.0551, 0.2600, 0.6809, 0.7869, 0.7811]
p_pre = [0.0474, 0.3639, 0.5755, 0.6781, 0.6726]
p_rec = [0.0175, 0.1091, 0.3982, 0.5922, 0.6062]
p_tim = [148.38, 64.932, 45.097, 30.960, 29.848] # in mins
```

In [3]:

```
## SVM GRAPHS

# Produce Line Plots

fig, axs = plt.subplots(2, 2, figsize=(15, 7.5))

# ACCURACY
axs[0, 0].plot(c, r_acc, color='cyan')
axs[0, 0].plot(c, p_acc, color='chartreuse')
axs[0, 0].plot(l_c, l_acc, color='orange', linestyle='dashed')
axs[0, 0].legend(('RBF','Polynomial','Linear'),loc='upper left')
axs[0, 0].set_title('SVM Accuracy')
#axs[0, 0].set_xlabel('C Value')
axs[0, 0].set_ylabel('Accuracy Score')
axs[0, 0].set_xscale('log')

# PRECISION
axs[0, 1].plot(c, r_pre, color='cyan')
axs[0, 1].plot(c, p_pre, color='chartreuse')
axs[0, 1].plot(l_c, l_pre, color='orange', linestyle='dashed')
#axs[0, 1].legend(('RBF','Polynomial','Linear'),loc='upper left')
axs[0, 1].set_title('SVM Precision')
#axs[0, 1].set_xlabel('C Value')
axs[0, 1].set_ylabel('Precision Score')
axs[0, 1].set_xscale('log')

# RECALL
axs[1, 0].plot(c, r_rec, color='cyan')
axs[1, 0].plot(c, p_rec, color='chartreuse')
axs[1, 0].plot(l_c, l_rec, color='orange', linestyle='dashed')
#axs[1, 0].legend(('RBF','Polynomial','Linear'),loc='upper left')
axs[1, 0].set_title('SVM Recall')
axs[1, 0].set_xlabel('C Value')
axs[1, 0].set_ylabel('Recall Score')
axs[1, 0].set_xscale('log')

# COMPUTATIONAL TIME
axs[1, 1].plot(c, r_tim, color='cyan')
axs[1, 1].plot(c, p_tim, color='chartreuse')
axs[1, 1].plot(l_c, l_tim, color='orange', linestyle='dashed')
#axs[1, 1].legend(('RBF','Polynomial','Linear'),loc='upper left')
axs[1, 1].set_title('SVM Computational Time')
axs[1, 1].set_xlabel('C Value')
axs[1, 1].set_ylabel('Time Taken [mins]')
axs[1, 1].set_xscale('log')

#plt.savefig('svm_experiment_singlelegend.pdf', bbox_inches='tight', pad_inches=0.3, dpi = 500)
```



In [4]:

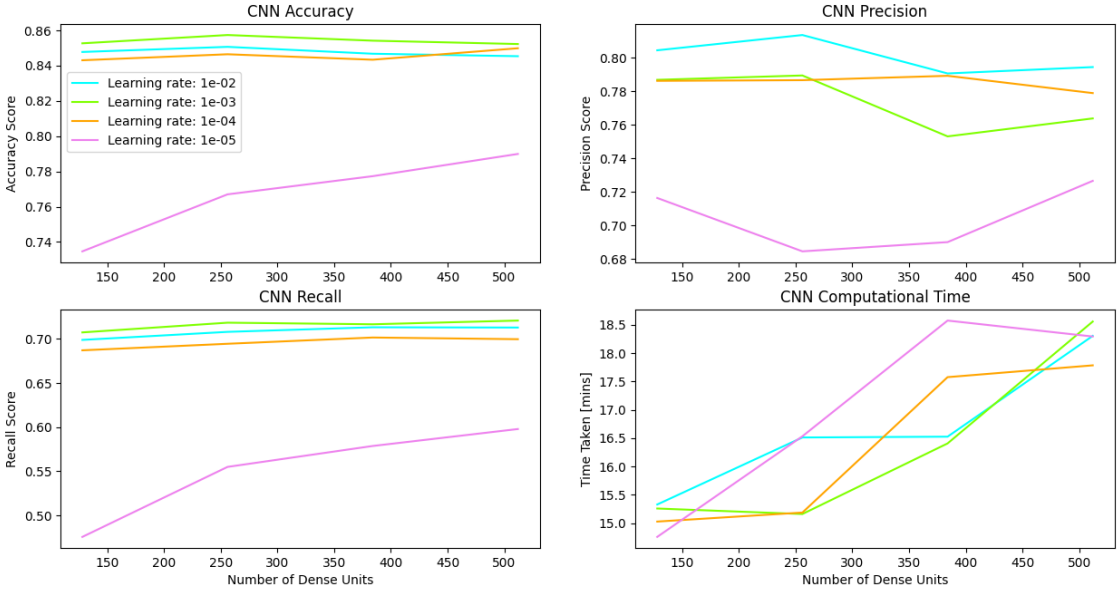
```
## CNN RESULTS

units = [128.00, 256.00, 384.00, 512.00]

