In [3]:

```python
# Import necessary libraries
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.model_selection import train_test_split
from sklearn.metrics import (classification_report, confusion_matrix, precision_sco
from sklearn.preprocessing import label_binarize
from tensorflow.keras.datasets import cifar10
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import (Conv2D, MaxPooling2D, Flatten, Dense, Dropout,
from tensorflow.keras.callbacks import EarlyStopping, ModelCheckpoint
from tensorflow.keras.preprocessing.image import ImageDataGenerator
from tensorflow.keras.regularizers import l2
from tensorflow.keras.optimizers import Adam

# Hyperparameters
BATCH_SIZE = 64
EPOCHS = 500
VALIDATION_SPLIT = 0.1
EARLY_STOPPING_PATIENCE = 20
IMAGE_SHAPE = (32, 32, 3)

# Load and preprocess CIFAR-10 dataset
def load_and_preprocess_data():
    (x_train, y_train), (x_test, y_test) = cifar10.load_data()

    # Normalize the data to range [0, 1]
    x_train = x_train.astype('float32') / 255.0
    x_test = x_test.astype('float32') / 255.0

    # Split the training data into training and validation sets
    x_train, x_val, y_train, y_val = train_test_split(x_train, y_train, test_size=V

    return (x_train, y_train), (x_val, y_val), (x_test, y_test)

(x_train, y_train), (x_val, y_val), (x_test, y_test) = load_and_preprocess_data()

# Build the CNN model
def build_model():
    model = Sequential([
        Conv2D(32, (3, 3), activation='relu', input_shape=IMAGE_SHAPE),
        BatchNormalization(),
        MaxPooling2D(pool_size=(2, 2)),

        Conv2D(64, (3, 3), activation='relu'),
        BatchNormalization(),
        MaxPooling2D(pool_size=(2, 2)),

        Conv2D(128, (3, 3), activation='relu'),
        BatchNormalization(),
        MaxPooling2D(pool_size=(2, 2)),

        Flatten(),
        Dense(128, activation='relu', kernel_regularizer=l2(0.01)),
```

```
        Dropout(0.5),
        Dense(10, activation='softmax')
    ])

    model.compile(optimizer=Adam(), loss='sparse_categorical_crossentropy', metrics
    return model

model = build_model()

# Set up EarlyStopping and ModelCheckpoint callbacks
early_stopping = EarlyStopping(monitor='val_loss', patience=EARLY_STOPPING_PATIENCE
model_checkpoint = ModelCheckpoint('best_model.keras', save_best_only=True)

# Image augmentation
datagen = ImageDataGenerator(rotation_range=15,
                             width_shift_range=0.2,
                             height_shift_range=0.2,
                             shear_range=0.2,
                             zoom_range=0.2,
                             horizontal_flip=True,
                             fill_mode='nearest')

datagen.fit(x_train)

# Train the model using data augmentation
history = model.fit(datagen.flow(x_train, y_train, batch_size=BATCH_SIZE),
                    validation_data=(x_val, y_val),
                    epochs=EPOCHS,
                    callbacks=[early_stopping, model_checkpoint])

# Function to plot training history
def plot_history(history):
    """Plot training & validation accuracy and loss."""
    plt.figure(figsize=(12, 5))

    # Plot training & validation accuracy
    plt.subplot(1, 2, 1)
    plt.plot(history.history['accuracy'], label='Train Accuracy')
    plt.plot(history.history['val_accuracy'], label='Validation Accuracy')
    plt.title('Model Accuracy')
    plt.xlabel('Epoch')
    plt.ylabel('Accuracy')
    plt.legend()

    # Plot training & validation loss
    plt.subplot(1, 2, 2)
    plt.plot(history.history['loss'], label='Train Loss')
    plt.plot(history.history['val_loss'], label='Validation Loss')
    plt.title('Model Loss')
    plt.xlabel('Epoch')
    plt.ylabel('Loss')
    plt.legend()

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

```python
# Plot the training history
plot_history(history)

# Generate predictions
y_pred = model.predict(x_test)
y_pred_classes = np.argmax(y_pred, axis=1)

# Classification report
print(classification_report(y_test, y_pred_classes))

# Confusion matrix
conf_matrix = confusion_matrix(y_test, y_pred_classes)

# Plot confusion matrix using seaborn heatmap
plt.figure(figsize=(8, 6))
sns.heatmap(conf_matrix, annot=True, fmt='d', cmap='Blues', xticklabels=np.arange(1
plt.xlabel('Predicted Label')
plt.ylabel('Actual Label')
plt.title('Confusion Matrix')
plt.show()

# Calculate precision, recall, and F1-score
print(f'Precision: {precision_score(y_test, y_pred_classes, average="weighted")}')
print(f'Recall: {recall_score(y_test, y_pred_classes, average="weighted")}')
print(f'F1 Score: {f1_score(y_test, y_pred_classes, average="weighted")}')

