```python
import os
import numpy as np
import pandas as pd
from matplotlib import pyplot as plt
import tensorflow as tf
from tensorflow import keras
from tensorflow.keras.preprocessing.image import ImageDataGenerator

# Check for all available physical devices using TensorFlow
for device in tf.config.list_physical_devices():
    print( device.name)
```

```
/physical_device:CPU:0
```

```python
datasets =
['/kaggle/input/arabic-letters-classification/Final_Arabic_Alpha_datas
et/Final_Arabic_Alpha_dataset/train',\

'/kaggle/input/arabic-letters-classification/Final_Arabic_Alpha_datase
t/Final_Arabic_Alpha_dataset/test',

'/kaggle/input/arabic-letters-classification/Final_Arabic_Alpha_datase
t/Final_Arabic_Alpha_dataset']
NUM_CLASS = 65
IMAGE_SIZE = (160,160)
BATCH_SIZE = 512
SEED = 43
EPOCHS = 50

train_images = tf.keras.utils.image_dataset_from_directory(
    datasets[0],
    validation_split=0.2,
    color_mode='grayscale',
    label_mode="categorical",
    subset="training",
    shuffle=True,
    seed=SEED,
    image_size=IMAGE_SIZE,
    batch_size=BATCH_SIZE)
```

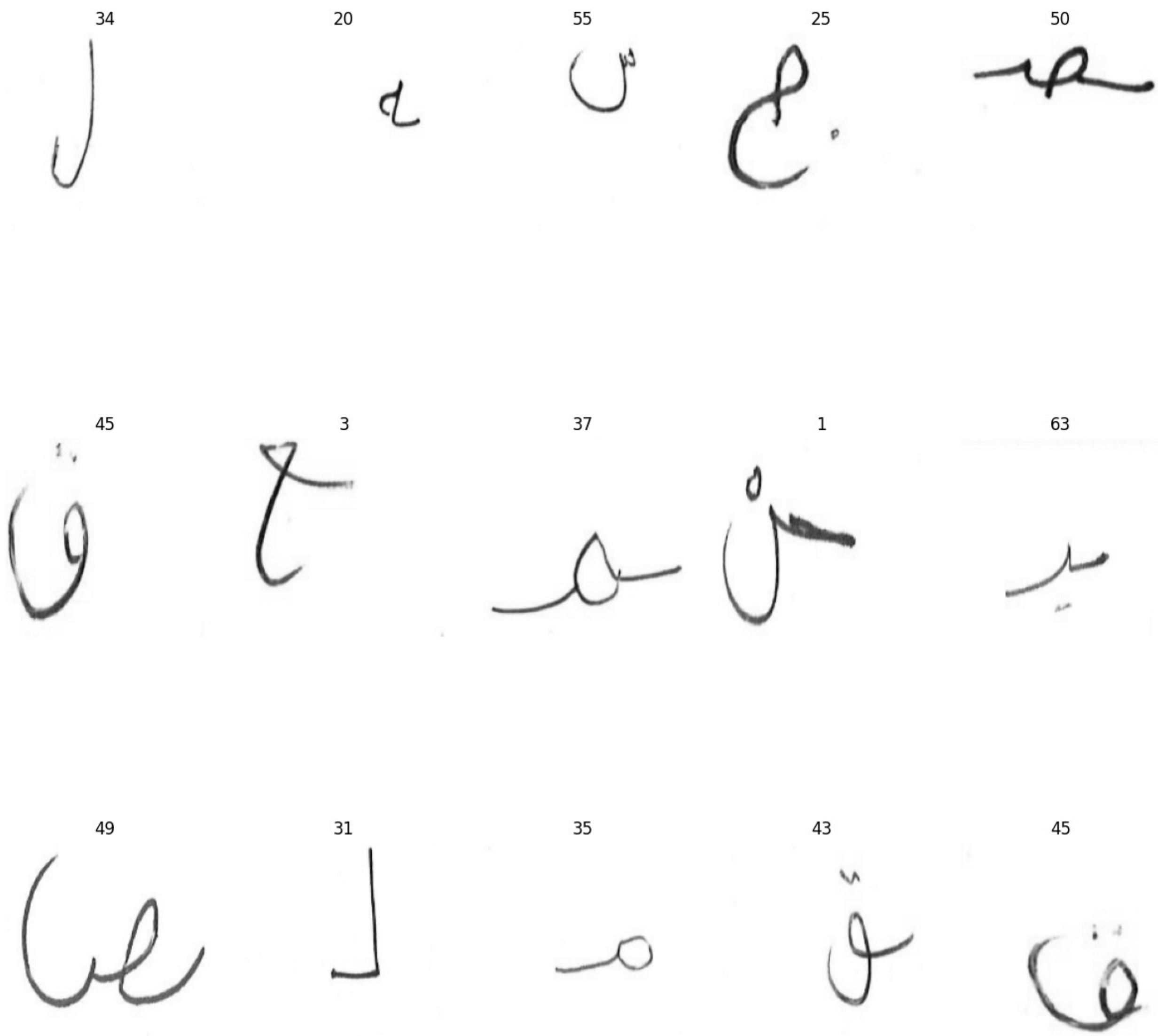
```
Found 42559 files belonging to 65 classes.
