DEEP NEURAL NETWORKS PROJECT

Rama Alyoubi 2110112
Rimas Alshehri 2110240

PROJECT INTRODUCTION

Objective : Develop a neural network tailored for the precise recognition of individual handwritten Arabic characters.

Challenge : Our task is to engineer and fine-tune a deep neural network architecture that excels in accuracy when confronted with the test set.

Journey :

- From problem introduction to neural network intricacies.
- Through dataset details to results and conclusions.

DATA OVERVIEW

عضصنملكططزرذدخحجثتبا
،ىلاوهءسشنفغ

ع ض ص ن م ل ك ط ط ز ر ذ د خ ح ج ث ت ب ا

01 - DATA DESCRIPTION

The Arabic Chars MNIST dataset is a collection of 28x28 grayscale images of handwritten Arabic characters, each labeled for classification.

02 - NATURE OF THE DATA

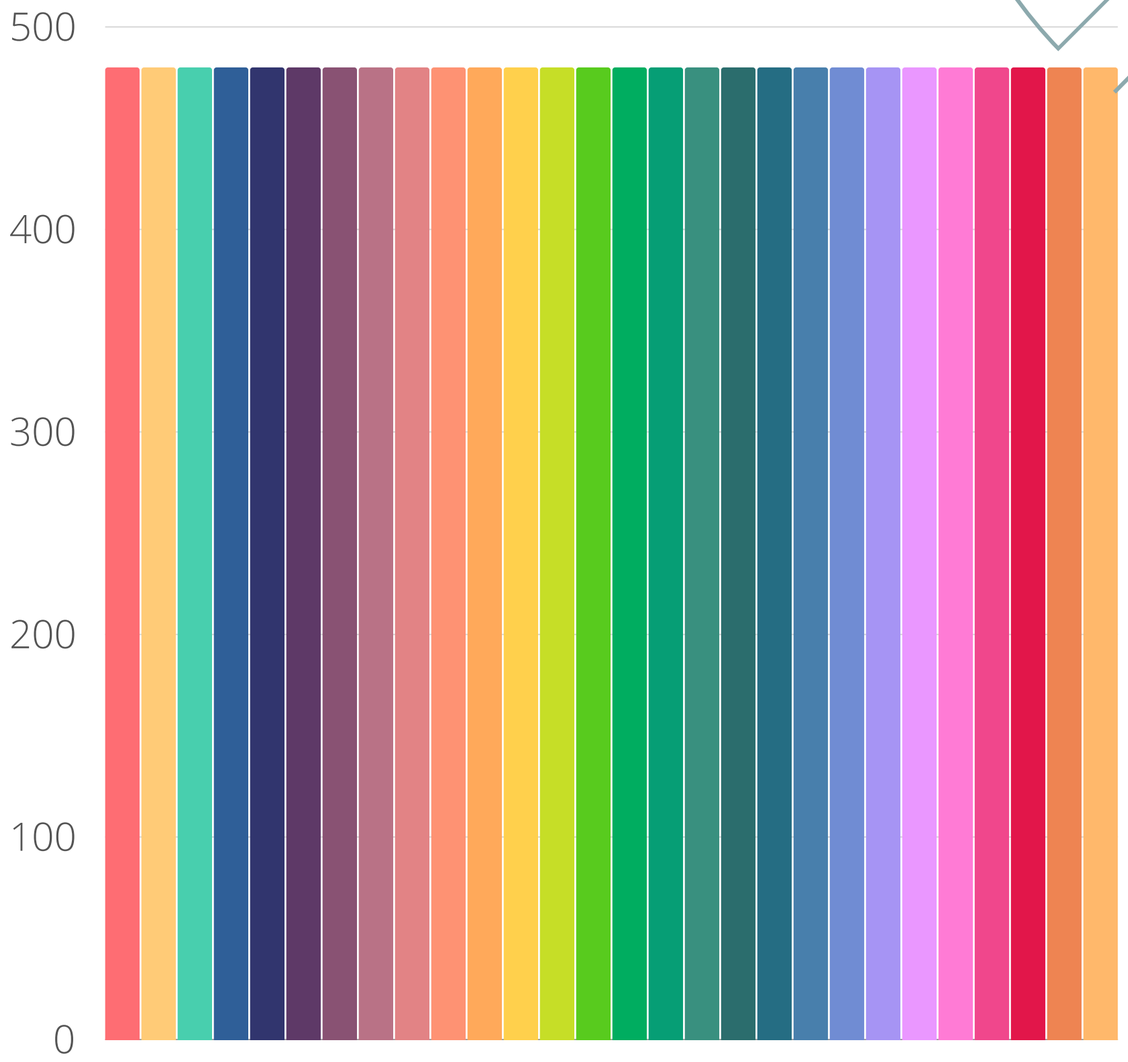
Grayscale images in a standardized size, featuring a range of handwriting styles, suitable for training OCR algorithms.

03 - SIZE OF THE DATA

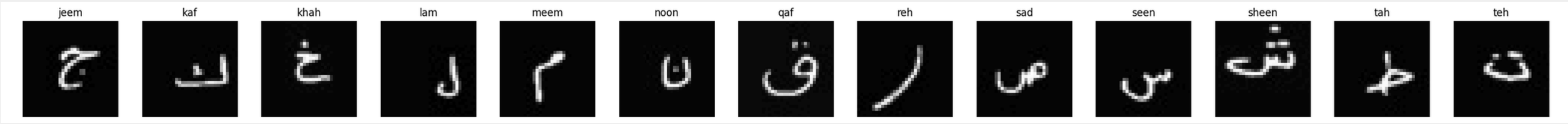
Comprising 16,800 images in total, with 13,440 for training and 3,360 for testing across 28 classes.

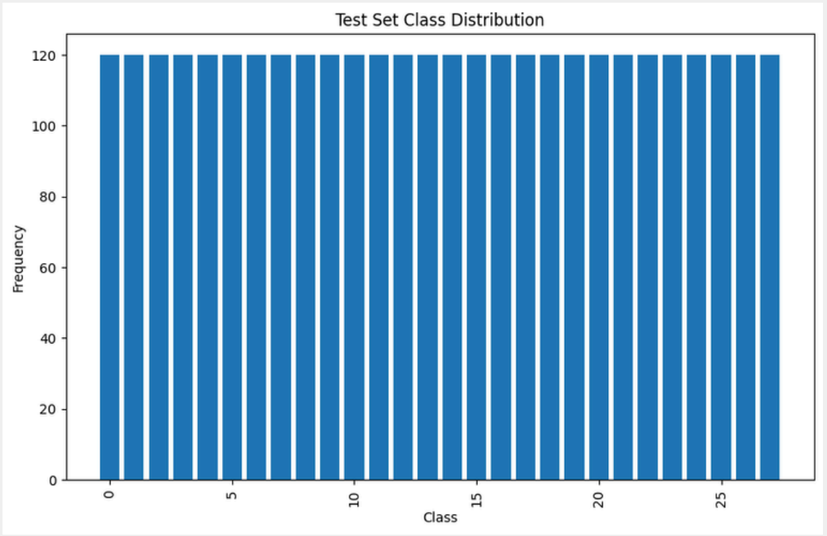
EXPLORATORY DATA ANALYSIS (EDA)

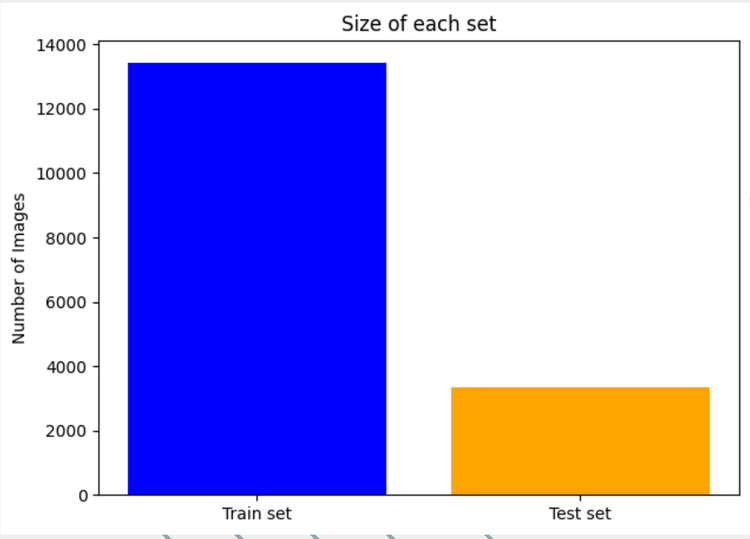
In our comprehensive EDA, we've confirmed that the dataset is perfectly balanced, with each of the 28 Arabic character classes represented by 480 images. This uniform distribution is ideal as it prevents model bias towards more frequently represented classes and ensures equal learning opportunity for all characters.