# Multi-class ROC curve
y_test_bin = label_binarize(y_test, classes=[0, 1, 2, 3, 4, 5, 6, 7, 8, 9])
n_classes = y_test_bin.shape[1]

fpr, tpr, roc_auc = {}, {}, {}
for i in range(n_classes):
    fpr[i], tpr[i], _ = roc_curve(y_test_bin[:, i], y_pred[:, i])
    roc_auc[i] = auc(fpr[i], tpr[i])

fpr["micro"], tpr["micro"], _ = roc_curve(y_test_bin.ravel(), y_pred.ravel())
roc_auc["micro"] = auc(fpr["micro"], tpr["micro"])

# Plot ROC curve
plt.figure(figsize=(10, 8))
plt.plot(fpr["micro"], tpr["micro"], label=f'Micro-average ROC curve (area = {roc_a
plt.plot([0, 1], [0, 1], linestyle='--', color='gray')
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.05])
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('ROC Curve')
plt.legend(loc="lower right")
plt.show()
```

```
C:\Users\Pan Mour\AppData\Roaming\Python\Python312\site-packages\keras\src\layers\co
nvolutional\base_conv.py:107: UserWarning: Do not pass an `input_shape`/`input_dim`
argument to a layer. When using Sequential models, prefer using an `Input(shape)` ob
ject as the first layer in the model instead.
  super().__init__(activity_regularizer=activity_regularizer, **kwargs)
C:\Users\Pan Mour\AppData\Roaming\Python\Python312\site-packages\keras\src\trainers
\data_adapters\py_dataset_adapter.py:121: UserWarning: Your `PyDataset` class should
call `super().__init__(**kwargs)` in its constructor. `**kwargs` can include `worker
s`, `use_multiprocessing`, `max_queue_size`. Do not pass these arguments to `fit()`,
as they will be ignored.
  self._warn_if_super_not_called()
```

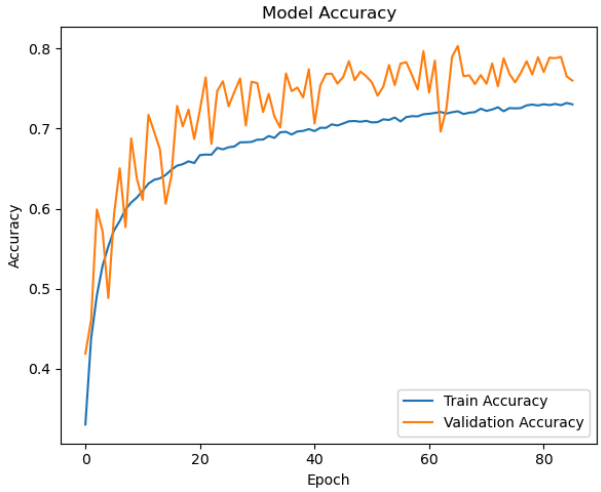
```
Epoch 1/500
704/704 ━━━━━━━━━━━━━━━━━━━━ 55s 75ms/step - accuracy: 0.2782 - loss: 3.3954 - val_a
ccuracy: 0.4190 - val_loss: 1.7519
Epoch 2/500
704/704 ━━━━━━━━━━━━━━━━━━━━ 51s 73ms/step - accuracy: 0.4180 - loss: 1.7702 - val_a
ccuracy: 0.4610 - val_loss: 1.6770
Epoch 3/500
704/704 ━━━━━━━━━━━━━━━━━━━━ 66s 94ms/step - accuracy: 0.4784 - loss: 1.5719 - val_a
ccuracy: 0.5990 - val_loss: 1.2410
Epoch 4/500
704/704 ━━━━━━━━━━━━━━━━━━━━ 55s 78ms/step - accuracy: 0.5242 - loss: 1.4793 - val_a
ccuracy: 0.5716 - val_loss: 1.3009
Epoch 5/500
704/704 ━━━━━━━━━━━━━━━━━━━━ 52s 73ms/step - accuracy: 0.5465 - loss: 1.4125 - val_a