Using 34048 files for training.
```

```python
train_validation = tf.keras.utils.image_dataset_from_directory(
    datasets[0],
    validation_split=0.2,
    label_mode="categorical",
    color_mode='grayscale',
    subset="validation",
    shuffle=True,
```

```
    seed=SEED,
    image_size=IMAGE_SIZE,
    batch_size=BATCH_SIZE)
```

```
Found 42559 files belonging to 65 classes.
Using 8511 files for validation.
```

```
class_names = train_images.class_names
plt.figure(figsize=(15, 15))
for images, labels in train_images.take(1):
    for i in range(15):
        ax = plt.subplot(3, 5, i + 1)
        plt.imshow(images[i].numpy().astype("uint8"),
cmap=plt.cm.Greys_r)
        plt.title(class_names[np.where(np.array(labels[i])==1)[0][0]])
        plt.axis("off")
```



```python
for images, labels in train_images:
    print(images.shape)
    print(labels.shape)
    break
```

```
(512, 160, 160, 1)
(512, 65)
```

```python
model = tf.keras.Sequential([
    # Rescale pixel values to the range [0, 1]
    tf.keras.layers.Rescaling(1./255),

    # Data augmentation: Random rotation
    tf.keras.layers.experimental.preprocessing.RandomRotation(0.1),

    # Data augmentation: Random zoom
    tf.keras.layers.experimental.preprocessing.RandomZoom(0.1),

    # Convolutional layers with max pooling
    tf.keras.layers.Conv2D(32, (3, 3), activation='relu',
padding='same'),
    tf.keras.layers.MaxPooling2D(2, 2),

    tf.keras.layers.Conv2D(64, (3, 3), activation='relu',
padding='same'),
    tf.keras.layers.MaxPooling2D(2, 2),

    tf.keras.layers.Conv2D(64, (3, 3), activation='relu',
padding='same'),
    tf.keras.layers.MaxPooling2D(2, 2),

    tf.keras.layers.Conv2D(128, (3, 3), activation='relu',
padding='same'),
    tf.keras.layers.MaxPooling2D(2, 2),

    tf.keras.layers.Conv2D(256, (3, 3), activation='relu',
padding='same'),
    tf.keras.layers.MaxPooling2D(2, 2),

    # Flatten the output for dense layers
    tf.keras.layers.Flatten(),

    # Dense layers with relu activation and dropout for regularization
