Experiment report: TensorFlow

Done by student: **Bylkova Kristina (伯汀娜), 1820249049**

Course: **Big Data Analysis Technology**

Date: **October 2024**

Summary

This report shows the MindMap of TensorFlow and NeuronNetworks and also the completed experiment.

MindMap

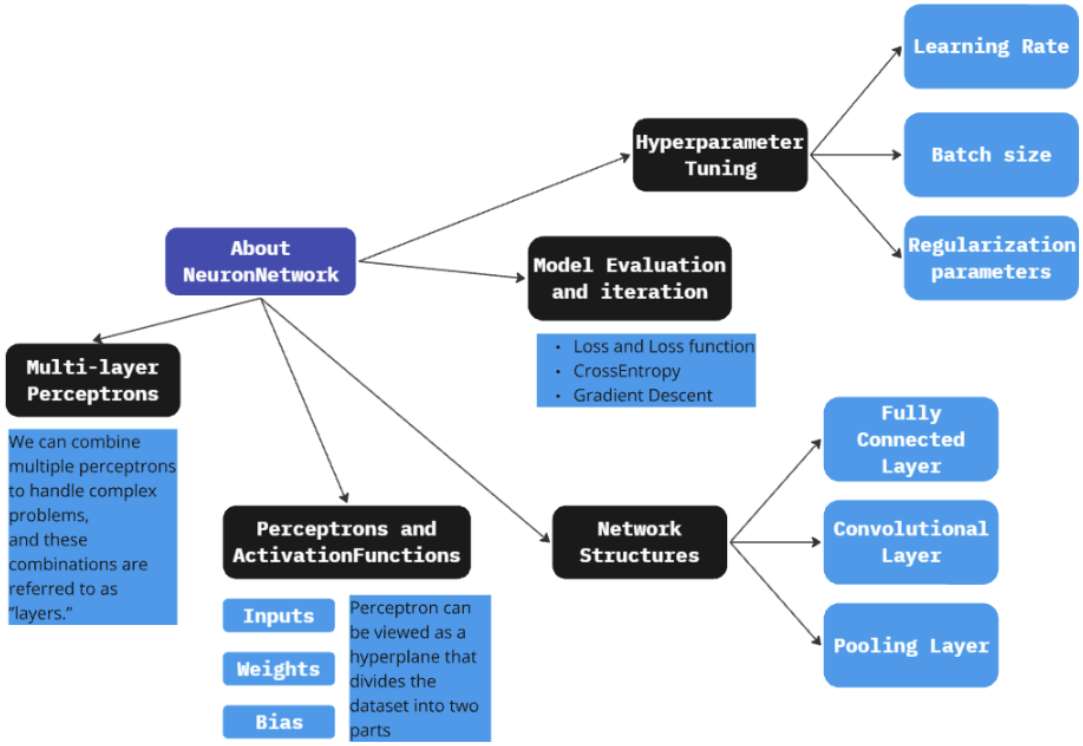

Figure 1: About NeuronNetworks

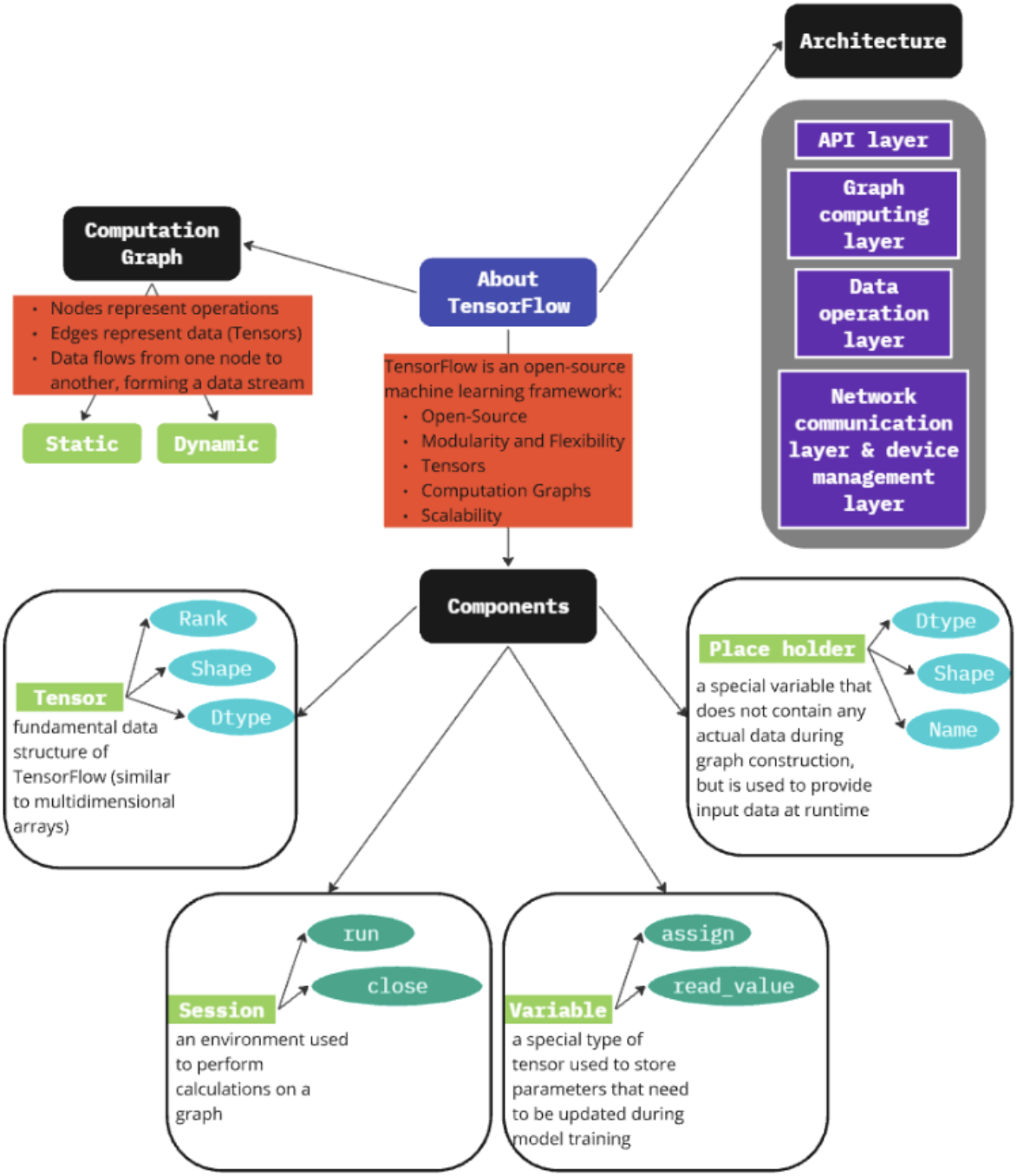

Figure 2: About TensorFlow

Basic Experiments

Experiment 0

The purpose of this experiment is training a neural network using the MNIST dataset with Keras. Here I ran the provided code to train a model, monitoring accuracy and loss.

```python
import tensorflow as tf

(x_train, y_train), (x_test, y_test) = (
    tf.keras.datasets.mnist.load_data())

x_train = x_train.astype("float32") / 255.0
x_test = x_test.astype("float32") / 255.0

x_train = x_train.reshape(-1, 28 * 28)
x_test = x_test.reshape(-1, 28 * 28)

# Step 1: Model Building using Keras Sequential API
model = tf.keras.models.Sequential([
    tf.keras.layers.InputLayer(input_shape=(784,)),
    tf.keras.layers.Dense(128, activation='relu'),
    # Neuron Count in Hidden Layer : : 128
    tf.keras.layers.Dense(10)
])

# Step 2: Model Compilation
model.compile(optimizer=tf.keras.optimizers.Adam(learning_rate=0.001),
              # learning rate : 0.001
              loss=tf.keras.losses.SparseCategoricalCrossentropy(from_logits=True),
              metrics=['accuracy'])

# Step 3: Model Training
history = model.fit(x_train, y_train, epochs=10, batch_size=32)
# number of epochs : 10, batch size : 32

# Step 4: Output Training Accuracy and Loss
for epoch in range(10):
    epoch_acc = history.history['accuracy'][epoch]
    epoch_loss = history.history['loss'][epoch]
    print(f'Epoch {epoch + 1}: Training Accuracy = '
          f'{epoch_acc:.4f}, Loss = {epoch_loss:.4f}')

# Step 5: Model Evaluation
test_loss, test_acc = model.evaluate(x_test, y_test)
print(f'Test loss: {test_loss:.4f}')
print(f'Test accuracy: {test_acc * 100:.2f}%')

# Step 6: Model Saving
model.save("keras_mnist_model.keras")
```

