PRACTICAL 7

Recognizing Handwritten Digits (MNIST) Using Keras

ACTIVITY - Submission Required!

In this exercise, we will be working on the MNIST which contains images of handwritten digits. This dataset was originally shared by Yann Lecun, one of the most renowned deep-learning researchers. We will build a CNN model and then train it to recognize handwritten digits.

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. Import the MNIST dataset from the TensorFlow API:

```
import tensorflow.keras.datasets.mnist as mnist
```

3. Load the mnist dataset using **mnist.load_data()** and save the results into **(features_train, label_train), (features_test, label_test)**:

```
(features_train, label_train), (features_test, label_test) = mnist.l
oad_data()
```

4. Reshape the training and testing sets with the dimensions **(number_observations, 28, 28, 1**):

```
features_train = features_train.reshape(60000, 28, 28, 1)
features_test = features_test.reshape(10000, 28, 28, 1)
```

5. Standardize **features_train** and **features_test** by dividing them by **255**.

```
features_train = features_train / 255.0
features_test = features_test / 255.0
```

6. Import **numpy as np**, **tensorflow as tf**, and layers from **tensorflow.keras**:

```
import numpy as np
import tensorflow as tf
from tensorflow.keras import layers
```

7. Set 8 as the seed for numpy and tensorflow.

```
np.random.seed(8)
tf.random.set_seed(8)
```

***NOTE**: The results may still differ slightly after setting the seeds.*

8. Instantiate a **tf.keras.Sequential()** class and save it to a variable called model:

```
model = tf.keras.Sequential()
```

9. Instantiate a **layers.Conv2D()** class with **64** kernels of **shape (3, 3)**, **activation='relu'**, and **input_shape=(28, 28, 1)**, and save it to a variable called **conv_layer1**:

```
conv_layer1 = layers.Conv2D(64, (3,3), activation='relu', input_shape=(28, 28, 1))
```

10. Instantiate a **layers.Conv2D()** class with 64 kernels of **shape (3,3)** and **activation='relu'** and save it to a variable called **conv_layer2.**

***NOTE**: It is only required to specify the input_shape parameter for the first layer. For the following layers, CNN would infer it automatically.*

```
conv_layer2 = layers.Conv2D(64, (3, 3), activation='relu')
```

11. Instantiate a **layers.Flatten()** class with **128** neurons, **activation='relu'**, and save it to a variable called **fc_layer1**:

```
fc_layer1 = layers.Dense(128, activation='relu')
```

12. Instantiate a **layers.Flatten()** class with **10** neurons, **activation='softmax'**, and save it to a variable called **fc_layer2**:

```
fc_layer2 = layers.Dense(10, activation='softmax')
```

13. Add the four layers you just defined to the model using **.add()**, add a **MaxPooling2D()** layer of size **(2,2)** in between each of the convolution layers, and add a **Flatten()** layer before the first fully connected layer to flatten the feature maps:

```
model.add(conv_layer1)
model.add(layers.MaxPooling2D(2, 2))
model.add(conv_layer2)
model.add(layers.MaxPooling2D(2, 2))
model.add(layers.Flatten())
model.add(fc_layer1)
model.add(fc_layer2)
```

14. Instantiate a **tf.keras.optimizers.Adam()** class with **0.001** as the learning rate and save it to a variable called **optimizer**:

```
optimizer = tf.keras.optimizers.Adam(0.001)
```

15. Compile the neural network using **.compile()** with **loss='sparse_categorical_crossentropy', optimizer=optimizer, metrics=['accuracy']:**

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

16. Print the summary of the model.

```
model.summary()
```

```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d (Conv2D)              (None, 26, 26, 64)        640
_________________________________________________________________
max_pooling2d (MaxPooling2D) (None, 13, 13, 64)        0
_________________________________________________________________
conv2d_1 (Conv2D)            (None, 11, 11, 64)        36928
_________________________________________________________________
max_pooling2d_1 (MaxPooling2 (None, 5, 5, 64)          0
_________________________________________________________________
flatten (Flatten)            (None, 1600)              0
_________________________________________________________________
dense (Dense)                (None, 128)               204928
_________________________________________________________________
dense_1 (Dense)              (None, 10)                1290
=================================================================
Total params: 243,786
Trainable params: 243,786
Non-trainable params: 0
_________________________________________________________________
```

17. Fit the neural networks with the training set and specify **epochs=5**, **validation_split=0.2**, and **verbose=2**:

```
model.fit(features_train, label_train, epochs=5, validation_split=0.2, verbose=2)
```

The expected output will be as follows:

```
Epoch 1/5
1500/1500 - 69s - loss: 0.1365 - accuracy: 0.9574 - val_loss: 0.0562 - val_accuracy: 0.9842 - 69s/epoch - 46ms/step
Epoch 2/5
1500/1500 - 67s - loss: 0.0436 - accuracy: 0.9868 - val_loss: 0.0408 - val_accuracy: 0.9876 - 67s/epoch - 45ms/step
Epoch 3/5
1500/1500 - 66s - loss: 0.0295 - accuracy: 0.9907 - val_loss: 0.0333 - val_accuracy: 0.9908 - 66s/epoch - 44ms/step
Epoch 4/5
1500/1500 - 66s - loss: 0.0211 - accuracy: 0.9931 - val_loss: 0.0346 - val_accuracy: 0.9902 - 66s/epoch - 44ms/step
Epoch 5/5
1500/1500 - 66s - loss: 0.0154 - accuracy: 0.9952 - val_loss: 0.0339 - val_accuracy: 0.9912 - 66s/epoch - 44ms/step
<keras.callbacks.History at 0x7f2657504490>
```

We trained our CNN on 48,000 samples, and we used 12,000 samples as the validation set. After training for five epochs, we achieved an accuracy score of **0.9952** for the training set and **0.991**2 for the validation set. Our model is overfitting a bit.

18. Let's evaluate the performance of the model on the testing set:

```
model.evaluate(features_test, label_test)
```

The expected output will be as follows:

```
313/313 [==============================] - 4s 14ms/step - loss: 0.0325 -
accuracy: 0.9902
[0.03251089155673981, 0.9901999831199646]
```

With this, we've achieved an accuracy score of 0.9902 on the testing set.

Activity

1. Explain the role of **features_train, label_train, features_test, label_test** in the training process.
2. Retrain the model by changing the validation_size to **0.3**.
3. Retrain the model with two more different epochs and notice the model performance.
4. Display the shape of the training and testing set.

****