

PROJECT DEVELOPMENT PHASE

SPRINT-III

Date	8 November 2022
Team ID	PNT2022TMID16547
Project Name	Natural Disaster Intensity Analysis and Classification using Artificial Intelligence

DETECTION AND ANALYSIS OF DATA:

After Testing and Training the model, data which given in dataset are analysed and visualised effectively to detect the Disaster Type. Using webcam, it can capture image or video stream of Disaster, to detect and analyse the type of Disaster.

```
✓ 1 print(x_train.class_indices)
   {'Cyclone': 0, 'Earthquake': 1, 'Flood': 2, 'Wildfire': 3}

✓ [24] from collections import Counter as c
      c(x_train.labels)

      Counter({0: 220, 1: 156, 2: 198, 3: 168})
```

IMAGE PREPROCESSING:

Image Pre-processing was done for Disaster intensity analysis and classification with three main tasks which includes for pre-processing of Images,

- Import ImageDataGenerator Library.
- Configure ImageDataGenerator Class.
- Applying ImageDataGenerator functionality to the trainset and test set.

```
[ ] #Configuring image Data Generator Class

#Setting Parameter for Image Augmentation for training data

train_datagen = ImageDataGenerator(rescale = 1./255, shear_range = 0.2, zoom_range = 0.2, horizontal_flip = True)

#Image Data Augmentation for testing data

test_datagen = ImageDataGenerator(rescale = 1./255)
```

IMPORTING THE IMAGEDATAGENERATOR LIBRARY:

By importing the ImageDataGenerator Library can expand the trainset data size using modified versions of dataset.

ImageDataGenerator class were importing from keras.

```
#importing image data generator library
[ ] from keras.preprocessing.image import ImageDataGenerator
```

CONFIGURE IMAGEDATAGENERATOR CLASS:

ImageDataGenerator class is instantiated and the configuration for the types of data augmentation.

An instance of the ImageDataGenerator class can be constructed for train and test dataset by ImageDataGenerator class.

```
[ ] #Configuring image Data Generator Class
#Setting Parameter for Image Augmentation for training data
train_datagen = ImageDataGenerator(rescale = 1./255, shear_range = 0.2, zoom_range = 0.2, horizontal_flip = True)
#Image Data Augmentation for testing data
test_datagen = ImageDataGenerator(rescale = 1./255)
```

APPLYING IMAGEDATAGENERATOR FUNCTIONALITY TO TRAINSET AND TESTSET

:

ImageDataGenerator functionality was applied to Trainset and Testset by using the following code,

“For Training set using flow_from_directory function”.

```
Apply ImageDataGenerator Functionality To Trainset And Testset
#Performing data augmentation to train data
x_train = train_datagen.flow_from_directory('/content/dataset/train_set', target_size = (64,64), batch_size = 5, color_mode = 'rgb', class_mode = 'categorical')
#performing data augmentation to test data
x_test = test_datagen.flow_from_directory('/content/dataset/test_set', target_size = (64,64), batch_size = 5, color_mode = 'rgb', class_mode = 'categorical')
Found 742 images belonging to 4 classes.
Found 198 images belonging to 4 classes.
```

MODEL BUILDING:

Building a Model with web application named “FLASK”, model building process consist several steps like,

- Import the model building Libraries
- Initializing the model
- Adding CNN Layers
- Adding Hidden Layer
- Adding Output Layer
- Configure the Learning Process
- Training and testing the model all the above processes are

done and saved in a model.

```
[ ] # initialising the model and adding CNN layers

model = Sequential()

# First convolution layer and pooling
model.add(Conv2D(32,(3,3),input_shape=(64,64,3),activation='relu'))
model.add(MaxPooling2D(pool_size=(2,2)))

#Second convolution layer and pooling
model.add(Conv2D(32,(3,3),activation='relu'))
model.add(MaxPooling2D(pool_size=(2,2)))

#Flattening the layers
model.add(Flatten())

#Adding Dense Layers
model.add(Dense(units=128,activation='relu'))
model.add(Dense(units=4,activation='softmax'))

[ ] # Summary of our model

model.summary()
```

```
[26] # Save the model

model.save('disaster.h5')
model_json = model.to_json()
with open("model-bw.json", "w") as json_file:
    json_file.write(model_json)
```

CREATING app.py:

```
10
11 # import the necessary packages
12 from flask import Flask,render_template,request
13 # Flask-It is our framework which we are going to use to run/serve our application.
14 #request-for accessing file which was uploaded by the user on our application.
15 #import operator
16 import cv2 # opencv library
17 from tensorflow.keras.models import load_model#to load our trained model
18 import numpy as np
19 #import os
20 from werkzeug.utils import secure_filename
21 #from playsound import playsound
22 #from gtts import gTTS
23 '''
24 def playaudio(text):
25     speech=gTTS(text)
26     print(type(speech))
27     speech.save("output1.mp3")
28     playsound("output1.mp3")
29     return
30 '''
31 app = Flask(__name__,template_folder="templates") # initializing a flask app
32 # Loading the model
33 model=load_model(r'C:\Users\user\Desktop\IBM\Flask\templates\disaster.h5')
34 print("Loaded model from disk")
35
36
37 app=Flask(__name__,template_folder="templates")
38 @app.route('/', methods=['GET'])
39 def index():
40     return render_template('home.html')
41 @app.route('/home', methods=['GET'])
42 def home():
43     return render_template('home.html')
44 @app.route('/intro', methods=['GET'])
45 def about():
46     return render_template('intro.html')
47 @app.route('/upload', methods=['GET', 'POST'])
```