In [1]:

```
#Importing libraries

import pandas as pd
import numpy as np
import cv2
import tensorflow as tf
import keras
import h5py
from keras.applications.vgg16 import VGG16, preprocess_input
from keras.models import Sequential, Model
from keras.layers import Conv2D, Flatten, MaxPool2D, Input, Dropout, Dense
from keras.layers.normalization import BatchNormalization
from keras.callbacks import EarlyStopping, Callback, ModelCheckpoint
from sklearn.preprocessing import OneHotEncoder
from sklearn.preprocessing import LabelEncoder
from keras.utils import to_categorical
from keras.optimizers import Adam
import os
from pathlib import Path
import glob
from sklearn.model_selection import train_test_split
import cv2
import matplotlib.pyplot as plt
import time
os.environ['TF_CPP_MIN_LOG_LEVEL'] = '3'

print('Imported Libraries')

print('Reading the data')
#Reading the data
member = ('/kaggle/input/garbage-classification/Garbage classification/Garbage class
catagories = os.listdir(member)
list_items = []
for cat in catagories:
    catagory_img = (member  + cat)
    for _ in (glob.glob(catagory_img +'/'+'*.jpg')):
        list_items.append([cat, _])

#Convert list into dataframe
data = pd.DataFrame(list_items,columns = ['catagory', 'filepath'], index = None)
data = data.sample(frac=1).reset_index(drop=True)
data.head(5)
data.shape


#print('Splitting the dataset')
train_data = data[1:2000]
val_data = data[2001:2200]
test_data = data[2201:2527]
```

```
Using TensorFlow backend.
Imported Libraries
Reading the data
```

In [2]:

```
#Preprocessing for Training set

def adv_preprocessing(image):
    #loading imageswith

    preimgs = []
    img = cv2.imread(image, cv2.IMREAD_UNCHANGED)

    #Setting dimensions to resize
    height = 224
    width = 224

    dim = (width, height)
    res = cv2.resize(img, dim, interpolation = cv2.INTER_LINEAR)
    preimgs.append(res)

#Removing noise from image - Gaussian blur

    blurred_img = cv2.GaussianBlur(res, (5,5),0)
    preimgs.append(blurred_img)

    #Segmentation
    #------------------------------------------------------------------
    image = res
    gray = cv2.cvtColor(image, cv2.COLOR_RGB2GRAY)
    ret,thresh = cv2.threshold(gray, 0,255,cv2.THRESH_BINARY+ cv2.THRESH_OTSU)

    #More noise removal
    #------------------------------------------------------------------
    kernal = np.ones((3,3), np.uint8)
    opening = cv2.morphologyEx(thresh, cv2.MORPH_OPEN, kernal, iterations=2)

    #Sure background area
    sure_bg = cv2.dilate(opening, kernal, iterations = 3)

    #Finding foreground area
    dist_transform = cv2.distanceTransform(opening, cv2.DIST_L2, 5)
    ret, sure_fg = cv2.threshold(dist_transform, 0.7 * dist_transform.max(), 255, 0)

    # Finding unknown region
    sure_fg = np.uint8(sure_fg)
    unknown = cv2.subtract(sure_bg, sure_fg)

    #Seperating different objects with different backgrounds
    #Markers labelling
    ret, markers  = cv2.connectedComponents(sure_fg)
    #Add one to all labels so that sure background is 0 not 1
    markers = markers+1

    #Mark the unknown region with 0
    markers[unknown == 255] = 0

    markers = cv2.watershed(res, markers)
    res[markers == -1] = [255,0,0]
    placeholder = np.random.rand(224,224)
    #Displaying the markers on image
    markers = np.dstack([markers,np.zeros((224,224)), placeholder])
    #Adding
    preimgs.append(res)
    preimgs.append(markers)

    return preimgs


#-----------------------------------------------------------------------------------
#Preprocessing for Validation and test data

def norm_data(data):
    batch_data = np.zeros((len(data),224,224,3))
    batch_labels = np.zeros((len(data),6))
    label_enc = LabelEncoder()
    y = label_enc.fit_transform(data['catagory']) #Converting data into labels
    y = y.reshape(-1,1)
    onehotenc = OneHotEncoder(handle_unknown= 'ignore')
    batch_labels = onehotenc.fit_transform(y).toarray() #Onehot Encoding data

    #Image normalization
    height = 224
    width = 224
    dim = (height, width)

    for i in range(len(data)):
