**Beggar v/s No-Beggar Classification**

**1)Data Preparation**

* Number of beggar images**- 430**

* Number of non-beggar images-**407**

**Image pre-processing and data augmentation-**

Flipping, rotations, sharpness variation, brightness variations has been done one the image set for augmentation and the resulted number of images are-

* Number of beggar images after augmentation**- 11777**
* Number of non-beggar images after augmentation -**17088**
* Image size- (150\*150) pixels



The code of image augmentation is in appendix 1.

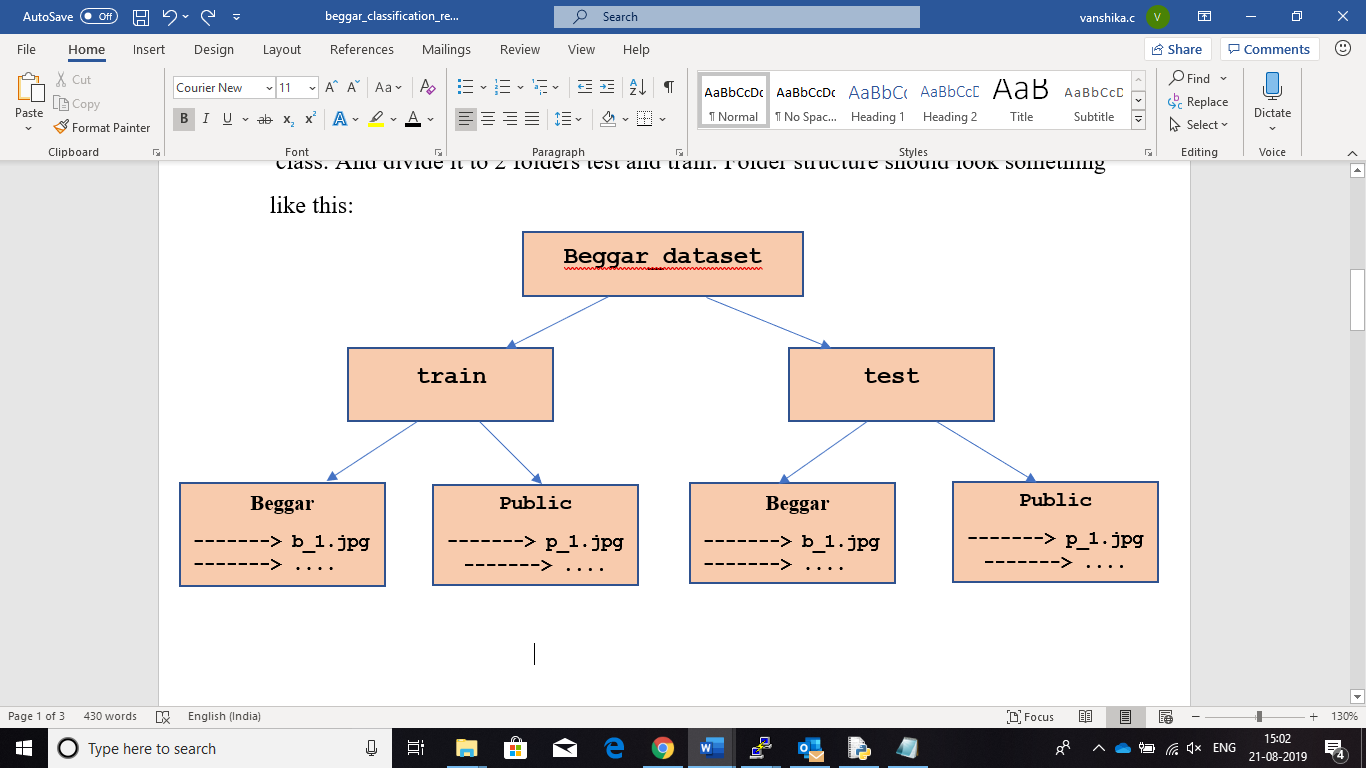
**2) Loading the Data**

Dataset has 2 classes: Beggar and public, So we have 2 folders called

Beggar and public. Each folders should contain images for that particular

class. And divide it to 2 folders test and train. Folder structure should look something

like this:



