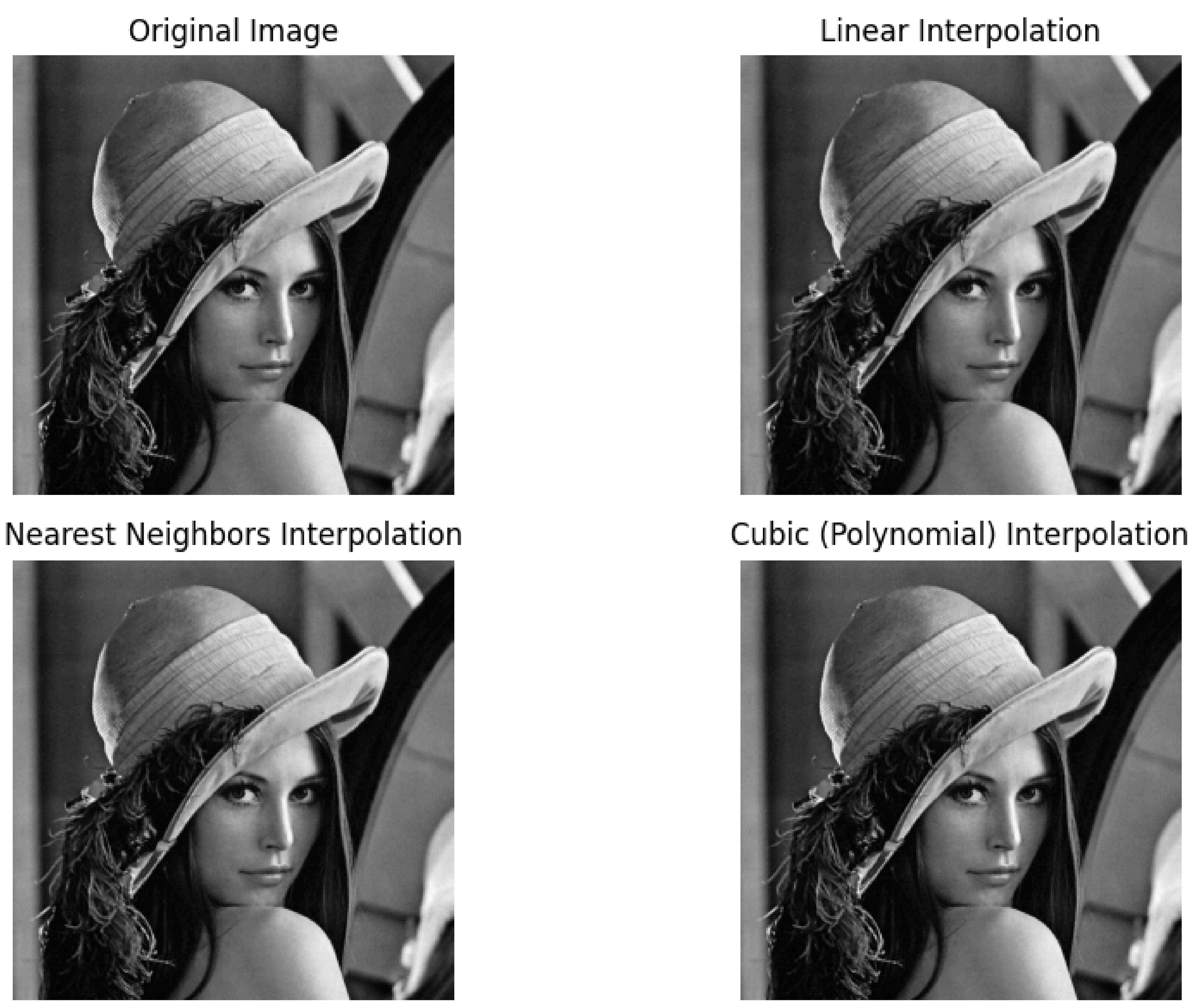
```
# TASK 1

import cv2
import matplotlib.pyplot as plt


image_path = '/content/cvlab1img.jpg'
image = cv2.imread(image_path)



plt.figure(figsize=(10, 6))
original_image_rgb = cv2.cvtColor(image, cv2.COLOR_BGR2RGB)
plt.subplot(221)
plt.imshow(original_image_rgb)
plt.title("Original Image")
plt.axis('off')

# Resize dimensions
scale_factor = 2  # Scale up by 2x
new_width = int(image.shape[1] * scale_factor)
new_height = int(image.shape[0] * scale_factor)
new_dimensions = (new_width, new_height)

# 1.1.1 Linear Interpolation
resized_linear = cv2.resize(image, new_dimensions, interpolation=cv2.INTER_LINEAR)
resized_linear_rgb = cv2.cvtColor(resized_linear, cv2.COLOR_BGR2RGB)

plt.subplot(222)
plt.imshow(resized_linear_rgb)
