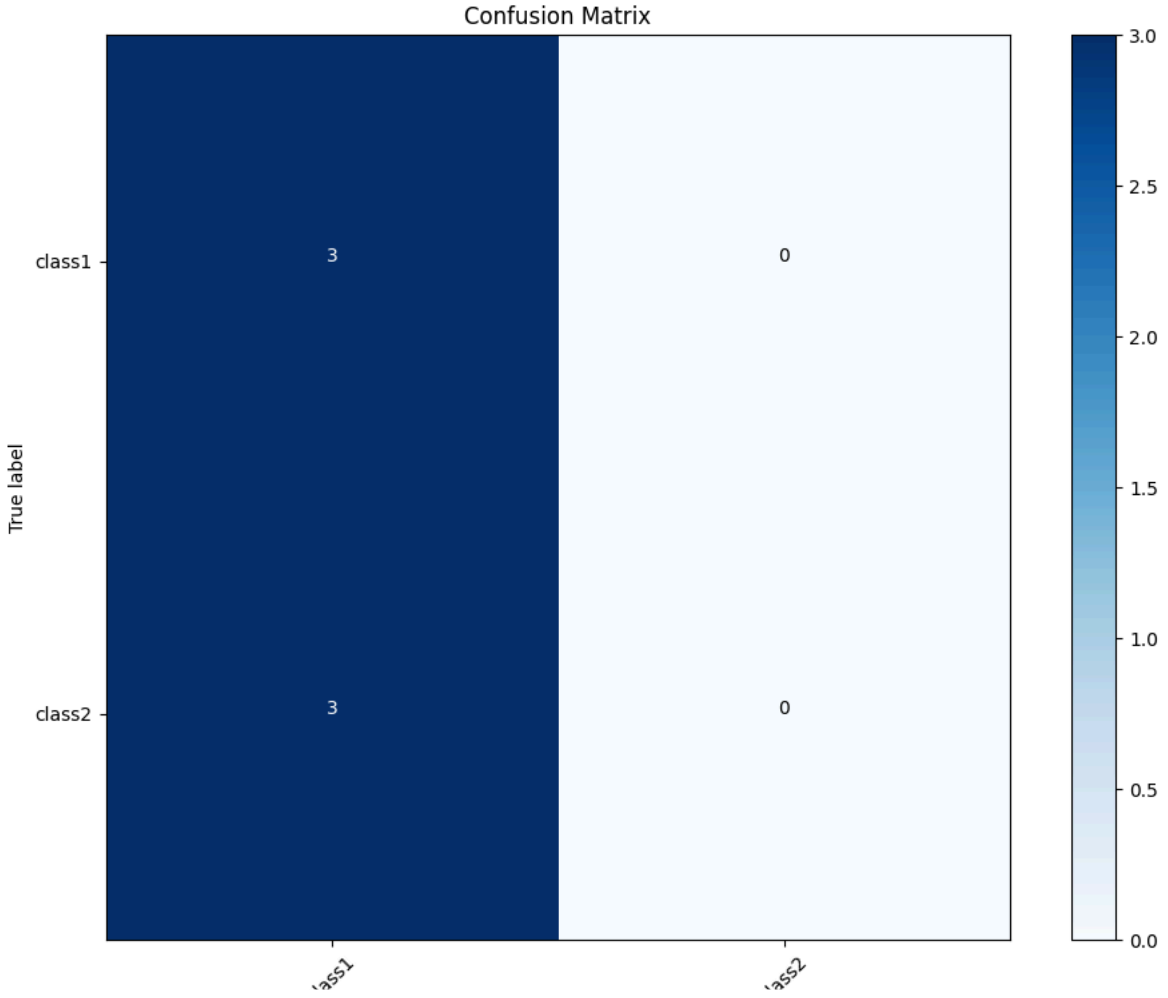
```
description = """
Assingnment - 02 : Apply transfer learning techniques using pre-trained models like ResNet-50 and VGG16 for image classification.
Use a custom image dataset.
Train the model using transfer learning, fine-tuning only the dense layers.
Evaluate the model using metrics like accuracy, precision, and recall.
"""

print(description)
```

```
Assingnment - 02 : Apply transfer learning techniques using pre-trained models like ResNet-50 and VGG16 for image classification.
