PROJECT TITLE: AI BASED NATURAL DISASTER INTENSITY ANALYSIS USING IBM WATSON

1.INTRODUCTION

1.1 Overview

For quick and efficient response, as well as for recovery after any natural or artificial catastrophe, one of the most important things are accurate and reliable spatial data in real or near real-time. It is essential to know the location as well as to track and analyse passive and active threats to quickly identify the possible dangers and hazards. As technology evolves and advances, there is a broader spectrum of sensors that provide spatial data, and nowadays, decision-making processes also include nontraditional, informal sources of information. Apart from the offer, demand for new spatial data is increasing as well. For quicker and enhanced integration and analysis of data, artificial intelligence (AI) tools are increasingly used which, in addition to immediate rapid reactions, can help to make better and smarter decisions in the future. Such software algorithms that imitate human intelligence can help in generating conclusions from natural phenomena presented by spatial data. Using AI in the data analysis can identify risk areas and determine future needs. This paper presents an overview of the use of AI in geospatial analysis in disaster management.

1.2 Purpose

Disaster can be caused by naturally occurring events such as earthquakes, cyclones, floods, and wildfires. Many deep learning techniques have been applied by various researchers to detect and classify natural disasters to overcome losses in ecosystems, but detection of natural disasters still faces issues due to the complex and imbalanced structures of images. To tackle this problem, we developed a multilayered deep convolutional neural network model that classifies the natural disaster and tells the intensity of disaster of natural The model uses an integrated webcam to capture the video frame and the video frame is compared with the Pre-trained model and the type of disaster is identified and showcased on the OpenCV window

2. LITERATURE SURVEY

2.1 Existing problem

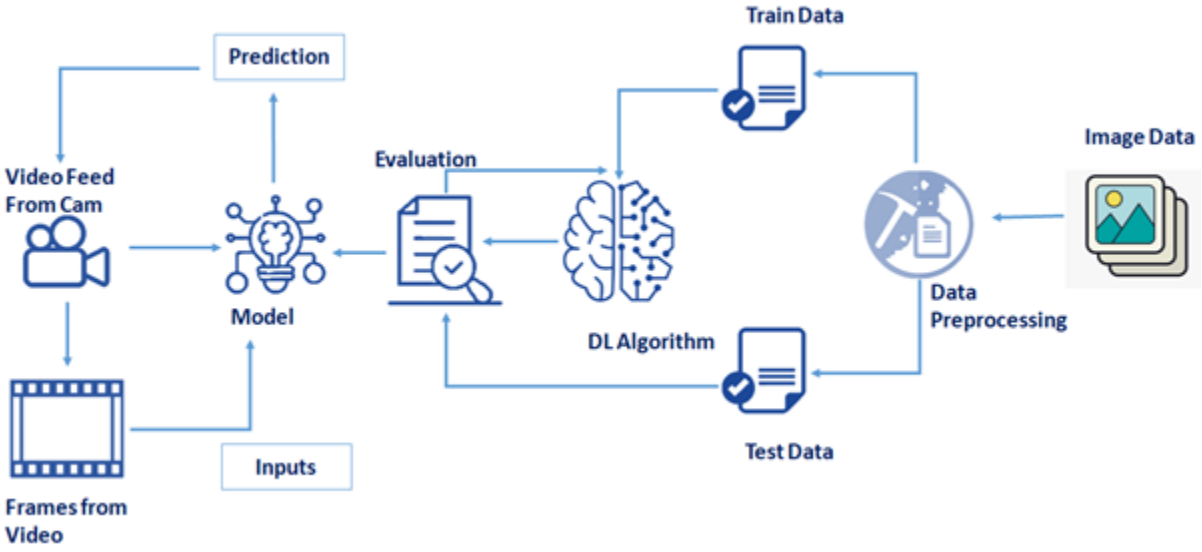
Natural disasters not only disturb the human ecological system but also destroy the properties and critical infrastructures of human societies and even lead to permanent change in the ecosystem.

