PROJECT DEVELOPMENT PHASE

NATURAL DISASTERS INTENSITY ANALYSIS AND CLASSIFICATION USING ARTIFICIAL INTELLIGENCE

PNT2022TMID06841

SPRINT-3(Model-Building & Model-Testing):

As Per Sprint Delivery plan, Sprint-3 includes:

User Story -4:

- Computer Vision Model for tropical Cyclone intensity estimation is important so that user can send the images for prediction.

User Story -5:

- Once the model is trained completely , test the model on data that it has not seen before to ensure its performance.

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.

```
print(x_train.class_indices)#checking the number of classes
```

```
print(x_test.class_indices)#checking the number of classes
```

```
from collections import Counter as c
c(x_train .labels)
```

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.

IMPORTING THE IMAGEDATAGENERATOR LIBRARY:

- By importing the ImageDataGenerator Library can expand the train_set data size using modified versions of dataset.
- ImageDataGenerator class were importing from keras.

In [4]:
```
import os
filenames = os.listdir('/home/wsuser/work/dataset/train_set')
print(filenames)
```

```
['Flood', 'Cyclone', 'Wildfire', 'Earthquake']
```

In [6]:
```
import tensorflow as tf
from keras.preprocessing.image import ImageDataGenerator
import numpy as np
```

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".

In [7]:
```
train_datagen = ImageDataGenerator(
        rescale=1./255,
        shear_range=0.2,
        zoom_range=0.2,
        horizontal_flip=True)
train_generator = train_datagen.flow_from_directory(
        '/home/wsuser/work/dataset/train_set',
        target_size=(64, 64),
        batch_size=32,
        class_mode='categorical')
```

```
Found 742 images belonging to 4 classes.
```

In [8]:
```
# Loading testing data
test_datagen = ImageDataGenerator(rescale=1./255)
test_generator = train_datagen.flow_from_directory(
        '/home/wsuser/work/dataset/test_set',
        target_size=(64, 64),
        batch_size=32,
        class_mode='categorical')
```

```
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.

CREATING THE MODEL:

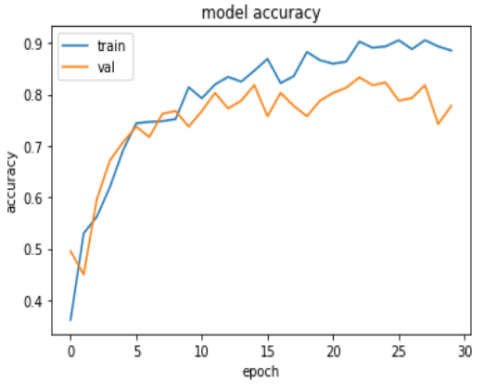
```
In [9]: # initialising sequential model and adding layers to it
cnn = tf.keras.models.Sequential()
cnn.add(tf.keras.layers.Conv2D(filters=48, kernel_size=3, activation='relu', input_shape=[64, 64, 3]))
cnn.add(tf.keras.layers.MaxPool2D(pool_size=2, strides=2))

cnn.add(tf.keras.layers.Conv2D(filters=48, kernel_size=3, activation='relu'))
cnn.add(tf.keras.layers.MaxPool2D(pool_size=2, strides=2))

cnn.add(tf.keras.layers.Conv2D(filters=32, kernel_size=3, activation='relu'))
cnn.add(tf.keras.layers.MaxPool2D(pool_size=2, strides=2))

cnn.add(tf.keras.layers.Flatten())

cnn.add(tf.keras.layers.Dense(128, activation='relu'))
cnn.add(tf.keras.layers.Dense(64, activation='relu'))
cnn.add(tf.keras.layers.Dense(4, activation='softmax'))

# finally compile and train the cnn
cnn.compile(optimizer="adam", loss="categorical_crossentropy", metrics=["accuracy"])
history = cnn.fit(x=train_generator, validation_data=test_generator, epochs=30)
```

