Fayrouz Ahmed 320210131 AID-Section 2

Arabic Handwriting recognition using Neural Networks

Goal?

This project is about using the Convolutional Neural Networks to classify a dataset of images of several handwritten Arabic letters.

The Dataset:

It is composed of two files:

1- Training images
2- Testing images

Criteria used and Code:

1. Importing Libraries

- NumPy and Pandas: For data manipulation.
- TensorFlow and Keras: For building and training the neural network.
- Matplotlib: For plotting images and graphs.

```
import numpy as np
import pandas as pd
import tensorflow as tf
from tensorflow.keras import regularizers, models
from tensorflow.keras.models import Sequential
from tensorflow.keras import layers
from tensorflow.keras.layers import Dense, Conv2D, Flatten, MaxPooling2D, Dropout
from tensorflow.keras.preprocessing.image import ImageDataGenerator
from matplotlib import pyplot as plt
```

2. Setting Model and Parameters

- Defines parameters like number of classes (**num_class**), batch size, image dimensions, and a random seed for reproducibility.

3. Data Preparation

- ImageDataGenerator: Used for real-time data augmentation and rescaling. The validation_split=0.2 indicates 20% of the data is reserved for validation.
- Training and Validation Data Loaders: The code loads training and validation data from a directory, setting image size, batch size, grayscale color mode, categorical class mode, and subsets for training and validation.

[10]:

```
# Model and Data parameters
num_class= 65
batch_size= 64
image_width= 64
image_height= 64
seed= 42

# Rescaling and Testing Splitting
training_datagen= ImageDataGenerator(rescale=1./255, validation_split= 0.2)

training_dir= '/kaggle/input/arabic-letters-classification/Final_Arabic_Alpha_dataset/Final_Arabic_Alpha_da
#testing_dir= '/kaggle/input/arabic-letters-classification/Final_Arabic_Alpha_dataset/Final_Arabic_Alpha_da

# Set as Training data
training_generator = training_datagen.flow_from_directory(
    training_dir,
    target_size= (image_width, image_height),
    batch_size= batch_size,
    seed=seed,
    color_mode='grayscale',
    class_mode= 'categorical',
    subset= 'training')

labels_dict = {class_name: label for label, class_name in training_generator.class_indices.items()}
print(labels_dict)
```

```
Found 34064 images belonging to 65 classes.
{0: '0', 1: '1', 2: '10', 3: '11', 4: '12', 5: '13', 6: '14', 7: '15', 8: '16', 9: '17', 10: '18', 11: '19', 12: '2', 13: '20',
14: '21', 15: '22', 16: '23', 17: '24', 18: '25', 19: '26', 20: '27', 21: '28', 22: '29', 23: '3', 24: '30', 25: '31', 26: '32',
27: '33', 28: '34', 29: '35', 30: '36', 31: '37', 32: '38', 33: '39', 34: '4', 35: '40', 36: '41', 37: '42', 38: '43', 39: '44',
40: '45', 41: '46', 42: '47', 43: '48', 44: '49', 45: '5', 46: '50', 47: '51', 48: '52', 49: '53', 50: '54', 51: '55', 52: '56',
53: '57', 54: '58', 55: '59', 56: '6', 57: '60', 58: '61', 59: '62', 60: '63', 61: '64', 62: '7', 63: '8', 64: '9'}
```

[11]:

```
# Set as validation data
validation_generator = training_datagen.flow_from_directory(
    training_dir,
    target_size=(image_width, image_height),
    batch_size= batch_size,
    seed=seed,
    color_mode='grayscale',
    class_mode='categorical',
    subset='validation')
```

```
Found 8495 images belonging to 65 classes.
```

+ Code + Markdown

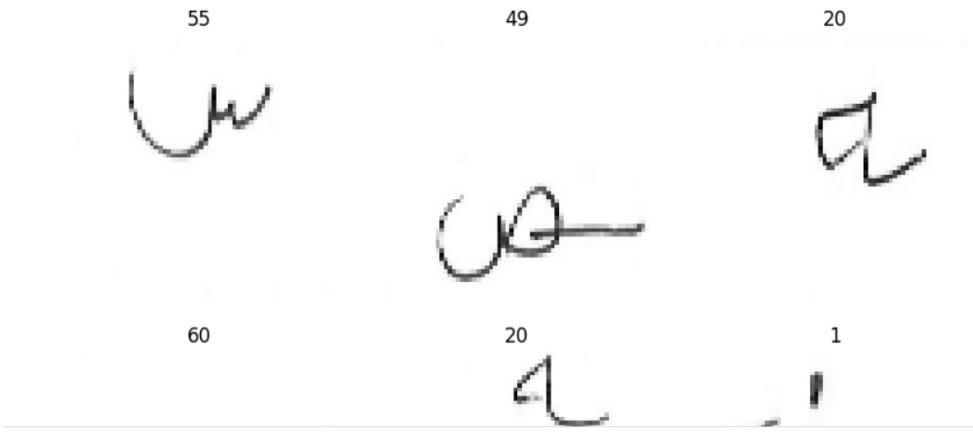
4. Exploring Training Data

- Displaying Images: The code plots a few images from the training set to visually inspect them.
- Label Extraction: Extracts and prints the mapping of class names to their corresponding labels.

