TEAM ID: PNT2022TMID25540

PROJECT NAME: Natural Disaster Intensity Analysis And Classification

DATE : 16 NOV 2022

PROJECT DEVELOPMENT PHASE

SPRINT – 2

IMPORTING NECESSARY LIBRARIES

It is a common problem that people want to import code from Jupyter Notebooks. This is made difficult by the fact that Notebooks are not plain Python files, and thus cannot be imported by the regular Python machinery.

Fortunately, Python provides some fairly sophisticated hooks into the import machinery, so we can actually make Jupyter notebooks importable without much difficulty, and only using public APIs.

Importing Neccessary Libraries

```
[ ] from google.colab import drive
    drive.mount('/content/drive')

    Mounted at /content/drive
```

```
[ ] import numpy as np#used for numerical analysis
    import tensorflow #open source used for both ML and DL for computation
    from tensorflow.keras.models import Sequential #it is a plain stack of layers
    from tensorflow.keras import layers #A layer consists of a tensor-in tensor-out computation function
    #Dense layer is the regular deeply connected neural network layer
    from tensorflow.keras.layers import Dense,Flatten
    #Faltten-used fot flattening the input or change the dimension
    from tensorflow.keras.layers import Conv2D,MaxPooling2D #Convolutional layer
    #MaxPooling2D-for downsampling the image
    from keras.preprocessing.image import ImageDataGenerator
```

```
[ ] tensorflow.__version__

    '2.9.2'
```

```
[ ] tensorflow.keras.__version__

    '2.9.0'
```

```
[ ]
```

IMAGE DATA AUGMENTATION

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

There are five main types of data augmentation techniques for image data; specifically:

Image shifts via the width_shift_range and height_shift_range arguments.
The image flips via the horizontal_flip and vertical_flip arguments.
Image rotations via the rotation_range argument
Image brightness via the brightness_range argument.
Image zoom via the zoom_range argument.

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

Image Data Agumentation

```
[ ] #setting parameter for Image Data agumentation to the training data
    train_datagen = ImageDataGenerator(rescale=1./255,shear_range=0.2,zoom_range=0.2,horizontal_flip=True)
    #Image Data agumentation to the testing data
    test_datagen=ImageDataGenerator(rescale=1./255)

[ ]
```

Loading our data and performing data agumentation

Let us apply ImageDataGenerator functionality to Trainset and Testset by using the following code
For Training set using flow_from_directory function.

This function will return batches of images from the subdirectories Cyclone, Earthquake, Flood, Wildfire together with labels 0 to 3{Cyclone: 0, Earthquake: 1, Flood: 2, Wildfire: 3, }

Arguments:

- directory: Directory where the data is located. If labels are "inferred", it should contain subdirectories, each containing images for a class. Otherwise, the directory structure is ignored.
- batch_size: Size of the batches of data. Default: 32.
- target_size: Size to resize images after they are read from disk.
- class_mode:
  - ‘int': means that the labels are encoded as integers (e.g. for sparse_categorical_crossentropy loss).
  - 'categorical' means that the labels are encoded as a categorical vector (e.g. for categorical_crossentropy loss).
  - 'binary' means that the labels (there can be only 2) are encoded as float32 scalars with values 0 or 1 (e.g. for binary_crossentropy).
  - None (no labels).

Loading our data and performing data agumentation

```
[ ] #performing data agumentation to train data
    x_train = train_datagen.flow_from_directory(r'/content/drive/MyDrive/PRIEE/AI-Based-Natural-Disaster-Intensity-Analysis-main/dataset/dataset/train_set',target_size=(64, 64),batch_size=5,
                                                color_mode='rgb',class_mode='categorical')
    #performing data agumentation to test data
    x_test = test_datagen.flow_from_directory(r'/content/drive/MyDrive/PRIEE/AI-Based-Natural-Disaster-Intensity-Analysis-main/dataset/dataset/test_set',target_size=(64, 64),batch_size=5,
                                                color_mode='rgb',class_mode='categorical')

    Found 737 images belonging to 4 classes.
    Found 178 images belonging to 4 classes.
```

