Task -Clsssification of bottles

To connect with drive

```
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=Tru
```

Import library files

```
import os
import cv2
import pickle
import seaborn as sns
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from google.colab.patches import cv2_imshow
import warnings
warnings.filterwarnings('ignore') # Hide all warnings
from tensorflow import keras
from tensorflow.keras.applications.inception_v3 import InceptionV3
from tensorflow.keras.preprocessing.image import ImageDataGenerator
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Conv2D, Flatten, MaxPooling2D, Dense, Dropout, GlobalAveragePooling2D
from tensorflow.keras import optimizers, losses
from tensorflow.keras.callbacks import ModelCheckpoint
from tensorflow.keras.preprocessing import image
import tensorflow as tf
from tensorflow.keras import layers
from tensorflow.keras.layers import Dense,Dropout,BatchNormalization
from tensorflow.keras.preprocessing import image
from tensorflow.keras.models import Sequential
from tensorflow.keras.preprocessing.image import ImageDataGenerator
from tensorflow.keras import optimizers
from keras.layers.pooling import GlobalAveragePooling2D
from tensorflow.keras import Model,layers
from tensorflow.keras.optimizers import Adam
from sklearn.metrics import confusion_matrix
```

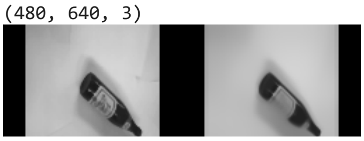
Dataset Description:

My dataset consist of cans,glass_bottles,plastic_bottles

Read a single image and resize & remove noise

```
image = cv2.imread('/content/drive/MyDrive/bottle/glass_bottles/glass_bottles/bdtmp.jpg')
print(image.shape)
input_size = 128
image_size = (input_size, input_size)
image = cv2.resize(image, image_size)
image1 = cv2.fastNlMeansDenoising(image,None,20,7,21)
Hori = np.concatenate((image, image1), axis=1)
cv2_imshow(Hori)
```

```
(480, 640, 3)
```



Preprocessing stage

Read each images and resize & remove noise each images of the dataset

```
data = []
labels = []
# Access the directory and sub-directories and so on
directory = "/content/drive/MyDrive/bottle"

# Extract all images file inside the folders and stored them into list
for sub_folder in os.listdir(directory):
    sub_folder_path = os.path.join(directory, sub_folder)
    for sub_sub_folder in os.listdir(sub_folder_path):
        sub_sub_folder_path = os.path.join(sub_folder_path, sub_sub_folder)
        for image_file in os.listdir(sub_sub_folder_path):
            if image_file.endswith(".jpg") or image_file.endswith(".png"): # Check if the file ends with either '.jped' or '.png'
                image_path = os.path.join(sub_sub_folder_path, image_file)
                # Read the image using OpenCV
                image = cv2.imread(image_path) #the decoded images stored in **B G R** order.
                # Resize the image to a standard size
                image = cv2.resize(image, image_size)
                image = cv2.fastNlMeansDenoising(image,None,20,7,21)
                # Append the image to the data list
                data.append(image)
                # Append the label to the labels list
                labels.append(sub_folder)

# Convert the data and labels lists into numpy arrays
data = np.array(data)
labels = np.array(labels)

# Print the dimension of dataset
print(f'data shape:{data.shape}')
print(f'labels shape:{labels.shape}')
```

```
data shape:(8215, 128, 128, 3)
labels shape:(8215,)
```

See how many numbers of each labels

```
df = pd.DataFrame({"label":labels})
df.value_counts()
```

```
label
plastic_bottles    7576
cans                495
glass_bottles       144
dtype: int64
```

```
df = pd.DataFrame({" bottle label":labels})
df.value_counts().plot(kind='bar')
plt.xticks(rotation = 0)
plt.show()
```

Generate augmented data

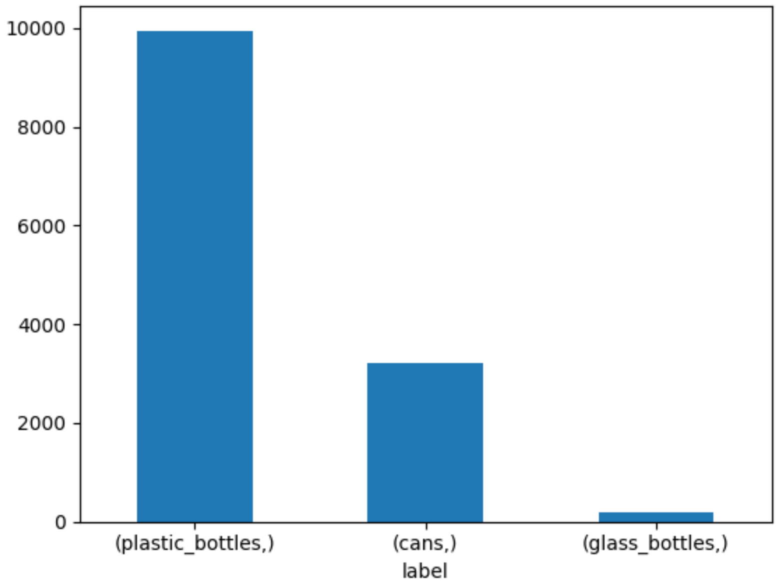
```
from keras.preprocessing.image import ImageDataGenerator

# Load the data
X = data # array of preprocessed data
y = labels # array of labels
n_gen = 40

# Create data generator
datagen = ImageDataGenerator(
        rotation_range=0, #0
        width_shift_range=0.2,
        height_shift_range=0.2,
        shear_range=0.2,
        zoom_range=0.2,
        horizontal_flip=True,
        fill_mode='nearest')

# Fit the data generator on the data
datagen.fit(X)

# Generate augmented data
X_augmented, y_augmented = [], []

# resampling with equaly labels ratio

# With resampling
for X_batch, y_batch in datagen.flow(X[:308], y[:308], batch_size=32):
    X_augmented.append(X_batch)
    y_augmented.append(y_batch)
    if len(X_augmented) >= n_gen: # Setting generated augmented data
        break

for X_batch, y_batch in datagen.flow(X[308:447], y[308:447], batch_size=32):
    X_augmented.append(X_batch)
    y_augmented.append(y_batch)
    if len(X_augmented) >= n_gen*2.3: # Setting generated augmented data
        break

for X_batch, y_batch in datagen.flow(X[447:], y[447:], batch_size=32):
    X_augmented.append(X_batch)
    y_augmented.append(y_batch)
    if len(X_augmented) >= n_gen*4.2: # Setting generated augmented data
        break

# Concatenate augmented data with original data
data = np.concatenate((X, np.concatenate(X_augmented)))
labels = np.concatenate((y, np.concatenate(y_augmented)))

print(f"data augmented shape : {data.shape}")
print(f"labels augmented shape : {labels.shape}")

import pandas as pd
df = pd.DataFrame({"label":labels})
df.value_counts()
```

```
data augmented shape : (13333, 128, 128, 3)
labels augmented shape : (13333,)
label
plastic_bottles    9948
cans               3196
```

```
glass_bottles       189
dtype: int64
```

