```
%tensorflow_version 2.x
import tensorflow
tensorflow.__version__
```

```
     Colab only includes TensorFlow 2.x; %tensorflow_version has no effect.
     '2.15.0'
```

```
#Important libraries for data manipulation
# Numpy is used for large, multi-dimensional arrays and matrices, along with mathematical operators on these arrays
# Pandas is used for data manipulation and analysis
import numpy as np
import pandas as pd
# Import necessary libraries
import os
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import matplotlib.image as mpimg  # Add this line
from zipfile import ZipFile
from google.colab import drive
from tensorflow.keras.preprocessing import image_dataset_from_directory
from tensorflow.keras.applications import ResNet50
from tensorflow.keras import layers, models, optimizers
from tensorflow.keras.optimizers import SGD
from sklearn.metrics import classification_report, roc_auc_score, confusion_matrix
from tensorflow.keras.utils import to_categorical
# Matplotlib is a data visualization library for 2D plots of arrays, built on NumPy arrays
import matplotlib.pyplot as plt

# Seaborn is based on matplotlib, which aids in drawing attractive and informative statistical graphics.
import seaborn as sns

# os is used to provide a way of using operating system dependent functionality
# We use it for setting working folder
import os
import random
import pickle

# Sklearn
from sklearn.preprocessing import LabelBinarizer
from sklearn.model_selection import train_test_split
from sklearn.metrics import classification_report
from sklearn.metrics import f1_score
from sklearn.metrics import classification_report, confusion_matrix

# Libraries for working with images and CNN
import cv2
from google.colab.patches import cv2_imshow
import tensorflow as tf
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import *
from tensorflow.keras.layers import Dense, Activation, Dropout, Flatten,Conv2D, MaxPooling2D
from tensorflow.keras import optimizers

from tensorflow.keras.applications import ResNet50, VGG16, VGG19, MobileNetV2
from tensorflow.keras.applications.resnet50 import preprocess_input as prepro_res50
# Ignore the warnings
import warnings
warnings.filterwarnings("ignore")

from zipfile import ZipFile
```

```
# Define the path to your data zip file
data_path = "/content/drive/MyDrive/56-ML.zip"

# Extract the dataset from the zip file
with ZipFile(data_path, 'r') as zip:
    zip.extractall()
    print('The data set has been extracted.')
```

```
     The data set has been extracted.
```

```
import os

# Define the extracted directory
extracted_dir = "/content/ML-Project"

# List the contents of the extracted directory
extracted_contents = os.listdir(extracted_dir)

# Print the list of contents
print("Contents of the extracted directory:")
for item in extracted_contents:
    print(item)
```

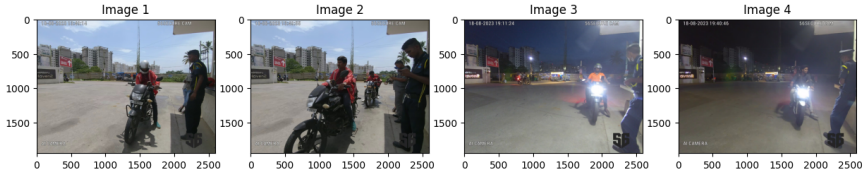
```
Contents of the extracted directory:
assign_classification
```

```
# Specify the directory path
base_dir = "/content/ML-Project/assign_classification"

# Get the list of file names in the directories
train_dir = os.path.join(base_dir, 'train')
test_dir = os.path.join(base_dir, 'test')

# Function to count the number of images in each class
def count_images_in_class(class_dir):
    return len(os.listdir(class_dir))

# Function to get image paths for visualization
def get_image_paths(class_dir, start_index, num_images):
    image_paths = [os.path.join(class_dir, img) for img in os.listdir(class_dir)[start_index:start_index + num_images]]
    return image_paths

# Display the number of images in each class
for class_name in os.listdir(train_dir):
    class_dir = os.path.join(train_dir, class_name)
    print(f"Number of images in '{class_name}' class: {count_images_in_class(class_dir)}")
```

```
Number of images in 'bike' class: 50
Number of images in 'others' class: 20
Number of images in 'car' class: 50
```

```
# Visualize some images from the 'bike' class
num_images_to_display = 4
class_name = 'bike'
class_dir = os.path.join(train_dir, class_name)
print(f"Number of images in '{class_name}' class: {count_images_in_class(class_dir)}")
image_paths_to_display = get_image_paths(class_dir, start_index=0, num_images=num_images_to_display)
print("Image paths to display:", image_paths_to_display)

# Display the images
fig, axes = plt.subplots(1, num_images_to_display, figsize=(15, 5))
for i, image_path in enumerate(image_paths_to_display):
    img = plt.imread(image_path)
    axes[i].imshow(img)
    axes[i].set_title(f"Image {i+1}")
plt.show()
```