```
Epoch 1/30
24/24 [==============================] - 40s 2s/step - loss: 1.3095 - accuracy: 0.3612 - val_loss: 1.2541 - val_accuracy: 0.4949
Epoch 2/30
24/24 [==============================] - 37s 2s/step - loss: 1.1077 - accuracy: 0.5296 - val_loss: 1.1826 - val_accuracy: 0.4495
Epoch 3/30
24/24 [==============================] - 37s 2s/step - loss: 1.0230 - accuracy: 0.5620 - val_loss: 0.9081 - val_accuracy: 0.5960
Epoch 4/30
24/24 [==============================] - 38s 2s/step - loss: 0.8863 - accuracy: 0.6199 - val_loss: 0.8335 - val_accuracy: 0.6717
Epoch 5/30
24/24 [==============================] - 37s 2s/step - loss: 0.7708 - accuracy: 0.6914 - val_loss: 0.7540 - val_accuracy: 0.7071
Epoch 6/30
24/24 [==============================] - 37s 2s/step - loss: 0.6646 - accuracy: 0.7439 - val_loss: 0.7229 - val_accuracy: 0.7374
Epoch 7/30
24/24 [==============================] - 37s 2s/step - loss: 0.6305 - accuracy: 0.7466 - val_loss: 0.7710 - val_accuracy: 0.7172
Epoch 8/30
24/24 [==============================] - 37s 2s/step - loss: 0.6764 - accuracy: 0.7480 - val_loss: 0.5899 - val_accuracy: 0.7626
Epoch 9/30
24/24 [==============================] - 37s 2s/step - loss: 0.6603 - accuracy: 0.7520 - val_loss: 0.6191 - val_accuracy: 0.7677
Epoch 10/30
24/24 [==============================] - 37s 2s/step - loss: 0.5335 - accuracy: 0.8140 - val_loss: 0.6714 - val_accuracy: 0.7374
Epoch 11/30
24/24 [==============================] - 37s 2s/step - loss: 0.5485 - accuracy: 0.7925 - val_loss: 0.5935 - val_accuracy: 0.7677
Epoch 12/30
24/24 [==============================] - 37s 2s/step - loss: 0.4789 - accuracy: 0.8194 - val_loss: 0.5811 - val_accuracy: 0.8030
```

```
24/24 [==============================] - 37s 2s/step - loss: 0.4804 - accuracy: 0.8221 - val_loss: 0.5309 - val_accuracy: 0.8030
Epoch 18/30
24/24 [==============================] - 37s 2s/step - loss: 0.4219 - accuracy: 0.8356 - val_loss: 0.6149 - val_accuracy: 0.7778
Epoch 19/30
24/24 [==============================] - 37s 2s/step - loss: 0.3280 - accuracy: 0.8827 - val_loss: 0.6493 - val_accuracy: 0.7576
Epoch 20/30
24/24 [==============================] - 37s 2s/step - loss: 0.3790 - accuracy: 0.8666 - val_loss: 0.6781 - val_accuracy: 0.7879
Epoch 21/30
24/24 [==============================] - 37s 2s/step - loss: 0.3687 - accuracy: 0.8639 - val_loss: 0.6118 - val_accuracy: 0.8131
Epoch 23/30
24/24 [==============================] - 36s 2s/step - loss: 0.2799 - accuracy: 0.9030 - val_loss: 0.5491 - val_accuracy: 0.8333
Epoch 24/30
24/24 [==============================] - 37s 2s/step - loss: 0.2985 - accuracy: 0.8908 - val_loss: 0.6183 - val_accuracy: 0.8182
Epoch 25/30
24/24 [==============================] - 37s 2s/step - loss: 0.3109 - accuracy: 0.8935 - val_loss: 0.5481 - val_accuracy: 0.8232
24/24 [==============================] - 37s 2s/step - loss: 0.2839 - accuracy: 0.9057 - val_loss: 0.5803 - val_accuracy: 0.7879
24/24 [==============================] - 37s 2s/step - loss: 0.2777 - accuracy: 0.8881 - val_loss: 0.6710 - val_accuracy: 0.7929
Epoch 28/30
24/24 [==============================] - 37s 2s/step - loss: 0.2624 - accuracy: 0.9057 - val_loss: 0.6343 - val_accuracy: 0.8182
Epoch 29/30
24/24 [==============================] - 38s 2s/step - loss: 0.2636 - accuracy: 0.8935 - val_loss: 0.8055 - val_accuracy: 0.7424
Epoch 30/30
24/24 [==============================] - 37s 2s/step - loss: 0.2921 - accuracy: 0.8854 - val_loss: 0.6966 - val_accuracy: 0.7778
```

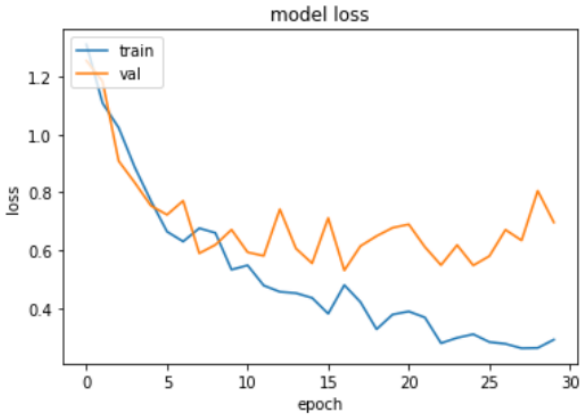
TRAINING AND TEST ACCURACY PLOT:

In [12]:
```
import matplotlib.pyplot as plt