2.2 Proposed Solution

we developed a multilayered deep convolutional neural network model that classifies the natural disaster and tells the intensity of disaster . The model uses an integrated webcam to capture the video frame and the video frame is compared with the Pre-trained model and the type of disaster is identified and showcased on the OpenCV window.

3.THEORITICAL ANALYSIS

3.1 Block Diagram



3.2 Hardware/Software designing

To complete this project, you must require the following software's, concepts, and packages

Anaconda Navigator is a free and open-source distribution of the Python and R programming languages for data science and machine learning related applications. It can be installed on Windows, Linux, and macOS.Conda is an open-source, cross-platform, package management system. Anaconda comes with so very nice tools like JupyterLab, Jupyter Notebook,

QtConsole, Spyder, Glueviz, Orange, Rstudio, Visual Studio Code. For this project, we will be using Jupyter notebook and spyder

To build Machine learning models you must require the following packages

Numpy:

- It is an open-source numerical Python library. It contains a multidimensional array and matrix data structures and can be used to perform mathematical operations

Scikit-learn:

- It is a free machine learning library for Python. It features various algorithms like support vector machine, random forests, and k-neighbors, and it also supports Python numerical and scientific libraries like NumPy and SciPy

OpenCV

- OpenCV is a library of programming functions mainly aimed at real-time computer vision. Here, OpenCV is used to capture frames by accessing the webcam in real-time.
- Open anaconda prompt and type command
  "pip install opencv-contrib-python"

Flask:

Web framework used for building Web applications

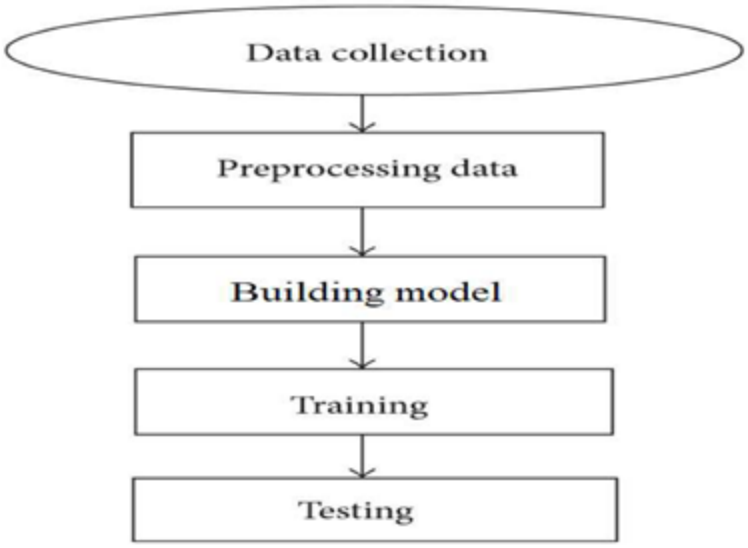
Python packages:

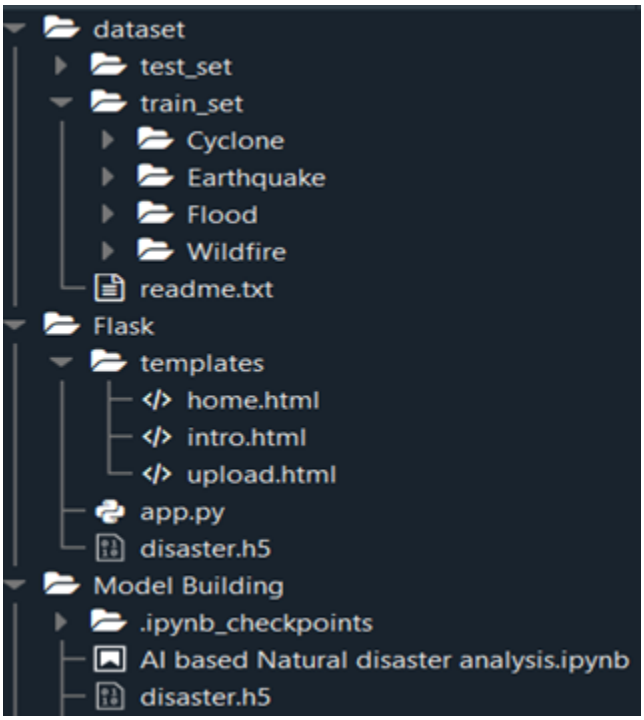
- open anaconda prompt as administrator
- Type "pip install numpy" and click enter.
- Type "pip install pandas" and click enter.
- Type "pip install scikit-learn" and click enter.
- Type "pip install opencv-contrib-python" and click enter.
- Type "pip install tensorflow==2.3.0" and click enter.
- Type "pip install keras==2.4.0" and click enter.
- Type "pip install Flask" and click enter.