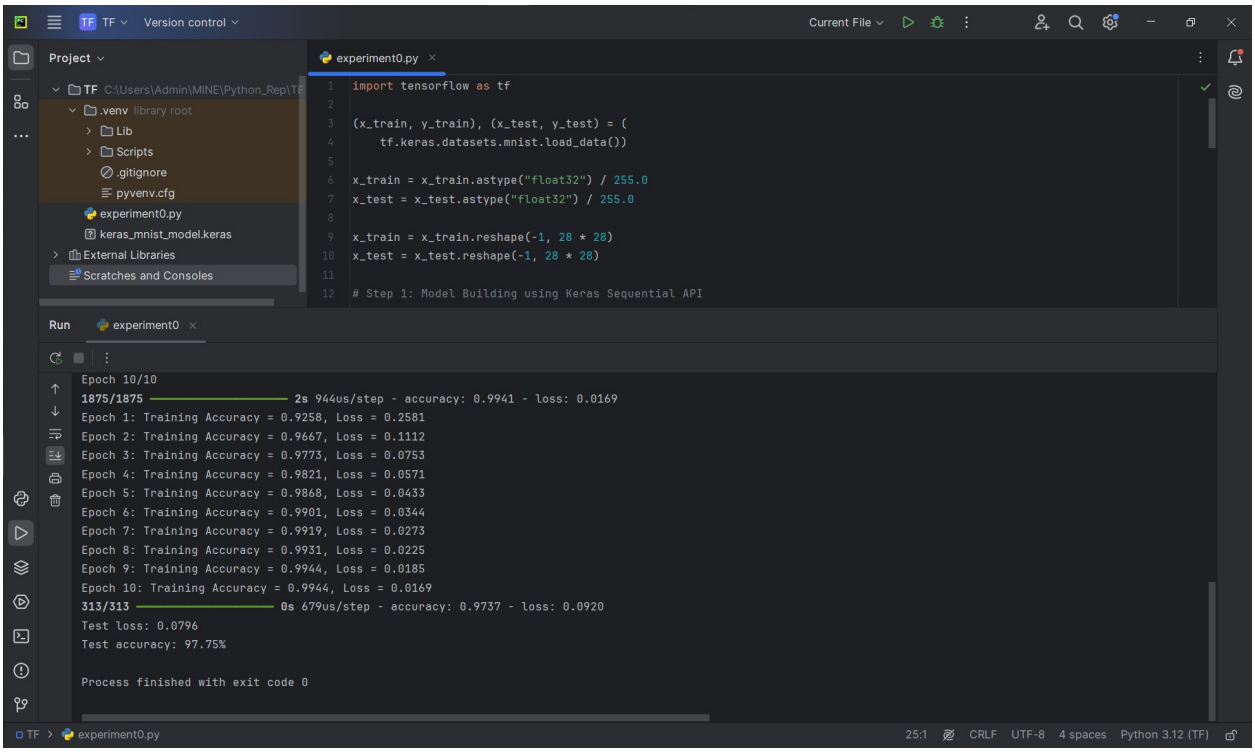

Figure 3: Experiment 0: execution

Experiment 1

This experiment helped me to understand how varying learning rates influence training outcomes.

```
import tensorflow as tf
import matplotlib.pyplot as plt

# Load and preprocess the MNIST dataset
(x_train, y_train), (x_test, y_test) =(
    tf.keras.datasets.mnist.load_data())

x_train = x_train.astype("float32") / 255.0
x_test = x_test.astype("float32") / 255.0

x_train = x_train.reshape(-1, 28 * 28)
x_test = x_test.reshape(-1, 28 * 28)

# Define learning rates to test
learning_rates = [0.0001, 0.001, 0.01]
histories = []

for lr in learning_rates:
    model = tf.keras.models.Sequential([
        tf.keras.layers.InputLayer(input_shape=(784,)),
        tf.keras.layers.Dense(128, activation='relu'),
        tf.keras.layers.Dense(10)
```

```
    ])

    model.compile(optimizer=tf.keras.optimizers.Adam(learning_rate=lr),
              loss=tf.keras.losses.SparseCategoricalCrossentropy(from_logits=True),
              metrics=['accuracy'])

    print(f"\nTraining model with learning rate = {lr}")
    history = model.fit(x_train, y_train, epochs=10,
        batch_size=32, verbose=0)
    histories.append((lr, history))

    test_loss, test_acc = model.evaluate(x_test, y_test,
        verbose=0)
    print(f"Test loss: {test_loss:.4f}")
    print(f"Test accuracy: {test_acc * 100:.2f}%")

#Plotting the results
plt.figure(figsize=(12, 5))

# Plot training accuracy
plt.subplot(1, 2, 1)
for lr, history in histories:
    plt.plot(history.history['accuracy'], label=f'lr={lr}')
plt.title('Training Accuracy vs Epochs')
plt.xlabel('Epochs')
plt.ylabel('Accuracy')
plt.legend()

# Plot training loss
plt.subplot(1, 2, 2)
for lr, history in histories:
    plt.plot(history.history['loss'], label=f'lr={lr}')
plt.title('Training Loss vs Epochs')
plt.xlabel('Epochs')
plt.ylabel('Loss')
plt.legend()

plt.tight_layout()
plt.show()
```

The graph shows the changing of loss and accuracy, depending on the learning rate value. We can see that, when the value of epoch is a bit more than 0, the graphs with learning rate 0.001 and 0.01 intersects. The same situation is with their loss. We can also see that graphs with learning rate 0.0001 and 0.001 are very close to each other (almost parallel) in the end of the experiment (when epoch value is more than 8).

