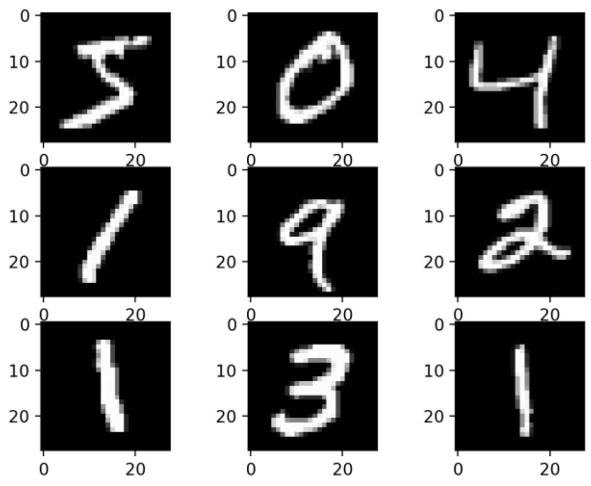
Lab.5

HCIA-AI TensorFlow Programming Basics

1. Build a neural network that classifies images.
2. Train this neural network.
3. And, finally, evaluate the accuracy of the model.



1. Download and install TensorFlow 2. Import TensorFlow into your program:

```
import tensorflow as tf
```

2. Load and prepare the MNIST dataset. Convert the samples from integers to floating-point numbers:

```
mnist = tf.keras.datasets.mnist

(x_train, y_train), (x_test, y_test) = mnist.load_data()
x_train, x_test = x_train / 255.0, x_test / 255.0
```

3. Build the tf.keras.Sequential model by stacking layers. Choose an optimizer and loss function for training:

```
model = tf.keras.models.Sequential([
  tf.keras.layers.Flatten(input_shape=(28, 28)),
  tf.keras.layers.Dense(128, activation='relu'),
  tf.keras.layers.Dropout(0.2),
  tf.keras.layers.Dense(10)
])
```

4. For each example the model returns a vector of "logits" or "log-odds" scores, one for each class.

```
predictions = model(x_train[:1]).numpy()
predictions
```

5. **The tf.nn.softmax function converts these logits to "probabilities" for each class:**

```
tf.nn.softmax(predictions).numpy()
```

Note: It is possible to bake this `tf.nn.softmax` in as the activation function for the last layer of the network. While this can make the model output more directly interpretable, this approach is discouraged as it's impossible to provide an exact and numerically stable loss calculation for all models when using a softmax output.

6. **The losses.SparseCategoricalCrossentropy loss takes a vector of logits and a True index and returns a scalar loss for each example. (Calculate loss)**

```
loss_fn = tf.keras.losses.SparseCategoricalCrossentropy(from_logits=True)
```

This loss is equal to the negative log probability of the true class: It is zero if the model is sure of the correct class.

This untrained model gives probabilities close to random (1/10 for each class), so the initial loss should be close to -tf.log(1/10) ~= 2.3.

```
loss_fn(y_train[:1], predictions).numpy()
```

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

The Model.fit method adjusts the model parameters to minimize the loss:

```
model.fit(x_train, y_train, epochs=5)
```

```
Epoch 1/5
1875/1875 [==============================] - 3s 2ms/step - loss: 0.2944 - accuracy: 0.9136
Epoch 2/5
1875/1875 [==============================] - 3s 2ms/step - loss: 0.1415 - accuracy: 0.9586
Epoch 3/5
1875/1875 [==============================] - 3s 2ms/step - loss: 0.1066 - accuracy: 0.9677
Epoch 4/5
1875/1875 [==============================] - 3s 2ms/step - loss: 0.0864 - accuracy: 0.9734
Epoch 5/5
1875/1875 [==============================] - 3s 2ms/step - loss: 0.0735 - accuracy: 0.9770

<tensorflow.python.keras.callbacks.History at 0x7f2b9e9fe438>
```

The Model.evaluate method checks the models performance, usually on a "Validation-set" or "Test-set".

```
model.evaluate(x_test,  y_test, verbose=2)
```

The image classifier is now trained to ~98% accuracy on this dataset.