Configuration Manual

MSc Research Project
Data Analytics

Prithvi Mysore Dayananda
Student ID: x19242204

School of Computing
National College of Ireland

Supervisor: Dr. Christian Horn

National College of Ireland
Project Submission Sheet
School of Computing

Student Name:	Prithvi Mysore Dayananda
Student ID:	x19242204
Programme:	Data Analytics
Year:	2021
Module:	MSc Research Project
Supervisor:	Dr. Christian Horn
Submission Due Date:	16/12/2021
Project Title:	Configuration Manual
Word Count:	1075
Page Count:	10

I hereby certify that the information contained in this (my submission) is information pertaining to research I conducted for this project. All information other than my own contribution will be fully referenced and listed in the relevant bibliography section at the rear of the project.

ALL internet material must be referenced in the bibliography section. Students are required to use the Referencing Standard specified in the report template. To use other author's written or electronic work is illegal (plagiarism) and may result in disciplinary action.

Signature:	
Date:	16th December 2021

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.	☐

Assignments that are submitted to the Programme Coordinator office must be placed into the assignment box located outside the office.

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

Configuration Manual

Prithvi Mysore Dayananda
x19242204

1 Introduction

This configuration manual is an overview and presents the hardware, software requirements, design details, implementation details, and settings of the projection detail: "A Comparative Analysis for Trash image classification using Deep Learning."

2 System Configuration

2.1 Hardware

- Processor:Intel(R) Core(TM) i7-10510U CPU @ 1.80GHz 2.30 GHz
- RAM: 8.00 GB
- System Type: Windows OS, 64-bit
- GPU: Intel(R) UHD Graphics Family, 8GB
- GPUStorage: 1 TB HDD

2.2 Software

- Jupyter Notebook (Version 6.0.3): The Jupyter Notebook is an open-source web application that allows data scientists to create and share documents that integrate live code, equations, computational output, visualizations, and other multimedia resources, along with explanatory text in a single document.
- Python (Version 3.8.3): Python is a computer programming language often used to build websites and software, automate tasks, and conduct data analysis. Python is a general purpose language, meaning it can be used to create a variety of different programs and isn't specialized for any specific problems.
- Excel: A spreadsheet program offered by Microsoft is used for visualization of data, plots, table formation.
- Tableau: Tableau Software is a tool that helps make Big Data small, and small data insightful and actionable. The main use of tableau software is to help people see and understand their data.

3 Walkthorugh of zipped Artecraft

The artecraft folder can be divided into two parts i.e, Data and Python files. A seperate file for each model has been created. VGG16, ResNet50 and Custom MLH-CNN model from Shi et al. (2021) can be seen. While on data there are two folders Garbage_classification and splitdata whch will be discussed further. Figure 1

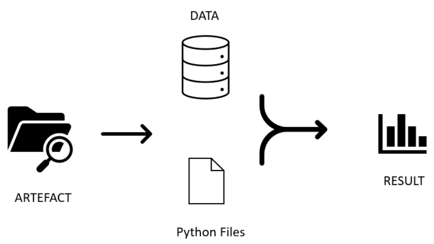


Figure 1: Explaination of zipped artefact folder

Since the data set is huge the link for the same is provided in the dataset pdf. This can be directly downloaded and uncomment the splitfolder code to divide correspondingly.

4 Data Exploration

4.1 Data Acquisition

Garbage classification Data has been obtained from public platform called Kaggle. The total number of images is 15515 which consists of 12 different categories as shown in Figure 2.

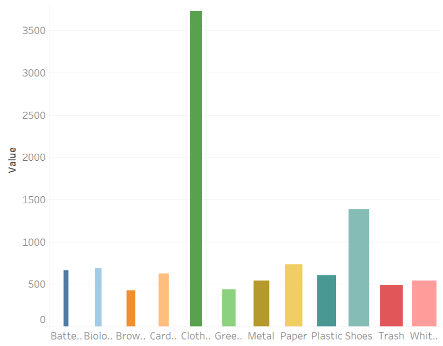


Figure 2: Image count of each category

4.2 Data Preprocessing and Augmentation

Before we begin preprocessing our data, it is necessary for us to import necessary libraries to perform our required action in python. Figure 3 lists all the libraries required.

```
In [1]:
import matplotlib.pyplot as plt
from keras.preprocessing.image import ImageDataGenerator
from tensorflow.keras.optimizers import Adam
from keras.preprocessing.image import img_to_array
from tensorflow.keras.utils import to_categorical
from tensorflow.keras import optimizers
from tensorflow.keras import models
from tensorflow.keras import layers
import numpy as np
import pandas as pd
import random
import cv2
import os
from PIL import Image
from glob import glob
from tensorflow.keras.layers import *
from tensorflow.keras.models import *
import keras
import tensorflow as tf
```