    tf.keras.layers.Dense(1024, activation='relu'),
    tf.keras.layers.Dropout(0.2),

    # Output layer with softmax activation (assuming a classification
task)
    tf.keras.layers.Dense(NUM_CLASS, activation=tf.nn.softmax)
])
```

```
model.build(input_shape=(BATCH_SIZE, IMAGE_SIZE[0], IMAGE_SIZE[1], 1))
model.summary()
```

```
Model: "sequential_2"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 rescaling_4 (Rescaling)     (512, 160, 160, 1)        0

 random_rotation_4 (RandomR  (512, 160, 160, 1)        0
 otation)

 random_zoom_4 (RandomZoom)  (512, 160, 160, 1)        0

 conv2d_20 (Conv2D)          (512, 160, 160, 32)       320

 max_pooling2d_20 (MaxPooli  (512, 80, 80, 32)         0
 ng2D)

 conv2d_21 (Conv2D)          (512, 80, 80, 64)         18496

 max_pooling2d_21 (MaxPooli  (512, 40, 40, 64)         0
 ng2D)

 conv2d_22 (Conv2D)          (512, 40, 40, 64)         36928

 max_pooling2d_22 (MaxPooli  (512, 20, 20, 64)         0
 ng2D)

 conv2d_23 (Conv2D)          (512, 20, 20, 128)        73856

 max_pooling2d_23 (MaxPooli  (512, 10, 10, 128)        0
 ng2D)

 conv2d_24 (Conv2D)          (512, 10, 10, 256)        295168

 max_pooling2d_24 (MaxPooli  (512, 5, 5, 256)          0
 ng2D)

 flatten_4 (Flatten)         (512, 6400)               0

 dense_6 (Dense)             (512, 1024)               6554624

 dropout_4 (Dropout)         (512, 1024)               0

 dense_7 (Dense)             (512, 65)                 66625

=================================================================
Total params: 7046017 (26.88 MB)
Trainable params: 7046017 (26.88 MB)
```

```
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
```

```
learning_rate = 0.001

# Exponential decay learning rate schedule
lr_schedule = tf.keras.optimizers.schedules.ExponentialDecay(
    initial_learning_rate=learning_rate,
    decay_steps=100000,
    decay_rate=0.96,
    staircase=True
)

# Compile the model with the specified optimizer, loss, and metrics
model.compile(
    optimizer=tf.keras.optimizers.Adam(learning_rate=lr_schedule),
    loss='categorical_crossentropy',
