from flask import Flask, render\_template, request, session

import torch

import torchvision

from torchvision import transforms

from torch.utils.data import DataLoader

from torch.utils.data.dataset import Dataset

from torchvision import models

from torch.autograd import Variable

import os

import numpy as np

import cv2

import matplotlib.pyplot as plt

import face\_recognition

import time

import sys

from torch import nn

app = Flask(\_\_name\_\_)

app.secret\_key = 'my\_key'

class Model(nn.Module):

def \_\_init\_\_(self, num\_classes,latent\_dim= 2048, lstm\_layers=1 , hidden\_dim = 2048, bidirectional = False):

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

model = models.resnext50\_32x4d(pretrained = True)

self.model = nn.Sequential(\*list(model.children())[:-2])

self.lstm = nn.LSTM(latent\_dim,hidden\_dim, lstm\_layers, bidirectional)

self.relu = nn.LeakyReLU()

self.dp = nn.Dropout(0.4)

self.linear1 = nn.Linear(2048,num\_classes)

self.avgpool = nn.AdaptiveAvgPool2d(1)

def forward(self, x):

batch\_size,seq\_length, c, h, w = x.shape

x = x.view(batch\_size \* seq\_length, c, h, w)

fmap = self.model(x)

x = self.avgpool(fmap)

x = x.view(batch\_size,seq\_length,2048)

x\_lstm,\_ = self.lstm(x,None)

return fmap,self.dp(self.linear1(x\_lstm[:,-1,:]))

im\_size = 112

mean=[0.485, 0.456, 0.406]

std=[0.229, 0.224, 0.225]

sm = nn.Softmax()

inv\_normalize = transforms.Normalize(mean=-1\*np.divide(mean,std),std=np.divide([1,1,1],std))

def im\_convert(tensor):

""" Display a tensor as an image. """

image = tensor.to("cpu").clone().detach()

image = image.squeeze()

image = inv\_normalize(image)

image = image.numpy()

image = image.transpose(1,2,0)

image = image.clip(0, 1)

# cv2.imwrite('./2.png',image\*255)

return image

def predict(model,img,path = './'):

fmap,logits = model(img.to('cpu'))

params = list(model.parameters())

weight\_softmax = model.linear1.weight.detach().cpu().numpy()

logits = sm(logits)

\_,prediction = torch.max(logits,1)

confidence = logits[:,int(prediction.item())].item()\*100

print('confidence of prediction:',logits[:,int(prediction.item())].item()\*100)

idx = np.argmax(logits.detach().cpu().numpy())

bz, nc, h, w = fmap.shape

out = np.dot(fmap[-1].detach().cpu().numpy().reshape((nc, h\*w)).T,weight\_softmax[idx,:].T)

predict = out.reshape(h,w)

predict = predict - np.min(predict)

predict\_img = predict / np.max(predict)

predict\_img = np.uint8(255\*predict\_img)

out = cv2.resize(predict\_img, (im\_size,im\_size))

heatmap = cv2.applyColorMap(out, cv2.COLORMAP\_JET)

img = im\_convert(img[:,-1,:,:,:])

result = heatmap \* 0.5 + img\*0.8\*255

cv2.imwrite('/content/1.png',result)

result1 = heatmap \* 0.5/255 + img\*0.8

r,g,b = cv2.split(result1)

result1 = cv2.merge((r,g,b))

# plt.imshow(result1)

# plt.show()

return [int(prediction.item()),confidence]

class validation\_dataset(Dataset):

def \_\_init\_\_(self,video\_names, sequence\_length, transform = None):

self.video\_names = video\_names

self.transform = transform

self.count = sequence\_length

def \_\_len\_\_(self):

return len(self.video\_names)

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

video\_path = self.video\_names[idx]

frames = []

a = int(100/self.count)

first\_frame = np.random.randint(0,a)

for i,frame in enumerate(self.frame\_extract(video\_path)):

#if(i % a == first\_frame):

faces = face\_recognition.face\_locations(frame)

try:

top,right,bottom,left = faces[0]

frame = frame[top:bottom,left:right,:]

except:

pass

frames.append(self.transform(frame))

if(len(frames) == self.count):

break

#print("no of frames",len(frames))

frames = torch.stack(frames)

frames = frames[:self.count]

return frames.unsqueeze(0)

def frame\_extract(self,path):

vidObj = cv2.VideoCapture(path)

success = 1

while success:

success, image = vidObj.read()

if success:

yield image

@app.route('/')

def index():

return render\_template('LandingPage.html')