Figure 3: Required python libraries

Once, the libraries are successfully loaded, we load the dataset downloaded from the garbage_classification folder. Once it is loaded, we see that it is not split into training, testing, and validation folders. Hence, we use python's splitfolder to split the data respectively as seen in Figure 4. This is now stored in a folder called splitdata which has already been created in the same directory.

```
In [3]:
#uncomment if you wish to split data, since a splitted folder has been packed in artecraft already under splitdata folder

import splitfolders

inputfolder = os.getcwd()
splitfolders.ratio(inputfolder, output = os.getcwd(),"splitdata\\", seed = 42, ratio = (.7, .2,.1), group_prefix = None)
```

Figure 4: Splitting Dataset

Now that we have our data, since all the images are of different shapes, we shape the input into 64*64 as suggested from the paper Shi et al. (2021). It is necessary that we rescale the values from 0 to 1 to better understanding for the machine. The Figure 5 shows us how. It also gives us the code implemented to perform data augmentation using

```
In [8]:
# re-size all the images to this
IMAGE_SIZE = [64, 64]

# training config:
epochs = 10 #can be set to a huge number for better perfromance
batch_size = 32

In [9]:
#Performing Data Augmentation, with scaling all images

train_gen = 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
)

val_gen = ImageDataGenerator(
    rescale = 1./255
)

test_gen = ImageDataGenerator(
    rescale = 1./255
)
```

Figure 5: Preprocessing images

the Image Generator package which dynamically creates images while training on the go for the provided parameters. We see that rotation, the sheer range of zoom, horizontal flip, and many other operations performed.

Figure 6 shows the conversion of the labels from images category into one hot encoding way and shuffling has been done before training will be done so that the randomness will have a positive impact on model and it learns on all different patterns.

In [10]:
```python
# create generators where all labels are set to categroical i.e, one hot encoded

train_generator = train_gen.flow_from_directory(
  train_dir,
  target_size=IMAGE_SIZE,
  shuffle=True,
  batch_size=batch_size,
  class_mode='categorical'
)
valid_generator = val_gen.flow_from_directory(
  validation_dir,
  target_size=IMAGE_SIZE,
  shuffle=True,
  batch_size=batch_size,
  class_mode='categorical'
)
```

Figure 6: Performing shuffling and Data Augmentation for Training

5 Modelling

5.1 Custom MLH-CNN Model

As we are perfroming a comparitive analysis on exiting model propsed by Shi et al. (2021), the same architecture has been implemnted as we see from Figure 7.

```python
from tensorflow.keras.utils import Sequence
input_shape = [64,64,3]
imodel = Sequential()
imodel.add(Conv2D(32, kernel_size=(3, 3), strides=1, activation='relu', input_shape=input_shape, padding = "same"))
imodel.add(BatchNormalization(momentum = 0.9))
imodel.add(Conv2D(32, kernel_size=(3, 3), strides=1, activation='relu', padding = "same"))
imodel.add(BatchNormalization(momentum = 0.9))
imodel.add(MaxPooling2D(pool_size=(2, 2), strides=(2,2))) #32 channels

imodel.add(Conv2D(64, kernel_size=(3, 3), strides=1, activation='relu', padding = "same"))
imodel.add(BatchNormalization())
imodel.add(Conv2D(64, kernel_size=(3, 3), strides=1, activation='relu', padding = "same"))
imodel.add(BatchNormalization(momentum = 0.9))
imodel.add(MaxPooling2D(pool_size=(2, 2), strides=(2,2))) #63 channels

imodel.add(Conv2D(128, kernel_size=(3, 3), strides=1, activation='relu', padding = "same"))
imodel.add(BatchNormalization())
imodel.add(Conv2D(128, kernel_size=(3, 3), strides=1, activation='relu', padding = "same"))
imodel.add(BatchNormalization())
imodel.add(MaxPooling2D(pool_size=(2, 2), strides=(2,2))) #128 channels

imodel.add(Conv2D(256, kernel_size=(3, 3), strides=1, activation='relu', padding = "same"))
imodel.add(BatchNormalization())
imodel.add(Conv2D(256, kernel_size=(3, 3), activation='relu'))
imodel.add(BatchNormalization())
imodel.add(MaxPooling2D(pool_size=(2, 2), strides=(2,2))) #256 channels

imodel.add(Flatten())
imodel.add(Dense(128, activation='relu'))
imodel.add(Dense(64, activation='relu'))

imodel.add(Dense(len(folders), activation='softmax'))
```