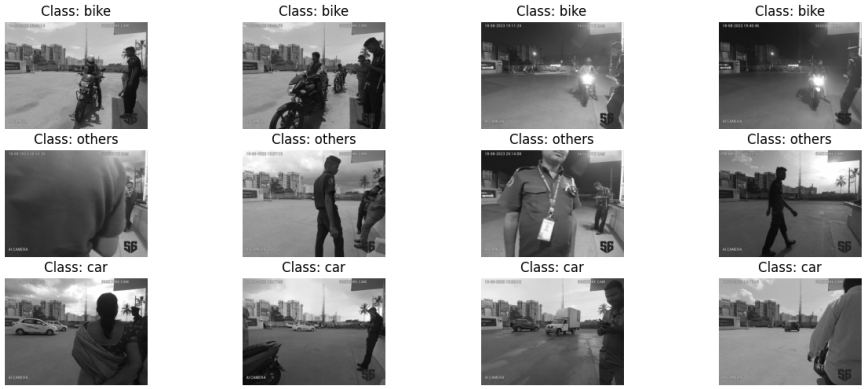
```
Number of images in 'bike' class: 50
Image paths to display: ['/content/ML-Project/assign_classification/train/bike/pfva_d
```




```
from google.colab import drive
drive.mount('/content/drive')
```

```
Drive already mounted at /content/drive; to attempt to forcibly remount, call drive.mount("/content/drive", force_remount=True).
```

```
# Function to convert RGB images to grayscale
def rgb_to_gray(rgb):
    return np.dot(rgb[..., :3], [0.2989, 0.5870, 0.1140])

# Get the list of class names
class_names = os.listdir(train_dir)

# Display all images for each class in grayscale
fig, axes = plt.subplots(len(class_names), 4, figsize=(15, 2 * len(class_names)))
for i, class_name in enumerate(class_names):
    class_dir = os.path.join(train_dir, class_name)
    image_files = os.listdir(class_dir)[:4]  # Display the first 4 images for each class
    for j, image_file in enumerate(image_files):
        image_path = os.path.join(class_dir, image_file)
        img = plt.imread(image_path)
        gray_img = rgb_to_gray(img)
        axes[i, j].imshow(gray_img, cmap='gray')
        axes[i, j].axis('off')
        axes[i, j].set_title(f"Class: {class_name}")
plt.show()
```




```
fig, axes = plt.subplots(len(class_names), 8, figsize=(20, 3 * len(class_names)))

for i, class_name in enumerate(class_names):
    class_dir = os.path.join(train_dir, class_name)
    image_files = os.listdir(class_dir)[:4]

    for j, image_file in enumerate(image_files):
        image_path = os.path.join(class_dir, image_file)
        img = mpimg.imread(image_path)
        gray_img = np.dot(img[..., :3], [0.2989, 0.5870, 0.1140])

        axes[i, 2 * j].imshow(img)
        axes[i, 2 * j].axis('off')
        axes[i, 2 * j].set_title(f"Class: {class_name}")
        axes[i, 2 * j].set_aspect('auto')

        axes[i, 2 * j + 1].imshow(gray_img, cmap='gray')
        axes[i, 2 * j + 1].axis('off')
        axes[i, 2 * j + 1].set_title(f"Class: {class_name} (Grayscale)")
        axes[i, 2 * j + 1].set_aspect('auto')

plt.subplots_adjust(wspace=0.5, hspace=0.5)
plt.show()
```


```
!pip install --upgrade tensorflow
```

```
Requirement already satisfied: tensorflow in /usr/local/lib/python3.10/dist-packages (2.15.0.post1)
Requirement already satisfied: absl-py>=1.0.0 in /usr/local/lib/python3.10/dist-packages (from tensorflow) (1.4.0)
Requirement already satisfied: astunparse>=1.6.0 in /usr/local/lib/python3.10/dist-packages (from tensorflow) (1.6.3)
Requirement already satisfied: flatbuffers>=23.5.26 in /usr/local/lib/python3.10/dist-packages (from tensorflow) (23.5.26)
Requirement already satisfied: gast!=0.5.0,!=0.5.1,!=0.5.2,>=0.2.1 in /usr/local/lib/python3.10/dist-packages (from tensorflow) (0.5
Requirement already satisfied: google-pasta>=0.1.1 in /usr/local/lib/python3.10/dist-packages (from tensorflow) (0.2.0)
Requirement already satisfied: h5py>=2.9.0 in /usr/local/lib/python3.10/dist-packages (from tensorflow) (3.9.0)
Requirement already satisfied: libclang>=13.0.0 in /usr/local/lib/python3.10/dist-packages (from tensorflow) (16.0.6)
Requirement already satisfied: ml-dtypes~=0.2.0 in /usr/local/lib/python3.10/dist-packages (from tensorflow) (0.2.0)
Requirement already satisfied: numpy<2.0.0,>=1.23.5 in /usr/local/lib/python3.10/dist-packages (from tensorflow) (1.23.5)
Requirement already satisfied: opt-einsum>=2.3.2 in /usr/local/lib/python3.10/dist-packages (from tensorflow) (3.3.0)
Requirement already satisfied: packaging in /usr/local/lib/python3.10/dist-packages (from tensorflow) (23.2)
Requirement already satisfied: protobuf!=4.21.0,!=4.21.1,!=4.21.2,!=4.21.3,!=4.21.4,!=4.21.5,<5.0.0dev,>=3.20.3 in /usr/local/lib/py
Requirement already satisfied: setuptools in /usr/local/lib/python3.10/dist-packages (from tensorflow) (67.7.2)
Requirement already satisfied: six>=1.12.0 in /usr/local/lib/python3.10/dist-packages (from tensorflow) (1.16.0)