DATA SPLITTING & VIZUALIZATION

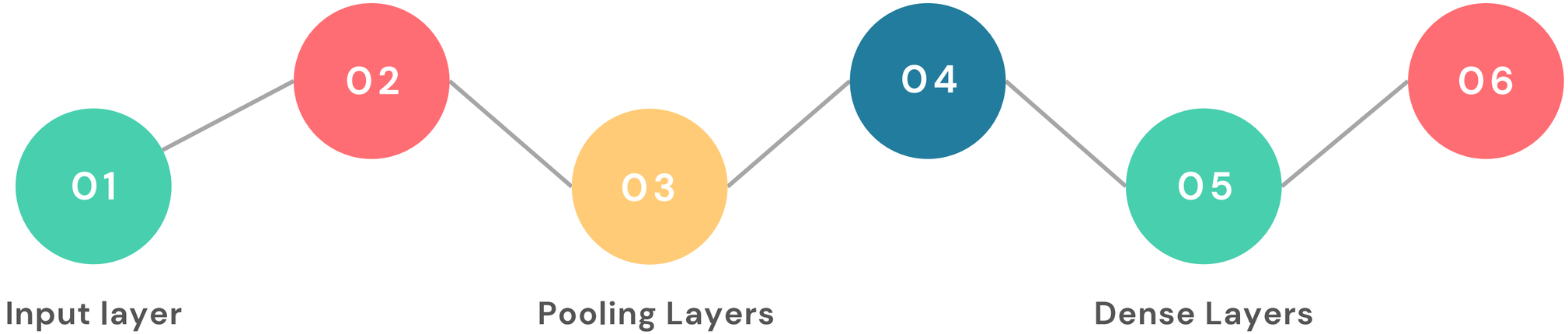






PROCESS OF BUILDING THE CNN MODEL

BASIC STRUCTURE



Input layer

Convolutional Layers
for feature exteraction

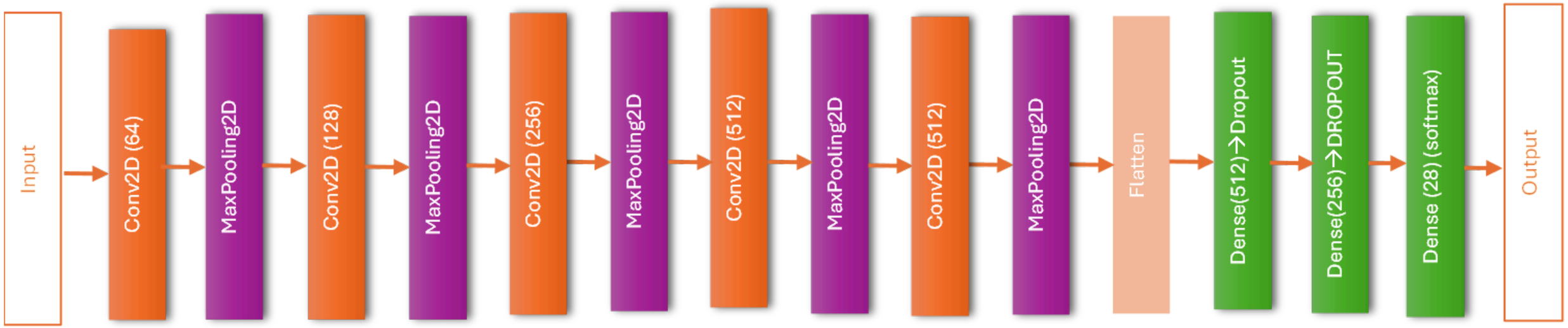
Pooling Layers

Flattening Layer

Dense Layers

Compilation

MODEL ARCHITECTURE



IMPLEMENTATION

```python
def create_advanced_cnn_model(input_shape, num_classes):
    model = Sequential([
        # First conv layer
        Conv2D(64, kernel_size=(3, 3), strides=(1, 1), activation='relu', input_shape=input_shape, padding='same', kernel_regularizer=l2(0.001)),
        MaxPooling2D(pool_size=(2, 2)),

        # Second conv layer
        Conv2D(128, kernel_size=(3, 3), strides=(1, 1), activation='relu', padding='same', kernel_regularizer=l2(0.001)),
        MaxPooling2D(pool_size=(2, 2)),

        # Third conv layer
        Conv2D(256, kernel_size=(3, 3), strides=(1, 1), activation='relu', padding='same', kernel_regularizer=l2(0.001)),
        MaxPooling2D(pool_size=(2, 2)),

        # Fourth conv layer
        Conv2D(512, kernel_size=(3, 3), strides=(1, 1), activation='relu', padding='same', kernel_regularizer=l2(0.001)),
        MaxPooling2D(pool_size=(2, 2)),

        # Fifth conv layer
        Conv2D(512, kernel_size=(3, 3), strides=(1, 1), activation='relu', padding='same', kernel_regularizer=l2(0.001)),
        MaxPooling2D(pool_size=(2, 2)),

        # Flattening the layers
        Flatten(),

        # Dense layers
        Dense(512, activation='relu', kernel_regularizer=l2(0.001)),
        Dropout(0.5),
        Dense(256, activation='relu', kernel_regularizer=l2(0.001)),
        Dropout(0.5),
        Dense(num_classes, activation='softmax')
    ])

    model.compile(optimizer=Adam(), loss='categorical_crossentropy', metrics=['accuracy'])
    return model
```

MODEL SUMMARY

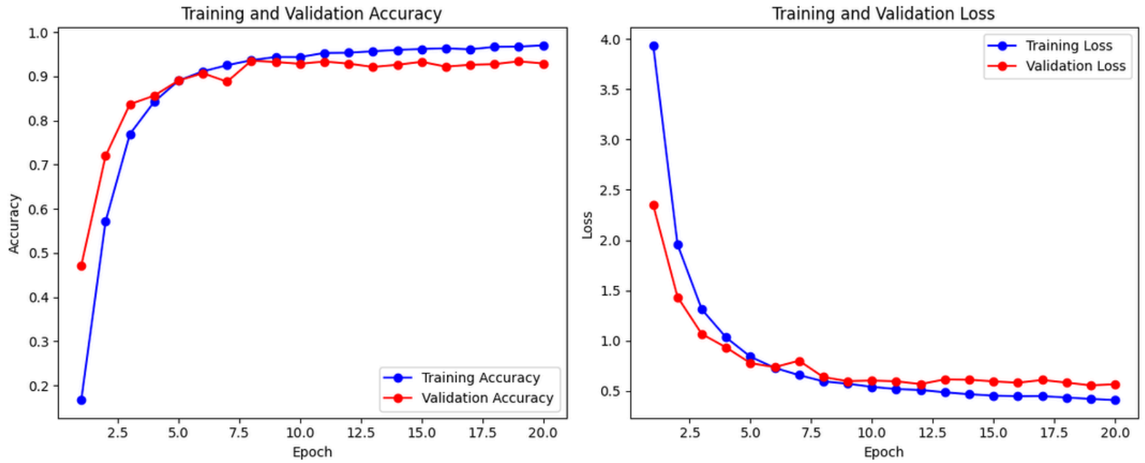
```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d (Conv2D)             (None, 32, 32, 64)        640

 max_pooling2d (MaxPooling2  (None, 16, 16, 64)        0
 D)

 conv2d_1 (Conv2D)           (None, 16, 16, 128)       73856

 max_pooling2d_1 (MaxPoolin  (None, 8, 8, 128)         0
 g2D)

 conv2d_2 (Conv2D)           (None, 8, 8, 256)         295168

 max_pooling2d_2 (MaxPoolin  (None, 4, 4, 256)         0
 g2D)

 conv2d_3 (Conv2D)           (None, 4, 4, 512)         1180160

 max_pooling2d_3 (MaxPoolin  (None, 2, 2, 512)         0
 g2D)

 conv2d_4 (Conv2D)           (None, 2, 2, 512)         2359808

 max_pooling2d_4 (MaxPoolin  (None, 1, 1, 512)         0
 g2D)

 flatten (Flatten)           (None, 512)               0

 dense (Dense)               (None, 512)               262656

 dropout (Dropout)           (None, 512)               0

 dense_1 (Dense)             (None, 256)               131328

 dropout_1 (Dropout)         (None, 256)               0

 dense_2 (Dense)             (None, 28)                7196

=================================================================
Total params: 4310812 (16.44 MB)
Trainable params: 4310812 (16.44 MB)
Non-trainable params: 0 (0.00 Byte)
```

TRAINING VS VALIDATION

```
history = model.fit(
    train_images, train_labels_one_hot,
    epochs=20,
    batch_size=32,
    validation_split=0.2,  # Using 20% of the data for validation
    verbose=1
)
```

```
Epoch 1/20
336/336 [==============================] - 12s 11ms/step - loss: 3.9375 - accuracy: 0.1667 - val_loss: 2.3484 - val_accuracy: 0.4714
Epoch 2/20
336/336 [==============================] - 4s 11ms/step - loss: 1.9556 - accuracy: 0.5714 - val_loss: 1.4337 - val_accuracy: 0.7199
Epoch 3/20
336/336 [==============================] - 4s 12ms/step - loss: 1.3098 - accuracy: 0.7693 - val_loss: 1.0653 - val_accuracy: 0.8371
Epoch 4/20
336/336 [==============================] - 4s 12ms/step - loss: 1.0307 - accuracy: 0.8424 - val_loss: 0.9318 - val_accuracy: 0.8564
Epoch 5/20
336/336 [==============================] - 4s 11ms/step - loss: 0.8395 - accuracy: 0.8905 - val_loss: 0.7761 - val_accuracy: 0.8906
Epoch 6/20
336/336 [==============================] - 3s 10ms/step - loss: 0.7288 - accuracy: 0.9114 - val_loss: 0.7363 - val_accuracy: 0.9074
Epoch 7/20
336/336 [==============================] - 4s 12ms/step - loss: 0.6567 - accuracy: 0.9254 - val_loss: 0.7999 - val_accuracy: 0.8880
Epoch 8/20
336/336 [==============================] - 4s 12ms/step - loss: 0.5949 - accuracy: 0.9367 - val_loss: 0.6371 - val_accuracy: 0.9356
Epoch 9/20
336/336 [==============================] - 4s 11ms/step - loss: 0.5715 - accuracy: 0.9441 - val_loss: 0.5971 - val_accuracy: 0.9327
Epoch 10/20
336/336 [==============================] - 4s 11ms/step - loss: 0.5399 - accuracy: 0.9439 - val_loss: 0.6029 - val_accuracy: 0.9286
Epoch 11/20
336/336 [==============================] - 4s 12ms/step - loss: 0.5180 - accuracy: 0.9531 - val_loss: 0.5952 - val_accuracy: 0.9338
Epoch 12/20
336/336 [==============================] - 4s 12ms/step - loss: 0.5090 - accuracy: 0.9537 - val_loss: 0.5677 - val_accuracy: 0.9289
Epoch 13/20
336/336 [==============================] - 4s 10ms/step - loss: 0.4858 - accuracy: 0.9569 - val_loss: 0.6146 - val_accuracy: 0.9215
Epoch 14/20
336/336 [==============================] - 4s 11ms/step - loss: 0.4669 - accuracy: 0.9600 - val_loss: 0.6110 - val_accuracy: 0.9263
Epoch 15/20
336/336 [==============================] - 4s 11ms/step - loss: 0.4530 - accuracy: 0.9622 - val_loss: 0.5946 - val_accuracy: 0.9330
Epoch 16/20
336/336 [==============================] - 4s 12ms/step - loss: 0.4464 - accuracy: 0.9636 - val_loss: 0.5811 - val_accuracy: 0.9222
Epoch 17/20
336/336 [==============================] - 4s 11ms/step - loss: 0.4480 - accuracy: 0.9614 - val_loss: 0.6084 - val_accuracy: 0.9263
Epoch 18/20
336/336 [==============================] - 4s 11ms/step - loss: 0.4340 - accuracy: 0.9670 - val_loss: 0.5824 - val_accuracy: 0.9278
Epoch 19/20
336/336 [==============================] - 4s 12ms/step - loss: 0.4200 - accuracy: 0.9674 - val_loss: 0.5550 - val_accuracy: 0.9342
Epoch 20/20
336/336 [==============================] - 4s 12ms/step - loss: 0.4081 - accuracy: 0.9705 - val_loss: 0.5666 - val_accuracy: 0.9293
```

