import cv2

import numpy as np

import pandas as pd

from tensorflow.keras.preprocessing.image import ImageDataGenerator

from sklearn.model\_selection import train\_test\_split

from sklearn.preprocessing import LabelEncoder

from keras.models import load\_model, Model

from tensorflow.keras.optimizers import RMSprop

from tensorflow.keras.layers import Dense, GlobalAveragePooling2D, Dropout, BatchNormalization

from tensorflow.keras.applications.resnet\_v2 import ResNet50V2, preprocess\_input

import os

*#read the csv file*

df\_labels = pd.read\_csv("./data\_l//data\_l//labels.csv")

*#store training and testing images folder location*

train\_file =os.path.join('data\_l\data\_l/train/')

print(train\_file)

test\_file = 'F://dog\_breed\_finil//Dogbreed\_new//Dogbreed\_new//data\_l//data\_l//train/'

print(test\_file)

*#check the total number of unique breed in our dataset file*

print("Total number of unique Dog Breeds :",len(df\_labels.breed.unique()))

*#specify number*

num\_breeds = 60

im\_size = 224

batch\_size = 64

encoder = LabelEncoder()

*#get only 60 unique breeds record*

breed\_dict = list(df\_labels['breed'].value\_counts().keys())

new\_list = sorted(breed\_dict,reverse=True)[:num\_breeds\*2+1:2]

*#change the dataset to have only those 60 unique breed records*

df\_labels = df\_labels.query('breed in @new\_list')

*#create new column which will contain image name with the image extension*

df\_labels['img\_file'] = df\_labels['id'].apply(**lambda** x: x + ".jpg")

*#create a numpy array of the shape*

*#(number of dataset records, image size , image size, 3 for rgb channel ayer)*

*#this will be input for model*

train\_x = np.zeros((len(df\_labels), im\_size, im\_size, 3), dtype='float32')

*#iterate over img\_file column of our dataset*

for i, img\_id in enumerate(df\_labels['img\_file']):

*#read the image file and convert into numeric format*

*#resize all images to one dimension i.e. 224x224*

*#we will get array with the shape of*

*# (224,224,3) where 3 is the RGB channels layers*

    trainfile=str(train\_file)+str(img\_id)

    img= cv2.resize(cv2.imread(trainfile,cv2.IMREAD\_COLOR),((im\_size,im\_size)))

*#scale array into the range of -1 to 1.*

*#preprocess the array and expand its dimension on the axis 0*

    img\_array = preprocess\_input(np.expand\_dims(np.array(img[...,::-1].astype(np.float32)).copy(), axis=0))

*#update the train\_x variable with new element*

    train\_x[i] = img\_array

*#This will be the target for the model.*

*#convert breed names into numerical format*

train\_y = encoder.fit\_transform(df\_labels["breed"].values)

*#split the dataset in the ratio of 80:20.*

*#80% for training and 20% for testing purpose*

x\_train, x\_test, y\_train, y\_test = train\_test\_split(train\_x,train\_y,test\_size=0.2,random\_state=42)

*#Image augmentation using ImageDataGenerator class*

train\_datagen = ImageDataGenerator(rotation\_range=45,

                                    width\_shift\_range=0.2,

                                    height\_shift\_range=0.2,

                                    shear\_range=0.2,

                                    zoom\_range=0.25,

                                    horizontal\_flip=True,

                                    fill\_mode='nearest')

*#generate images for training sets*

train\_generator = train\_datagen.flow(x\_train,

                                        y\_train,

                                        batch\_size=batch\_size)

*#same process for Testing sets also by declaring the instance*

test\_datagen = ImageDataGenerator()

test\_generator = test\_datagen.flow(x\_test,

                                        y\_test,

                                        batch\_size=batch\_size)

resnet = ResNet50V2(input\_shape = [im\_size,im\_size,3], weights='imagenet', include\_top=False)

*#freeze all trainable layers and train only top layers*

for layer in resnet.layers:

        layer.trainable = False

*#add global average pooling layer and Batch Normalization layer*

x = resnet.output

x = BatchNormalization()(x)

x = GlobalAveragePooling2D()(x)

x = Dropout(0.5)(x)

