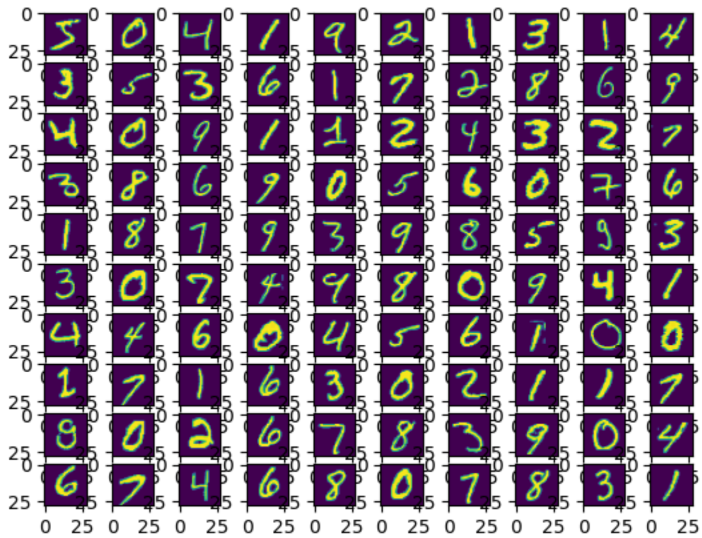
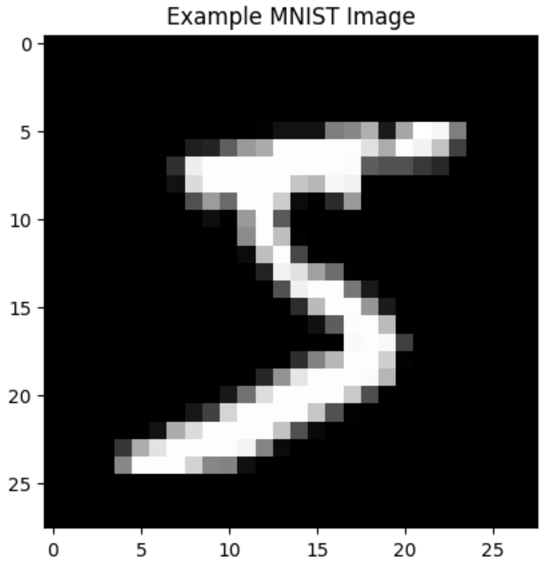
```
import cv2   # it will allow us to load our images into the script
import numpy as np   # used for reformatting our own images
import tensorflow as tf   # main library used to load data sets, build neural networks, train them, etc.
import matplotlib.pyplot as plt   # used for visualization


# Load MNIST data
(x_train, y_train), (x_test, y_test) = tf.keras.datasets.mnist.load_data()


print('The shape of the training inputs:', x_train.shape)
print('The shape of the training labels:',y_train.shape)
print('The shape of the testing inputs:',x_test.shape)
print('The shape of the testing labels:',y_test.shape)
```

```
The shape of the training inputs: (60000, 28, 28)
The shape of the training labels: (60000,)
The shape of the testing inputs: (10000, 28, 28)
The shape of the testing labels: (10000,)
```

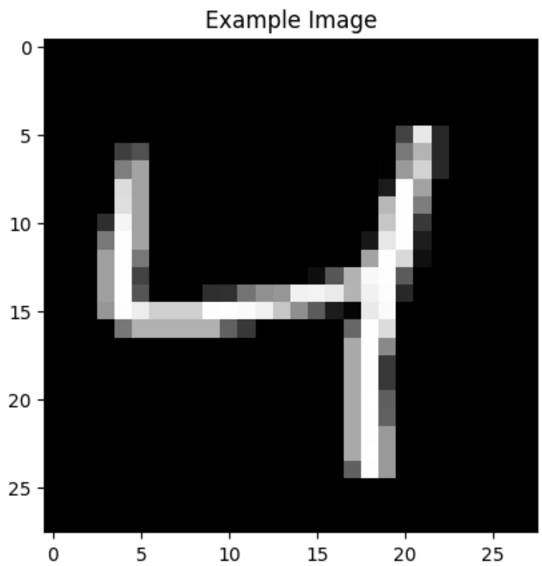
```
# plotting the first 9 images in the train set of MNIST

fig, axs = plt.subplots(10, 10)
cnt = 0
for i in range(10):
    for j in range(10):
        axs[i, j].imshow(x_train[cnt])
        cnt += 1
```



```
import matplotlib.pyplot as plt

# Display an example image
plt.imshow(x_train[0].reshape(28,28), cmap='gray')
plt.title('Example MNIST Image')
plt.show()
```




```
import matplotlib.pyplot as plt

# Preprocess the data
x_train = x_train.astype('float32') / 255.0
x_test = x_test.astype('float32') / 255.0

# Display an example image
plt.imshow(x_train[2].reshape(28, 28), cmap='gray') # Reshape the image to 28x28
plt.title('Example Image')
plt.show()
```