See how many numbers of each labels after I regenerated data.

```
df = pd.DataFrame({"label":labels})
df.value_counts().plot(kind='bar')
plt.xticks(rotation = 0) # Rotates X-Axis Ticks by 45-degrees
plt.show()
```



spliting dataset as train and test

```
from sklearn.model_selection import train_test_split

X_train, X_test, y_train, y_test = train_test_split(data, labels, test_size=0.2, random_state=42)

data = X_train # Split training data
labels = y_train # Split training labels

X_test = X_test # Test data
y_test = y_test # Test labels
```

```
import pandas as pd

print(f'data shape:{data.shape}')
print(f'labels shape:{labels.shape}')
df = pd.DataFrame({"label":labels})
print(df.value_counts())
print("")
print(f'test_date shape:{X_test.shape}')
print(f'test_labels shape:{y_test.shape}')
df = pd.DataFrame({"test_labels":y_test})
print(df.value_counts())
```

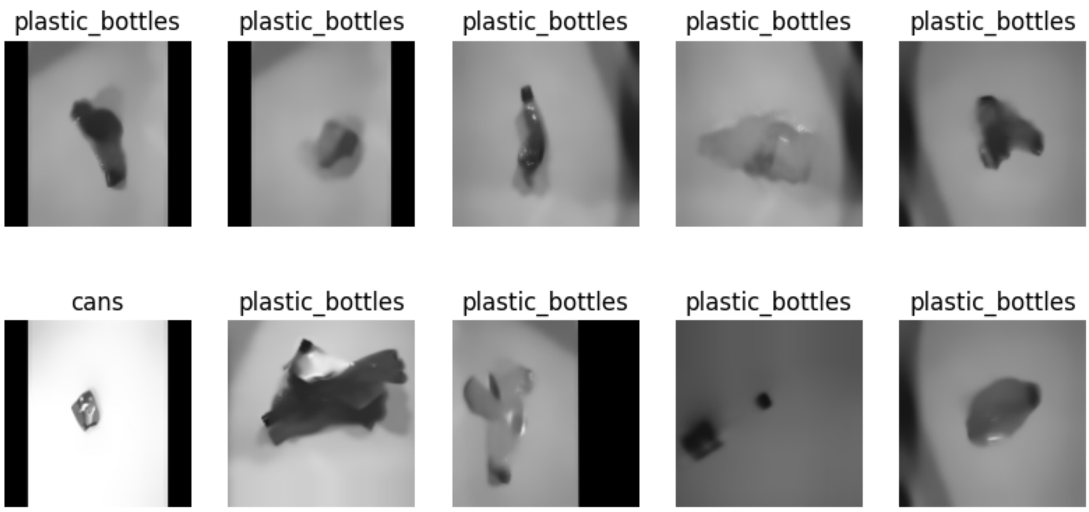
```
data shape:(10666, 128, 128, 3)
labels shape:(10666,)
label
plastic_bottles    7997
cans               2520
glass_bottles       149
dtype: int64

test_date shape:(2667, 128, 128, 3)
test_labels shape:(2667,)
test_labels
plastic_bottles    1951
cans                676
glass_bottles        40
dtype: int64
```

```
# Normalize the pixel values to a range between 0 and 1
data = data / 255.0
X_test = X_test / 225.0
```

```
labels = labels
# Convert the labels into one-hot encoded arrays
labels_one_hot = np.zeros((labels.shape[0], 3))

for i, label in enumerate(labels):
    if label == "plastic_bottles":
        labels_one_hot[i, 0] = 1
    elif label == "cans":
        labels_one_hot[i, 1] = 1
    else:
        labels_one_hot[i, 2] = 1
```

Show a sample of images from the dataset

```
data = data

# choose 20 random indices
indices = np.random.randint(0, len(data), 10)

# Get 20 sample images
sample_images = data[indices]

# Plot the images
fig = plt.figure(figsize=(10,10))
for i, img in enumerate(sample_images):
    plt.subplot(4, 5, i+1)
    plt.imshow(img)
    plt.axis('off')
    plt.title(labels[indices[i]])

plt.show()
```



Create my CNN model

```
def run_custom_model(batch_size, epochs):

    import tensorflow as tf
    from tensorflow import keras
    from tensorflow.keras import layers
    from tensorflow.keras.optimizers import Adam, SGD
    from tensorflow.keras.callbacks import ModelCheckpoint

    # set seed value for randomization
    # np.random.seed(42)
    tf.random.set_seed(42)
```

```
    # Build the model using a Convolutional Neural Network
    model = keras.Sequential([
        keras.layers.Conv2D(32, (3,3), activation='relu', input_shape=(input_size,input_size,3)),
        keras.layers.Conv2D(32, (3,3), activation='relu'),
        keras.layers.MaxPooling2D(2,2),
        keras.layers.Dropout(0.2),

        keras.layers.Conv2D(64, (3,3), activation='relu'),
        keras.layers.Conv2D(64, (3,3), activation='relu'),
        keras.layers.MaxPooling2D(2,2),
        keras.layers.Dropout(0.2),

        keras.layers.Conv2D(256, (3,3), activation='relu'),
        keras.layers.Conv2D(256, (3,3), activation='relu'),
        keras.layers.MaxPooling2D(2,2),
        keras.layers.Dropout(0.2),

        keras.layers.Flatten(),
        keras.layers.Dense(1024, activation='relu'),
        keras.layers.Dropout(0.5),
        keras.layers.Dense(3, activation='softmax')
    ])


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

    # See an overview of the model architecture and to debug issues related to the model layers.
    model.summary()

###########################################################################################

    import time
    start_time = time.time() #To show the training time

    # Train the model

    # set an early stopping mechanism
    # set patience to be tolerant against random validation loss increases
    early_stopping = tf.keras.callbacks.EarlyStopping(patience=5)
    filepath = "/content/drive/MyDrive/cnnmodel_{epoch:02d}-{val_accuracy:.2f}.h5"

    # Using the ModelCheckpoint function to train and store all the best models
    checkpoint1 = ModelCheckpoint(filepath, monitor='val_accuracy', verbose=1, save_best_only=True, mode='max')

    callbacks_list = [checkpoint1]

    # history = model.fit(data, labels_one_hot, batch_size=32, epochs=10, validation_split=0.2)
    history = model.fit(x=data,
                        y=labels_one_hot,
                        batch_size=batch_size,
                        epochs=epochs,
                        validation_split=0.2,
                        callbacks=callbacks_list)


    # Evaluate the model
    print("Test accuracy: ", max(history.history['val_accuracy']))

    # Assign the trained model
    self_train_model = history

    end_time = time.time() # To show the training time
    training_time = end_time - start_time
    print("Training time:", training_time, "seconds")

    self_train_model_time = training_time

    return self_train_model, self_train_model_time
```

```
# Run CNN model
self_train_model, self_train_model_time = run_custom_model(batch_size = 256,epochs = 1)
```

