Team Id	PNT2022TMID39414
Project Name	Natural Disaster Intensity Analysis and Classification Using Artificial Intelligence

Natural Disaster Intensity Analysis and Classification Using Artificial Intelligence

Application Building

In this section, we will be building a web application that is integrated into the model we built. A UI is provided for the uses where he has uploaded an image. The uploaded image is given to the saved model and prediction is showcased on the UI.
This section has the following tasks

. Building HTML Pages

. Building server-side script

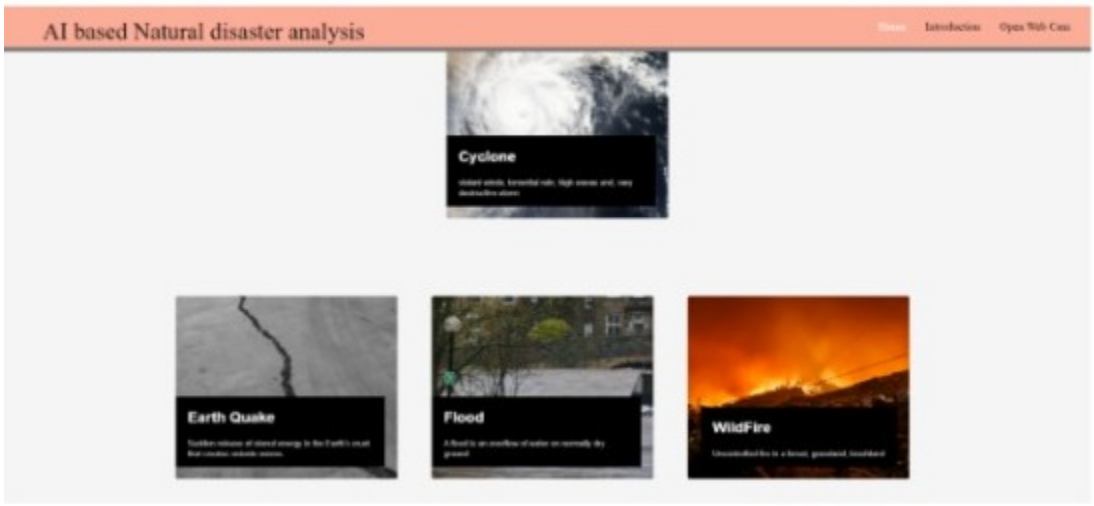
Build HTML Pages

We use HTML to create the front end part of the web page.

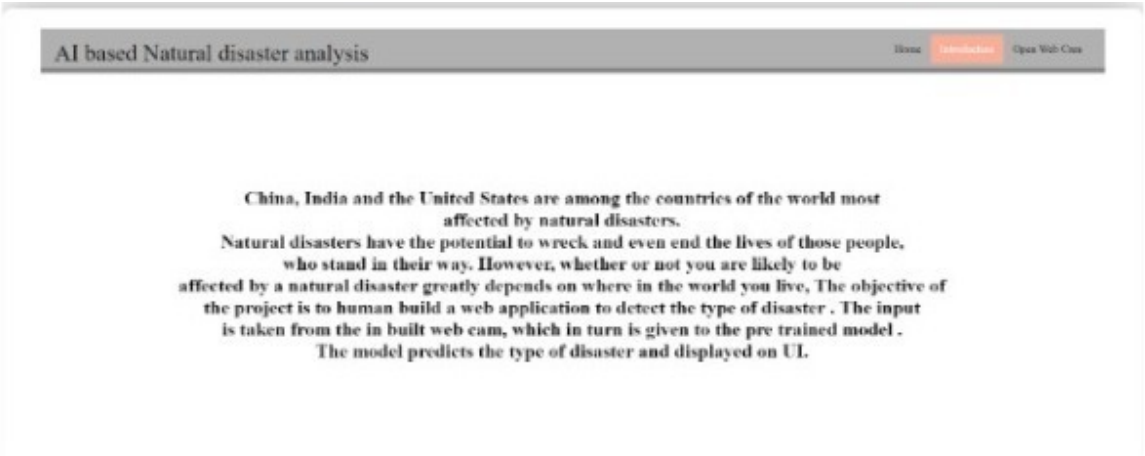
- Here, we have created 3 HTML pages- home.html, intro.html, and upload.html
- home.html displays the home page.
- Intro.html displays an introduction about the project
- upload.html gives the emergency alert

