**SAMPLE CODE**

**User side views.py**

from django.contrib import messages

from django.shortcuts import render, HttpResponse

from django.core.files.storage import FileSystemStorage

# Create your views here.

from user.forms import UserRegistrationForm

from user.models import UserRegistrationModel, UserFirstImageModel, UserSecondImageModel

def UserLoginCheck(request):

if request.method == "POST":

loginid = request.POST.get('loginid')

pswd = request.POST.get('pswd')

print("Login ID = ", loginid, ' Password = ', pswd)

try:

check = UserRegistrationModel.objects.get(loginid=loginid, password=pswd)

status = check.status

print('Status is = ', status)

if status == "activated":

request.session['id'] = check.id

request.session['loggeduser'] = check.name

request.session['loginid'] = loginid

request.session['email'] = check.email

print("User id At", check.id, status)

return render(request, 'user/UserHomePage.html', {})

else:

messages.success(request, 'Your Account Not at activated')

return render(request, 'userlogin.html')

# return render(request, 'user/userpage.html',{})

except Exception as e:

print('Exception is ', str(e))

pass

messages.success(request, 'Invalid Login id and password')

return render(request, 'userlogin.html', {})

def UserHome(request):

return render(request,"user/UserHomePage.html",{})

def UserUploadImageForm(request):

loginid = request.session['loginid']

data = UserFirstImageModel.objects.filter(username= loginid)

return render(request,"user/UploadImageform.html", {"data": data})

def UserImageProcessFirst(request):

if request.method== 'POST':

myfile = request.FILES['file']

fs = FileSystemStorage()

filename = fs.save(myfile.name, myfile)

uploaded\_file\_url = fs.url(filename)

from .utility.ImageRecognition import Recognitions

obj = Recognitions()

resultDict = obj.start\_process(filename)

resultDict = str(resultDict)

print('Result:', resultDict)

loginid = request.session['loginid']

email = request.session['email']

UserFirstImageModel.objects.create(username=loginid, email=email, filename=filename, results=resultDict, file=uploaded\_file\_url)

messages.success(request, 'Image Processed Success')

print("File Image Name "+uploaded\_file\_url)

data = UserFirstImageModel.objects.filter(username=loginid)

return render(request, "user/UploadImageform.html", {"data": data})

def UserImageAIForm(request):

loginid = request.session['loginid']

data = UserSecondImageModel.objects.filter(username=loginid)

return render(request, "user/UserImageAI.html", {"data": data})

def ProcessUserImageAI(request):

if request.method == 'POST':

myfile = request.FILES['file']

fs = FileSystemStorage()

filename = fs.save(myfile.name, myfile)

uploaded\_file\_url = fs.url(filename)

from .utility.ImageAiPredict import run\_inference

resultDict = run\_inference(filename)

print('Result:', resultDict)

loginid = request.session['loginid']

email = request.session['email']

UserSecondImageModel.objects.create(username=loginid, email=email, filename=filename, results=resultDict,

file=uploaded\_file\_url)

messages.success(request, 'Image Processed Success')

print("File Image Name " + uploaded\_file\_url)

data = UserSecondImageModel.objects.filter(username=loginid)

return render(request, "user/UserImageAI.html", {"data": data})

def StartTraining(request):

from .utility import ImageAiPredict

ImageAiPredict.train\_network()

return render(request, "user/UserHomePage.html", {})

imageAI predict

# from \_\_future\_\_ import print\_function

from django.conf import settings

from keras.preprocessing.image import ImageDataGenerator

from keras.callbacks import LearningRateScheduler

import os

from keras.callbacks import ModelCheckpoint

from io import open

import requests

import shutil

from zipfile import ZipFile

import keras

from keras.layers import Dense, Activation, Conv2D, MaxPool2D, GlobalAvgPool2D, BatchNormalization, add, Input

from keras.models import Model

from tensorflow.python.keras.preprocessing import image

import numpy as np

import json

execution\_path = os.getcwd()

# ----------------- The Section Responsible for Downloading the Dataset ---------------------

SOURCE\_PATH = "https://github.com/OlafenwaMoses/IdenProf/releases/download/v1.0/idenprof-jpg.zip"

FILE\_DIR = os.path.join(execution\_path, "idenprof-jpg.zip")

DATASET\_DIR = os.path.join(execution\_path, "idenprof")

DATASET\_TRAIN\_DIR = os.path.join(DATASET\_DIR, "train")

DATASET\_TEST\_DIR = os.path.join(DATASET\_DIR, "test")

def download\_idenprof():

if (os.path.exists(FILE\_DIR) == False):

print("Downloading idenprof-jpg.zip")

data = requests.get(SOURCE\_PATH,

stream=True)

with open(FILE\_DIR, "wb") as file:

shutil.copyfileobj(data.raw, file)

del data

extract = ZipFile(FILE\_DIR)

extract.extractall(execution\_path)

extract.close()

