```
import cv2
import numpy as np
import matplotlib.pyplot as plt
from google.colab import files
from PIL import Image
import os

# Upload an image
uploaded = files.upload()

# Get the uploaded filename
image_filename = list(uploaded.keys())[0]

# Read the image
image = cv2.imread(image_filename)

# Convert to RGB (OpenCV loads in BGR format)
image = cv2.cvtColor(image, cv2.COLOR_BGR2RGB)

# Display original image
plt.figure(figsize=(8, 4))
plt.imshow(image)
plt.title("Original Image")
plt.axis("off")
plt.show()
```

Choose Files sample_640×426.bmp

- **sample_640×426.bmp**(image/bmp) - 818058 bytes, last modified: 2/13/2025 - 100% done

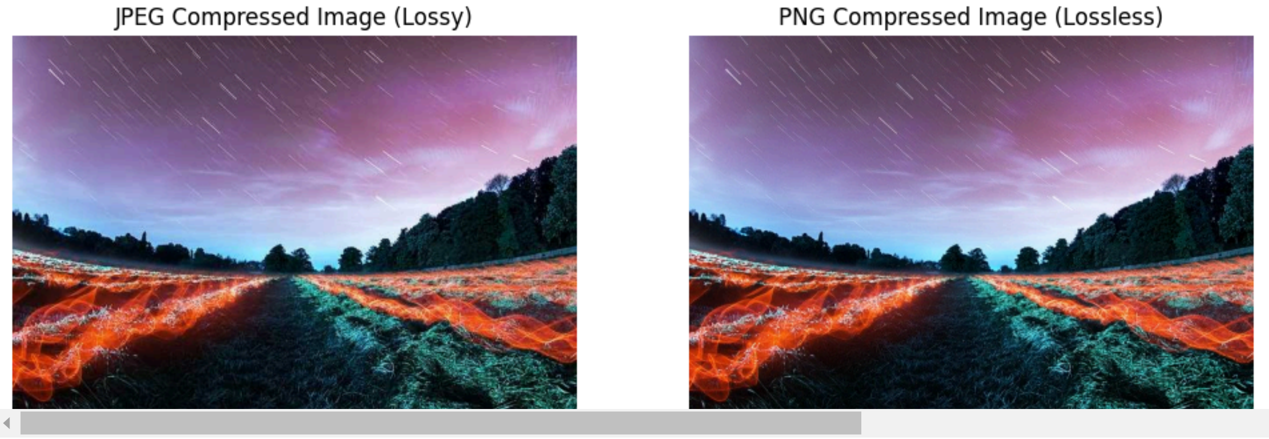
```
Saving sample_640×426.bmp to sample_640×426.bmp
```

Original Image

```
# Save as JPEG (Lossy Compression)
jpeg_filename = "compressed_image.jpg"
cv2.imwrite(jpeg_filename, image, [int(cv2.IMWRITE_JPEG_QUALITY), 50])  # 50% quality

# Save as PNG (Lossless Compression)
png_filename = "compressed_image.png"
cv2.imwrite(png_filename, image, [int(cv2.IMWRITE_PNG_COMPRESSION), 9])  # Max compression

# Check file sizes
original_size = os.path.getsize(image_filename) / 1024  # KB
jpeg_size = os.path.getsize(jpeg_filename) / 1024  # KB
png_size = os.path.getsize(png_filename) / 1024  # KB

print(f"Original Image Size: {original_size:.2f} KB")
print(f"JPEG Compressed Size: {jpeg_size:.2f} KB")
print(f"PNG Compressed Size: {png_size:.2f} KB")

# Display compressed images
compressed_jpeg = Image.open(jpeg_filename)
compressed_png = Image.open(png_filename)

fig, ax = plt.subplots(1, 2, figsize=(12, 6))

ax[0].imshow(compressed_jpeg)
ax[0].set_title("JPEG Compressed Image (Lossy)")
ax[0].axis("off")

ax[1].imshow(compressed_png)
ax[1].set_title("PNG Compressed Image (Lossless)")
ax[1].axis("off")
```

```
plt.show()
```

```
Original Image Size: 798.88 KB
JPEG Compressed Size: 38.76 KB
PNG Compressed Size: 509.95 KB
```