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d (Conv2D)             (None, 126, 126, 32)      896
```

```
 conv2d_1 (Conv2D)           (None, 124, 124, 32)      9248

 max_pooling2d (MaxPooling2D  (None, 62, 62, 32)       0
 )

 dropout (Dropout)           (None, 62, 62, 32)        0

 conv2d_2 (Conv2D)           (None, 60, 60, 64)        18496

 conv2d_3 (Conv2D)           (None, 58, 58, 64)        36928

 max_pooling2d_1 (MaxPooling  (None, 29, 29, 64)       0
 2D)

 dropout_1 (Dropout)         (None, 29, 29, 64)        0

 conv2d_4 (Conv2D)           (None, 27, 27, 256)       147712

 conv2d_5 (Conv2D)           (None, 25, 25, 256)       590080

 max_pooling2d_2 (MaxPooling  (None, 12, 12, 256)      0
 2D)

 dropout_2 (Dropout)         (None, 12, 12, 256)       0

 flatten (Flatten)           (None, 36864)             0

 dense (Dense)               (None, 1024)              37749760

 dropout_3 (Dropout)         (None, 1024)              0

 dense_1 (Dense)             (None, 3)                 3075

=================================================================
Total params: 38,556,195
Trainable params: 38,556,195
Non-trainable params: 0
_________________________________________________________________
34/34 [==============================] - ETA: 0s - loss: 0.9178 - accuracy: 0.6997
Epoch 1: val_accuracy improved from -inf to 0.76523, saving model to /content/drive/MyDrive/cnnmodel_01-0.77.h5
34/34 [==============================] - 295s 8s/step - loss: 0.9178 - accuracy: 0.6997 - val_loss: 0.5700 - val_accuracy: 0
Test accuracy:  0.7652296423912048
Training time: 326.83559703826904 seconds
```

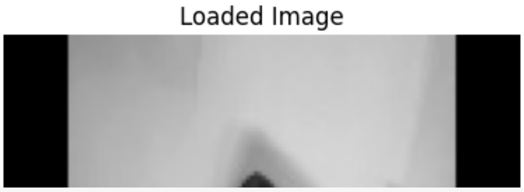
```
# Check our folder and import the model with best validation accuracy
from tensorflow.keras.preprocessing import image
loaded_best_model = keras.models.load_model("/content/drive/MyDrive/cnnmodel_04-0.89.h5")

# Custom function to load and predict label for the image
def predict(img_rel_path):

    img = image.load_img(img_rel_path, target_size=(128, 128))

    # Convert Image to a numpy array
    img = image.img_to_array(img, dtype=np.uint8)

    # Scaling the Image Array values between 0 and 1
    img = np.array(img)/255.0

    # Plotting the Loaded Image
    plt.title("Loaded Image")
    plt.axis('off')
    plt.imshow(img.squeeze())
    plt.show()

    # Get the Predicted Label for the loaded Image
    p = loaded_best_model.predict(img[np.newaxis, ...])

    # Label array
    labels = {0: 'cans', 1: 'glass_bottles',2:'plastic_bottles'}

    print("\n\nMaximum Probability: ", np.max(p[0], axis=-1))
    predicted_class = labels[np.argmax(p[0], axis=-1)]
    print("Classified:", predicted_class, "\n\n")

    classes=[]
    prob=[]
    print("\n-------------------Individual Probability--------------------------------\n")

    for i,j in enumerate (p[0],0):
```

```
        print(labels[i].upper(),':',round(j*100,2),'%')
        classes.append(labels[i])
        prob.append(round(j*100,2))

    def plot_bar_x():
        # this is for plotting purpose
        index = np.arange(len(classes))
        plt.bar(index, prob)
        plt.xlabel('Labels', fontsize=8)
        plt.ylabel('Probability', fontsize=8)
        plt.xticks(index, classes, fontsize=8, rotation=20)
        plt.title('Probability for loaded image')
        plt.show()
    plot_bar_x()
```

```
image = cv2.imread('/content/drive/MyDrive/bottle/glass_bottles/glass_bottles/bdtmp.jpg')
input_size = 128
image_size = (input_size, input_size)
image = cv2.resize(image, image_size)
cv2.imwrite('sample.jpg', image)
```

```
True
```

```
from tensorflow.keras.preprocessing import image
predict("/content/sample.jpg")
```




```
def output_converter(model_output):

    import numpy as np

    output = model_output

    # assume that 'output' is a numpy array of shape (n, 3)
    output_labels = ['gan', 'glass', 'plastci']
    predictions = np.argmax(output, axis=1)
    predicted_labels = [output_labels[p] for p in predictions]

    return predicted_labels
```

```
'''
Plot a Heatmap-Crosstab table out of predicted labels and True labels
'''
def plot_hm_ct(y_true, y_pred):
    import pandas as pd
    import seaborn as sns
    import matplotlib.pyplot as plt