# LR 1e-2
e2acc = [0.8477, 0.8506, 0.8467, 0.8453]
e2pre = [0.8043, 0.8134, 0.7905, 0.7943]
e2rec = [0.6988, 0.7079, 0.7131, 0.7128]
e2tim = [15.330, 16.511, 16.525, 18.301] # in mins

# LR 1e-3
e3acc = [0.8526, 0.8573, 0.8541, 0.8522]
e3pre = [0.7867, 0.7893, 0.7530, 0.7637]
e3rec = [0.7073, 0.7183, 0.7165, 0.7207]
e3tim = [15.259, 15.162, 16.406, 18.554] # in mins

# LR 1e-4
e4acc = [0.8430, 0.8464, 0.8433, 0.8498]
e4pre = [0.7861, 0.7865, 0.7891, 0.7788]
e4rec = [0.6870, 0.6944, 0.7015, 0.6996]
e4tim = [15.029, 15.188, 17.574, 17.782] # in mins

# LR 1e-5
e5acc = [0.7347, 0.7670, 0.7773, 0.7899]
e5pre = [0.7163, 0.6845, 0.6900, 0.7265]
e5rec = [0.4756, 0.5549, 0.5786, 0.5978]
e5tim = [14.759, 16.532, 18.571, 18.290] # in mins
```

In [5]:

```
## CNN GRAPHS

# Produce Line Plots

fig, axs = plt.subplots(2, 2, figsize=(15, 7.5))

# ACCURACY
axs[0, 0].plot(units, e2acc, color='cyan')
axs[0, 0].plot(units, e3acc, color='chartreuse')
axs[0, 0].plot(units, e4acc, color='orange')
axs[0, 0].plot(units, e5acc, color='violet')
axs[0, 0].legend(('Learning rate: 1e-02','Learning rate: 1e-03','Learning rate: 1e-04','Learning
axs[0, 0].set_title('CNN Accuracy')
#axs[0, 0].set_xlabel('Number of Dense Units')
axs[0, 0].set_ylabel('Accuracy Score')
#axs[0, 0].set_xscale('log')

# PRECISION
axs[0, 1].plot(units, e2pre, color='cyan')
axs[0, 1].plot(units, e3pre, color='chartreuse')
axs[0, 1].plot(units, e4pre, color='orange')
axs[0, 1].plot(units, e5pre, color='violet')
#axs[0, 1].legend(('Learning rate: 1e-02','Learning rate: 1e-03','Learning rate: 1e-04','Learnir
axs[0, 1].set_title('CNN Precision')
#axs[0, 1].set_xlabel('Number of Dense Units')
axs[0, 1].set_ylabel('Precision Score')
#axs[0, 1].set_xscale('log')

# RECALL
axs[1, 0].plot(units, e2rec, color='cyan')
axs[1, 0].plot(units, e3rec, color='chartreuse')
axs[1, 0].plot(units, e4rec, color='orange')
axs[1, 0].plot(units, e5rec, color='violet')
#axs[1, 0].legend(('Learning rate: 1e-02','Learning rate: 1e-03','Learning rate: 1e-04','Learnir
axs[1, 0].set_title('CNN Recall')
axs[1, 0].set_xlabel('Number of Dense Units')
axs[1, 0].set_ylabel('Recall Score')
#axs[1, 0].set_xscale('log')

# COMPUTATIONAL TIME
axs[1, 1].plot(units, e2tim, color='cyan')
axs[1, 1].plot(units, e3tim, color='chartreuse')
axs[1, 1].plot(units, e4tim, color='orange')
axs[1, 1].plot(units, e5tim, color='violet')
#axs[1, 1].legend(('Learning rate: 1e-02','Learning rate: 1e-03','Learning rate: 1e-04','Learnir
axs[1, 1].set_title('CNN Computational Time')
axs[1, 1].set_xlabel('Number of Dense Units')
axs[1, 1].set_ylabel('Time Taken [mins]')
#axs[1, 0].set_xscale('log')