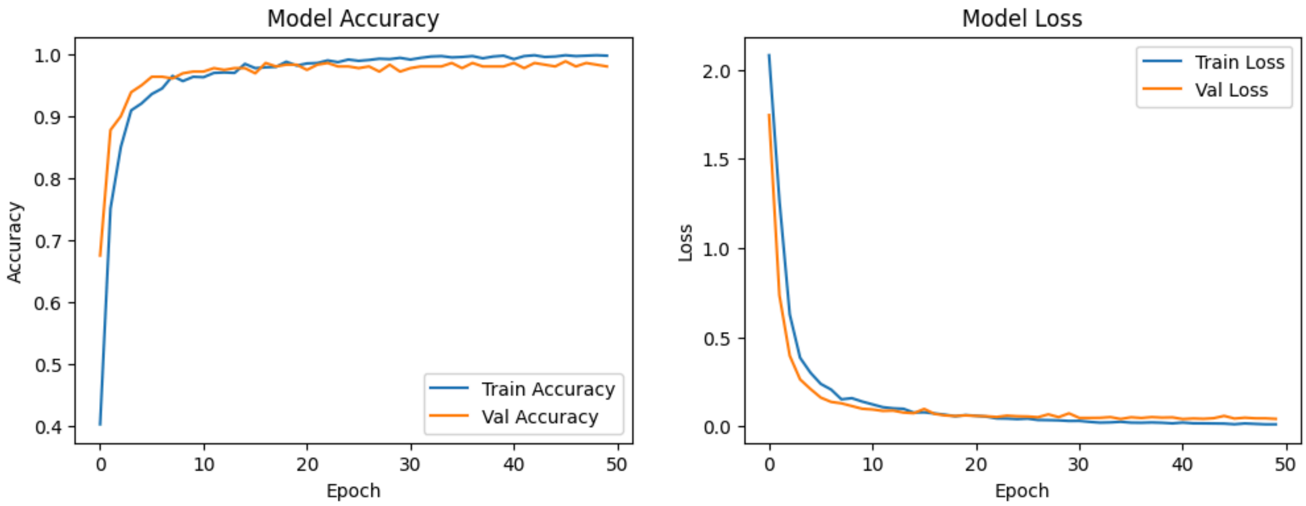

JPEG Compressed Image (Lossy)
PNG Compressed Image (Lossless)

```
import numpy as np
import tensorflow as tf
import matplotlib.pyplot as plt
from sklearn.datasets import load_digits
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import OneHotEncoder
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Conv2D, MaxPooling2D, Flatten, Dense, Dropout
from tensorflow.keras.utils import to_categorical
from tensorflow.keras.datasets import cifar10
from sklearn.metrics import classification_report, confusion_matrix, roc_curve, auc
import seaborn as sns

# Function to plot training history
def plot_history(history):
    plt.figure(figsize=(12, 4))

    plt.subplot(1, 2, 1)
    plt.plot(history.history['accuracy'], label='Train Accuracy')
    plt.plot(history.history['val_accuracy'], label='Val Accuracy')
    plt.title('Model Accuracy')
    plt.xlabel('Epoch')
    plt.ylabel('Accuracy')
    plt.legend()

    plt.subplot(1, 2, 2)
    plt.plot(history.history['loss'], label='Train Loss')
    plt.plot(history.history['val_loss'], label='Val Loss')
    plt.title('Model Loss')
    plt.xlabel('Epoch')
    plt.ylabel('Loss')
    plt.legend()

    plt.show()

# Function to evaluate model
def evaluate_model(model, X_test, y_test):
    y_pred = np.argmax(model.predict(X_test), axis=1)
    y_true = np.argmax(y_test, axis=1)

    print("\nClassification Report:\n", classification_report(y_true, y_pred))

    # Confusion Matrix
    plt.figure(figsize=(8, 6))
    cm = confusion_matrix(y_true, y_pred)
    sns.heatmap(cm, annot=True, fmt='d', cmap='Blues', xticklabels=range(10), yticklabels=range(10))
    plt.xlabel("Predicted")
    plt.ylabel("Actual")
    plt.title("Confusion Matrix")
    plt.show()

    # ROC & AUC
    fpr, tpr, _ = roc_curve(y_test.ravel(), model.predict(X_test).ravel())
    roc_auc = auc(fpr, tpr)

    plt.figure(figsize=(6, 6))
    plt.plot(fpr, tpr, color='blue', label=f'ROC curve (AUC = {roc_auc:.2f})')
    plt.plot([0, 1], [0, 1], color='gray', linestyle='--')
    plt.xlabel("False Positive Rate")
    plt.ylabel("True Positive Rate")
    plt.title("ROC Curve")
    plt.legend()
    plt.show()
```