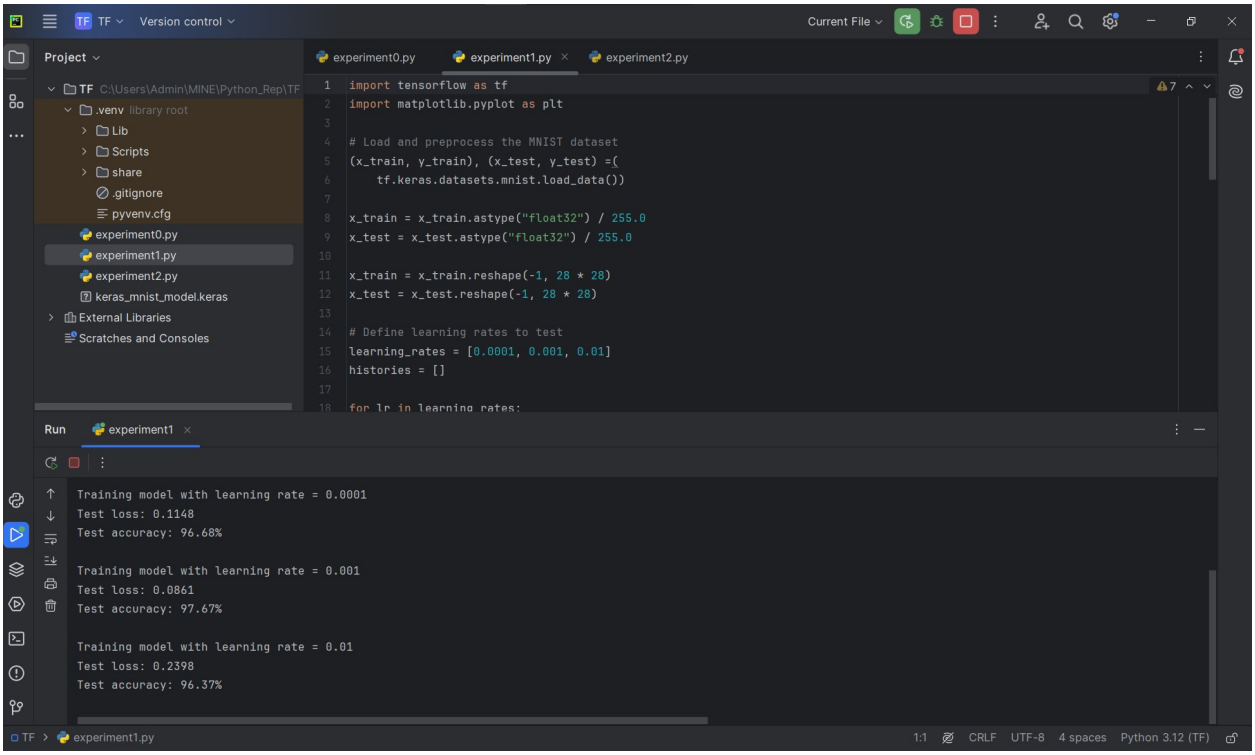

Figure 4: Experiment 1: execution

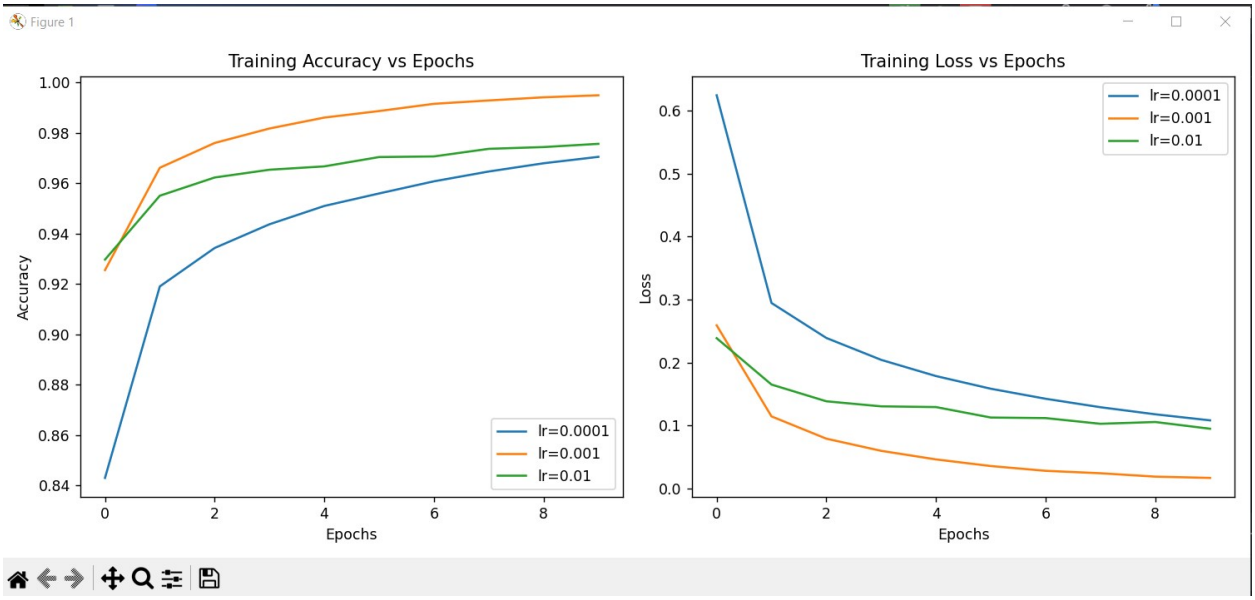

Figure 5: Experiment 1: graph

Experiment 2

In this experiment I examined the role of neuron counts in network architecture and understood how changes in the number of neurons within the hidden layer influence training outcomes.

```
import tensorflow as tf
import matplotlib.pyplot as plt

# Load and preprocess the MNIST dataset
(x_train, y_train), (x_test, y_test) =(
    tf.keras.datasets.mnist.load_data())

x_train = x_train.astype("float32") / 255.0
x_test = x_test.astype("float32") / 255.0

x_train = x_train.reshape(-1, 28 * 28)
x_test = x_test.reshape(-1, 28 * 28)

# Define learning rate to test
learning_rate = 0.001
histories = []

values = [64, 128, 256]

for val in values:
    model = tf.keras.models.Sequential([
        tf.keras.layers.InputLayer(input_shape=(784,)),
        tf.keras.layers.Dense(val, activation='relu'),
        tf.keras.layers.Dense(10)
    ])

    model.compile(optimizer=tf.keras.optimizers.Adam(learning_rate=learning_rate),
              loss=tf.keras.losses.SparseCategoricalCrossentropy(from_logits=True),
              metrics=['accuracy'])

    print(f"\nTraining model with neuron count = {learning_rate}")
    history = model.fit(x_train, y_train, epochs=10, batch_size=32, verbose=0)
    histories.append((val, history))

    test_loss, test_acc = model.evaluate(x_test, y_test, verbose=0)
    print(f"Test loss: {test_loss:.4f}")
    print(f"Test accuracy: {test_acc * 100:.2f}%")

#Plotting the results
plt.figure(figsize=(12, 5))

# Plot training accuracy
plt.subplot(1, 2, 1)
for val, history in histories:
    plt.plot(history.history['accuracy'], label=f'neuron count={val}')
plt.title('Training Accuracy vs Epochs')
plt.xlabel('Epochs')
plt.ylabel('Accuracy')
```

```
plt.legend()

# Plot training loss
plt.subplot(1, 2, 2)
for val, history in histories:
    plt.plot(history.history['loss'], label=f'neuron count={val}')
plt.title('Training Loss vs Epochs')
plt.xlabel('Epochs')
plt.ylabel('Loss')
plt.legend()

plt.tight_layout()
plt.show()
```

