```
import os
import sys
module_path = os.path.abspath(os.path.join(os.pardir, os.pardir))
if module_path not in sys.path:
    sys.path.append(module_path)

import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import matplotlib.image as mpimg
import seaborn as sns

from sklearn.metrics import confusion_matrix, ConfusionMatrixDisplay
from sklearn import preprocessing
import tensorflow as tf
from tensorflow import keras
from tensorflow.keras.layers import MaxPooling2D
from keras.applications.vgg16 import VGG16
from keras.preprocessing.image import ImageDataGenerator #
from keras.layers import Conv2D, MaxPool2D, Dense, Flatten, Dropout, BatchNormalization, Activation, Input
from keras.models import Sequential
from keras.callbacks import EarlyStopping
from keras import regularizers
from keras.wrappers.scikit_learn import KerasClassifier

import warnings
warnings.filterwarnings("ignore")

from keras import layers
from keras import models
from keras import optimizers

from sklearn.utils.class_weight import compute_class_weight
import joblib
from sklearn.metrics import accuracy_score
```

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

```
Mounted at /content/drive
```

```
%cd /content/drive/MyDrive/Garbage.1
```

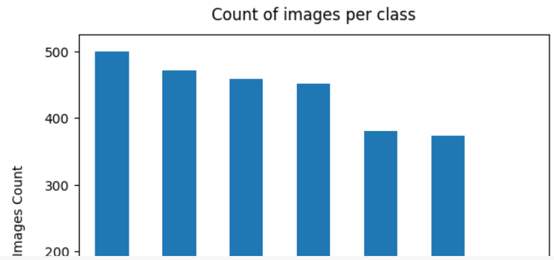
```
/content/drive/MyDrive/Garbage.1
```

```
# Define directories for train, test and validation data
train_dir = "/content/drive/MyDrive/Garbage.1/train"
test_dir = "/content/drive/MyDrive/Garbage.1/test"
validation_dir = "/content/drive/MyDrive/Garbage.1/validation"

# Define batch size and image size
batch_size = 64
image_size = 128
```

```
# list to store the corresponding category, note that each folder of the dataset has one class of data
categories_list = []

# list containing all the filenames in the dataset
filenames_list = []

# Iterate through categories and filenames, adding them to the respective lists
for category in os.listdir(train_dir):
    for filename in os.listdir(os.path.join(train_dir, category)):
        filenames_list.append(os.path.join(category, filename))
        categories_list.append(category)

# Create a DataFrame with filenames and categories
df = pd.DataFrame({
    'filename': filenames_list,
    'category': categories_list
})

# Shuffle the dataframe
df = df.sample(frac=1).reset_index(drop=True)

print('number of elements = ' , len(df))
```

```
number of elements =  2741
```

```
# Createting a barplot to visualize the distribution of images
df_visualization = df.copy()

df_visualization['category'].value_counts().plot.bar(x = 'count', y = 'category' )

plt.xlabel("Garbage Classes", labelpad=14)
plt.ylabel("Images Count", labelpad=14)
plt.title("Count of images per class", y=1.02);

# We can see here that this data is imbalanced
```




```
# Initialize the ImageDataGenerators for the training and validation sets
# Upsampling train data
train_datagen = ImageDataGenerator(rescale=1./255,
                                   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')

val_datagen = ImageDataGenerator(rescale=1./255)

train_generator = train_datagen.flow_from_directory(
        # This is the target directory
        train_dir,
        shuffle=True,
        # All images will be resized to image_size set earlier
        target_size=(image_size, image_size),
        batch_size=batch_size,
        # Since we use categorical_crossentropy loss, we need categorical labels
        class_mode='categorical')

validation_generator = val_datagen.flow_from_directory(validation_dir,
                                                       shuffle=False,
                                                       target_size=(image_size, image_size),
                                                       batch_size=batch_size,
                                                       class_mode='categorical')
```

```
Found 2741 images belonging to 7 classes.
Found 105 images belonging to 7 classes.
```

```
# Adding class weights to balance the data
# Get class labels and indices from the generators
class_labels = list(train_generator.class_indices.keys())
class_indices = np.array(list(train_generator.class_indices.values()))

# Calculate class weights using scikit-learn's compute_class_weight function
class_weights = compute_class_weight(class_weight='balanced', classes=np.unique(train_generator.classes), y=train_generator.classes)

class_weights = dict(enumerate(class_weights))

# Convert class_weights to a dictionary to pass it to the model.fit() method
class_weights = {i: class_weights[i] for i in range(len(class_labels))}
```

```
# Looking at the weight distribution
print(class_labels)
print(class_indices)
class_weights
```

```
['cardboard', 'carton', 'glass', 'metal', 'paper', 'plastic', 'trash']
[0 1 2 3 4 5 6]
{0: 1.0497893527384143,
 1: 0.85495945102932,
 2: 0.8313618441006976,
 3: 1.030451127819549,
