

PROJECT DEVELOPMENT PHASE

SPRINT-II

Date	08 November 2022
Team ID	PNT2022TMID16547
Project Name	Natural Disaster Intensity Analysis and Classification using Artificial Intelligence

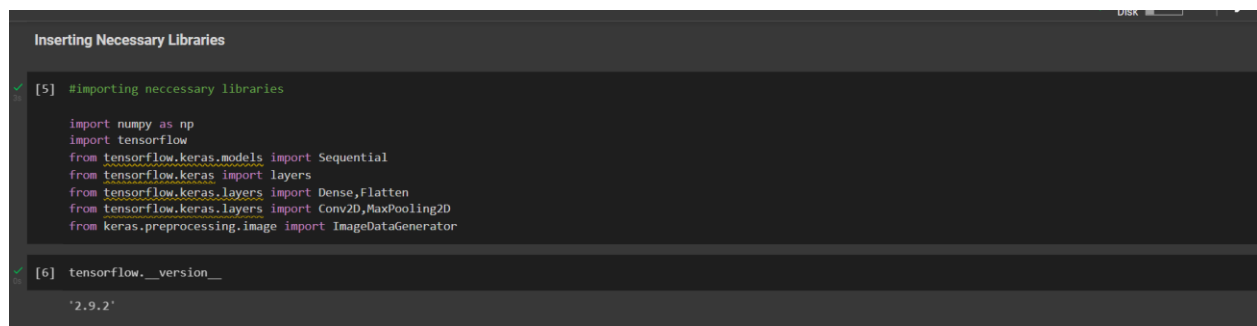
INSERTING NECESSARY LIBRARIES:

Numpy: It is an open-source numerical python library. **Scikit-**

learn: It is a machine learning library for python.

OpenCV: OpenCV is a library of programming functions mainly aimed at real-time computer vision.

Flask: Web framework used for building web application.



```
Inserting Necessary Libraries

[5] #importing necessary libraries

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

[6] tensorflow.__version__

'2.9.2'
```

LOADING DATA AND PERFORMING DATA AUGUMENTATION:

Loading the data into the Jupyter notebook by using RR dataset path.

```
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.

[21] 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})
```

CREATING THE MODEL:

Creating the Model a Classifier Sequential. Classifier is a machine learning algorithm that determines the class of the input element based on the set of the feature. In this model using convolution2D function. Convolution2D parameter is a number of filters that convolution layer will be learn from. Then we will be using MaxPooling2D function. Then, using a Flatten () function that flatten the multidimensional input denser into the denser.

```
[ ] # 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'))
```

Using classifier. Summary () function summary of our model

```
[12] # Summary of our model
model.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 (MaxPooling2D) (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
```

COMPILING THE MODEL:

The model is compiled using the following code.

```
[13] # Compiling the model
model.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['accuracy'])
```

FITTING THE MODEL:

Fitting the Model with 20 epochs.

```
[25] Epoch 4/20
149/149 [=====] - 40s 266ms/step - loss: 0.7471 - accuracy: 0.6873 - val_loss: 0.8075 - val_accuracy: 0.6919
Epoch 5/20
149/149 [=====] - 38s 259ms/step - loss: 0.6727 - accuracy: 0.7264 - val_loss: 0.6708 - val_accuracy: 0.7879
Epoch 6/20
149/149 [=====] - 40s 267ms/step - loss: 0.5779 - accuracy: 0.7682 - val_loss: 0.7330 - val_accuracy: 0.7071
Epoch 7/20
149/149 [=====] - 40s 267ms/step - loss: 0.5556 - accuracy: 0.7817 - val_loss: 0.7700 - val_accuracy: 0.7020
Epoch 8/20
149/149 [=====] - 40s 272ms/step - loss: 0.4917 - accuracy: 0.8154 - val_loss: 0.7167 - val_accuracy: 0.7475
Epoch 9/20
149/149 [=====] - 44s 296ms/step - loss: 0.4037 - accuracy: 0.8544 - val_loss: 1.0250 - val_accuracy: 0.6768
Epoch 10/20
149/149 [=====] - 40s 269ms/step - loss: 0.4380 - accuracy: 0.8383 - val_loss: 0.7964 - val_accuracy: 0.7727
Epoch 11/20
149/149 [=====] - 40s 272ms/step - loss: 0.3941 - accuracy: 0.8558 - val_loss: 0.6930 - val_accuracy: 0.7929
Epoch 12/20
149/149 [=====] - 40s 267ms/step - loss: 0.3716 - accuracy: 0.8679 - val_loss: 0.9208 - val_accuracy: 0.7222
Epoch 13/20
149/149 [=====] - 43s 288ms/step - loss: 0.3313 - accuracy: 0.8733 - val_loss: 0.8335 - val_accuracy: 0.7828
Epoch 14/20
149/149 [=====] - 39s 260ms/step - loss: 0.2898 - accuracy: 0.8881 - val_loss: 0.8075 - val_accuracy: 0.7980
Epoch 15/20
149/149 [=====] - 44s 294ms/step - loss: 0.3099 - accuracy: 0.8908 - val_loss: 0.7624 - val_accuracy: 0.7778
Epoch 16/20
149/149 [=====] - 40s 267ms/step - loss: 0.2878 - accuracy: 0.8935 - val_loss: 0.9509 - val_accuracy: 0.7172
Epoch 17/20
149/149 [=====] - 40s 266ms/step - loss: 0.2442 - accuracy: 0.9191 - val_loss: 0.8569 - val_accuracy: 0.7576
Epoch 18/20
149/149 [=====] - 45s 300ms/step - loss: 0.2249 - accuracy: 0.9191 - val_loss: 0.9880 - val_accuracy: 0.7879
Epoch 19/20
149/149 [=====] - 40s 267ms/step - loss: 0.1853 - accuracy: 0.9245 - val_loss: 1.0102 - val_accuracy: 0.7879
Epoch 20/20
149/149 [=====] - 40s 272ms/step - loss: 0.1931 - accuracy: 0.9272 - val_loss: 0.8715 - val_accuracy: 0.7828
<keras.callbacks.History at 0x7f062d79250>
```

SAVING THE MODEL:

Saving the Model as disaster.h5. disaster.h5 file is used to find the image classification files. Model.json represents that Jason stands for JavaScript object rotation, Jason is a lite weight data format used for data inserting between multiple different language.

```
[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)
```

PREDICTING RESULTS:

Loading model from the TensorFlow keras models and loading the image then converting image into array. Then predicting our model.

```
# Load the saved model
from tensorflow.keras.models import load_model
from tensorflow.keras.preprocessing import image
model = load_model('disaster.h5')

[28] x_train.class_indices
{'Cyclone': 0, 'Earthquake': 1, 'Flood': 2, 'Wildfire': 3}

[40] # taking image as input
img = image.load_img('content/dataset/test_set/Earthquake/1330.jpg', target_size=(64,64))
x = image.img_to_array(img)
x = np.expand_dims(x, axis=0)
index = ['Cyclone', 'Earthquake', 'Flood', 'Wildfire']
y = np.argmax(model.predict(x), axis=1)
print(index[int(y)])

1/1 [=====] - 0s 29ms/step
Earthquake

# input 2
img = image.load_img('content/dataset/test_set/Wildfire/1065.jpg', target_size=(64,64))
x = image.img_to_array(img)
x = np.expand_dims(x, axis=0)
index = ['Cyclone', 'Earthquake', 'Flood', 'Wildfire']
y = np.argmax(model.predict(x), axis=1)
print(index[int(y)])

1/1 [=====] - 0s 33ms/step
Wildfire
```