- We also use JavaScript-main.js and CSS-main.css to enhance our functionality and view of HTML pages.
    - Link :CSS , JS

Home.html looks like this



Intro.html



Upload.html

Build Python Code

- Let us build the flask file ‘app.py’ which is a web framework written in python for server-side scripting. Let’s see the step by step procedure for building the backend application.
- The app starts running when the “__name__” constructor is called in main.
- render_template is used to return an HTML file.
- “GET” method is used to take input from the user.
- “POST” method is used to display the output to the user.

The first step is usually importing the libraries that will be needed in the program

```
from flask import Flask,render_template,request
# Flask-It is our framework which we are going to use to r
#request-for accessing file which was uploaded by the user
#import operator
import cv2 # opencv library
from tensorflow.keras.models import load_model#to load our
import numpy as np
#import os
from werkzeug.utils import secure_filename
```

Task 2: Creating our flask application and loading our model by using load_model method

```
app = Flask(__name__,template_folder="templates") # initializ
# Loading the model
model=load_model('disaster.h5')
print("Loaded model from disk")
```

Task 3: Routing to the html Page

Here, the declared constructor is used to route to the HTML page created earlier.

In the above example, the '/' URL is bound with the home.html function. Hence, when the home page of the web server is opened in the browser, the HTML page is rendered. Whenever you enter the values from the HTML page the values can be retrieved using the POST Method.
Here, "home.html" is rendered when the home button is clicked on the UI.

```
@app.route('/', methods=['GET'])
def index():
    return render_template('home.html')
@app.route('/home', methods=['GET'])
def home():
    return render_template('home.html')
@app.route('/intro', methods=['GET'])
def about():
    return render_template('intro.html')
```

When "Open Web Cam " is clicked on the UI, predict function is executed

```
@app.route('/upload', methods=['GET', 'POST'])
def predict():
```

And the upload route is used for prediction and it contains all the codes which are used for predicting our results.

The tasks involved are

- Grab the frames from the web cam.

- Loop over the frames from the video stream
- Convert the image from BGR to RGB
- Predicting our results
- Displaying the result
- Run the application

Grab the frames from the webcam

To recognize the type of disaster we have to capture the video stream. There are two ways we can capture the input video

1.using in-built webcam

2. using video file residing on the disk

```
cap = cv2.VideoCapture(0)
while True:
    _, frame = cap.read() #capturing the video frame values
    # Simulating mirror image
    frame = cv2.flip(frame, 1)
```

Loop over the frames from the video stream

Let us grab the video frames from the video by looping over the frames and check if the frame was not grabbed, then we have reached the end of the stream. clone the output frame for showcasing the output

```
loop over frames from the video file stream
        while True:
            # read the next frame from the file
            (grabbed, frame) = vs.read()

            # if the frame was not grabbed, then we have reac
            # of the stream
            if not grabbed:
                break

            # if the frame dimensions are empty, grab them
            if W is None or H is None:
                (H, W) = frame.shape[:2]

            # clone the output frame, then convert it from BG
            # ordering and resize the frame to a fixed 224x22
            output = frame.copy()
```

Convert the image from BGR to RGB

The frame we have captured is a 3-channel RGB colored image.

convert it from BGR to RGB , resize the frame to a fixed 64x64 and expand the dimensions to give it to the model for prediction.

```
frame = cv2.cvtColor(frame, cv2.COLOR_BGR2RGB)
frame = cv2.resize(frame, (64, 64))
#frame = frame.astype("float32")
x=np.expand_dims(frame, axis=0)
```

predicting the results

We then proceed to detect all types of disaster in the input image using model.predict function and the result is stored in the result variable.

```
result = np.argmax(model.predict(x), axis=-1)
index=['Cyclone','Earthquake','Flood','Wildfire']
result=str(index[result[0]])
#print(result)
#result=result.tolist()
```

Displaying the result

After we recognize the type of disaster, we have to display the same on the live video stream for visualization.