**3) Flowing data**

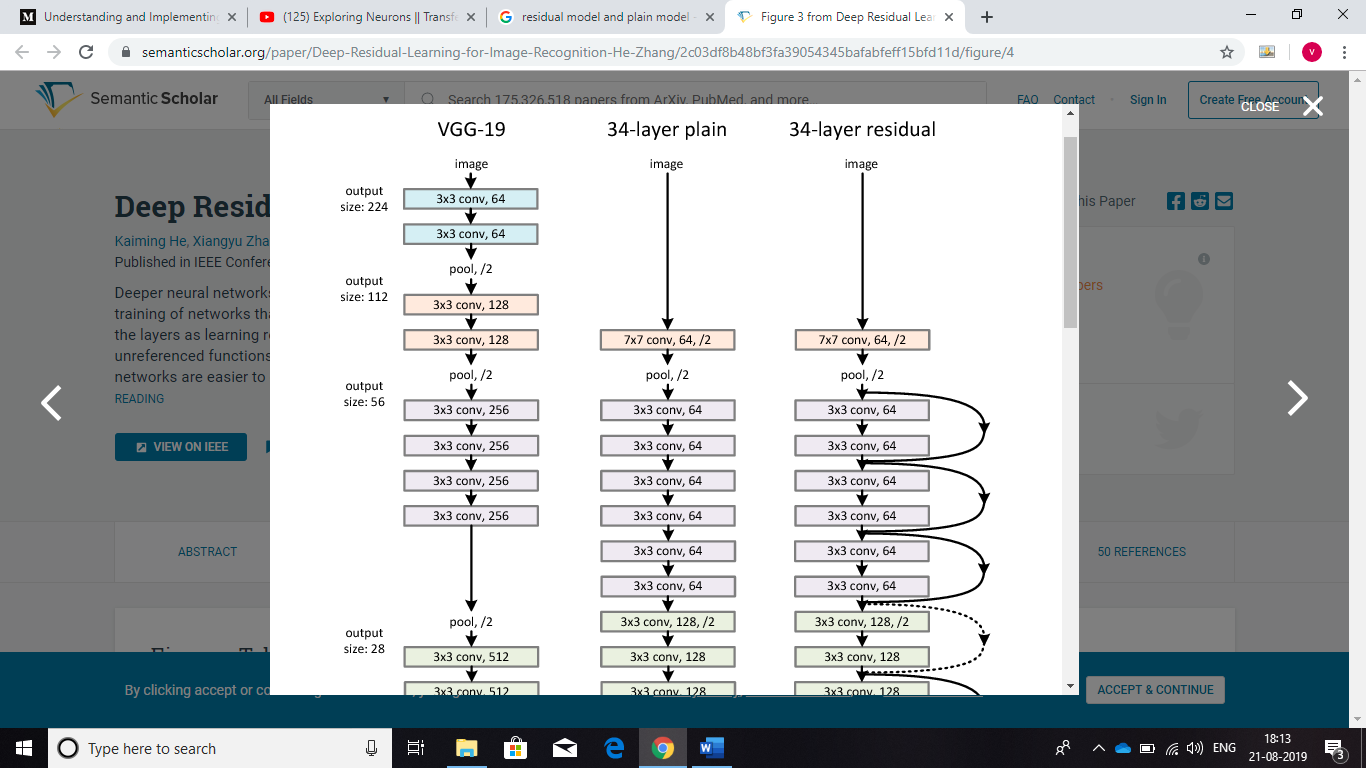
Data generator is used to get our data from our folders and into Keras in an automated way. Keras provides convenient python generator functions for this purpose.

final generator is created using the **flow\_from\_directory** function, which will use a queue to maintain a continuous flow of loading and preparing our images. See the code in appendix 2.

