Configuration Manual

MSc Research Project
Data Analytics (MSCDA-B)

Himanshu Gupta
Student ID: x18203302

School of Computing
National College of Ireland

Supervisor: Dr. Muhammad Iqbal

National College of Ireland
Project Submission Sheet
School of Computing

Student Name:	Himanshu Gupta
Student ID:	x18203302
Programme:	Data Analytics (MSCDA-B)
Year:	2020
Module:	MSc Research Project
Supervisor:	Dr. Muhammad Iqbal
Submission Due Date:	28/09/2020
Project Title:	Configuration Manual
Word Count:	1294
Page Count:	14

Signature:	
Date:	28th September 2020

PLEASE READ THE FOLLOWING INSTRUCTIONS AND CHECKLIST:

Attach a completed copy of this sheet to each project (including multiple copies).	☐
Attach a Moodle submission receipt of the online project submission, to each project (including multiple copies).	☐
You must ensure that you retain a HARD COPY of the project, both for your own reference and in case a project is lost or mislaid. It is not sufficient to keep a copy on computer.	☐

Office Use Only	
Signature:	
Date:	
Penalty Applied (if applicable):	

Configuration Manual

Himanshu Gupta
x18203302

28th September 2020

1 Introduction

This manual presents the system configuration required to run the submitted project. It contains all the packages, libraries, and programming codes written and used during the project implementation of : ”Trash Image Classification System using Machine Learning and Deep Learning Algorithms”.

2 System Configurations

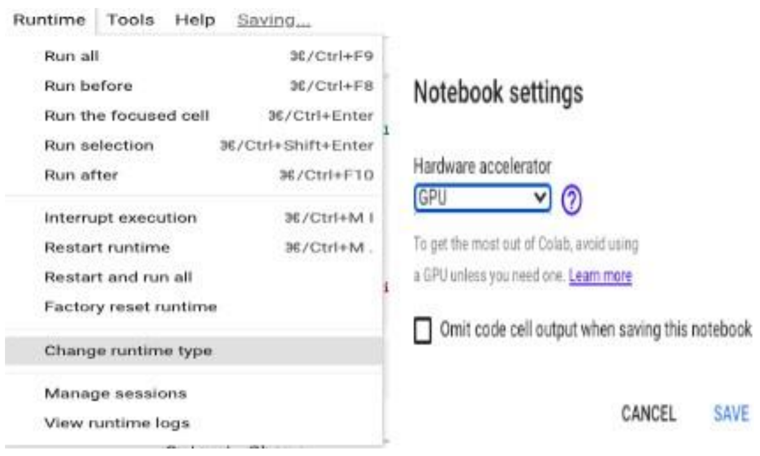
2.1 Hardware

Following Hardware configuration used:
RAM:8GB **System type**: Macintosh 64 bit **Processor**: Dual-Core Intel Core i5 **CPU**:1.8GHz **Storage**:1 TB HDD **GPU:**:Intel HD Graphics 6000 1536 MB.

2.2 Software

- **PyCharm**: It is an IDE that is majorly used to run the python code. Two versions are there one is professional and one is community version. For this project Community edition has been downloaded from this website[1].
- **Google Colaboratory:** Also known as Colab, this is an online cloud service that provides an environment to run your Jupyter notebooks freely. All the basic packages for machine learning problems are already installed in the environment like TensorFlow, Keras, pandas, and user needs to import these packages according to their usage. However, to run a specific version of software the version should be mentioned in the notebook before calling their functions. Three modes are provided to run the notebook which are None, GPU, and TPU. GPU setting was used to execute the notebooks. Authenticated google drive access is necessary to access Colab. Sometimes GPU is available for limited usage per day and in that case, None can be selected in the settings as shown in Figure 1.

[1]https://www.jetbrains.com/pycharm/



Notebook settings

Hardware accelerator
GPU

To get the most out of Colab, avoid using a GPU unless you need one. Learn more

☐ Omit code cell output when saving this notebook

Figure 1: Settings: Google Colaboratary

3 Project Development

The main steps in this research development are data pre-processing(data downloading, data analyses, new data structure creation, removing unwanted columns), conversion from image to NumPy arrays, creating dummy variables, data split, data array reshaping for each model, data normalization in several stages. Several codes have been written for successfully performing and evaluating all the experiments such: creating a baseline Sequential Keras, ResNet-50, VGG-19 neural network, adding several layers with different weights and defining training parameters, selecting hyper-parameters for the XGBoost model. Writing codes for running all the models at different k-folds for cross-validation, changing sample size and many epochs, creating classification matrix, training testing accuracy, and plotting evaluation graphs.

3.1 Data Gathering

The TACO (Trash Annotation in Context) dataset used in this study is not available directly. This involves two below steps:

1) Go to the Taco site [2] and click on the Download button. Download 'annotations.json' from [3] which contains information like URL path of images hosted on Flickr server, filenames, categories, bounding boxes, images width, and height, etc. Store this annotation file in the local drive directory.

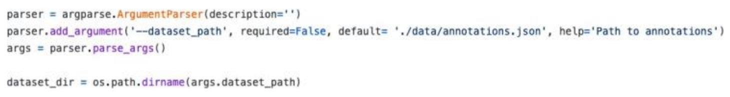
2) Download python file 'download.py' from the same site [4] and open this file in PyCharm IDE give the directory path of 'annotation file' downloaded above as shown in Figure 1. This download script downloaded all 1500 images in 10 sub folders batches in the mentioned path of the local drive.

3. To access data in Google Colab all the data needs to upload on google drive from local drive in a folder 'data' (folder name could be any).

4. Both 'annotations.json' and 'download.py' files were attached in the project artifacts after giving the proper credits and references to the original authors of data.

5. Created two folders in main directory path with names as shown in figure 3

[2]http://tacodataset.org/
[3]https://github.com/pedropro/TACO/blob/master/data/annotations.json
[4]https://github.com/pedropro/TACO/blob/master/download.py



```
parser = argparse.ArgumentParser(description='')
parser.add_argument('--dataset_path', required=False, default= './data/annotations.json', help='Path to annotations')
args = parser.parse_args()

dataset_dir = os.path.dirname(args.dataset_path)
```

Figure 2: Annotation file path

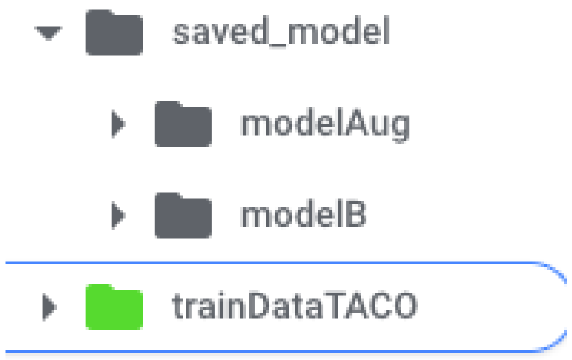
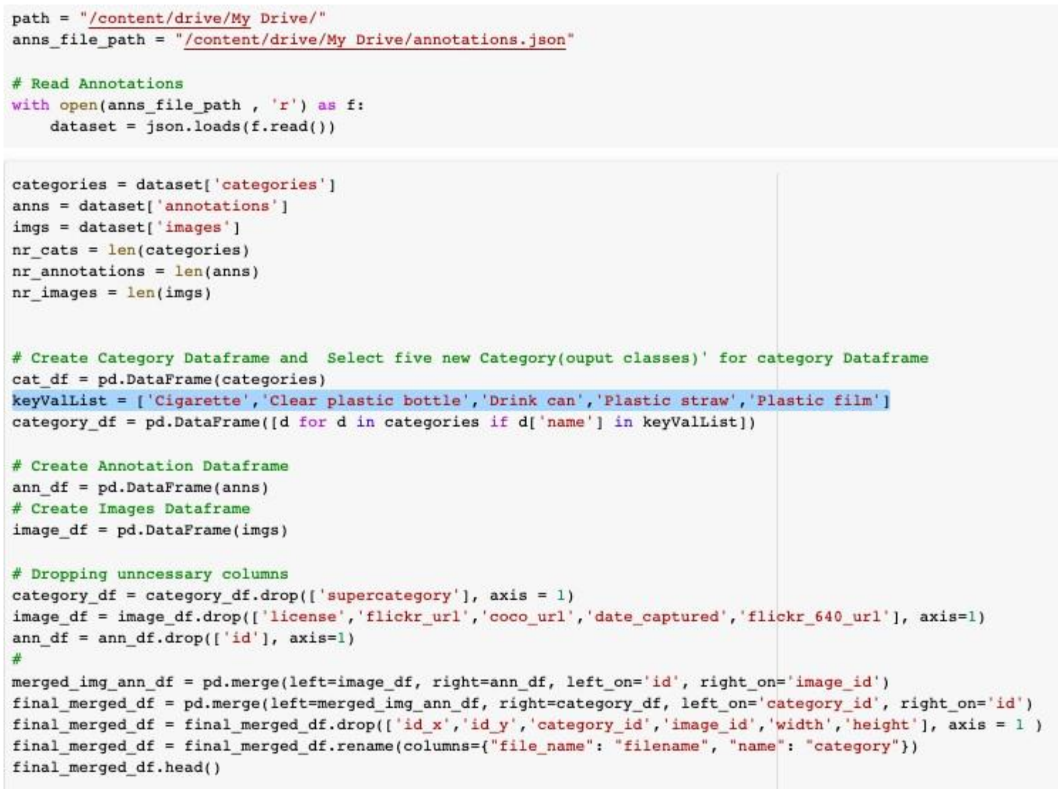