*#add fully connected layer*

x = Dense(1024, activation='relu')(x)

x = Dropout(0.5)(x)

*#add output layer having the shape equal to number of breeds*

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

*#create model class with inputs and outputs*

model = Model(inputs=resnet.input, outputs=predictions)

*#model.summary()*

*#epochs for model training and learning rate for optimizer*

epochs = 20

learning\_rate = 1e-3

*#using RMSprop optimizer to compile or build the model*

optimizer = RMSprop(learning\_rate=learning\_rate,rho=0.9)

model.compile(optimizer=optimizer,

                loss='sparse\_categorical\_crossentropy',

                metrics=["accuracy"])

*#fit the training generator data and train the model*

hist = model.fit(train\_generator,

                    steps\_per\_epoch= x\_train.shape[0] // batch\_size,

                    epochs= epochs,

                    validation\_data= test\_generator,

                    validation\_steps= x\_test.shape[0] // batch\_size)

*#Save the model for prediction*

model.save("model")

i

import cv2

import numpy as np

import pandas as pd

from tensorflow.keras.preprocessing.image import ImageDataGenerator

from sklearn.model\_selection import train\_test\_split

from sklearn.preprocessing import LabelEncoder

from tensorflow.keras.models import load\_model, Model

from tensorflow.keras.optimizers import RMSprop

from tensorflow.keras.layers import Dense, GlobalAveragePooling2D, Dropout, BatchNormalization

from tensorflow.keras.applications.resnet\_v2 import ResNet50V2, preprocess\_input

from tensorflow.keras.preprocessing import image

import numpy as np

from PIL import Image

*#load the model*

model = load\_model("model")

df\_labels = pd.read\_csv("data\_l//data\_l//labels.csv")

breed\_dict = list(df\_labels['breed'].value\_counts().keys())

new\_list = sorted(breed\_dict,reverse=True)[:120\*2+1:2]

*#change the dataset to have only those 60 unique breed records*

df\_labels = df\_labels.query('breed in @new\_list')

im\_size=224

*#get the image of the dog for prediction*

imag ="F:/dog\_breed\_finil/Dogbreed\_new/Dogbreed\_new/data\_l/data\_l/train/0a1b0b7df2918d543347050ad8b16051.jpg"

*# imag=Image.open(r"../data\_l//data\_l//test//6c1add2344fd41b7a73e06b61727a64.jpg")*

imag=image.img\_to\_array(imag,target\_size=(224,224))

imag=np.expand\_dims(imag,axis=0)

prd=model.predict(imag)

prd=np.argmax(prd)

print(df\_labels)

*#read the image file and convert into numeric format*

*#     #resize all images to one dimension i.e. 224x224*

pred\_img\_array = cv2.resize(cv2.imread(imag,cv2.IMREAD\_COLOR),((im\_size,im\_size)))

*#scale array into the range of -1 to 1.*

*#expand the dimension on the axis 0 and normalize the array values*

pred\_img\_array = preprocess\_input(np.expand\_dims(np.array(pred\_img\_array[...,::-1].astype(np.float32)).copy(), axis=0))

*#feed the model with the image array for prediction*

pred\_val = model.predict(np.array(pred\_img\_array,dtype="float32"))

*#display the image of dog*

cv2.imshow("hai hello",cv2.resize(cv2.imread(imag,cv2.IMREAD\_COLOR),((im\_size,im\_size))))

*#display the predicted breed of dog*

pred\_breed = sorted(new\_list)[np.argmax(pred\_val)]

print("Predicted Breed for this Dog is :",pred\_breed)