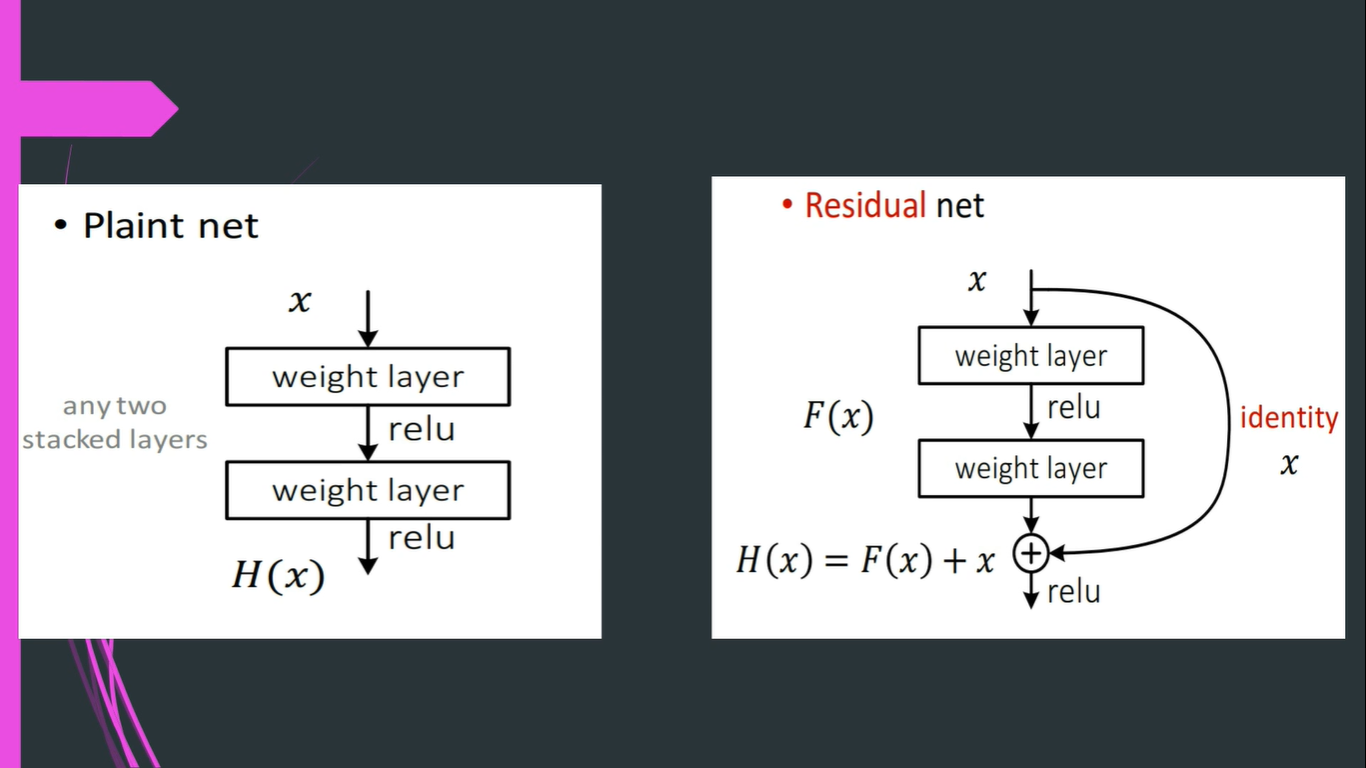