Figure 3: folder structure

3.2 Data Preparation

Data was prepared for baseline model and then data augmentation has been done to generate more data. Python script for data preparation shown in Figure 4. Bounding boxes were fetched from the initial data frame and the padding of '20' has adjusted to find the minimum and maximum value of the x-axis and y-axis. A null check has been run on the data and then converted the data frame into CSV format and saved in the drive directory path in Figure 5.



```
path = "/content/drive/My Drive/"
anns_file_path = "/content/drive/My Drive/annotations.json"

# Read Annotations
with open(anns_file_path , 'r') as f:
    dataset = json.loads(f.read())
```

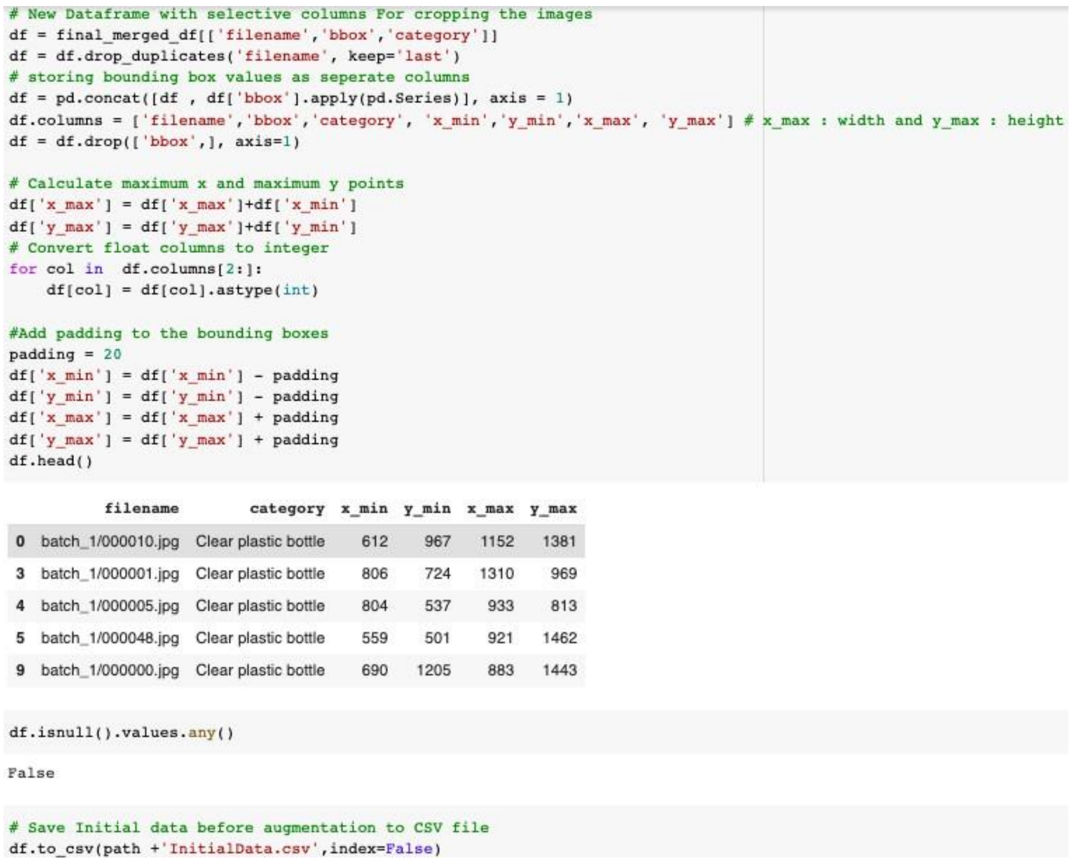
```
categories = dataset['categories']
anns = dataset['annotations']
imgs = dataset['images']
nr_cats = len(categories)
nr_annotations = len(anns)
nr_images = len(imgs)


# Create Category Dataframe and  Select five new Category(ouput classes)' for category Dataframe
cat_df = pd.DataFrame(categories)
keyValList = ['Cigarette','Clear plastic bottle','Drink can','Plastic straw','Plastic film']
category_df = pd.DataFrame([d for d in categories if d['name'] in keyValList])

# Create Annotation Dataframe
ann_df = pd.DataFrame(anns)
# Create Images Dataframe
image_df = pd.DataFrame(imgs)

# Dropping unncessary columns
category_df = category_df.drop(['supercategory'], axis = 1)
image_df = image_df.drop(['license','flickr_url','coco_url','date_captured','flickr_640_url'], axis=1)
ann_df = ann_df.drop(['id'], axis=1)
#
merged_img_ann_df = pd.merge(left=image_df, right=ann_df, left_on='id', right_on='image_id')
final_merged_df = pd.merge(left=merged_img_ann_df, right=category_df, left_on='category_id', right_on='id')
final_merged_df = final_merged_df.drop(['id_x','id_y','category_id','image_id','width','height'], axis = 1 )
final_merged_df = final_merged_df.rename(columns={"file_name": "filename", "name": "category"})
final_merged_df.head()
```

Figure 4: First Part of the script which create data frame with selective five categories and remove duplicate values from the data

In the continuation of above script, after creating the initial data frame augmented data generated through the script.

- A free open source Python image library ’PIL’ has been imported.
- Image cropping, Rotation, Gaussian blur ,horizontal flip functions applied to get these four types of images.
- Images belongs to ’Drink Can’ and ’Plastic Straw’ were very lessor as compare to other categories and therefore another set of vertical flipped images generated for these two categories.
- Bounding boxes columns dropped from the dataset.