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

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

```
model.add(tf.keras.layers.Flatten(input_shape=(28,28)))
```

```
/usr/local/lib/python3.10/dist-packages/keras/src/layers/reshaping/flatten.py:37: UserWarning: Do not pass an `input_shape`/`input_c
  super().__init__(**kwargs)
```

```
#Hidden Layer
model.add(tf.keras.layers.Dense(units=128, activation=tf.nn.relu))
```

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

```
model.summary()
```

```
Model: "sequential_8"
```

Layer (type)	Output Shape	Param #
flatten_4 (Flatten)	(None, 784)	0
dense_10 (Dense)	(None, 128)	100,480
dense_11 (Dense)	(None, 128)	16,512
dense_12 (Dense)	(None, 10)	1,290

```
Total params: 118,282 (462.04 KB)
Trainable params: 118,282 (462.04 KB)
Non-trainable params: 0 (0.00 B)
```

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

```
# Define a single layer perceptron model
model = tf.keras.models.Sequential()
model.add(tf.keras.layers.Dense(10, activation='softmax', input_shape=(28*28,)))
```

```
# Define a single layer perceptron model
model = tf.keras.models.Sequential()
model.add(tf.keras.layers.Dense(10, activation='softmax', input_shape=(28*28,)))

# Compile the model - It is important to compile the model after it is defined.
model.compile(optimizer='adam', loss='sparse_categorical_crossentropy', metrics=['accuracy'])

# Reshape x_train to flatten the images
x_train_flattened = x_train.reshape(-1, 28*28) # -1 infers the number of samples

# Train the model using the flattened data
history=model.fit(x_train_flattened,
                  y_train,
                  epochs=10,
                  batch_size=100,
                  validation_split=0.2)
```

```
/usr/local/lib/python3.10/dist-packages/keras/src/layers/core/dense.py:87: UserWarning: Do not pass an `input_shape`/`input_dim` arg
  super().__init__(activity_regularizer=activity_regularizer, **kwargs)
Epoch 1/10
480/480 ━━━━━━━━━━━━━━━━━━━━ 4s 5ms/step - accuracy: 0.7142 - loss: 1.3514 - val_accuracy: 0.8798 - val_loss: 0.5113
Epoch 2/10
480/480 ━━━━━━━━━━━━━━━━━━━━ 4s 4ms/step - accuracy: 0.8724 - loss: 0.5048 - val_accuracy: 0.8988 - val_loss: 0.3919
Epoch 3/10
480/480 ━━━━━━━━━━━━━━━━━━━━ 2s 2ms/step - accuracy: 0.8922 - loss: 0.3981 - val_accuracy: 0.9062 - val_loss: 0.3500
Epoch 4/10
480/480 ━━━━━━━━━━━━━━━━━━━━ 2s 4ms/step - accuracy: 0.9018 - loss: 0.3612 - val_accuracy: 0.9120 - val_loss: 0.3274
Epoch 5/10
480/480 ━━━━━━━━━━━━━━━━━━━━ 2s 2ms/step - accuracy: 0.9043 - loss: 0.3384 - val_accuracy: 0.9142 - val_loss: 0.3143
Epoch 6/10
480/480 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - accuracy: 0.9086 - loss: 0.3250 - val_accuracy: 0.9153 - val_loss: 0.3053
Epoch 7/10
480/480 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - accuracy: 0.9141 - loss: 0.3057 - val_accuracy: 0.9172 - val_loss: 0.2977
Epoch 8/10
480/480 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - accuracy: 0.9150 - loss: 0.3020 - val_accuracy: 0.9183 - val_loss: 0.2934
Epoch 9/10
480/480 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - accuracy: 0.9170 - loss: 0.2947 - val_accuracy: 0.9187 - val_loss: 0.2890
Epoch 10/10
480/480 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - accuracy: 0.9189 - loss: 0.2896 - val_accuracy: 0.9183 - val_loss: 0.2871
```

```
# Evaluate the model
x_test_flattened = x_test.reshape(-1, 28*28) # Flatten the x_test data
test_loss, test_acc = model.evaluate(x_test_flattened, y_test)
print(f'Test accuracy: {test_acc}')
```