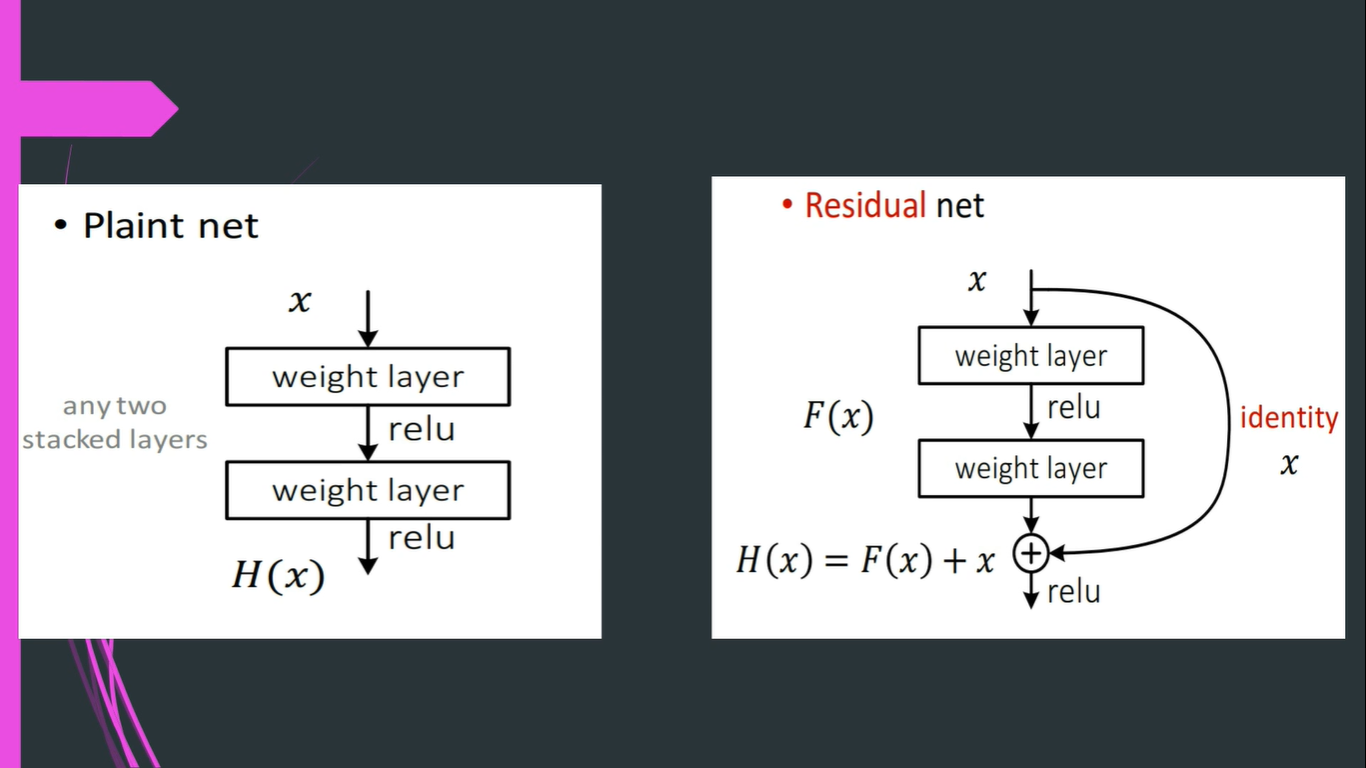
**4) Model**