# ----------------- The Section Responsible for Training ResNet50 on the IdenProf dataset ---------------------

# Directory in which to create models

save\_direc = os.path.join(os.getcwd(), 'idenprof\_models')

# Name of model files

model\_name = 'idenprof\_weight\_model.{epoch:03d}-{val\_acc}.h5'

# Create Directory if it doesn't exist

if not os.path.isdir(save\_direc):

os.makedirs(save\_direc)

# Join the directory with the model file

modelpath = os.path.join(save\_direc, model\_name)

# Checkpoint to save best model

checkpoint = ModelCheckpoint(filepath=modelpath,

monitor='val\_acc',

verbose=1,

save\_best\_only=True,

save\_weights\_only=True,

period=1)

# Function for adjusting learning rate and saving dummy file

def lr\_schedule(epoch):

"""

Learning Rate Schedule

"""

# Learning rate is scheduled to be reduced after 80, 120, 160, 180 epochs. Called automatically every

# epoch as part of callbacks during training.

lr = 1e-3

if epoch > 180:

lr \*= 1e-4

elif epoch > 160:

lr \*= 1e-3

elif epoch > 120:

lr \*= 1e-2

elif epoch > 80:

lr \*= 1e-1

print('Learning rate: ', lr)

return lr

lr\_scheduler = LearningRateScheduler(lr\_schedule)

def resnet\_module(input, channel\_depth, strided\_pool=False):

residual\_input = input

stride = 1

if (strided\_pool):

stride = 2

residual\_input = Conv2D(channel\_depth, kernel\_size=1, strides=stride, padding="same",

kernel\_initializer="he\_normal")(residual\_input)

residual\_input = BatchNormalization()(residual\_input)

input = Conv2D(int(channel\_depth / 4), kernel\_size=1, strides=stride, padding="same",

kernel\_initializer="he\_normal")(input)

input = BatchNormalization()(input)

input = Activation("relu")(input)

input = Conv2D(int(channel\_depth / 4), kernel\_size=3, strides=1, padding="same", kernel\_initializer="he\_normal")(

input)

input = BatchNormalization()(input)

input = Activation("relu")(input)

input = Conv2D(channel\_depth, kernel\_size=1, strides=1, padding="same", kernel\_initializer="he\_normal")(input)

input = BatchNormalization()(input)

input = add([input, residual\_input])

input = Activation("relu")(input)

return input

def resnet\_first\_block\_first\_module(input, channel\_depth):

residual\_input = input

stride = 1

residual\_input = Conv2D(channel\_depth, kernel\_size=1, strides=1, padding="same", kernel\_initializer="he\_normal")(

residual\_input)

residual\_input = BatchNormalization()(residual\_input)

input = Conv2D(int(channel\_depth / 4), kernel\_size=1, strides=stride, padding="same",

kernel\_initializer="he\_normal")(input)

input = BatchNormalization()(input)

input = Activation("relu")(input)

input = Conv2D(int(channel\_depth / 4), kernel\_size=3, strides=stride, padding="same",

kernel\_initializer="he\_normal")(input)

input = BatchNormalization()(input)

input = Activation("relu")(input)

input = Conv2D(channel\_depth, kernel\_size=1, strides=stride, padding="same", kernel\_initializer="he\_normal")(input)

input = BatchNormalization()(input)

input = add([input, residual\_input])

input = Activation("relu")(input)

return input

def resnet\_block(input, channel\_depth, num\_layers, strided\_pool\_first=False):

for i in range(num\_layers):

pool = False

if (i == 0 and strided\_pool\_first):

pool = True

input = resnet\_module(input, channel\_depth, strided\_pool=pool)

return input

def ResNet50(input\_shape, num\_classes=10):

input\_object = Input(shape=input\_shape)

layers = [3, 4, 6, 3]

channel\_depths = [256, 512, 1024, 2048]

output = Conv2D(64, kernel\_size=7, strides=2, padding="same", kernel\_initializer="he\_normal")(input\_object)

output = BatchNormalization()(output)

output = Activation("relu")(output)

output = MaxPool2D(pool\_size=(3, 3), strides=(2, 2))(output)

output = resnet\_first\_block\_first\_module(output, channel\_depths[0])

for i in range(4):

channel\_depth = channel\_depths[i]

num\_layers = layers[i]

strided\_pool\_first = True

if (i == 0):

strided\_pool\_first = False

num\_layers = num\_layers - 1

output = resnet\_block(output, channel\_depth=channel\_depth, num\_layers=num\_layers,

strided\_pool\_first=strided\_pool\_first)

output = GlobalAvgPool2D()(output)

output = Dense(num\_classes)(output)

output = Activation("softmax")(output)

model = Model(inputs=input\_object, outputs=output)

return model

def train\_network():