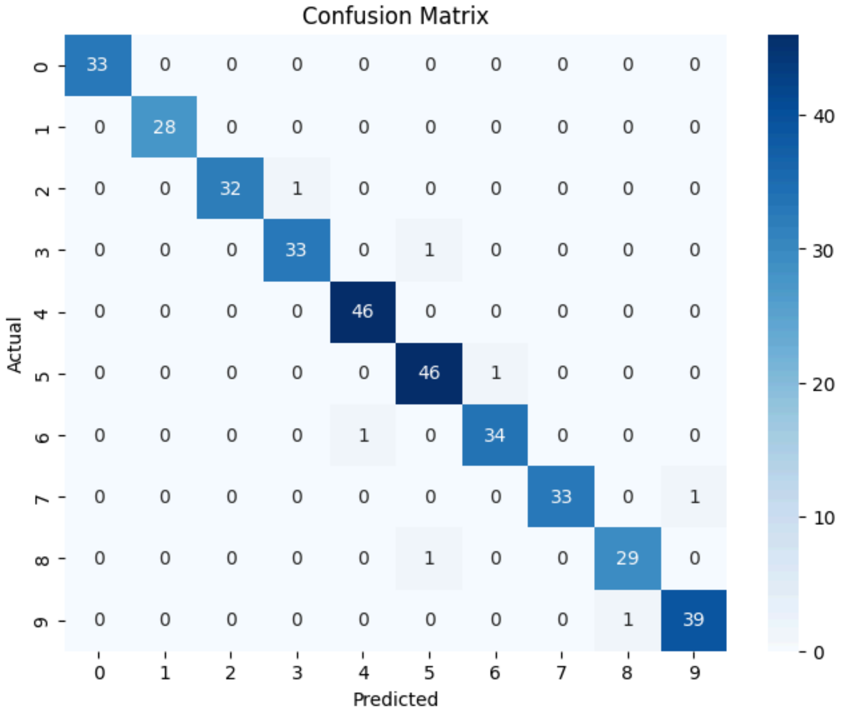
```
# Load Digits dataset
digits = load_digits()
X, y = digits.images, digits.target

# Normalize pixel values
X = X / 16.0

# Reshape to (n_samples, 8, 8, 1) for CNN
X = X.reshape(-1, 8, 8, 1)

# One-hot encode labels
y = to_categorical(y, num_classes=10)

# Split dataset into 80% train, 20% test
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Define CNN model
model_digits = Sequential([
    Conv2D(32, (3, 3), activation='relu', input_shape=(8, 8, 1)),
    MaxPooling2D((2, 2)),
    Flatten(),
    Dense(128, activation='relu'),
    Dropout(0.3),
    Dense(10, activation='softmax')
])

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

# Train model
history_digits = model_digits.fit(X_train, y_train, epochs=50, batch_size=32, validation_data=(X_test, y_test), verbose=1)

# Evaluate
plot_history(history_digits)
evaluate_model(model_digits, X_test, y_test)
```

```
/usr/local/lib/python3.11/dist-packages/keras/src/layers/convolutional/base_conv.py:107: UserWarning: Do not pass an `input_shape
  super().__init__(activity_regularizer=activity_regularizer, **kwargs)
Epoch 1/50
45/45 ━━━━━━━━━━━━━━━━━━━━ 6s 50ms/step - accuracy: 0.2437 - loss: 2.2152 - val_accuracy: 0.6750 - val_loss: 1.7456
Epoch 2/50
45/45 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - accuracy: 0.6904 - loss: 1.4999 - val_accuracy: 0.8778 - val_loss: 0.7349
Epoch 3/50
45/45 ━━━━━━━━━━━━━━━━━━━━ 0s 5ms/step - accuracy: 0.8356 - loss: 0.7102 - val_accuracy: 0.9000 - val_loss: 0.3954
Epoch 4/50
45/45 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - accuracy: 0.8931 - loss: 0.4288 - val_accuracy: 0.9389 - val_loss: 0.2627
Epoch 5/50
45/45 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - accuracy: 0.9193 - loss: 0.3117 - val_accuracy: 0.9500 - val_loss: 0.2091
Epoch 6/50
45/45 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - accuracy: 0.9362 - loss: 0.2464 - val_accuracy: 0.9639 - val_loss: 0.1606
Epoch 7/50
45/45 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - accuracy: 0.9389 - loss: 0.2023 - val_accuracy: 0.9639 - val_loss: 0.1364
Epoch 8/50
45/45 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - accuracy: 0.9723 - loss: 0.1409 - val_accuracy: 0.9611 - val_loss: 0.1284
Epoch 9/50
45/45 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - accuracy: 0.9587 - loss: 0.1551 - val_accuracy: 0.9694 - val_loss: 0.1133
Epoch 10/50
45/45 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - accuracy: 0.9727 - loss: 0.1249 - val_accuracy: 0.9722 - val_loss: 0.0980
Epoch 11/50
45/45 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - accuracy: 0.9624 - loss: 0.1268 - val_accuracy: 0.9722 - val_loss: 0.0940
Epoch 12/50