Figure 7: Custom MLH-CNN model Architecture

The Figure 8 shows the optimizer and callback set for the model which runs for 10

In [13]:
```python
from tensorflow.keras import optimizers
imodel.compile(
    loss='categorical_crossentropy',
    optimizer = keras.optimizers.Adam(learning_rate=0.01),
    metrics=['accuracy']
)
```

In [14]:
```python
#Setting Early stopping

from tensorflow.keras.callbacks import EarlyStopping, CSVLogger, ModelCheckpoint
path = "Savedmodels/model.hdf5"
checkpoint = ModelCheckpoint(path, monitor='accuracy', verbose=1, save_best_only=True, mode='max')
early_stop = EarlyStopping(monitor='loss', patience = 30, verbose=1)
log_csv = CSVLogger('mylogs.csv', separator= ',', append= False)
callbacklist=[checkpoint, early_stop, log_csv]
```

In [15]:
```python
# fit the model

import time
start_time = time.time()

checkpoint_filepath = os.path.join(os.getcwd(), "check")
r = imodel.fit(
  train_generator,
  validation_data=valid_generator,
  epochs=epochs,
  steps_per_epoch=len(image_files) // batch_size,
  validation_steps=len(valid_image_files) // batch_size,
  callbacks=callbacklist
)
print("The model took --- %s seconds ---" % (time.time() - start_time))
```

```
Epoch 1/10
339/339 [==============================] - 267s 785ms/step - loss: 2.1464 - accuracy: 0.3489 - val_loss: 1.9810 - val_accura
cy: 0.3877

Epoch 00001: accuracy improved from -inf to 0.34892, saving model to Savedmodels\model.hdf5
Epoch 2/10
339/339 [==============================] - 252s 743ms/step - loss: 1.7577 - accuracy: 0.4152 - val_loss: 2.0735 - val_accura
cy: 0.3733
```

Figure 8: Callback and optimizer setting

epochs where the model accuracy can be seen per epoch in Figure 9, we can see an accuracy of 55.75%.

5.2 ResNet50

The data preparation for all the models implemeted is same except the input size of the images is of 224*224, hence the model strucutre has been presented in Figure **??** and Figure 11 gives us the ResNet50 structure built from scratch.

```
Epoch 1/10
339/339 [==============================] - 267s 785ms/step - loss: 2.1464 - accuracy: 0.3489 - val_loss: 1.9810 - val_accur
acy: 0.3877

Epoch 00001: accuracy improved from -inf to 0.34892, saving model to Savedmodels\model.hdf5
Epoch 2/10
339/339 [==============================] - 252s 743ms/step - loss: 1.7577 - accuracy: 0.4152 - val_loss: 2.0735 - val_accur
acy: 0.3733

Epoch 00002: accuracy improved from 0.34892 to 0.41517, saving model to Savedmodels\model.hdf5
Epoch 3/10
339/339 [==============================] - 261s 771ms/step - loss: 1.6598 - accuracy: 0.4427 - val_loss: 1.5372 - val_accur
acy: 0.4707

Epoch 00003: accuracy improved from 0.41517 to 0.44271, saving model to Savedmodels\model.hdf5
Epoch 4/10
339/339 [==============================] - 243s 718ms/step - loss: 1.5723 - accuracy: 0.4681 - val_loss: 1.4130 - val_accur
acy: 0.5081

Epoch 00004: accuracy improved from 0.44271 to 0.46812, saving model to Savedmodels\model.hdf5
Epoch 5/10
339/339 [==============================] - 255s 753ms/step - loss: 1.5067 - accuracy: 0.4934 - val_loss: 1.5364 - val_accur
acy: 0.4902

Epoch 00005: accuracy improved from 0.46812 to 0.49344, saving model to Savedmodels\model.hdf5
Epoch 6/10
339/339 [==============================] - 273s 805ms/step - loss: 1.4501 - accuracy: 0.5150 - val_loss: 1.4246 - val_accur
acy: 0.5026

Epoch 00006: accuracy improved from 0.49344 to 0.51497, saving model to Savedmodels\model.hdf5
Epoch 7/10
339/339 [==============================] - 261s 769ms/step - loss: 1.4220 - accuracy: 0.5246 - val_loss: 1.4535 - val_accur
acy: 0.5212

Epoch 00007: accuracy improved from 0.51497 to 0.52458, saving model to Savedmodels\model.hdf5
Epoch 8/10
339/339 [==============================] - 280s 826ms/step - loss: 1.3751 - accuracy: 0.5369 - val_loss: 1.3562 - val_accur
acy: 0.5517

Epoch 00008: accuracy improved from 0.52458 to 0.53687, saving model to Savedmodels\model.hdf5
Epoch 9/10
339/339 [==============================] - 275s 811ms/step - loss: 1.3443 - accuracy: 0.5539 - val_loss: 1.3306 - val_accur
acy: 0.5563

Epoch 00009: accuracy improved from 0.53687 to 0.55387, saving model to Savedmodels\model.hdf5
Epoch 10/10
339/339 [==============================] - 282s 832ms/step - loss: 1.3185 - accuracy: 0.5571 - val_loss: 1.2074 - val_accur
acy: 0.5757

Epoch 00010: accuracy improved from 0.55387 to 0.55711, saving model to Savedmodels\model.hdf5
The model took --- 2650.4514832496643 seconds ---
```