plt.title("Linear Interpolation")
plt.axis('off')

# 1.1.2 Nearest Neighbors Interpolation
resized_nearest = cv2.resize(image, new_dimensions, interpolation=cv2.INTER_NEAREST)
resized_nearest_rgb = cv2.cvtColor(resized_nearest, cv2.COLOR_BGR2RGB)

plt.subplot(223)
plt.imshow(resized_nearest_rgb)
plt.title("Nearest Neighbors Interpolation")
plt.axis('off')

# 1.1.3 Polynomial Interpolation (Cubic)
resized_cubic = cv2.resize(image, new_dimensions, interpolation=cv2.INTER_CUBIC)
resized_cubic_rgb = cv2.cvtColor(resized_cubic, cv2.COLOR_BGR2RGB)

plt.subplot(224)
plt.imshow(resized_cubic_rgb)
plt.title("Cubic (Polynomial) Interpolation")
plt.axis('off')

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



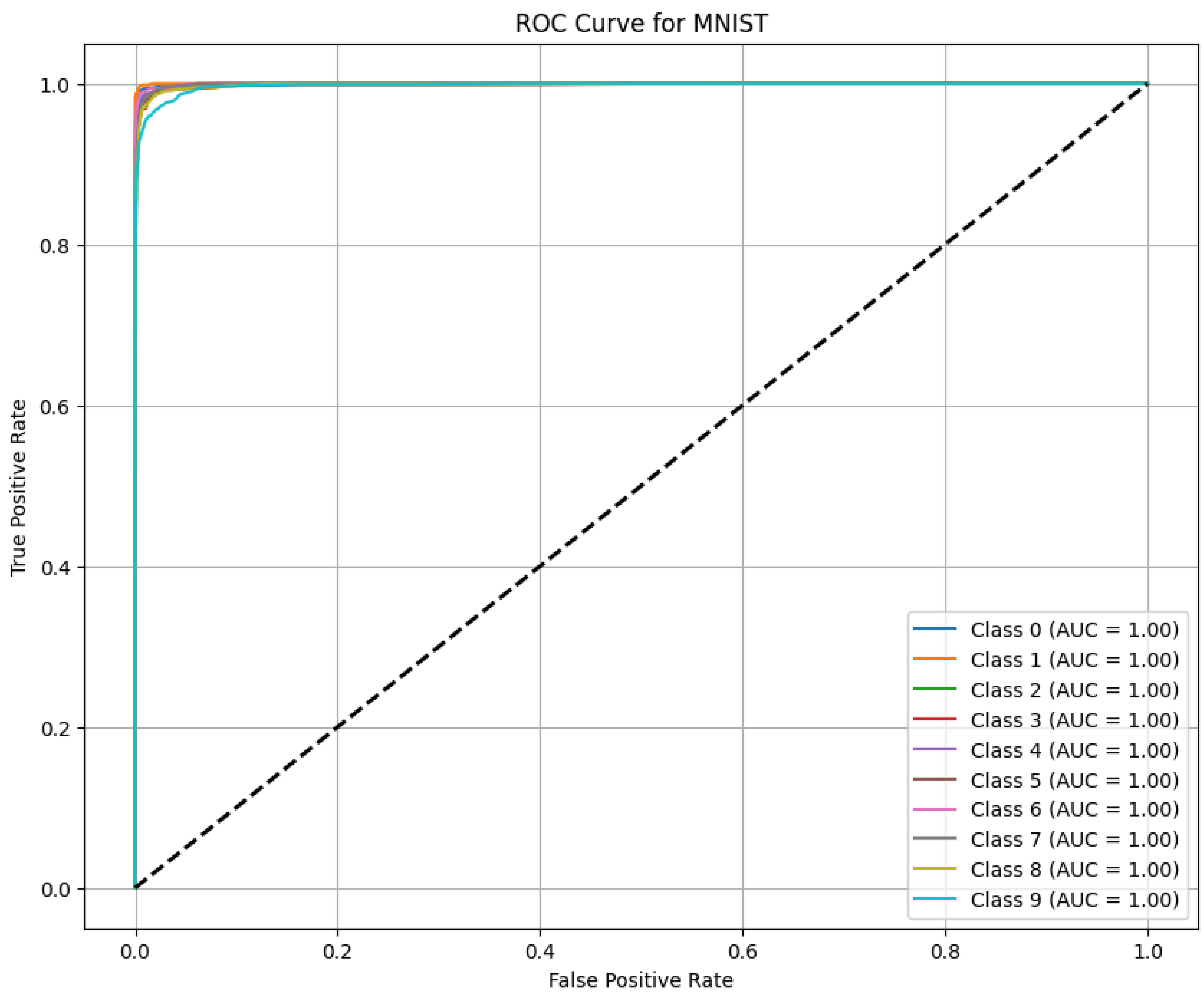