45/45 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - accuracy: 0.9626 - loss: 0.1260 - val_accuracy: 0.9778 - val_loss: 0.0860
Epoch 13/50
45/45 ━━━━━━━━━━━━━━━━━━━━ 0s 6ms/step - accuracy: 0.9731 - loss: 0.0951 - val_accuracy: 0.9750 - val_loss: 0.0876
Epoch 14/50
45/45 ━━━━━━━━━━━━━━━━━━━━ 1s 6ms/step - accuracy: 0.9727 - loss: 0.0971 - val_accuracy: 0.9778 - val_loss: 0.0765
Epoch 15/50
45/45 ━━━━━━━━━━━━━━━━━━━━ 0s 6ms/step - accuracy: 0.9881 - loss: 0.0691 - val_accuracy: 0.9778 - val_loss: 0.0731
Epoch 16/50
45/45 ━━━━━━━━━━━━━━━━━━━━ 0s 6ms/step - accuracy: 0.9804 - loss: 0.0772 - val_accuracy: 0.9694 - val_loss: 0.0970
Epoch 17/50
45/45 ━━━━━━━━━━━━━━━━━━━━ 0s 7ms/step - accuracy: 0.9730 - loss: 0.0788 - val_accuracy: 0.9861 - val_loss: 0.0692
Epoch 18/50
45/45 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - accuracy: 0.9732 - loss: 0.0773 - val_accuracy: 0.9806 - val_loss: 0.0609
Epoch 19/50
45/45 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - accuracy: 0.9909 - loss: 0.0537 - val_accuracy: 0.9833 - val_loss: 0.0572
Epoch 20/50
45/45 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - accuracy: 0.9805 - loss: 0.0637 - val_accuracy: 0.9833 - val_loss: 0.0609
Epoch 21/50
45/45 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - accuracy: 0.9852 - loss: 0.0673 - val_accuracy: 0.9750 - val_loss: 0.0578
Epoch 22/50
45/45 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - accuracy: 0.9873 - loss: 0.0532 - val_accuracy: 0.9833 - val_loss: 0.0559
Epoch 23/50
45/45 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - accuracy: 0.9912 - loss: 0.0443 - val_accuracy: 0.9861 - val_loss: 0.0518
Epoch 24/50
45/45 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - accuracy: 0.9888 - loss: 0.0398 - val_accuracy: 0.9806 - val_loss: 0.0592
Epoch 25/50
45/45 ━━━━━━━━━━━━━━━━━━━━ 0s 5ms/step - accuracy: 0.9927 - loss: 0.0398 - val_accuracy: 0.9806 - val_loss: 0.0556
Epoch 26/50
45/45 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - accuracy: 0.9928 - loss: 0.0383 - val_accuracy: 0.9778 - val_loss: 0.0545
Epoch 27/50
45/45 ━━━━━━━━━━━━━━━━━━━━ 0s 5ms/step - accuracy: 0.9888 - loss: 0.0420 - val_accuracy: 0.9806 - val_loss: 0.0506
Epoch 28/50
45/45 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - accuracy: 0.9915 - loss: 0.0335 - val_accuracy: 0.9722 - val_loss: 0.0666
Epoch 29/50
45/45 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - accuracy: 0.9905 - loss: 0.0367 - val_accuracy: 0.9833 - val_loss: 0.0509
Epoch 30/50
45/45 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - accuracy: 0.9944 - loss: 0.0292 - val_accuracy: 0.9722 - val_loss: 0.0723
Epoch 31/50
45/45 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - accuracy: 0.9910 - loss: 0.0309 - val_accuracy: 0.9778 - val_loss: 0.0465