ccuracy: 0.4884 - val_loss: 1.6091
Epoch 6/500
704/704 ━━━━━━━━━━━━━━━━━━━━ 51s 73ms/step - accuracy: 0.5725 - loss: 1.3554 - val_a
ccuracy: 0.5928 - val_loss: 1.3409
Epoch 7/500
704/704 ━━━━━━━━━━━━━━━━━━━━ 55s 78ms/step - accuracy: 0.5839 - loss: 1.3393 - val_a
ccuracy: 0.6504 - val_loss: 1.1026
Epoch 8/500
704/704 ━━━━━━━━━━━━━━━━━━━━ 59s 84ms/step - accuracy: 0.5999 - loss: 1.2924 - val_a
ccuracy: 0.5770 - val_loss: 1.4329
Epoch 9/500
704/704 ━━━━━━━━━━━━━━━━━━━━ 72s 102ms/step - accuracy: 0.6098 - loss: 1.2734 - val_
accuracy: 0.6876 - val_loss: 1.0425
Epoch 10/500
704/704 ━━━━━━━━━━━━━━━━━━━━ 71s 100ms/step - accuracy: 0.6140 - loss: 1.2649 - val_
accuracy: 0.6368 - val_loss: 1.2058
Epoch 11/500
704/704 ━━━━━━━━━━━━━━━━━━━━ 72s 103ms/step - accuracy: 0.6199 - loss: 1.2516 - val_
accuracy: 0.6110 - val_loss: 1.2434
Epoch 12/500
704/704 ━━━━━━━━━━━━━━━━━━━━ 72s 102ms/step - accuracy: 0.6347 - loss: 1.2212 - val_
accuracy: 0.7172 - val_loss: 0.9645
Epoch 13/500
704/704 ━━━━━━━━━━━━━━━━━━━━ 63s 89ms/step - accuracy: 0.6357 - loss: 1.2077 - val_a
ccuracy: 0.6960 - val_loss: 1.0286
Epoch 14/500
704/704 ━━━━━━━━━━━━━━━━━━━━ 72s 102ms/step - accuracy: 0.6377 - loss: 1.2012 - val_
accuracy: 0.6742 - val_loss: 1.0625
Epoch 15/500
704/704 ━━━━━━━━━━━━━━━━━━━━ 70s 100ms/step - accuracy: 0.6433 - loss: 1.1921 - val_
accuracy: 0.6062 - val_loss: 1.3863
Epoch 16/500
704/704 ━━━━━━━━━━━━━━━━━━━━ 69s 98ms/step - accuracy: 0.6500 - loss: 1.1662 - val_a
ccuracy: 0.6402 - val_loss: 1.2003
Epoch 17/500
704/704 ━━━━━━━━━━━━━━━━━━━━ 70s 100ms/step - accuracy: 0.6533 - loss: 1.1759 - val_
accuracy: 0.7282 - val_loss: 0.9332
Epoch 18/500
704/704 ━━━━━━━━━━━━━━━━━━━━ 68s 97ms/step - accuracy: 0.6591 - loss: 1.1534 - val_a
ccuracy: 0.7028 - val_loss: 1.0199
Epoch 19/500
704/704 ━━━━━━━━━━━━━━━━━━━━ 64s 90ms/step - accuracy: 0.6580 - loss: 1.1570 - val_a
```

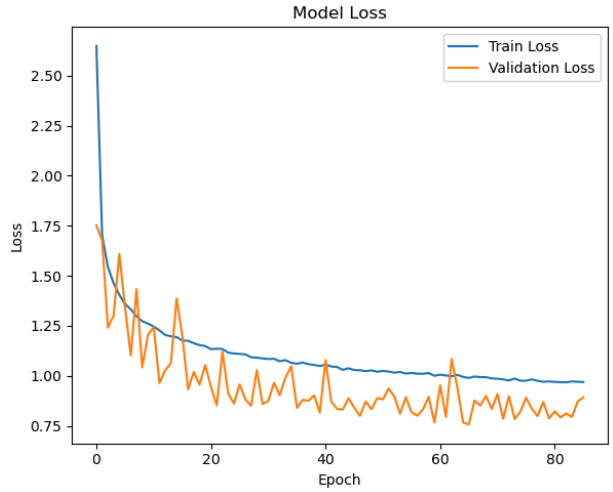
```
ccuracy: 0.7236 - val_loss: 0.9561
Epoch 20/500
704/704 ━━━━━━━━━━━━━━━━━━━━ 64s 91ms/step - accuracy: 0.6569 - loss: 1.1494 - val_a
ccuracy: 0.6870 - val_loss: 1.0531
Epoch 21/500
704/704 ━━━━━━━━━━━━━━━━━━━━ 65s 92ms/step - accuracy: 0.6679 - loss: 1.1217 - val_a