    metrics=['accuracy']
)

h1 = model.fit(train_images,
              validation_data=train_validation,
              epochs=EPOCHS)
```

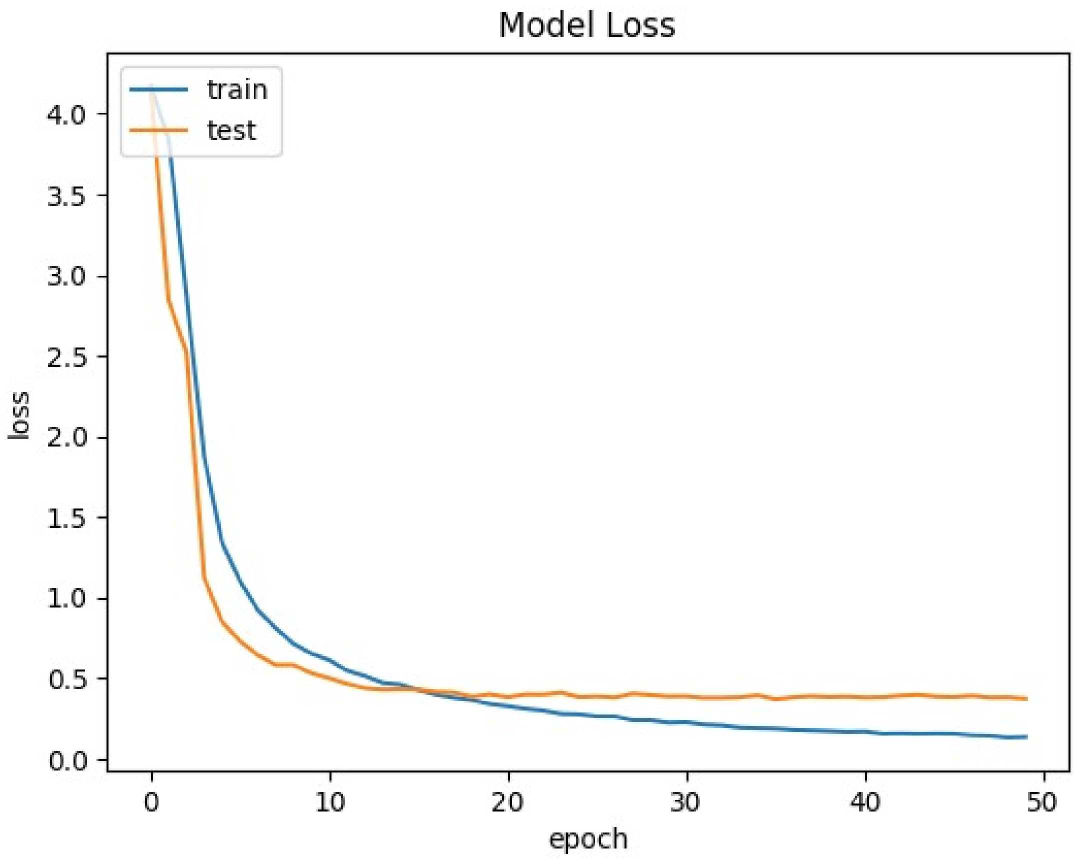
```
Epoch 1/50
67/67 [==============================] - 29s 366ms/step - loss: 4.1752
- accuracy: 0.0150 - val_loss: 4.1685 - val_accuracy: 0.0213
Epoch 2/50
67/67 [==============================] - 26s 363ms/step - loss: 3.8465
- accuracy: 0.0729 - val_loss: 2.8456 - val_accuracy: 0.2424
Epoch 3/50
67/67 [==============================] - 26s 363ms/step - loss: 2.8780
- accuracy: 0.2411 - val_loss: 2.5210 - val_accuracy: 0.3118
Epoch 4/50
67/67 [==============================] - 26s 363ms/step - loss: 1.8749
- accuracy: 0.4606 - val_loss: 1.1226 - val_accuracy: 0.6562
Epoch 5/50
67/67 [==============================] - 26s 366ms/step - loss: 1.3408
- accuracy: 0.5959 - val_loss: 0.8489 - val_accuracy: 0.7308
Epoch 6/50
67/67 [==============================] - 27s 366ms/step - loss: 1.1033
- accuracy: 0.6582 - val_loss: 0.7295 - val_accuracy: 0.7708
Epoch 7/50
67/67 [==============================] - 27s 378ms/step - loss: 0.9222
- accuracy: 0.7148 - val_loss: 0.6430 - val_accuracy: 0.7965
Epoch 8/50
67/67 [==============================] - 27s 380ms/step - loss: 0.8104
- accuracy: 0.7521 - val_loss: 0.5816 - val_accuracy: 0.8199
Epoch 9/50
67/67 [==============================] - 27s 373ms/step - loss: 0.7125
```

```
- accuracy: 0.7772 - val_loss: 0.5817 - val_accuracy: 0.8176
Epoch 10/50
67/67 [==============================] - 27s 370ms/step - loss: 0.6527
- accuracy: 0.7941 - val_loss: 0.5305 - val_accuracy: 0.8406
Epoch 11/50
67/67 [==============================] - 27s 371ms/step - loss: 0.6114
- accuracy: 0.8074 - val_loss: 0.4993 - val_accuracy: 0.8438
Epoch 12/50
67/67 [==============================] - 27s 374ms/step - loss: 0.5467
- accuracy: 0.8270 - val_loss: 0.4648 - val_accuracy: 0.8595
Epoch 13/50
67/67 [==============================] - 27s 372ms/step - loss: 0.5128
- accuracy: 0.8358 - val_loss: 0.4375 - val_accuracy: 0.8616