 4: 0.7831428571428571,
 5: 0.8663084702907712,
 6: 3.65954606141522}
```

```
# Baseline model
# Defining model architecture
model = models.Sequential()

model.add(Conv2D(32, kernel_size=(3, 3), activation='relu'))
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Dropout(0.25))  # Add dropout with rate of 0.25

model.add(Conv2D(64, kernel_size=(3, 3), activation='relu'))
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Dropout(0.25))  # Add dropout with rate of 0.25

model.add(Conv2D(128, kernel_size=(3, 3), activation='relu'))
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Dropout(0.25))  # Add dropout with rate of 0.25

model.add(Flatten())
model.add(Dense(128, activation='relu'))
model.add(Dropout(0.5))   # Add dropout with rate of 0.5
model.add(Dense(7, activation='softmax'))

# This was the combination of layers and regularizations that gave me the best results
```

```
# Compile the model
model.compile(loss='categorical_crossentropy',
              optimizer=optimizers.RMSprop(lr=1e-4),
              metrics=['acc'])
```

```
# Print model summary
model.summary()
```

```
# Define early stopping callback
early_stop = EarlyStopping(monitor='val_loss', patience=5)

# Train the model on the training set and validate on the validation set
history = model.fit(train_generator,
                    steps_per_epoch=len(train_generator),
                    epochs=30,
                    validation_data=validation_generator,
                    validation_steps=len(validation_generator),
                    verbose=2,
                    callbacks=[early_stop],
                    class_weight=class_weights)
```

```
# Define test data generator and get test set
test_datagen = ImageDataGenerator(rescale=1./255)

test_generator = test_datagen.flow_from_directory(test_dir,
                                                  shuffle=False,
                                                  target_size=(image_size, image_size),
                                                  batch_size=batch_size,
                                                  class_mode='categorical')

# Evaluating the model on the test set
test_x, test_y = next(test_generator)
results_test = model.evaluate(test_x, test_y)

# After trying all diferent kinds of combinations without making any relevant improvement, I decided to use transfer learning to see if I could get better results
```

```
# Plot the training and validation accuracy and loss
plt.figure(figsize=(20,5))
plt.subplot(1,2,1)
plt.plot(history.history['acc'], label='acc')
plt.plot(history.history['val_acc'], label='val_acc')
plt.legend()

plt.subplot(1,2,2)
plt.plot(history.history['loss'], label='loss')
plt.plot(history.history['val_loss'], label='val_loss')
plt.legend()
```

```
# Get the predicted probabilities for the validation set
pred_probs = model.predict(validation_generator)

# Convert the probabilities to predicted class labels
pred_labels = np.argmax(pred_probs, axis=1)

# Get the true class labels for the validation set
true_labels = validation_generator.classes
```

```
# Calculate confusion matrix
cm = confusion_matrix(true_labels, pred_labels)

# Plot confusion matrix
disp = ConfusionMatrixDisplay(confusion_matrix=cm, display_labels=test_generator.class_indices.keys())
fig, ax = plt.subplots(figsize=(10,10))
disp.plot(cmap=plt.cm.Blues, ax=ax)
plt.show()
```

```
# Get class names from test generator
class_names = list(test_generator.class_indices.keys())

# Get classification report
report = classification_report(true_labels, pred_labels, target_names=class_names)

# Print report
print(report)
```

```
# Using transfer learning to see if I can get better results with a pre-trained model
# Load VGG16 model without top layer and freeze its layers
base_model = VGG16(weights='imagenet', include_top=False, input_shape=(image_size, image_size) + (3,))

# Freeze the layers of VGG16 model
for layer in base_model.layers:
    layer.trainable = False

# Create new model and add VGG16 base model
model = Sequential()
model.add(base_model)
model.add(Flatten())

# Add new fully connected layers
model.add(Dense(7, activation='softmax'))

# Compile the model
model.compile(loss='categorical_crossentropy',
              optimizer=tf.keras.optimizers.Adam(),
              metrics=['acc'])

# Print model summary
model.summary()

early_stop = EarlyStopping(monitor='val_loss', patience=5)

# Train the model
history = model.fit(train_generator,
                    epochs=50,
                    steps_per_epoch=train_generator.n//train_generator.batch_size,
                    validation_data=validation_generator,
                    validation_steps=validation_generator.n//validation_generator.batch_size,
                    verbose=2,
                    validation_freq=1,
                    callbacks=[early_stop],
                    class_weight=class_weights)
```

