In [1]:

```
# Import libraries

from keras.models import Sequential
from keras.layers import Conv2D, Activation,Dropout
from keras.models import Model,load_model
from tensorflow.keras.layers import BatchNormalization
from keras.layers.pooling import MaxPooling2D
from keras.layers.core import Flatten, Dense
from keras.callbacks import ModelCheckpoint, EarlyStopping, ReduceLROnPlateau
import tensorflow.python.keras.engine
from keras.preprocessing.image import ImageDataGenerator
from keras import backend as K
from sklearn.model_selection import train_test_split
from sklearn.metrics import confusion_matrix
from sklearn.datasets import load_files
import itertools
import numpy as np
import pandas as pd
import tensorflow as tf
import cv2
import matplotlib.pyplot as plt
import itertools
%matplotlib inline
```

In [2]:

```
train_dir = '/kaggle/input/waste-classification-data/dataset/DATASET/TRAIN'
test_dir = '/kaggle/input/waste-classification-data/dataset/DATASET/TEST'

def load_dataset(path):
    data = load_files(path) #load all files from the path
    files = np.array(data['filenames']) #get the file
    targets = np.array(data['target'])#get the the classification labels as integer index
    target_labels = np.array(data['target_names'])#get the the classification labels
    return files,targets,target_labels

x_train, y_train,target_labels = load_dataset(train_dir)
x_test, y_test,_ = load_dataset(test_dir)

print('Training set size : ' , x_train.shape[0])
print('Testing set size : ', x_test.shape[0])
```

In [3]:

```
x_train,x_validate,y_train,y_validate = train_test_split(x_train,y_train,test_size = 0.2
,random_state = 1)
```

In [4]:

```
print ("x_train shape: " + str(x_train.shape))
print ("x_train shape: " + str(y_train.shape))
print ("x_validate shape: " + str(x_validate.shape))
print ("y_validate shape: " + str(y_validate.shape))
print ("x_test shape: " + str(x_test.shape))
print ("y_test shape: " + str(y_test.shape))
```

In [5]:

```
# Convert jpg file to numpy array to feed to the CNN.
#By using Opencv .

def convert_image_to_array(files):
    width, height, channels = 100, 100, 3
    images_as_array = np.empty((files.shape[0], width, height, channels), dtype=np.uint8
) #define train and test data shape
    for idx,file in enumerate(files):
        img = cv2.imread(file)
        res = cv2.resize(img, dsize=(width, height), interpolation=cv2.INTER_CUBIC)  #As
images have different size, resizing all images to have same shape of image array
        images_as_array[idx] = res
    return images_as_array

x_train = np.array(convert_image_to_array(x_train))
print('Training set shape : ',x_train.shape)

x_valid = np.array(convert_image_to_array(x_validate))
print('Validation set shape : ',x_valid.shape)

x_test = np.array(convert_image_to_array(x_test))
print('Test set shape : ',x_test.shape)
```

In [6]:

```
x_train = x_train.astype('float32')/255
x_valid = x_valid.astype('float32')/255
x_test = x_test.astype('float32')/255
y_train = y_train.reshape(y_train.shape[0],1)
y_test = y_test.reshape(y_test.shape[0],1)
y_validate = y_validate.reshape(y_validate.shape[0],1)
```

In [7]:

```
plt.figure(figsize=(20,20))
classes = ['O','R']
for i in range(1,26):
    index = np.random.randint(x_train.shape[0])
    plt.subplot(5, 5, i)
    plt.imshow(np.squeeze(x_train[index]), cmap='cool')
    plt.title(classes[int(y_train[index])])
    plt.tight_layout()
plt.show()
```

In [8]:

```
from glob import glob

className = glob(train_dir + '/*' )
numberOfClass = len(className)
print("Number Of Class: ",numberOfClass)
```