Epoch 14/50
67/67 [==============================] - 27s 379ms/step - loss: 0.4707
- accuracy: 0.8492 - val_loss: 0.4284 - val_accuracy: 0.8692
Epoch 15/50
67/67 [==============================] - 27s 378ms/step - loss: 0.4606
- accuracy: 0.8528 - val_loss: 0.4312 - val_accuracy: 0.8664
Epoch 16/50
67/67 [==============================] - 27s 372ms/step - loss: 0.4258
- accuracy: 0.8628 - val_loss: 0.4285 - val_accuracy: 0.8699
Epoch 17/50
67/67 [==============================] - 27s 383ms/step - loss: 0.3974
- accuracy: 0.8704 - val_loss: 0.4142 - val_accuracy: 0.8751
Epoch 18/50
67/67 [==============================] - 28s 377ms/step - loss: 0.3787
- accuracy: 0.8764 - val_loss: 0.4099 - val_accuracy: 0.8759
Epoch 19/50
67/67 [==============================] - 27s 379ms/step - loss: 0.3639
- accuracy: 0.8818 - val_loss: 0.3853 - val_accuracy: 0.8804
Epoch 20/50
67/67 [==============================] - 27s 378ms/step - loss: 0.3396
- accuracy: 0.8868 - val_loss: 0.3997 - val_accuracy: 0.8869
Epoch 21/50
67/67 [==============================] - 27s 375ms/step - loss: 0.3257
- accuracy: 0.8930 - val_loss: 0.3824 - val_accuracy: 0.8820
Epoch 22/50
67/67 [==============================] - 27s 381ms/step - loss: 0.3098
- accuracy: 0.8952 - val_loss: 0.3988 - val_accuracy: 0.8851
Epoch 23/50
67/67 [==============================] - 26s 372ms/step - loss: 0.2986
- accuracy: 0.9017 - val_loss: 0.3979 - val_accuracy: 0.8846
Epoch 24/50
67/67 [==============================] - 27s 376ms/step - loss: 0.2785
- accuracy: 0.9068 - val_loss: 0.4105 - val_accuracy: 0.8805
Epoch 25/50
67/67 [==============================] - 27s 375ms/step - loss: 0.2757
- accuracy: 0.9085 - val_loss: 0.3828 - val_accuracy: 0.8847
```

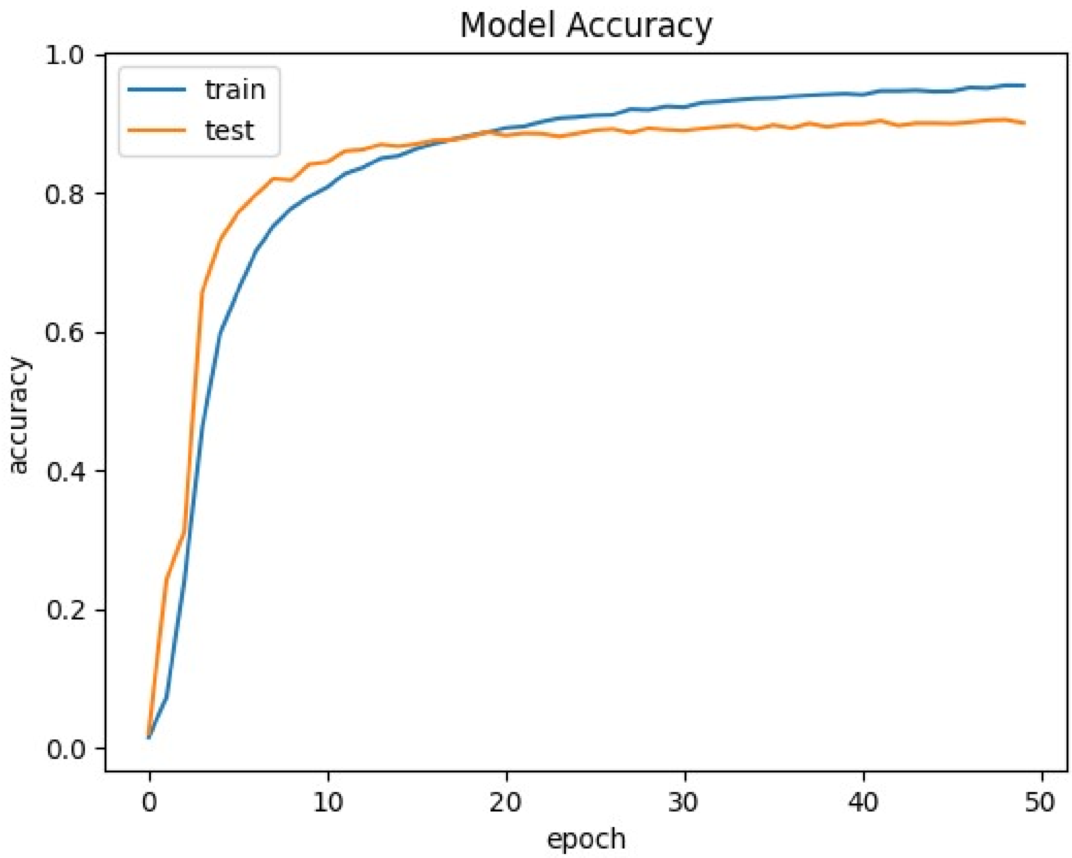
```
Epoch 26/50
67/67 [==============================] - 27s 375ms/step - loss: 0.2625
- accuracy: 0.9111 - val_loss: 0.3869 - val_accuracy: 0.8897
Epoch 27/50
67/67 [==============================] - 27s 381ms/step - loss: 0.2623
- accuracy: 0.9118 - val_loss: 0.3803 - val_accuracy: 0.8914