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

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

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

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

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

    Counter({0: 215, 1: 156, 2: 198, 3: 168})
```

CREATING THE MODEL

We are ready with the augmented and pre-processed image data, Lets begin our model building.

Creating the model

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

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

    # 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
```

```
[ ] classifier.summary()#summary of our model

    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

     conv2d_2 (Conv2D)           (None, 27, 27, 32)        9248

     max_pooling2d_1 (MaxPooling  (None, 13, 13, 32)       0
     2D)

     conv2d_3 (Conv2D)           (None, 11, 11, 32)        9248

     flatten (Flatten)           (None, 3872)              0

     dense (Dense)               (None, 128)               495744

     dense_1 (Dense)             (None, 4)                 516

    =================================================================
    Total params: 524,900
    Trainable params: 524,900
    Non-trainable params: 0
    _________________________________________________________________
```

COMPILING THE MODEL

Once your model looks good, configure its learning process with .compile():

Compiling the model

```
[ ] # Compiling the CNN
    # categorical_crossentropy for more than 2
    classifier.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['accuracy'])
```

FITTING THE MODEL

If you need to, you can further configure your optimizer. The Keras philosophy is to keep simple things simple, while allowing the user to be fully in control when they need to (the ultimate control being the easy extensibility of the source code via subclassing).

Fitting the model

```
[ ] classifier.fit_generator(
            generator=x_train,steps_per_epoch = len(x_train),
            epochs=40, validation_data=x_test,validation_steps = len(x_test))# No of images in test set
```

```
/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:3: UserWarning: `Model.fit_generator` is deprecated and will be removed in a future version. Please use `Model.fit`, which supports generators.
  This is separate from the ipykernel package so we can avoid doing imports until
Epoch 1/40
 31/148 [=====>........................] - ETA: 7:26 - loss: 1.3785 - accuracy: 0.2829
---------------------------------------------------------------------------
KeyboardInterrupt                         Traceback (most recent call last)
<ipython-input-14-814a542bbd4a> in <module>
      1 classifier.fit_generator(
      2         generator=x_train,steps_per_epoch = len(x_train),
----> 3         epochs=40, validation_data=x_test,validation_steps = len(x_test))# No of images in test set

9 frames
/usr/local/lib/python3.7/dist-packages/tensorflow/python/eager/execute.py in quick_execute(op_name, num_outputs, inputs, attrs, ctx, name)
     53     ctx.ensure_initialized()
     54     tensors = pywrap_tfe.TFE_Py_Execute(ctx._handle, device_name, op_name,
---> 55                                         inputs, attrs, num_outputs)
     56   except core._NotOkStatusException as e:
     57     if name is not None:

KeyboardInterrupt:
```

SEARCH STACK OVERFLOW

SAVING THE MODEL

The model is saved with .h5 extension as follows
An H5 file is a data file saved in the Hierarchical Data Format (HDF). It contains multidimensional arrays of scientific data.

Saving our model

```
[ ] # Save the model
    classifier.save('disaster_f.h5')
```

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

```
[ ]
```

PREDICTING THE MODEL

By using the model we are predicting the output for the given input image

The predicted class index name will be printed here.

Predicting our results

```
[ ] from tensorflow.keras.models import load_model
    from keras.preprocessing import image
    #model = load_model("disaster_f.h5") #loading the model for testing
```

```
[ ] img = image.load_img(r"E:\SB1\owntest\eq.jpeg",grayscale=False,
                         target_size= (64,64))#loading of the image
    x = image.img_to_array(img)#image to array
    x = np.expand_dims(x,axis = 0)#changing the shape
    pred = classifier.predict_classes(x)#predicting the classes
    pred
```

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

```
[ ] !pip install jupyterthemes as jt
```

```
[ ] !jt -t monokai
```

```
[ ]
```