Figure 9: Training of MLH-CNN model

In [11]:

```python
#Building ResNet50 model architecture from scratch:

def identity_block(input_, kernel_size, filters):
    f1, f2, f3 = filters

    x = Conv2D(f1, (1, 1),
               kernel_initializer='he_normal'
    )(input_)
    x = BatchNormalization()(x)
    x = Activation('relu')(x)

    x = Conv2D(f2, kernel_size, padding='same',
               kernel_initializer='he_normal'
    )(x)
    x = BatchNormalization()(x)
    x = Activation('relu')(x)

    x = Conv2D(f3, (1, 1),
               kernel_initializer='he_normal'
    )(x)
    x = BatchNormalization()(x)

    x = add([x, input_])
    x = Activation('relu')(x)
    return x
```

In [12]:

```python
def conv_block(input_,
               kernel_size,
               filters,
               strides=(2, 2)):
    f1, f2, f3 = filters

    x = Conv2D(f1, (1, 1), strides=strides,
               kernel_initializer='he_normal'
    )(input_)
    x = BatchNormalization()(x)
    x = Activation('relu')(x)

    x = Conv2D(f2, kernel_size, padding='same',
               kernel_initializer='he_normal'
    )(x)
    x = BatchNormalization()(x)
    x = Activation('relu')(x)

    x = Conv2D(f3, (1, 1),
               kernel_initializer='he_normal'
    )(x)
    x = BatchNormalization()(x)

    shortcut = Conv2D(f3, (1, 1), strides=strides,
                      kernel_initializer='he_normal'
    )(input_)
    shortcut = BatchNormalization()(shortcut)

    x = add([x, shortcut])
    x = Activation('relu')(x)
    return x
```

Figure 10: Architecture of ResNet50 model

In [13]:

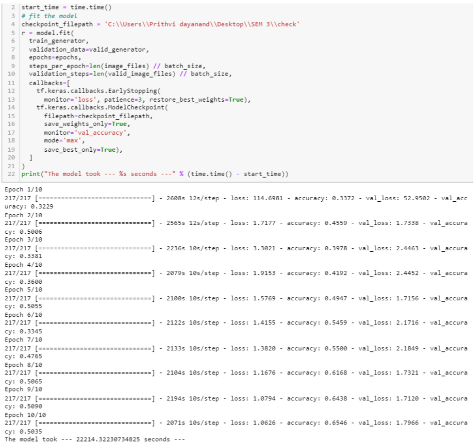
```python
# our custom resnet
i = Input(shape=IMAGE_SIZE + [3])
x = ZeroPadding2D(padding=(3, 3))(i)
x = Conv2D(64, (7, 7),
           strides=(2, 2),
           padding='valid',
           kernel_initializer='he_normal'
           )(x)
x = BatchNormalization()(x)
x = Activation('relu')(x)
x = ZeroPadding2D(padding=(1, 1))(x)
x = MaxPooling2D((3, 3), strides=(2, 2))(x)

x = conv_block(x, 3, [64, 64, 256], strides=(1, 1))
x = identity_block(x, 3, [64, 64, 256])
x = identity_block(x, 3, [64, 64, 256])

x = conv_block(x, 3, [128, 128, 512])
x = identity_block(x, 3, [128, 128, 512])
x = identity_block(x, 3, [128, 128, 512])
x = identity_block(x, 3, [128, 128, 512])
```

In [14]:

```python
# our layers
x = Flatten()(x)
# x = Dense(1000, activation='relu')(x)
prediction = Dense(len(folders), activation='softmax')(x)
```

Figure 11: Architecture of ResNet50 model

The model has been trained and ran for 10 epochs with a learning rate of 0.01 as we see from Figure 12