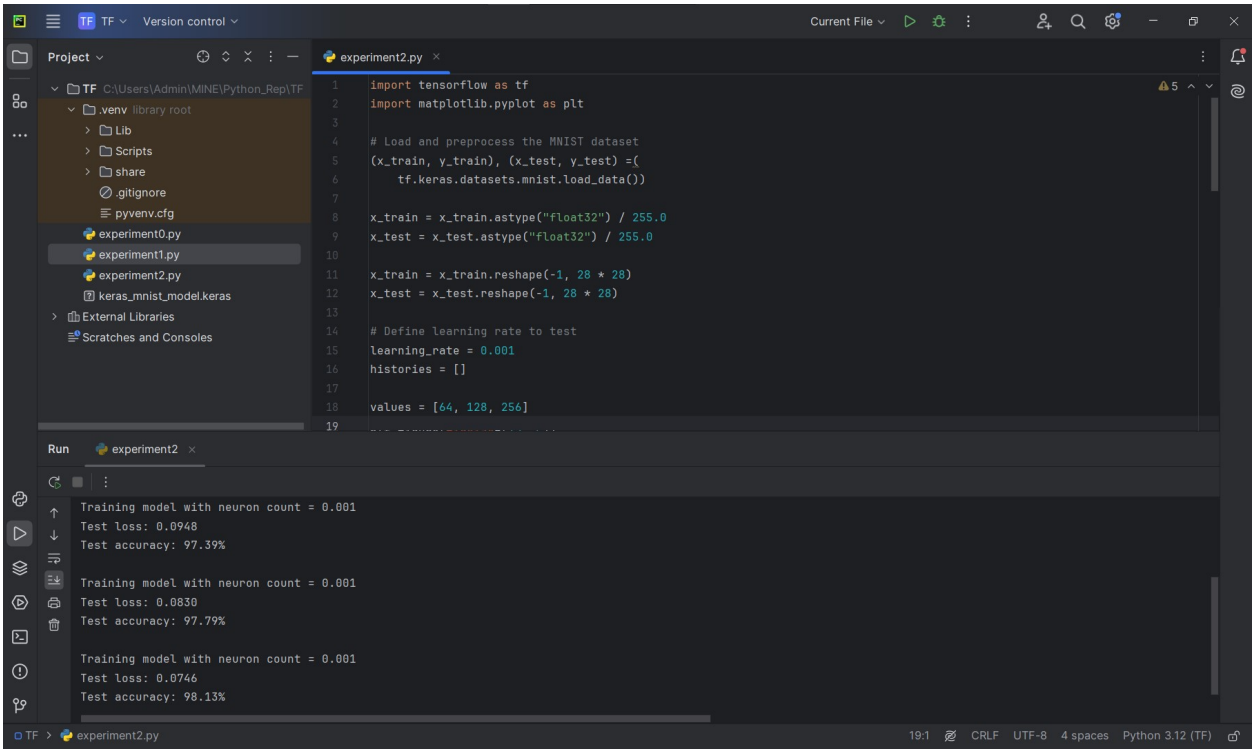

Figure 6: Experiment 2: execution

The graph shows the changing of loss and accuracy, depending on the neuron count value. Due to the