4.EXPERIMENTAL INVESTIGATIONS

Natural disaster prediction of our proposed system consist of 2 phases; training phase and testing phase. The aerial images of disaster is taken from webcam is taken as input images.

5.FLOWCHART

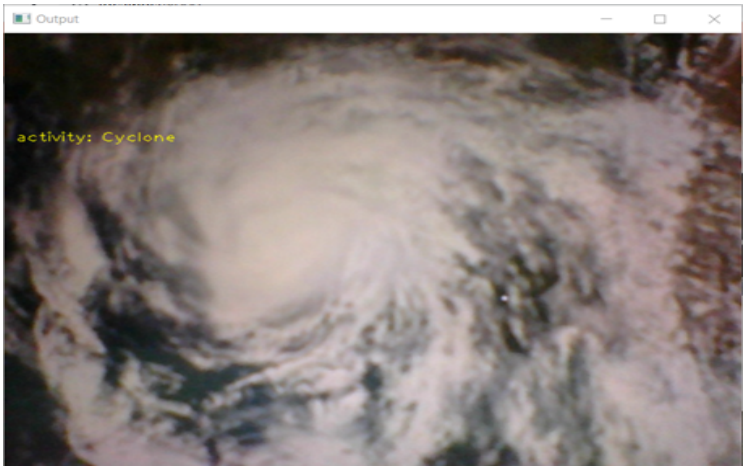


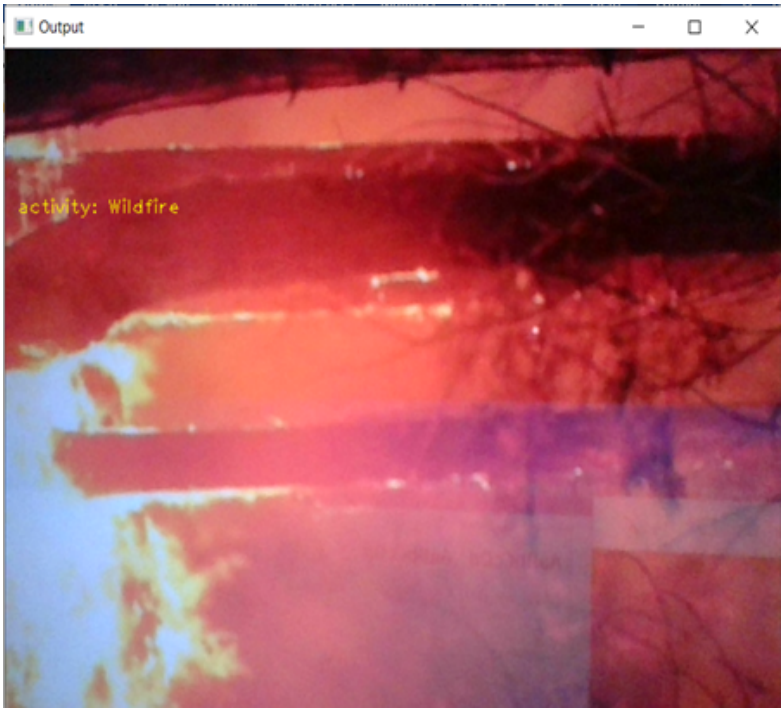


- Dataset folder contains the training and testing images for training our model.
- We are building a Flask Application that needs HTML pages stored in the templates folder and a python script app.py for serverside scripting
- we need the model which is saved and the saved model in this content is a disaster.h5
- templates folder contains home.html,intro.html,upload.html pages.

