PRACTICAL 10

Recognising Handwritten Dzongkha Digits Using Keras - Saving and Restoring Models

ACTIVITY - Submission Required!

In this exercise, we will be developing a model for the recognition of Dzongkha handwritten digits. We will be using **ImageDataGenerator** to load the digits from Google Drive and split them into train and validation sets. After training the model, we will save, load and make predictions on the new data. For this exercise, only 3 classes (zero, one and, two) will be used.

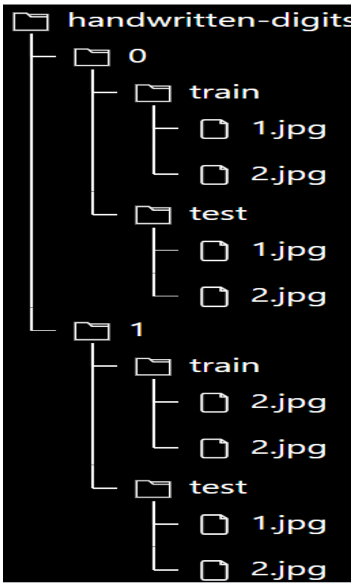
NOTE:

The dataset can be split by following either of the two methods:

1. Upload the complete training images in Google Drive and use **ImageDataGenerator** to split them **(RECOMMENDED)**

```
handwritten-digits
├─ 0
│  ├─ 1.jpg
│  ├─ 2.jpg
│  └─ 3.jpg
└─ 1
   ├─ 1.jpg
   └─ 2.jpg
```

2. Split the images into train and validation sets (manually or write a program) and then upload them to Google Drive.



For the second step, you can use the split-folders library which splits the dataset into **train, validation** and **test** folders**.**

In this tutorial, we will be using the first step to split the dataset into train and validation sets.

Reminder: Use the **Dzo_MNIST** dataset from Week 8 practical.

Perform the following steps to complete this exercise:

1. Open a new Jupyter Notebook file and save it inside Google Drive (your working directory)
2. Mount the Google Drive
3. Import the required libraries:

```
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import BatchNormalization,
Conv2D, MaxPooling2D, Activation, Flatten, Dropout, Dense
from tensorflow.keras.preprocessing.image import
ImageDataGenerator
```

4. Create a train and validation data generator.

```
train_datagen = ImageDataGenerator(
    rescale = 1./255,
    shear_range = 0.1,
    rotation_range = 10,
    zoom_range = 0.1,
    width_shift_range = 0.1,
    height_shift_range = 0.1,
    fill_mode = "nearest",
```

```
    validation_split = 0.2
)

val_datagen = ImageDataGenerator(rescale=1./255,
validation_split =  0.2)
```

5. Now, load the digits from Google Drive and split them into train and validation sets.

```
train_generator = train_datagen.flow_from_directory(
    path_to_Dzo_MNIST,
    target_size = (28, 28),
    batch_size = 32,
    classes = ['0', '1', '2'],
    class_mode = 'categorical',
    color_mode = 'grayscale',
    subset = 'training') # set as training data

validation_generator = val_datagen.flow_from_directory(
    path_to_Dzo_MNIST, # same directory as training data
    target_size = (28, 28),
    batch_size=32,
    classes = ['0', '1', '2'],
    class_mode='categorical',
    color_mode = 'grayscale',
    subset = 'validation') # set as validation data
```

NOTE: Here, you have to edit path_to_Dzo_MNIST

6. Print summary of the data:

```
print("[INFO]- Training Set")
print("Number of samples in train set: ",
train_generator.samples)
print("Number of classes in test set: ",
len(train_generator.class_indices))
print("Number of samples per class[train-set]: ",
int(train_generator.samples /
len(train_generator.class_indices)))
print("****************************************")

print("\n[INFO]- Validation Set")
print("Number of samples in validation set: ",
validation_generator.samples)
```

```
print("Number of classes in validation set: ",
len(validation_generator.class_indices))
print("Number of samples per class[validation-set]: ",
int(validation_generator.samples /
len(validation_generator.class_indices)))
print("************************************")
```

7. Instantiate a **Sequential()** class into a variable called **model** with and add the following layers:
    a. A first convolution layer with **32** kernels of shape **3**, ReLU as the activation function.
    b. A second convolution layer with **32** kernels of shape 3, ReLU as the activation function, a max pooling layer with size 2 by 2 and dropout of rate 0.25.
    c. A third convolution layer with **64** kernels of shape, ReLU as the activation function, a max pooling layer with size 2 by 2 and dropout of rate 0.25, and a flattened layer.
    d. Add a fully connected layer with **512** neurons and ReLU as the activation function, a batch norm.
    e. Add a dropout layer with a rate of **0.5** and an output layer with 3 neurons.
    f. Use softmax as the activation function.

The summary of the model should be the same as below:

```
_________________________________________________________________
 Layer (type)                Output Shape              Param #   
=================================================================
 conv2d_8 (Conv2D)           (None, 28, 28, 32)        320       
                                                                 
 activation_11 (Activation)  (None, 28, 28, 32)        0         
                                                                 
 conv2d_9 (Conv2D)           (None, 28, 28, 32)        9248      
                                                                 
 activation_12 (Activation)  (None, 28, 28, 32)        0         
                                                                 
 max_pooling2d_4 (MaxPooling  (None, 14, 14, 32)       0         
 2D)                                                             
                                                                 
 dropout_6 (Dropout)         (None, 14, 14, 32)        0         
                                                                 
 conv2d_10 (Conv2D)          (None, 14, 14, 64)        18496     
                                                                 
 activation_13 (Activation)  (None, 14, 14, 64)        0         
                                                                 
 max_pooling2d_5 (MaxPooling  (None, 7, 7, 64)         0         
 2D)                                                             
                                                                 
 dropout_7 (Dropout)         (None, 7, 7, 64)          0         
                                                                 
 flatten_2 (Flatten)         (None, 3136)              0         
                                                                 
 dense_4 (Dense)             (None, 512)               1606144   
                                                                 
 activation_14 (Activation)  (None, 512)               0         
                                                                 
 batch_normalization_2 (Batc  (None, 512)              2048      
 hNormalization)                                                 
                                                                 
 dropout_8 (Dropout)         (None, 512)               0         
                                                                 
 dense_5 (Dense)             (None, 3)                 1539      
                                                                 
 activation_15 (Activation)  (None, 3)                 0         
                                                                 
=================================================================
```

8. Compile the model with **categorical_crossentropy** as loss, **adam** as the optimizer and **accuracy** as the metric.
9. Fit the neural networks.

```
H = model.fit(
    train_generator,
    steps_per_epoch = len(train_generator),
    validation_data = validation_generator,
    validation_steps = len(validation_generator),
    epochs = 10
)
```

More on **fit()**: https://keras.io/api/models/model_training_apis/

10. Let's evaluate the performance of the model on the training and validation set.

```
# Train accuracy
```

```
scores = model.evaluate(train_generator,
steps=len(train_generator), verbose=1)
print("Train Accuracy: %.2f%%" % (scores[1]*100))

# Validation accuracy
scores = model.evaluate(validation_generator,
steps=len(validation_generator), verbose=1)
print("Validation (Seen) Accuracy: %.2f%%" % (scores[1]*100))
```

Activity

1. Write the code to save the entire model, architecture and weights only.
2. Using the saved model, write a program to predict the new data. Click here to download the sample handwritten digits.
3. Explain what is happening in step 9. You need to explain the code line by line.

****