    # create a DataFrame from y_true and y_pred
    df = pd.DataFrame({'y_true': y_true, 'y_pred': y_pred})

    # create cross-tabulation matrix
    ctab = pd.crosstab(df['y_true'], df['y_pred'])

    # create heatmap using seaborn
    sns.heatmap(ctab, annot=True, cmap='Blues', fmt='d')

    # add labels and title
    plt.xlabel('Predicted label')
    plt.ylabel('True label')
    plt.title('Confusion Matrix')

    # show the plot
    plt.show()
```

```
def generate_cf(model, name):

    import pandas as pd
    import seaborn as sns
    import matplotlib.pyplot as plt

    # Assign model to variable 'history'
    history = model

    # Load output data
    y_pred = output_converter(history.model.predict(X_test))
    y_true = y_test

    # Plot the confusion matrix
    # create a DataFrame from y_true and y_pred
    df = pd.DataFrame({'y_true': y_true, 'y_pred': y_pred})

    # create cross-tabulation matrix
    ctab = pd.crosstab(df['y_true'], df['y_pred'])

    # create heatmap using seaborn
    sns.heatmap(ctab, annot=True, cmap='Blues', fmt='d')

    # add labels and title
    plt.xlabel('Predicted label')
    plt.ylabel('True label')
    plt.title('{} Confusion Matrix'.format(name))

    # show the plot
    plt.show()
    from sklearn.metrics import classification_report
```

```
    target_names = ['cans','glass_bottles','plastic_bottles']
    print(classification_report(y_test.classes, y_pred, target_names=target_names))

    # Calculate accuracy score
    from sklearn.metrics import accuracy_score
    accuracy = accuracy_score(y_true, y_pred)
    print("{} accuracy score: {}".format(name, accuracy))
```

```
generate_cf(self_train_model, 'Self Train CNNs')
```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
<ipython-input-3-3edb4c54a3e9> in <cell line: 1>()
----> 1 generate_cf(self_train_model, 'Self Train CNNs')

NameError: name 'self_train_model' is not defined
```

SEARCH STACK OVERFLOW

Hyperparameter tunning-my cnn model

```
from sklearn.model_selection import GridSearchCV
from keras.wrappers.scikit_learn import KerasClassifier
from keras.models import Sequential
from keras.layers import Conv2D, MaxPooling2D, Flatten, Dense, Dropout
import tensorflow as tf
from keras.callbacks import EarlyStopping
from tensorflow.keras.callbacks import ModelCheckpoint


import warnings
warnings.filterwarnings('ignore') # Hide all warnings

import time
start_time = time.time() #To show the training time

tf.random.set_seed(42)
batch_size = [32,64,128 ,256]
epochs = [5,10]
optimizer = ['adam']
# optimizer = ['adam', 'rmsprop']
# cv = 5 # None mean default (K-fold=5)
cv = [(slice(None), slice(None))]


# Design Model Layers
def create_model(optimizer):
    model = Sequential()
    model.add(Conv2D(32, (3,3), activation='relu', input_shape=(input_size, input_size, 3)))
    model.add(Conv2D(32, (3, 3),activation='relu'))
    model.add(MaxPooling2D(pool_size=(2, 2)))
    model.add(Dropout(0.2))

    model.add(Conv2D(64, (3, 3), activation='relu',padding='same'))
    model.add(Conv2D(64, (3, 3), activation='relu'))
    model.add(MaxPooling2D(pool_size=(2, 2)))
    model.add(Dropout(0.2))

    model.add(Conv2D(256, (3, 3), activation='relu',padding='same'))
    model.add(Conv2D(256, (3, 3),activation='relu'))
    model.add(MaxPooling2D(pool_size=(2, 2)))
    model.add(Dropout(0.2))

    model.add(Flatten())
    model.add(Dense(1024, activation='relu'))
    model.add(Dropout(0.5))
    model.add(Dense(3, activation='softmax'))
    model.compile(loss='categorical_crossentropy', optimizer=optimizer, metrics=['accuracy'])
    return model


model = KerasClassifier(build_fn=create_model)
filepath = "/content/drive/MyDrive/Tune_cnnmodel_{epoch:02d}-{val_accuracy:.2f}.h5"

# Using the ModelCheckpoint function to train and store all the best models
```

```
checkpoint1 = ModelCheckpoint(filepath, monitor='val_accuracy', verbose=1, save_best_only=True, mode='max')

callbacks_list = [checkpoint1]

param_grid = {'batch_size': batch_size,
              'epochs': epochs,
              'optimizer': optimizer,
              'callbacks': callbacks_list}


grid = GridSearchCV(estimator=model, param_grid=param_grid, cv=cv)
grid_result = grid.fit(data, labels_one_hot, verbose=0)

print("Best: %f using %s" % (grid_result.best_score_, grid_result.best_params_))
means = grid_result.cv_results_['mean_test_score']
stds = grid_result.cv_results_['std_test_score']
params = grid_result.cv_results_['params']
for mean, stdev, param in zip(means, stds, params):
    print("%f (%f) with: %r" % (mean, stdev, param))

end_time = time.time() # To show the training time
training_time = end_time - start_time
print("Training time:", training_time, "seconds")
grid_time = training_time
```

```
import pandas as pd
print(pd.DataFrame(grid_result.cv_results_))
```

```
output_labels = ['plastic_bottle', 'cans', 'glass_bottle']
result = grid.predict(X_test)

predicted_labels = list(map(lambda x: output_labels[x], result))
```

```
import seaborn as sns

# Load output data
y_pred = predicted_labels
y_true = y_test

# Plot the confusion matrix
# create a DataFrame from y_true and y_pred
df = pd.DataFrame({'y_true': y_true, 'y_pred': y_pred})

# create cross-tabulation matrix
ctab = pd.crosstab(df['y_true'], df['y_pred'])

# create heatmap using seaborn
sns.heatmap(ctab, annot=True, cmap='Blues', fmt='d')

# add labels and title
plt.xlabel('Predicted label')
plt.ylabel('True label')
plt.title('GridSerachCV result Confusion Matrix')

# show the plot
plt.show()
# Calculate accuracy score
from sklearn.metrics import accuracy_score
accuracy = accuracy_score(y_true, y_pred)
print("GridSerachCV accuracy score:{}".format(accuracy))
```

```
import matplotlib.pyplot as plt

# Load the data
X_test = X_test

# choose 20 random indices
indices = np.random.randint(0, len(X_test), 10)

# Get 20 sample images
sample_images = X_test[indices]

# Plot the images
```

```
fig = plt.figure(figsize=(10,10))
for i, img in enumerate(sample_images):
    plt.subplot(4, 5, i+1)
    plt.imshow(img)
    plt.axis('off')
    plt.title( y_true[indices[i]] + "\n" + "Predicted result: " + "\n"+ y_pred[indices[i]])

plt.show()
```