training accuracy: 0.97
validation accuracy: 0.92

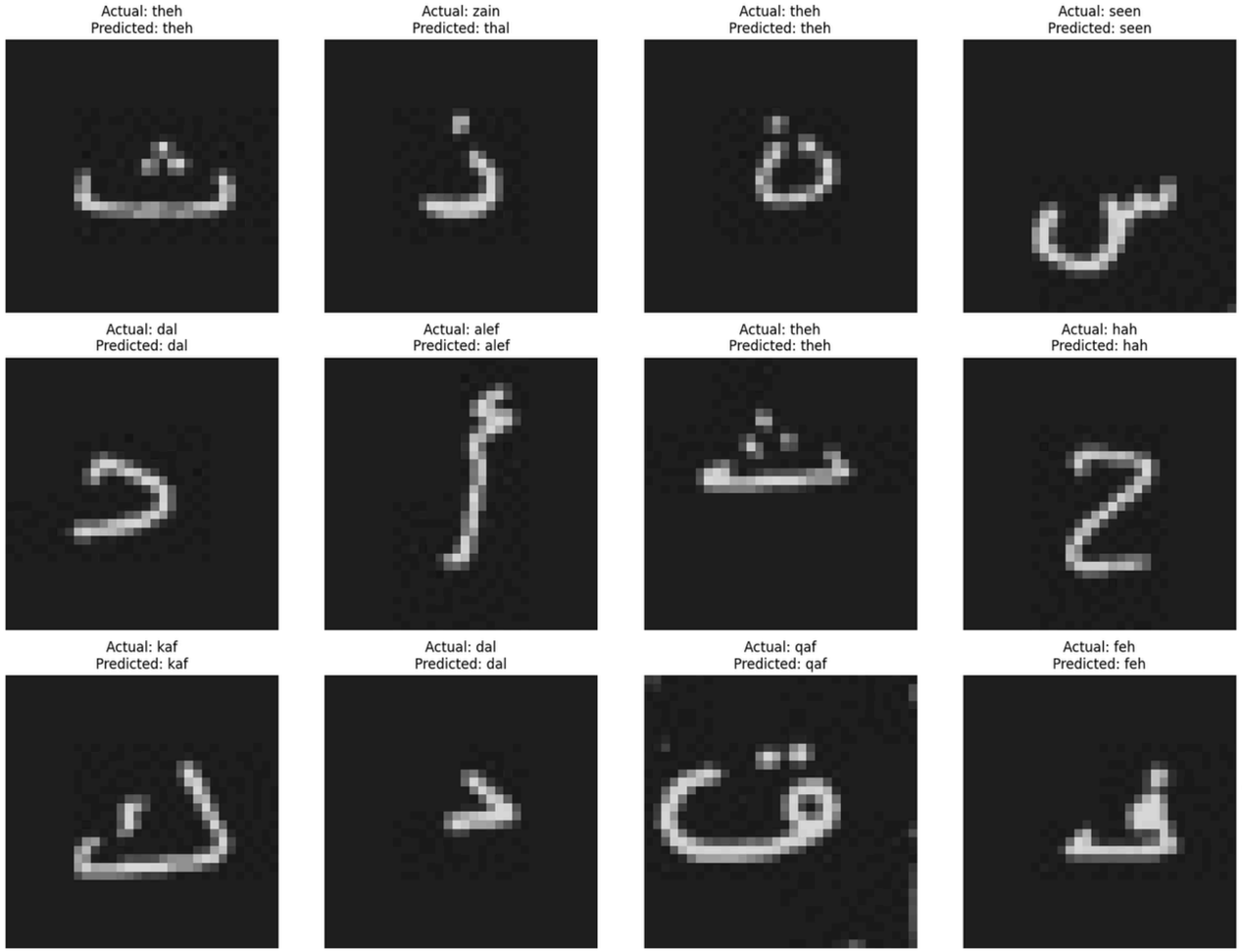


TESTING

```
# Evaluate the model on the test set
test_loss, test_accuracy = model.evaluate(test_images, test_labels_one_hot)
print(f"Test Loss: {test_loss}")
print(f"Test Accuracy: {test_accuracy}")
```

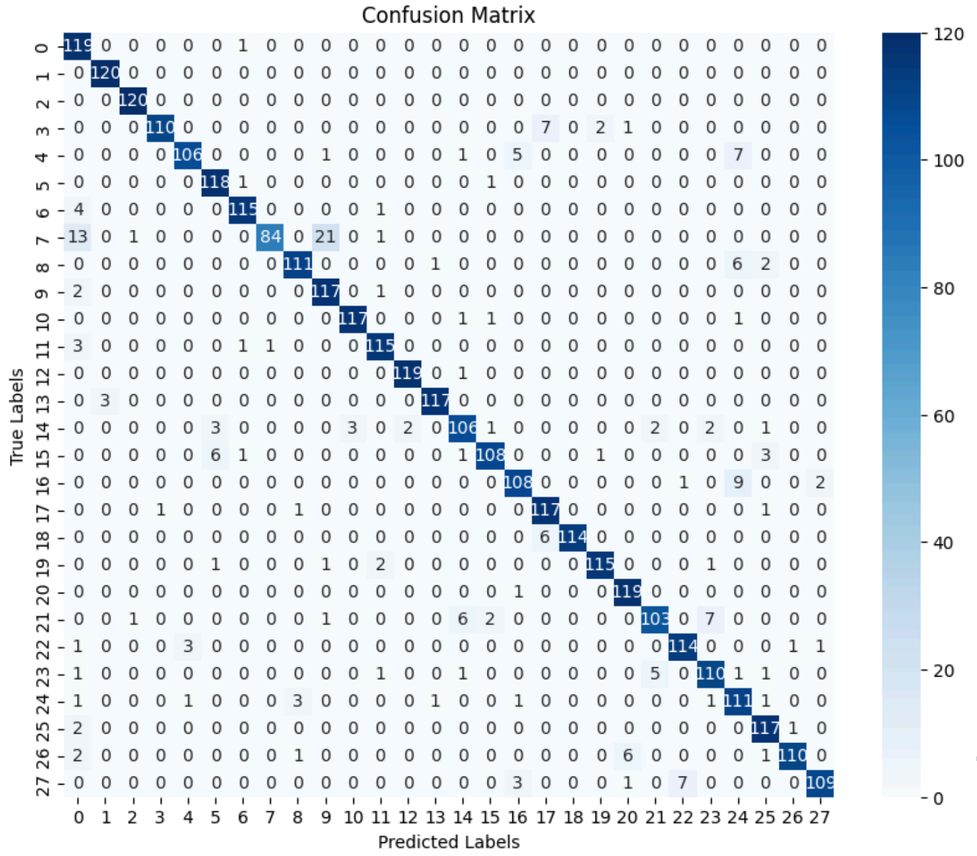
```
105/105 [==============================] - 0s 4ms/step - loss: 0.5559 - accuracy: 0.9372
Test Loss: 0.5559284090995789
Test Accuracy: 0.9372023940086365
```

testing accuracy: 0.93



CLASSIFICATION REPORT & CONFUSION MATRIX

```
105/105 [==============================] - 0s 3ms/step
              precision    recall  f1-score   support

           0       0.80      0.99      0.89       120
           1       0.98      1.00      0.99       120
           2       0.98      1.00      0.99       120
           3       0.99      0.92      0.95       120
           4       0.96      0.88      0.92       120
           5       0.92      0.98      0.95       120
           6       0.97      0.96      0.96       120
           7       0.99      0.70      0.82       120
           8       0.96      0.93      0.94       120
           9       0.83      0.97      0.90       120
          10       0.97      0.97      0.97       120
          11       0.95      0.96      0.95       120
          12       0.98      0.99      0.99       120
          13       0.98      0.97      0.98       120
          14       0.91      0.88      0.89       120
          15       0.96      0.90      0.93       120
          16       0.92      0.90      0.91       120
          17       0.90      0.97      0.94       120
          18       1.00      0.95      0.97       120
          19       0.97      0.96      0.97       120
          20       0.94      0.99      0.96       120
          21       0.94      0.86      0.90       120
          22       0.93      0.95      0.94       120
          23       0.91      0.92      0.91       120
          24       0.82      0.93      0.87       120
          25       0.92      0.97      0.95       120
          26       0.98      0.92      0.95       120
          27       0.97      0.91      0.94       120

    accuracy                           0.94      3360
   macro avg       0.94      0.94      0.94      3360
weighted avg       0.94      0.94      0.94      3360
```



DIFFERENT VARIATIONS OF PARAMETERS TO THE MODEL

CHANGE 1 TO THE MODEL

changing convolution size, kernel size, L2 regularization

IMPLEMENTATION

```python
def create_advanced_cnn_model(input_shape, num_classes):
    model = Sequential([
        # First conv layer
        Conv2D(64, kernel_size=(3, 3), strides=(1, 1), activation='relu', input_shape=input_shape, padding='same', kernel_regularizer=l2(0.002)),
        MaxPooling2D(pool_size=(2, 2)),

        # Second conv layer
        Conv2D(32, kernel_size=(6, 6), strides=(1, 1), activation='relu', padding='same', kernel_regularizer=l2(0.002)),
        MaxPooling2D(pool_size=(2, 2)),

        # Third conv layer
        Conv2D(256, kernel_size=(6, 6), strides=(1, 1), activation='relu', padding='same', kernel_regularizer=l2(0.002)),
        MaxPooling2D(pool_size=(2, 2)),

        # Fourth conv layer
        Conv2D(64, kernel_size=(3, 3), strides=(1, 1), activation='relu', padding='same', kernel_regularizer=l2(0.002)),
        MaxPooling2D(pool_size=(2, 2)),

        # Fifth conv layer
        Conv2D(32, kernel_size=(6, 6), strides=(1, 1), activation='relu', padding='same', kernel_regularizer=l2(0.002)),
        MaxPooling2D(pool_size=(2, 2)),

        # Flattening the layers
        Flatten(),

        # Dense layers
        Dense(512, activation='relu', kernel_regularizer=l2(0.002)),
        Dropout(0.5),
        Dense(256, activation='relu', kernel_regularizer=l2(0.002)),
        Dropout(0.5),
        Dense(num_classes, activation='softmax')
    ])

    model.compile(optimizer=Adam(), loss='categorical_crossentropy', metrics=['accuracy'])
    return model
```