In [9]:

```
datagen = ImageDataGenerator(
        featurewise_center=False,  # set input mean to 0 over the dataset
        samplewise_center=False,  # set each sample mean to 0
        featurewise_std_normalization=False,  # divide inputs by std of the dataset
        samplewise_std_normalization=False,  # divide each input by its std
        zca_whitening=False,  # apply ZCA whitening
        rotation_range=0,  # randomly rotate images in the range (degrees, 0 to 180)
        zoom_range = 0.1, # Randomly zoom image
        width_shift_range=0.2,  # randomly shift images horizontally (fraction of total
width)
        height_shift_range=0.2,  # randomly shift images vertically (fraction of total he
ight)
        horizontal_flip=False,  # randomly flip images
        vertical_flip=False)  # randomly flip images
datagen.fit(x_train)
```

Convolutional Neural Network - CNN

'''model = Sequential() model.add(Conv2D(32,(3,3),input_shape = (224,224,3))) model.add(Activation("relu")) model.add(MaxPooling2D())

model.add(Conv2D(64,(3,3))) model.add(Activation("relu")) model.add(MaxPooling2D())

model.add(Flatten()) model.add(Conv2D(128,(3,3))) model.add(Activation("relu")) model.add(MaxPooling2D()) model.add(Dense(numberOfClass)) # output model.add(Activation("sigmoid"))

model.compile(loss = "binary_crossentropy", optimizer = "adam", metrics = ["accuracy"]) batch_size = 128 '''

model = Sequential() model.add(Conv2D(16,kernel_size=(3, 3),activation='relu',input_shape=(224,224,3))) model.add(MaxPooling2D())

model.add(Conv2D(32, kernel_size=(3, 3),activation='relu')) model.add(MaxPooling2D())

model.add(Dropout(0.2)) model.add(Flatten()) model.add(Dense(512, activation='relu')) model.add(Dense(2, activation='sigmoid'))

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

In [10]:

```
train_datagen = ImageDataGenerator(rescale= 1./255)
test_datagen = ImageDataGenerator(rescale= 1./255)
```

In [11]:

```
train_generator = train_datagen.flow_from_directory(
        train_dir,
        target_size= (224,224),
        batch_size = 128,
        color_mode= "rgb",
        class_mode= "categorical")

test_generator = test_datagen.flow_from_directory(
        test_dir,
        target_size= (224,224),
        batch_size = 128,
        color_mode= "rgb",
        class_mode= "categorical")
```

In [12]:

```
'''model = Sequential()
model.add(Conv2D(32,(3,3),input_shape = (224,224,3)))
model.add(Activation("relu"))
model.add(MaxPooling2D())

model.add(Conv2D(64,(3,3)))
model.add(Activation("relu"))
model.add(MaxPooling2D())

model.add(Flatten())
model.add(Conv2D(128,(3,3)))
model.add(Activation("relu"))
model.add(MaxPooling2D())
model.add(Dense(numberOfClass)) # output
model.add(Activation("sigmoid"))

model.compile(loss = "binary_crossentropy",
              optimizer = "adam",
              metrics = ["accuracy"])
batch_size = 128 '''

model = Sequential()
model.add(Conv2D(32,kernel_size=(3, 3),activation='relu',input_shape=(100,100,3)))
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Conv2D(32, kernel_size=(3, 3),activation='relu'))
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Dropout(0.2))
model.add(Flatten())
model.add(Dense(512, activation='relu'))
model.add(Dense(2, activation='sigmoid'))
model.compile(loss="sparse_categorical_crossentropy",optimizer='adam',metrics=['accuracy'
])
```

In [13]:

```
earlystop = EarlyStopping(monitor = 'val_loss', # value being monitored for improvement
                          min_delta = 0, #Abs value and is the min change required befor
e we stop
                          patience = 15, #Number of epochs we wait before stopping
                          verbose = 1,
                          restore_best_weights = True) #keeps the best weigths once stop
ped
```

In [14]:

```
ReduceLR = ReduceLROnPlateau(patience=3, verbose=1)
```

In [15]:

```
callbacks = [earlystop, ReduceLR]
```

In [16]:

```
history = model.fit_generator(datagen.flow(x_train, y_train, batch_size= 32), epochs = 1
0, verbose=1, validation_data=(x_valid,y_validate))
```