6.RESULT

These are the final outputs obtained from the project





7. ADVANTAGES AND DISADVANTAGES

Advantages:

- Ability to prepare belongings and evacuate areas.
- Allows for life to be saved -human and animal.
- Evacuation centres can be organised and prepared.

Disadvantages:

- No concrete evidence to prove a disaster is about to occur.
- Disaster may occur within a certain radius of the prediction and false preparation or lower magnitude disaster could occur.

8.APPLICATIONS

Predicting the time, location and magnitude of an earthquake is a challenging job as an earthquake does not show specific patterns resulting in inaccurate predictions. Techniques based on Artificial Intelligence (AI) are well known for their capability to find hidden patterns in data. In the case of earthquake prediction, these models also produce a promising outcome. This work systematically explores the contributions made to date in earthquake prediction using AI-based techniques.

9. CONCLUSION

The proposed system helps to reduce the impact of hazards occur during natural disaster. This provides an efficient way to warn and educate people about disaster prone areas. Natural disaster possibility prediction helps to minimize loss of lives and loss of damages to mankind and the nature.

10.FUTURE SCOPE

One step ahead of IoT stands AI -smart technology, which has enabled accurate and speedy solutions. If harnessed properly, the technology has the potential of predicting, preventing and providing response faster than ever.

AI data setups are trained to predict seismic data to analyze the patterns of earthquake occurrences, rainfall records and monitor flooding, measure the intensity of hurricanes and read the geological data to understand volcanic eruptions, such systems can reduce the catastrophic impact of natural disasters.

11.BIBILOGRAPHY

1. Ma, C.-K.; Awang, A.Z.; Omar, W. Structural and material performance of geopolymer concrete: A review. Constr. Build. Mater. 2018, 186, 90–102. [CrossRef]

2. Singh, B.; Ishwarya, G.; Gupta, M.; Bhattacharyya, S.K. Geopolymer concrete: A review of some recent developments. Constr. Build. Mater. 2015, 85, 78–90. [CrossRef]

APPENDIX

SOURCECODE:

In [1]:
```
import numpy as np
import tensorflow
from tensorflow.keras.models import Sequential
from tensorflow.keras import layers

from tensorflow.keras.layers import Dense,Flatten

from tensorflow.keras.layers import Conv2D,MaxPooling2D
from keras.preprocessing.image import ImageDataGenerator
```

```
Using TensorFlow backend.
```

In [2]:
```
tensorflow.__version__
```

Out[2]: '1.14.0'

In [3]:
```
tensorflow.keras.__version__
```

Out[3]: '2.2.4-tf'

Image Data Agumentation

In [4]:
```
train_datagen = ImageDataGenerator(rescale=1./255,shear_range=0.2,zoom_range=0.2,horizontal_flip=True)
test_datagen=ImageDataGenerator(rescale=1./255)
```

Loading our data and performing data agumentation

In [5]:
```
x_train = train_datagen.flow_from_directory(r"C:\Users\GOPAGONI HARIKA\Desktop\dataset\train_set",target_size=(64, 64),batch_siz
                                            color_mode='rgb',class_mode="categorical")

x_test = test_datagen.flow_from_directory(r"C:\Users\GOPAGONI HARIKA\Desktop\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.
```

In [6]:
```
print(x_train.class_indices)
```

```
{'Cyclone': 0, 'Earthquake': 1, 'Flood': 2, 'Wildfire': 3}
```