ResNet50 model is used for transfer learning.

The ResNet50 model was trained with some very specific pre-processing, which we will want to re-use in order to re-train it properly.



Plain -network residual network



It is important to set **include\_top=False** . This setting is important, as it means that we won’t be keeping the Fully Connected (FC) layers at the end of the model.

We start by freezing all the base model’s layers. We don’t want to train those layers since we are trying to leverage the knowledge learned by the network from the previous dataset (in this case ImageNet). By setting the **layer.trainable=False** , we are telling Keras not to update those weights during training.

FC layers is added along with some dropout to each FC layer to reduce the chances of overfitting. Some of the input parameters of model are-

**class\_list = ["beggar1", "public1"]**

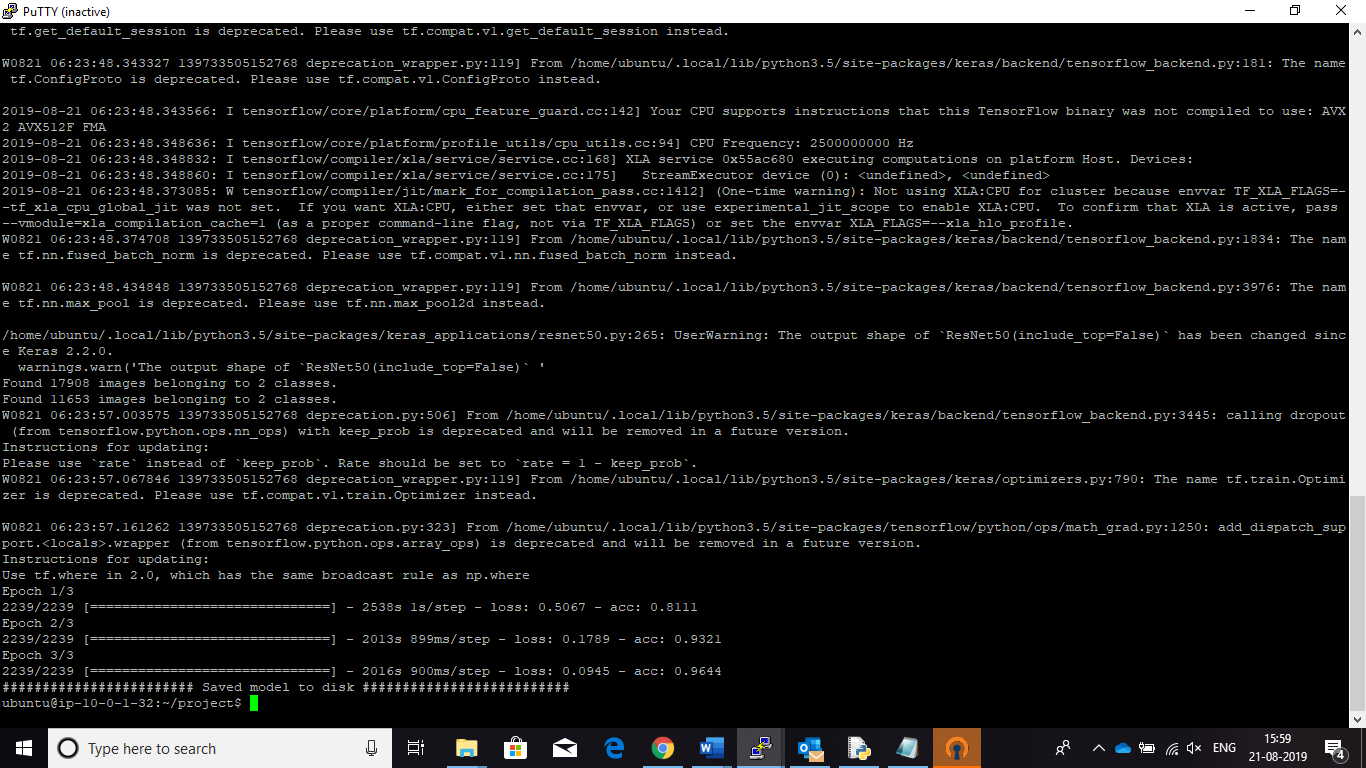
**FC\_LAYERS = [1024, 1024]**

**dropout = 0.5**

At the end final Softmax layer is added and build the Keras model.