The code snippet is shown in Figure 7. Generated data further divided into train and test dataset using scikit-learn which is a machine learning open-source library.



```python
# New Dataframe with selective columns For cropping the images
df = final_merged_df[['filename','bbox','category']]
df = df.drop_duplicates('filename', keep='last')
# storing bounding box values as seperate columns
df = pd.concat([df , df['bbox'].apply(pd.Series)], axis = 1)
df.columns = ['filename','bbox','category', 'x_min','y_min','x_max', 'y_max'] # x_max : width and y_max : height
df = df.drop(['bbox',], axis=1)

# Calculate maximum x and maximum y points
df['x_max'] = df['x_max']+df['x_min']
df['y_max'] = df['y_max']+df['y_min']
# Convert float columns to integer
for col in  df.columns[2:]:
    df[col] = df[col].astype(int)

#Add padding to the bounding boxes
padding = 20
df['x_min'] = df['x_min'] - padding
df['y_min'] = df['y_min'] - padding
df['x_max'] = df['x_max'] + padding
df['y_max'] = df['y_max'] + padding
df.head()
```

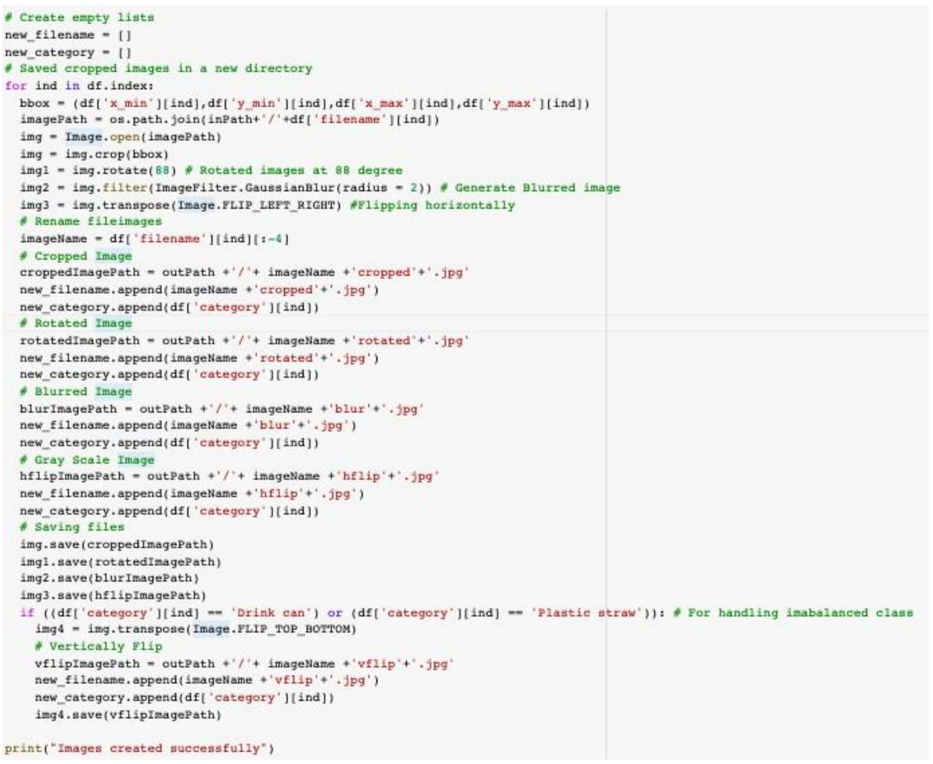
	filename	category	x_min	y_min	x_max	y_max
0	batch_1/000010.jpg	Clear plastic bottle	612	967	1152	1381
3	batch_1/000001.jpg	Clear plastic bottle	806	724	1310	969
4	batch_1/000005.jpg	Clear plastic bottle	804	537	933	813
5	batch_1/000048.jpg	Clear plastic bottle	559	501	921	1462
9	batch_1/000000.jpg	Clear plastic bottle	690	1205	883	1443

```python
df.isnull().values.any()
```

```
False
```

```python
# Save Initial data before augmentation to CSV file
df.to_csv(path +'InitialData.csv',index=False)
```

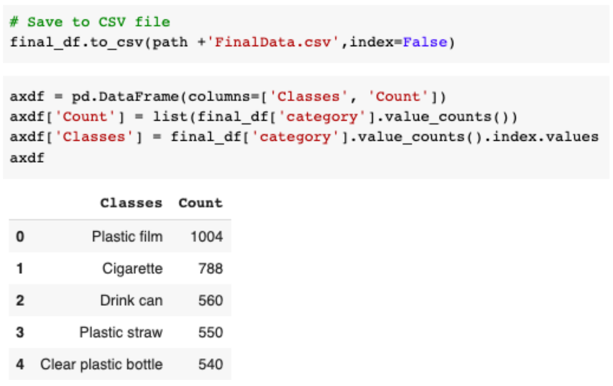
Figure 5: Initial data frame saved into CSV file which used by various base model



```python
# Create empty lists
new_filename = []
new_category = []
# Saved cropped images in a new directory
for ind in df.index:
  bbox = (df['x_min'][ind],df['y_min'][ind],df['x_max'][ind],df['y_max'][ind])
  imagePath = os.path.join(inPath+'/'+df['filename'][ind])
  img = Image.open(imagePath)
  img = img.crop(bbox)
  img1 = img.rotate(88) # Rotated images at 88 degree
  img2 = img.filter(ImageFilter.GaussianBlur(radius = 2)) # Generate Blurred image
  img3 = img.transpose(Image.FLIP_LEFT_RIGHT) #Flipping horizontally
  # Rename fileimages
  imageName = df['filename'][ind][:-4]
  # Cropped Image
  croppedImagePath = outPath +'/'+ imageName +'cropped'+'.jpg'
  new_filename.append(imageName +'cropped'+'.jpg')
  new_category.append(df['category'][ind])
  # Rotated Image
  rotatedImagePath = outPath +'/'+ imageName +'rotated'+'.jpg'
  new_filename.append(imageName +'rotated'+'.jpg')
  new_category.append(df['category'][ind])
  # Blurred Image
  blurImagePath = outPath +'/'+ imageName +'blur'+'.jpg'
  new_filename.append(imageName +'blur'+'.jpg')
  new_category.append(df['category'][ind])
  # Gray Scale Image
  hflipImagePath = outPath +'/'+ imageName +'hflip'+'.jpg'
  new_filename.append(imageName +'hflip'+'.jpg')
  new_category.append(df['category'][ind])
  # Saving files
  img.save(croppedImagePath)
  img1.save(rotatedImagePath)
  img2.save(blurImagePath)
  img3.save(hflipImagePath)
  if ((df['category'][ind] == 'Drink can') or (df['category'][ind] == 'Plastic straw')): # For handling imabalanced class
    img4 = img.transpose(Image.FLIP_TOP_BOTTOM)
    # Vertically Flip
    vflipImagePath = outPath +'/'+ imageName +'vflip'+'.jpg'
    new_filename.append(imageName +'vflip'+'.jpg')
    new_category.append(df['category'][ind])
    img4.save(vflipImagePath)

print("Images created successfully")
```

Figure 6: Code for generating Augmented data

Final data has saved into csv file format in the main directory folder and data has been fetched from there while implementing the model.



```
# Save to CSV file
final_df.to_csv(path +'FinalData.csv',index=False)
```

```
axdf = pd.DataFrame(columns=['Classes', 'Count'])
axdf['Count'] = list(final_df['category'].value_counts())
axdf['Classes'] = final_df['category'].value_counts().index.values
axdf
```

	Classes	Count
0	Plastic film	1004
1	Cigarette	788
2	Drink can	560
3	Plastic straw	550
4	Clear plastic bottle	540

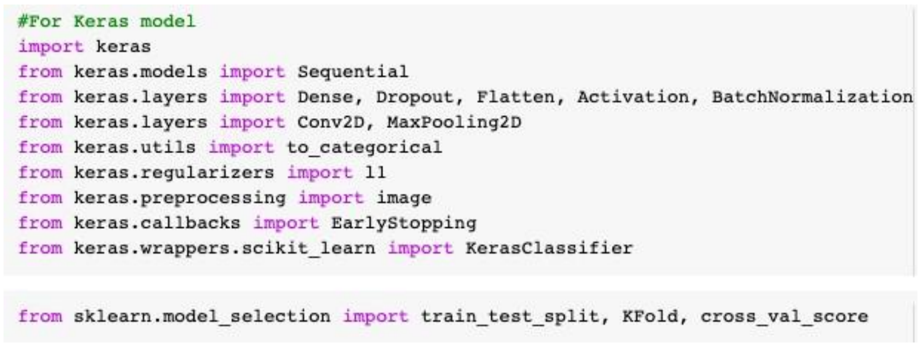
Figure 7: Final data

4 Codes for machine and deep learning models

The codes for neural network models involve importing Keras sequential model layers and model initialization. Neural network models can not read images directly so need to convert it into NumPy array and reshaping according to the model input dimensions.