#Traning and test accuracy plot
plt.plot(history.history['accuracy'])
plt.plot(history.history['val_accuracy'])
plt.title('model accuracy')
plt.ylabel('accuracy')
plt.xlabel('epoch')
plt.legend(['train', 'val'], loc='upper left')
plt.show()
```



MODEL LOSS PLOT:

In [14]:
```
#model loss plot
plt.plot(history.history['loss'])
plt.plot(history.history['val_loss'])
plt.title('model loss')
plt.ylabel('loss')
plt.xlabel('epoch')
plt.legend(['train', 'val'], loc='upper left')
plt.show()
```



EVALUATE THE MODEL:

In [ ]:
```
# evaluate the model
_, train_acc = cnn.evaluate(train_generator , verbose=0)
_, test_acc = cnn.evaluate(test_generator,  verbose=0)
print('Train: %.3f, Test: %.3f' % (train_acc, test_acc))
```

In [22]:
```
cnn.summary()
```

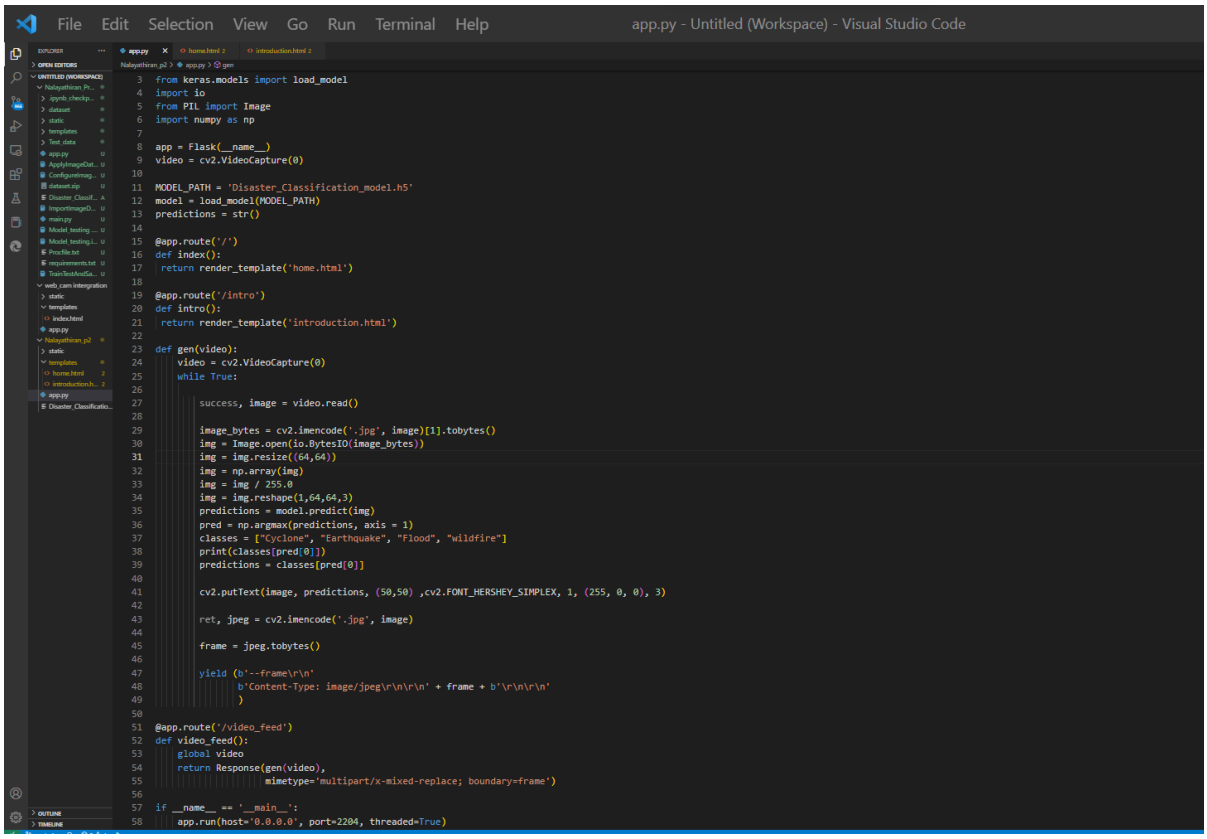
```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d (Conv2D)             (None, 62, 62, 48)        1344