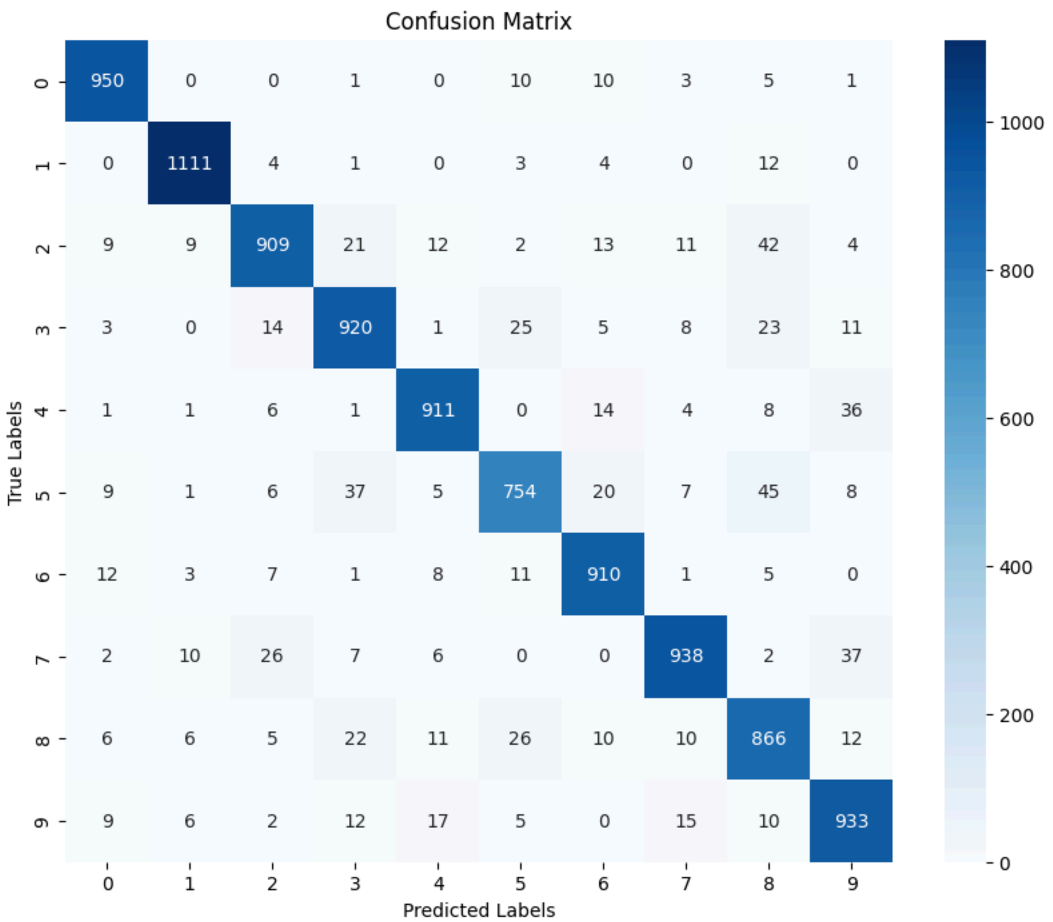
```
313/313 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - accuracy: 0.9077 - loss: 0.3242
Test accuracy: 0.920199990272522
```

```
from sklearn.metrics import confusion_matrix, classification_report
import numpy as np

# Generate predictions using the model
y_pred = model.predict(x_test_flattened)
y_pred_classes = np.argmax(y_pred, axis=1) # Convert predictions to class labels

# Generate the confusion matrix
cm = confusion_matrix(y_test, y_pred_classes)
print("Confusion Matrix:")
print(cm)
```