@app.route('/index')

def second():

scrollValueText = 10

return render\_template('uploader.html', scrollValueText = scrollValueText)

@app.route('/upload', methods=['POST'])

def upload():

fileReader = request.files['file']

scroll\_value = int(request.form['scrollValue'])

fileReader.save('./static/video/' + fileReader.filename)

path\_to\_videos= ["./static/video/" + fileReader.filename]

print("This is the Path ", path\_to\_videos[0])

session['video\_filename'] = fileReader.filename

train\_transforms = transforms.Compose([

transforms.ToPILImage(),

transforms.Resize((im\_size,im\_size)),

transforms.ToTensor(),

transforms.Normalize(mean,std)])

pathProvider = path\_to\_videos[0]

video\_dataset = validation\_dataset(path\_to\_videos,sequence\_length = scroll\_value,transform = train\_transforms)

device = torch.device('cpu')

model = Model(2).to(device)

path\_to\_model = './models/model\_90\_acc\_60\_frames\_final\_data.pt'

model.load\_state\_dict(torch.load(path\_to\_model, device))

model.eval()

predictions = ""

for i in range(0,len(path\_to\_videos)):

print(path\_to\_videos[i])

prediction = predict(model,video\_dataset[i],'./')

accuracy = prediction[1]

print("This is me ", accuracy)

if prediction[0] == 1:

prediction = "REAL"

else:

prediction = "FAKE"

cap = cv2.VideoCapture(path\_to\_videos[0])

total\_frames = int(cap.get(cv2.CAP\_PROP\_FRAME\_COUNT))

frame\_interval = total\_frames // int(scroll\_value)

print(scroll\_value)

frame\_count = 0

frame\_index = 0

frame\_path = []

face\_index = 0

face\_path = []

while cap.isOpened():

ret, frame = cap.read()

if not ret:

break

if frame\_count % frame\_interval == 0:

frame\_path.append('./static/images/'+f'frame\_{frame\_index}.jpg')

output\_path = os.path.join('./static/images/', f'frame\_{frame\_index}.jpg')

# Convert the frame to RGB for face\_recognition library

frame\_rgb = cv2.cvtColor(frame, cv2.COLOR\_BGR2RGB)

# Detect faces in the frame

face\_locations = face\_recognition.face\_locations(frame\_rgb)

# Draw bounding boxes and labels around the faces

for (top, right, bottom, left) in face\_locations:

face\_image = frame[top:bottom, left:right]

face\_path.append('./static/images/'+f'face\_{face\_index}.jpg')

face\_output\_path = os.path.join('./static/images/', f'face\_{face\_index}.jpg')

cv2.imwrite(face\_output\_path, face\_image)

face\_index += 1

if prediction == 'REAL':

cv2.rectangle(frame, (left, top), (right, bottom), (0, 255, 0), 2)

else:

cv2.rectangle(frame, (left, top), (right, bottom), (0, 0, 255), 2)

# Add a label to the bounding box

label = f'{prediction}'

font = cv2.FONT\_HERSHEY\_SIMPLEX

font\_scale = 1.5

text\_size = cv2.getTextSize(label, font, font\_scale, 1)[0]

text\_left = left + 5

text\_top = top - text\_size[1] - 5

if prediction == 'REAL':

cv2.rectangle(frame, (text\_left - 5, text\_top - 5), (text\_left + text\_size[0] + 5, text\_top + text\_size[1] + 5), (0, 255, 0), cv2.FILLED)

else:

cv2.rectangle(frame, (text\_left - 5, text\_top - 5), (text\_left + text\_size[0] + 5, text\_top + text\_size[1] + 5), (0, 0, 255), cv2.FILLED)

cv2.putText(frame, label, (text\_left, text\_top + text\_size[1]), font, font\_scale, (0, 0, 0), 1, cv2.LINE\_AA)

cv2.imwrite(output\_path, frame)

frame\_index += 1

frame\_count += 1

cap.release()

return render\_template('results.html',prediction=prediction, accuracy=accuracy, frame\_path=frame\_path, video\_path= '.'+pathProvider, face\_path=face\_path)

if \_\_name\_\_ == "\_\_main\_\_":

app.run(debug=True)