Requirement already satisfied: termcolor>=1.1.0 in /usr/local/lib/python3.10/dist-packages (from tensorflow) (2.4.0)
Requirement already satisfied: typing-extensions>=3.6.6 in /usr/local/lib/python3.10/dist-packages (from tensorflow) (4.5.0)
Requirement already satisfied: wrapt<1.15,>=1.11.0 in /usr/local/lib/python3.10/dist-packages (from tensorflow) (1.14.1)
Requirement already satisfied: tensorflow-io-gcs-filesystem>=0.23.1 in /usr/local/lib/python3.10/dist-packages (from tensorflow) (0
Requirement already satisfied: grpcio<2.0,>=1.24.3 in /usr/local/lib/python3.10/dist-packages (from tensorflow) (1.60.0)
Requirement already satisfied: tensorboard<2.16,>=2.15 in /usr/local/lib/python3.10/dist-packages (from tensorflow) (2.15.1)
Requirement already satisfied: tensorflow-estimator<2.16,>=2.15.0 in /usr/local/lib/python3.10/dist-packages (from tensorflow) (2.15
Requirement already satisfied: keras<2.16,>=2.15.0 in /usr/local/lib/python3.10/dist-packages (from tensorflow) (2.15.0)
Requirement already satisfied: wheel<1.0,>=0.23.0 in /usr/local/lib/python3.10/dist-packages (from astunparse>=1.6.0->tensorflow) (0
Requirement already satisfied: google-auth<3,>=1.6.3 in /usr/local/lib/python3.10/dist-packages (from tensorboard<2.16,>=2.15->tensc
Requirement already satisfied: google-auth-oauthlib<2,>=0.5 in /usr/local/lib/python3.10/dist-packages (from tensorboard<2.16,>=2.15
Requirement already satisfied: markdown>=2.6.8 in /usr/local/lib/python3.10/dist-packages (from tensorboard<2.16,>=2.15->tensorflow
Requirement already satisfied: requests<3,>=2.21.0 in /usr/local/lib/python3.10/dist-packages (from tensorboard<2.16,>=2.15->tensorf
Requirement already satisfied: tensorboard-data-server<0.8.0,>=0.7.0 in /usr/local/lib/python3.10/dist-packages (from tensorboard<2
Requirement already satisfied: werkzeug>=1.0.1 in /usr/local/lib/python3.10/dist-packages (from tensorboard<2.16,>=2.15->tensorflow
Requirement already satisfied: cachetools<6.0,>=2.0.0 in /usr/local/lib/python3.10/dist-packages (from google-auth<3,>=1.6.3->tensor
Requirement already satisfied: pyasn1-modules>=0.2.1 in /usr/local/lib/python3.10/dist-packages (from google-auth<3,>=1.6.3->tensorb
Requirement already satisfied: rsa<5,>=3.1.4 in /usr/local/lib/python3.10/dist-packages (from google-auth<3,>=1.6.3->tensorboard<2.1
Requirement already satisfied: requests-oauthlib>=0.7.0 in /usr/local/lib/python3.10/dist-packages (from google-auth-oauthlib<2,>=0
Requirement already satisfied: charset-normalizer<4,>=2 in /usr/local/lib/python3.10/dist-packages (from requests<3,>=2.21.0->tensor
Requirement already satisfied: idna<4,>=2.5 in /usr/local/lib/python3.10/dist-packages (from requests<3,>=2.21.0->tensorboard<2.16,>
Requirement already satisfied: urllib3<3,>=1.21.1 in /usr/local/lib/python3.10/dist-packages (from requests<3,>=2.21.0->tensorboard<
Requirement already satisfied: certifi>=2017.4.17 in /usr/local/lib/python3.10/dist-packages (from requests<3,>=2.21.0->tensorboard<
Requirement already satisfied: MarkupSafe>=2.1.1 in /usr/local/lib/python3.10/dist-packages (from werkzeug>=1.0.1->tensorboard<2.16,
Requirement already satisfied: pyasn1<0.6.0,>=0.4.6 in /usr/local/lib/python3.10/dist-packages (from pyasn1-modules>=0.2.1->google-a
Requirement already satisfied: oauthlib>=3.0.0 in /usr/local/lib/python3.10/dist-packages (from requests-oauthlib>=0.7.0->google-aut
```

```
#from keras.applications import ResNet50
from tensorflow.keras.applications.resnet50 import ResNet50

res_model = Sequential()

res = ResNet50(include_top=False, weights='imagenet',input_shape = (32, 32, 3))

res_model.add(res)

# Add new layers
res_model.add(Flatten())
res_model.add(Dense(512, activation='relu'))
res_model.add(Dropout(0.5))
res_model.add(Dense(256, activation='relu'))
res_model.add(Dropout(0.5))
res_model.add(Dense(128, activation='relu'))
res_model.add(Dropout(0.3))
# Output layer
res_model.add(Dense(10, activation='softmax'))

res_model.summary()
```

