```
import cv2
import numpy as np
from PIL import Image
import io
import os

from google.colab import files
uploaded = files.upload()

filename = list(uploaded.keys())[0]

image = cv2.imread(filename)


image_rgb = cv2.cvtColor(image, cv2.COLOR_BGR2RGB)

original_size = os.path.getsize(filename) / 1024
print(f"Original Image Size: {original_size:.2f} KB")


jpeg_filename = "compressed_image.jpg"
cv2.imwrite(jpeg_filename, image, [cv2.IMWRITE_JPEG_QUALITY, 30])


jpeg_size = os.path.getsize(jpeg_filename) / 1024
print(f"JPEG Compressed Size: {jpeg_size:.2f} KB")


png_filename = "compressed_image.png"
cv2.imwrite(png_filename, image, [cv2.IMWRITE_PNG_COMPRESSION, 9])


png_size = os.path.getsize(png_filename) / 1024
print(f"PNG Compressed Size: {png_size:.2f} KB")


import matplotlib.pyplot as plt

fig, ax = plt.subplots(1, 3, figsize=(15,5))


ax[0].imshow(image_rgb)
ax[0].set_title("Original Image")
ax[0].axis("off")


jpeg_img = Image.open(jpeg_filename)
ax[1].imshow(jpeg_img)
ax[1].set_title("JPEG Compressed Image")
ax[1].axis("off")


png_img = Image.open(png_filename)
ax[2].imshow(png_img)
ax[2].set_title("PNG Compressed Image")
ax[2].axis("off")

plt.show()
```

```
Choose Files nature-4k-p…-preview.jpg
• nature-4k-pc-full-hd-wallpaper-preview.jpg(image/jpeg) - 104758 bytes, last modified: 2/14/2025 - 100% done
Saving nature-4k-pc-full-hd-wallpaper-preview.jpg to nature-4k-pc-full-hd-wallpaper-preview (1).jpg
Original Image Size: 102.30 KB
JPEG Compressed Size: 40.21 KB
PNG Compressed Size: 587.04 KB
```

Original Image

JPEG Compressed Image

PNG Compressed Image

```
import tensorflow as tf
from tensorflow import keras
from tensorflow.keras import layers
from sklearn.metrics import classification_report, roc_auc_score, roc_curve, confusion_matrix
import matplotlib.pyplot as plt
import seaborn as sns
import numpy as np

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