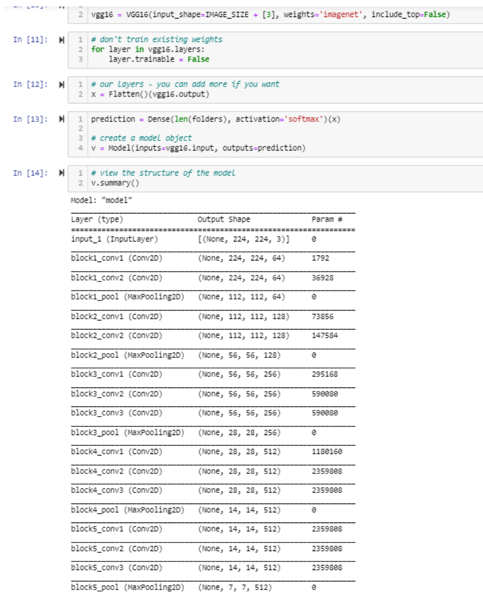
```
start_time = time.time()
# fit the model
checkpoint_filepath = 'C:\\Users\\Prithvi dayanand\\Desktop\\SEM 3\\check'
r = model.fit(
  train_generator,
  validation_data=valid_generator,
  epochs=epochs,
  steps_per_epoch=len(image_files) // batch_size,
  validation_steps=len(valid_image_files) // batch_size,
  callbacks=[
    tf.keras.callbacks.EarlyStopping(
      monitor='loss', patience=3, restore_best_weights=True),
    tf.keras.callbacks.ModelCheckpoint(
      filepath=checkpoint_filepath,
      save_weights_only=True,
      monitor='val_accuracy',
      mode='max',
      save_best_only=True),
  ]
)
print("The model took --- %s seconds ---" % (time.time() - start_time))
```

```
Epoch 1/10
217/217 [==============================] - 2608s 12s/step - loss: 114.6981 - accuracy: 0.3372 - val_loss: 52.9502 - val_acc
uracy: 0.3229
Epoch 2/10
217/217 [==============================] - 2565s 12s/step - loss: 1.7177 - accuracy: 0.4559 - val_loss: 1.7338 - val_accura
cy: 0.5006
Epoch 3/10
217/217 [==============================] - 2236s 10s/step - loss: 3.3021 - accuracy: 0.3978 - val_loss: 2.4463 - val_accura
cy: 0.3381
Epoch 4/10
217/217 [==============================] - 2079s 10s/step - loss: 1.9153 - accuracy: 0.4192 - val_loss: 2.4452 - val_accura
cy: 0.3600
Epoch 5/10
217/217 [==============================] - 2100s 10s/step - loss: 1.5769 - accuracy: 0.4947 - val_loss: 1.7156 - val_accura
cy: 0.5055
Epoch 6/10
217/217 [==============================] - 2122s 10s/step - loss: 1.4155 - accuracy: 0.5459 - val_loss: 2.1716 - val_accura
cy: 0.3345
Epoch 7/10
217/217 [==============================] - 2133s 10s/step - loss: 1.3820 - accuracy: 0.5500 - val_loss: 2.1849 - val_accura
cy: 0.4765
Epoch 8/10
217/217 [==============================] - 2104s 10s/step - loss: 1.1676 - accuracy: 0.6168 - val_loss: 1.7321 - val_accura
cy: 0.5065
Epoch 9/10
217/217 [==============================] - 2194s 10s/step - loss: 1.0794 - accuracy: 0.6438 - val_loss: 1.7120 - val_accura
cy: 0.5090
Epoch 10/10
217/217 [==============================] - 2071s 10s/step - loss: 1.0626 - accuracy: 0.6546 - val_loss: 1.7966 - val_accura
cy: 0.5035
The model took --- 22214.32230734825 seconds ---
```

Figure 12: Training of ResNet50 model

5.3 VGG16

The images are scaled for a dimension of 224*224 and the model is imported from keras in built application pretrained models. The last layer is freezed and a softmax layer of 12 diffent categories are implemented isntead of default 1000 categories. as we observe from the Figure **??**.



```
vgg16 = VGG16(input_shape=IMAGE_SIZE + [3], weights='imagenet', include_top=False)
```

In [11]:
```
# don't train existing weights
for layer in vgg16.layers:
    layer.trainable = False
```

In [12]:
```
# our layers - you can add more if you want
x = Flatten()(vgg16.output)
```

In [13]:
```
prediction = Dense(len(folders), activation='softmax')(x)

# create a model object
v = Model(inputs=vgg16.input, outputs=prediction)
```

In [14]:
```
# view the structure of the model
v.summary()
```

Model: "model"