#plt.savefig('cnn_experiment_singlelegend.pdf', bbox_inches='tight', pad_inches=0.3, dpi = 500)
```

Out[5]:

```
Text(0, 0.5, 'Time Taken [mins]')
```



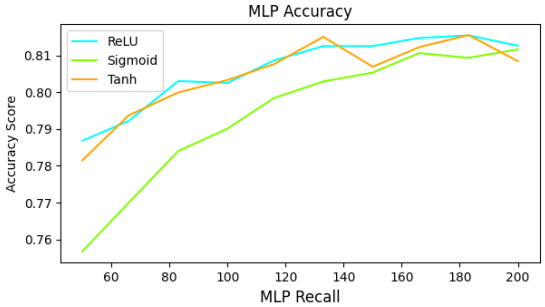

```
## MLP RESULTS

n_neu = [50.000, 66.000, 83.000, 100.00, 116.00, 133.00, 150.00, 166.00, 183.00, 200.00]

# RELU
r_acc = [0.7868, 0.7922, 0.8030, 0.8025, 0.8086, 0.8125, 0.8125, 0.8147, 0.8154, 0.8126]
r_pre = [0.6636, 0.6652, 0.6812, 0.6863, 0.6939, 0.6982, 0.7013, 0.6930, 0.6940, 0.7021]
r_rec = [0.6185, 0.6344, 0.6429, 0.6486, 0.6528, 0.6641, 0.6544, 0.6728, 0.6648, 0.6722]
r_tim = [0.2138, 0.2233, 0.2327, 0.2923, 0.3003, 0.4704, 0.5358, 0.4253, 0.4250, 0.4445] # in mi

# SIGMOID
s_acc = [0.7568, 0.7700, 0.7840, 0.7901, 0.7984, 0.8029, 0.8053, 0.8106, 0.8093, 0.8116]
s_pre = [0.6096, 0.6250, 0.6600, 0.6758, 0.6876, 0.6874, 0.6887, 0.7030, 0.6879, 0.6993]
s_rec = [0.5539, 0.5751, 0.5995, 0.6182, 0.6297, 0.6400, 0.6351, 0.6487, 0.6430, 0.6541]
s_tim = [0.2808, 0.2752, 0.2767, 0.3148, 0.3440, 0.3292, 0.3756, 0.3317, 0.3293, 0.5226] # in mi

# TANH
t_acc = [0.7815, 0.7937, 0.7999, 0.8033, 0.8076, 0.8150, 0.8069, 0.8122, 0.8155, 0.8084]
t_pre = [0.6459, 0.6639, 0.6921, 0.6831, 0.6873, 0.7001, 0.6904, 0.6900, 0.7004, 0.7131]
t_rec = [0.6050, 0.6251, 0.6362, 0.6559, 0.6567, 0.6618, 0.6547, 0.6677, 0.6664, 0.6643]
t_tim = [0.2080, 0.2269, 0.2690, 0.2566, 0.2838, 0.2688, 0.2873, 0.2582, 0.4052, 0.4425] # in mi
```

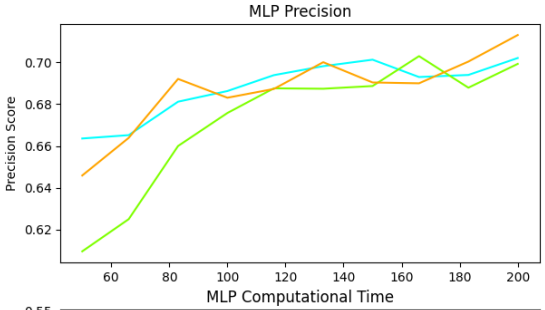
In [7]:

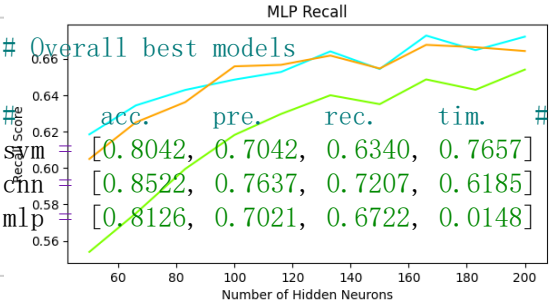
```
## MLP GRAPHS

# Produce Line Plots

fig, axs = plt.subplots(2, 2, figsize=(15, 7.5))