[12]:
```
train_images = tf.keras.utils.image_dataset_from_directory(
    training_dir,
    color_mode="grayscale",
    label_mode="categorical",
    shuffle=True,
    seed=seed,
    image_size=(image_width,image_height),
    batch_size=batch_size)
```

```
Found 42559 files belonging to 65 classes.
```

[13]:
```
class_names = train_images.class_names
plt.figure(figsize=(10, 10))
for images, labels in train_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)
        plt.title(class_names[np.where(np.array(labels[i])==1)[0][0]])
        plt.axis("off")
```



5. Building the CNN Model

- Model Architecture: The model consists of convolutional layers (Conv2D) with ReLU activation, followed by max pooling layers (MaxPooling2D). This is followed by a flattening layer (Flatten) and dense layers (Dense), with the final layer having softmax activation for multi-class classification.
- Model Compilation: The model is compiled with the Adam optimizer and categorical crossentropy loss, tracking the accuracy metric.

```
model = Sequential([
    # Convolutional layer with 32 filters, kernel size of 3x3, and ReLU activation
    Conv2D(32, (3, 3), activation='relu'),  # Adjust input shape as needed
    MaxPooling2D(2, 2),
    # Add more convolutional layers as needed
    Conv2D(64, (3, 3), activation='relu'),
    MaxPooling2D(2, 2),
    # Flatten the output to feed into a DNN
    Flatten(),
    # Dense (fully connected) layers
    Dense(128, activation='relu'),
    # Output layer with one neuron per class (adjust number of classes)
    Dense(training_generator.num_classes, activation='softmax')  # Assuming 28 classes for Arabic letters
])
```

[17]:

```
model.build(input_shape=(None,image_width, image_height, 1))
# Model summary
model.summary()
```

```
Model: "sequential_1"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d_2 (Conv2D)           (None, 62, 62, 32)        320

 max_pooling2d_2 (MaxPoolin  (None, 31, 31, 32)        0
 g2D)

 conv2d_3 (Conv2D)           (None, 29, 29, 64)        18496

 max_pooling2d_3 (MaxPoolin  (None, 14, 14, 64)        0
 g2D)

 flatten_1 (Flatten)         (None, 12544)             0

 dense_2 (Dense)             (None, 128)               1605760

 dense_3 (Dense)             (None, 65)                8385

=================================================================
Total params: 1632961 (6.23 MB)
Trainable params: 1632961 (6.23 MB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
```

+ Code + Markdown

[29]:

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

6. Training the Model

- Trains the model on the training data for a specified number of epochs, validating against the validation data.

7. Evaluating the Model

- Evaluates the model's performance on the training data and prints the accuracy.

[244]:
```
epochs = 15
history = model.fit(
    training_generator,
    steps_per_epoch=training_generator.samples // batch_size,
    epochs=epochs,
    validation_data=validation_generator,
    validation_steps=validation_generator.samples
)
```

```
Epoch 1/15
532/532 [==============================] - 75s 140ms/step - loss: 3.3552 - accuracy: 0.1705 - val_loss: 2.3012 - val_accuracy:
0.3907
Epoch 2/15
532/532 [==============================] - 63s 118ms/step - loss: 1.9540 - accuracy: 0.4575
Epoch 3/15
532/532 [==============================] - 62s 116ms/step - loss: 1.4202 - accuracy: 0.5914
Epoch 4/15
532/532 [==============================] - 63s 118ms/step - loss: 1.0802 - accuracy: 0.6810
Epoch 5/15
532/532 [==============================] - 62s 116ms/step - loss: 0.8346 - accuracy: 0.7504
Epoch 6/15
532/532 [==============================] - 62s 116ms/step - loss: 0.6741 - accuracy: 0.7934
Epoch 7/15
532/532 [==============================] - 62s 116ms/step - loss: 0.5383 - accuracy: 0.8339
Epoch 8/15
532/532 [==============================] - 62s 117ms/step - loss: 0.4342 - accuracy: 0.8665
Epoch 9/15
532/532 [==============================] - 61s 115ms/step - loss: 0.3413 - accuracy: 0.8953
Epoch 10/15
532/532 [==============================] - 62s 116ms/step - loss: 0.2723 - accuracy: 0.9174
Epoch 11/15
532/532 [==============================] - 62s 116ms/step - loss: 0.2101 - accuracy: 0.9360
Epoch 12/15
532/532 [==============================] - 63s 118ms/step - loss: 0.1795 - accuracy: 0.9440
Epoch 13/15
532/532 [==============================] - 63s 118ms/step - loss: 0.1339 - accuracy: 0.9601
Epoch 14/15
532/532 [==============================] - 62s 116ms/step - loss: 0.1206 - accuracy: 0.9631
Epoch 15/15
532/532 [==============================] - 62s 117ms/step - loss: 0.1012 - accuracy: 0.9691
```

[245]:
```
test_loss, test_acc = model.evaluate(training_generator, verbose=2)
print('\nTest accuracy:', test_acc)
```

```
533/533 - 45s - loss: 0.0718 - accuracy: 0.9812 - 45s/epoch - 84ms/step

Test accuracy: 0.9812411665916443
```