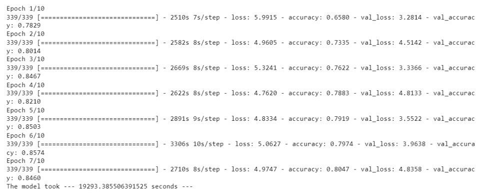
Layer (type)	Output Shape	Param #
input_1 (InputLayer)	[(None, 224, 224, 3)]	0
block1_conv1 (Conv2D)	(None, 224, 224, 64)	1792
block1_conv2 (Conv2D)	(None, 224, 224, 64)	36928
block1_pool (MaxPooling2D)	(None, 112, 112, 64)	0
block2_conv1 (Conv2D)	(None, 112, 112, 128)	73856
block2_conv2 (Conv2D)	(None, 112, 112, 128)	147584
block2_pool (MaxPooling2D)	(None, 56, 56, 128)	0
block3_conv1 (Conv2D)	(None, 56, 56, 256)	295168
block3_conv2 (Conv2D)	(None, 56, 56, 256)	590080
block3_conv3 (Conv2D)	(None, 56, 56, 256)	590080
block3_pool (MaxPooling2D)	(None, 28, 28, 256)	0
block4_conv1 (Conv2D)	(None, 28, 28, 512)	1180160
block4_conv2 (Conv2D)	(None, 28, 28, 512)	2359808
block4_conv3 (Conv2D)	(None, 28, 28, 512)	2359808
block4_pool (MaxPooling2D)	(None, 14, 14, 512)	0
block5_conv1 (Conv2D)	(None, 14, 14, 512)	2359808
block5_conv2 (Conv2D)	(None, 14, 14, 512)	2359808
block5_conv3 (Conv2D)	(None, 14, 14, 512)	2359808
block5_pool (MaxPooling2D)	(None, 7, 7, 512)	0

Figure 13: Training of VGG16 model

Training of 10 epochs has been done while it gives an accuracy of 80.47% Figure **??**

6 Evaluation

The model evaluation is done in terms of accuracy, confusion matrix, precision, and recall values. The measures of all models are mentioned as followed.



```
Epoch 1/10
339/339 [==============================] - 2510s 7s/step - loss: 5.9915 - accuracy: 0.6580 - val_loss: 3.2814 - val_accurac
y: 0.7829
Epoch 2/10
339/339 [==============================] - 2582s 8s/step - loss: 4.9605 - accuracy: 0.7335 - val_loss: 4.5142 - val_accurac
y: 0.8014
Epoch 3/10
339/339 [==============================] - 2669s 8s/step - loss: 5.3241 - accuracy: 0.7622 - val_loss: 3.3366 - val_accurac
y: 0.8467
Epoch 4/10
339/339 [==============================] - 2622s 8s/step - loss: 4.7620 - accuracy: 0.7883 - val_loss: 4.8133 - val_accurac
y: 0.8210
Epoch 5/10
339/339 [==============================] - 2891s 9s/step - loss: 4.8334 - accuracy: 0.7919 - val_loss: 3.5522 - val_accurac
y: 0.8503
Epoch 6/10
339/339 [==============================] - 3306s 10s/step - loss: 5.0627 - accuracy: 0.7974 - val_loss: 3.9638 - val_accura
cy: 0.8574
Epoch 7/10
339/339 [==============================] - 2710s 8s/step - loss: 4.9747 - accuracy: 0.8047 - val_loss: 4.8358 - val_accurac
y: 0.8460
The model took --- 19293.385506391525 seconds ---
```

Figure 14: Training of VGG16 model

6.1 Evaluation of Custom MLH-CNN Model

Figure 24 gives us accuracy change across each epoch and we see a rise of accuracy from the 3rd epoch and a pretty constant and small improvement can be seen further. Figure 17 presents the loss and the validation accuracy is higher which presents the goodness of the model than the rest of the model.

Figure 15: Loss of MLH-CNN model

Figure 16: Accuracy of MLH-CNN model

Figure 17 bring out the confusion matrix and Figure 18 gives the us the overall precision and recall of the model implemented.