*# print(np.argmax(pred\_val))*

from django.http import HttpResponse

from django.shortcuts import render,redirect

from . models import Usertbl,Predict

from image.models import Shop\_up

from django.contrib.sessions.models import Session

from sklearn.datasets import load\_files

*#from keras.utils import np\_utils*

import numpy as np

from glob import glob

import cv2

import cv2

import numpy as np

import pandas as pd

from tensorflow.keras.preprocessing.image import ImageDataGenerator

from sklearn.model\_selection import train\_test\_split

from sklearn.preprocessing import LabelEncoder

from tensorflow.keras.models import load\_model, Model

from tensorflow.keras.optimizers import RMSprop

from tensorflow.keras.layers import Dense, GlobalAveragePooling2D, Dropout, BatchNormalization

from tensorflow.keras.applications.resnet\_v2 import ResNet50V2, preprocess\_input

from tensorflow.keras.preprocessing import image

from PIL import Image

*# Create your views here.*

**def** userreg(request):

    if request.method=="POST":

        fn=request.POST.get("name")

        user=request.POST.get("uname")

        passw=request.POST.get("password")

        eml=request.POST.get("email")

        mb=request.POST.get("cnum")

        obj=Usertbl.objects.create(fname=fn,username=user,password=passw,email=eml,mobile=mb)

        obj.save()

        return HttpResponse("<h4> Registrtion Successfully</h4> <a href='/user/login'>Login </a>")

    return render(request,'userreg.html')

**def** index(request):

    return render(request,'index.html')

**def** login(req):

    if req.method=="POST":

        user=req.POST.get("eml")

        pss=req.POST.get("passw")

        req.session['username']=user

        req.session['password']=pss

        obj=Usertbl.objects.filter(email=user,password=pss)

*# pr=Predict.objects.filter(id=idn)*

        if obj:

            for l in obj:

                idn=l.id

                nam=l.fname

            req.session['idno']=idn

            shp=Shop\_up.objects.all()

            return render(req,'homepage.html',{'fname':nam,'data':shp,'url':""})

        else:

            return HttpResponse("Check Your Username and Password <a href='/user/login'>Login</a>")

    return render(req,'userreg.html')

**def** home(req):

    if True:

        user=req.session['username']

        pss=req.session['password']

        obj=Usertbl.objects.filter(email=user,password=pss)

        if obj:

            for l in obj:

                idn=l.id

                nam=l.fname

                req.session['idno']=idn

            print(idn,"idno user")

            pr=Shop\_up.objects.all()

        return render(req,'homepage.html',{'fname':nam,'data':pr})

    else:

        return render(req,'index.html')

**def** face\_detector(img\_path):

        face\_cascade=cv2.CascadeClassifier('data\haarcascade\_frontalface\_default.xml')

        img=cv2.imread("upload//predict//alanjose.jpg")

        gray=cv2.cvtColor(img,cv2.COLOR\_BGR2GRAY)

        faces=face\_cascade.detectMultiScale(gray)

        if len(faces)>0:

            print("human")

        return len(faces) > 0

**def** dog\_detector(img\_path):

        face\_cascade = cv2.CascadeClassifier('data\haarcascade\_frontalface\_default.xml')

        img = cv2.imread(img\_path)

        gray = cv2.cvtColor(img, cv2.COLOR\_BGR2GRAY)

        faces = face\_cascade.detectMultiScale(gray)

        if len(faces)>0:

            print("dog")

        return len(faces) > 0

*# def prediction(image\_p):*

*#         human=face\_detector(image\_p)*

*#         print("Human",human)*

*#         dog\_d=dog\_detector(image\_p)*

*#         print("Dog",dog\_d)*

*#         if human:*

*#             rs="Human Face"*

*#             return rs*

*#         elif dog\_d:*

*#             rs="Dog Detected"*

*#             return rs*

*#         else:*

*#             rs="?"*

*#             return rs*

*# model=load\_model("dogbreed.models.best.hdf5")*

*# figure = plt.figure(figsize=(20,10))*

*# image = cv2.imread(imagePath)*

*# ax = figure.add\_subplot(3,4,1)*

*# ax.imshow(image)*

*# res = cv2.resize(image,dsize=(299,299),interpolation=cv2.INTER\_CUBIC)*

*# res1 = np.zeros([1,299,299,3])*

*# res1[0,:,:,:] = res*

*# print(res.shape)*

*# bottle\_neck = model.predict(preprocess\_input(res1),verbose=1)*

*# prediction = int(np.argmax(model1.predict(bottle\_neck),axis=1))*

*# ax.set\_title("I think it is "+dog\_names[prediction])*

**def** prediction(img):

    im\_size = 224

    df\_labels = pd.read\_csv("data\_l//data\_l//labels.csv")