#download\_idenprof()

#print(os.listdir(os.path.join(execution\_path, "idenprof")))

optimizer = keras.optimizers.Adam(lr=0.01, decay=1e-4)

batch\_size = 32

num\_classes = 10

epochs = 200

model = ResNet50((224, 224, 3), num\_classes=num\_classes)

model.compile(loss="categorical\_crossentropy", optimizer=optimizer, metrics=["accuracy"])

model.summary()

print("Using real time Data Augmentation")

train\_datagen = ImageDataGenerator(

rescale=1. / 255,

horizontal\_flip=True)

test\_datagen = ImageDataGenerator(

rescale=1. / 255)

train\_generator = train\_datagen.flow\_from\_directory(DATASET\_TRAIN\_DIR, target\_size=(224, 224),

batch\_size=batch\_size, class\_mode="categorical")

test\_generator = test\_datagen.flow\_from\_directory(DATASET\_TEST\_DIR, target\_size=(224, 224), batch\_size=batch\_size,

class\_mode="categorical")

model.fit\_generator(train\_generator, steps\_per\_epoch=int(9000 / batch\_size), epochs=epochs,

validation\_data=test\_generator,

validation\_steps=int(2000 / batch\_size), callbacks=[checkpoint, lr\_scheduler])

# ----------------- The Section Responsible for Inference ---------------------

CLASS\_INDEX = None

MODEL\_PATH = os.path.join(execution\_path, settings.MEDIA\_ROOT + "\\" + "idenprof\_061-0.7933.h5")

JSON\_PATH = os.path.join(execution\_path, settings.MEDIA\_ROOT + "\\" + "idenprof\_model\_class.json")

def preprocess\_input(x):

x \*= (1. / 255)

return x

def decode\_predictions(preds, top=5, model\_json=""):

global CLASS\_INDEX

if CLASS\_INDEX is None:

CLASS\_INDEX = json.load(open(model\_json))

results = []

for pred in preds:

top\_indices = pred.argsort()[-top:][::-1]

for i in top\_indices:

each\_result = []

each\_result.append(CLASS\_INDEX[str(i)])

each\_result.append(pred[i])

results.append(each\_result)

return results

def run\_inference(filename):

model = ResNet50(input\_shape=(224, 224, 3), num\_classes=10)

model.load\_weights(MODEL\_PATH)

#picture = os.path.join(execution\_path, "Haitian-fireman.jpg")

picture = settings.MEDIA\_ROOT + "\\" + filename # os.path.join(execution\_path, filename)

image\_to\_predict = image.load\_img(picture, target\_size=(224, 224))

image\_to\_predict = image.img\_to\_array(image\_to\_predict, data\_format="channels\_last")

image\_to\_predict = np.expand\_dims(image\_to\_predict, axis=0)

image\_to\_predict = preprocess\_input(image\_to\_predict)

prediction = model.predict(x=image\_to\_predict, steps=1)

predictiondata = decode\_predictions(prediction, top=int(5), model\_json=JSON\_PATH)

resultDict = {}

for result in predictiondata:

print(str(result[0]), " : ", str(result[1] \* 100))

resultDict.update({result[0]: round(result[1] \* 100, 2)})

return resultDict

# run\_inference()

# train\_network()

ImageRecognitioans

from imageai.Prediction import ImagePrediction

import os

from django.conf import settings

class Recognitions:

def start\_process(self, filepath):

execution\_path = os.getcwd()

prediction = ImagePrediction()

prediction.setModelTypeAsResNet()

dataset = settings.MEDIA\_ROOT + "\\" + 'resnet50\_weights\_tf\_dim\_ordering\_tf\_kernels.h5'

filepath = settings.MEDIA\_ROOT + "\\" + filepath

print("Path is " + dataset)

print("Execution path :" + execution\_path)

prediction.setModelPath(dataset)

prediction.loadModel()

resultDict = {}

predictions, percentage\_probabilities = prediction.predictImage(filepath, result\_count=100)

for index in range(5):

print(predictions[index], " : ", percentage\_probabilities[index])

resultDict.update({predictions[index]: round(percentage\_probabilities[index], 2)})

return resultDict;

Base.html

{% load static %}

<!DOCTYPE html>

<html lang="en">

<head>

<!-- basic -->

<meta charset="utf-8">

<meta http-equiv="X-UA-Compatible" content="IE=edge">

<!-- mobile metas -->

<meta name="viewport" content="width=device-width, initial-scale=1">

<meta name="viewport" content="initial-scale=1, maximum-scale=1">

<!-- site metas -->

<title>Image Recognition</title>

<meta name="keywords" content="">

<meta name="description" content="">

<meta name="author" content="">

<!-- bootstrap css -->

<link rel="stylesheet" href="{% static 'css/bootstrap.min.css' %}">

<!-- style css -->

<link rel="stylesheet" href="{% static 'css/style.css' %}">

<!-- Responsive-->

<link rel="stylesheet" href="{% static 'css/responsive.css' %}">

<!-- fevicon -->

<link rel="icon" href="{% static 'images/fevicon.png' %}" type="image/gif" />

<!-- Scrollbar Custom CSS -->

<link rel="stylesheet" href="{% static 'css/jquery.mCustomScrollbar.min.css' %}">

<!-- Tweaks for older IEs-->

<link rel="stylesheet" href="https://netdna.bootstrapcdn.com/font-awesome/4.0.3/css/font-awesome.css">

<link rel="stylesheet" href="https://cdnjs.cloudflare.com/ajax/libs/fancybox/2.1.5/jquery.fancybox.min.css" media="screen">

<!--[if lt IE 9]>

<script src="https://oss.maxcdn.com/html5shiv/3.7.3/html5shiv.min.js"></script>

<script src="https://oss.maxcdn.com/respond/1.4.2/respond.min.js"></script><![endif]-->

</head>

<!-- body -->

<body class="main-layout">

<!-- loader -->

<div class="loader\_bg">

<div class="loader"><img src="{% static 'images/loading.gif' %}" alt="#" /></div>

</div>

<!-- end loader -->

<!-- header -->

<header>

<!-- header inner -->

<div class="header">

<div class="container-fluid">

<div class="row">

<div class="col-xl-3 col-lg-3 col-md-3 col-sm-3 col logo\_section">

<div class="full">

<div class="center-desk">

<div class="logo">

<a href="{% url 'index' %}">Image Recognition </a>

<!-- <a href="index.html"><img src="{% static 'images/logo.png' %}" alt="#" /></a>-->

</div>

</div>

</div>

</div>

<div class="col-xl-9 col-lg-9 col-md-9 col-sm-9">

<nav class="navigation navbar navbar-expand-md navbar-dark ">

<button class="navbar-toggler" type="button" data-toggle="collapse" data-target="#navbarsExample04" aria-controls="navbarsExample04" aria-expanded="false" aria-label="Toggle navigation">

<span class="navbar-toggler-icon"></span>

</button>

<div class="collapse navbar-collapse" id="navbarsExample04">

<ul class="navbar-nav mr-auto">

<li class="nav-item">

<a class="nav-link" href="{% url 'index' %}">Home</a>

</li>

<li class="nav-item">

<a class="nav-link" href="{% url 'UserLogin' %}">User</a>

</li>

<li class="nav-item">

<a class="nav-link" href="{% url 'adminlogin' %}">Admin</a>

</li>

<li class="nav-item">

<a class="nav-link" href="{% url 'UserRegister' %}">Registration</a>

</li>

</ul>

</div>

</nav>

</div>

</div>

</div>

</div>

</div>

</header>

<section class="banner\_main">

<div class="container">

<div class="row d\_flex">

<div class="col-md-4">

<div class="text-bg">

<h1><span class="black">Image Recognition<br>

Using <br>

Artificial Intelligence

</span></h1>

<!-- <a href="#contact">Contact Us</a>-->

</div>

</div><br>

<div class="col-md-8">

<div class="text-img">

<figure><img src="{% static 'images/ml.jpg' %}" alt="#"/></figure>

</div>

<div class="text-img1">

<figure><img src="{% static 'images/ai.jpg' %}" alt="#"/></figure>

</div>

</div>

</div>

</div>

</section>

<div id="we\_do" class="we\_do">

<div class="container">

<div class="row">

<div class="col-md-12">

<div class="titlepage">

{%block contents%}

{%endblock%}

<br><br><br>

</div>

</div>

</div>

<!-- footer -->

<footer>

<div class="copyright">

<div class="container">

<div class="row">

<div class="col-md-12">

<p>Copyright 2021 All Right Reserved By <a href="#"> Alex Corporation</a></p>

</div>

</div>

</div>

</div>

</div>

</footer>

<!-- end footer -->

<!-- Javascript files-->

<script src="{% static 'js/jquery.min.js' %}"></script>

<script src="{% static 'js/popper.min.js' %}"></script>

<script src="{% static 'js/bootstrap.bundle.min.js' %}"></script>

<script src="{% static 'js/jquery-3.0.0.min.js' %}"></script>

<script src="{% static 'js/plugin.js' %}"></script>

<!-- sidebar -->

<script src="{% static 'js/jquery.mCustomScrollbar.concat.min.js' %}"></script>

<script src="{% static 'js/custom.js' %}"></script>

<script src="https:cdnjs.cloudflare.com/ajax/libs/fancybox/2.1.5/jquery.fancybox.min.js"></script>

</body>

</html>