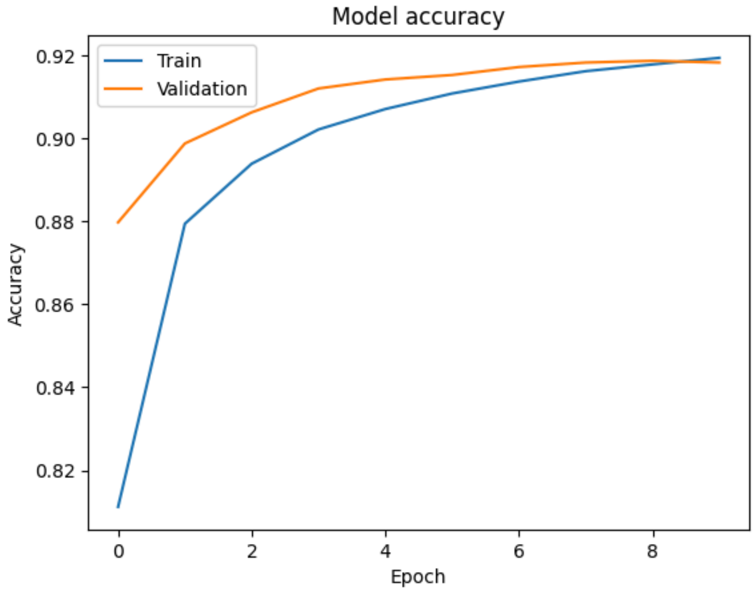
```
# Generate the classification report
cr = classification_report(y_test, y_pred_classes)
print("\nClassification Report:")
print(cr)
```