ccuracy: 0.7236 - val_loss: 0.9449
Epoch 22/500
704/704 ━━━━━━━━━━━━━━━━━━━━ 64s 90ms/step - accuracy: 0.6636 - loss: 1.1349 - val_a
ccuracy: 0.7638 - val_loss: 0.8532
Epoch 23/500
704/704 ━━━━━━━━━━━━━━━━━━━━ 58s 82ms/step - accuracy: 0.6596 - loss: 1.1567 - val_a
ccuracy: 0.6808 - val_loss: 1.1212
Epoch 24/500
704/704 ━━━━━━━━━━━━━━━━━━━━ 52s 74ms/step - accuracy: 0.6745 - loss: 1.1234 - val_a
ccuracy: 0.7468 - val_loss: 0.9115
Epoch 25/500
704/704 ━━━━━━━━━━━━━━━━━━━━ 52s 74ms/step - accuracy: 0.6737 - loss: 1.1084 - val_a
ccuracy: 0.7590 - val_loss: 0.8620
Epoch 26/500
704/704 ━━━━━━━━━━━━━━━━━━━━ 82s 73ms/step - accuracy: 0.6804 - loss: 1.0989 - val_a
ccuracy: 0.7278 - val_loss: 0.9574
Epoch 27/500
704/704 ━━━━━━━━━━━━━━━━━━━━ 52s 74ms/step - accuracy: 0.6794 - loss: 1.1059 - val_a
ccuracy: 0.7456 - val_loss: 0.8833
Epoch 28/500
704/704 ━━━━━━━━━━━━━━━━━━━━ 52s 74ms/step - accuracy: 0.6830 - loss: 1.0905 - val_a
ccuracy: 0.7624 - val_loss: 0.8503
Epoch 29/500
704/704 ━━━━━━━━━━━━━━━━━━━━ 52s 74ms/step - accuracy: 0.6859 - loss: 1.0793 - val_a
ccuracy: 0.7038 - val_loss: 1.0295
Epoch 30/500
704/704 ━━━━━━━━━━━━━━━━━━━━ 52s 74ms/step - accuracy: 0.6814 - loss: 1.0888 - val_a
ccuracy: 0.7584 - val_loss: 0.8597
Epoch 31/500
704/704 ━━━━━━━━━━━━━━━━━━━━ 52s 74ms/step - accuracy: 0.6859 - loss: 1.0901 - val_a
ccuracy: 0.7566 - val_loss: 0.8748
Epoch 32/500
704/704 ━━━━━━━━━━━━━━━━━━━━ 53s 75ms/step - accuracy: 0.6867 - loss: 1.0821 - val_a
ccuracy: 0.7206 - val_loss: 0.9660
Epoch 33/500
704/704 ━━━━━━━━━━━━━━━━━━━━ 53s 75ms/step - accuracy: 0.6900 - loss: 1.0759 - val_a
ccuracy: 0.7432 - val_loss: 0.9035
Epoch 34/500
704/704 ━━━━━━━━━━━━━━━━━━━━ 54s 76ms/step - accuracy: 0.6904 - loss: 1.0727 - val_a
ccuracy: 0.7148 - val_loss: 0.9912
Epoch 35/500
704/704 ━━━━━━━━━━━━━━━━━━━━ 53s 75ms/step - accuracy: 0.6946 - loss: 1.0683 - val_a
ccuracy: 0.7012 - val_loss: 1.0481
Epoch 36/500
704/704 ━━━━━━━━━━━━━━━━━━━━ 53s 75ms/step - accuracy: 0.6974 - loss: 1.0614 - val_a
ccuracy: 0.7688 - val_loss: 0.8402
Epoch 37/500
704/704 ━━━━━━━━━━━━━━━━━━━━ 53s 75ms/step - accuracy: 0.6929 - loss: 1.0639 - val_a
ccuracy: 0.7466 - val_loss: 0.8802
Epoch 38/500
```