Use a custom image dataset.
Train the model using transfer learning, fine-tuning only the dense layers.
Evaluate the model using metrics like accuracy, precision, and recall.
```

```
# Step 1: Mount Google Drive
from google.colab import drive
drive.mount('/content/drive', force_remount=True)

# Step 2: Set the path to your dataset
data_dir = '/content/drive/MyDrive/dataset'  # Adjust as necessary

# Step 3: Import required libraries
import numpy as np
import tensorflow as tf
from tensorflow.keras.preprocessing.image import ImageDataGenerator
from tensorflow.keras.applications import ResNet50, VGG16
from tensorflow.keras import layers, models
from sklearn.metrics import classification_report, confusion_matrix
import matplotlib.pyplot as plt

# Step 4: Data preprocessing
train_datagen = ImageDataGenerator(
    rescale=1./255,
    rotation_range=20,
    width_shift_range=0.2,
    height_shift_range=0.2,
    shear_range=0.2,
    zoom_range=0.2,
    horizontal_flip=True,
    fill_mode='nearest'
)

val_datagen = ImageDataGenerator(rescale=1./255)

# Create generators
train_generator = train_datagen.flow_from_directory(
    data_dir + '/train',
    target_size=(224, 224),
    batch_size=32,
    class_mode='categorical'
)

validation_generator = val_datagen.flow_from_directory(
    data_dir + '/validation',
    target_size=(224, 224),
    batch_size=32,
    class_mode='categorical'
)

# Step 5: Load ResNet50 model
base_model = ResNet50(weights='imagenet', include_top=False, input_shape=(224, 224, 3))

# Freeze the base model
base_model.trainable = False

# Create a new model on top
model = models.Sequential()
model.add(base_model)
model.add(layers.GlobalAveragePooling2D())
model.add(layers.Dense(256, activation='relu'))
model.add(layers.Dropout(0.5))
model.add(layers.Dense(train_generator.num_classes, activation='softmax'))

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

# Step 6: Train the model
history = model.fit(
    train_generator,
    steps_per_epoch=train_generator.samples // train_generator.batch_size,
    validation_data=validation_generator,
    validation_steps=validation_generator.samples // validation_generator.batch_size,
    epochs=10  # Adjust as needed
)

# Step 7: Evaluate the model
val_loss, val_accuracy = model.evaluate(validation_generator)
print(f'Validation Accuracy: {val_accuracy:.4f}')

# Predict classes
predictions = model.predict(validation_generator)
predicted_classes = np.argmax(predictions, axis=1)

# Get true classes
true_classes = validation_generator.classes
class_labels = list(validation_generator.class_indices.keys())

# Print classification report
report = classification_report(true_classes, predicted_classes, target_names=class_labels)
print(report)

# Print confusion matrix
confusion_mtx = confusion_matrix(true_classes, predicted_classes)
print(confusion_mtx)

# Plot confusion matrix
plt.figure(figsize=(10, 8))
plt.imshow(confusion_mtx, interpolation='nearest', cmap=plt.cm.Blues)
plt.title('Confusion Matrix')
plt.colorbar()
tick_marks = np.arange(len(class_labels))
plt.xticks(tick_marks, class_labels, rotation=45)
plt.yticks(tick_marks, class_labels)

thresh = confusion_mtx.max() / 2.
for i, j in np.ndindex(confusion_mtx.shape):
    plt.text(j, i, format(confusion_mtx[i, j]),
             horizontalalignment="center",
             color="white" if confusion_mtx[i, j] > thresh else "black")

plt.ylabel('True label')
plt.xlabel('Predicted label')
plt.tight_layout()
plt.show()

# Optional: Fine-tuning the model
base_model.trainable = True
fine_tune_at = 143  # Adjust as necessary for the model architecture

for layer in base_model.layers[:fine_tune_at]:
    layer.trainable = False

model.compile(optimizer=tf.keras.optimizers.Adam(learning_rate=0.0001),
              loss='categorical_crossentropy',
              metrics=['accuracy'])

# Continue training
history_fine = model.fit(
    train_generator,
    steps_per_epoch=train_generator.samples // train_generator.batch_size,
    validation_data=validation_generator,
    validation_steps=validation_generator.samples // validation_generator.batch_size,
    epochs=10  # Adjust as needed
)
```

```
Mounted at /content/drive
Found 10 images belonging to 2 classes.