In [17]:

```
import matplotlib.pyplot as plt

# Plot training & validation accuracy values
plt.plot(history.history['accuracy'])
plt.plot(history.history['val_accuracy'])
plt.title('Model accuracy')
plt.ylabel('Accuracy')
plt.xlabel('Epoch')
plt.legend(['Train', 'Validate'], loc='upper left')
plt.show()

# Plot training & validation loss values
plt.plot(history.history['loss'])
plt.plot(history.history['val_loss'])
plt.title('Model loss')
plt.ylabel('Loss')
plt.xlabel('Epoch')
plt.legend(['Train', 'Validate'], loc='upper left')
plt.show()
```

In [23]:

```
model = Sequential()
model.add(Conv2D(64,(3,3),input_shape = (100,100,3)))
model.add(Activation("relu"))
model.add(MaxPooling2D())

model.add(Conv2D(128,(3,3)))
model.add(Activation("relu"))
model.add(MaxPooling2D())

model.add(Conv2D(256,(3,3)))
model.add(Activation("relu"))
model.add(MaxPooling2D())

#model.add(Flatten())
model.add(Dense(256))
model.add(Activation("relu"))
model.add(Dropout(0.5))
model.add(Dense(64))
model.add(Activation("relu"))
model.add(Dropout(0.5))
model.add(Flatten())
model.add(Dense(512, activation='relu'))
model.add(Dense(2, activation='softmax'))

model.compile(loss = "sparse_categorical_crossentropy",
              optimizer = "adam",
              metrics = ["accuracy"])
```

In [24]:

```
earlystop = EarlyStopping(monitor = 'val_loss', # value being monitored for improvement
                          min_delta = 0, #Abs value and is the min change required befor
e we stop
                          patience = 15, #Number of epochs we wait before stopping
                          verbose = 1,
                          restore_best_weights = True) #keeps the best weigths once stop
ped
```

In [25]:

```
ReduceLR = ReduceLROnPlateau(patience=3, verbose=1)
```

In [26]:

```
callbacks = [earlystop, ReduceLR]
```

In [27]:

```
history = model.fit_generator(datagen.flow(x_train, y_train, batch_size= 256), epochs =
10, verbose=1, validation_data=(x_valid,y_validate))
```

In [28]:

```
model_3 = Sequential()
model_3.add(Conv2D(32,(3,3),input_shape = (224,224,3)))
model_3.add(Activation("relu"))
model_3.add(MaxPooling2D())
model_3.add(Conv2D(64,(3,3)))
model_3.add(Activation("relu"))
model_3.add(MaxPooling2D())
model_3.add(Conv2D(128,(3,3)))
model_3.add(Activation("relu"))
model_3.add(MaxPooling2D())
model_3.add(Flatten())
model_3.add(Dense(256))
model_3.add(Activation("relu"))
model_3.add(Dropout(0.5))
model_3.add(Dense(64))
model_3.add(Activation("relu"))
model_3.add(Dropout(0.5))
model_3.add(Dense(numberOfClass)) # output
model_3.add(Activation("sigmoid"))
model_3.compile(loss = "binary_crossentropy",
optimizer = "adam",
metrics = ["accuracy"])
batch_size = 256
```

In [30]:

```
train_datagen1 = ImageDataGenerator(rescale= 1./255)
test_datagen1 = ImageDataGenerator(rescale= 1./255)
```

In [31]:

```
train_generator1 = train_datagen1.flow_from_directory(
train_dir,
target_size= (224,224),
batch_size = batch_size,
color_mode= "rgb",
class_mode= "categorical")
test_generator1 = test_datagen1.flow_from_directory(
test_dir,
target_size= (224,224),
batch_size = batch_size,
color_mode= "rgb",
class_mode= "categorical")
```

In [32]:

```
hist = model_3.fit_generator(
generator = train_generator1,
epochs=10,
validation_data = test_generator1)
```

In [33]:

```
plt.figure(figsize=[10,6])
plt.plot(hist.history["accuracy"], label = "Train acc")
plt.plot(hist.history["val_accuracy"], label = "Validation acc")
plt.legend()
plt.show()
```

In [34]:

```
plt.figure(figsize=(10,6))
plt.plot(hist.history['loss'], label = "Train loss")
plt.plot(hist.history['val_loss'], label = "Validation loss")
plt.legend()
plt.show()
```

In [ ]:

```

```