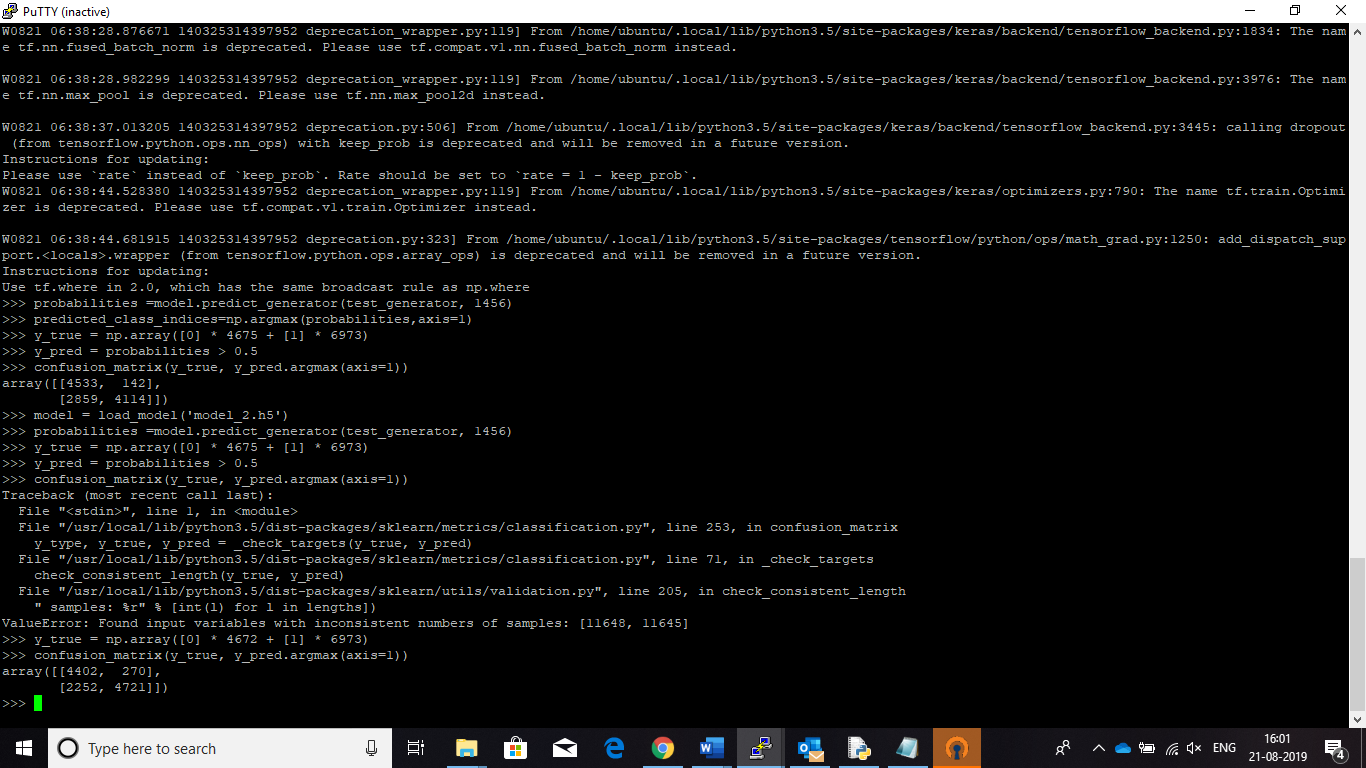
**5) Train and evaluate the Model.**

The final step is to set up our training . Adam optimiser with a small value for the learning rate, epoch is 3 and number of train images are 17913.



The model is evaluated on 11654 images with a batch size of 8.

Confusion matrix is used for describe the performance of a classification model. And the output is as following-



|  |  |  |
| --- | --- | --- |
| **N=11645** | **Predicted**  **0** | **Predicted**  **1** |
| **Beggar**  **(Actual 0)** | **4402** | **270** |
| **Public**  **(Actual 1)** | **2252** | **4721** |

The test accuracy= ((4405+4721)/ (4405+4721+2252+ 270)) \*100

= 78.34%

**Appendix-1**

from PIL import Image

import os, sys

import numpy as np

import cv2

from PIL import ImageEnhance

path = "beggar\_with\_baby/"

dirs = os.listdir( path )

for item in dirs:

im = Image.open(path+item)

f, e = os.path.splitext(path+item)

imResize = im.resize((200,200), Image.ANTIALIAS)

imResize.save('{}\_resized.{}'.format(f,e), quality=90)

for item in dirs:

img = Image.open(path+item)

f, e = os.path.splitext(path+item)

img = np.array(img)

flipped\_img = np.fliplr(img)

img = Image.fromarray(flipped\_img, 'RGB')

img.save('{}\_fliped.{}'.format(f,e), quality=90)

for item in dirs:

image =Image.open(path+item)

f, e = os.path.splitext(path+item)

rotated= image.rotate(45)

trans= image.transpose(Image.ROTATE\_90)

rotated1= image.rotate(30)

rotated2= image.rotate(60)

rotated3= image.rotate(135)

rotated.save('{}\_rot\_1.{}'.format(f,e), quality=90)

rotated1.save('{}\_rot\_2.{}'.format(f,e), quality=90)

rotated2.save('{}\_rot\_3.{}'.format(f,e), quality=90)

rotated3.save('{}\_rot\_4.{}'.format(f,e), quality=90)

trans.save('{}\_rot\_5.{}'.format(f,e), quality=90)

from PIL import ImageEnhance

for item in dirs:

image =Image.open(path+item)

f, e = os.path.splitext(path+item)

enhancer = ImageEnhance.Brightness(image)

factor = 8 / 4.0

en=enhancer.enhance(factor)

en.save('{}\_b1.{}'.format(f,e), quality=90)

factor\_1 = 1 / 4.0

en\_1=enhancer.enhance(factor\_1)

en\_1.save('{}\_b2.{}'.format(f,e), quality=90)

factor\_2 = 5 / 4.0

en\_2=enhancer.enhance(factor\_2)

en\_2.save('{}\_b3.{}'.format(f,e), quality=90)

factor\_3 = 3 / 4.0

en\_3=enhancer.enhance(factor\_3)