```
704/704 ━━━━━━━━━━━━━━━━━━━━ 53s 75ms/step - accuracy: 0.6971 - loss: 1.0570 - val_a
ccuracy: 0.7512 - val_loss: 0.8760
Epoch 39/500
704/704 ━━━━━━━━━━━━━━━━━━━━ 53s 76ms/step - accuracy: 0.6913 - loss: 1.0668 - val_a
ccuracy: 0.7388 - val_loss: 0.9020
Epoch 40/500
704/704 ━━━━━━━━━━━━━━━━━━━━ 53s 76ms/step - accuracy: 0.7013 - loss: 1.0438 - val_a
ccuracy: 0.7740 - val_loss: 0.8173
Epoch 41/500
704/704 ━━━━━━━━━━━━━━━━━━━━ 53s 75ms/step - accuracy: 0.6961 - loss: 1.0568 - val_a
ccuracy: 0.7062 - val_loss: 1.0793
Epoch 42/500
704/704 ━━━━━━━━━━━━━━━━━━━━ 53s 75ms/step - accuracy: 0.7011 - loss: 1.0442 - val_a
ccuracy: 0.7538 - val_loss: 0.8718
Epoch 43/500
704/704 ━━━━━━━━━━━━━━━━━━━━ 54s 76ms/step - accuracy: 0.7014 - loss: 1.0405 - val_a
ccuracy: 0.7682 - val_loss: 0.8349
Epoch 44/500
704/704 ━━━━━━━━━━━━━━━━━━━━ 53s 75ms/step - accuracy: 0.7098 - loss: 1.0196 - val_a
ccuracy: 0.7684 - val_loss: 0.8315
Epoch 45/500
704/704 ━━━━━━━━━━━━━━━━━━━━ 53s 76ms/step - accuracy: 0.7037 - loss: 1.0460 - val_a
ccuracy: 0.7560 - val_loss: 0.8884
Epoch 46/500
704/704 ━━━━━━━━━━━━━━━━━━━━ 54s 76ms/step - accuracy: 0.7049 - loss: 1.0273 - val_a
ccuracy: 0.7644 - val_loss: 0.8407
Epoch 47/500
704/704 ━━━━━━━━━━━━━━━━━━━━ 53s 75ms/step - accuracy: 0.7100 - loss: 1.0257 - val_a
ccuracy: 0.7840 - val_loss: 0.8000
Epoch 48/500
704/704 ━━━━━━━━━━━━━━━━━━━━ 53s 75ms/step - accuracy: 0.7124 - loss: 1.0145 - val_a
ccuracy: 0.7604 - val_loss: 0.8719
Epoch 49/500
704/704 ━━━━━━━━━━━━━━━━━━━━ 53s 75ms/step - accuracy: 0.7143 - loss: 1.0123 - val_a
ccuracy: 0.7712 - val_loss: 0.8336
Epoch 50/500
704/704 ━━━━━━━━━━━━━━━━━━━━ 53s 75ms/step - accuracy: 0.7129 - loss: 1.0155 - val_a
ccuracy: 0.7654 - val_loss: 0.8889
Epoch 51/500
704/704 ━━━━━━━━━━━━━━━━━━━━ 52s 74ms/step - accuracy: 0.7094 - loss: 1.0230 - val_a
ccuracy: 0.7576 - val_loss: 0.8828
Epoch 52/500
704/704 ━━━━━━━━━━━━━━━━━━━━ 52s 74ms/step - accuracy: 0.7103 - loss: 1.0141 - val_a
ccuracy: 0.7410 - val_loss: 0.9369
Epoch 53/500
704/704 ━━━━━━━━━━━━━━━━━━━━ 51s 73ms/step - accuracy: 0.7082 - loss: 1.0244 - val_a
ccuracy: 0.7524 - val_loss: 0.8971
Epoch 54/500
704/704 ━━━━━━━━━━━━━━━━━━━━ 52s 73ms/step - accuracy: 0.7132 - loss: 1.0147 - val_a
ccuracy: 0.7792 - val_loss: 0.8102
Epoch 55/500
704/704 ━━━━━━━━━━━━━━━━━━━━ 51s 73ms/step - accuracy: 0.7142 - loss: 1.0065 - val_a
ccuracy: 0.7542 - val_loss: 0.8934
Epoch 56/500
704/704 ━━━━━━━━━━━━━━━━━━━━ 52s 74ms/step - accuracy: 0.7070 - loss: 1.0203 - val_a
ccuracy: 0.7810 - val_loss: 0.8186
```

