Classify Handwritten-Digits with TensorFlow

MNIST database:

The MNIST database (Modified National Institute of Standards and Technology database) is a large database of handwritten digits that is commonly used for training various image processing systems. The database is also widely used for training and testing in the field of machine learning. It was created by "re-mixing" the samples from NIST's original datasets.

THE MNIST HANDWRITTEN DIGIT IS A DATASET FOR EVALUATING MACHINE LEARNING AND DEEP LEARNING MODELS ON THE HANDWRITTEN DIGIT CLASSIFICATION PROBLEM, IT IS A DATASET OF 60,000 SMALL SQUARE 28×28 PIXEL GRAYSCALE IMAGES OF HANDWRITTEN SINGLE DIGITS BETWEEN 0 AND 9.

Convolutional Neural Network:

A convolutional neural network (CNN or ConvNet), is a network architecture for deep learning which learns directly from data, eliminating the need for manual feature extraction. CNNs are particularly useful for finding patterns in images to recognize objects, faces, and scenes. They canalso be quite effective for classifying non-image data such as audio, time series, and signal data. Applications that call for object recognition and computer vision — such as self-driving vehicles and face-recognition applications — rely heavily on CNNs.

Importing the libraries

```
import tensorflow as tf # Import tensorflow library

import matplotlib.pyplot as plt # Import matplotlib library
```

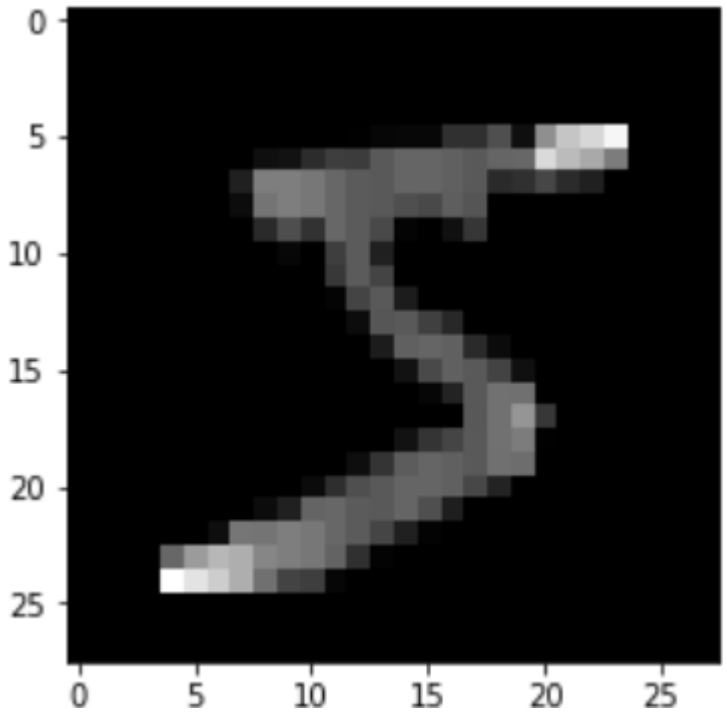
Create a variable named mnist, and set it to an object of the MNIST dataset from the Keras library and we’re gonna unpack it to a training dataset (x_train, y_train) and testing dataset (x_test, y_test):

```
mnist = tf.keras.datasets.mnist # Object of the MNIST dataset

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

Preprocessing the data

To make sure that our data was imported correctly, we are going to plot the first image from the training dataset using matplotlib:

```
plt.imshow(x_train[0], cmap="gray") # Import the image

plt.show() # Plot the image
```



Before we feed the data into the neural network, we need to normalize it by scaling the pixels value in a range from 0 to 1 instead of being from 0 to 255 and that make the neural network needs less computational power:

```
# Normalize the train dataset

x_train = tf.keras.utils.normalize(x_train, axis=1)

# Normalize the test dataset

x_test = tf.keras.utils.normalize(x_test, axis=1)
```

Build the model

Now, we are going to build the model or in other words the neural network that will train and learn how to classify these images. It worth noting that the layers are the most important thing in building an artificial neural network since it will extract the features of the data.

First and foremost, we start by creating a model object that lets you add the different layers.

Second, we are going to flatten the data which is the image pixels in this case. So, the images are 28×28 dimensional we need to make it 1×784 dimensional so the input layer of the neural network can read it or deal with it. This is an important concept you need to know.

Third, we define input and a hidden layer with 128 neurons and an activation function which is the relu function.

And the Last thing we create the output layer with 10 neurons and a SoftMax activation function that will transform the score returned by the model to a value so it will be interpreted by humans.

```
#Build the model object

model = tf.keras.models.Sequential()

# Add the Flatten Layer

model.add(tf.keras.layers.Flatten())

# Build the input and the hidden layers

model.add(tf.keras.layers.Dense(128, activation=tf.nn.relu))

model.add(tf.keras.layers.Dense(128, activation=tf.nn.relu))

# Build the output layer

model.add(tf.keras.layers.Dense(10, activation=tf.nn.softmax))
```

Compile the model

Since we finished building the neural network, we need to compile the model by adding some few parameters that will tell the neural network how to start the training process.

First, we add the optimizer which will create or in other word update the parameter of the neural network to fit our data. Second, the loss function that will tell you the performance of your model. Third, the Metrics which give indicative tests of the quality of the model.

```
# Compile the model

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

Train the model

We are ready to train our model, we call the fit subpackage and feed it with the training data and the labeled data that correspond to the training dataset and how many epoch should run or how many times should make a guess.

```
model.fit(x=x_train, y=y_train, epochs=5) # Start training process
```

```
Epoch 1/5
60000/60000 [==============================] - 7s 111us/sample - loss: 0.2581 - acc: 0.9241
Epoch 2/5
60000/60000 [==============================] - 6s 100us/sample - loss: 0.1064 - acc: 0.9680
Epoch 3/5
60000/60000 [==============================] - 6s 99us/sample - loss: 0.0728 - acc: 0.9771
Epoch 4/5
60000/60000 [==============================] - 6s 99us/sample - loss: 0.0533 - acc: 0.9827
Epoch 5/5
60000/60000 [==============================] - 6s 101us/sample - loss: 0.0397 - acc: 0.9869
<tensorflow.python.keras.callbacks.History at 0x7f93bd580fd0>
```

Training Process

Evaluate the model

Let's see how the model performs after the training process has finished.

```
# Evaluate the model performance

test_loss, test_acc = model.evaluate(x=x_test, y=y_test)

# Print out the model accuracy

print('\nTest accuracy:', test_acc)
```

```
10000/10000 [==============================] - 1s 66us/sample - loss: 0.0900 - acc: 0.9739

Test accuracy: 0.9739
```

Evaluating the Model Performance

It shows that the neural network has reached 97.39% accuracy which is pretty good since we train the model just with 5 epochs.

Make predictions

Now, we will start making a prediction by importing the test dataset images.

```
predictions = model.predict([x_test]) # Make prediction
```

We are going to make a prediction for numbers or images that the model has never seen before. For instance, we try to predict the number that corresponds to the image number 1000 in the test dataset:

```
print(np.argmax(predictions[1000])) # Print out the number
```

```
print(np.argmax(predictions[1000])) # Print out the number
```

```
9
```

Prediction

As you see, the prediction is number nine but how we can make sure that this prediction was true? well, we need to plot the image number 1000 in the test dataset using matplotlib:

```
plt.imshow(x_test[1000], cmap="gray") # Import the image

plt.show() # Show the image
```