Found 6 images belonging to 2 classes.
Downloading data from https://storage.googleapis.com/tensorflow/keras-applications/resnet/resnet50_weights_tf_dim_ordering_tf_k
94765736/94765736 ━━━━━━━━━━━━━━━━━━━━ 1s 0us/step
Epoch 1/10
/usr/local/lib/python3.10/dist-packages/keras/src/trainers/data_adapters/py_dataset_adapter.py:121: UserWarning: Your `PyDatase
  self._warn_if_super_not_called()
1/1 ━━━━━━━━━━━━━━━━━━━━ 22s 22s/step - accuracy: 0.9000 - loss: 0.4117 - val_accuracy: 0.5000 - val_loss: 0.8758
Epoch 2/10
1/1 ━━━━━━━━━━━━━━━━━━━━ 3s 3s/step - accuracy: 0.4000 - loss: 1.2409 - val_accuracy: 0.5000 - val_loss: 0.7123
Epoch 3/10
1/1 ━━━━━━━━━━━━━━━━━━━━ 7s 7s/step - accuracy: 0.8000 - loss: 0.4917 - val_accuracy: 0.5000 - val_loss: 0.7400
Epoch 4/10
1/1 ━━━━━━━━━━━━━━━━━━━━ 4s 4s/step - accuracy: 0.4000 - loss: 1.0441 - val_accuracy: 0.5000 - val_loss: 0.8714
Epoch 5/10
1/1 ━━━━━━━━━━━━━━━━━━━━ 4s 4s/step - accuracy: 0.5000 - loss: 1.1171 - val_accuracy: 0.5000 - val_loss: 0.8171
Epoch 6/10
1/1 ━━━━━━━━━━━━━━━━━━━━ 6s 6s/step - accuracy: 0.5000 - loss: 0.9132 - val_accuracy: 0.5000 - val_loss: 0.7093
Epoch 7/10
1/1 ━━━━━━━━━━━━━━━━━━━━ 3s 3s/step - accuracy: 0.7000 - loss: 0.6662 - val_accuracy: 0.5000 - val_loss: 0.6915
Epoch 8/10
1/1 ━━━━━━━━━━━━━━━━━━━━ 7s 7s/step - accuracy: 0.5000 - loss: 0.8838 - val_accuracy: 0.5000 - val_loss: 0.7092
Epoch 9/10
1/1 ━━━━━━━━━━━━━━━━━━━━ 5s 5s/step - accuracy: 0.5000 - loss: 0.7719 - val_accuracy: 0.5000 - val_loss: 0.7617
Epoch 10/10
1/1 ━━━━━━━━━━━━━━━━━━━━ 3s 3s/step - accuracy: 0.6000 - loss: 0.7226 - val_accuracy: 0.5000 - val_loss: 0.8111
1/1 ━━━━━━━━━━━━━━━━━━━━ 1s 1s/step - accuracy: 0.5000 - loss: 0.8111
Validation Accuracy: 0.5000
1/1 ━━━━━━━━━━━━━━━━━━━━ 4s 4s/step
              precision    recall  f1-score   support

      class1       0.50      1.00      0.67         3
      class2       0.00      0.00      0.00         3

    accuracy                           0.50         6
   macro avg       0.25      0.50      0.33         6
weighted avg       0.25      0.50      0.33         6

[[3 0]
 [3 0]]
/usr/local/lib/python3.10/dist-packages/sklearn/metrics/_classification.py:1531: UndefinedMetricWarning: Precision is ill-defin
  _warn_prf(average, modifier, f"{metric.capitalize()} is", len(result))
/usr/local/lib/python3.10/dist-packages/sklearn/metrics/_classification.py:1531: UndefinedMetricWarning: Precision is ill-defin
  _warn_prf(average, modifier, f"{metric.capitalize()} is", len(result))
/usr/local/lib/python3.10/dist-packages/sklearn/metrics/_classification.py:1531: UndefinedMetricWarning: Precision is ill-defin
  _warn_prf(average, modifier, f"{metric.capitalize()} is", len(result))
```


Confusion Matrix
class1
class2
True label
lass1
lass2