4.1 Experiments with Sequential Keras Baseline Model

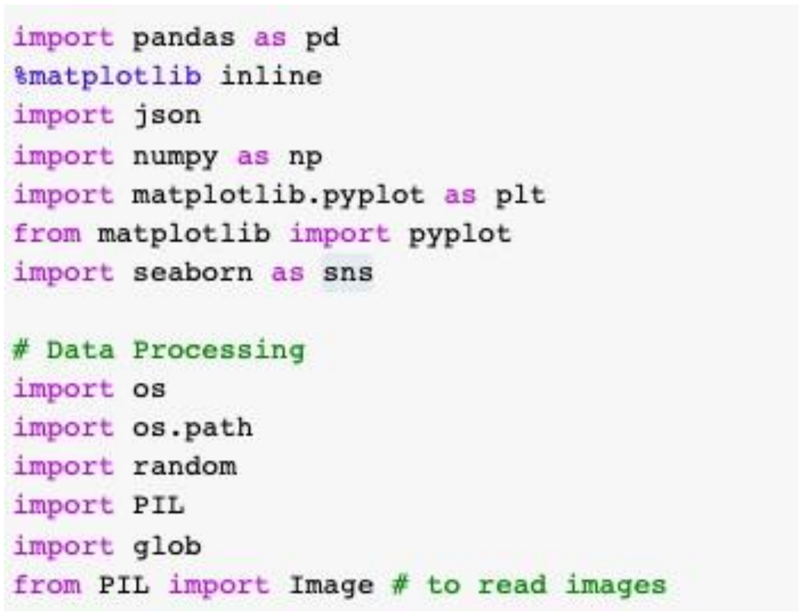
Two baseline models were developed for initial data and second for the augmented data. The input array needs to created before splitting into training and testing data. The early parameters setting keeps the same for both the model. After model evaluation, the cross-validation k-fold method implemented with 10 splits as shown in figure 13. But first needs to import all the required libraries like in below figure 9. Pillow is the main image processing library used to augment image data and have functions like rotate, crop etc. After that all keras layers and models along with early stopping package which is used for model optimization as shown in Figure 8



```
#For Keras model
import keras
from keras.models import Sequential
from keras.layers import Dense, Dropout, Flatten, Activation, BatchNormalization
from keras.layers import Conv2D, MaxPooling2D
from keras.utils import to_categorical
from keras.regularizers import l1
from keras.preprocessing import image
from keras.callbacks import EarlyStopping
from keras.wrappers.scikit_learn import KerasClassifier
```

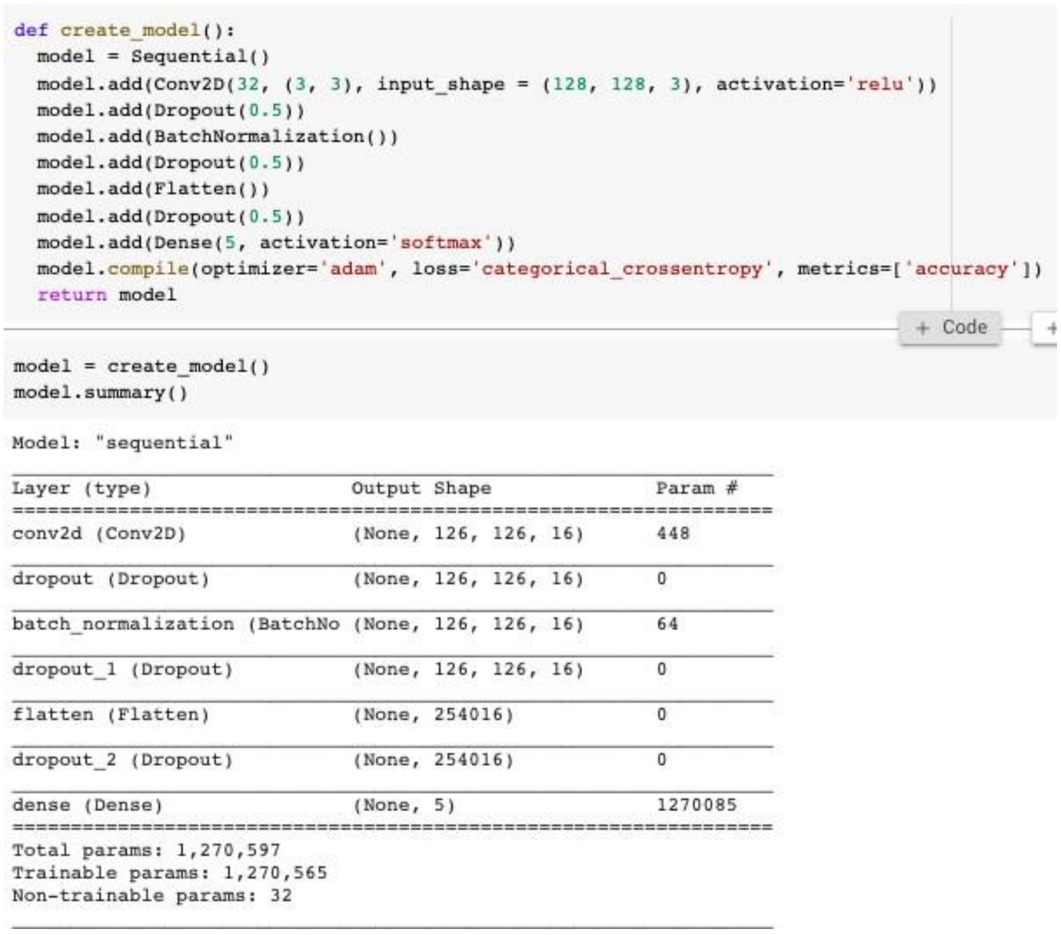
```
from sklearn.model_selection import train_test_split, KFold, cross_val_score
```

Figure 8: Required Keras Libraries



```python
import pandas as pd
%matplotlib inline
import json
import numpy as np
import matplotlib.pyplot as plt
from matplotlib import pyplot
import seaborn as sns

# Data Processing
import os
import os.path
import random
import PIL
import glob
from PIL import Image # to read images
```

Figure 9: Required Libraries

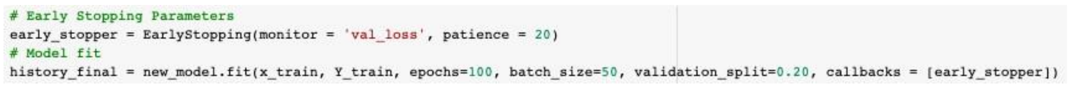


```python
def create_model():
  model = Sequential()
  model.add(Conv2D(32, (3, 3), input_shape = (128, 128, 3), activation='relu'))
  model.add(Dropout(0.5))
  model.add(BatchNormalization())
  model.add(Dropout(0.5))
  model.add(Flatten())
  model.add(Dropout(0.5))
  model.add(Dense(5, activation='softmax'))
  model.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['accuracy'])
  return model
```

```python
model = create_model()
model.summary()
```

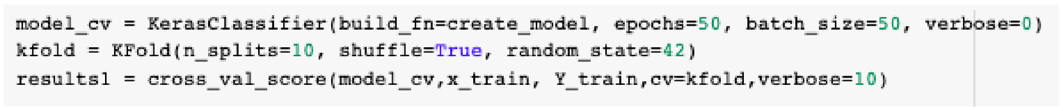
```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d (Conv2D)              (None, 126, 126, 16)      448
_________________________________________________________________
dropout (Dropout)            (None, 126, 126, 16)      0
_________________________________________________________________
batch_normalization (BatchNo (None, 126, 126, 16)      64
_________________________________________________________________
dropout_1 (Dropout)          (None, 126, 126, 16)      0
_________________________________________________________________
flatten (Flatten)            (None, 254016)            0
_________________________________________________________________
dropout_2 (Dropout)          (None, 254016)            0
_________________________________________________________________
dense (Dense)                (None, 5)                 1270085
=================================================================
Total params: 1,270,597
Trainable params: 1,270,565
Non-trainable params: 32
_________________________________________________________________
```