8. Data Shape Inspection

- Prints the shape of the images and labels in one batch of the training data

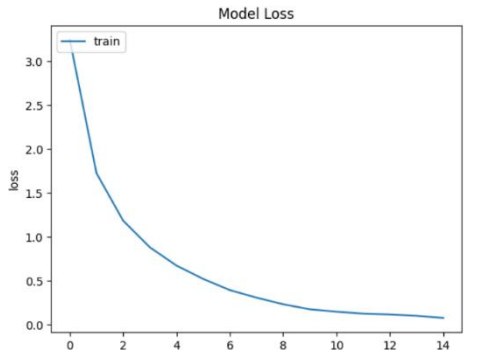
[247]:
```
# (Batch Size, Width, Height, 1 Image Tensor Channel (Greyscale))
# (Batch Size, Number of Classes)
for images, labels in train_images:
    print(images.shape)
    print(labels.shape)
    break
```

```
(64, 64, 64, 1)
(64, 65)
```

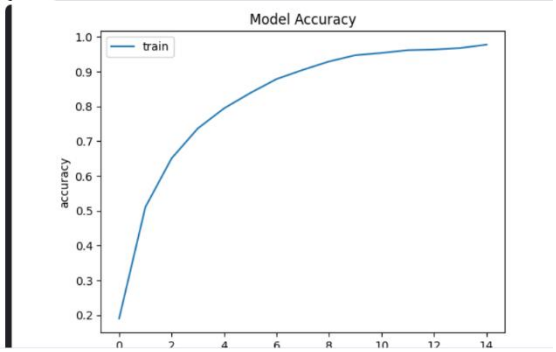
9. Plotting Training and Accuracy

- Plots the training and accuracy over epochs to visualize the learning process.

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




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



10. Preparing Test Data

- Loading Test Data: Loads the test data from a directory, setting appropriate parameters similar to the training data loader.
- Displaying Test Images: Visualizes a few test images.

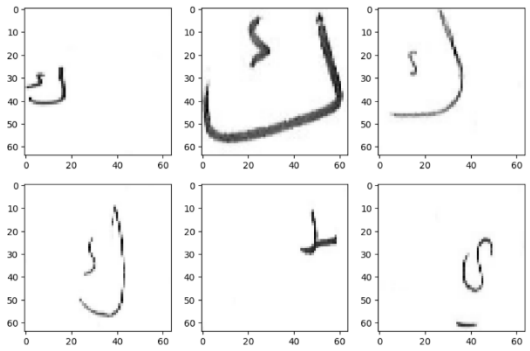
[34]:
```
testing_dir= '/kaggle/input/arabic-letters-classification/Final_Arabic_Alpha_dataset/Final_Arabic_Alpha_dat
```

[35]:
```
test_images = tf.keras.utils.image_dataset_from_directory(
        testing_dir,
        labels=None,
        label_mode="categorical",
        color_mode='grayscale',
        shuffle=False,
        image_size=(image_width,image_height),
        batch_size=batch_size)
```

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

+ Code + Markdown

[36]:
```
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)
```



11. Model Predictions on Test Data

- Uses the trained model to predict classes for the test images.

[37]:
```
for images in test_images:
    print(images.shape)
    break
```

```
(64, 64, 64, 1)
```

[38]:
```
predictions = model.predict(test_images)
```

```
167/167 [==============================] - 13s 77ms/step
```

[43]:
```
img_list = os.listdir(testing_dir)
```

[44]:
```
labels_list = sorted(os.listdir(training_dir))
print(labels_list)
```

```
['0', '1', '10', '11', '12', '13', '14', '15', '16', '17', '18', '19', '2', '20', '21', '22', '23', '24', '25', '26', '27', '2
8', '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']
```

[45]:
```
predictions[1][22]
```

[45]:
```
0.99999994
```

12. Handling Predictions

- Processing Predictions: Converts the model's SoftMax output to class indices.
- Mapping and Storing Predictions: Maps predicted indices to class labels and stores them in a Data Frame.
- Exporting Predictions: The predictions are exported to a CSV file for further use or submission (if this is a competition).

[46]:
```
predictions.shape
```
[46]: (10640, 65)

[48]:
```
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',
    df_predictions = pd.concat([df_predictions, df2])
```

[50]:
```
print(predictions_mod)
```
```
[36 22 22 ... 61 22 36]
```

[51]:
```
df_predictions.head()
```
[51]:

	ID	Label
0	0	41
0	1	29
0	10	29
0	100	29
0	1000	1

[53]:
```
df_predictions.sort_values(by='ID', inplace=True)
```

[54]:
```
df_predictions.reset_index(drop=True)
```
[54]:

	ID	Label
0	0	41
1	1	29
2	2	29
3	3	29
4	4	12
...	...	...
10635	10635	10
10636	10636	61
10637	10637	61
10638	10638	61
10639	10639	61

10640 rows × 2 columns

[56]:
```
df_predictions.to_csv('predictions.csv', index=False, header=True)
```