MODEL SUMMARY

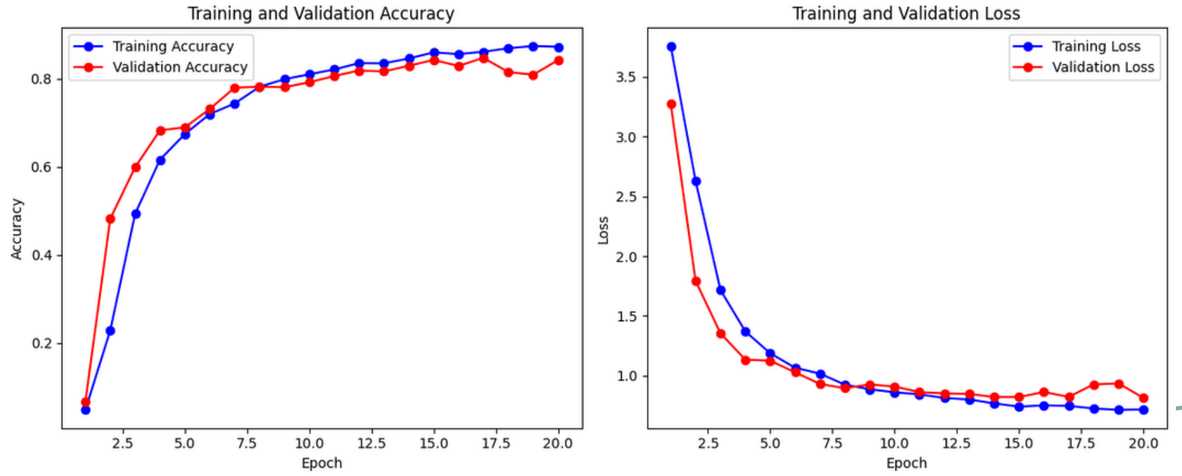
```
Model: "sequential_2"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d_10 (Conv2D)          (None, 32, 32, 64)        640

 max_pooling2d_10 (MaxPooli  (None, 16, 16, 64)        0
 ng2D)

 conv2d_11 (Conv2D)          (None, 16, 16, 32)        73760

 max_pooling2d_11 (MaxPooli  (None, 8, 8, 32)          0
 ng2D)

 conv2d_12 (Conv2D)          (None, 8, 8, 256)         295168

 max_pooling2d_12 (MaxPooli  (None, 4, 4, 256)         0
 ng2D)

 conv2d_13 (Conv2D)          (None, 4, 4, 64)          147520

 max_pooling2d_13 (MaxPooli  (None, 2, 2, 64)          0
 ng2D)

 conv2d_14 (Conv2D)          (None, 2, 2, 32)          73760

 max_pooling2d_14 (MaxPooli  (None, 1, 1, 32)          0
 ng2D)

 flatten_2 (Flatten)         (None, 32)                0

 dense_6 (Dense)             (None, 512)               16896

 dropout_4 (Dropout)         (None, 512)               0

 dense_7 (Dense)             (None, 256)               131328

 dropout_5 (Dropout)         (None, 256)               0

 dense_8 (Dense)             (None, 28)                7196

=================================================================
Total params: 746268 (2.85 MB)
Trainable params: 746268 (2.85 MB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
```

TRAINING VS VALIDATION

```python
history = model.fit(
    train_images, train_labels_one_hot,
    epochs=20,
    batch_size=32,
    validation_split=0.2,  # Using 20% of the data for validation
    verbose=1
)
```

```
Epoch 1/20
336/336 [==============================] - 6s 10ms/step - loss: 3.7567 - accuracy: 0.0484 - val_loss: 3.2730 - val_accuracy: 0.0666
Epoch 2/20
336/336 [==============================] - 3s 8ms/step - loss: 2.6273 - accuracy: 0.2286 - val_loss: 1.7924 - val_accuracy: 0.4821
Epoch 3/20
336/336 [==============================] - 2s 7ms/step - loss: 1.7165 - accuracy: 0.4936 - val_loss: 1.3547 - val_accuracy: 0.5993
Epoch 4/20
336/336 [==============================] - 2s 7ms/step - loss: 1.3704 - accuracy: 0.6164 - val_loss: 1.1346 - val_accuracy: 0.6830
Epoch 5/20
336/336 [==============================] - 2s 7ms/step - loss: 1.1876 - accuracy: 0.6746 - val_loss: 1.1258 - val_accuracy: 0.6897
Epoch 6/20
336/336 [==============================] - 2s 7ms/step - loss: 1.0667 - accuracy: 0.7199 - val_loss: 1.0285 - val_accuracy: 0.7314
Epoch 7/20
336/336 [==============================] - 3s 9ms/step - loss: 1.0178 - accuracy: 0.7439 - val_loss: 0.9313 - val_accuracy: 0.7798
Epoch 8/20
336/336 [==============================] - 3s 8ms/step - loss: 0.9253 - accuracy: 0.7818 - val_loss: 0.8957 - val_accuracy: 0.7820
Epoch 9/20
336/336 [==============================] - 2s 7ms/step - loss: 0.8870 - accuracy: 0.7991 - val_loss: 0.9272 - val_accuracy: 0.7809
Epoch 10/20
336/336 [==============================] - 2s 7ms/step - loss: 0.8614 - accuracy: 0.8102 - val_loss: 0.9096 - val_accuracy: 0.7920
Epoch 11/20
336/336 [==============================] - 2s 7ms/step - loss: 0.8450 - accuracy: 0.8211 - val_loss: 0.8630 - val_accuracy: 0.8062
Epoch 12/20
336/336 [==============================] - 2s 7ms/step - loss: 0.8140 - accuracy: 0.8355 - val_loss: 0.8528 - val_accuracy: 0.8188
Epoch 13/20
336/336 [==============================] - 3s 9ms/step - loss: 0.8011 - accuracy: 0.8350 - val_loss: 0.8477 - val_accuracy: 0.8166
Epoch 14/20
336/336 [==============================] - 3s 9ms/step - loss: 0.7694 - accuracy: 0.8461 - val_loss: 0.8225 - val_accuracy: 0.8296
Epoch 15/20
336/336 [==============================] - 2s 7ms/step - loss: 0.7417 - accuracy: 0.8599 - val_loss: 0.8220 - val_accuracy: 0.8426
Epoch 16/20
336/336 [==============================] - 2s 7ms/step - loss: 0.7524 - accuracy: 0.8560 - val_loss: 0.8640 - val_accuracy: 0.8292
Epoch 17/20
336/336 [==============================] - 2s 7ms/step - loss: 0.7479 - accuracy: 0.8614 - val_loss: 0.8234 - val_accuracy: 0.8475
Epoch 18/20
336/336 [==============================] - 2s 7ms/step - loss: 0.7263 - accuracy: 0.8692 - val_loss: 0.9274 - val_accuracy: 0.8155
Epoch 19/20
336/336 [==============================] - 3s 8ms/step - loss: 0.7149 - accuracy: 0.8745 - val_loss: 0.9363 - val_accuracy: 0.8092
Epoch 20/20
336/336 [==============================] - 3s 9ms/step - loss: 0.7179 - accuracy: 0.8727 - val_loss: 0.8153 - val_accuracy: 0.8423
```

training accuracy: 0.87
validation accuracy: 0.84

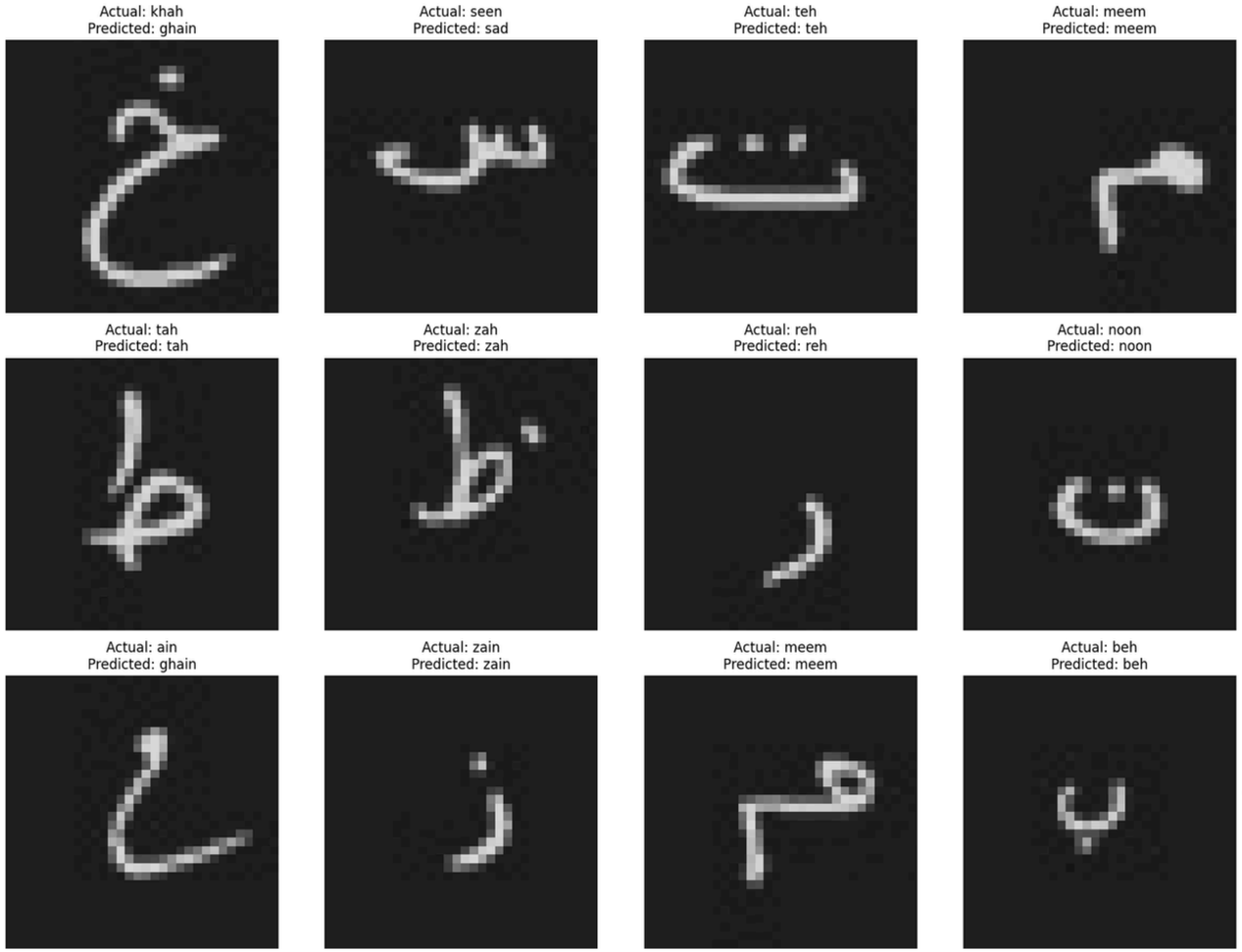


TESTING

```
# Evaluate the model on the test set
test_loss, test_accuracy = model.evaluate(test_images, test_labels_one_hot)
print(f"Test Loss: {test_loss}")
print(f"Test Accuracy: {test_accuracy}")
```