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 resnet50 (Functional)       (None, 1, 1, 2048)        23587712

 flatten (Flatten)           (None, 2048)              0

 dense (Dense)               (None, 512)               1049088

 dropout (Dropout)           (None, 512)               0

 dense_1 (Dense)             (None, 256)               131328

 dropout_1 (Dropout)         (None, 256)               0

 dense_2 (Dense)             (None, 128)               32896

 dropout_2 (Dropout)         (None, 128)               0

 dense_3 (Dense)             (None, 10)                1290

=================================================================
Total params: 24802314 (94.61 MB)
Trainable params: 24749194 (94.41 MB)
Non-trainable params: 53120 (207.50 KB)
_________________________________________________________________
```

```
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, Dropout, Flatten
from tensorflow.keras.applications.resnet50 import ResNet50
from tensorflow.keras.layers import GlobalAveragePooling2D
from tensorflow.keras.preprocessing.image import ImageDataGenerator
from tensorflow.keras.callbacks import EarlyStopping, ModelCheckpoint
from tensorflow.keras.preprocessing import image
import matplotlib.pyplot as plt
import numpy as np
import os

# Define the image size and batch size for the dataset
image_size = (224, 224)
batch_size = 32

# Define the paths to your training and test datasets
base_dir = "/content/ML-Project/assign_classification"
train_dir = os.path.join(base_dir, 'train')
test_dir = os.path.join(base_dir, 'test')  # Change 'validation' to 'test'

# Create the training dataset using TensorFlow's image_dataset_from_directory
train_dataset = image_dataset_from_directory(
    directory=train_dir,
    labels='inferred',
    label_mode='categorical',
    validation_split=0.1,
    subset='training',
    seed=1,
    image_size=image_size,
    batch_size=batch_size,
    shuffle=True
)

# Create the test dataset with similar parameters
test_dataset = image_dataset_from_directory(
    directory=test_dir,  # Change 'validation' to 'test'
    labels='inferred',
    label_mode='categorical',
    image_size=image_size,
    batch_size=batch_size,
    shuffle=False
)

# Display class names
class_names = train_dataset.class_names
print("Class names:", class_names)

# Create a Sequential model
res_model = Sequential()

# Load the pre-trained ResNet50 model (excluding the top layer)
res = ResNet50(include_top=False, weights='imagenet', input_shape=(224, 224, 3))

# Add ResNet50 to the model
res_model.add(res)

# Add Global Average Pooling layer
res_model.add(GlobalAveragePooling2D())

# Add new layers with dropout
res_model.add(Dense(512, activation='relu'))
res_model.add(Dropout(0.5))  # Adding dropout layer
res_model.add(Dense(256, activation='relu'))
res_model.add(Dropout(0.5))  # Adding dropout layer
res_model.add(Dense(128, activation='relu'))
res_model.add(Dropout(0.3))  # Adding dropout layer

# Output layer with the correct number of units (assuming 3 classes)
res_model.add(Dense(3, activation='softmax'))

# Compile the model with SGD optimizer and categorical crossentropy loss
res_model.compile(optimizer='sgd', loss='categorical_crossentropy', metrics=['accuracy'])

# Define callbacks
early_stopping = EarlyStopping(monitor='val_loss', patience=5, restore_best_weights=True)
checkpoint_path = "model_checkpoint.h5"
model_checkpoint = ModelCheckpoint(checkpoint_path, save_best_only=True, save_weights_only=False, monitor='val_loss', mode='min', verbos

# Train the model without data augmentation
history = res_model.fit(
    train_dataset,
    epochs=15,
```

```
    validation_data=test_dataset,  # Change 'validation' to 'test'
    callbacks=[early_stopping, model_checkpoint]
)
```