In [7]:
```
print(x_test.class_indices)
```

```
{'Cyclone': 0, 'Earthquake': 1, 'Flood': 2, 'Wildfire': 3}
```

In [8]:
```
from collections import Counter as c
c(x_train .labels)
```

Out[8]: Counter({0: 220, 1: 156, 2: 198, 3: 168})

creating the model

In [9]:
```
# Initializing the CNN
classifier = Sequential()

# First convolution layer and pooling
classifier.add(Conv2D(32, (3, 3), input_shape=(64, 64, 3), activation='relu'))
classifier.add(MaxPooling2D(pool_size=(2, 2)))
# Second convolution layer and pooling
classifier.add(Conv2D(32, (3, 3), activation='relu'))
# input_shape is going to be the pooled feature maps from the previous convolution layer
classifier.add(MaxPooling2D(pool_size=(2, 2)))

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

# Adding a fully connected layer
classifier.add(Dense(units=128, activation='relu'))
classifier.add(Dense(units=4, activation='softmax')) # softmax for more than 2
```

```
WARNING:tensorflow:From C:\Users\GOPAGONI HARIKA\Anaconda3\lib\site-packages\tensorflow\python\ops\init_ops.py:1251: calling Va
rianceScaling.__init__ (from tensorflow.python.ops.init_ops) with dtype is deprecated and will be removed in a future version.
Instructions for updating:
Call initializer instance with the dtype argument instead of passing it to the constructor
```

In [10]:
```
classifier.summary()
```

```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d (Conv2D)              (None, 62, 62, 32)        896
_________________________________________________________________
max_pooling2d (MaxPooling2D) (None, 31, 31, 32)        0
_________________________________________________________________
conv2d_1 (Conv2D)            (None, 29, 29, 32)        9248
_________________________________________________________________
max_pooling2d_1 (MaxPooling2 (None, 14, 14, 32)        0
_________________________________________________________________
flatten (Flatten)            (None, 6272)              0
_________________________________________________________________
dense (Dense)                (None, 128)               802944
_________________________________________________________________
dense_1 (Dense)              (None, 4)                 516
=================================================================
Total params: 813,604
Trainable params: 813,604
Non-trainable params: 0
_________________________________________________________________
```