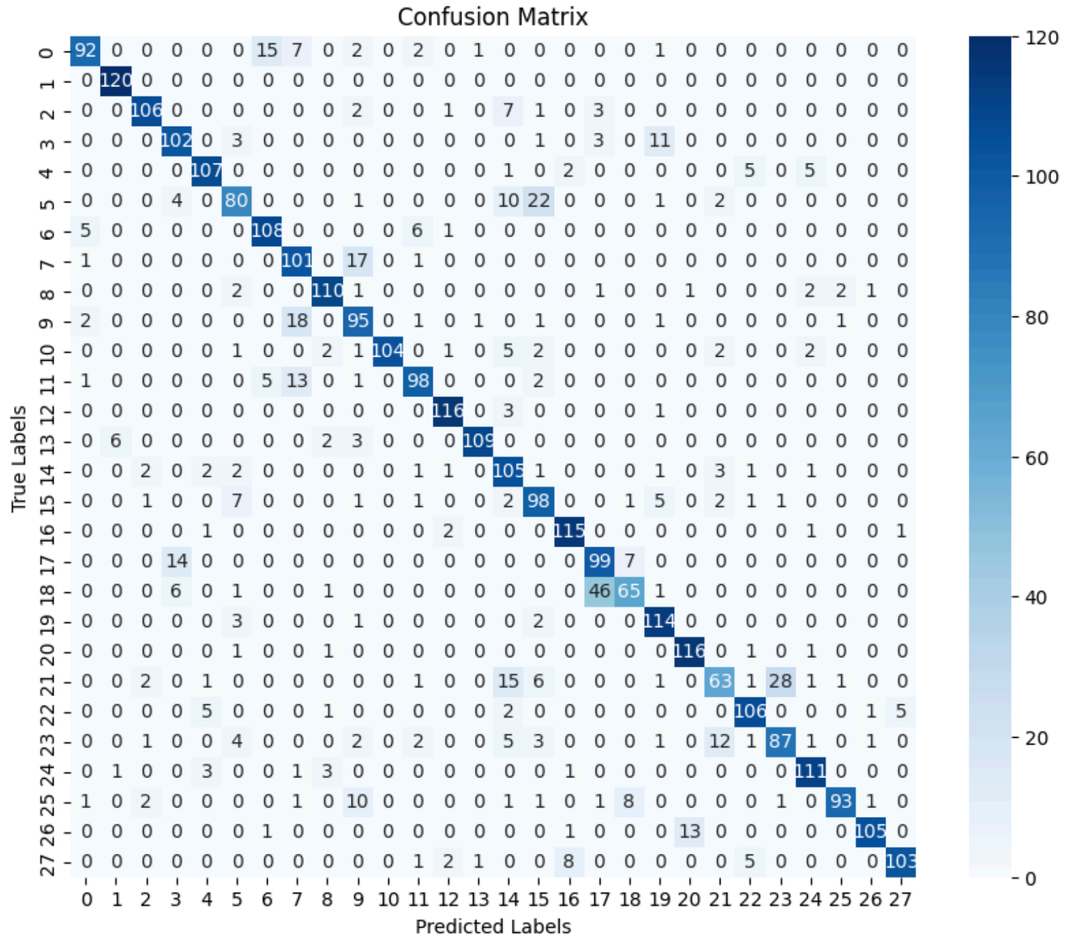
```
105/105 [==============================] - 0s 4ms/step - loss: 0.8378 - accuracy: 0.8417
Test Loss: 0.8377987146377563
Test Accuracy: 0.8416666388511658
```

testing accuracy: 0.84



CLASSIFICATION REPORT & CONFUSION MATRIX

```
105/105 [==============================] - 0s 3ms/step
              precision    recall  f1-score   support

           0       0.90      0.77      0.83       120
           1       0.94      1.00      0.97       120
           2       0.93      0.88      0.91       120
           3       0.81      0.85      0.83       120
           4       0.90      0.89      0.90       120
           5       0.77      0.67      0.71       120
           6       0.84      0.90      0.87       120
           7       0.72      0.84      0.77       120
           8       0.92      0.92      0.92       120
           9       0.69      0.79      0.74       120
          10       1.00      0.87      0.93       120
          11       0.86      0.82      0.84       120
          12       0.94      0.97      0.95       120
          13       0.97      0.91      0.94       120
          14       0.67      0.88      0.76       120
          15       0.70      0.82      0.75       120
          16       0.91      0.96      0.93       120
          17       0.65      0.82      0.73       120
          18       0.80      0.54      0.65       120
          19       0.83      0.95      0.88       120
          20       0.89      0.97      0.93       120
          21       0.75      0.53      0.62       120
          22       0.88      0.88      0.88       120
          23       0.74      0.72      0.73       120
          24       0.89      0.93      0.91       120
          25       0.96      0.78      0.86       120
          26       0.96      0.88      0.92       120
          27       0.94      0.86      0.90       120

    accuracy                           0.84      3360
   macro avg       0.85      0.84      0.84      3360
weighted avg       0.85      0.84      0.84      3360
```



CHANGE 2 TO THE MODEL

changing convolution size, kernel size, L2 regularization
&
Early Stopping

IMPLEMENTATION

```python
def create_advanced_cnn_model(input_shape, num_classes):
    model = Sequential([
        # First conv layer
        Conv2D(128, kernel_size=(5, 5), strides=(1, 1), activation='relu', input_shape=input_shape, padding='same', kernel_regularizer=l2(0.01)),
        MaxPooling2D(pool_size=(2, 2)),

        # Second conv layer
        Conv2D(256, kernel_size=(5, 5), strides=(1, 1), activation='relu', padding='same', kernel_regularizer=l2(0.01)),
        MaxPooling2D(pool_size=(2, 2)),

        # Third conv layer
        Conv2D(512, kernel_size=(3, 3), strides=(1, 1), activation='relu', padding='same', kernel_regularizer=l2(0.01)),
        MaxPooling2D(pool_size=(2, 2)),

        # Fourth conv layer
        Conv2D(512, kernel_size=(3, 3), strides=(1, 1), activation='relu', padding='same', kernel_regularizer=l2(0.01)),
        MaxPooling2D(pool_size=(2, 2)),

        # Fifth conv layer
        Conv2D(512, kernel_size=(3, 3), strides=(1, 1), activation='relu', padding='same', kernel_regularizer=l2(0.01)),
        MaxPooling2D(pool_size=(2, 2)),

        # Flattening the layers
        Flatten(),

        # Dense layers
        Dense(1024, activation='relu', kernel_regularizer=l2(0.01)),
        Dropout(0.5),
        Dense(512, activation='relu', kernel_regularizer=l2(0.01)),
        Dropout(0.5),
        Dense(num_classes, activation='softmax')
    ])

    model.compile(optimizer=Adam(), loss='categorical_crossentropy', metrics=['accuracy'])

    # Adding Early Stopping callback
    early_stopping = EarlyStopping(monitor='val_loss', patience=5, restore_best_weights=True)

    return model, early_stopping
```

MODEL SUMMARY

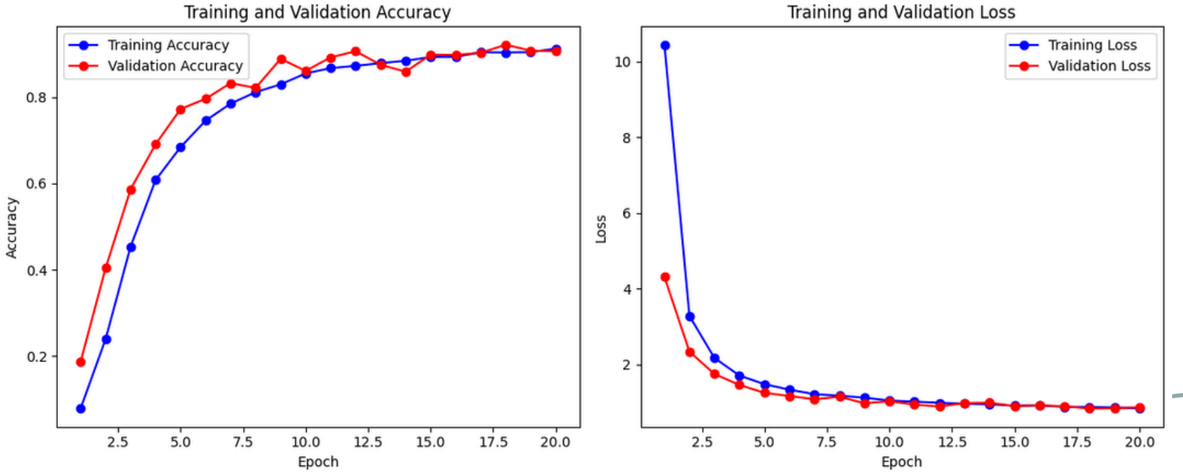
```
Model: "sequential_5"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d_25 (Conv2D)          (None, 32, 32, 128)       3328

 max_pooling2d_25 (MaxPooli  (None, 16, 16, 128)       0
 ng2D)

 conv2d_26 (Conv2D)          (None, 16, 16, 256)       819456

 max_pooling2d_26 (MaxPooli  (None, 8, 8, 256)         0
 ng2D)

 conv2d_27 (Conv2D)          (None, 8, 8, 512)         1180160

 max_pooling2d_27 (MaxPooli  (None, 4, 4, 512)         0
 ng2D)

 conv2d_28 (Conv2D)          (None, 4, 4, 512)         2359808

 max_pooling2d_28 (MaxPooli  (None, 2, 2, 512)         0
 ng2D)

 conv2d_29 (Conv2D)          (None, 2, 2, 512)         2359808

 max_pooling2d_29 (MaxPooli  (None, 1, 1, 512)         0
 ng2D)

 flatten_5 (Flatten)         (None, 512)               0

 dense_15 (Dense)            (None, 1024)              525312

 dropout_10 (Dropout)        (None, 1024)              0

 dense_16 (Dense)            (None, 512)               524800

 dropout_11 (Dropout)        (None, 512)               0

 dense_17 (Dense)            (None, 28)                14364

=================================================================
Total params: 7787036 (29.71 MB)
Trainable params: 7787036 (29.71 MB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
```