```
Epoch 1: val_loss improved from inf to 1.07982, saving model to model_checkpoint.h5
4/4 [==============================] - 120s 24s/step - loss: 1.3095 - accuracy: 0.3426 - val_loss: 1.0798 - val_accuracy: 0.5200
Epoch 2/15
4/4 [==============================] - ETA: 0s - loss: 1.2190 - accuracy: 0.4167
Epoch 2: val_loss improved from 1.07982 to 1.07734, saving model to model_checkpoint.h5
4/4 [==============================] - 104s 25s/step - loss: 1.2190 - accuracy: 0.4167 - val_loss: 1.0773 - val_accuracy: 0.5200
Epoch 3/15
4/4 [==============================] - ETA: 0s - loss: 0.9097 - accuracy: 0.5278
Epoch 3: val_loss improved from 1.07734 to 1.05451, saving model to model_checkpoint.h5
4/4 [==============================] - 96s 23s/step - loss: 0.9097 - accuracy: 0.5278 - val_loss: 1.0545 - val_accuracy: 0.6000
Epoch 4/15
4/4 [==============================] - ETA: 0s - loss: 0.8847 - accuracy: 0.6111
Epoch 4: val_loss improved from 1.05451 to 1.05360, saving model to model_checkpoint.h5
4/4 [==============================] - 99s 23s/step - loss: 0.8847 - accuracy: 0.6111 - val_loss: 1.0536 - val_accuracy: 0.5600
Epoch 5/15
4/4 [==============================] - ETA: 0s - loss: 0.8022 - accuracy: 0.6296
Epoch 5: val_loss improved from 1.05360 to 0.88021, saving model to model_checkpoint.h5
4/4 [==============================] - 97s 23s/step - loss: 0.8022 - accuracy: 0.6296 - val_loss: 0.8802 - val_accuracy: 0.7200
Epoch 6/15
4/4 [==============================] - ETA: 0s - loss: 0.7166 - accuracy: 0.6667
Epoch 6: val_loss did not improve from 0.88021
4/4 [==============================] - 96s 23s/step - loss: 0.7166 - accuracy: 0.6667 - val_loss: 1.0321 - val_accuracy: 0.5200
Epoch 7/15
4/4 [==============================] - ETA: 0s - loss: 0.7302 - accuracy: 0.6852
Epoch 7: val_loss did not improve from 0.88021
4/4 [==============================] - 105s 25s/step - loss: 0.7302 - accuracy: 0.6852 - val_loss: 0.8865 - val_accuracy: 0.6000
Epoch 8/15
4/4 [==============================] - ETA: 0s - loss: 0.6334 - accuracy: 0.7593
Epoch 8: val_loss improved from 0.88021 to 0.88006, saving model to model_checkpoint.h5
4/4 [==============================] - 101s 24s/step - loss: 0.6334 - accuracy: 0.7593 - val_loss: 0.8801 - val_accuracy: 0.6400
Epoch 9/15
4/4 [==============================] - ETA: 0s - loss: 0.5098 - accuracy: 0.7870
Epoch 9: val_loss improved from 0.88006 to 0.84460, saving model to model_checkpoint.h5
4/4 [==============================] - 100s 24s/step - loss: 0.5098 - accuracy: 0.7870 - val_loss: 0.8446 - val_accuracy: 0.7600
Epoch 10/15
4/4 [==============================] - ETA: 0s - loss: 0.4439 - accuracy: 0.8148
Epoch 10: val_loss did not improve from 0.84460
4/4 [==============================] - 100s 23s/step - loss: 0.4439 - accuracy: 0.8148 - val_loss: 0.9154 - val_accuracy: 0.7200
Epoch 11/15
4/4 [==============================] - ETA: 0s - loss: 0.4444 - accuracy: 0.8241
Epoch 11: val_loss did not improve from 0.84460
4/4 [==============================] - 96s 22s/step - loss: 0.4444 - accuracy: 0.8241 - val_loss: 0.9568 - val_accuracy: 0.6800
Epoch 12/15
4/4 [==============================] - ETA: 0s - loss: 0.3020 - accuracy: 0.9259
Epoch 12: val_loss improved from 0.84460 to 0.84293, saving model to model_checkpoint.h5
4/4 [==============================] - 103s 23s/step - loss: 0.3020 - accuracy: 0.9259 - val_loss: 0.8429 - val_accuracy: 0.7600
Epoch 13/15
4/4 [==============================] - ETA: 0s - loss: 0.3171 - accuracy: 0.8889
Epoch 13: val_loss did not improve from 0.84293
4/4 [==============================] - 95s 22s/step - loss: 0.3171 - accuracy: 0.8889 - val_loss: 0.8856 - val_accuracy: 0.7200
Epoch 14/15
4/4 [==============================] - ETA: 0s - loss: 0.2209 - accuracy: 0.9352
Epoch 14: val_loss did not improve from 0.84293
4/4 [==============================] - 98s 22s/step - loss: 0.2209 - accuracy: 0.9352 - val_loss: 1.0046 - val_accuracy: 0.7200
Epoch 15/15
4/4 [==============================] - ETA: 0s - loss: 0.1957 - accuracy: 0.9907
Epoch 15: val_loss did not improve from 0.84293
4/4 [==============================] - 97s 23s/step - loss: 0.1957 - accuracy: 0.9907 - val_loss: 0.9457 - val_accuracy: 0.7600
```

```
from sklearn.metrics import classification_report, confusion_matrix, roc_auc_score
import numpy as np


# Make predictions on the test data
y_pred_probs = res_model.predict(test_dataset)
y_pred = np.argmax(y_pred_probs, axis=1)

# Convert one-hot encoded labels to class indices
y_true = np.argmax(np.concatenate([y for x, y in test_dataset]), axis=1)

# Classification Report
classification_rep = classification_report(y_true, y_pred, target_names=class_names)
print("Classification Report:\n", classification_rep)

# Confusion Matrix
conf_matrix = confusion_matrix(y_true, y_pred)
print("Confusion Matrix:\n", conf_matrix)
```

```
1/1 [==============================] - 7s 7s/step
Classification Report:
              precision    recall  f1-score   support
```

```
              bike       0.89      0.80      0.84        10
               car       0.67      1.00      0.80        10
            others       1.00      0.20      0.33         5

          accuracy                           0.76        25
         macro avg       0.85      0.67      0.66        25
      weighted avg       0.82      0.76      0.72        25