        img = cv2.imread(data.iloc[i]['filepath'])
        res_img = cv2.resize(img, dim,interpolation = cv2.INTER_LINEAR)
        res_img = res_img.astype(np.float32)/255
        batch_data[i] = res_img

    return batch_data, batch_labels
```

In [3]:

```
#Generator for training data

def adv_gendata(data, batch_size):
    labelenc = LabelEncoder()
    n = len(data)
    steps = n//batch_size
    data['labels'] = labelenc.fit_transform(data['catagory'])
    enc = OneHotEncoder(handle_unknown='ignore')
    enc_df = pd.DataFrame(enc.fit_transform(data[['labels']]).toarray())

    #Defining numpy array to contain label and image data

    batch_data = np.zeros((batch_size, 224,224,3), dtype = np.float32)
    batch_labels = np.zeros((batch_size, 6))
    indices = np.arange(n)

    c1 = 0
    i = 0
    #Initialize counter
    while True:
        np.random.shuffle(indices)
        next_batch = indices[(i*batch_size):(i+1)*batch_size]
        count = 0


        for j, idx in enumerate(next_batch):
            if count <= batch_size-4:
                img_name = data.iloc[idx]['filepath']
                aug_img = adv_preprocessing(data['filepath'].iloc[idx])
                encoded_label = enc_df.iloc[idx]
                batch_data[count+0] = aug_img[0]
                batch_labels[count+0] = encoded_label
                batch_data[count+1] = aug_img[1]
                batch_labels[count+1] = encoded_label
                batch_data[count+2] = aug_img[2]
                batch_labels[count+2] = encoded_label
                batch_data[count+3] = aug_img[3]
                batch_labels[count+3] = encoded_label
                count +=4
                c1 = c1+1

            else:
                count+=1

            if count==batch_size:
                i += 1
                break
        i+=1
        yield batch_data, batch_labels

    if i>=steps:
        i=0
```

In [4]:

```
print('Building the model')


#Working on the model
def build_model():
    model = Sequential()
    input_size  = Input(shape = (224,224,3), name  =  'Input_Image')

    #Layer 1 - Deapth Layer 1,2
    x = Conv2D(64,(3,3), activation = 'relu', padding = 'same', name = 'ConvLayer1'
    x = Conv2D(64,(3,3), activation = 'relu', padding = 'same', name = 'ConvLayer2'
    x = MaxPool2D((2,2), name = 'Maxpool1')(x)
    x = BatchNormalization(name = 'bn1')(x)

    #Layer 2 - Deapth layer 3,4
    x = Conv2D(128,(3,3), activation = 'relu', padding = 'same', name = 'ConvLayer3'
    x = Conv2D(128,(3,3), activation = 'relu', padding = 'same', name = 'ConvLayer4'
    x = MaxPool2D((2,2), name = 'Maxpoo12')(x)
    x = BatchNormalization(name = 'bn2')(x)
#     #Layer 3 - Deapth layer 3
    x = Conv2D(256,(3,3), activation= 'relu',padding = 'same',  name = 'ConvLayer5')
    x = MaxPool2D((2,2), name = 'Maxpool3')(x)
    x = BatchNormalization(name = 'bn3')(x)

    #Flatten the model

    x = Flatten(name = 'Flatten')(x)
    x = Dense(256, activation = 'relu', name = 'FC1')(x)
    x = Dropout(0.7, name = 'Dropout1')(x)
    x = Dense(256, activation = 'relu', name = 'FC2')(x)
    x = Dropout(0.7, name = 'Dropout2')(x)
    x = Dense(6, activation = 'softmax', name = 'Fc3')(x)

    model = Model(input = input_size , output = x)
    return model

#Building the model and summary

model = build_model()
model.summary()

#Initialize the first layer with the weights of Imagenet

f = h5py.File('/kaggle/input/vgg16/vgg16_weights_tf_dim_ordering_tf_kernels_notop.h5

#Selecting the layers for which weights needs to be set
w,b = f['block1_conv1']['block1_conv1_W_1:0'], f['block1_conv1']['block1_conv1_b_1:0
model.layers[1].set_weights = [w,b]

w,b = f['block1_conv2']['block1_conv2_W_1:0'], f['block1_conv2']['block1_conv2_b_1:0
model.layers[2].set_weights = [w,b]

f.close()
model.summary()
```

```python
#Compiling  the model
from keras.optimizers import SGD
opt = SGD(lr=0.01)
model.compile(loss = "categorical_crossentropy", optimizer = opt)
es = EarlyStopping(patience=5)
chkpt = ModelCheckpoint(filepath= 'bestmodel',save_best_only=True, save_weights_only