My customised model-Resnet-50

```
from tensorflow.keras.applications.inception_v3 import InceptionV3
```

```
train_dir = '/content/drive/MyDrive/bottle'
os.path.exists(train_dir)
```

```
True
```

```
from keras.callbacks import EarlyStopping
Callback = EarlyStopping(monitor = 'val_loss',
                          min_delta = 0,
                          patience = 5,
                          verbose = 1,
                          restore_best_weights = True)
```

```
# augmentation train only
train_datagen = ImageDataGenerator(rescale = 1./255.,
                                   validation_split=0.15,
                                   rotation_range = 40,
                                   width_shift_range = 0.2,
                                   height_shift_range = 0.2,
                                   shear_range = 0.2,
                                   zoom_range = 0.2,
                                   horizontal_flip = True,
                                   fill_mode = 'nearest'
                                  )

validation_datagen = ImageDataGenerator(rescale = 1./255., validation_split=0.15)
```

```
HYP = dict(
        seed = 77,
        img_size = (225,225)
        )
```

```
# flow_from_directory
train_generator = train_datagen.flow_from_directory(train_dir,
                                                    target_size=HYP['img_size'],
                                                    shuffle=True,
                                                    seed=HYP['seed'],
                                                    class_mode='categorical',
                                                    subset="training")

validation_generator = validation_datagen.flow_from_directory(train_dir,
                                                    target_size=HYP['img_size'],
                                                    shuffle=False,
                                                    seed=HYP['seed'],
                                                    class_mode='categorical',
                                                    subset="validation")
```

```
Found 7076 images belonging to 3 classes.
Found 1247 images belonging to 3 classes.
```

```
# load the pre-trained ResNet50 model
base_model = InceptionV3(
                        include_top = False,
                        weights = "imagenet",
                        input_shape = None
)
```

```
Downloading data from https://storage.googleapis.com/tensorflow/keras-applications/inception_v3/inception_v3_weights_tf_dim_c
87910968/87910968 [==============================] - 0s 0us/step
```

```
# New Construction of Fully Connected Layer
from keras import regularizers
x = base_model.output
x = GlobalAveragePooling2D()(x)
x = Dense(512, activation='relu',kernel_regularizer= regularizers.l1(0.001))(x)
predictions = Dense(3, activation='softmax')(x)
```

```
from keras import regularizers

# network definition
model = Model(inputs = base_model.input, outputs = predictions)

# Train layer 250 and above
for layer in model.layers[:249]:
    layer.trainable = False

    # Batch Normalization improves the generalization performance of the model by updating parameters during training.
    if layer.name.startswith('batch_normalization'):
        layer.trainable = True

for layer in model.layers[249:]:
    layer.trainable = True

# After setting layer.trainable, be sure to compile.
model.compile(
    optimizer = Adam(),
    loss = 'categorical_crossentropy',
    metrics = ["accuracy"]
)
filepath = "/content/drive/MyDrive/newmodel_{epoch:02d}-{val_accuracy:.2f}.h5"

# Using the ModelCheckpoint function to train and store all the best models
checkpoint1 = ModelCheckpoint(filepath, monitor='val_accuracy', verbose=1, save_best_only=True, mode='max')

callbacks_list = [checkpoint1]
```