      Confusion Matrix:
       [[ 8  2  0]
       [ 0 10  0]
       [ 1  3  1]]
```

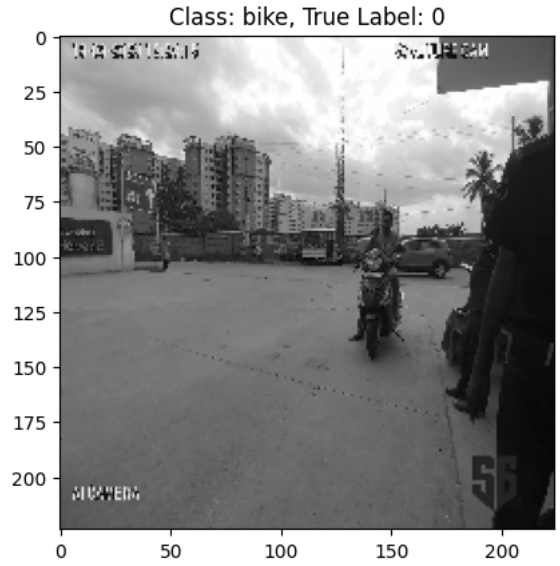
```
import numpy as np
import tensorflow as tf
from tensorflow.keras.preprocessing import image
from sklearn.metrics import confusion_matrix, classification_report, roc_auc_score
import matplotlib.pyplot as plt
import seaborn as sns

# Load the pre-trained ResNet50 model
model = tf.keras.applications.ResNet50(weights='imagenet', include_top=True)

# Define the category mapping
category_mapping = {
    0: 'car',
    1: 'bike',
    2: 'others'
}
```

```
import os
import numpy as np
import tensorflow as tf
from tensorflow.keras.preprocessing import image
from sklearn.metrics import confusion_matrix, classification_report, roc_auc_score
from sklearn.metrics import roc_curve

# Load the pre-trained ResNet50 model
model = tf.keras.applications.ResNet50(weights='imagenet', include_top=True)

# Define the base path for test data
test_base_path = "/content/ML-Project/assign_classification/test"

# Specify the class name
class_name = "bike"

# Construct the path for the specific class
class_directory = os.path.join(test_base_path, class_name)

# List all image paths in the class directory
test_data_paths = [os.path.join(class_directory, img) for img in os.listdir(class_directory)]

test_data = []
true_labels = []

# Define the mapping from class names to numeric labels
label_mapping = {"bike": 0, "car": 1, "others": 2}

for path in test_data_paths:
    img = image.load_img(path, target_size=(224, 224))
    img_array = image.img_to_array(img)
    img_array = np.expand_dims(img_array, axis=0)
    img_array = tf.keras.applications.resnet50.preprocess_input(img_array)  # Fix here

    test_data.append(img_array)

    label = path.split("/")[-2]
    true_labels.append(label_mapping[label])

test_data = np.vstack(test_data)
true_labels = np.array(true_labels)
```

```
import matplotlib.pyplot as plt
from skimage.color import rgb2gray

# Print the first few true labels
print("True Labels:", true_labels[:5])

# Convert the image to grayscale
```

```
gray_image = rgb2gray(test_data[0])

# Visualize the first grayscale image without normalization and clipping
plt.imshow(gray_image, cmap='gray')
plt.title(f"Class: {class_name}, True Label: {true_labels[0]}")
plt.show()
```

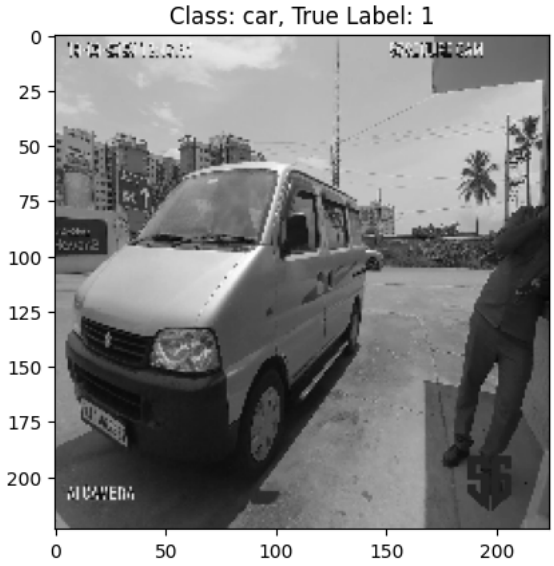
```
True Labels: [0 0 0 0 0]
```




```
import os
import numpy as np
import tensorflow as tf
from tensorflow.keras.preprocessing import image
from sklearn.metrics import confusion_matrix, classification_report, roc_auc_score
from sklearn.metrics import roc_curve

# Load the pre-trained ResNet50 model
model = tf.keras.applications.ResNet50(weights='imagenet', include_top=True)

# Define the base path for test data
test_base_path = "/content/ML-Project/assign_classification/test"

# Specify the class name
class_name = "car"

# Construct the path for the specific class
class_directory = os.path.join(test_base_path, class_name)

# List all image paths in the class directory
test_data_paths = [os.path.join(class_directory, img) for img in os.listdir(class_directory)]

test_data = []
true_labels = []