Figure 10: Keras Sequential model baseline structure from the scratch



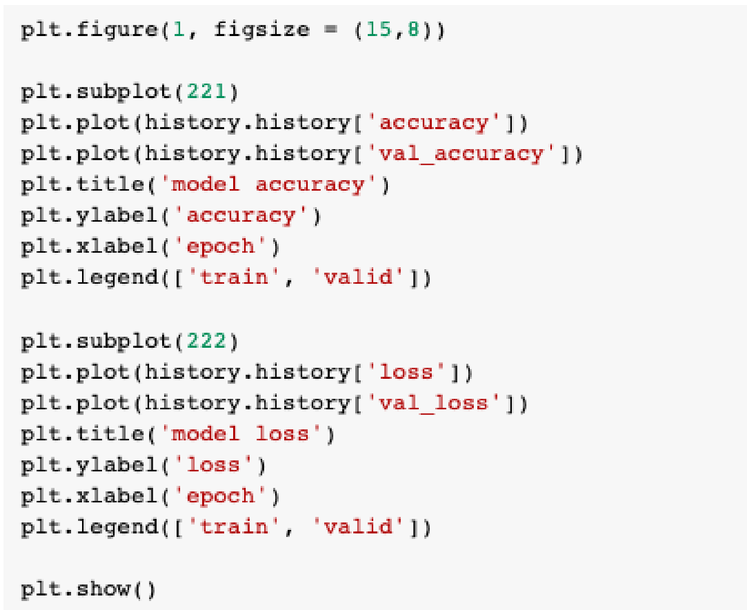
```
# Early Stopping Parameters
early_stopper = EarlyStopping(monitor = 'val_loss', patience = 20)
# Model fit
history_final = new_model.fit(x_train, Y_train, epochs=100, batch_size=50, validation_split=0.20, callbacks = [early_stopper])
```

Figure 11: Keras Early stopping package imported and use to stop the training if it is not improving to same the computational and time cost. Also, for overfitting the model



```
model_cv = KerasClassifier(build_fn=create_model, epochs=50, batch_size=50, verbose=0)
kfold = KFold(n_splits=10, shuffle=True, random_state=42)
results1 = cross_val_score(model_cv,x_train, Y_train,cv=kfold,verbose=10)
```

Figure 12: K-fold cross validation

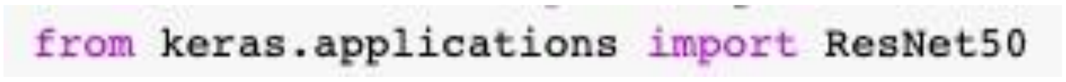


```
plt.figure(1, figsize = (15,8))

plt.subplot(221)
plt.plot(history.history['accuracy'])
plt.plot(history.history['val_accuracy'])
plt.title('model accuracy')
plt.ylabel('accuracy')
plt.xlabel('epoch')
plt.legend(['train', 'valid'])

plt.subplot(222)
plt.plot(history.history['loss'])
plt.plot(history.history['val_loss'])
plt.title('model loss')
plt.ylabel('loss')
plt.xlabel('epoch')
plt.legend(['train', 'valid'])

plt.show()
```

Figure 13: Plot evaluation results

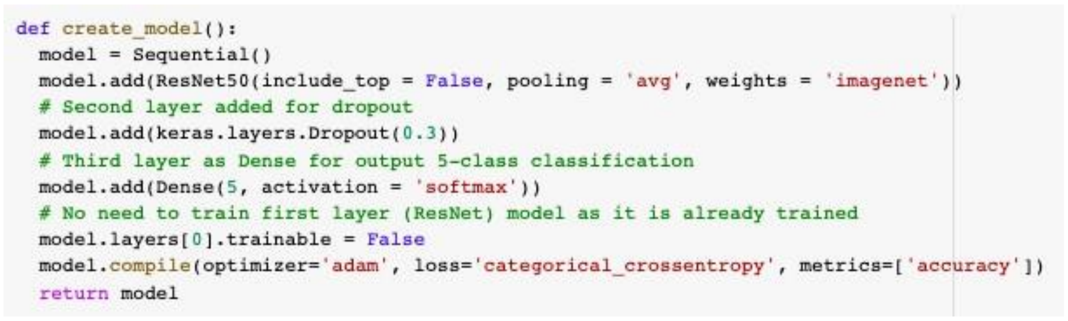
4.2 Experiments with ResNet-50 Model

The code for ResNet-50 model initialisation is shown in Figure 15. The weights for the ResNet which trained on 'imagenet' has downloaded in the model call. The first layer of the model is not trainable because it has already trained on imagenet. The summary of the model is shown in Figure 16. Import ResNet-50 model from keras application which will load the ResNet-50 object during the initialisation call.



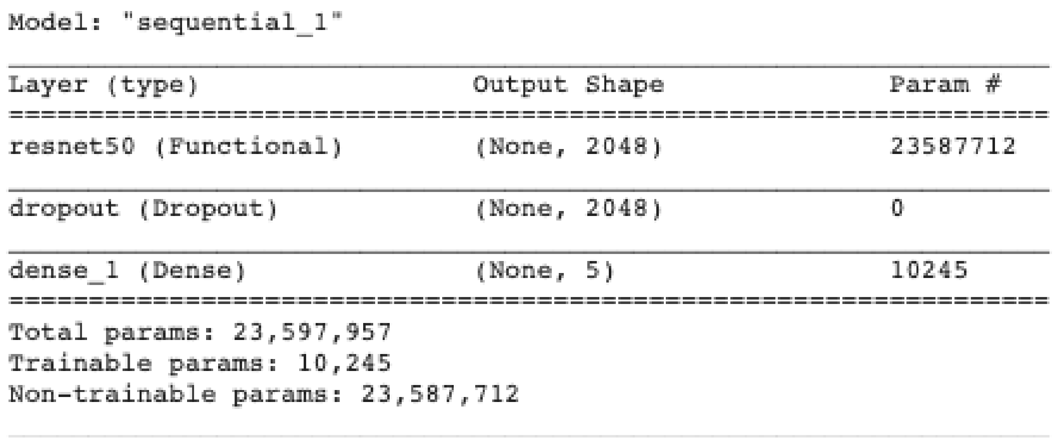
```
from keras.applications import ResNet50
```

Figure 14: ResNet-50 Model Import



```
def create_model():
  model = Sequential()
  model.add(ResNet50(include_top = False, pooling = 'avg', weights = 'imagenet'))
  # Second layer added for dropout
  model.add(keras.layers.Dropout(0.3))
  # Third layer as Dense for output 5-class classification
  model.add(Dense(5, activation = 'softmax'))
  # No need to train first layer (ResNet) model as it is already trained
  model.layers[0].trainable = False
  model.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['accuracy'])
  return model
```

Figure 15: ResNet-50 Model initial object creation.



```
Model: "sequential_1"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
resnet50 (Functional)        (None, 2048)              23587712
_________________________________________________________________
dropout (Dropout)            (None, 2048)              0
_________________________________________________________________
dense_1 (Dense)              (None, 5)                 10245
=================================================================
Total params: 23,597,957
Trainable params: 10,245
Non-trainable params: 23,587,712
_________________________________________________________________
```

Figure 16: Trainable and Non-trainable parameters of ResNet-50

After plotting all the evaluation plots the confusion matrix created using sklearn metrics plot of confusion matrix. The code reference for plotting the confusion matrix is referred from [5]

[5]https://analyticsindiamag.com/transfer-learning-for-multi-class-image-classification-using-deep-convolutional-neural-network/

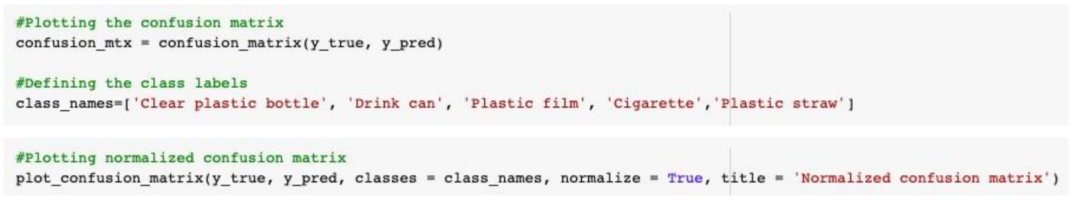

```
#Plotting the confusion matrix
confusion_mtx = confusion_matrix(y_true, y_pred)

#Defining the class labels
class_names=['Clear plastic bottle', 'Drink can', 'Plastic film', 'Cigarette','Plastic straw']

#Plotting normalized confusion matrix
plot_confusion_matrix(y_true, y_pred, classes = class_names, normalize = True, title = 'Normalized confusion matrix')
```

Figure 17: Confusion Matrix Code

4.3 Experiments with VGG-19 Model

VGG-19 model first need to import from keras applicaiton along with preprocess input which is used to convert input image data array into preprocessed train and test features required by VGG-19 for feature extraction and model training. The code is shown below in **??** . The code for VGG-19 model initialisation is shown in Figure 19. The weights for the VGG-19 was downloaded itself with the function call. This summary of model is shown in Figure 20. To run the cross validation input processes again according to VGG-19 and the code shown in Figure 22 for the same.