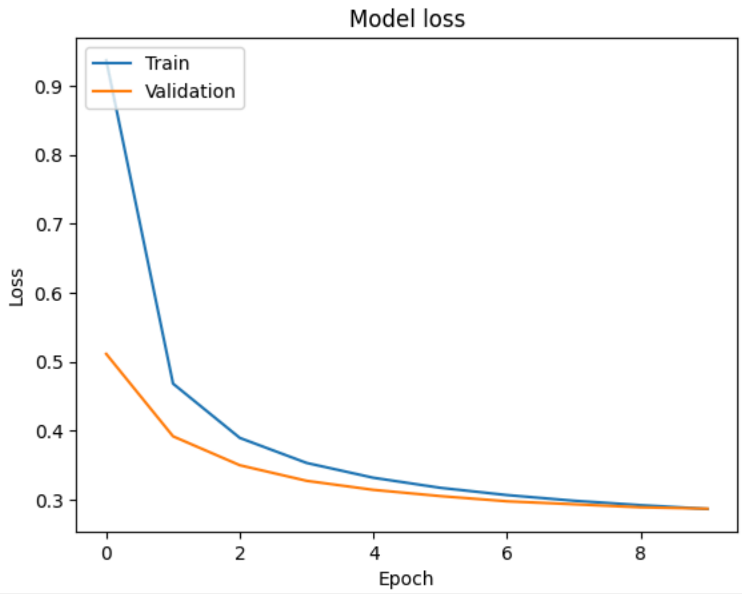
```
313/313 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step
Confusion Matrix:
[[ 950    0    0    1    0   10   10    3    5    1]
 [    0 1111    4    1    0    3    4    0   12    0]
 [    9    9  909   21   12    2   13   11   42    4]
 [    3    0   14  920    1   25    5    8   23   11]
 [    1    1    6    1  911    0   14    4    8   36]
 [    9    1    6   37    5  754   20    7   45    8]
 [   12    3    7    1    8   11  910    1    5    0]
 [    2   10   26    7    6    0    0  938    2   37]
 [    6    6    5   22   11   26   10   10  866   12]
 [    9    6    2   12   17    5    0   15   10  933]]

Classification Report:
              precision    recall  f1-score   support

           0       0.95      0.97      0.96       980
           1       0.97      0.98      0.97      1135
           2       0.93      0.88      0.90      1032
           3       0.90      0.91      0.91      1010
           4       0.94      0.93      0.93       982
           5       0.90      0.85      0.87       892
           6       0.92      0.95      0.94       958
           7       0.94      0.91      0.93      1028
           8       0.85      0.89      0.87       974
           9       0.90      0.92      0.91      1009

    accuracy                           0.92     10000
   macro avg       0.92      0.92      0.92     10000
weighted avg       0.92      0.92      0.92     10000
```

```
import seaborn as sns
import matplotlib.pyplot as plt
import numpy as np # Added import for numpy

# Generate the confusion matrix
y_pred_classes = np.argmax(y_pred, axis=1) # Convert predictions to class labels
cm = confusion_matrix(y_test, y_pred_classes) # Use class labels for y_pred

# Plot the confusion matrix
plt.figure(figsize=(10, 8))
sns.heatmap(cm, annot=True, fmt="d", cmap="Blues")
plt.title("Confusion Matrix")
plt.xlabel("Predicted Labels")
plt.ylabel("True Labels")
plt.show()
```




```
import matplotlib.pyplot as plt
# Plot training & validation accuracy values
plt.plot(history.history['accuracy'])
plt.plot(history.history['val_accuracy'])
plt.title('Model accuracy')
plt.ylabel('Accuracy')
plt.xlabel('Epoch')
plt.legend(['Train', 'Validation'], loc='upper left')
plt.show()

# Plot training & validation loss values
plt.plot(history.history['loss'])
plt.plot(history.history['val_loss'])
plt.title('Model loss')
plt.ylabel('Loss')
plt.xlabel('Epoch')
plt.legend(['Train', 'Validation'], loc='upper left')
plt.show()
```