# Define the mapping from class names to numeric labels
label_mapping = {"bike": 0, "car": 1, "others": 2}

for path in test_data_paths:
    img = image.load_img(path, target_size=(224, 224))
    img_array = image.img_to_array(img)
    img_array = np.expand_dims(img_array, axis=0)
    img_array = tf.keras.applications.resnet50.preprocess_input(img_array)  # Fix here

    test_data.append(img_array)

    label = path.split("/")[-2]
    true_labels.append(label_mapping[label])

test_data = np.vstack(test_data)
true_labels = np.array(true_labels)



import matplotlib.pyplot as plt
from skimage.color import rgb2gray

# Print the first few true labels
print("True Labels:", true_labels[:5])
```

```
# Convert the image to grayscale
gray_image = rgb2gray(test_data[3])

# Visualize the first grayscale image without normalization and clipping
plt.imshow(gray_image, cmap='gray')
plt.title(f"Class: {class_name}, True Label: {true_labels[0]}")
plt.show()
```

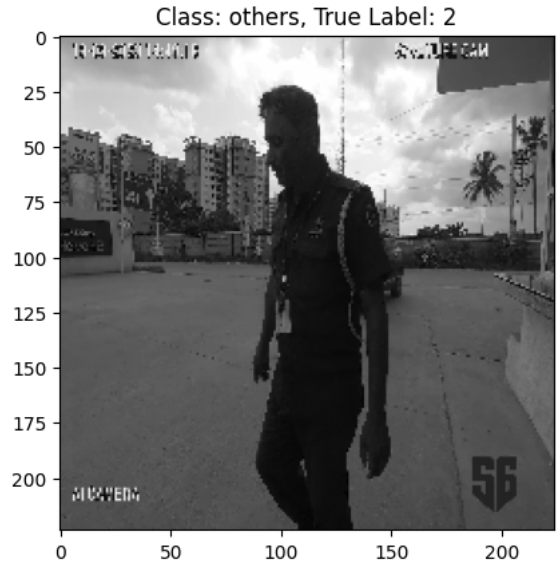
```
True Labels: [1 1 1 1 1]
```




```
import os
import numpy as np
import tensorflow as tf
from tensorflow.keras.preprocessing import image
from sklearn.metrics import confusion_matrix, classification_report, roc_auc_score
from sklearn.metrics import roc_curve

# Load the pre-trained ResNet50 model
model = tf.keras.applications.ResNet50(weights='imagenet', include_top=True)

# Define the base path for test data
test_base_path = "/content/ML-Project/assign_classification/test"

# Specify the class name
class_name = "others"

# Construct the path for the specific class
class_directory = os.path.join(test_base_path, class_name)

# List all image paths in the class directory
test_data_paths = [os.path.join(class_directory, img) for img in os.listdir(class_directory)]

test_data = []
true_labels = []

# Define the mapping from class names to numeric labels
label_mapping = {"bike": 0, "car": 1, "others": 2}

for path in test_data_paths:
    img = image.load_img(path, target_size=(224, 224))
    img_array = image.img_to_array(img)
    img_array = np.expand_dims(img_array, axis=0)
    img_array = tf.keras.applications.resnet50.preprocess_input(img_array)  # Fix here

    test_data.append(img_array)

    label = path.split("/")[-2]
    true_labels.append(label_mapping[label])

test_data = np.vstack(test_data)
true_labels = np.array(true_labels)



import matplotlib.pyplot as plt
from skimage.color import rgb2gray

# Print the first few true labels
```

```
print("True Labels:", true_labels[:5])

# Convert the image to grayscale
gray_image = rgb2gray(test_data[3])

# Visualize the first grayscale image without normalization and clipping
plt.imshow(gray_image, cmap='gray')
plt.title(f"Class: {class_name}, True Label: {true_labels[0]}")
plt.show()
```

```
True Labels: [2 2 2 2 2]
```

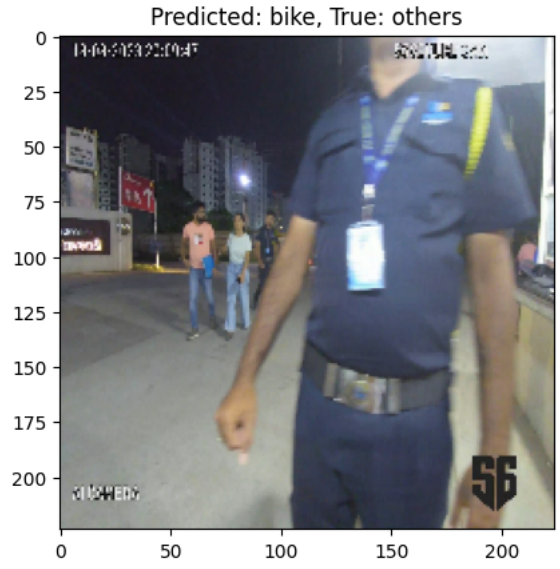



```
from sklearn.metrics import classification_report, confusion_matrix
from sklearn.metrics import precision_recall_curve, auc
import matplotlib.pyplot as plt
import seaborn as sns
from tensorflow.keras.models import load_model
import numpy as np