TRAINING VS VALIDATION

```
history = model.fit(
    train_images, train_labels_one_hot,
    epochs=20,
    batch_size=32,
    validation_split=0.2,  # Using 20% of the data for validation
    verbose=1
)
```

```
Epoch 1/20
336/336 [==============================] - 11s 22ms/step - loss: 10.4446 - accuracy: 0.0789 - val_loss: 4.3122 - val_accuracy: 0.1871
Epoch 2/20
336/336 [==============================] - 7s 22ms/step - loss: 3.2791 - accuracy: 0.2398 - val_loss: 2.3394 - val_accuracy: 0.4040
Epoch 3/20
336/336 [==============================] - 7s 22ms/step - loss: 2.1712 - accuracy: 0.4540 - val_loss: 1.7466 - val_accuracy: 0.5867
Epoch 4/20
336/336 [==============================] - 8s 22ms/step - loss: 1.7061 - accuracy: 0.6082 - val_loss: 1.4585 - val_accuracy: 0.6908
Epoch 5/20
336/336 [==============================] - 7s 22ms/step - loss: 1.4795 - accuracy: 0.6844 - val_loss: 1.2502 - val_accuracy: 0.7727
Epoch 6/20
336/336 [==============================] - 7s 21ms/step - loss: 1.3325 - accuracy: 0.7456 - val_loss: 1.1662 - val_accuracy: 0.7961
Epoch 7/20
336/336 [==============================] - 7s 21ms/step - loss: 1.2157 - accuracy: 0.7851 - val_loss: 1.0806 - val_accuracy: 0.8326
Epoch 8/20
336/336 [==============================] - 7s 21ms/step - loss: 1.1742 - accuracy: 0.8116 - val_loss: 1.1469 - val_accuracy: 0.8214
Epoch 9/20
336/336 [==============================] - 7s 21ms/step - loss: 1.1223 - accuracy: 0.8296 - val_loss: 0.9756 - val_accuracy: 0.8888
Epoch 10/20
336/336 [==============================] - 7s 20ms/step - loss: 1.0459 - accuracy: 0.8549 - val_loss: 1.0275 - val_accuracy: 0.8609
Epoch 11/20
336/336 [==============================] - 7s 22ms/step - loss: 1.0167 - accuracy: 0.8673 - val_loss: 0.9394 - val_accuracy: 0.8921
Epoch 12/20
336/336 [==============================] - 7s 20ms/step - loss: 0.9865 - accuracy: 0.8725 - val_loss: 0.8881 - val_accuracy: 0.9062
Epoch 13/20
336/336 [==============================] - 7s 22ms/step - loss: 0.9625 - accuracy: 0.8788 - val_loss: 0.9801 - val_accuracy: 0.8750
Epoch 14/20
336/336 [==============================] - 7s 20ms/step - loss: 0.9483 - accuracy: 0.8843 - val_loss: 0.9882 - val_accuracy: 0.8590
Epoch 15/20
336/336 [==============================] - 7s 22ms/step - loss: 0.9087 - accuracy: 0.8929 - val_loss: 0.9007 - val_accuracy: 0.8984
Epoch 16/20
336/336 [==============================] - 7s 21ms/step - loss: 0.9185 - accuracy: 0.8935 - val_loss: 0.9169 - val_accuracy: 0.8977
Epoch 17/20
336/336 [==============================] - 7s 21ms/step - loss: 0.8769 - accuracy: 0.9040 - val_loss: 0.8936 - val_accuracy: 0.9025
Epoch 18/20
336/336 [==============================] - 7s 21ms/step - loss: 0.8775 - accuracy: 0.9036 - val_loss: 0.8343 - val_accuracy: 0.9215
Epoch 19/20
336/336 [==============================] - 7s 21ms/step - loss: 0.8610 - accuracy: 0.9042 - val_loss: 0.8435 - val_accuracy: 0.9077
Epoch 20/20
336/336 [==============================] - 7s 21ms/step - loss: 0.8393 - accuracy: 0.9124 - val_loss: 0.8673 - val_accuracy: 0.9062
```

training accuracy: 0.91
validation accuracy: 0.90

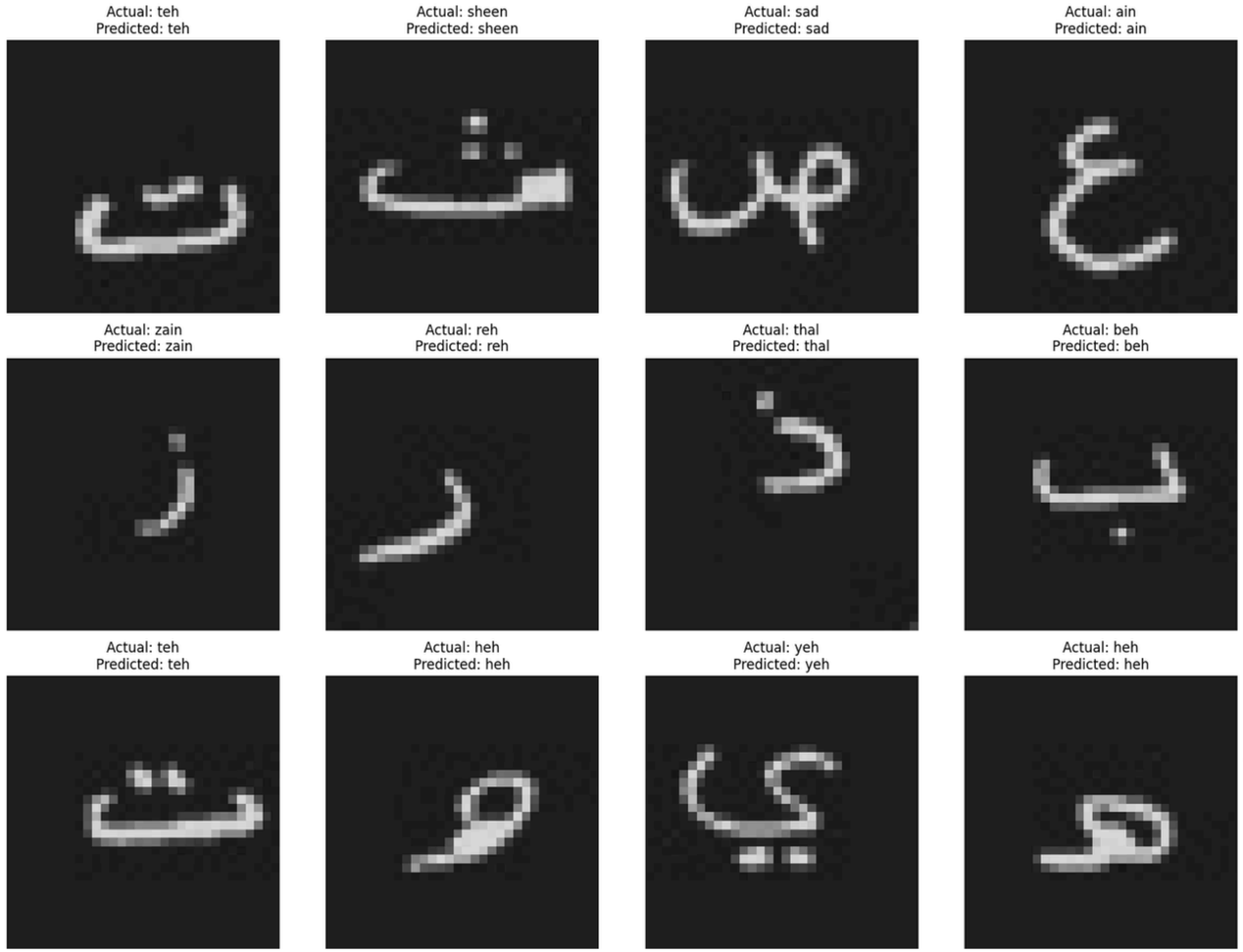


TESTING

```
# Evaluate the model on the test set
test_loss, test_accuracy = model.evaluate(test_images, test_labels_one_hot)
print(f"Test Loss: {test_loss}")
print(f"Test Accuracy: {test_accuracy}")
```

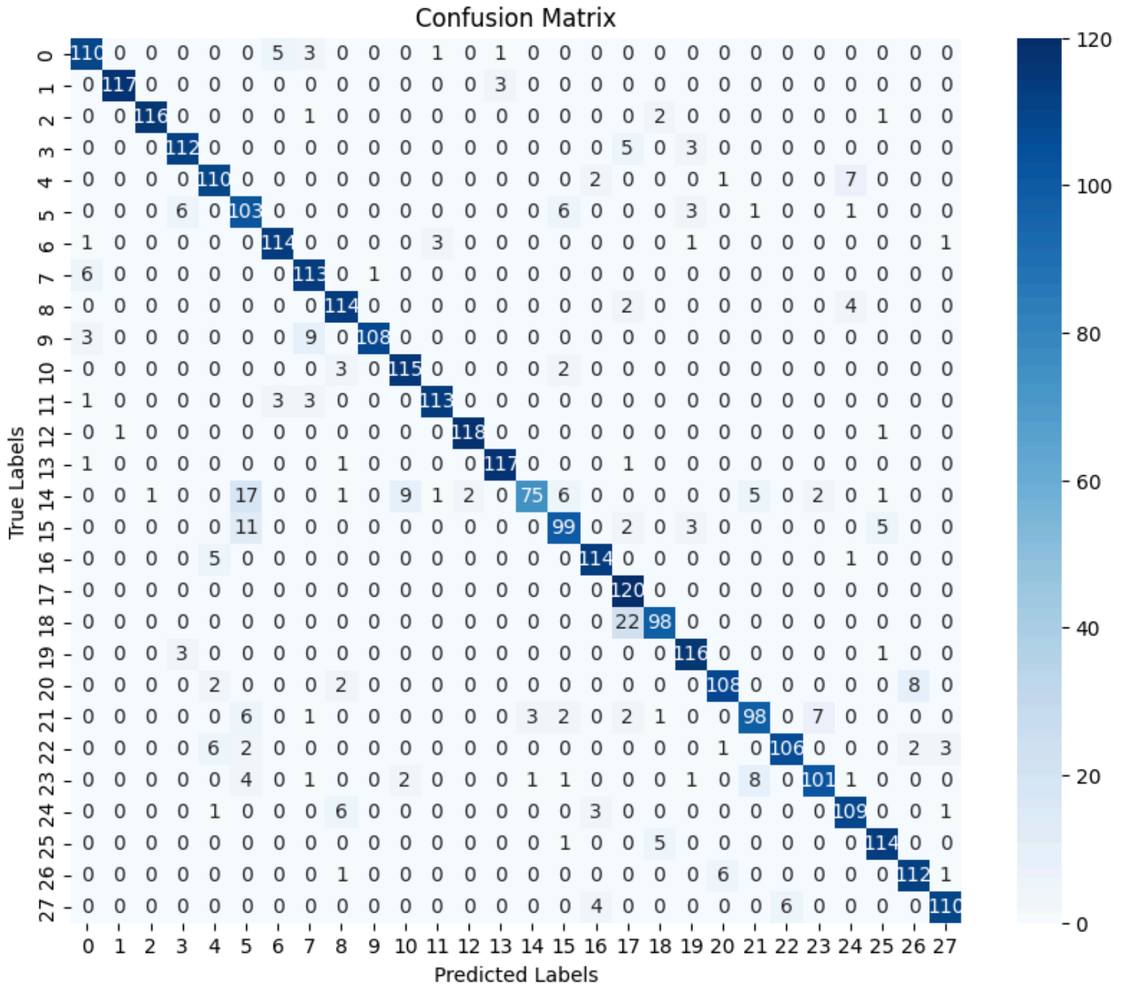
```
105/105 [==============================] - 1s 7ms/step - loss: 0.8369 - accuracy: 0.9107
Test Loss: 0.8368542194366455
Test Accuracy: 0.9107142686843872
```