# Normalize pixel values
x_train, x_test = x_train / 255.0, x_test / 255.0

# Reshape to match CNN input (28x28 images with 1 channel)
x_train = x_train.reshape(-1, 28, 28, 1)
x_test = x_test.reshape(-1, 28, 28, 1)

# One-hot encode labels
y_train = keras.utils.to_categorical(y_train, 10)
y_test = keras.utils.to_categorical(y_test, 10)
```

```
Downloading data from https://storage.googleapis.com/tensorflow/tf-keras-datasets/mnist.npz
11490434/11490434 ━━━━━━━━━━━━━━━━━━━━ 0s 0us/step
```

```
# Define CNN model
model = keras.Sequential([
    layers.Conv2D(32, (3,3), strides=2, activation='relu', input_shape=(28,28,1)),
    layers.MaxPooling2D((2,2)),
    layers.Conv2D(64, (3,3), strides=2, activation='relu'),
    layers.MaxPooling2D((2,2)),
    layers.Flatten(),
    layers.Dense(128, activation='relu'),
    layers.Dense(10, activation='softmax')  # 10 classes
])

# Compile the model
model.compile(optimizer='adam',
              loss='categorical_crossentropy',
              metrics=['accuracy'])

history = model.fit(x_train, y_train, epochs=3, batch_size=128, validation_split=0.2)
```

```
/usr/local/lib/python3.11/dist-packages/keras/src/layers/convolutional/base_conv.py:107: UserWarning: Do not pass an `i
  super().__init__(activity_regularizer=activity_regularizer, **kwargs)
Epoch 1/3
375/375 ━━━━━━━━━━━━━━━━━━━━ 9s 19ms/step - accuracy: 0.6667 - loss: 1.1541 - val_accuracy: 0.9377 - val_loss: 0.2090
Epoch 2/3
375/375 ━━━━━━━━━━━━━━━━━━━━ 6s 16ms/step - accuracy: 0.9413 - loss: 0.1948 - val_accuracy: 0.9557 - val_loss: 0.1491
Epoch 3/3
375/375 ━━━━━━━━━━━━━━━━━━━━ 10s 15ms/step - accuracy: 0.9570 - loss: 0.1456 - val_accuracy: 0.9607 - val_loss: 0.1342
```

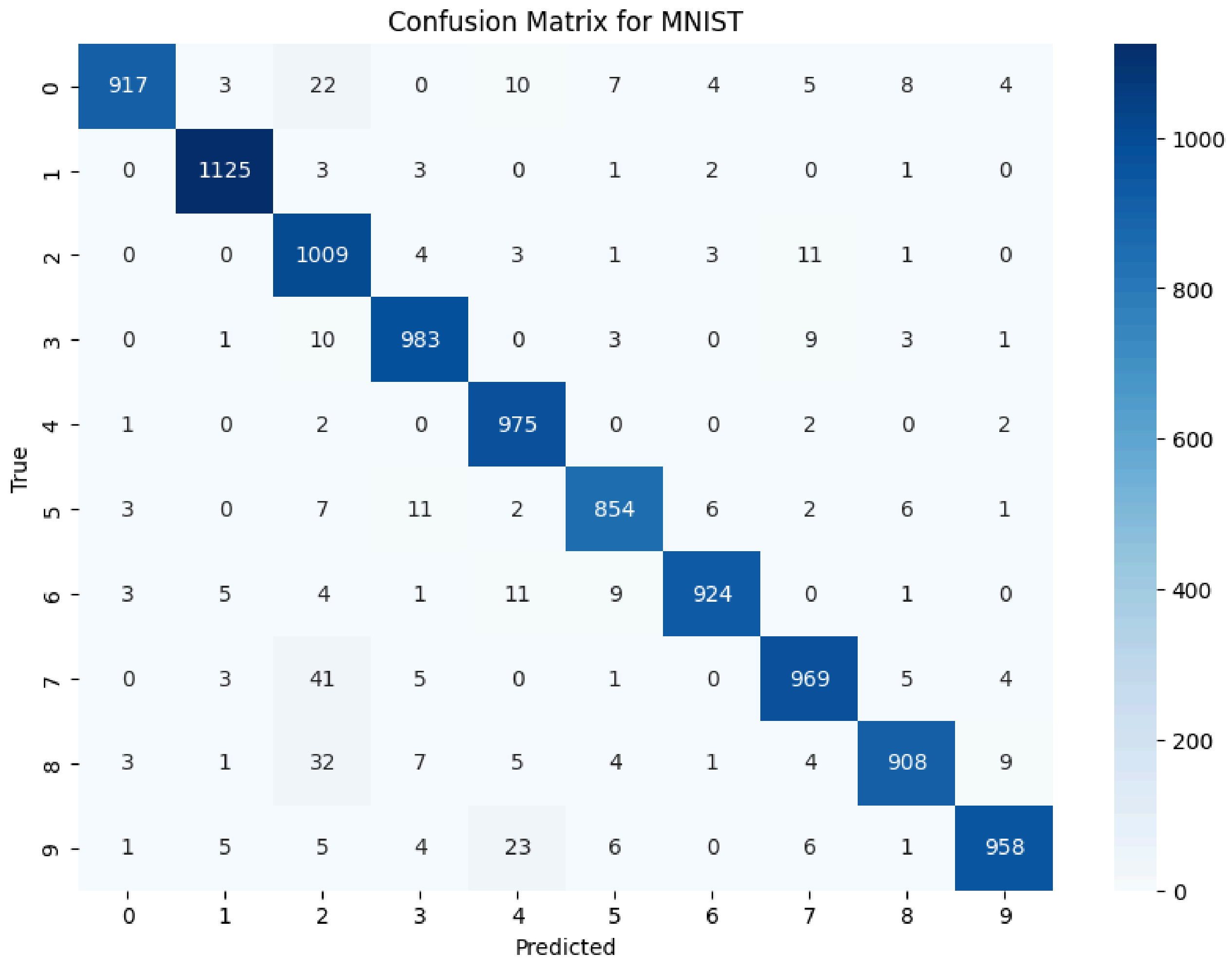
```
# Predictions
y_pred = model.predict(x_test)
y_pred_classes = np.argmax(y_pred, axis=1)
y_true = np.argmax(y_test, axis=1)

# Model Evaluation
accuracy = np.mean(y_pred_classes == y_true)