Epoch 28/50
67/67 [==============================] - 27s 379ms/step - loss: 0.2390
- accuracy: 0.9201 - val_loss: 0.4037 - val_accuracy: 0.8858
Epoch 29/50
67/67 [==============================] - 26s 371ms/step - loss: 0.2387
- accuracy: 0.9191 - val_loss: 0.3956 - val_accuracy: 0.8925
Epoch 30/50
67/67 [==============================] - 27s 376ms/step - loss: 0.2236
- accuracy: 0.9240 - val_loss: 0.3870 - val_accuracy: 0.8904
Epoch 31/50
67/67 [==============================] - 27s 377ms/step - loss: 0.2264
- accuracy: 0.9229 - val_loss: 0.3879 - val_accuracy: 0.8891
Epoch 32/50
67/67 [==============================] - 27s 382ms/step - loss: 0.2118
- accuracy: 0.9292 - val_loss: 0.3782 - val_accuracy: 0.8920
Epoch 33/50
67/67 [==============================] - 27s 378ms/step - loss: 0.2067
- accuracy: 0.9309 - val_loss: 0.3787 - val_accuracy: 0.8946
Epoch 34/50
67/67 [==============================] - 28s 376ms/step - loss: 0.1932
- accuracy: 0.9332 - val_loss: 0.3816 - val_accuracy: 0.8968
Epoch 35/50
67/67 [==============================] - 27s 377ms/step - loss: 0.1895
- accuracy: 0.9352 - val_loss: 0.3938 - val_accuracy: 0.8911
Epoch 36/50
67/67 [==============================] - 27s 376ms/step - loss: 0.1870
- accuracy: 0.9360 - val_loss: 0.3694 - val_accuracy: 0.8973
Epoch 37/50
67/67 [==============================] - 28s 381ms/step - loss: 0.1796
- accuracy: 0.9382 - val_loss: 0.3816 - val_accuracy: 0.8927
Epoch 38/50
67/67 [==============================] - 27s 376ms/step - loss: 0.1758
- accuracy: 0.9398 - val_loss: 0.3884 - val_accuracy: 0.8995
Epoch 39/50
67/67 [==============================] - 28s 384ms/step - loss: 0.1722
- accuracy: 0.9411 - val_loss: 0.3835 - val_accuracy: 0.8947
Epoch 40/50
67/67 [==============================] - 27s 378ms/step - loss: 0.1672
- accuracy: 0.9423 - val_loss: 0.3860 - val_accuracy: 0.8986
Epoch 41/50
67/67 [==============================] - 27s 374ms/step - loss: 0.1694
- accuracy: 0.9408 - val_loss: 0.3802 - val_accuracy: 0.8991
Epoch 42/50
```

```