testing accuracy: 0.91



CLASSIFICATION REPORT & CONFUSION MATRIX

```
105/105 [==============================] - 1s 5ms/step
              precision    recall  f1-score   support

           0       0.90      0.92      0.91       120
           1       0.99      0.97      0.98       120
           2       0.99      0.97      0.98       120
           3       0.93      0.93      0.93       120
           4       0.89      0.92      0.90       120
           5       0.72      0.86      0.78       120
           6       0.93      0.95      0.94       120
           7       0.86      0.94      0.90       120
           8       0.89      0.95      0.92       120
           9       0.99      0.90      0.94       120
          10       0.91      0.96      0.93       120
          11       0.96      0.94      0.95       120
          12       0.98      0.98      0.98       120
          13       0.97      0.97      0.97       120
          14       0.95      0.62      0.75       120
          15       0.85      0.82      0.84       120
          16       0.93      0.95      0.94       120
          17       0.78      1.00      0.88       120
          18       0.92      0.82      0.87       120
          19       0.91      0.97      0.94       120
          20       0.93      0.90      0.92       120
          21       0.88      0.82      0.84       120
          22       0.95      0.88      0.91       120
          23       0.92      0.84      0.88       120
          24       0.89      0.91      0.90       120
          25       0.93      0.95      0.94       120
          26       0.92      0.93      0.93       120
          27       0.95      0.92      0.93       120

    accuracy                           0.91      3360
   macro avg       0.91      0.91      0.91      3360
weighted avg       0.91      0.91      0.91      3360
```



CHANGE 3 TO THE MODEL

The use of Keras Tuner library

IMPLEMENTATION

```python
def create_advanced_cnn_model(hp):
    model = Sequential()

    # Define hyperparameters
    hp_units = hp.Int('units', min_value=32, max_value=1024, step=32)
    hp_dropout = hp.Float('dropout', min_value=0.1, max_value=0.5, step=0.1)
    hp_learning_rate = hp.Choice('learning_rate', values=[1e-2, 1e-3, 1e-4])

    # First conv layer
    model.add(Conv2D(64, kernel_size=(3, 3), strides=(1, 1), activation='relu', input_shape=input_shape, padding='same', kernel_regularizer=l2(0.001)))
    model.add(MaxPooling2D(pool_size=(2, 2)))

    # Second conv layer
    model.add(Conv2D(128, kernel_size=(3, 3), strides=(1, 1), activation='relu', padding='same', kernel_regularizer=l2(0.001)))
    model.add(MaxPooling2D(pool_size=(2, 2)))

    # Third conv layer
    model.add(Conv2D(256, kernel_size=(3, 3), strides=(1, 1), activation='relu', padding='same', kernel_regularizer=l2(0.001)))
    model.add(MaxPooling2D(pool_size=(2, 2)))

    # Fourth conv layer
    model.add(Conv2D(512, kernel_size=(3, 3), strides=(1, 1), activation='relu', padding='same', kernel_regularizer=l2(0.001)))
    model.add(MaxPooling2D(pool_size=(2, 2)))

    # Fifth conv layer
    model.add(Conv2D(512, kernel_size=(3, 3), strides=(1, 1), activation='relu', padding='same', kernel_regularizer=l2(0.001)))
    model.add(MaxPooling2D(pool_size=(2, 2)))

    # Flattening the layers
    model.add(Flatten())

    # Dense layers with hyperparameters
    model.add(Dense(units=hp_units, activation='relu', kernel_regularizer=l2(0.001)))
    model.add(Dropout(hp_dropout))
    model.add(Dense(units=hp_units//2, activation='relu', kernel_regularizer=l2(0.001)))
    model.add(Dropout(hp_dropout))

    # Output layer
    model.add(Dense(num_classes, activation='softmax'))

    # Compile the model with the defined learning rate
    model.compile(optimizer=Adam(learning_rate=hp_learning_rate), loss='categorical_crossentropy', metrics=['accuracy'])

    return model
```

```python
class CNNHyperModel(HyperModel):
    def __init__(self, input_shape, num_classes):
        self.input_shape = input_shape
        self.num_classes = num_classes

    def build(self, hp):
        return create_advanced_cnn_model(hp)
```

```python
hypermodel = CNNHyperModel(input_shape, num_classes)
tuner = Hyperband(
    hypermodel,
    objective='val_accuracy',
    max_epochs=10,
    directory='tuner_dir',
    project_name='cnn_hyperband'
)
```

```python
tuner.search(x_train, y_train, epochs=10, validation_data=(x_val, y_val))
```

```
Trial 30 Complete [00h 01m 25s]
val_accuracy: 0.9367559552192688

Best val_accuracy So Far: 0.9419642686843872
Total elapsed time: 00h 12m 37s
```

```python
# Retrieve the best hyperparameters
best_hyperparameters = tuner.get_best_hyperparameters(1)[0]
best_model = tuner.hypermodel.build(best_hyperparameters)
print('the best hyperparameter is: ', best_hyperparameters)
print('the best model is: ', best_model)
```

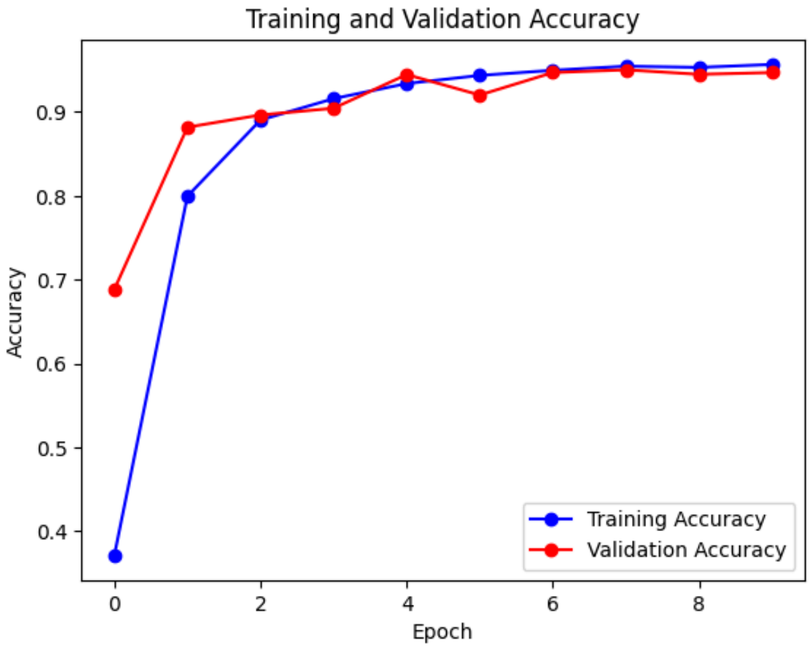
```
the best hyperparameter is:  <keras_tuner.src.engine.hyperparameters.hyperparameters.HyperParameters object at 0x7e7a807dec50>
the best model is:  <keras.src.engine.sequential.Sequential object at 0x7e7a8851be50>
```

TRAINING VS VALIDATION

```
# Train the best model on the full training dataset
history = best_model.fit(train_images, train_labels_one_hot, epochs=10, validation_data=(test_images, test_labels_one_hot))
```

```
Epoch 1/10
420/420 [==============================] - 7s 12ms/step - loss: 3.1701 - accuracy: 0.3712 - val_loss: 1.6809 - val_accuracy: 0.6884
Epoch 2/10
420/420 [==============================] - 5s 12ms/step - loss: 1.2753 - accuracy: 0.7996 - val_loss: 0.9502 - val_accuracy: 0.8818
Epoch 3/10
420/420 [==============================] - 5s 12ms/step - loss: 0.8290 - accuracy: 0.8904 - val_loss: 0.7531 - val_accuracy: 0.8964
Epoch 4/10
420/420 [==============================] - 5s 11ms/step - loss: 0.6553 - accuracy: 0.9159 - val_loss: 0.6798 - val_accuracy: 0.9045
Epoch 5/10
420/420 [==============================] - 5s 12ms/step - loss: 0.5574 - accuracy: 0.9339 - val_loss: 0.5198 - val_accuracy: 0.9446
Epoch 6/10
420/420 [==============================] - 5s 12ms/step - loss: 0.4935 - accuracy: 0.9436 - val_loss: 0.5540 - val_accuracy: 0.9202
Epoch 7/10
420/420 [==============================] - 5s 11ms/step - loss: 0.4576 - accuracy: 0.9497 - val_loss: 0.4662 - val_accuracy: 0.9470
Epoch 8/10
420/420 [==============================] - 5s 11ms/step - loss: 0.4294 - accuracy: 0.9546 - val_loss: 0.4600 - val_accuracy: 0.9503
Epoch 9/10
420/420 [==============================] - 5s 13ms/step - loss: 0.4293 - accuracy: 0.9532 - val_loss: 0.4672 - val_accuracy: 0.9449
Epoch 10/10
420/420 [==============================] - 5s 11ms/step - loss: 0.4094 - accuracy: 0.9567 - val_loss: 0.4414 - val_accuracy: 0.9470
```