model.compile(loss= 'binary_crossentropy', metrics= ['accuracy'], optimizer= opt)
```

```
Building the model
Model: "model_1"
_________________________________________________________________
Layer (type)                 Output Shape              Param #   
=================================================================
Input_Image (InputLayer)     (None, 224, 224, 3)       0         
_________________________________________________________________
ConvLayer1 (Conv2D)          (None, 224, 224, 64)      1792      
_________________________________________________________________
ConvLayer2 (Conv2D)          (None, 224, 224, 64)      36928     
_________________________________________________________________
Maxpool1 (MaxPooling2D)      (None, 112, 112, 64)      0         
_________________________________________________________________
bn1 (BatchNormalization)     (None, 112, 112, 64)      256       
_________________________________________________________________
ConvLayer3 (Conv2D)          (None, 112, 112, 128)     73856     
_________________________________________________________________
ConvLayer4 (Conv2D)          (None, 112, 112, 128)     147584    
_________________________________________________________________
Maxpoo12 (MaxPooling2D)      (None, 56, 56, 128)       0         
_________________________________________________________________
bn2 (BatchNormalization)     (None, 56, 56, 128)       512       
_________________________________________________________________
ConvLayer5 (Conv2D)          (None, 56, 56, 256)       295168    
_________________________________________________________________
Maxpool3 (MaxPooling2D)      (None, 28, 28, 256)       0         
_________________________________________________________________
bn3 (BatchNormalization)     (None, 28, 28, 256)       1024      
_________________________________________________________________
Flatten (Flatten)            (None, 200704)            0         
_________________________________________________________________
FC1 (Dense)                  (None, 256)               51380480  
_________________________________________________________________
Dropout1 (Dropout)           (None, 256)               0         
_________________________________________________________________
FC2 (Dense)                  (None, 256)               65792     
_________________________________________________________________
Dropout2 (Dropout)           (None, 256)               0         
_________________________________________________________________
Fc3 (Dense)                  (None, 6)                 1542      
=================================================================
Total params: 52,004,934
Trainable params: 52,004,038
Non-trainable params: 896
_________________________________________________________________
Model: "model_1"
_________________________________________________________________
Layer (type)                 Output Shape              Param #   
=================================================================
Input_Image (InputLayer)     (None, 224, 224, 3)       0         
_________________________________________________________________
ConvLayer1 (Conv2D)          (None, 224, 224, 64)      1792      
_________________________________________________________________
ConvLayer2 (Conv2D)          (None, 224, 224, 64)      36928     
_________________________________________________________________
Maxpool1 (MaxPooling2D)      (None, 112, 112, 64)      0         
_________________________________________________________________
bn1 (BatchNormalization)     (None, 112, 112, 64)      256       
_________________________________________________________________
ConvLayer3 (Conv2D)          (None, 112, 112, 128)     73856     
_________________________________________________________________
```

```
ConvLayer4 (Conv2D)          (None, 112, 112, 128)     147584    
_________________________________________________________________
Maxpoo12 (MaxPooling2D)      (None, 56, 56, 128)       0         