Epoch 32/50
45/45 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - accuracy: 0.9965 - loss: 0.0237 - val_accuracy: 0.9806 - val_loss: 0.0466
Epoch 33/50
45/45 ━━━━━━━━━━━━━━━━━━━━ 0s 5ms/step - accuracy: 0.9979 - loss: 0.0188 - val_accuracy: 0.9806 - val_loss: 0.0471
Epoch 34/50
45/45 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - accuracy: 0.9993 - loss: 0.0161 - val_accuracy: 0.9806 - val_loss: 0.0512
Epoch 35/50
45/45 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - accuracy: 0.9941 - loss: 0.0264 - val_accuracy: 0.9861 - val_loss: 0.0409
Epoch 36/50
45/45 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - accuracy: 0.9980 - loss: 0.0193 - val_accuracy: 0.9778 - val_loss: 0.0502
Epoch 37/50
45/45 ━━━━━━━━━━━━━━━━━━━━ 0s 5ms/step - accuracy: 0.9988 - loss: 0.0184 - val_accuracy: 0.9861 - val_loss: 0.0463
Epoch 38/50
45/45 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - accuracy: 0.9904 - loss: 0.0262 - val_accuracy: 0.9806 - val_loss: 0.0513
Epoch 39/50
45/45 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - accuracy: 0.9962 - loss: 0.0182 - val_accuracy: 0.9806 - val_loss: 0.0484
Epoch 40/50
45/45 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - accuracy: 0.9982 - loss: 0.0176 - val_accuracy: 0.9806 - val_loss: 0.0497
Epoch 41/50
45/45 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - accuracy: 0.9925 - loss: 0.0197 - val_accuracy: 0.9861 - val_loss: 0.0404
Epoch 42/50
45/45 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - accuracy: 0.9980 - loss: 0.0150 - val_accuracy: 0.9778 - val_loss: 0.0437
Epoch 43/50
45/45 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - accuracy: 0.9993 - loss: 0.0146 - val_accuracy: 0.9861 - val_loss: 0.0418
Epoch 44/50
45/45 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - accuracy: 0.9988 - loss: 0.0122 - val_accuracy: 0.9833 - val_loss: 0.0452
```

```
Epoch 45/50
45/45 ━━━━━━━━━━━━━━━━━━━━ 0s 5ms/step - accuracy: 0.9974 - loss: 0.0118 - val_accuracy: 0.9806 - val_loss: 0.0578
Epoch 46/50
45/45 ━━━━━━━━━━━━━━━━━━━━ 0s 5ms/step - accuracy: 0.9993 - loss: 0.0110 - val_accuracy: 0.9889 - val_loss: 0.0432
Epoch 47/50
45/45 ━━━━━━━━━━━━━━━━━━━━ 0s 5ms/step - accuracy: 0.9980 - loss: 0.0136 - val_accuracy: 0.9806 - val_loss: 0.0479
Epoch 48/50
45/45 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - accuracy: 0.9980 - loss: 0.0138 - val_accuracy: 0.9861 - val_loss: 0.0445
Epoch 49/50
45/45 ━━━━━━━━━━━━━━━━━━━━ 0s 5ms/step - accuracy: 0.9994 - loss: 0.0106 - val_accuracy: 0.9833 - val_loss: 0.0444
Epoch 50/50
45/45 ━━━━━━━━━━━━━━━━━━━━ 0s 5ms/step - accuracy: 0.9980 - loss: 0.0112 - val_accuracy: 0.9806 - val_loss: 0.0414
```

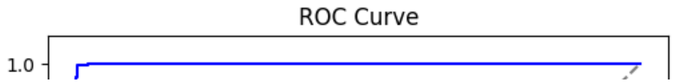


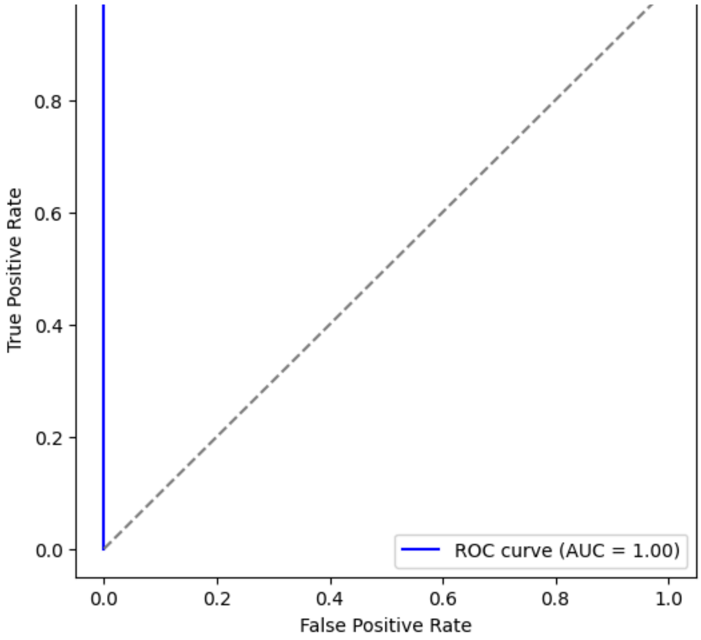
```
12/12 ━━━━━━━━━━━━━━━━━━━━ 0s 21ms/step

Classification Report:
              precision    recall  f1-score   support

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

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



	0	1	2	3	4	5	6	7	8	9
0	33	0	0	0	0	0	0	0	0	0
1	0	28	0	0	0	0	0	0	0	0
2	0	0	32	1	0	0	0	0	0	0
3	0	0	0	33	0	1	0	0	0	0
4	0	0	0	0	46	0	0	0	0	0
5	0	0	0	0	0	46	1	0	0	0
6	0	0	0	0	1	0	34	0	0	0
7	0	0	0	0	0	0	0	33	0	1
8	0	0	0	0	0	1	0	0	29	0
9	0	0	0	0	0	0	0	0	1	39

```
12/12 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step
```




0.8
0.6
0.4
0.2
0.0
True Positive Rate
ROC curve (AUC = 1.00)
0.0
0.2
0.4
0.6
0.8
1.0
False Positive Rate

```
# Load CIFAR-10 dataset
(X_train, y_train), (X_test, y_test) = cifar10.load_data()

# Normalize images (0 to 1 range)
X_train, X_test = X_train / 255.0, X_test / 255.0

# One-hot encode labels
y_train, y_test = to_categorical(y_train, num_classes=10), to_categorical(y_test, num_classes=10)

# Split dataset into 80% train, 20% test
X_train, X_val, y_train, y_val = train_test_split(X_train, y_train, test_size=0.2, random_state=42)

# Define CNN model
model_cifar = Sequential([
    Conv2D(32, (3, 3), activation='relu', input_shape=(32, 32, 3)),
    MaxPooling2D((2, 2)),
    Conv2D(64, (3, 3), activation='relu'),
    MaxPooling2D((2, 2)),
    Flatten(),
    Dense(128, activation='relu'),
    Dropout(0.3),
    Dense(10, activation='softmax')
])

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

# Train model
history_cifar = model_cifar.fit(X_train, y_train, epochs=50, batch_size=64, validation_data=(X_val, y_val), verbose=1)

# Evaluate
plot_history(history_cifar)
evaluate_model(model_cifar, X_test, y_test)
```

```
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 `input_shape
  super().__init__(activity_regularizer=activity_regularizer, **kwargs)
Epoch 1/50
625/625 ━━━━━━━━━━━━━━━━━━━━ 8s 6ms/step - accuracy: 0.3265 - loss: 1.8250 - val_accuracy: 0.5180 - val_loss: 1.3385
Epoch 2/50
625/625 ━━━━━━━━━━━━━━━━━━━━ 7s 5ms/step - accuracy: 0.5187 - loss: 1.3408 - val_accuracy: 0.5987 - val_loss: 1.1316
Epoch 3/50
625/625 ━━━━━━━━━━━━━━━━━━━━ 5s 4ms/step - accuracy: 0.5813 - loss: 1.1836 - val_accuracy: 0.6176 - val_loss: 1.0773
Epoch 4/50
625/625 ━━━━━━━━━━━━━━━━━━━━ 3s 4ms/step - accuracy: 0.6202 - loss: 1.0830 - val_accuracy: 0.6452 - val_loss: 1.0091
Epoch 5/50
625/625 ━━━━━━━━━━━━━━━━━━━━ 5s 5ms/step - accuracy: 0.6525 - loss: 0.9899 - val_accuracy: 0.6528 - val_loss: 0.9794
Epoch 6/50
625/625 ━━━━━━━━━━━━━━━━━━━━ 5s 4ms/step - accuracy: 0.6678 - loss: 0.9422 - val_accuracy: 0.6641 - val_loss: 0.9450
Epoch 7/50
625/625 ━━━━━━━━━━━━━━━━━━━━ 6s 5ms/step - accuracy: 0.6898 - loss: 0.8843 - val_accuracy: 0.6791 - val_loss: 0.9096
Epoch 8/50
625/625 ━━━━━━━━━━━━━━━━━━━━ 3s 4ms/step - accuracy: 0.7045 - loss: 0.8402 - val_accuracy: 0.6848 - val_loss: 0.9054
Epoch 9/50
625/625 ━━━━━━━━━━━━━━━━━━━━ 3s 5ms/step - accuracy: 0.7205 - loss: 0.7881 - val_accuracy: 0.6854 - val_loss: 0.8910
Epoch 10/50
625/625 ━━━━━━━━━━━━━━━━━━━━ 3s 4ms/step - accuracy: 0.7375 - loss: 0.7450 - val_accuracy: 0.6921 - val_loss: 0.8983
Epoch 11/50
625/625 ━━━━━━━━━━━━━━━━━━━━ 3s 4ms/step - accuracy: 0.7457 - loss: 0.7185 - val_accuracy: 0.6754 - val_loss: 0.9490
Epoch 12/50
625/625 ━━━━━━━━━━━━━━━━━━━━ 3s 5ms/step - accuracy: 0.7636 - loss: 0.6611 - val_accuracy: 0.6933 - val_loss: 0.8984
Epoch 13/50
625/625 ━━━━━━━━━━━━━━━━━━━━ 3s 5ms/step - accuracy: 0.7760 - loss: 0.6356 - val_accuracy: 0.6952 - val_loss: 0.9029
Epoch 14/50
625/625 ━━━━━━━━━━━━━━━━━━━━ 3s 4ms/step - accuracy: 0.7844 - loss: 0.5894 - val_accuracy: 0.6983 - val_loss: 0.8968
Epoch 15/50
625/625 ━━━━━━━━━━━━━━━━━━━━ 3s 4ms/step - accuracy: 0.7991 - loss: 0.5659 - val_accuracy: 0.6982 - val_loss: 0.9180
Epoch 16/50
625/625 ━━━━━━━━━━━━━━━━━━━━ 3s 5ms/step - accuracy: 0.8063 - loss: 0.5403 - val_accuracy: 0.6973 - val_loss: 0.9529
Epoch 17/50
625/625 ━━━━━━━━━━━━━━━━━━━━ 3s 5ms/step - accuracy: 0.8140 - loss: 0.5181 - val_accuracy: 0.6960 - val_loss: 0.9539
Epoch 18/50
625/625 ━━━━━━━━━━━━━━━━━━━━ 3s 4ms/step - accuracy: 0.8209 - loss: 0.4916 - val_accuracy: 0.7022 - val_loss: 0.9663
Epoch 19/50
625/625 ━━━━━━━━━━━━━━━━━━━━ 3s 4ms/step - accuracy: 0.8329 - loss: 0.4619 - val_accuracy: 0.6954 - val_loss: 0.9849