fitting the model

In [12]:
```
classifier.fit_generator(
       generator=x_train,steps_per_epoch = len(x_train),
       epochs=20, validation_data=x_test,validation_steps = len(x_test))
```

```
Epoch 1/20
149/149 [==============================] - 91s 612ms/step - loss: 1.1149 - acc: 0.5162 - val_loss: 0.9563 - val_acc: 0.6364
Epoch 2/20
149/149 [==============================] - 65s 434ms/step - loss: 0.7782 - acc: 0.6954 - val_loss: 0.7872 - val_acc: 0.7121
Epoch 3/20
149/149 [==============================] - 64s 430ms/step - loss: 0.6939 - acc: 0.7412 - val_loss: 0.8221 - val_acc: 0.7020
Epoch 4/20
149/149 [==============================] - 61s 406ms/step - loss: 0.6346 - acc: 0.7588 - val_loss: 0.8998 - val_acc: 0.6970
Epoch 5/20
149/149 [==============================] - 65s 438ms/step - loss: 0.5673 - acc: 0.7803 - val_loss: 0.7950 - val_acc: 0.7222
Epoch 6/20
149/149 [==============================] - 68s 454ms/step - loss: 0.5137 - acc: 0.8140 - val_loss: 0.7336 - val_acc: 0.7273
Epoch 7/20
149/149 [==============================] - 61s 408ms/step - loss: 0.5001 - acc: 0.8315 - val_loss: 0.6288 - val_acc: 0.7828
Epoch 8/20
149/149 [==============================] - 82s 553ms/step - loss: 0.3963 - acc: 0.8585 - val_loss: 0.7666 - val_acc: 0.7576
Epoch 9/20
149/149 [==============================] - 74s 497ms/step - loss: 0.3807 - acc: 0.8612 - val_loss: 1.0191 - val_acc: 0.6919
Epoch 10/20
149/149 [==============================] - 71s 477ms/step - loss: 0.4357 - acc: 0.8302 - val_loss: 0.7628 - val_acc: 0.7778
Epoch 11/20
149/149 [==============================] - 61s 411ms/step - loss: 0.3139 - acc: 0.8774 - val_loss: 0.6883 - val_acc: 0.7980
Epoch 12/20
149/149 [==============================] - 65s 438ms/step - loss: 0.3102 - acc: 0.8720 - val_loss: 0.7370 - val_acc: 0.7778
Epoch 13/20
149/149 [==============================] - 73s 487ms/step - loss: 0.2730 - acc: 0.8976 - val_loss: 0.9578 - val_acc: 0.7626
Epoch 14/20
149/149 [==============================] - 70s 468ms/step - loss: 0.2500 - acc: 0.9111 - val_loss: 1.2401 - val_acc: 0.7424
Epoch 15/20
149/149 [==============================] - 61s 411ms/step - loss: 0.2424 - acc: 0.9111 - val_loss: 0.9159 - val_acc: 0.7828
Epoch 16/20
149/149 [==============================] - 60s 402ms/step - loss: 0.2219 - acc: 0.9272 - val_loss: 0.9784 - val_acc: 0.7828
Epoch 17/20
149/149 [==============================] - 60s 404ms/step - loss: 0.2194 - acc: 0.9164 - val_loss: 0.8507 - val_acc: 0.7929
Epoch 18/20
149/149 [==============================] - 61s 406ms/step - loss: 0.1916 - acc: 0.9353 - val_loss: 1.1700 - val_acc: 0.7020
Epoch 19/20
149/149 [==============================] - 62s 414ms/step - loss: 0.1659 - acc: 0.9380 - val_loss: 0.8588 - val_acc: 0.8434
Epoch 20/20
149/149 [==============================] - 59s 398ms/step - loss: 0.1628 - acc: 0.9461 - val_loss: 0.9904 - val_acc: 0.7475
```

Out[12]:
```
<tensorflow.python.keras.callbacks.History at 0x115e300ef28>
```

saving our model ¶

In [13]:
```
classifier.save('disaster.h5')
```

In [14]:
```
model_json = classifier.to_json()
with open("model-bw.json", "w") as json_file:
    json_file.write(model_json)
```

Predicting our results

In [15]:
```
from tensorflow.keras.models import load_model
from keras.preprocessing import image
model = load_model("disaster.h5")
```

```
WARNING:tensorflow:From C:\Users\GOPAGONI HARIKA\Anaconda3\lib\site-packages\tensorflow\python\ops\init_ops.py:97: calling Glor
otUniform.__init__ (from tensorflow.python.ops.init_ops) with dtype is deprecated and will be removed in a future version.
Instructions for updating:
Call initializer instance with the dtype argument instead of passing it to the constructor
WARNING:tensorflow:From C:\Users\GOPAGONI HARIKA\Anaconda3\lib\site-packages\tensorflow\python\ops\init_ops.py:97: calling Zero
s.__init__ (from tensorflow.python.ops.init_ops) with dtype is deprecated and will be removed in a future version.
Instructions for updating:
Call initializer instance with the dtype argument instead of passing it to the constructor
```

In [16]:
```
img = image.load_img(r"D:\geethanjaliexternship\cyclone.jpg",grayscale=False,
                     target_size= (64,64))
x = image.img_to_array(img)
x = np.expand_dims(x,axis = 0)
pred = model.predict_classes(x)
pred
```

Out[16]:
```
array([0], dtype=int64)
```

In [17]:
```
index=['Cyclone','Earthquake','Flood','Wildfire']
result=str(index[pred[0]])
result
```

Out[17]:
```
'Cyclone'
```