 max_pooling2d (MaxPooling2D  (None, 31, 31, 48)       0
 )

 conv2d_1 (Conv2D)           (None, 29, 29, 48)        20784

 max_pooling2d_1 (MaxPooling  (None, 14, 14, 48)       0
 2D)

 conv2d_2 (Conv2D)           (None, 12, 12, 32)        13856

 max_pooling2d_2 (MaxPooling  (None, 6, 6, 32)         0
 2D)

 flatten (Flatten)           (None, 1152)              0

 dense (Dense)               (None, 128)               147584

 dense_1 (Dense)             (None, 64)                8256

 dense_2 (Dense)             (None, 4)                 260

=================================================================
Total params: 192,084
Trainable params: 192,084
Non-trainable params: 0
_________________________________________________________________
```

SAVING THE MODEL:

```
In [24]: probability_model = tf.keras.Sequential([cnn,
                                                  tf.keras.layers.Softmax()])
```

```
In [25]: cnn.save("Disaster_Classification_model.h5")
```

CREATING app.py:




```python
from keras.models import load_model
import io
from PIL import Image
import numpy as np

app = Flask(__name__)
video = cv2.VideoCapture(0)

MODEL_PATH = 'Disaster_Classification_model.h5'
model = load_model(MODEL_PATH)
predictions = str()

@app.route('/')
def index():
 return render_template('home.html')

@app.route('/intro')
def intro():
 return render_template('introduction.html')

def gen(video):
    video = cv2.VideoCapture(0)
    while True:

        success, image = video.read()

        image_bytes = cv2.imencode('.jpg', image)[1].tobytes()
        img = Image.open(io.BytesIO(image_bytes))
        img = img.resize((64,64))
        img = np.array(img)
        img = img / 255.0
        img = img.reshape(1,64,64,3)
        predictions = model.predict(img)
        pred = np.argmax(predictions, axis = 1)
        classes = ["Cyclone", "Earthquake", "Flood", "wildfire"]
        print(classes[pred[0]])
        predictions = classes[pred[0]]

        cv2.putText(image, predictions, (50,50) ,cv2.FONT_HERSHEY_SIMPLEX, 1, (255, 0, 0), 3)

        ret, jpeg = cv2.imencode('.jpg', image)

        frame = jpeg.tobytes()

        yield (b'--frame\r\n'
               b'Content-Type: image/jpeg\r\n\r\n' + frame + b'\r\n\r\n'
               )

@app.route('/video_feed')
def video_feed():
    global video
    return Response(gen(video),
                    mimetype='multipart/x-mixed-replace; boundary=frame')

if __name__ == '__main__':
    app.run(host='0.0.0.0', port=2204, threaded=True)
```