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 vgg16 (Functional)          (None, 4, 4, 512)         14714688

 flatten (Flatten)           (None, 8192)              0

 dense (Dense)               (None, 7)                 57351

=================================================================
Total params: 14,772,039
Trainable params: 57,351
Non-trainable params: 14,714,688
_________________________________________________________________
Epoch 1/50
42/42 - 624s - loss: 1.3615 - acc: 0.4905 - val_loss: 0.8757 - val_acc: 0.6719 - 624s/epoch - 15s/step
Epoch 2/50
42/42 - 592s - loss: 0.9911 - acc: 0.6403 - val_loss: 0.8924 - val_acc: 0.6562 - 592s/epoch - 14s/step
Epoch 3/50
42/42 - 582s - loss: 0.9012 - acc: 0.6735 - val_loss: 0.6498 - val_acc: 0.7500 - 582s/epoch - 14s/step
Epoch 4/50
42/42 - 590s - loss: 0.8304 - acc: 0.7004 - val_loss: 0.6670 - val_acc: 0.7969 - 590s/epoch - 14s/step
Epoch 5/50
42/42 - 595s - loss: 0.7757 - acc: 0.7310 - val_loss: 0.6705 - val_acc: 0.7812 - 595s/epoch - 14s/step
Epoch 6/50
42/42 - 591s - loss: 0.7362 - acc: 0.7434 - val_loss: 0.5502 - val_acc: 0.7969 - 591s/epoch - 14s/step
Epoch 7/50
42/42 - 587s - loss: 0.7111 - acc: 0.7370 - val_loss: 0.5343 - val_acc: 0.8438 - 587s/epoch - 14s/step
Epoch 8/50
42/42 - 594s - loss: 0.6899 - acc: 0.7579 - val_loss: 0.7060 - val_acc: 0.7031 - 594s/epoch - 14s/step
Epoch 9/50
42/42 - 593s - loss: 0.6725 - acc: 0.7665 - val_loss: 0.6338 - val_acc: 0.7969 - 593s/epoch - 14s/step
Epoch 10/50
42/42 - 593s - loss: 0.6509 - acc: 0.7643 - val_loss: 0.5660 - val_acc: 0.7812 - 593s/epoch - 14s/step
Epoch 11/50
42/42 - 589s - loss: 0.6141 - acc: 0.7848 - val_loss: 0.5336 - val_acc: 0.7656 - 589s/epoch - 14s/step
Epoch 12/50
42/42 - 588s - loss: 0.6144 - acc: 0.7822 - val_loss: 0.5034 - val_acc: 0.7812 - 588s/epoch - 14s/step
Epoch 13/50
42/42 - 599s - loss: 0.5761 - acc: 0.7919 - val_loss: 0.5286 - val_acc: 0.7969 - 599s/epoch - 14s/step
Epoch 14/50
42/42 - 591s - loss: 0.5568 - acc: 0.8005 - val_loss: 0.5240 - val_acc: 0.7656 - 591s/epoch - 14s/step
Epoch 15/50
42/42 - 592s - loss: 0.5688 - acc: 0.7972 - val_loss: 0.4512 - val_acc: 0.8281 - 592s/epoch - 14s/step
Epoch 16/50
42/42 - 592s - loss: 0.5513 - acc: 0.8009 - val_loss: 0.5435 - val_acc: 0.7812 - 592s/epoch - 14s/step
Epoch 17/50
42/42 - 603s - loss: 0.5613 - acc: 0.7998 - val_loss: 0.3327 - val_acc: 0.8594 - 603s/epoch - 14s/step
Epoch 18/50
42/42 - 634s - loss: 0.5401 - acc: 0.8020 - val_loss: 0.3727 - val_acc: 0.8438 - 634s/epoch - 15s/step
Epoch 19/50
42/42 - 618s - loss: 0.5229 - acc: 0.8173 - val_loss: 0.4695 - val_acc: 0.8281 - 618s/epoch - 15s/step
Epoch 20/50
42/42 - 595s - loss: 0.5213 - acc: 0.8099 - val_loss: 0.4290 - val_acc: 0.8281 - 595s/epoch - 14s/step
Epoch 21/50
42/42 - 585s - loss: 0.5144 - acc: 0.8270 - val_loss: 0.3948 - val_acc: 0.8750 - 585s/epoch - 14s/step
Epoch 22/50
42/42 - 585s - loss: 0.5039 - acc: 0.8166 - val_loss: 0.4290 - val_acc: 0.8438 - 585s/epoch - 14s/step
```

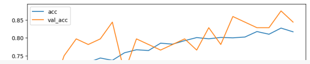
```python
# Saving model
model.save("recycling_cnn_2.h5")
```