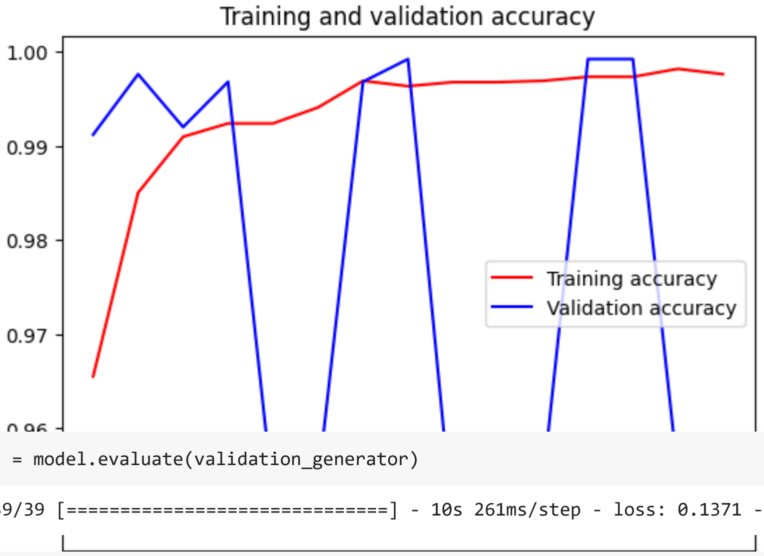
```
model.summary()
from keras.utils.vis_utils import plot_model
plot_model(model, show_shapes=True, show_layer_names=True)
```

```
fit_history = model.fit(
                        train_generator,
                        validation_data=validation_generator,
                        callbacks=callbacks_list,
                        epochs=15,
                        verbose=1
                        )
```

```
Epoch 1: val_accuracy improved from -inf to 0.99118, saving model to /content/drive/MyDrive/newmodel_01-0.99.h5
222/222 [==============================] - 2259s 10s/step - loss: 3.6520 - accuracy: 0.9655 - val_loss: 0.2615 - val_accur
Epoch 2/15
222/222 [==============================] - ETA: 0s - loss: 0.2403 - accuracy: 0.9850
Epoch 2: val_accuracy improved from 0.99118 to 0.99759, saving model to /content/drive/MyDrive/newmodel_02-1.00.h5
222/222 [==============================] - 145s 652ms/step - loss: 0.2403 - accuracy: 0.9850 - val_loss: 0.2234 - val_accu
Epoch 3/15
222/222 [==============================] - ETA: 0s - loss: 0.2084 - accuracy: 0.9910
Epoch 3: val_accuracy did not improve from 0.99759
222/222 [==============================] - 138s 622ms/step - loss: 0.2084 - accuracy: 0.9910 - val_loss: 0.2116 - val_accu
Epoch 4/15
222/222 [==============================] - ETA: 0s - loss: 0.1887 - accuracy: 0.9924
Epoch 4: val_accuracy did not improve from 0.99759
222/222 [==============================] - 138s 620ms/step - loss: 0.1887 - accuracy: 0.9924 - val_loss: 0.1961 - val_accu
Epoch 5/15
222/222 [==============================] - ETA: 0s - loss: 0.2007 - accuracy: 0.9924
Epoch 5: val_accuracy did not improve from 0.99759
222/222 [==============================] - 139s 625ms/step - loss: 0.2007 - accuracy: 0.9924 - val_loss: 0.2590 - val_accu
Epoch 6/15
222/222 [==============================] - ETA: 0s - loss: 0.1727 - accuracy: 0.9941
Epoch 6: val_accuracy did not improve from 0.99759
222/222 [==============================] - 139s 625ms/step - loss: 0.1727 - accuracy: 0.9941 - val_loss: 0.2273 - val_accu
Epoch 7/15
222/222 [==============================] - ETA: 0s - loss: 0.1613 - accuracy: 0.9969
Epoch 7: val_accuracy did not improve from 0.99759
222/222 [==============================] - 140s 629ms/step - loss: 0.1613 - accuracy: 0.9969 - val_loss: 0.1757 - val_accu
```

```
222/222 [==============================] - 142s 636ms/step - loss: 0.1588 - accuracy: 0.9963 - val_loss: 0.1707 - val_accu
Epoch 9/15
222/222 [==============================] - ETA: 0s - loss: 0.1584 - accuracy: 0.9967
Epoch 9: val_accuracy did not improve from 0.99920
222/222 [==============================] - 138s 623ms/step - loss: 0.1584 - accuracy: 0.9967 - val_loss: 0.2504 - val_accu
Epoch 10/15
222/222 [==============================] - ETA: 0s - loss: 0.1567 - accuracy: 0.9967
Epoch 10: val_accuracy did not improve from 0.99920
222/222 [==============================] - 138s 621ms/step - loss: 0.1567 - accuracy: 0.9967 - val_loss: 0.1930 - val_accu
Epoch 11/15
222/222 [==============================] - ETA: 0s - loss: 0.1503 - accuracy: 0.9969
Epoch 11: val_accuracy did not improve from 0.99920
222/222 [==============================] - 138s 622ms/step - loss: 0.1503 - accuracy: 0.9969 - val_loss: 0.1875 - val_accu
Epoch 12/15
222/222 [==============================] - ETA: 0s - loss: 0.1504 - accuracy: 0.9973
Epoch 12: val_accuracy did not improve from 0.99920
222/222 [==============================] - 138s 623ms/step - loss: 0.1504 - accuracy: 0.9973 - val_loss: 0.1663 - val_accu
Epoch 13/15
222/222 [==============================] - ETA: 0s - loss: 0.1437 - accuracy: 0.9973
Epoch 13: val_accuracy did not improve from 0.99920
222/222 [==============================] - 138s 622ms/step - loss: 0.1437 - accuracy: 0.9973 - val_loss: 0.1671 - val_accu
Epoch 14/15
222/222 [==============================] - ETA: 0s - loss: 0.1406 - accuracy: 0.9982
Epoch 14: val_accuracy did not improve from 0.99920
222/222 [==============================] - 138s 620ms/step - loss: 0.1406 - accuracy: 0.9982 - val_loss: 0.2029 - val_accu
Epoch 15/15
222/222 [==============================] - ETA: 0s - loss: 0.1428 - accuracy: 0.9976
Epoch 15: val_accuracy did not improve from 0.99920
```

```
acc = fit_history.history['accuracy']
val_acc = fit_history.history['val_accuracy']
loss = fit_history.history['loss']
val_loss = fit_history.history['val_loss']

epochs = range(len(acc))

plt.plot(epochs, acc, 'r', label='Training accuracy')
plt.plot(epochs, val_acc, 'b', label='Validation accuracy')
plt.title('Training and validation accuracy')
plt.legend()
plt.figure()

plt.plot(epochs, loss, 'r', label='Training Loss')
plt.plot(epochs, val_loss, 'b', label='Validation Loss')
plt.title('Training and validation loss')

plt.legend()

plt.show()
```




```
scores = model.evaluate(validation_generator)
```

```
39/39 [==============================] - 10s 261ms/step - loss: 0.1371 - accuracy: 0.9976
```

```
# Check our folder and import the model with best validation accuracy
loaded_best_model = keras.models.load_model("/content/drive/MyDrive/newmodel_08-1.00.h5")

# Custom function to load and predict label for the image
def predict(img_rel_path):
    # Import Image from the path with size of (300, 300)
    img = image.load_img(img_rel_path, target_size=(300, 300))

    # Convert Image to a numpy array
    img = image.img_to_array(img, dtype=np.uint8)

    # Scaling the Image Array values between 0 and 1
    img = np.array(img)/255.0

    # Plotting the Loaded Image
    plt.title("Loaded Image")
    plt.axis('off')
    plt.imshow(img.squeeze())
    plt.show()

    # Get the Predicted Label for the loaded Image
    p = loaded_best_model.predict(img[np.newaxis, ...])

    # Label array
    labels = {0: 'cans', 1: 'glass_bottles',2:'plastic_bottles'}

    print("\n\nMaximum Probability: ", np.max(p[0], axis=-1))
    predicted_class = labels[np.argmax(p[0], axis=-1)]
    print("Classified:", predicted_class, "\n\n")

    classes=[]
    prob=[]
    print("\n-------------------Individual Probability--------------------------------\n")

    for i,j in enumerate (p[0],0):
        print(labels[i].upper(),':',round(j*100,2),'%')
        classes.append(labels[i])
        prob.append(round(j*100,2))

    def plot_bar_x():
        # this is for plotting purpose
        index = np.arange(len(classes))
        plt.bar(index, prob)
        plt.xlabel('Labels', fontsize=8)
        plt.ylabel('Probability', fontsize=8)
        plt.xticks(index, classes, fontsize=8, rotation=20)
        plt.title('Probability for loaded image')
        plt.show()
    plot_bar_x()
```