```
# TASK 1.2

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

# Display original image
plt.figure(figsize=(12, 8))
plt.subplot(231)
plt.imshow(image_rgb)
plt.title("Original Image")
plt.axis('off')

# 1.2.1 Box Blurring
# Kernel size (e.g., 5x5)
box_blurred = cv2.blur(image, (5, 5))
box_blurred_rgb = cv2.cvtColor(box_blurred, cv2.COLOR_BGR2RGB)

plt.subplot(232)
plt.imshow(box_blurred_rgb)
plt.title("Box Blurring")
plt.axis('off')

# 1.2.2 Gaussian Blurring
# Kernel size (e.g., 5x5) and standard deviation
gaussian_blurred = cv2.GaussianBlur(image, (5, 5), 0)
gaussian_blurred_rgb = cv2.cvtColor(gaussian_blurred, cv2.COLOR_BGR2RGB)
```

```
plt.subplot(233)
plt.imshow(gaussian_blurred_rgb)
plt.title("Gaussian Blurring")
plt.axis('off')

# 1.2.3 Adaptive Blurring (Bilateral Filter)
# Diameter, sigmaColor, sigmaSpace (to preserve edges while blurring)
adaptive_blurred = cv2.bilateralFilter(image, 9, 75, 75)
adaptive_blurred_rgb = cv2.cvtColor(adaptive_blurred, cv2.COLOR_BGR2RGB)

plt.subplot(234)
plt.imshow(adaptive_blurred_rgb)
plt.title("Adaptive Blurring (Bilateral)")
plt.axis('off')

# Additional: Apply a stronger blur for comparison
strong_gaussian_blurred = cv2.GaussianBlur(image, (15, 15), 0)
strong_gaussian_rgb = cv2.cvtColor(strong_gaussian_blurred, cv2.COLOR_BGR2RGB)

plt.subplot(235)
plt.imshow(strong_gaussian_rgb)
plt.title("Strong Gaussian Blur")
plt.axis('off')

