PROJECT DEVELOPMENT PHASE

SPRINT-III

Date	17th November 2022
Team ID	PNT2022TMID38828
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.

IMAGE PREPROCESSING:

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

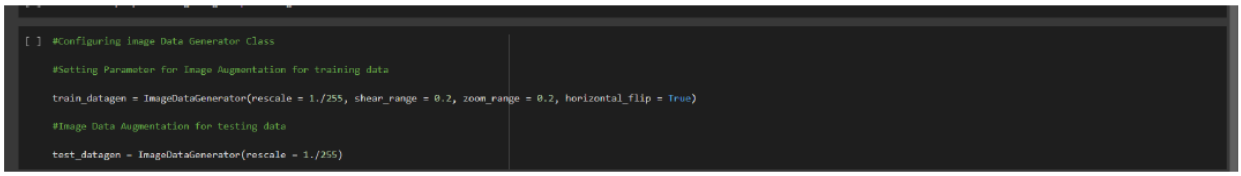
- Import ImageDataGenerator Library.
- Configure ImageDataGenerator Class.
- Applying ImageDataGenerator functionality to the trainset and test set.

```
[ ] #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.

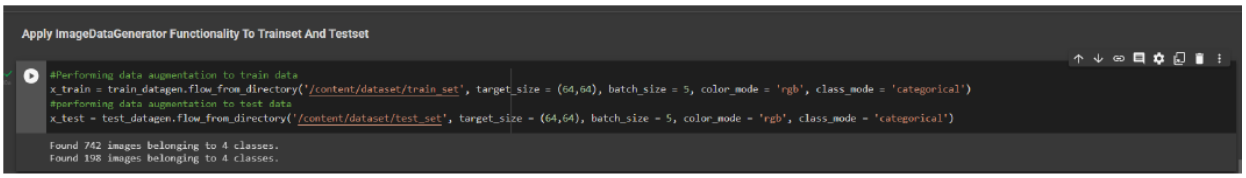


CONFIGURE IMAGEDATAGENERATOR CLASS:

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.

APPLYING



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”.

```
[ ] # 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)
```

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.

```python
# import the necessary packages
from flask import Flask,render_template,request
# Flask-It is our framework which we are going to use to run/serve our application.
#request-for accessing file which was uploaded by the user on our application.
#import operator
import cv2 # opencv library
from tensorflow.keras.models import load_model#to load our trained model
import numpy as np
#import os
from werkzeug.utils import secure_filename
#from playsound import playsound
#from gtts import gTTS
'''
def playaudio(text):
    speech=gTTS(text)
    print(type(speech))
    speech.save("output1.mp3")
    playsound("output1.mp3")
    return
'''
app = Flask(__name__,template_folder="templates") # initializing a flask app
# Loading the model
model=load_model(r'C:\Users\user\Desktop\IBM\Flask\templates\disaster.h5')
print("Loaded model from disk")


app=Flask(__name__,template_folder="templates")
@app.route('/', methods=['GET'])
def index():
```

```python
app=Flask(__name__,template_folder="templates")
@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')
@app.route('/upload', methods=['GET', 'POST'])
```

CREATING app.py :

```python
app=Flask(__name__,template_folder="templates")
@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')
@app.route('/upload', methods=['GET', 'POST'])
```