```python
# Define test data generator and get test set
test_datagen = ImageDataGenerator(rescale=1./255)

test_generator = test_datagen.flow_from_directory(test_dir,
                                                  shuffle=False,
                                                  target_size=(image_size, image_size),
                                                  batch_size=batch_size,
                                                  class_mode='categorical')

# Evaluating the model on the test set
test_x, test_y = next(test_generator)
results_test = model.evaluate(test_x, test_y)

# Much better results using VGG16
```

```
Found 105 images belonging to 7 classes.
2/2 [==============================] - 13s 8s/step - loss: 1.0314 - acc: 0.7188
```

```python
# Plot the training and validation accuracy and loss
plt.figure(figsize=(20,5))
plt.subplot(1,2,1)
plt.plot(history.history['acc'], label='acc')
plt.plot(history.history['val_acc'], label='val_acc')
plt.legend()

plt.subplot(1,2,2)
plt.plot(history.history['loss'], label='loss')
plt.plot(history.history['val_loss'], label='val_loss')
plt.legend()
```

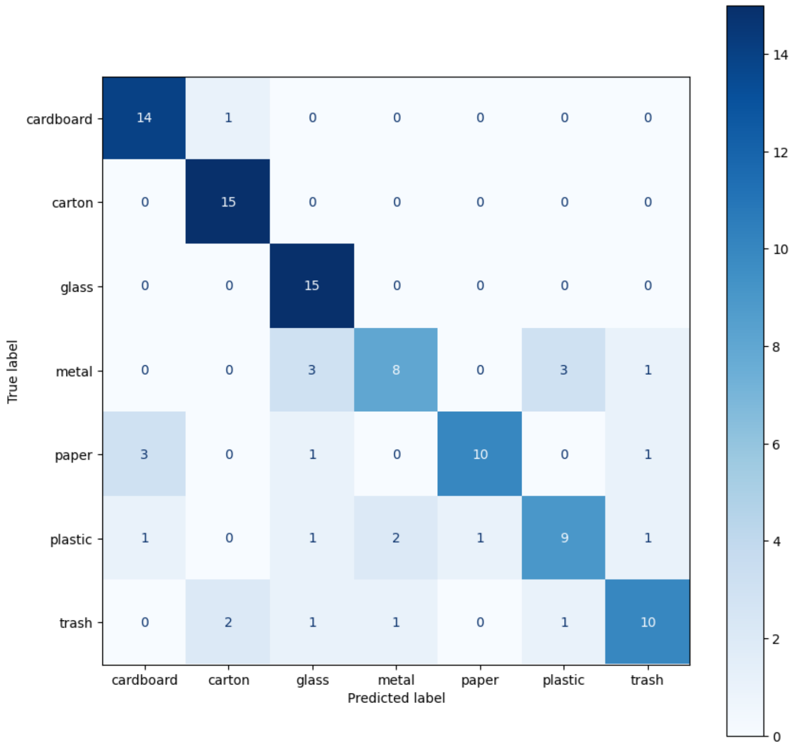
```
<matplotlib.legend.Legend at 0x7f1e8d272070>
```



```
# Get the predicted probabilities for the validation set
pred_probs = model.predict(validation_generator)

# Convert the probabilities to predicted class labels
pred_labels = np.argmax(pred_probs, axis=1)

# Get the true class labels for the validation set
true_labels = validation_generator.classes
```

```
2/2 [==============================] - 26s 9s/step
```

```
# Calculate confusion matrix
cm = confusion_matrix(true_labels, pred_labels)

# Plot confusion matrix
disp = ConfusionMatrixDisplay(confusion_matrix=cm, display_labels=test_generator.class_indices.keys())
fig, ax = plt.subplots(figsize=(10,10))
disp.plot(cmap=plt.cm.Blues, ax=ax)
plt.show()
```




```
# Get class names from test generator
class_names = list(test_generator.class_indices.keys())

# Get classification report
report = classification_report(true_labels, pred_labels, target_names=class_names)

# Print report
print(report)
```

```
              precision    recall  f1-score   support

   cardboard       0.78      0.93      0.85        15
      carton       0.83      1.00      0.91        15
       glass       0.71      1.00      0.83        15
       metal       0.73      0.53      0.62        15
       paper       0.91      0.67      0.77        15
     plastic       0.69      0.60      0.64        15
       trash       0.77      0.67      0.71        15

    accuracy                           0.77       105
   macro avg       0.77      0.77      0.76       105
weighted avg       0.77      0.77      0.76       105
```

✓ 0s completed at 1:47 PM