precision = classification_report(y_true, y_pred_classes, output_dict=True)['weighted avg']['precision']
recall = classification_report(y_true, y_pred_classes, output_dict=True)['weighted avg']['recall']
f1_score = classification_report(y_true, y_pred_classes, output_dict=True)['weighted avg']['f1-score']

# Confusion Matrix
conf_matrix = confusion_matrix(y_true, y_pred_classes)
plt.figure(figsize=(10, 7))
sns.heatmap(conf_matrix, annot=True, fmt='d', cmap="Blues")
plt.xlabel('Predicted')
plt.ylabel('True')
plt.title('Confusion Matrix for MNIST')
plt.show()

# ROC Curve & AUC
auc_score = roc_auc_score(y_test, y_pred, multi_class='ovr')
print(f"AUC Score: {auc_score:.4f}")
```

```
313/313 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step
```



```
AUC Score: 0.9989
```

```
# Load CIFAR-10 dataset
(x_train, y_train), (x_test, y_test) = keras.datasets.cifar10.load_data()

# Normalize pixel values
x_train, x_test = x_train / 255.0, x_test / 255.0

# One-hot encode labels
y_train = keras.utils.to_categorical(y_train, 10)
y_test = keras.utils.to_categorical(y_test, 10)

# Define CNN model
model = keras.Sequential([
    layers.Conv2D(32, (3,3), activation='relu', input_shape=(32,32,3)),
    layers.MaxPooling2D((2,2)),
    layers.Conv2D(64, (3,3), activation='relu'),
    layers.MaxPooling2D((2,2)),
    layers.Conv2D(128, (3,3), activation='relu'),
    layers.MaxPooling2D((2,2)),
    layers.Flatten(),
    layers.Dense(128, activation='relu'),
    layers.Dense(10, activation='softmax')  # 10 classes
])

# Compile the model
model.compile(optimizer='adam',
              loss='categorical_crossentropy',
              metrics=['accuracy'])

# Train the model
history = model.fit(x_train, y_train, epochs=20, batch_size=128, validation_split=0.2)
```

```
Downloading data from https://www.cs.toronto.edu/~kriz/cifar-10-python.tar.gz
170498071/170498071 ━━━━━━━━━━━━━━━━━━━━ 4s 0us/step