The cv2.imshow() function always takes two more functions to load and close the image. These two functions are cv2.waitKey() and cv2.destroyAllWindows(). Inside the cv2.waitKey() function, you can provide any value to close the image and continue with further lines of code.

```
    cv2.putText(output, "activity: {}".format(result), (10, 120), cv2.FONT_HERSHEY_PLAIN,
                1, (0,255,255), 1)
    #playaudio("Emergency it is a disaster")
    cv2.imshow("Output", output)
    key = cv2.waitKey(1) & 0xFF

        # if the `q` key was pressed, break from the loop
    if key == ord("q"):
        break

# release the file pointers
print("[INFO] cleaning up...")
vs.release()
cv2.destroyAllWindows()
return render_template("upload.html")
```

Run the application

This is used to run the application in a local host. The local host runs on port number 8000.(We can give different port numbers)

```python
if __name__ == '__main__':
    app.run(host='0.0.0.0', port=8000, debug=False)
```

- Open the anaconda prompt from the start menu.
- Navigate to the folder where your app.py resides.
- Now type “python app.py” command.
- It will show the local host where your app is running on http://127.0.0.1.8000/
- Copy that local host URL and open that URL in the browser. It does navigate me to where you can view your web page.
- Enter the values, click on the predict button and see the result/prediction on the web page.

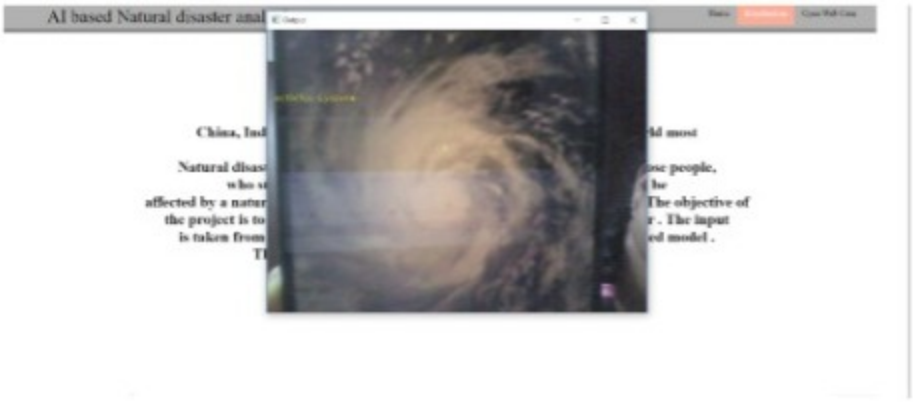
```
(base) D:\ML_training may 2020\Projects_50\Final\AI based Natural disaster analysis\Flask>
(base) D:\ML_training may 2020\Projects_50\Final\AI based Natural disaster analysis\Flask>python app.py
```

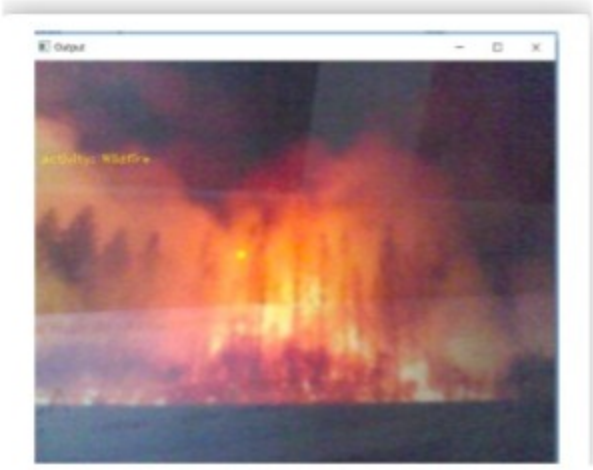
Then it will run on localhost:8000

```
* Serving Flask app "app" (lazy loading)
* Environment: production
  WARNING: This is a development server. Do not use it in a production deployment.
  Use a production WSGI server instead.
* Debug mode: off
* Running on http://127.0.0.1:5000/ (Press CTRL+C to quit)
```

Navigate to the localhost (http://127.0.0.1:8000/)where you can view your web page. Click on the open webcam and then you can see another spyder window which is opened to view the opened webcam.

Output screenshots:


AI based Natural disaster anal
Output
China, Ind
ld most
Natural disas
ose people,
who s
be
affected by a natur
The objective of
the project is to
r . The input
is taken from
ed model .
T


Output