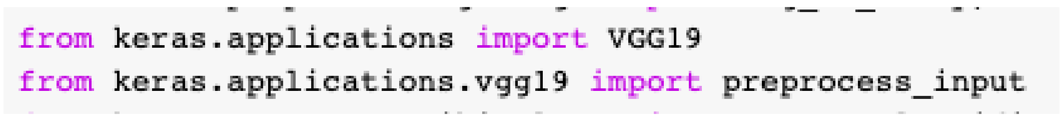

```
from keras.applications import VGG19
from keras.applications.vgg19 import preprocess_input
```

Figure 18: VGG-19 Import

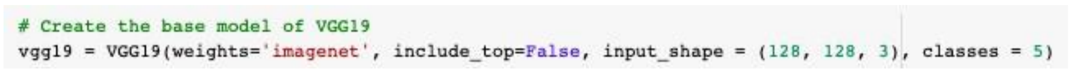
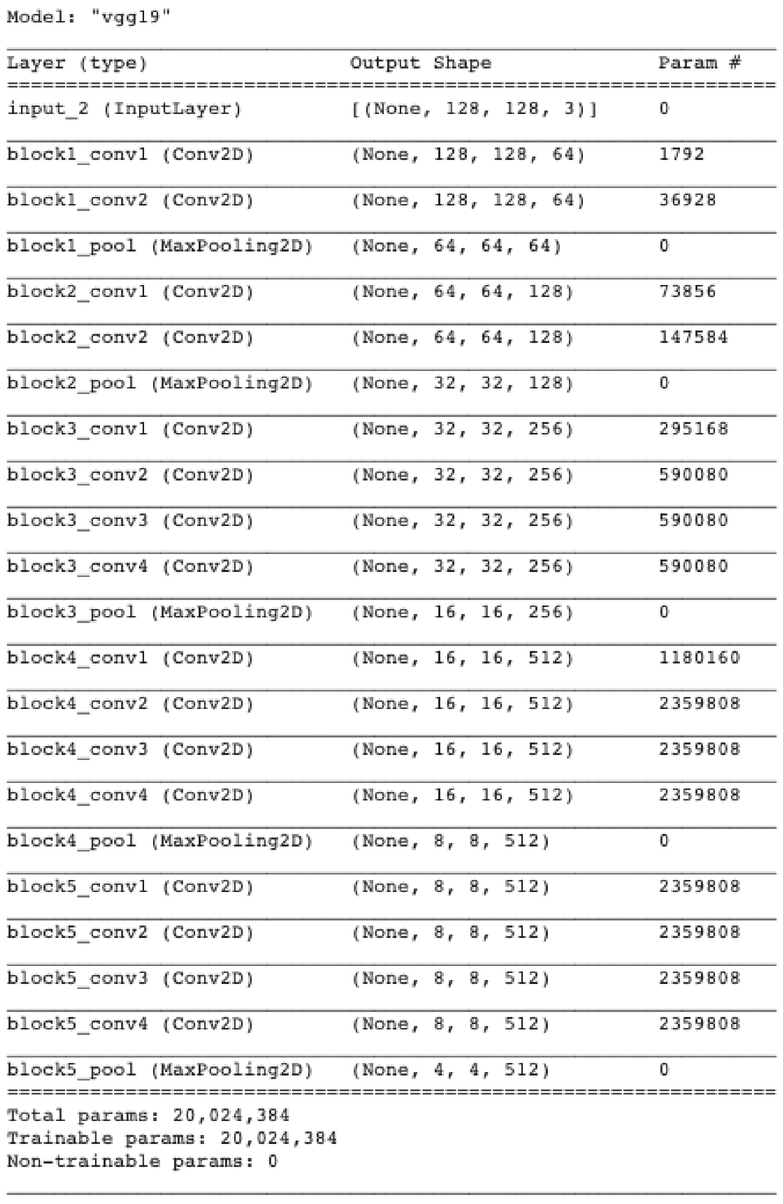

```
# Create the base model of VGG19
vgg19 = VGG19(weights='imagenet', include_top=False, input_shape = (128, 128, 3), classes = 5)
```

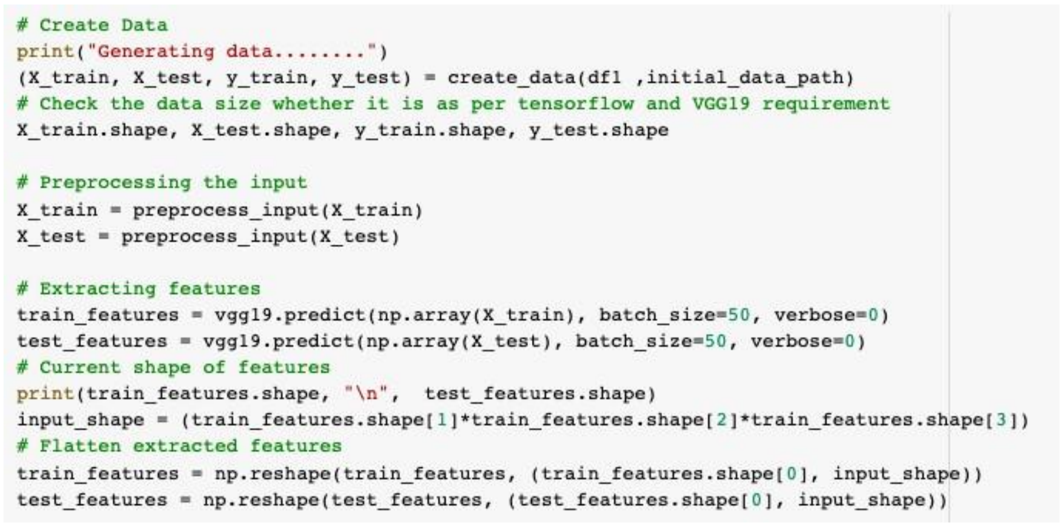
Figure 19: VGG-19 Function call

In Figure 21 the code for converting the input train and test array into features according to the format required by VGG-19.