/usr/local/lib/python3.11/dist-packages/keras/src/layers/convolutional/base_conv.py:107: UserWarning: Do not pass an `i
  super().__init__(activity_regularizer=activity_regularizer, **kwargs)
Epoch 1/20
313/313 ━━━━━━━━━━━━━━━━━━━━ 52s 160ms/step - accuracy: 0.2894 - loss: 1.9187 - val_accuracy: 0.4972 - val_loss: 1.4064
Epoch 2/20
313/313 ━━━━━━━━━━━━━━━━━━━━ 82s 161ms/step - accuracy: 0.5148 - loss: 1.3564 - val_accuracy: 0.5576 - val_loss: 1.2370
Epoch 3/20
313/313 ━━━━━━━━━━━━━━━━━━━━ 79s 154ms/step - accuracy: 0.5816 - loss: 1.1830 - val_accuracy: 0.5931 - val_loss: 1.1507
Epoch 4/20
313/313 ━━━━━━━━━━━━━━━━━━━━ 84s 160ms/step - accuracy: 0.6282 - loss: 1.0615 - val_accuracy: 0.6238 - val_loss: 1.0842
Epoch 5/20
313/313 ━━━━━━━━━━━━━━━━━━━━ 48s 152ms/step - accuracy: 0.6560 - loss: 0.9841 - val_accuracy: 0.6532 - val_loss: 1.0039
Epoch 6/20
313/313 ━━━━━━━━━━━━━━━━━━━━ 48s 152ms/step - accuracy: 0.6858 - loss: 0.9070 - val_accuracy: 0.6708 - val_loss: 0.9650
Epoch 7/20
313/313 ━━━━━━━━━━━━━━━━━━━━ 85s 163ms/step - accuracy: 0.7027 - loss: 0.8554 - val_accuracy: 0.6819 - val_loss: 0.9211
Epoch 8/20
313/313 ━━━━━━━━━━━━━━━━━━━━ 48s 154ms/step - accuracy: 0.7200 - loss: 0.8094 - val_accuracy: 0.7000 - val_loss: 0.8831
Epoch 9/20
313/313 ━━━━━━━━━━━━━━━━━━━━ 81s 152ms/step - accuracy: 0.7391 - loss: 0.7529 - val_accuracy: 0.6942 - val_loss: 0.9102
Epoch 10/20
313/313 ━━━━━━━━━━━━━━━━━━━━ 85s 161ms/step - accuracy: 0.7520 - loss: 0.7138 - val_accuracy: 0.6905 - val_loss: 0.9209
Epoch 11/20
313/313 ━━━━━━━━━━━━━━━━━━━━ 48s 154ms/step - accuracy: 0.7661 - loss: 0.6711 - val_accuracy: 0.7055 - val_loss: 0.8788
Epoch 12/20
313/313 ━━━━━━━━━━━━━━━━━━━━ 84s 161ms/step - accuracy: 0.7836 - loss: 0.6260 - val_accuracy: 0.7003 - val_loss: 0.9017
Epoch 13/20
313/313 ━━━━━━━━━━━━━━━━━━━━ 79s 151ms/step - accuracy: 0.7909 - loss: 0.5969 - val_accuracy: 0.7126 - val_loss: 0.8844