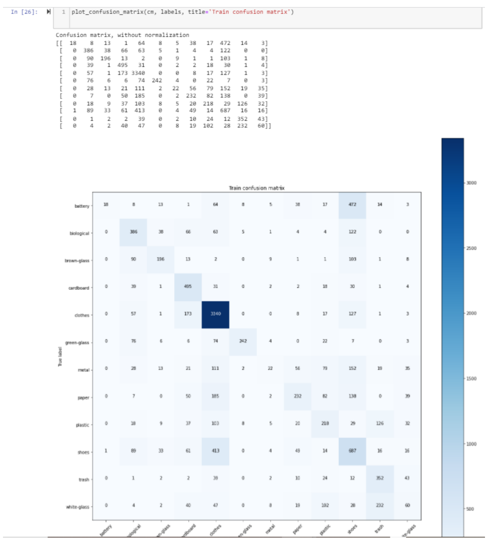


Figure 17: Confusion Matrix of MLH-CNN model

```
In [30]:
print("OverAll Recall ---",np.mean(recall))
print("OverAll Precision ---",np.mean(precision))

OverAll Recall --- 0.444795204839397
OverAll Precision --- 0.5440140543162063
```

Figure 18: Precision and Recall of MLH-CNN model

6.2 Evaluation of ResNet50 model

Figure 20 gives us accuracy change across each epoch and we see a rise of accuracy from the 3rd epoch and a pretty constant and small improvement can be seen further. We can see that this is the highest of all the models Figure 19 presents the loss and the validation accuracy is higher which presents the goodness of the model than the rest of the model.

```
In [23]:
# loss
plt.plot(r.history['loss'], label='train loss')
plt.plot(r.history['val_loss'], label='val loss')
plt.legend()
plt.show()
```

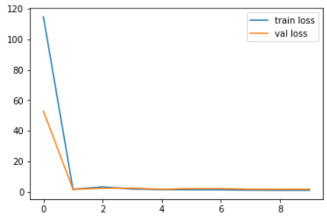


Figure 19: Loss of ResNet50 model

```
In [19]:
# accuracy
plt.plot(r.history['accuracy'], label='train acc')
plt.plot(r.history['val_accuracy'], label='val acc')
plt.legend()
plt.show()
```

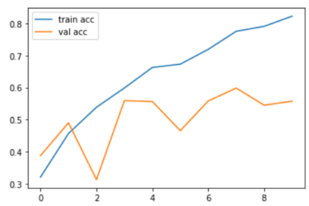


Figure 20: Accuracy of ResNet50 model

Figure 21 and Figure 22 represents the code implemented to bring out the confusion matrix and precision and recall.

```
In [34]:
plot_confusion_matrix(cm, labels, title='Train confusion matrix')

Confusion matrix, without normalization
[[ 628    0    2    0   10    1    1    3    1   14    1    0]
 [  13  453   52   11   97    3    4    8    1   41    2    4]
 [   2    2  398    2    4    0    3    1    2    9    1    0]
 [  10    8    5  467   47    0    7   12    4   34   24    5]
 [  15    2    5    2 3645    6    1   20    1   30    0    0]
 [  24    1    4    1    5  388    2    2    2    9    1    1]
 [  22    1    7    8   38    2  392   13    6   33   11    5]
 [   6    0    1   12   28    0    2  657   15   11    0    3]
 [  17    0    7   12   56    2   14   23  407   14   31   22]
 [  51   11   23    6  163    0    3   10    3 1101   10    2]
 [  22    0    1    3   27    0    7    7    4   13  390   13]
 [  11    0    5   13   84    0   12   14   25   15   40  323]]
```

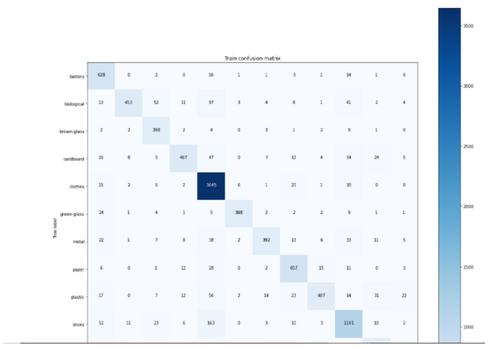


Figure 21: Code for confusion matrix ResNet50 model

```
In [35]:
recall = np.diag(cm) / np.sum(cm, axis = 1)
precision = np.diag(cm) / np.sum(cm, axis = 0)

In [36]:
recall

Out[36]: array([0.95007564, 0.6574746 , 0.93867925, 0.74959872, 0.97799839,
       0.88181818, 0.72862454, 0.89387755, 0.67272727, 0.79609544,
       0.80082136, 0.59594096])

In [37]:
precision

Out[37]: array([0.76492083, 0.94769874, 0.78039216, 0.86964618, 0.8670314 ,
       0.96517413, 0.875     , 0.85324675, 0.8641189 , 0.831571  ,
       0.76320939, 0.85449735])

In [38]:
print("OverAll Recall ---",np.mean(recall))
print("OverAll Precision ---",np.mean(precision))

OverAll Recall --- 0.8036443246124213
OverAll Precision --- 0.8530422362035602
```