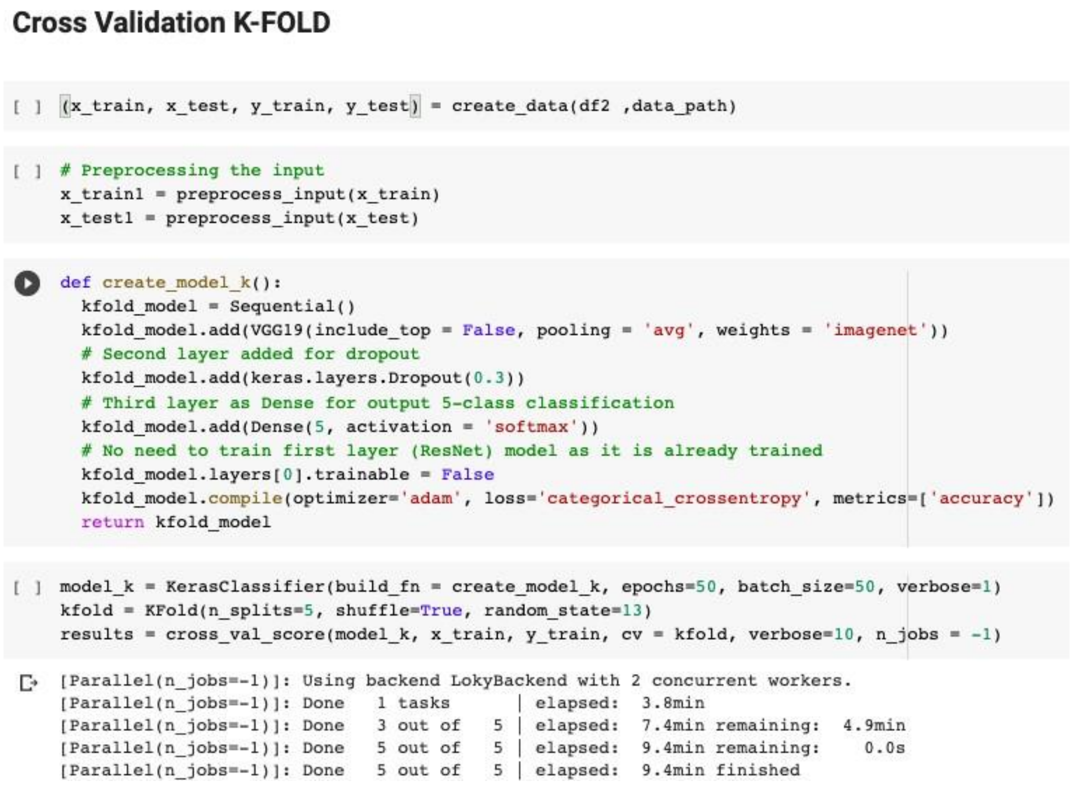
```
Model: "vgg19"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
input_2 (InputLayer)         [(None, 128, 128, 3)]     0
_________________________________________________________________
block1_conv1 (Conv2D)        (None, 128, 128, 64)      1792
_________________________________________________________________
block1_conv2 (Conv2D)        (None, 128, 128, 64)      36928
_________________________________________________________________
block1_pool (MaxPooling2D)   (None, 64, 64, 64)        0
_________________________________________________________________
block2_conv1 (Conv2D)        (None, 64, 64, 128)       73856
_________________________________________________________________
block2_conv2 (Conv2D)        (None, 64, 64, 128)       147584
_________________________________________________________________
block2_pool (MaxPooling2D)   (None, 32, 32, 128)       0
_________________________________________________________________
block3_conv1 (Conv2D)        (None, 32, 32, 256)       295168
_________________________________________________________________
block3_conv2 (Conv2D)        (None, 32, 32, 256)       590080
_________________________________________________________________
block3_conv3 (Conv2D)        (None, 32, 32, 256)       590080
_________________________________________________________________
block3_conv4 (Conv2D)        (None, 32, 32, 256)       590080
_________________________________________________________________
block3_pool (MaxPooling2D)   (None, 16, 16, 256)       0
_________________________________________________________________
block4_conv1 (Conv2D)        (None, 16, 16, 512)       1180160
_________________________________________________________________
block4_conv2 (Conv2D)        (None, 16, 16, 512)       2359808
_________________________________________________________________
block4_conv3 (Conv2D)        (None, 16, 16, 512)       2359808
_________________________________________________________________
block4_conv4 (Conv2D)        (None, 16, 16, 512)       2359808
_________________________________________________________________
block4_pool (MaxPooling2D)   (None, 8, 8, 512)         0
_________________________________________________________________
block5_conv1 (Conv2D)        (None, 8, 8, 512)         2359808
_________________________________________________________________
block5_conv2 (Conv2D)        (None, 8, 8, 512)         2359808
_________________________________________________________________
block5_conv3 (Conv2D)        (None, 8, 8, 512)         2359808
_________________________________________________________________
block5_conv4 (Conv2D)        (None, 8, 8, 512)         2359808
_________________________________________________________________
block5_pool (MaxPooling2D)   (None, 4, 4, 512)         0
=================================================================
Total params: 20,024,384
Trainable params: 20,024,384
Non-trainable params: 0
_________________________________________________________________
```

Figure 20: Trainable and Non-trainable parameters of VGG-19 Model



```
# Create Data
print("Generating data........")
(X_train, X_test, y_train, y_test) = create_data(df1 ,initial_data_path)
# Check the data size whether it is as per tensorflow and VGG19 requirement
X_train.shape, X_test.shape, y_train.shape, y_test.shape

# Preprocessing the input
X_train = preprocess_input(X_train)
X_test = preprocess_input(X_test)

# Extracting features
train_features = vgg19.predict(np.array(X_train), batch_size=50, verbose=0)
test_features = vgg19.predict(np.array(X_test), batch_size=50, verbose=0)
# Current shape of features
print(train_features.shape, "\n",  test_features.shape)
input_shape = (train_features.shape[1]*train_features.shape[2]*train_features.shape[3])
# Flatten extracted features
train_features = np.reshape(train_features, (train_features.shape[0], input_shape))
test_features = np.reshape(test_features, (test_features.shape[0], input_shape))
```

Figure 21: Converting to VGG-19 input features



Cross Validation K-FOLD

```
[ ] (x_train, x_test, y_train, y_test) = create_data(df2 ,data_path)
```

```
[ ] # Preprocessing the input
    x_train1 = preprocess_input(x_train)
    x_test1 = preprocess_input(x_test)
```

```
def create_model_k():
  kfold_model = Sequential()
  kfold_model.add(VGG19(include_top = False, pooling = 'avg', weights = 'imagenet'))
  # Second layer added for dropout
  kfold_model.add(keras.layers.Dropout(0.3))
  # Third layer as Dense for output 5-class classification
  kfold_model.add(Dense(5, activation = 'softmax'))
  # No need to train first layer (ResNet) model as it is already trained
  kfold_model.layers[0].trainable = False
  kfold_model.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['accuracy'])
  return kfold_model
```

```
[ ] model_k = KerasClassifier(build_fn = create_model_k, epochs=50, batch_size=50, verbose=1)
    kfold = KFold(n_splits=5, shuffle=True, random_state=13)
    results = cross_val_score(model_k, x_train, y_train, cv = kfold, verbose=10, n_jobs = -1)
```

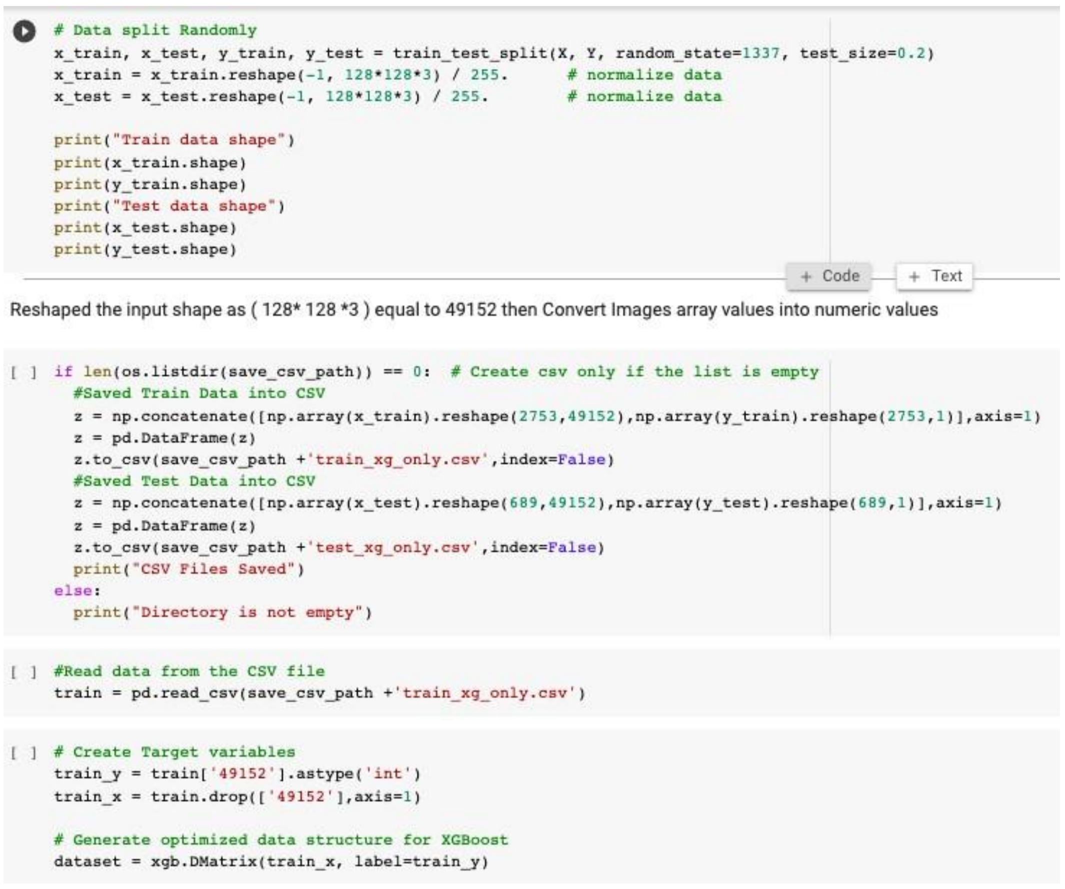
```
[Parallel(n_jobs=-1)]: Using backend LokyBackend with 2 concurrent workers.
[Parallel(n_jobs=-1)]: Done   1 tasks      | elapsed:  3.8min
[Parallel(n_jobs=-1)]: Done   3 out of   5 | elapsed:  7.4min remaining:  4.9min
[Parallel(n_jobs=-1)]: Done   5 out of   5 | elapsed:  9.4min remaining:    0.0s
[Parallel(n_jobs=-1)]: Done   5 out of   5 | elapsed:  9.4min finished
```