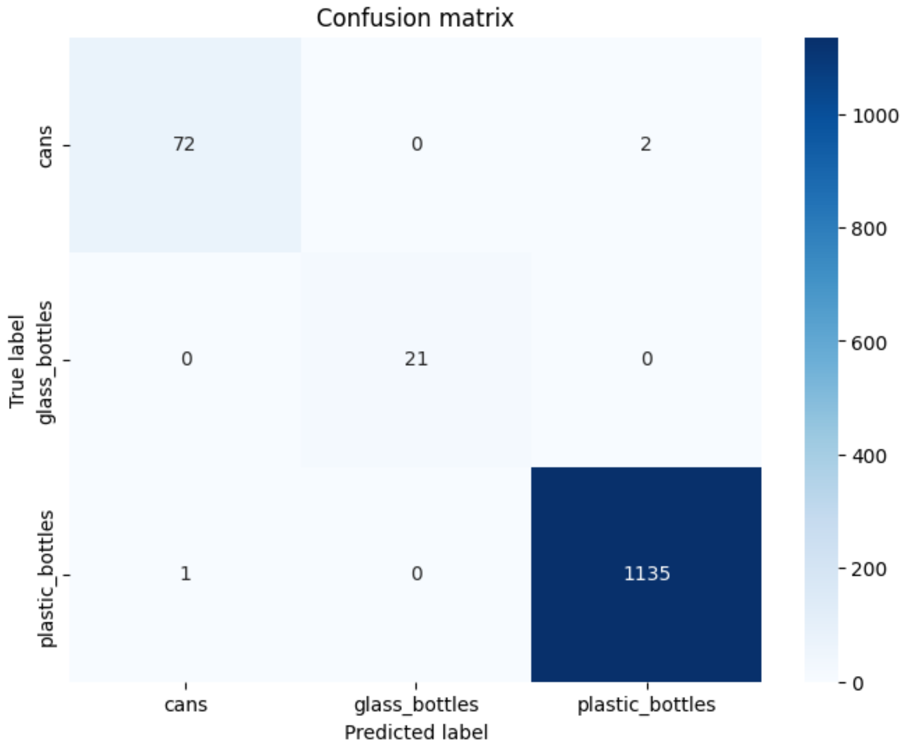
```
predict("/content/drive/MyDrive/bottle/cans/cans/agfie.jpg")
```

```
predict("/content/drive/MyDrive/bottle/glass_bottles/glass_bottles/ahnxy.jpg")
```

```
predict("/content/drive/MyDrive/bottle/plastic_bottles/plastic_bottles/abwiq.jpg")
```

```
classes = train_generator.class_indices.keys()

from sklearn.metrics import confusion_matrix

y_pred = np.argmax(model.predict(validation_generator), axis=1)
cm = confusion_matrix(validation_generator.classes, y_pred)

# Heatmap
plt.figure(figsize=(8,6))
sns.heatmap(cm, annot=True, fmt='d', cbar=True, cmap='Blues',xticklabels=classes, yticklabels=classes)
plt.xlabel('Predicted label')
plt.ylabel('True label')
plt.title('Confusion matrix')
plt.show()
```

```
39/39 [==============================] - 12s 260ms/step
```




```
from sklearn.metrics import classification_report
target_names = ['cans','glass_bottles','plastic_bottles']
print(classification_report(validation_generator.classes, y_pred, target_names=target_names))
```

```
                 precision    recall  f1-score   support

           cans       0.99      0.97      0.98        74
  glass_bottles       1.00      1.00      1.00        21
plastic_bottles       1.00      1.00      1.00      1136

       accuracy                           1.00      1231
      macro avg       0.99      0.99      0.99      1231
   weighted avg       1.00      1.00      1.00      1231
```

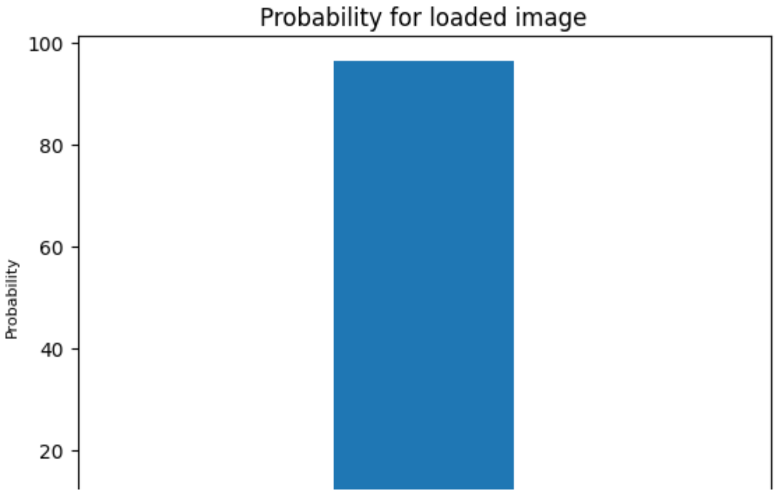
Evaluation

```
import numpy as np
import matplotlib.pyplot as plt

# set width of bar
barWidth = 0.17
fig = plt.subplots(figsize =(15, 10))

# set height of bar
a= [93.4,93,92.5,92.5]
b= [94,92.4,94,93]
```

```
c= [99,99,98,98]


br1 = np.arange(len(a))
br2 = [x + barWidth for x in br1]
br3 = [x + barWidth for x in br2]


# Make the plot
plt.bar(br1,a, color ='lightgreen', width = barWidth,
        edgecolor ='grey', label ='CNN')
plt.bar(br1 + barWidth, b, color ='gold', width = barWidth,
        edgecolor ='grey', label ='Hyb-CNN')
plt.bar(br3, c, color ='aqua', width = barWidth,
        edgecolor ='grey', label ='Resnet-50')


#plt.xlim(0,20)
plt.ylim(0,100)

# Adding Xticks
#plt.xlabel('Branch', fontweight ='bold', fontsize = 15)
plt.ylabel('Metric(%)',  fontsize = 25,fontweight="bold")
plt.xticks([r + barWidth for r in range(len(a))],
        ['Accuracy','Precision','Recall','f1-score'],fontsize = 20,fontweight="bold")
plt.title('Evaluation',fontsize = 25,fontweight="bold")
plt.yticks(fontsize=20,fontweight='bold')

plt.legend(loc='upper left', bbox_to_anchor = (1.05, 0.6),
          ncol=1, fancybox=True, shadow=True,fontsize=20)

plt.show()
```