67/67 [==============================] - 28s 385ms/step - loss: 0.1547
- accuracy: 0.9464 - val_loss: 0.3823 - val_accuracy: 0.9033
Epoch 43/50
67/67 [==============================] - 27s 373ms/step - loss: 0.1567
- accuracy: 0.9463 - val_loss: 0.3915 - val_accuracy: 0.8965
Epoch 44/50
67/67 [==============================] - 27s 376ms/step - loss: 0.1547
- accuracy: 0.9472 - val_loss: 0.3972 - val_accuracy: 0.8999
Epoch 45/50
67/67 [==============================] - 27s 379ms/step - loss: 0.1561
- accuracy: 0.9454 - val_loss: 0.3859 - val_accuracy: 0.8999
Epoch 46/50
67/67 [==============================] - 27s 373ms/step - loss: 0.1551
- accuracy: 0.9459 - val_loss: 0.3832 - val_accuracy: 0.8992
Epoch 47/50
67/67 [==============================] - 27s 372ms/step - loss: 0.1455
- accuracy: 0.9511 - val_loss: 0.3909 - val_accuracy: 0.9012
Epoch 48/50
67/67 [==============================] - 27s 381ms/step - loss: 0.1428
- accuracy: 0.9502 - val_loss: 0.3806 - val_accuracy: 0.9040
Epoch 49/50
67/67 [==============================] - 27s 376ms/step - loss: 0.1314
- accuracy: 0.9542 - val_loss: 0.3816 - val_accuracy: 0.9047
Epoch 50/50
67/67 [==============================] - 27s 379ms/step - loss: 0.1346
- accuracy: 0.9539 - val_loss: 0.3733 - val_accuracy: 0.9002

plt.plot(h1.history['loss'])
plt.plot(h1.history['val_loss'])
plt.title('Model Loss')
plt.ylabel('loss')
plt.xlabel('epoch')
plt.legend(['train', 'test'], loc='upper left')
plt.show()
```




```
plt.plot(h1.history['accuracy'])
plt.plot(h1.history['val_accuracy'])
plt.title('Model Accuracy')
plt.ylabel('accuracy')
plt.xlabel('epoch')
plt.legend(['train', 'test'], loc='upper left')
plt.show()
```

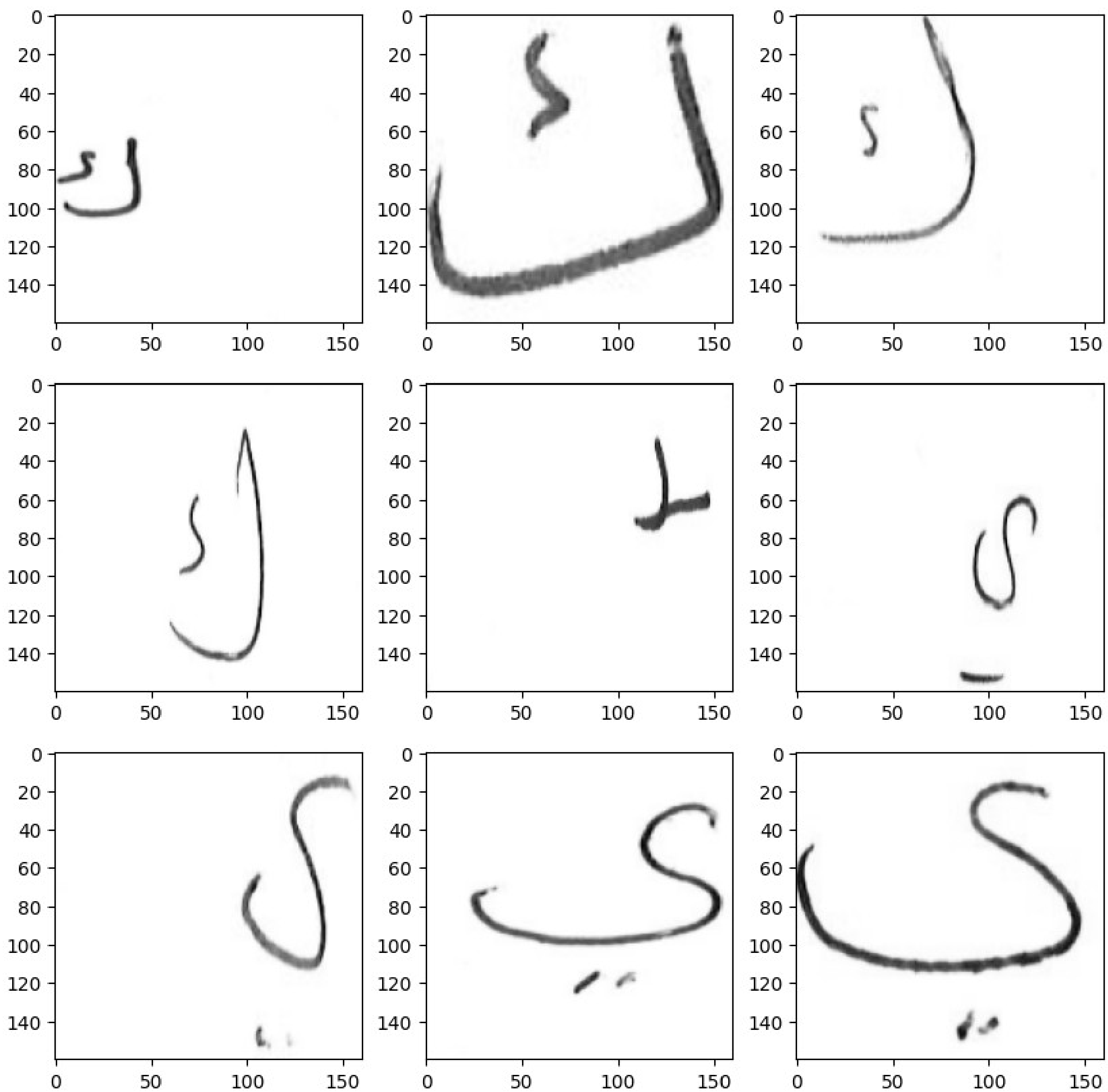



```
model.save('NN_320210321.keras')

test_images = tf.keras.utils.image_dataset_from_directory(
        datasets[1],
        labels=None,
        label_mode="categorical",
        color_mode='grayscale',
        shuffle=False,
        image_size=IMAGE_SIZE,
        batch_size=BATCH_SIZE)
```

```
Found 10640 files belonging to 1 classes.