Figure 22: Cross Validation parameters of VGG-19 Model

4.4 Experiments with XGBoost Model

Apart from the neural network, the XGBoost classifier has also implemented in which data has first converted into the array and then saved as numeric input in the CSV file. The number of columns converted according to the input array shape (128*128*3) and converted into that much of columns (49152). Log loss has been calculating while applying cross-validation of xgboost as shown in Figure 26. The code for xgboost cross validation methods has been referenced from official python API[6].

[6]https://xgboost.readthedocs.io/en/latest/python/python$_a$*pi.html*



```
# Data split Randomly
x_train, x_test, y_train, y_test = train_test_split(X, Y, random_state=1337, test_size=0.2)
x_train = x_train.reshape(-1, 128*128*3) / 255.      # normalize data
x_test = x_test.reshape(-1, 128*128*3) / 255.        # normalize data

print("Train data shape")
print(x_train.shape)
print(y_train.shape)
print("Test data shape")
print(x_test.shape)
print(y_test.shape)
```

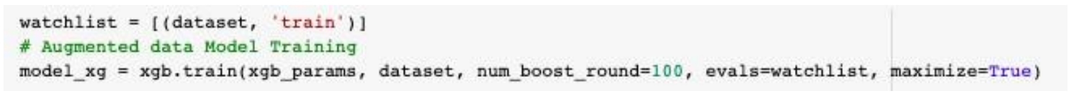
Reshaped the input shape as (128* 128 *3) equal to 49152 then Convert Images array values into numeric values

```
[ ] if len(os.listdir(save_csv_path)) == 0:  # Create csv only if the list is empty
      #Saved Train Data into CSV
      z = np.concatenate([np.array(x_train).reshape(2753,49152),np.array(y_train).reshape(2753,1)],axis=1)
      z = pd.DataFrame(z)
      z.to_csv(save_csv_path +'train_xg_only.csv',index=False)
      #Saved Test Data into CSV
      z = np.concatenate([np.array(x_test).reshape(689,49152),np.array(y_test).reshape(689,1)],axis=1)
      z = pd.DataFrame(z)
      z.to_csv(save_csv_path +'test_xg_only.csv',index=False)
      print("CSV Files Saved")
    else:
      print("Directory is not empty")
```

```
[ ] #Read data from the CSV file
    train = pd.read_csv(save_csv_path +'train_xg_only.csv')
```

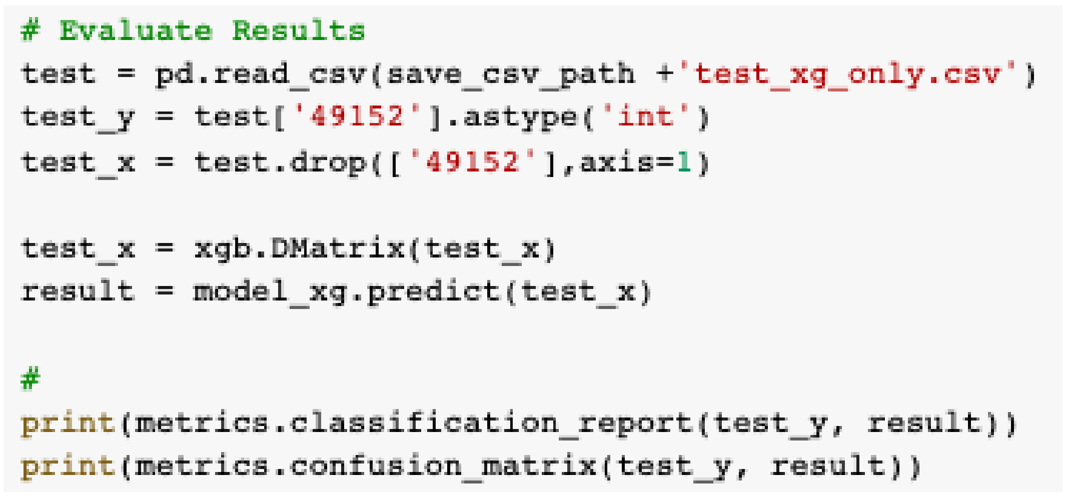
```
[ ] # Create Target variables
    train_y = train['49152'].astype('int')
    train_x = train.drop(['49152'],axis=1)

    # Generate optimized data structure for XGBoost
    dataset = xgb.DMatrix(train_x, label=train_y)
```

Figure 23: Data conversion to csv before fetching for the model training. DMatrix optimization method applied on input dataset



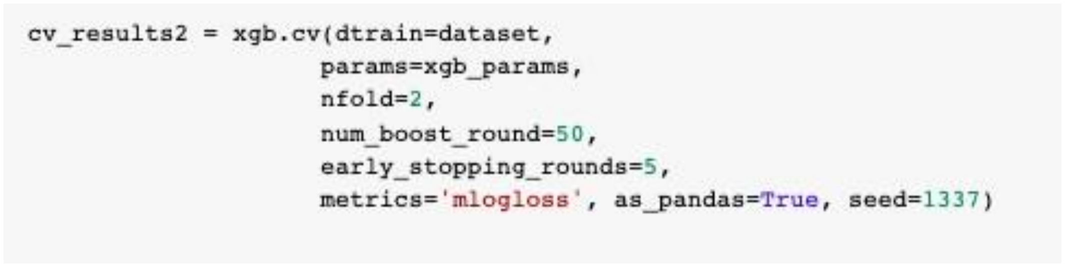
```
watchlist = [(dataset, 'train')]
# Augmented data Model Training
model_xg = xgb.train(xgb_params, dataset, num_boost_round=100, evals=watchlist, maximize=True)
```

Figure 24: XGBoost Model training function call



```
# Evaluate Results
test = pd.read_csv(save_csv_path +'test_xg_only.csv')
test_y = test['49152'].astype('int')
test_x = test.drop(['49152'],axis=1)

test_x = xgb.DMatrix(test_x)
result = model_xg.predict(test_x)

#
print(metrics.classification_report(test_y, result))
print(metrics.confusion_matrix(test_y, result))
```

Figure 25: XGBoost Model Evaluation on unseen data for obtaining the classification report



```
cv_results2 = xgb.cv(dtrain=dataset,
                     params=xgb_params,
                     nfold=2,
                     num_boost_round=50,
                     early_stopping_rounds=5,
                     metrics='mlogloss', as_pandas=True, seed=1337)
```

Figure 26: Log loss function calculation using xgboost cross validation method