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


```
# TASK 2
# USING DECISION TREE/RANDOM FOREST

import numpy as np
import pandas as pd
from sklearn.model_selection import train_test_split, KFold, cross_val_score
from sklearn.ensemble import RandomForestClassifier
from sklearn.tree import DecisionTreeClassifier
from sklearn.metrics import (
    accuracy_score, precision_score, recall_score, f1_score,
    confusion_matrix, roc_curve, auc, classification_report
)
from tensorflow.keras.datasets import mnist
from sklearn.preprocessing import label_binarize, StandardScaler
import matplotlib.pyplot as plt


(X_train_full, y_train_full), (X_test, y_test) = mnist.load_data()


X_train_full = X_train_full.reshape(-1, 784) / 255.0
X_test = X_test.reshape(-1, 784) / 255.0


X_train, X_val, y_train, y_val = train_test_split(X_train_full, y_train_full, test_size=0.2, random_state=42)


y_binarized = label_binarize(y_val, classes=np.unique(y_val))


model = RandomForestClassifier(n_estimators=100, random_state=42)


kf = KFold(n_splits=5, shuffle=True, random_state=42)
cv_scores = cross_val_score(model, X_train, y_train, cv=kf, scoring="accuracy")


model.fit(X_train, y_train)


y_pred = model.predict(X_test)
y_proba = model.predict_proba(X_test)


accuracy = accuracy_score(y_test, y_pred)
precision = precision_score(y_test, y_pred, average="macro")
```

```
recall = recall_score(y_test, y_pred, average="macro")
f1 = f1_score(y_test, y_pred, average="macro")
conf_matrix = confusion_matrix(y_test, y_pred)


fpr = {}
tpr = {}
roc_auc = {}
for i in range(10):
    fpr[i], tpr[i], _ = roc_curve(label_binarize(y_test, classes=np.unique(y_test))[:, i], y_proba[:, i])
    roc_auc[i] = auc(fpr[i], tpr[i])

# Plot ROC Curve
plt.figure(figsize=(10, 8))
for i in range(10):
    plt.plot(fpr[i], tpr[i], label=f"Class {i} (AUC = {roc_auc[i]:.2f})")
plt.plot([0, 1], [0, 1], "k--", lw=2)
plt.xlabel("False Positive Rate")
plt.ylabel("True Positive Rate")
plt.title("ROC Curve for MNIST")
plt.legend(loc="best")
plt.grid()
plt.show()

# Results Summary
results_summary = pd.DataFrame({
    "Metric": ["Accuracy", "Precision", "Recall", "F1-Score", "Mean CV Accuracy"],
    "Value": [accuracy, precision, recall, f1, cv_scores.mean()]
})
print("\nResults Summary:")
print(results_summary)

print("\nConfusion Matrix:")
print(conf_matrix)

print("\nClassification Report:")
print(classification_report(y_test, y_pred))
```