en\_3.save('{}\_b4.{}'.format(f,e), quality=90)

for item in dirs:

image =Image.open(path+item)

f, e = os.path.splitext(path+item)

enhancer = ImageEnhance.Color(image)

factor = 8 / 1

en=enhancer.enhance(factor)

en.save('{}\_b1.{}'.format(f,e), quality=90)

factor\_1 = 1 / 1.0

en\_1=enhancer.enhance(factor\_1)

en\_1.save('{}\_b2.{}'.format(f,e), quality=90)

factor\_2 = 5 / 1.0

en\_2=enhancer.enhance(factor\_2)

en\_2.save('{}\_b3.{}'.format(f,e), quality=90)

factor\_3 = 3 / 1.0

en\_3=enhancer.enhance(factor\_3)

en\_3.save('{}\_b4.{}'.format(f,e), quality=90)

**Appendix-2**

###################################Importing the libraries ##########################

from keras.callbacks import ModelCheckpoint

from keras.applications.resnet50 import ResNet50, preprocess\_input

from keras.preprocessing.image import ImageDataGenerator

from matplotlib import pyplot as plt

from sklearn.metrics import confusion\_matrix

import numpy as np

from keras.models import load\_model

from keras.layers import Dense, Activation, Flatten, Dropout

from keras.models import Sequential, Model

from keras.optimizers import SGD, Adam

##############Setting base\_model by calling ResNet50 pre-trained model ###########

HEIGHT = 150

WIDTH = 150

BATCH\_SIZE = 8

base\_model = ResNet50(weights='imagenet',

include\_top=False,

input\_shape=(HEIGHT, WIDTH, 3))

############ path to the data ###############################################

TRAIN\_DIR = "data1/train"

TEST\_DIR = "data1/test"

###############Data generator is used to get our data from our folders###########

train\_datagen = ImageDataGenerator( preprocessing\_function=preprocess\_input )

train\_generator = train\_datagen.flow\_from\_directory(TRAIN\_DIR,

target\_size=(HEIGHT, WIDTH),

batch\_size=BATCH\_SIZE)

test\_datagen= ImageDataGenerator(preprocessing\_function=preprocess\_input)

test\_generator = test\_datagen.flow\_from\_directory(TEST\_DIR, target\_size=(HEIGHT,

WIDTH), batch\_size=BATCH\_SIZE,class\_mode=None,shuffle=False)

#################### Building actual model ##############################

def build\_finetune\_model(base\_model, dropout, fc\_layers, num\_classes):

for layer in base\_model.layers:

layer.trainable = False

x = base\_model.output

x = Flatten()(x)

for fc in fc\_layers:

# New FC layer, random init

x = Dense(fc, activation='relu')(x)

x = Dropout(dropout)(x)

# New softmax layer

predictions = Dense(num\_classes, activation='softmax')(x)

finetune\_model = Model(inputs=base\_model.input, outputs=predictions)

return finetune\_model

class\_list = ["beggar1", "public1"]

FC\_LAYERS = [1024, 1024]

dropout = 0.5

finetune\_model = build\_finetune\_model(base\_model,

dropout=dropout,

fc\_layers=FC\_LAYERS,

num\_classes=len(class\_list))

NUM\_EPOCHS = 3

BATCH\_SIZE = 8

num\_train\_images = 17913

adam = Adam(lr=0.00001)

##################### compile the model ##############################

finetune\_model.compile(adam, loss='categorical\_crossentropy', metrics=['accuracy'])

######################### fitting the model ##############################

history = finetune\_model.fit\_generator(train\_generator, epochs=NUM\_EPOCHS, workers=8,

steps\_per\_epoch=num\_train\_images // BATCH\_SIZE,

shuffle=True)

finetune\_model.save("model\_2.h5")

print("######################## Saved model to disk##########################")

#steps = 11654/8=1456

############################# evaluating the model ##########################

probabilities =finetune\_model.predict\_generator(test\_generator, 1456)

predicted\_class\_indices=np.argmax(probabilities,axis=1)

y\_true = np.array([0] \* 4675 + [1] \* 6973)

y\_pred = probabilities > 0.5

confusion\_matrix(y\_true, y\_pred.argmax(axis=1))