```
Epoch 57/500
704/704 ━━━━━━━━━━━━━━━━━━━━ 52s 74ms/step - accuracy: 0.7155 - loss: 1.0074 - val_a
ccuracy: 0.7828 - val_loss: 0.8013
Epoch 58/500
704/704 ━━━━━━━━━━━━━━━━━━━━ 52s 74ms/step - accuracy: 0.7191 - loss: 0.9997 - val_a
ccuracy: 0.7666 - val_loss: 0.8331
Epoch 59/500
704/704 ━━━━━━━━━━━━━━━━━━━━ 52s 74ms/step - accuracy: 0.7126 - loss: 1.0251 - val_a
ccuracy: 0.7488 - val_loss: 0.8955
Epoch 60/500
704/704 ━━━━━━━━━━━━━━━━━━━━ 52s 74ms/step - accuracy: 0.7177 - loss: 1.0053 - val_a
ccuracy: 0.7966 - val_loss: 0.7673
Epoch 61/500
704/704 ━━━━━━━━━━━━━━━━━━━━ 52s 74ms/step - accuracy: 0.7213 - loss: 0.9987 - val_a
ccuracy: 0.7448 - val_loss: 0.9524
Epoch 62/500
704/704 ━━━━━━━━━━━━━━━━━━━━ 53s 75ms/step - accuracy: 0.7217 - loss: 0.9920 - val_a
ccuracy: 0.7848 - val_loss: 0.7955
Epoch 63/500
704/704 ━━━━━━━━━━━━━━━━━━━━ 52s 74ms/step - accuracy: 0.7223 - loss: 0.9880 - val_a
ccuracy: 0.6962 - val_loss: 1.0840
Epoch 64/500
704/704 ━━━━━━━━━━━━━━━━━━━━ 52s 74ms/step - accuracy: 0.7178 - loss: 1.0058 - val_a
ccuracy: 0.7244 - val_loss: 0.9424
Epoch 65/500
704/704 ━━━━━━━━━━━━━━━━━━━━ 52s 73ms/step - accuracy: 0.7186 - loss: 1.0010 - val_a
ccuracy: 0.7896 - val_loss: 0.7680
Epoch 66/500
704/704 ━━━━━━━━━━━━━━━━━━━━ 52s 74ms/step - accuracy: 0.7250 - loss: 0.9755 - val_a
ccuracy: 0.8030 - val_loss: 0.7572
Epoch 67/500
704/704 ━━━━━━━━━━━━━━━━━━━━ 53s 75ms/step - accuracy: 0.7197 - loss: 0.9902 - val_a
ccuracy: 0.7652 - val_loss: 0.8769
Epoch 68/500
704/704 ━━━━━━━━━━━━━━━━━━━━ 52s 74ms/step - accuracy: 0.7227 - loss: 0.9880 - val_a
ccuracy: 0.7664 - val_loss: 0.8518
Epoch 69/500
704/704 ━━━━━━━━━━━━━━━━━━━━ 52s 74ms/step - accuracy: 0.7220 - loss: 0.9819 - val_a
ccuracy: 0.7554 - val_loss: 0.8999
Epoch 70/500
704/704 ━━━━━━━━━━━━━━━━━━━━ 52s 74ms/step - accuracy: 0.7194 - loss: 0.9985 - val_a
ccuracy: 0.7668 - val_loss: 0.8345
Epoch 71/500
704/704 ━━━━━━━━━━━━━━━━━━━━ 53s 75ms/step - accuracy: 0.7224 - loss: 0.9899 - val_a
ccuracy: 0.7558 - val_loss: 0.9091
Epoch 72/500
704/704 ━━━━━━━━━━━━━━━━━━━━ 58s 82ms/step - accuracy: 0.7263 - loss: 0.9745 - val_a
ccuracy: 0.7812 - val_loss: 0.7860
Epoch 73/500
704/704 ━━━━━━━━━━━━━━━━━━━━ 61s 86ms/step - accuracy: 0.7256 - loss: 0.9775 - val_a
ccuracy: 0.7526 - val_loss: 0.8981
Epoch 74/500
704/704 ━━━━━━━━━━━━━━━━━━━━ 65s 93ms/step - accuracy: 0.7212 - loss: 0.9847 - val_a
ccuracy: 0.7876 - val_loss: 0.7846
Epoch 75/500
704/704 ━━━━━━━━━━━━━━━━━━━━ 58s 82ms/step - accuracy: 0.7275 - loss: 0.9698 - val_a
```