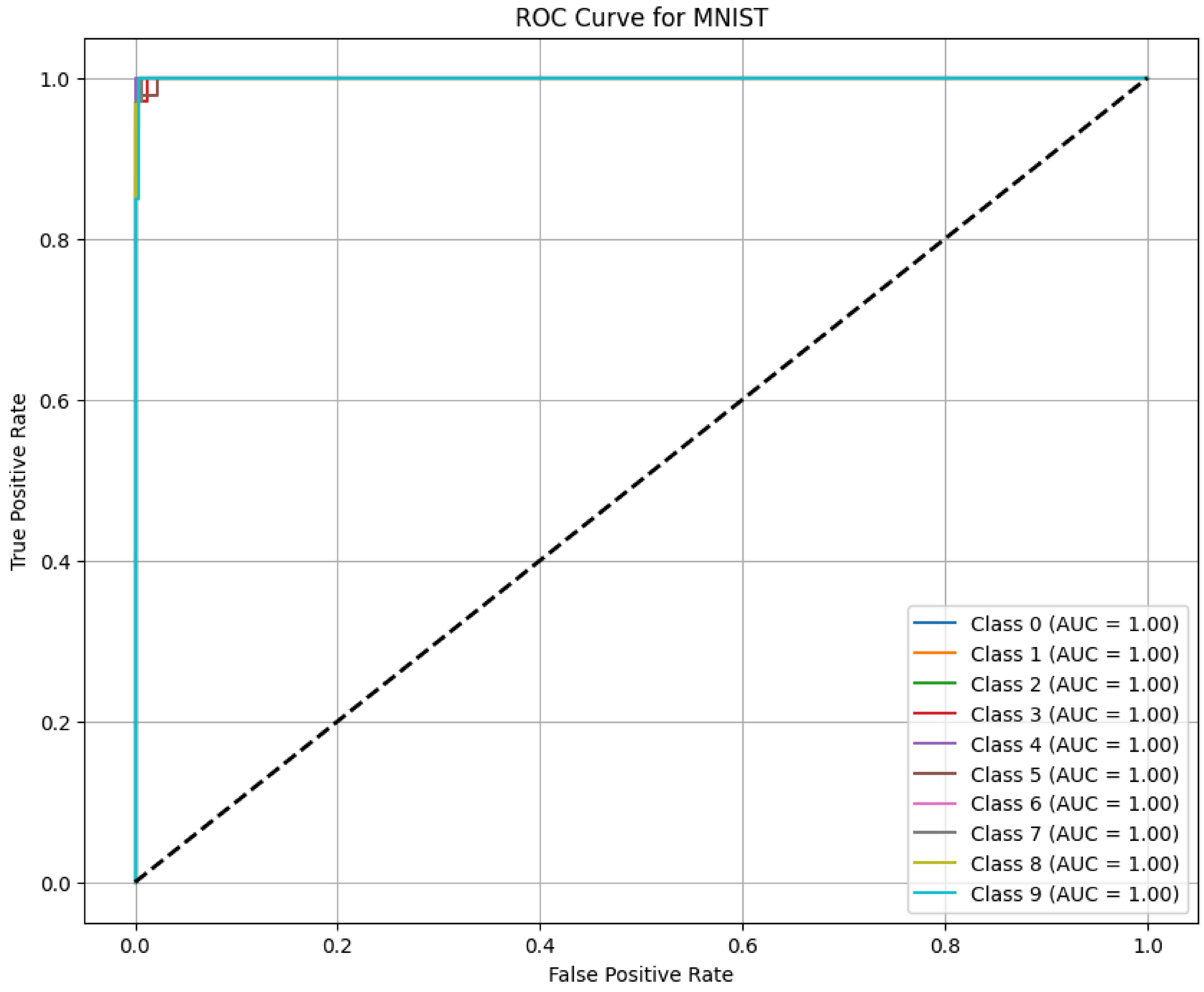
```
Downloading data from https://storage.googleapis.com/tensorflow/tf-keras-datasets/mnist.npz
11490434/11490434 ━━━━━━━━━━━━━━━━━━━━ 0s 0us/step
```




```
Results Summary:
             Metric     Value
0          Accuracy  0.968700
1         Precision  0.968596
2            Recall  0.968458
3          F1-Score  0.968500
4  Mean CV Accuracy  0.964875

Confusion Matrix:
[[ 971    0    1    0    0    2    2    1    3    0]
 [   0 1123    3    2    0    2    2    1    2    0]
 [   6    0  998    3    4    0    4    8    9    0]
 [   0    0   13  971    0    5    0    9   10    2]
 [   1    0    3    0  956    0    4    0    2   16]
 [   4    0    1   11    3  857    7    1    4    4]
 [   8    3    0    0    2    4  938    0    3    0]
 [   1    3   19    4    1    0    0  985    4   11]
 [   3    0    2   13    5    4    2    4  931   10]
 [   5    6    4   10   13    3    2    5    4  957]]

Classification Report:
              precision    recall  f1-score   support

           0       0.97      0.99      0.98       980
           1       0.99      0.99      0.99      1135
           2       0.96      0.97      0.96      1032
           3       0.96      0.96      0.96      1010
           4       0.97      0.97      0.97       982
           5       0.98      0.96      0.97       892
           6       0.98      0.98      0.98       958
           7       0.97      0.96      0.96      1028
           8       0.96      0.96      0.96       974
           9       0.96      0.95      0.95      1009

    accuracy                           0.97     10000
   macro avg       0.97      0.97      0.97     10000
weighted avg       0.97      0.97      0.97     10000
```

```python
# USING ANN

from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense
from tensorflow.keras.utils import to_categorical
import matplotlib.pyplot as plt

from sklearn.datasets import load_digits
from sklearn.model_selection import train_test_split, KFold, cross_val_score
from sklearn.preprocessing import StandardScaler, label_binarize

# Normalize features
scaler = StandardScaler()
X = scaler.fit_transform(X)


y_binarized = to_categorical(y)

# Split the data into train and test sets (80-20 split)
X_train, X_test, y_train, y_test = train_test_split(X, y_binarized, test_size=0.2, random_state=42)

# ANN model
def build_model():
    model = Sequential()
    model.add(Dense(64, activation='relu', input_shape=(X_train.shape[1],)))
    model.add(Dense(32, activation='relu'))
    model.add(Dense(y_binarized.shape[1], activation='softmax'))  # Output layer for multi-class classification
    model.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['accuracy'])
    return model


kf = KFold(n_splits=5, shuffle=True, random_state=42)
fold_accuracies = []


for train_index, val_index in kf.split(X_train):
    X_fold_train, X_fold_val = X_train[train_index], X_train[val_index]
    y_fold_train, y_fold_val = y_train[train_index], y_train[val_index]


    model = build_model()
    model.fit(X_fold_train, y_fold_train, epochs=20, batch_size=32, verbose=0)


    loss, accuracy = model.evaluate(X_fold_val, y_fold_val, verbose=0)
    fold_accuracies.append(accuracy)


model = build_model()
model.fit(X_train, y_train, epochs=20, batch_size=32, verbose=0)


y_pred_proba = model.predict(X_test)
y_pred = np.argmax(y_pred_proba, axis=1)
y_test_labels = np.argmax(y_test, axis=1)


accuracy = accuracy_score(y_test_labels, y_pred)
precision = precision_score(y_test_labels, y_pred, average='macro')
recall = recall_score(y_test_labels, y_pred, average='macro')
f1 = f1_score(y_test_labels, y_pred, average='macro')
conf_matrix = confusion_matrix(y_test_labels, y_pred)


fpr = {}
tpr = {}
roc_auc = {}
for i in range(y_binarized.shape[1]):
    fpr[i], tpr[i], _ = roc_curve(y_test[:, i], y_pred_proba[:, i])
    roc_auc[i] = auc(fpr[i], tpr[i])

# Plot ROC Curve
plt.figure(figsize=(10, 8))
for i in range(y_binarized.shape[1]):
    plt.plot(fpr[i], tpr[i], label=f"Class {i} (AUC = {roc_auc[i]:.2f})")
plt.plot([0, 1], [0, 1], "k--", lw=2)
plt.xlabel("False Positive Rate")
plt.ylabel("True Positive Rate")
plt.title("ROC Curve for MNIST")
plt.legend(loc="best")
plt.grid()
plt.show()

# Results Summary
print("\nResults Summary:")
print(f"Accuracy: {accuracy:.4f}")
print(f"Precision: {precision:.4f}")
print(f"Recall: {recall:.4f}")
print(f"F1-Score: {f1:.4f}")
print(f"Mean CV Accuracy: {np.mean(fold_accuracies):.4f}")

print("\nConfusion Matrix:")
print(conf_matrix)

print("\nClassification Report:")
print(classification_report(y_test_labels, y_pred))
```

```
/usr/local/lib/python3.11/dist-packages/keras/src/layers/core/dense.py:87: UserWarning: Do not pass an `input_shape`/`input_dim` argument to a layer. When using Sequential models, prefer
  super().__init__(activity_regularizer=activity_regularizer, **kwargs)
12/12 ━━━━━━━━━━━━━━━━━━━━ 0s 5ms/step
```




```
Results Summary:
Accuracy: 0.9778
Precision: 0.9779
Recall: 0.9771
F1-Score: 0.9773
Mean CV Accuracy: 0.9652

Confusion Matrix:
[[32  0  1  0  0  0  0  0  0  0]
 [ 0 27  1  0  0  0  0  0  0  0]
 [ 0  0 33  0  0  0  0  0  0  0]
 [ 0  0  0 33  0  1  0  0  0  0]
 [ 0  0  0  0 46  0  0  0  0  0]
 [ 0  0  0  0  0 45  1  0  0  1]
 [ 1  0  0  0  0  0 34  0  0  0]
 [ 0  0  0  0  0  0  0 33  0  1]
 [ 0  1  0  0  0  0  0  0 29  0]
 [ 0  0  0  0  0  0  0  0  0 40]]

Classification Report:
              precision    recall  f1-score   support

           0       0.97      0.97      0.97        33
           1       0.96      0.96      0.96        28
           2       0.94      1.00      0.97        33
           3       1.00      0.97      0.99        34
           4       1.00      1.00      1.00        46
           5       0.98      0.96      0.97        47
           6       0.97      0.97      0.97        35
           7       1.00      0.97      0.99        34
           8       1.00      0.97      0.98        30
           9       0.95      1.00      0.98        40

    accuracy                           0.98       360
   macro avg       0.98      0.98      0.98       360
weighted avg       0.98      0.98      0.98       360
```