Epoch 14/20
313/313 ━━━━━━━━━━━━━━━━━━━━ 51s 161ms/step - accuracy: 0.8083 - loss: 0.5510 - val_accuracy: 0.7075 - val_loss: 0.8844
Epoch 15/20
313/313 ━━━━━━━━━━━━━━━━━━━━ 82s 161ms/step - accuracy: 0.8144 - loss: 0.5265 - val_accuracy: 0.7084 - val_loss: 0.9089
Epoch 16/20
313/313 ━━━━━━━━━━━━━━━━━━━━ 81s 158ms/step - accuracy: 0.8278 - loss: 0.4946 - val_accuracy: 0.7159 - val_loss: 0.9150
Epoch 17/20
313/313 ━━━━━━━━━━━━━━━━━━━━ 81s 153ms/step - accuracy: 0.8369 - loss: 0.4686 - val_accuracy: 0.7137 - val_loss: 0.9149
Epoch 18/20
313/313 ━━━━━━━━━━━━━━━━━━━━ 51s 162ms/step - accuracy: 0.8496 - loss: 0.4281 - val_accuracy: 0.7185 - val_loss: 0.9235
Epoch 19/20
313/313 ━━━━━━━━━━━━━━━━━━━━ 79s 154ms/step - accuracy: 0.8638 - loss: 0.3958 - val_accuracy: 0.7150 - val_loss: 0.9577
Epoch 20/20
313/313 ━━━━━━━━━━━━━━━━━━━━ 50s 158ms/step - accuracy: 0.8718 - loss: 0.3699 - val_accuracy: 0.6989 - val_loss: 1.0282
```

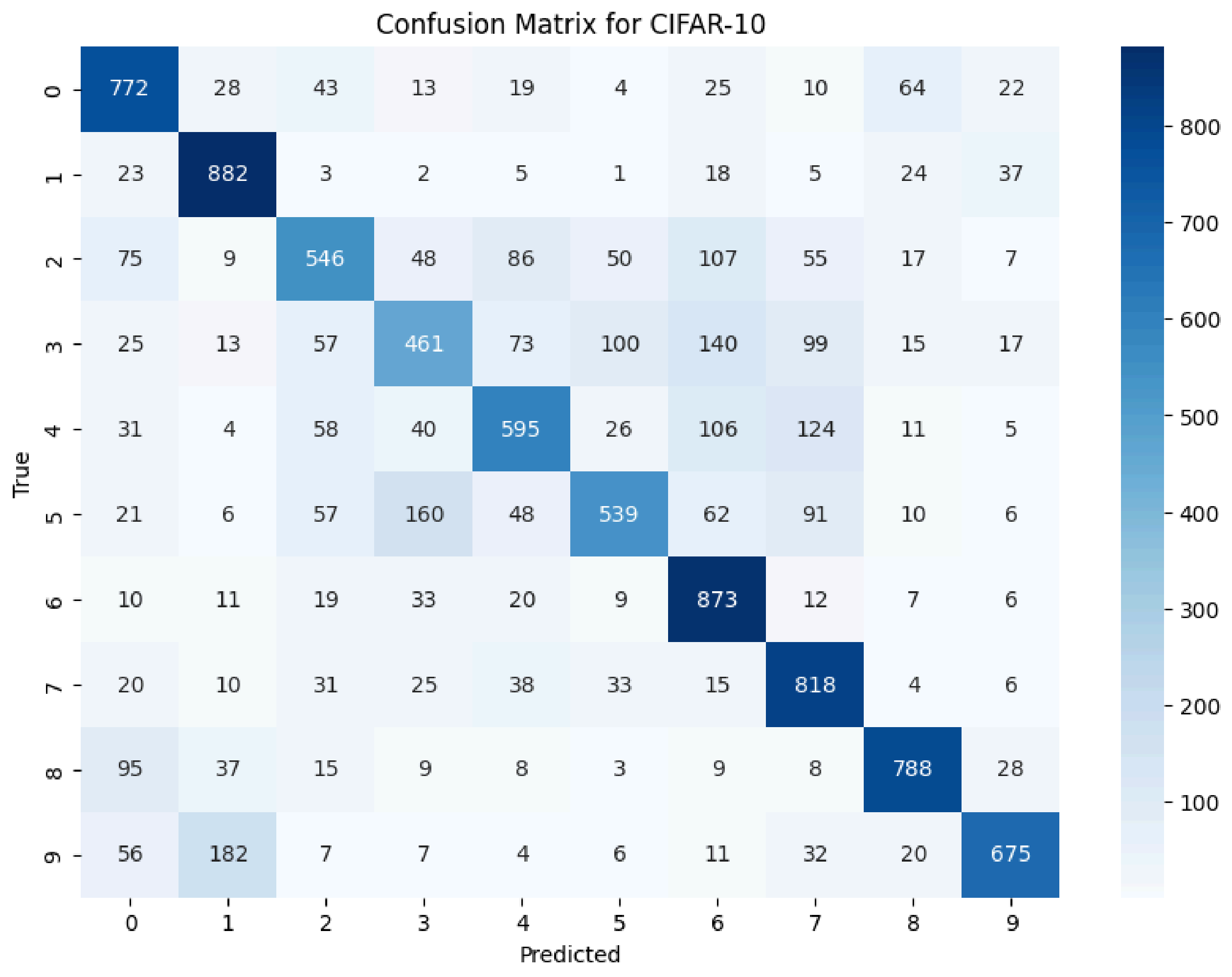
```
y_pred = model.predict(x_test)
y_pred_classes = np.argmax(y_pred, axis=1)
y_true = np.argmax(y_test, axis=1)


accuracy = np.mean(y_pred_classes == y_true)
precision = classification_report(y_true, y_pred_classes, output_dict=True)['weighted avg']['precision']
recall = classification_report(y_true, y_pred_classes, output_dict=True)['weighted avg']['recall']
f1_score = classification_report(y_true, y_pred_classes, output_dict=True)['weighted avg']['f1-score']


conf_matrix = confusion_matrix(y_true, y_pred_classes)
conf_matrix = confusion_matrix(y_true, y_pred_classes)
plt.figure(figsize=(10, 7))
sns.heatmap(conf_matrix, annot=True, fmt='d', cmap="Blues")
plt.xlabel('Predicted')
plt.ylabel('True')
plt.title('Confusion Matrix for CIFAR-10')
plt.show()


auc_score = roc_auc_score(y_test, y_pred, multi_class='ovr')
print(f"AUC Score: {auc_score:.4f}")
```

```
313/313 ━━━━━━━━━━━━━━━━━━━━ 5s 14ms/step
```



```
AUC Score: 0.9529
```