*#Save the model for prediction*

*#get only 60 unique breeds record*

    breed\_dict = list(df\_labels['breed'].value\_counts().keys())

    new\_list = sorted(breed\_dict,reverse=True)[:60\*2+1:2]

*#change the dataset to have only those 60 unique breed records*

    df\_labels = df\_labels.query('breed in @new\_list')

*#load the model*

    model = load\_model("model")

    imag=Image.open(img)

    imag.save("upload//predict//prd.jpg")

*#get the image of the dog for prediction*

    pred\_img\_path ="D:/django work/Dogbreed\_new/Dogbreed\_new/upload/predict/prd.jpg"

*#pred\_img\_path=imag*

*#read the image file and convert into numeric format*

*#resize all images to one dimension i.e. 224x224*

    pred\_img\_array = cv2.resize(cv2.imread(pred\_img\_path,cv2.IMREAD\_COLOR),((im\_size,im\_size)))

*#scale array into the range of -1 to 1.*

*#expand the dimension on the axis 0 and normalize the array values*

    pred\_img\_array = preprocess\_input(np.expand\_dims(np.array(pred\_img\_array[...,::-1].astype(np.float32)).copy(), axis=0))

*#feed the model with the image array for prediction*

    pred\_val = model.predict(np.array(pred\_img\_array,dtype="float32"))

*#display the image of dog*

*# cv2.imshow(“TechVidvan”,cv2.resize(cv2.imread(pred\_img\_path,cv2.IMREAD\_COLOR),((im\_size,im\_size))))*

*#display the predicted breed of dog*

    pred\_breed = sorted(new\_list)[np.argmax(pred\_val)]

    print("Predicted Breed for this Dog is :",pred\_breed)

    return pred\_breed

**def** detec(req):

    if req.method=="POST":

        pdimg=req.FILES.get('dg')

        fl=req.POST.get("txt")

        print(pdimg)

*# im= cv2.imread(pdimg)*

        idn=req.session['idno']

        pd=Predict.objects.create(usid=idn,prd="predic",location=pdimg)

        if pd:

            pd.save()

            pdimg1="predict/"+str(pdimg)

            obj=Predict.objects.filter(location=pdimg1)

            if obj:

                for l in obj:

                    s=l.location

                dete=prediction(pdimg)

                print(s)

                print("hello",dete)

*# pdimg=str(pdimg)*

*# prd=pdimg.split('.')[0]*

                prd=dete

                s="Predicted Breed for this Dog is : "

                print(prd)

                return render(req,'detect.html',{'msg':'image uploaded Successfully','prd':'Under developing','msg2':s+prd,'data':obj,'wk':dete})

            else:

                return render(req,'detect.html',{'msg':'error is detecting'})

        else:

            return render(req,'detect.html',{'msg':'error is detecting'})

    else:

        return render(req,'detect.html')

**def** details(req):

    if req.GET.get("sub"):

        d=req.GET.get("sub")

        detail=wikipedias(d)

    return HttpResponse(detail)

**def** logout(request):

    request.session['username']="nill"

    request.session['password']="nill"

    return render(request,"index.html")

**def** img\_predict(request):

    return render(request,"index.html")

**def** payment(request):

    ids=request.GET.get('p\_id')

    obj=Shop\_up.objects.filter(id=ids)

    request.session['bkid']=ids

    paymentOb= Shop\_up.objects.get(id=ids)

    return render(request,'detail.html',{'prod':obj,'plan':paymentOb})

**def** resp(request):

    ids=request.GET.get('mchkey')

    djf={'dk':'http//splietispaymgateway.com//key?839050930'}

    return HttpResponse("payment error<br>Merchent key failer.")

**def** searchpic(req):

    if req.method=="POST":

        srh=req.POST.get("srch")

        obj=Shop\_up.objects.filter(bnam=srh)

        if obj:

            return render(req,'homepage.html',{'data':obj})

        else:

            return render(req,'homepage.html',{'pro':" ","msg":"Breed Not Found"})

    else:

        return redirect("/user/home")

**def** wikipedias(data):

    import wikipedia

    if data!="":

        result = wikipedia.summary(data, sentences = 10)

    else:

        result="Not Found"

    return result