Figure 22: Precision and Recall of ResNet50 model

6.3 Evaluation of VGG16 model

Figure 24 gives us accuracy change across each epoch and we see a rise of accuracy from the 3rd epoch and a pretty constant and small improvement can be seen further. Figure 23 presents the loss and the validation accuracy is higher which presents the goodness of the model than the rest of the model.

```
In [21]:
# loss
plt.plot(r.history['loss'], label='train loss')
plt.plot(r.history['val_loss'], label='validation loss')
plt.legend()
plt.show()
```

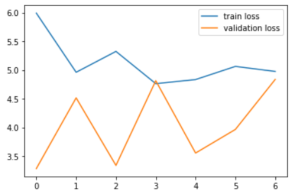


Figure 23: Loss of VGG16 model

```
In [22]:
plt.plot(r.history['accuracy'], label='train accuracy')
plt.plot(r.history['val_accuracy'], label='validation accuracy')
plt.legend()
plt.show()
```

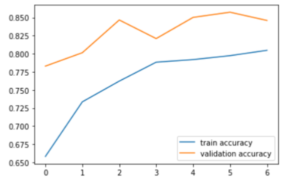


Figure 24: Accuracy of VGG16 model

Figure 25 and Figure 26 represents the code implemented to bring out the confusion matrix.

```
In [25]:
def get_confusion_matrix(data_path, N):
  # we need to see the data in the same order
  # for both predictions and targets
    print("Generating confusion matrix", N)
    predictions = []
    targets = []
    i = 0
    n_images = 0
    for x, y in test_gen.flow_from_directory(
        data_path,
        target_size=IMAGE_SIZE,
        shuffle=False,
        batch_size=batch_size * 2):
        i += 1
        n_images += len(y)
        if i % 50 == 0:
            print(f'{n_images} images processed.')
        p = v.predict(x)
        p = np.argmax(p, axis=1)
        y = np.argmax(y, axis=1)
        predictions = np.concatenate((predictions, p))
        targets = np.concatenate((targets, y))
        if len(targets) >= N:
            break

    cm = confusion_matrix(targets, predictions)
    return cm

In [34]:
from sklearn.metrics import confusion_matrix
cm = get_confusion_matrix(train_dir, len(image_files))

In [27]:
test_cm = get_confusion_matrix(test_dir, len(test_image_files))

Generating confusion matrix 1561
Found 1561 images belonging to 12 classes.
```

Figure 25: Code for confusion matrix VGG16 model

```
In [28]:
def plot_confusion_matrix(cm, classes,
                          normalize=False,
                          title='Confusion matrix',
                          cmap=plt.cm.Blues):
  """
  This function prints and plots the confusion matrix.
  Normalization can be applied by setting `normalize=True`.
  """
  if normalize:
      cm = cm.astype('float') / cm.sum(axis=1)[:, np.newaxis]
      print("Normalized confusion matrix")
  else:
      print('Confusion matrix, without normalization')

  print(cm)

  plt.figure(figsize=(15, 15))
  plt.imshow(cm, interpolation='nearest', cmap=cmap)
  plt.title(title)
  plt.colorbar()
  tick_marks = np.arange(len(classes))
  plt.xticks(tick_marks, classes, rotation=45)
  plt.yticks(tick_marks, classes)

  fmt = '.2f' if normalize else 'd'
  thresh = cm.max() / 2.
  for i in range (cm.shape[0]):
    for j in range (cm.shape[1]):
      plt.text(j, i, format(cm[i, j], fmt),
               horizontalalignment="center",
               color="white" if cm[i, j] > thresh else "black")

  plt.tight_layout()
  plt.ylabel('True label')
  plt.xlabel('Predicted label')
  plt.show()

In [31]:
plot_confusion_matrix(cm, labels, title='Train confusion matrix')
```

Figure 26: Confusion Matrix of VGG16 model

Figure 27 gives the us the overall precision and recall of the model implemented.

```
In [32]:
recall = np.diag(cm) / np.sum(cm, axis = 1)
precision = np.diag(cm) / np.sum(cm, axis = 0)

In [35]:
print("OverAll Recall ---",np.mean(recall))
print("OverAll Precision ---",np.mean(precision))

OverAll Recall --- 0.7894639933866543
OverAll Precision --- 0.8327940994085766
```

Figure 27: Precision and Recall of VGG16 model

References

Shi, C., Tan, C., Wang, T. and Wang, L. (2021). A waste classification method based on a multilayer hybrid convolution neural network, *Applied Sciences* **11**(18).
URL: *https://www.mdpi.com/2076-3417/11/18/8572*