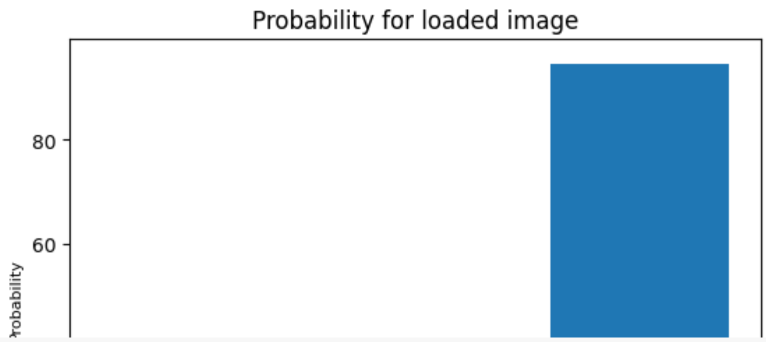
```
predict("/content/sample_data/images.jfif")
```


```
1/1 [==============================] - 1s 1s/step


Maximum Probability:  0.9639429
Classified: glass_bottles



-------------------Individual Probability--------------------------------

CANS : 0.49 %
GLASS_BOTTLES : 96.39 %
PLASTIC_BOTTLES : 3.12 %
```




```
predict("/content/sample_data/images (3).jfif")
```


```
1/1 [==============================] - 0s 27ms/step


Maximum Probability:  0.9460981
Classified: plastic_bottles



-------------------Individual Probability--------------------------------

CANS : 0.25 %
GLASS_BOTTLES : 5.14 %
PLASTIC_BOTTLES : 94.61 %
```




```
predict("/content/sample_data/images (2).jfif")
```

Loaded Image

```
1/1 [==============================] - 0s 40ms/step


Maximum Probability:  0.9136326
Classified: plastic_bottles



-------------------Individual Probability--------------------------------
```

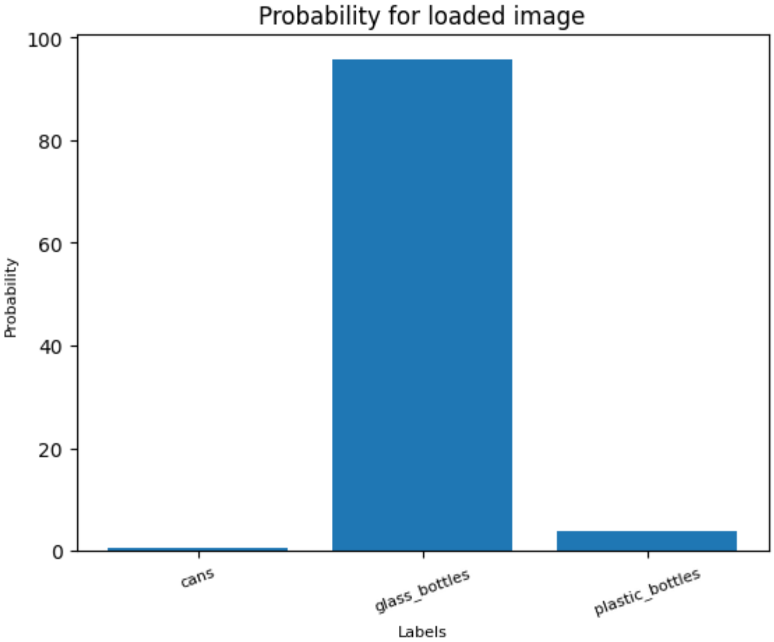
```
predict("/content/sample_data/images (1).jfif")
```

Loaded Image

```
predict("/content/sample_data/download.jfif")
```

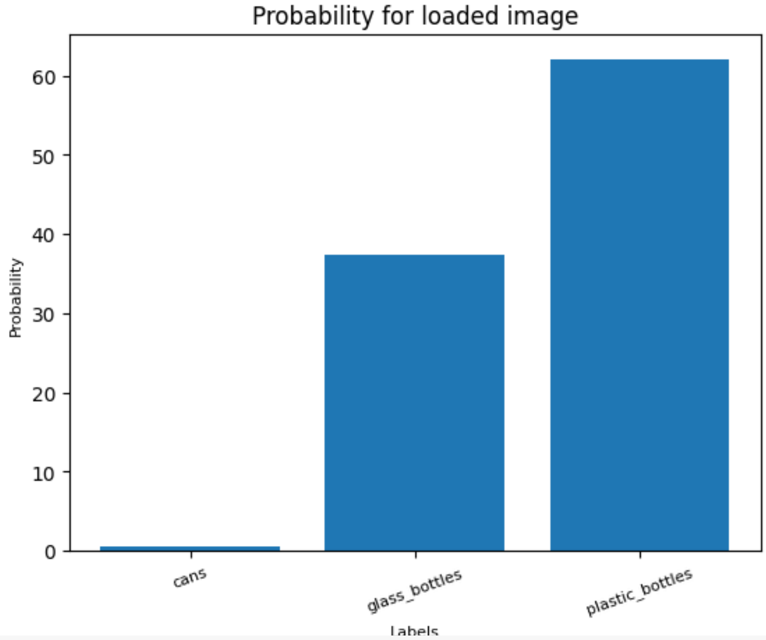
Loaded Image

```
1/1 [==============================] - 0s 27ms/step


Maximum Probability:  0.95745456
Classified: glass_bottles



-------------------Individual Probability--------------------------------

CANS : 0.42 %
GLASS_BOTTLES : 95.75 %
PLASTIC_BOTTLES : 3.84 %
```




```
predict("/content/sample_data/download (1).jfif")
```

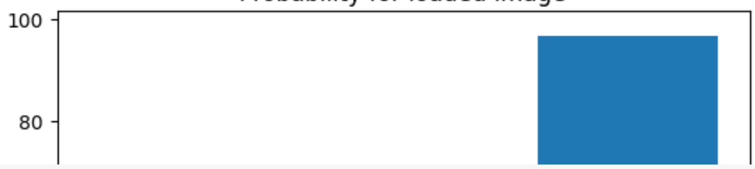

```
1/1 [==============================] - 0s 27ms/step


Maximum Probability:  0.62041795
Classified: plastic_bottles



-------------------Individual Probability--------------------------------

CANS : 0.58 %
GLASS_BOTTLES : 37.38 %
PLASTIC_BOTTLES : 62.04 %
```




```
predict("/content/sample_data/crushed-pet-bottle-scrap-1571123911-5116489.jpeg")
```

Loaded Image

```
1/1 [==============================] - 0s 37ms/step


Maximum Probability:  0.96610516
Classified: plastic_bottles



-------------------Individual Probability--------------------------------

CANS : 1.42 %
GLASS_BOTTLES : 1.96 %
PLASTIC_BOTTLES : 96.61 %
```




```
predict("/content/sample_data/78156221.jpg")
```