# ACCURACY
axs[0, 0].plot(n_neu, r_acc, color='cyan')
axs[0, 0].plot(n_neu, s_acc, color='chartreuse')
axs[0, 0].plot(n_neu, t_acc, color='orange')
axs[0, 0].legend(('ReLU','Sigmoid','Tanh'),loc='upper left')
axs[0, 0].set_title('MLP Accuracy')
#axs[0, 0].set_xlabel('Number of Hidden Neurons')
axs[0, 0].set_ylabel('Accuracy Score')

# PRECISION
axs[0, 1].plot(n_neu, r_pre, color='cyan')
axs[0, 1].plot(n_neu, s_pre, color='chartreuse')
axs[0, 1].plot(n_neu, t_pre, color='orange')
#axs[0, 1].legend(('ReLU','Sigmoid','Tanh'),loc='upper left')
axs[0, 1].set_title('MLP Precision')
#axs[0, 1].set_xlabel('Number of Hidden Neurons')
axs[0, 1].set_ylabel('Precision Score')

# RECALL
axs[1, 0].plot(n_neu, r_rec, color='cyan')
axs[1, 0].plot(n_neu, s_rec, color='chartreuse')
axs[1, 0].plot(n_neu, t_rec, color='orange')
#axs[1, 0].legend(('ReLU','Sigmoid','Tanh'),loc='upper left')
axs[1, 0].set_title('MLP Recall')
axs[1, 0].set_xlabel('Number of Hidden Neurons')
axs[1, 0].set_ylabel('Recall Score')

# COMPUTATIONAL TIME
axs[1, 1].plot(n_neu, r_tim, color='cyan')
axs[1, 1].plot(n_neu, s_tim, color='chartreuse')
axs[1, 1].plot(n_neu, t_tim, color='orange')
#axs[1, 1].legend(('ReLU','Sigmoid','Tanh'),loc='upper left')
axs[1, 1].set_title('MLP Computational Time')
axs[1, 1].set_xlabel('Number of Hidden Neurons')
axs[1, 1].set_ylabel('Time Taken [mins]')

#plt.savefig('mlp_experiment_singlelegend.pdf', bbox_inches='tight', pad_inches=0.3, dpi = 500)
```

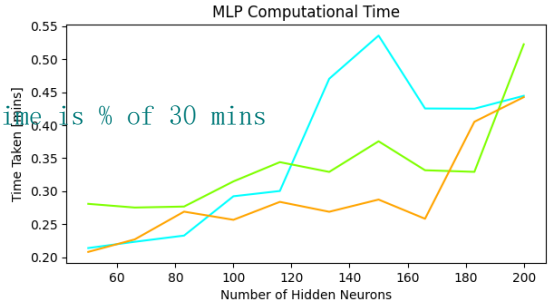
Out[7]:

```
Text(0, 0.5, 'Time Taken [mins]')
```










```
# Overall best models
#          acc.      pre.      rec.      tim.    # time is % of 30 mins
svm = [0.8042, 0.7042, 0.6340, 0.7657]
cnn = [0.8522, 0.7637, 0.7207, 0.6185]
mlp = [0.8126, 0.7021, 0.6722, 0.0148]
```

In [9]:

```
measures = ("Accuracy Score", "Precision Score", "Recall Score", "Time Taken [30mins]")
scores = {
    'Support Vector Machine': (0.804, 0.704, 0.634, 0.766),
    'Convolutional Neura': (0.852, 0.764, 0.721, 0.619),
    'MLP': (0.813, 0.702, 0.672, 0.015),
}

x = np.arange(len(measures))  # the label locations
width = 0.18  # the width of the bars
multiplier = 0
colors = ['cyan','chartreuse','orange']

fig, ax = plt.subplots(figsize=(12, 6))

for attribute, measurement in scores.items():
    offset = width * multiplier
    rects = ax.bar(x + offset, measurement, width, label=attribute, color=colors[multiplier])
    ax.bar_label(rects, padding=3)
    multiplier += 1

# Add some text for labels, title and custom x-axis tick labels, etc.
ax.set_title('Comparison of Evaluation Metrics for All Models')
ax.set_xticks(x + width, measures)
ax.legend(loc='upper left', ncols=3)
ax.set_ylim(0, 1)


#plt.savefig('best_models.pdf', bbox_inches='tight', pad_inches=0.3, dpi = 500) #save plot as pd
```

Out[9]:

(0.0, 1.0)