```
ccuracy: 0.7684 - val_loss: 0.8217
Epoch 76/500
704/704 ━━━━━━━━━━━━━━━━━━━━ 58s 82ms/step - accuracy: 0.7262 - loss: 0.9730 - val_a
ccuracy: 0.7576 - val_loss: 0.8913
Epoch 77/500
704/704 ━━━━━━━━━━━━━━━━━━━━ 57s 82ms/step - accuracy: 0.7284 - loss: 0.9765 - val_a
ccuracy: 0.7696 - val_loss: 0.8355
Epoch 78/500
704/704 ━━━━━━━━━━━━━━━━━━━━ 58s 82ms/step - accuracy: 0.7331 - loss: 0.9668 - val_a
ccuracy: 0.7840 - val_loss: 0.7992
Epoch 79/500
704/704 ━━━━━━━━━━━━━━━━━━━━ 58s 83ms/step - accuracy: 0.7329 - loss: 0.9634 - val_a
ccuracy: 0.7674 - val_loss: 0.8700
Epoch 80/500
704/704 ━━━━━━━━━━━━━━━━━━━━ 58s 82ms/step - accuracy: 0.7317 - loss: 0.9710 - val_a
ccuracy: 0.7890 - val_loss: 0.7876
Epoch 81/500
704/704 ━━━━━━━━━━━━━━━━━━━━ 58s 83ms/step - accuracy: 0.7345 - loss: 0.9556 - val_a
ccuracy: 0.7706 - val_loss: 0.8227
Epoch 82/500
704/704 ━━━━━━━━━━━━━━━━━━━━ 58s 82ms/step - accuracy: 0.7329 - loss: 0.9579 - val_a
ccuracy: 0.7884 - val_loss: 0.7926
Epoch 83/500
704/704 ━━━━━━━━━━━━━━━━━━━━ 58s 82ms/step - accuracy: 0.7354 - loss: 0.9577 - val_a
ccuracy: 0.7878 - val_loss: 0.8119
Epoch 84/500
704/704 ━━━━━━━━━━━━━━━━━━━━ 85s 86ms/step - accuracy: 0.7314 - loss: 0.9641 - val_a
ccuracy: 0.7894 - val_loss: 0.7949
Epoch 85/500
704/704 ━━━━━━━━━━━━━━━━━━━━ 75s 107ms/step - accuracy: 0.7331 - loss: 0.9707 - val_