Epoch 20/50
625/625 ━━━━━━━━━━━━━━━━━━━━ 3s 5ms/step - accuracy: 0.8375 - loss: 0.4432 - val_accuracy: 0.6939 - val_loss: 1.0375
Epoch 21/50
625/625 ━━━━━━━━━━━━━━━━━━━━ 3s 5ms/step - accuracy: 0.8449 - loss: 0.4226 - val_accuracy: 0.6928 - val_loss: 1.0501
Epoch 22/50
625/625 ━━━━━━━━━━━━━━━━━━━━ 5s 5ms/step - accuracy: 0.8527 - loss: 0.4029 - val_accuracy: 0.6944 - val_loss: 1.0508
Epoch 23/50
625/625 ━━━━━━━━━━━━━━━━━━━━ 3s 5ms/step - accuracy: 0.8629 - loss: 0.3739 - val_accuracy: 0.6929 - val_loss: 1.0989
Epoch 24/50
625/625 ━━━━━━━━━━━━━━━━━━━━ 3s 5ms/step - accuracy: 0.8620 - loss: 0.3734 - val_accuracy: 0.6968 - val_loss: 1.1216
Epoch 25/50
625/625 ━━━━━━━━━━━━━━━━━━━━ 4s 4ms/step - accuracy: 0.8665 - loss: 0.3571 - val_accuracy: 0.6935 - val_loss: 1.1811
Epoch 26/50
625/625 ━━━━━━━━━━━━━━━━━━━━ 5s 5ms/step - accuracy: 0.8737 - loss: 0.3394 - val_accuracy: 0.6915 - val_loss: 1.1861
Epoch 27/50
625/625 ━━━━━━━━━━━━━━━━━━━━ 3s 5ms/step - accuracy: 0.8831 - loss: 0.3187 - val_accuracy: 0.6883 - val_loss: 1.1935
Epoch 28/50
625/625 ━━━━━━━━━━━━━━━━━━━━ 5s 4ms/step - accuracy: 0.8807 - loss: 0.3206 - val_accuracy: 0.6887 - val_loss: 1.2441
Epoch 29/50
625/625 ━━━━━━━━━━━━━━━━━━━━ 6s 6ms/step - accuracy: 0.8858 - loss: 0.3132 - val_accuracy: 0.6883 - val_loss: 1.2760
Epoch 30/50
625/625 ━━━━━━━━━━━━━━━━━━━━ 3s 5ms/step - accuracy: 0.8888 - loss: 0.3027 - val_accuracy: 0.6906 - val_loss: 1.2889
Epoch 31/50
625/625 ━━━━━━━━━━━━━━━━━━━━ 5s 4ms/step - accuracy: 0.8896 - loss: 0.2968 - val_accuracy: 0.6835 - val_loss: 1.3502
Epoch 32/50
625/625 ━━━━━━━━━━━━━━━━━━━━ 6s 6ms/step - accuracy: 0.8963 - loss: 0.2799 - val_accuracy: 0.6947 - val_loss: 1.3277
Epoch 33/50
625/625 ━━━━━━━━━━━━━━━━━━━━ 3s 5ms/step - accuracy: 0.8984 - loss: 0.2751 - val_accuracy: 0.6865 - val_loss: 1.3888
Epoch 34/50
625/625 ━━━━━━━━━━━━━━━━━━━━ 3s 4ms/step - accuracy: 0.9022 - loss: 0.2664 - val_accuracy: 0.6916 - val_loss: 1.3793
Epoch 35/50
625/625 ━━━━━━━━━━━━━━━━━━━━ 6s 5ms/step - accuracy: 0.9043 - loss: 0.2600 - val_accuracy: 0.6859 - val_loss: 1.3609
Epoch 36/50
625/625 ━━━━━━━━━━━━━━━━━━━━ 4s 4ms/step - accuracy: 0.9066 - loss: 0.2563 - val_accuracy: 0.6903 - val_loss: 1.4144
Epoch 37/50
625/625 ━━━━━━━━━━━━━━━━━━━━ 5s 5ms/step - accuracy: 0.9106 - loss: 0.2455 - val_accuracy: 0.6767 - val_loss: 1.5201
Epoch 38/50
625/625 ━━━━━━━━━━━━━━━━━━━━ 3s 5ms/step - accuracy: 0.9083 - loss: 0.2471 - val_accuracy: 0.6876 - val_loss: 1.5302
Epoch 39/50
625/625 ━━━━━━━━━━━━━━━━━━━━ 5s 5ms/step - accuracy: 0.9170 - loss: 0.2291 - val_accuracy: 0.6867 - val_loss: 1.5355
Epoch 40/50
```