```

```
plt.figure(figsize=(10, 10))
for images in test_images.take(1): # Takes a batch and shows the first
9 images
    for i in range(9):
        ax = plt.subplot(3, 3, i + 1)
        plt.imshow(images[i].numpy().astype("uint8"),
cmap=plt.cm.Greys_r)
```



```
for images in test_images:
    print(images.shape)
    break
```

```
(512, 160, 160, 1)
```

```
predictions = model.predict(test_images)
```

```
21/21 [==============================] - 15s 738ms/step
```

```
img_list = os.listdir(datasets[1])
```

```
labels_list = sorted(os.listdir(datasets[0]))
print(labels_list)
```

```
['0', '1', '10', '11', '12', '13', '14', '15', '16', '17', '18', '19',
'2', '20', '21', '22', '23', '24', '25', '26', '27', '28', '29', '3',
'30', '31', '32', '33', '34', '35', '36', '37', '38', '39', '4', '40',
'41', '42', '43', '44', '45', '46', '47', '48', '49', '5', '50', '51',
'52', '53', '54', '55', '56', '57', '58', '59', '6', '60', '61', '62',
'63', '64', '7', '8', '9']
```

```
predictions.shape
```

```
(10640, 65)
```

```
df_predictions = pd.DataFrame(columns=['ID','Label'],
dtype=(np.int32,np.int32))

predictions_mod = np.argmax(predictions, 1)

for idx,image in enumerate(sorted(img_list)):
#     print(image, predictions_mod[idx])
    df2 = pd.DataFrame([[int(image.split(".")[0]),
int(labels_list[predictions_mod[idx]])]], columns=['ID','Label'])
    df_predictions = pd.concat([df_predictions, df2])

print(predictions_mod)
```

```
[22 22 22 ... 61 61 61]
```

```
df_predictions.head()
```

```
     ID  Label
0     0     29
0     1     29
0    10     29
0   100     29
0  1000     13
```

```
df_predictions.reset_index(drop=True)
```

```
         ID  Label
0         0     29
1         1     29
2        10     29
3       100     29
4      1000     13
...     ...    ...
10635  9995     64
10636  9996     64
10637  9997     64
10638  9998     64
10639  9999     64

[10640 rows x 2 columns]
```

```python
df_predictions.to_csv('predictions.csv', index=False, header=True)

import numpy as np
from sklearn.metrics import confusion_matrix

# Assuming you have a model and test data
# model = ...  # Your trained model
# test_data = ...  # Your test data

# Make predictions on the test data
predictions = model.predict(test_data)

# Convert predictions and true labels to class indices
predicted_labels = np.argmax(predictions, axis=1)
true_labels = np.argmax(true_labels, axis=1)  # Assuming you have true
labels

# Compute the confusion matrix
conf_matrix = confusion_matrix(true_labels, predicted_labels)

print("Confusion Matrix:")
print(conf_matrix)
```

```
---------------------------------------------------------------------------
-----
NameError                                 Traceback (most recent call
last)
Cell In[80], line 9
      2 from sklearn.metrics import confusion_matrix
      4 # Assuming you have a model and test data
      5 # model = ...  # Your trained model
      6 # test_data = ...  # Your test data
      7
      8 # Make predictions on the test data
----> 9 predictions = model.predict(test_data)
     11 # Convert predictions and true labels to class indices
     12 predicted_labels = np.argmax(predictions, axis=1)

NameError: name 'test_data' is not defined
```