_________________________________________________________________
bn2 (BatchNormalization)     (None, 56, 56, 128)       512       
_________________________________________________________________
ConvLayer5 (Conv2D)          (None, 56, 56, 256)       295168    
_________________________________________________________________
Maxpool3 (MaxPooling2D)      (None, 28, 28, 256)       0         
_________________________________________________________________
bn3 (BatchNormalization)     (None, 28, 28, 256)       1024      
_________________________________________________________________
Flatten (Flatten)            (None, 200704)            0         
_________________________________________________________________
FC1 (Dense)                  (None, 256)               51380480  
_________________________________________________________________
Dropout1 (Dropout)           (None, 256)               0         
_________________________________________________________________
FC2 (Dense)                  (None, 256)               65792     
_________________________________________________________________
Dropout2 (Dropout)           (None, 256)               0         
_________________________________________________________________
Fc3 (Dense)                  (None, 6)                 1542      
=================================================================
Total params: 52,004,934
Trainable params: 52,004,038
Non-trainable params: 896
_________________________________________________________________
/opt/conda/lib/python3.6/site-packages/ipykernel_launcher.py:34: UserWarning: Update
your `Model` call to the Keras 2 API: `Model(inputs=Tensor("In..., outputs=Tensor("F
c...)`
```

In [5]:

```
#Training the model

batch_size = 20
train_data_gen = adv_gendata(train_data,20)
val_data, val_labels = norm_data(val_data)
test_data, test_labels = norm_data(test_data)

nb_train_steps = train_data.shape[0]//batch_size

epochs = 10
model.fit_generator(train_data_gen,epochs=epochs, verbose = 1, steps_per_epoch=nb_tr
```

```
Epoch 1/10
/opt/conda/lib/python3.6/site-packages/ipykernel_launcher.py:7: SettingWithCopyWarni
ng:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: http://pandas.pydata.org/pandas-docs/stable/us
er_guide/indexing.html#returning-a-view-versus-a-copy
  import sys
99/99 [==============================] - 747s 8s/step - loss: 2.0365 - accuracy: 0.7
969 - val_loss: 1.8714 - val_accuracy: 0.7487
Epoch 2/10
99/99 [==============================] - 743s 8s/step - loss: 0.9206 - accuracy: 0.9
219 - val_loss: 4.4187 - val_accuracy: 0.7119
Epoch 3/10
99/99 [==============================] - 744s 8s/step - loss: 0.8683 - accuracy: 0.9
311 - val_loss: 4.4187 - val_accuracy: 0.7119
Epoch 4/10
99/99 [==============================] - 741s 7s/step - loss: 0.7981 - accuracy: 0.9
378 - val_loss: 4.4187 - val_accuracy: 0.7119
Epoch 5/10
99/99 [==============================] - 741s 7s/step - loss: 0.7982 - accuracy: 0.9
383 - val_loss: 3.8535 - val_accuracy: 0.7487
Epoch 6/10
```

```
99/99 [==============================] - 743s 8s/step - loss: 0.8400 - accuracy: 0.9
369 - val_loss: 4.4187 - val_accuracy: 0.7119
Epoch 7/10
99/99 [==============================] - 745s 8s/step - loss: 0.8146 - accuracy: 0.9
407 - val_loss: 4.4187 - val_accuracy: 0.7119
Epoch 8/10
99/99 [==============================] - 746s 8s/step - loss: 0.7704 - accuracy: 0.9
436 - val_loss: 4.4187 - val_accuracy: 0.7119
Epoch 9/10
99/99 [==============================] - 744s 8s/step - loss: 0.7882 - accuracy: 0.9
426 - val_loss: 4.4187 - val_accuracy: 0.7119
Epoch 10/10
99/99 [==============================] - 744s 8s/step - loss: 0.8022 - accuracy: 0.9
412 - val_loss: 4.4187 - val_accuracy: 0.7119
```

Out[5]:

```
<keras.callbacks.callbacks.History at 0x7f6f6c3a3eb8>
```

In [6]:

```
#Evaluating the model on test data

score = model.evaluate(test_data, test_labels, verbose=0)
print('Test loss:', score[0])
print('Test accuracy:', score[1])
```

```
Test loss: 4.312577487500898
Test accuracy: 0.7188138961791992
```