training accuracy: 0.95
validation accuracy: 0.94

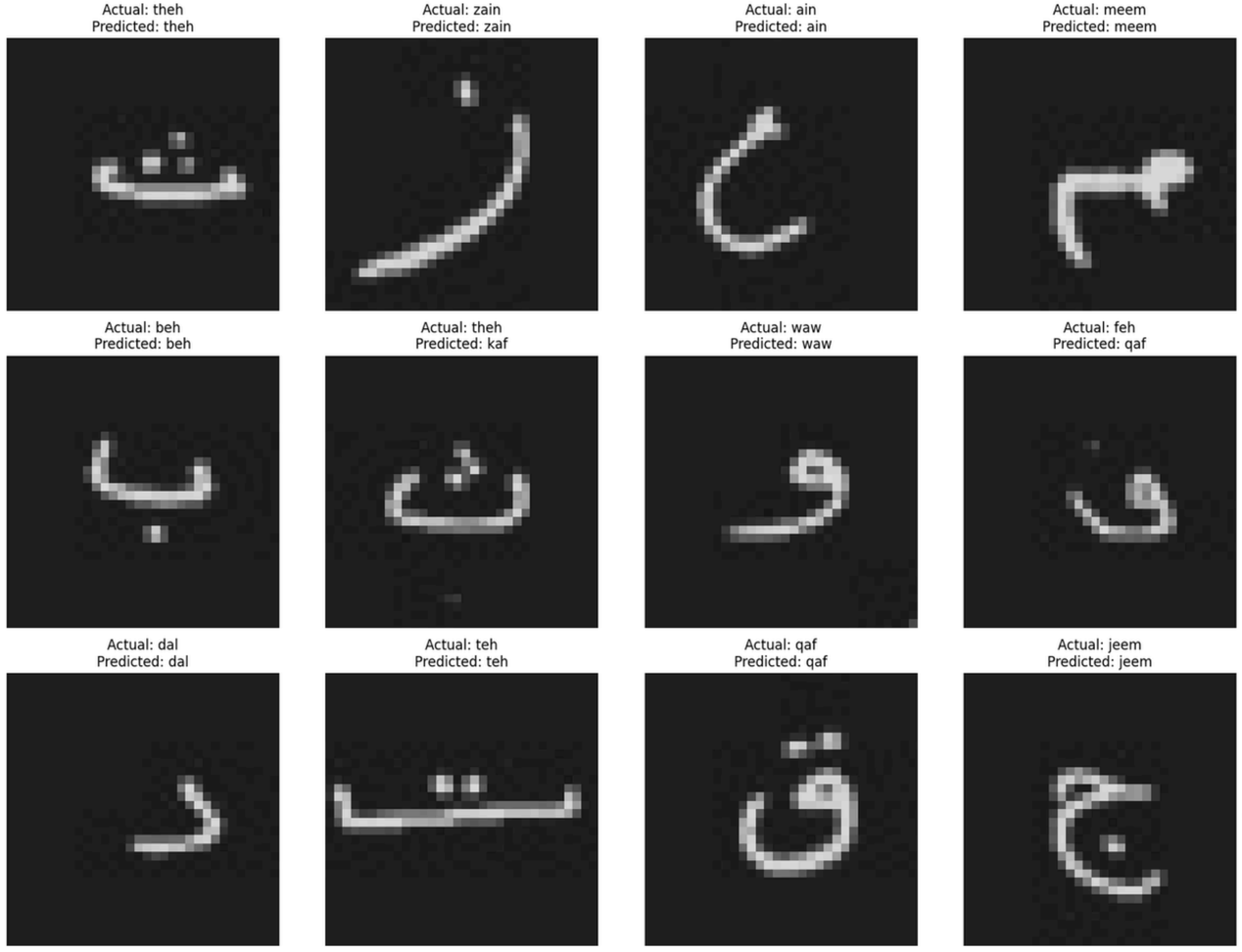


TESTING

```
# Evaluate the best model on the test dataset
test_loss, test_accuracy = best_model.evaluate(test_images, test_labels_one_hot)
print("Test Loss:", test_loss)
print("Test Accuracy:", test_accuracy)
```

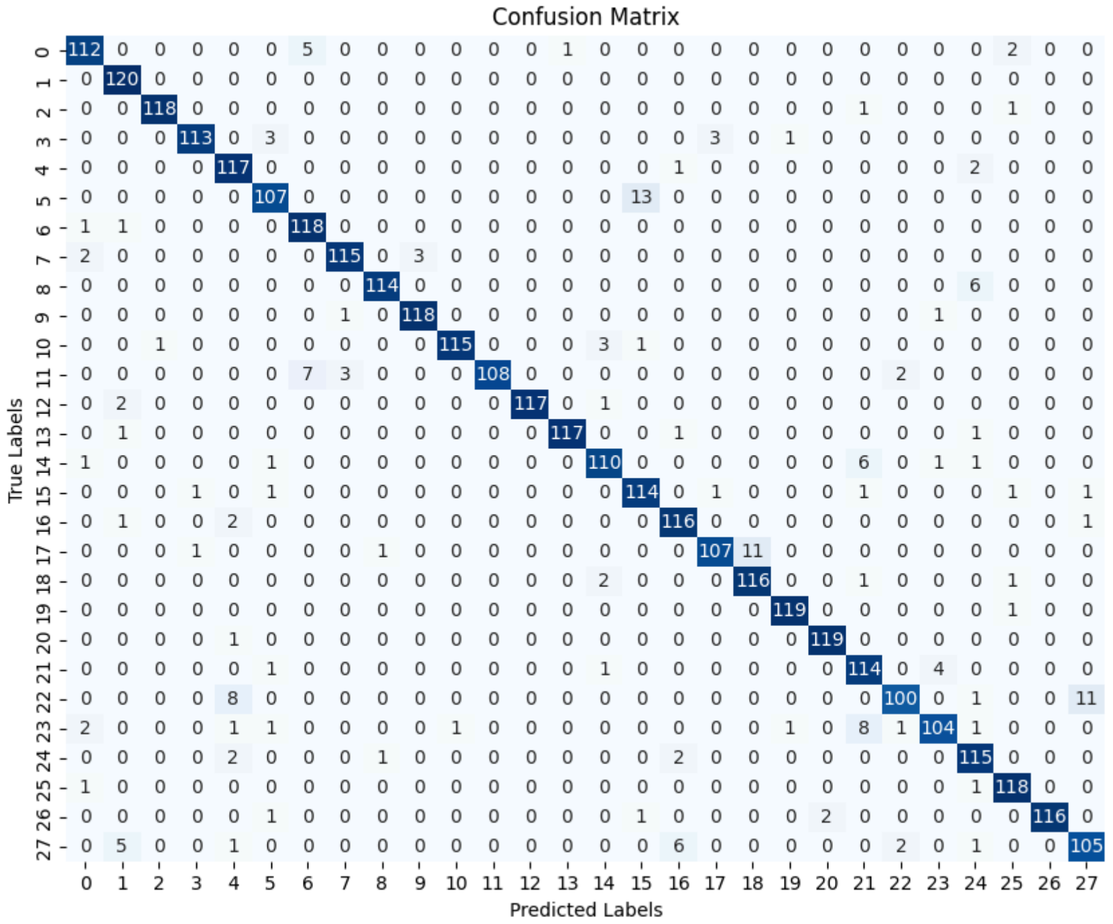
```
105/105 [==============================] - 0s 4ms/step - loss: 0.4414 - accuracy: 0.9470
Test Loss: 0.4414333403110504
Test Accuracy: 0.9470238089561462
```

testing accuarcy: 0.94



CLASSIFICATION REPORT & CONFUSION MATRIX

```
105/105 [==============================] - 0s 3ms/step
              precision    recall  f1-score   support

           0       0.94      0.93      0.94       120
           1       0.92      1.00      0.96       120
           2       0.99      0.98      0.99       120
           3       0.98      0.94      0.96       120
           4       0.89      0.97      0.93       120
           5       0.93      0.89      0.91       120
           6       0.91      0.98      0.94       120
           7       0.97      0.96      0.96       120
           8       0.98      0.95      0.97       120
           9       0.98      0.98      0.98       120
          10       0.99      0.96      0.97       120
          11       1.00      0.90      0.95       120
          12       1.00      0.97      0.99       120
          13       0.99      0.97      0.98       120
          14       0.94      0.92      0.93       120
          15       0.88      0.95      0.92       120
          16       0.92      0.97      0.94       120
          17       0.96      0.89      0.93       120
          18       0.91      0.97      0.94       120
          19       0.98      0.99      0.99       120
          20       0.98      0.99      0.99       120
          21       0.87      0.95      0.91       120
          22       0.95      0.83      0.89       120
          23       0.95      0.87      0.90       120
          24       0.89      0.96      0.92       120
          25       0.95      0.98      0.97       120
          26       1.00      0.97      0.98       120
          27       0.89      0.88      0.88       120

    accuracy                           0.95      3360
   macro avg       0.95      0.95      0.95      3360
weighted avg       0.95      0.95      0.95      3360
```



CONCLUSION

Model	Training	Validation	Testing
CNN model	0.97	0.92	0.93
CNN with changes 1	0.87	0.84	0.84
CNN with changes 2	0.91	0.90	0.91
CNN with changes 3	0.95	0.94	0.94

CNN Model: shows high accuracy across training, validation, and testing with scores of 0.97, 0.92, and 0.93, respectively. This model demonstrates a strong ability to generalize from training data to unseen validation and testing data, although there's a slight overfitting as indicated by the difference between training and validation/testing accuracy.

CNN Model change 1: has the lowest performance among the four, with a training accuracy of 0.87 and both validation and testing accuracies at 0.84. This model appears to be more consistent between training and external data results but has lower overall accuracy.

CNN Model change 2:shows good performance with training, validation, and testing accuracies of 0.91, 0.90, and 0.91, respectively. The consistency across these metrics suggests that the model generalizes well without significant overfitting.

CNN Model change 3: has the **highest performance** with a training accuracy of 0.95 and validation and testing accuracies of 0.94. It presents a very high level of generalization capability and minimal overfitting.

THANK YOU