CNN models for detecting handwritten digits, using MNIST

June 9, 2023

Name: Trịnh Lê Nguyên Vũ

ID : 20120630

1 Installing Dependencies and Libraries

This project will be using Keras with Tensorflow backend to build the CNN models.

[ ]:
```
# Install tensorflow
# This might takes time
%pip install tensorflow
```

[ ]:
```
# Install neccessary dependencies
%pip install numpy scipy
%pip install scikit-learn
%pip install pillow
%pip install h5py
%pip install matplotlib
%pip install opencv-python
```

[ ]:
```
# Install keras
%pip install keras
```

2 Importing Libraries

[ ]:
```
# Create environment
import os
os.environ["TF_CPP_MIN_LOG_LEVEL"] = "2"
```

[ ]:
```
# Import dependencies
import numpy as np
import keras
import tensorflow as tf
import matplotlib.pyplot as plt
%matplotlib inline
import tensorflow.keras.layers as KL
import tensorflow.keras.models as KM
```

3 Establising Datasets

```
[ ]: # Establish datasets
mnist = tf.keras.datasets.mnist
```

```
[ ]: # Load training and testing datasets
# This might takes some times to download

(x_train, y_train), (x_test, y_test) = mnist.load_data()
```

```
[ ]: # Check datasets dimensions:
print(x_train.shape, y_train.shape)
print(x_test.shape, y_test.shape)
```

```
(60000, 28, 28) (60000,)
(10000, 28, 28) (10000,)
```

4 Displaying sample images

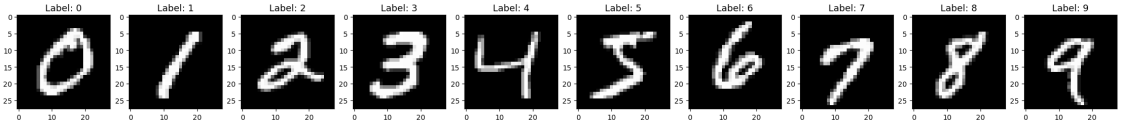
These lines of codes below will be displaying images from the MNIST datasets, as well as the label they should be labeled with

If the codes run properly, this image below should be the result

```
[ ]: # Show n image(s)
n = 10

f, ax = plt.subplots(1, n, figsize=(28,28))

# Display each image
for i in range(0, n):
    sample = x_train[y_train == i][0]
    ax[i].imshow(sample, cmap='gray')
    ax[i].set_title("Label: {}".format(i), fontsize=13)
```



5 Preparing Data

5.1 Normalizing Data

Here we define our normalizing function. The goal is to compress the values of the datasets to the range between 0 and 1

Without normalizing the data, it will be impossible for the network to learn properly

[ ]:
```
# Define normalize function
def normalize(x):
    return x / 255.0
```

[ ]:
```
# Normalize values
x_train, x_test = normalize(x_train), normalize(x_test)
```

5.2 Reshaping the data

The input data is originally formatted to be a 28x28 matrix

[ ]:
```
# This represented the 3-dimensional array that holds all the 28x28 input␣
↪images.
print(x_train.shape, x_test.shape)
```

```
(60000, 28, 28) (10000, 28, 28)
```

[ ]:
```
# This is the dimensions of a single image, presented in the form of a 28x28␣
↪matrix
print(x_train[0].shape)
```

```
(28, 28)
```

For the sake of simpler operation, we make the 2-d matrix become a 3-d one, effectively turning it into a layer, with the width of 1

[ ]:
```
# Reshaping
x_train, x_test = np.expand_dims(x_train, axis = -1), np.expand_dims(x_test,␣
↪axis = -1)
```

After getting reshaped into a layer, the new shape should be (28x28x1). These two blocks of code will demonstrate the result

[ ]:
```
print(x_train.shape, x_test.shape)
```

```
(60000, 28, 28, 1) (10000, 28, 28, 1)
```

[ ]:
```
print(x_train[0].shape)
```

```
(28, 28, 1)
```

6 Creating the Model

The first step is to define the model's structure and the filters (the convolution kernel)