# Load the trained model
res_model = load_model("model_checkpoint.h5")  # Use the correct path

# Load and preprocess the test dataset
test_dir = os.path.join(base_dir, 'test')
test_dataset = image_dataset_from_directory(
    directory=test_dir,
    labels='inferred',
    label_mode='categorical',
    image_size=image_size,
    batch_size=batch_size,
    shuffle=False
)

# Extract true labels and predictions
true_labels = np.concatenate([y for x, y in test_dataset], axis=0)
predictions = res_model.predict(test_dataset)

# Classification Report
report = classification_report(np.argmax(true_labels, axis=1), np.argmax(predictions, axis=1), target_names=class_names)
print("Classification Report:\n", report)

# Confusion Matrix
conf_matrix = confusion_matrix(np.argmax(true_labels, axis=1), np.argmax(predictions, axis=1))
plt.figure(figsize=(8, 6))
sns.heatmap(conf_matrix, annot=True, fmt='d', cmap='Blues', xticklabels=class_names, yticklabels=class_names)
plt.title("Confusion Matrix")
plt.xlabel("Predicted")
plt.ylabel("True")
plt.show()

# Precision-Recall Curve and AUC
precision = dict()
recall = dict()
auc_score = dict()

for i in range(len(class_names)):
    precision[i], recall[i], _ = precision_recall_curve(true_labels[:, i], predictions[:, i])
    auc_score[i] = auc(recall[i], precision[i])
```

```
# Plot Precision-Recall Curve
plt.figure(figsize=(12, 8))
for i in range(len(class_names)):
    plt.plot(recall[i], precision[i], label=f'{class_names[i]} (AUC = {auc_score[i]:.2f})')

plt.xlabel('Recall')
plt.ylabel('Precision')
plt.title('Precision-Recall Curve')
plt.legend()
plt.show()
```

```
Classification Report:
              precision    recall  f1-score   support

        bike       0.67      0.80      0.73        10
         car       0.83      1.00      0.91        10
      others       1.00      0.20      0.33         5

    accuracy                           0.76        25
   macro avg       0.83      0.67      0.66        25
weighted avg       0.80      0.76      0.72        25
```

```
# Identify and visualize a few mistakes
mistake_indices = np.where(predicted_labels != true_labels)[0]
for idx in mistake_indices[:5]:  # Visualize the first 5 mistakes
    # Use the take method to extract the specific element from the dataset
    images, _ = validation_dataset.take(idx + 1).as_numpy_iterator().next()

    image = images[0]  # Extract the first (and only) image from the batch
    predicted_class = predicted_labels[idx]
    true_class = true_labels[idx]

    plt.imshow(image.astype(np.uint8))
    plt.title(f'Predicted: {class_names[predicted_class]}, True: {class_names[true_class]}')
    plt.show()
```




```
import matplotlib.pyplot as plt
import numpy as np
from tensorflow.keras.models import Model
from tensorflow.keras.preprocessing import image
from tensorflow.keras.applications.resnet50 import preprocess_input

def visualize_feature_maps(model, layer_number, image_path):
    # Get the specified layer from the model
    layer = model.layers[layer_number]

    # Create a sub-model that includes layers up to the specified layer
    sub_model = Model(inputs=model.inputs, outputs=layer.output)

    # Load and preprocess an example image
    img = image.load_img(image_path, target_size=(224, 224))
    img_array = image.img_to_array(img)
    img_array = np.expand_dims(img_array, axis=0)
    img_array = preprocess_input(img_array)

    # Get the feature maps for the input image
    feature_maps = sub_model.predict(img_array)

    # Number of feature maps in the layer
    num_feature_maps = feature_maps.shape[-1]

    # Determine the layout for subplots
    rows = int(np.sqrt(num_feature_maps))
    cols = int(np.ceil(num_feature_maps / rows))

    # Plot each feature map
    plt.figure(figsize=(15, 15))
    for i in range(num_feature_maps):
        plt.subplot(rows, cols, i + 1)
        plt.imshow(feature_maps[0, :, :, i], cmap='viridis')
```

```
        plt.axis('off')
    plt.show()

def visualize_filters(model, layer_number):
    # Get the specified layer from the model
    layer = model.layers[layer_number]

    # Get the filters (weights) of the layer
    filters, _ = layer.get_weights()

    # Normalize filter values to between 0 and 1 for visualization
    filters = (filters - filters.min()) / (filters.max() - filters.min())

    # Number of filters in the layer
    num_filters = filters.shape[-1]

    # Determine the layout for subplots
    rows = int(np.sqrt(num_filters))
    cols = int(np.ceil(num_filters / rows))

    # Plot each filter
    plt.figure(figsize=(15, 15))
    for i in range(num_filters):
        plt.subplot(rows, cols, i + 1)
        plt.imshow(filters[:, :, 0, i], cmap='gray')
        plt.axis('off')
    plt.show()

# Visualize feature maps for the 2nd layer
visualize_feature_maps(resnet_model, layer_number=2, image_path='/content/ML-Project/assign_classification/test/car/pfva_dataset_file_G_1

# Visualize filters for the 2nd layer
visualize_filters(resnet_model, layer_number=2)
```

```
1/1 [==============================] - 0s 108ms/step
```