graph we can see that the more neuron count is, the higher is accuracy and the lower is training loss.

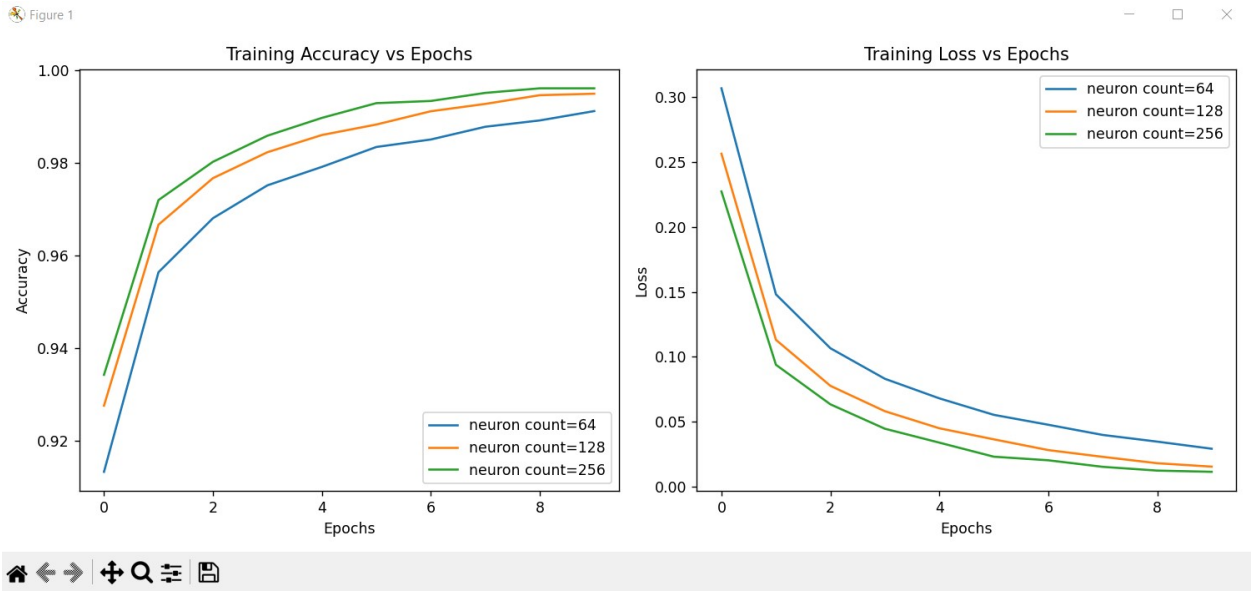

Figure 7: Experiment 2: graph