[ ]:
```
model = KM.Sequential()

model.add(KL.Conv2D(32, kernel_size = (3, 3), activation = 'relu', input_shape␣
↪= (28, 28, 1), padding="valid"))
```

```
model.add(KL.Conv2D(64, kernel_size = (3, 3), activation = 'relu',␣
 ↪padding="valid"))
model.add(KL.MaxPooling2D(pool_size = (2, 2)))
model.add(KL.Dropout(0.5))

model.add(KL.Conv2D(128, kernel_size = (3, 3), activation = 'relu',␣
 ↪padding="valid"))
model.add(KL.MaxPooling2D(pool_size = (2, 2)))
model.add(KL.Flatten())
model.add(KL.Dense(10, activation = 'softmax'))
```

If the code run properly, this will be the result of the block of code below

[ ]:
```
# Visualizing the model
model.summary()
```

```
Model: "sequential_2"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d_6 (Conv2D)           (None, 26, 26, 32)        320

 conv2d_7 (Conv2D)           (None, 24, 24, 64)        18496

 max_pooling2d_4 (MaxPooling  (None, 12, 12, 64)       0
 2D)

 dropout_2 (Dropout)         (None, 12, 12, 64)        0

 conv2d_8 (Conv2D)           (None, 10, 10, 128)       73856

 max_pooling2d_5 (MaxPooling  (None, 5, 5, 128)        0
 2D)

 flatten_2 (Flatten)         (None, 3200)              0

 dense_2 (Dense)             (None, 10)                32010

=================================================================
Total params: 124,682
Trainable params: 124,682
Non-trainable params: 0
_________________________________________________________________
```

The next step is to compile the model with the selected optimizer, loss function and metrics

In this particular example, the ‘adam’ optimizer, the ‘sparse_categorical_crossentropy’ loss function, and the ‘accuracy’ metrics

[ ]:
```
model.compile(loss = "sparse_categorical_crossentropy", optimizer = "adam",␣
↪metrics = ['accuracy'])
```

7 Training the Model

The training is done using the fit() method

[ ]:
```
model.fit(x_train, y_train, epochs = 5, validation_data=(x_test, y_test))
```

```
Epoch 1/5
1875/1875 [==============================] - 56s 30ms/step - loss: 0.1396 -
accuracy: 0.9568 - val_loss: 0.0435 - val_accuracy: 0.9866
Epoch 2/5
1875/1875 [==============================] - 55s 29ms/step - loss: 0.0473 -
accuracy: 0.9845 - val_loss: 0.0299 - val_accuracy: 0.9906
Epoch 3/5
1875/1875 [==============================] - 55s 29ms/step - loss: 0.0362 -
accuracy: 0.9888 - val_loss: 0.0341 - val_accuracy: 0.9892
Epoch 4/5
1875/1875 [==============================] - 57s 30ms/step - loss: 0.0314 -
accuracy: 0.9898 - val_loss: 0.0278 - val_accuracy: 0.9921
Epoch 5/5
1875/1875 [==============================] - 55s 29ms/step - loss: 0.0280 -
accuracy: 0.9918 - val_loss: 0.0317 - val_accuracy: 0.9893
```

[ ]:
```
<keras.callbacks.History at 0x238c7e93370>
```

8 Evaluate the model

The evaluation is done using the evaluate() method

The result is shown by executing the block of codes below

[ ]:
```
loss, accuracy = model.evaluate(x_test, y_test, verbose = 0)

print('Test loss:', loss)
print('Test accuracy:', accuracy)
```

```
Test loss: 0.031728871166706085
Test accuracy: 0.989300012588501
```

9 Predict

First, copy the output data and reshaping it

[ ]:
```
y_test_copy = keras.utils.to_categorical(y_test, 10)
```

[ ]:
```
pred = model.predict(x_test)
pred = np.argmax(pred, axis = 1)[:10]
label = np.argmax(y_test_copy,axis = 1)[:10]

print(pred)
print(label)
```

```
313/313 [==============================] - 3s 10ms/step
[7 2 1 0 4 1 4 9 5 9]
[7 2 1 0 4 1 4 9 5 9]
```

If the code run properly, both array should be identical, indicating that the model correctly predicts the first 10 images.

10 References

Keras installation with Tensorflow backend: https://pyimagesearch.com/2016/11/14/installing-keras-with-tensorflow-backend/

CNN using Keras: https://www.tutorialspoint.com/keras/keras_convolution_neural_network.htm

Jupyter notebook bash commands: https://docs.jupyter.org/en/latest/running.html

Jupyter Notebook markdown cheatsheet: https://www.ibm.com/docs/en/watson-studio-local/1.2.3?topic=notebooks-markdown-jupyter-cheatsheet