accuracy: 0.7650 - val_loss: 0.8728
Epoch 86/500
704/704 ━━━━━━━━━━━━━━━━━━━━ 60s 85ms/step - accuracy: 0.7343 - loss: 0.9568 - val_a
ccuracy: 0.7598 - val_loss: 0.8927
```

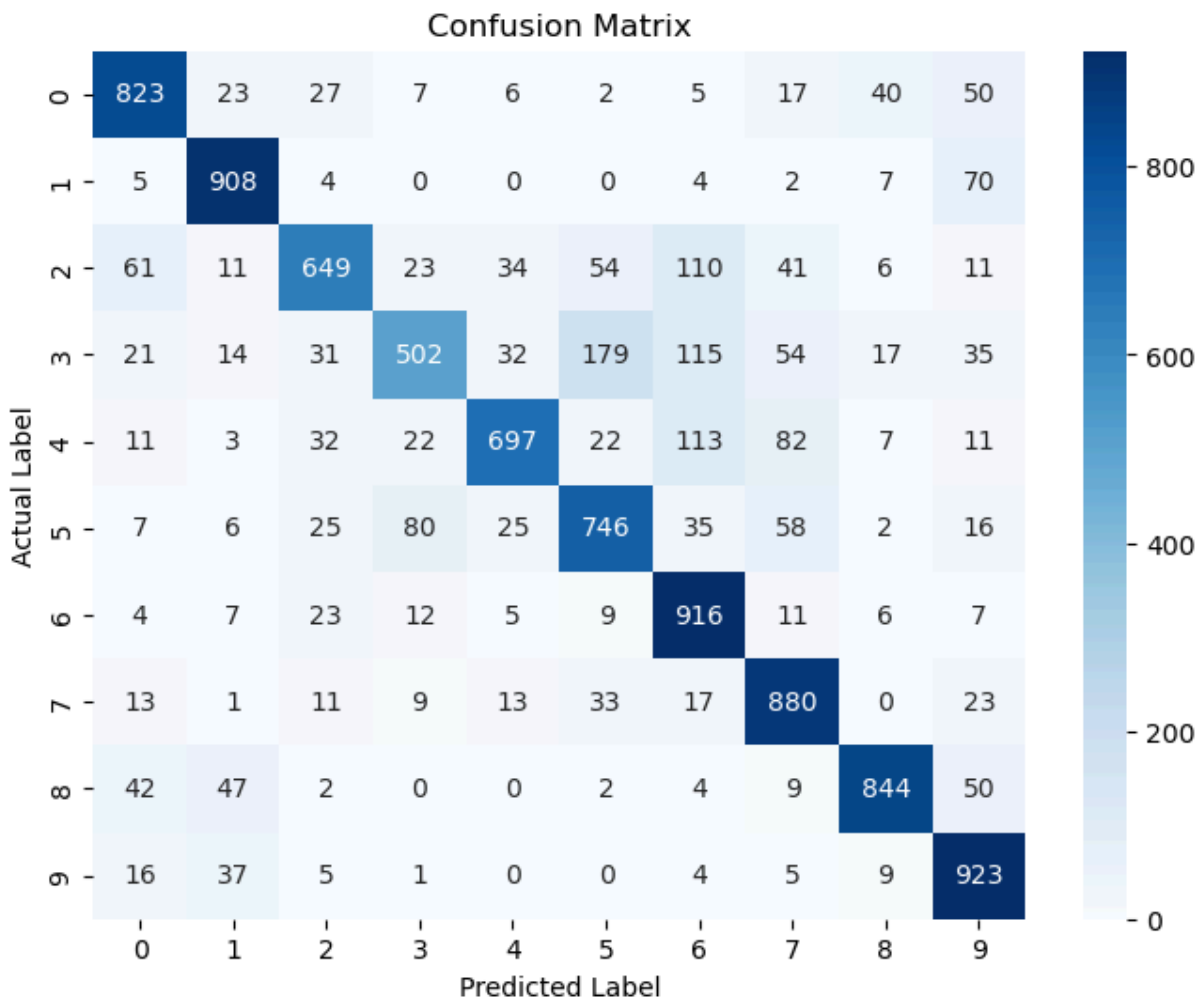




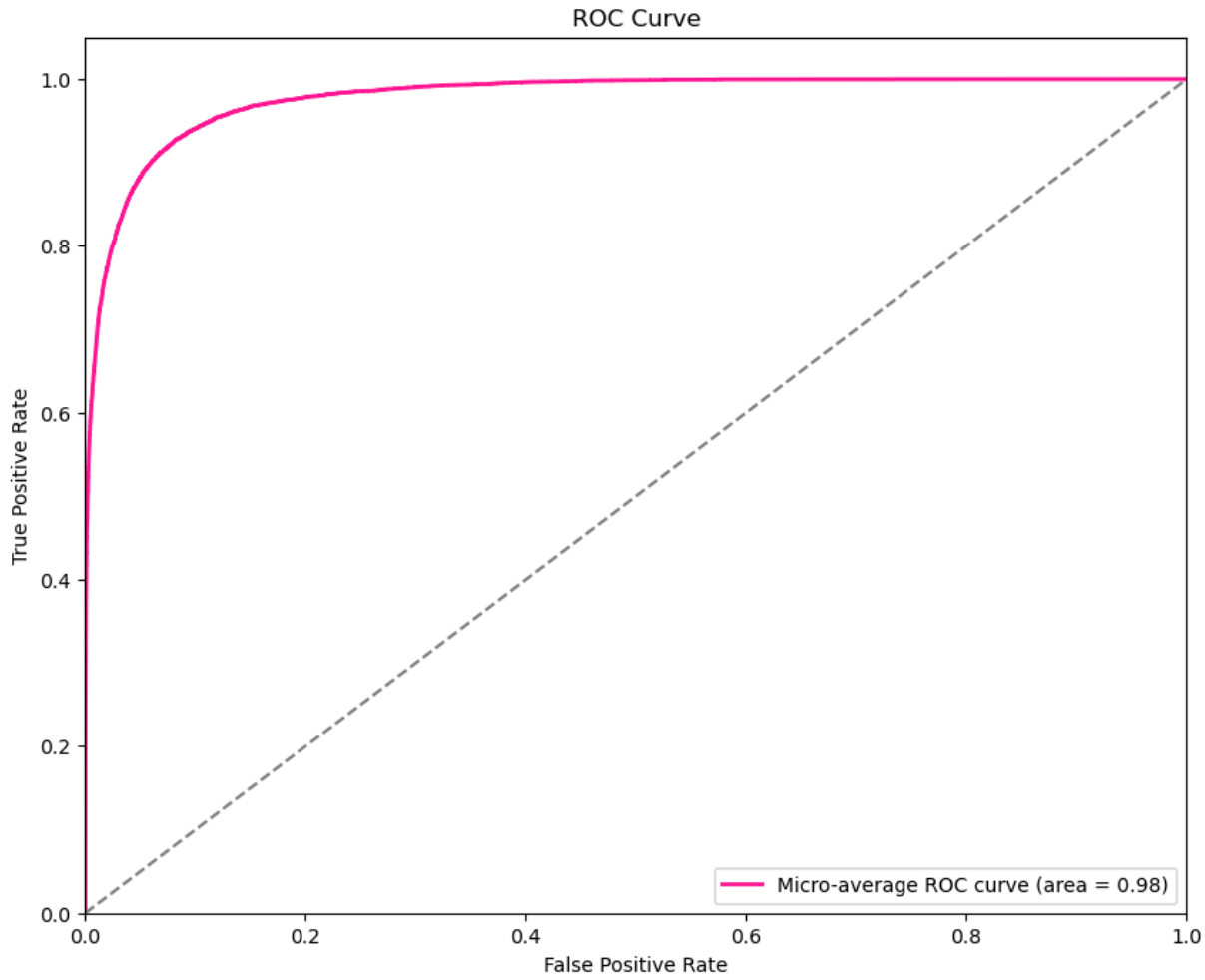
```
313/313 ━━━━━━━━━━━━━━━━━━━━ 3s 8ms/step
              precision    recall  f1-score   support

           0       0.82      0.82      0.82      1000
           1       0.86      0.91      0.88      1000
           2       0.80      0.65      0.72      1000
           3       0.77      0.50      0.61      1000
           4       0.86      0.70      0.77      1000
           5       0.71      0.75      0.73      1000
           6       0.69      0.92      0.79      1000
           7       0.76      0.88      0.82      1000
           8       0.90      0.84      0.87      1000
           9       0.77      0.92      0.84      1000

    accuracy                           0.79     10000
   macro avg       0.79      0.79      0.78     10000
weighted avg       0.79      0.79      0.78     10000
```




```
Precision: 0.7941095567537001
Recall: 0.7888
F1 Score: 0.7842045258211058
```


ROC Curve
True Positive Rate
False Positive Rate
0.0
0.2
0.4
0.6
0.8
1.0
Micro-average ROC curve (area = 0.98)

In [ ]: