**Views.py**

from django.shortcuts import render , redirect

from . import forms

from django.contrib import messages

from django.conf import settings

from django.core.files.storage import FileSystemStorage

from .models import User\_SigUp

import numpy as np

import torch

from torch.utils.data import Dataset, DataLoader

import glob

import os

import matplotlib.pyplot as plt

from sklearn.metrics import confusion\_matrix,  accuracy\_score

import cv2

import random

import sys

import os

import torch.nn as nn

import torch.nn.functional as F

import seaborn as  sns

# Create your views here.

def SigUp(request):

    if  request.method=='POST':

        form = forms.User\_SigupForm(request.POST)

        if form.is\_valid():

            form.save()

            messages.success(request , 'Account Created Successfully')

        else:

            messages.error(request , 'Invalid Credentails')

    form = forms.User\_SigupForm()

    return render(request , 'Register.html' , {'form':form})

def UserLogin(request):

    if request.method=='POST':

        username = request.POST.get('name')

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

        try:

            data = User\_SigUp.objects.get(Username=username , Password=password)

            if data.Status=='active':

                return redirect('UserHome')

            else:

             messages.error(request , 'You are not activated yet')

        except Exception as e:

            messages.error (request , 'Invalid data')

            return render(request , 'UserLogin.html')

    return render(request , 'UserLogin.html')

def UserHome(request):

    return render(request , 'Users/UserHome.html')

def Traning(request):

        tumor = []

        path = os.path.join(settings.MEDIA\_ROOT , 'brain\_tumor\_dataset' , 'yes' , '\*.jpg')

        for f in glob.glob(path):

            img = cv2.imread(f)

            img = cv2.resize(img,(128 , 128))

            b, g, r = cv2.split(img)

            cv2.merge([r, g, b])

            tumor.append(img)

        healthy = []

        path =os.path.join(settings.MEDIA\_ROOT , 'brain\_tumor\_dataset' , 'no' , '\*.jpg')

        for f in glob.glob(path):

            img = cv2.imread(f)

            img = cv2.resize(img,(128 , 128))

            b, g, r = cv2.split(img)

            cv2.merge([r, g, b])

            healthy.append(img)

        healthy = np.array(healthy)

        tumor = np.array(tumor)

        All= np.concatenate((tumor, healthy))

        plt.imshow(healthy[0])

        plt.axis('off')

        plt.show()

        plt.imshow(tumor[0])

        plt.axis('off')

        plt.show()

        def pot\_random(healthy , tumor , num=5):

            healthy\_images = healthy[np.random.choice(healthy.shape[0], num, replace=False)]

            tumor\_images = tumor[np.random.choice(tumor.shape[0], num, replace=False)]

            plt.figure(figsize=(20, 8))

            for i in range(num):

                plt.subplot(1, num , i+1)

                plt.title('healthy')

                plt.imshow(healthy\_images[i])

                plt.axis('off')

            plt.figure(figsize=(20, 8))

            for j in range(num):

                plt.subplot(1, num , j+1)

                plt.title('tumor')

                plt.imshow(tumor\_images[j])

                plt.axis('off')

            plt.show()

        pot\_random(healthy , tumor)

        class Dataset(object):

            def \_\_getitem\_\_(self, index):

                raise NotImplementedError

            def \_\_len\_\_(self):

                raise NotImplementedError

            def \_\_add\_\_(self, other):

                raise NotImplementedError

        class MRI(Dataset):

            def \_\_init\_\_(self):

                tumor = []

                healthy = []

                path = os.path.join(settings.MEDIA\_ROOT , 'brain\_tumor\_dataset' , 'yes' , '\*.jpg')

                for f in glob.glob(path):

                    img = cv2.imread(f)

                    img = cv2.resize(img,(128 , 128))

                    b, g, r = cv2.split(img)

                    cv2.merge([r, g, b])

                    img = img.reshape(img.shape[2], img.shape[0], img.shape[1])

                    tumor.append(img)

                path = os.path.join(settings.MEDIA\_ROOT , 'brain\_tumor\_dataset' , 'no' , '\*.jpg')

                for f in glob.glob(path):

                    img = cv2.imread(f)

                    img = cv2.resize(img,(128 , 128))

                    b, g, r = cv2.split(img)

                    cv2.merge([r, g, b])

                    img = img.reshape(img.shape[2], img.shape[0], img.shape[1])

                    healthy.append(img)

                #out images

                tumor = np.array(tumor, dtype=np.float32)

                healthy = np.array(healthy, dtype=np.float32)

                #our labels

                tumor\_labels = np.ones(tumor.shape[0], dtype=np.float32)

                healthy\_labels = np.zeros(healthy.shape[0], dtype=np.float32)

            #concatenate

                self.images =np.concatenate((tumor, healthy) , axis=0)

                self.labels = np.concatenate((tumor\_labels, healthy\_labels) , axis=0)

            def \_\_len\_\_(self):

                return self.images.shape[0]

            def \_\_getitem\_\_(self, index):

                sample = {'images': self.images[index], 'labels': self.labels[index]}

                return  sample

            def normalize(self):

                self.images = self.images / 255.0

        class CNN(nn.Module):

            def \_\_init\_\_(self):

                super(CNN, self).\_\_init\_\_()

                self.cnn\_model = nn.Sequential (

                    nn.Conv2d(in\_channels=3, out\_channels=6, kernel\_size=5),

                    nn.Tanh(),

                    nn.AvgPool2d(kernel\_size=2, stride=5),

                    nn.Conv2d(in\_channels=6, out\_channels=16, kernel\_size=5),

                    nn.Tanh(),

                    nn.AvgPool2d(kernel\_size=2, stride=5),

                )

                self.fc\_model = nn.Sequential (

                    nn.Linear(in\_features=256, out\_features=120),

                    nn.Tanh(),

                    nn.Linear(in\_features=120, out\_features=84),

                    nn.Tanh(),

                    nn.Linear(in\_features=84, out\_features=1),

                )

            def forward(self, x):

                x = self.cnn\_model(x)

                x = x.view(x.size(0), -1)

                x = self.fc\_model(x)

                x = F.sigmoid(x)

                return x

        mri\_dataset = MRI()

        mri\_dataset.normalize()

        model = CNN()

        dataloader = DataLoader(mri\_dataset , batch\_size = 32 , shuffle = False)

        model.eval()

        outputs = []

        y\_true = []

        with torch.no\_grad():

            for D in dataloader:

                image = D['images']

                print(image[0].shape)

                label = D['labels']

                y\_hat = model(image)

                outputs.append(y\_hat.cpu().detach().numpy())

                y\_true.append(label.cpu().detach().numpy())

        outputs = np.concatenate(outputs , axis=0).squeeze()

        y\_true  = np.concatenate(y\_true , axis=0).squeeze()

        def threshold(scores , threshold = 0.5 , minimum =0 , maximum = 1):

            x=np.array(list(scores))

            x[x>=threshold]= maximum

            x[x<threshold]= minimum

            return x

        accuracy\_score(y\_true , threshold(outputs))

        plt.figure(figsize=(10,5))

        plt.plot(outputs)

        plt.axvline(x=len(tumor) , color='r' ,linestyle='dashed')

        plt.grid()

        plt.show()

        eta = 0.0001

        EPOCHS = 300

        optimizer = torch.optim.Adam(model.parameters() , lr = eta)

        dataloader = DataLoader(mri\_dataset , batch\_size = 32 , shuffle = True)

        model.train()

        for epoch in range(1 , EPOCHS):

            losses = []

            for D in dataloader:

                optimizer.zero\_grad()

                data = D['images']

                label = D['labels']

                y\_hat = model(data)

                #define losses function

                error = nn. BCELoss()

                loss= torch.sum(error(y\_hat.squeeze() , label))

                loss.backward()

                optimizer.step()

                losses.append(loss.item())

                print('Tain Epoch {} Loss {:.3f}'.format(epoch, np.mean(losses)))

        model.eval()

        dataloader = DataLoader(mri\_dataset , batch\_size = 32 , shuffle = False)

        outputs = []

        y\_true = []

        with torch.no\_grad():

            for D in dataloader:

                image = D['images']

                print(image[0].shape)

                label = D['labels']

                y\_hat = model(image)

                outputs.append(y\_hat.cpu().detach().numpy())

                y\_true.append(label.cpu().detach().numpy())

        outputs = np.concatenate(outputs , axis=0).squeeze()

        y\_true  = np.concatenate(y\_true , axis=0).squeeze()

        acc= accuracy\_score(y\_true , threshold(outputs))

        plt.figure(figsize=(5,5))

        cm = confusion\_matrix(y\_true , threshold(outputs))

        ax=plt.subplot()

        sns.heatmap(cm , annot=True, fmt='g' , ax=ax , annot\_kws={'size':20})

        ax.set\_xlabel('Predicted labels', fontsize=20)

        ax.set\_ylabel('True labels', fontsize=20)

        ax.set\_title('Confusion Matrix' , fontsize=20)

        ax.xaxis.set\_ticklabels(['healthy', 'tumor'], fontsize=15)

        ax.yaxis.set\_ticklabels(['healthy', 'tumor'], fontsize=15)

        plt.show()

        plt.figure(figsize=(10,5))

        plt.plot(outputs)

        plt.axvline(x=len(tumor) , color='r' ,linestyle='dashed')

        plt.grid()

        plt.show()

        return render(request , 'Users/UserTraning.html' , {'acc':acc})

def predict(request):

    if request.method=='POST':

        img = request.FILES.get('img')

        fs = FileSystemStorage(location=settings.MEDIA\_ROOT)

        filename = fs.save(img.name, img)  # Save the uploaded file with its original name

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

        image\_path = os.path.join(settings.MEDIA\_ROOT , filename)

        class CNN(nn.Module):

            def \_\_init\_\_(self):

                super(CNN, self).\_\_init\_\_()

                self.cnn\_model = nn.Sequential (

                    nn.Conv2d(in\_channels=3, out\_channels=6, kernel\_size=5),

                    nn.Tanh(),

                    nn.AvgPool2d(kernel\_size=2, stride=5),

                    nn.Conv2d(in\_channels=6, out\_channels=16, kernel\_size=5),

                    nn.Tanh(),

                    nn.AvgPool2d(kernel\_size=2, stride=5),

                )

                self.fc\_model = nn.Sequential (

                    nn.Linear(in\_features=256, out\_features=120),

                    nn.Tanh(),

                    nn.Linear(in\_features=120, out\_features=84),

                    nn.Tanh(),

                    nn.Linear(in\_features=84, out\_features=1),

                )

            def forward(self, x):

                x = self.cnn\_model(x)

                x = x.view(x.size(0), -1)

                x = self.fc\_model(x)

                x = F.sigmoid(x)

                return x

        model = CNN()

        model.load\_state\_dict(torch.load(os.path.join(settings.MEDIA\_ROOT , 'weights' , 'model.pt')))

        model.eval()

        # Load and process the image

        # image\_path = os.path.join(settings.MEDIA\_ROOT , filename)

        image = cv2.imread(image\_path)

        image\_resized = cv2.resize(image, (128, 128))

        b, g, r = cv2.split(image\_resized)

        image\_resized = cv2.merge([r, g, b])  # Convert BGR to RGB

        # Reshape to match input size and normalize

        image\_input = image\_resized.reshape(1, 3, 128, 128)

        image\_input = torch.from\_numpy(image\_input).float() / 255.0

        # Define the thresholding function

        def threshold(scores, threshold=0.5, minimum=0, maximum=1):

            x = np.array(list(scores))

            x[x >= threshold] = maximum

            x[x < threshold] = minimum

            return x

        # Make prediction

        with torch.no\_grad():

            output = model(image\_input)

        # Apply threshold

        prediction = threshold(output.cpu().numpy())

        # Show the image with the prediction result

        plt.imshow(image\_resized)

        if prediction == 1:

            plt.title("Tumor detected")

        else:

            plt.title("No tumor detected")

        plt.axis('off')  # Hide axis

        plt.show()

    return render(request , 'Users/UserPredict.html')

**Base.html**

<!DOCTYPE html>

<html lang="en">

{% load static %}

<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, shrink-to-fit=no" />

    <!-- Site Metas -->

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

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

    <meta name="author" content="" />

    <title>MRI Brain Tumor Detection| MAIN PAGE</title>

    <!-- slider stylesheet -->

    <link rel="stylesheet" type="text/css"

        href="https://cdnjs.cloudflare.com/ajax/libs/OwlCarousel2/2.1.3/assets/owl.carousel.min.css" />

    <!-- bootstrap core css -->

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

    <!-- fonts style -->

    <link href="https://fonts.googleapis.com/css?family=Open+Sans:400,700|Poppins:400,700|Roboto:400,700&display=swap"

        rel="stylesheet" />

    <!-- Custom styles for this template -->

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

    <!-- responsive style -->

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

    <style>

        body {

            background-image: url("{% static 'Images/hero-bg.png' %}");

        }

    </style>

</head>

<body>

    <div class="hero\_area">

        <!-- header section strats -->

        <header class="header\_section">

          <div class="container-fluid">

            <nav class="navbar navbar-expand-lg custom\_nav-container pt-3">

              <a class="navbar-brand" href="{% url 'home' %}">

                <img src="{% static 'Images/logo.png' %}" alt="" />

                <span>

                    Mri Brain Tumor Detection

                </span>

              </a>

     </nav>

     </div>

     </header>

     </div>

    <section class="about\_section layout\_padding mt-5">

        <div class="container">

            <div class="row">

                <div class="col-md-5">

                    <div class="img-box">

                        <img src="{% static 'Images/bri.jpeg'%}" alt="" />

                    </div>

                </div>

                <div class="col-md-6">

                    <div class="detail-box">

                        <div>

                            <a href="{% url 'home'%}"  style="text-decoration: none;"> Home</a>

                            <a href="{% url 'adminlogin' %}" style="text-decoration: none;"> Admin</a>

                            <a href="{% url 'UserLogin' %}" style="text-decoration: none;">User</a>

                            <a href="{% url 'userlogin' %}" style="text-decoration: none;">Sigup</a>

                        </div><br>

                        {%  block contents %}

                        <div class="custom\_heading-container">

                            <h2>

                                Tumor Detection

                            </h2>

                        </div>

                        <p>

                            Largely, brain-based illnesses are prevalent in

                            society. The main reason brain diseases occur is because of cell

                            growth. It therefore impacts the brain's regular operation,

                            which in turn impacts the health of other essential organs'

                            functionality. Aggressive brain cancer is ultimately the outcome

                            of cell growth. Early detection of tumors in the brain is one of

                            the primary strategies to lower the number of fatalities

                        </p>

                       {% endblock %}

                    </div>

                </div>

            </div>

        </div>

    </section>

<footer style="background-color: black ; color: white; text-align: center; margin-top: 50px;" >

    <div class="container">

        <div class="social-box" style="padding-top: 4px;">

            <a href="">

                <img src="{% static 'Images/fb.png' %}" alt="">

            </a>

            <a href="">

                <img src="{% static 'Images/twitter-logo.png' %}" alt="">

            </a>

            <a href="">

                <img src="{% static 'Images/linkedin-sign.png' %}" alt="">

            </a>

            <a href="">

                <img src="{% static 'Images/instagram.png' %}" alt="">

            </a>

        </div>

        <br>

        <p >Copyright &copy; 2022. All rights reserved.</p>

    </div>

</footer>

</body>

</html>