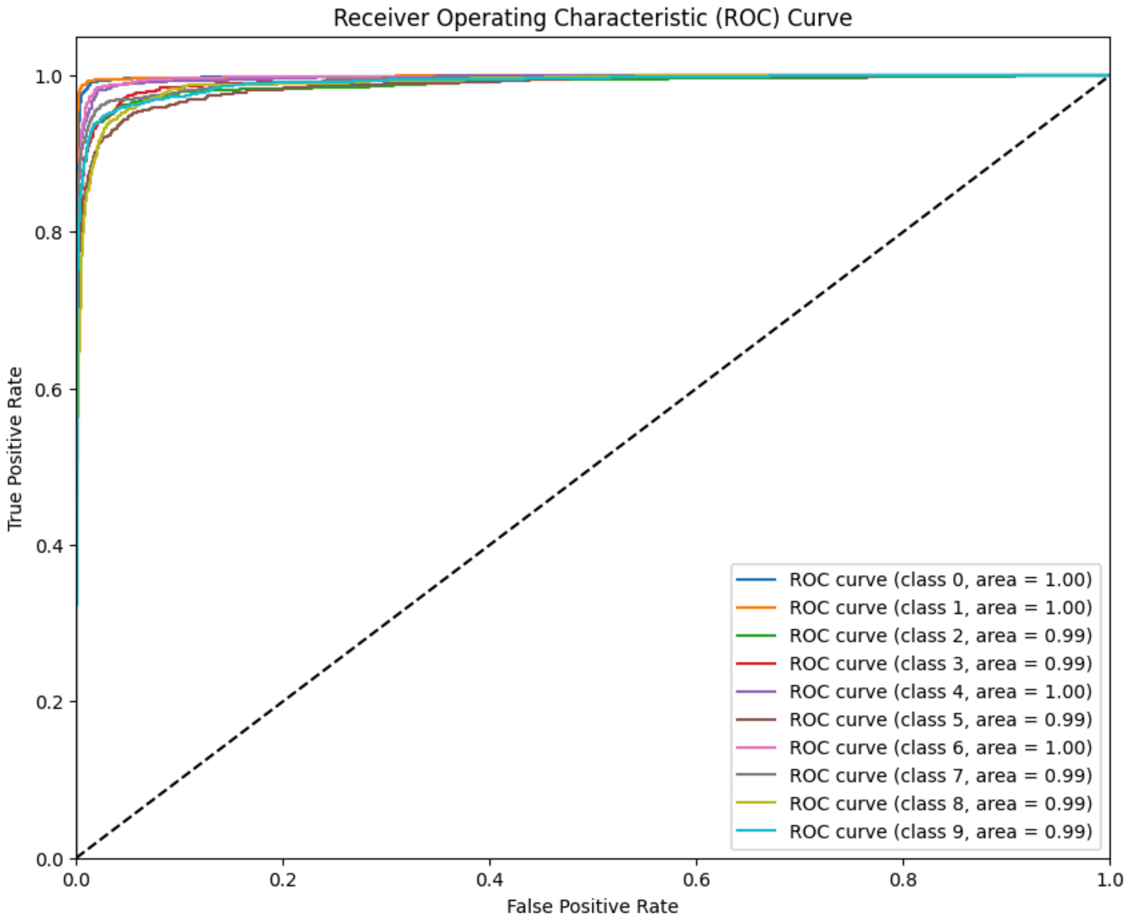





```
import numpy as np
from sklearn.metrics import roc_curve, auc
from sklearn.preprocessing import label_binarize
import matplotlib.pyplot as plt

# Get predicted probabilities for each class
# Reshape x_test to be 2-dimensional
x_test_flattened = x_test.reshape(-1, 28*28)
y_pred_proba = model.predict(x_test_flattened)

# Binarize the labels
y_test_bin = label_binarize(y_test, classes=np.arange(10))

# Compute ROC curve and AUC for each class
fpr = dict()
tpr = dict()
roc_auc = dict()
for i in range(10):
    fpr[i], tpr[i], _ = roc_curve(y_test_bin[:, i], y_pred_proba[:, i])
    roc_auc[i] = auc(fpr[i], tpr[i])

# Plot ROC curve for each class
plt.figure(figsize=(10, 8))
for i in range(10):
    plt.plot(fpr[i], tpr[i], label='ROC curve (class %d, area = %0.2f)' % (i, roc_auc[i]))

plt.plot([0, 1], [0, 1], 'k--')
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.05])
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('Receiver Operating Characteristic (ROC) Curve')
plt.legend(loc="lower right")
```

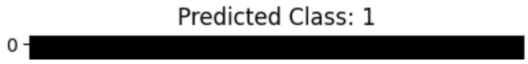
```
plt.show()
```

313/313 ━━━━━━━━━━━━━━━━━━━━ 27s 87ms/step




```
import matplotlib.pyplot as plt
import numpy as np
# Select a random image from the test dataset
image_index = np.random.randint(0, len(x_test))
test_image = x_test[image_index]

# Reshape the image to match the model's input shape
test_image = test_image.reshape(1, 28*28) # Changed to flatten the input image

# Make a prediction
prediction = model.predict(test_image)

# Get the predicted class
predicted_class = np.argmax(prediction)

# Display the image and the prediction
plt.imshow(test_image.reshape(28, 28), cmap='gray')
plt.title(f'Predicted Class: {predicted_class}')
plt.show()
```

```
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 38ms/step
```




```
import matplotlib.pyplot as plt
import numpy as np
from sklearn.metrics import confusion_matrix, classification_report

#Import the models module from keras
from tensorflow import keras
from keras import models
from keras import layers

# Define the MLP model
model = models.Sequential()
model.add(layers.Flatten(input_shape=(28, 28)))  # Flatten the input images
model.add(layers.Dense(128, activation='relu'))  # Hidden layer 1
model.add(layers.Dense(64, activation='relu'))   # Hidden layer 2
model.add(layers.Dense(10, activation='softmax')) # Output layer
```