Team Id	PNT2022TMID39414
Project Name	Natural Disaster Intensity Analysis And Classification Using Artificial Intelligence

Natural Disaster Intensity Analysis And Classification

Using Artificial Intelligence

Model Building

This activity includes the following steps

- 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
- Saving the model

Import the model building Libraries

The first step in building a model is to import the libraries

Importing Neccessary Libraries

```
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
```

Initializing the Model

Keras has 2 ways to define a neural network:

- Sequential
- Function API

The Sequential class is used to define a linear initialization of network layers which then, collectively, constitute a model. We use sequential constructor to create a model, which will then have layers added to it using the add() method.

```
model=Sequential()
```

Adding CNN Layers

- As the input image contains three channels, we are specifying the input shape as (64,64,3).
- We are adding a convolution layer with activation function as “relu” and with a small filter size (3,3) and the number of filters (32) followed by a max-pooling layer.
- Max pool layer is used to downsample the input.
- Flatten layer flattens the input. Does not affect the batch size.
-

```
model.add(Conv2D(32,(3,3),input_shape=(64,64,3),activation='relu'))
model.add(MaxPooling2D(pool_size=(2,2)))
model.add(Conv2D(32,(3,3),activation='relu'))
model.add(MaxPooling2D(pool_size=(2,2)))
model.add(Flatten())
```

Adding Dense Layers

A dense layer is a deeply connected neural network layer. It is the most common and frequently used layer.

```
model.add(Dense(units=128,activation='relu'))
model.add(Dense(units=4,activation='softmax'))
```

The number of neurons in the Dense layer is same as the number of classes in the training set. The neurons in the last Dense layer, use softmax activation to convert their output into respective probabilities.

Understanding the model is very important phase to properly use it for training purposes. Keras provides a simple method, summary to get the full information about the model and its layers.

```
[ ]  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 (MaxPooling  (None, 14, 14, 32)       0
 2D)

 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
_________________________________________________________________
```

Configure the learning Process

- The compilation is the final step in creating a model. Once the compilation is done, we can move on to the training phase. The loss function is used to find errors or deviations in the learning process. Keras requires loss function during the model compilation process.
- Optimization is an important process that optimizes the input weights by comparing the prediction and the loss function. Here we are using adam optimizer.
- Metrics are used to evaluate the performance of your model. It is similar to the loss function, but not used in the training process.
-

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

Train the model

Now, let us train our model with our image dataset. The model is trained for 20 epochs and after every epoch, the current model state is saved if the model has the least loss decreases in almost every epoch till 20 epochs and probably there is further scope to improve the model.

Fit_generator functions used to train a deep learning neural network

Arguments:

- Steps_per_epoch: it specifies the total number of steps taken from the generator as soon as one epoch is finished and the next epoch has started. We can total number of samples in your dataset divided by the batch size.
- Epochs: an integer and number of epochs we want to train our model for.
- Validation_data can be either:
  -an inputs and targets list
  -a generator
  -an inputs, targets, and sample_weights list which can be used to evaluate the loss and metrics for any model after any epoch has ended.
- Validation_steps: only if the validation_data is a generator can be used. It specifies the total number of steps taken from the generator before it is stopped at every epoch and its value is calculated as the total number of validation data points in your dataset divided by the validation batch size.

```
model.fit_generator(generator=x_train,steps_per_epoch=len(x_train),validation_data=x_test,validation_steps=len(x_test),epochs=20)
```

```
/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:1: UserWarning: `Model.fit_generator` is deprecated and will be removed in a future version. Please use `Model.fit`, which
  """Entry point for launching an IPython kernel.
Epoch 1/20
149/149 [==============================] - 429s 3s/step - loss: 1.1075 - accuracy: 0.5431 - val_loss: 0.8727 - val_accuracy: 0.6414
Epoch 2/20
149/149 [==============================] - 35s 231ms/step - loss: 0.7546 - accuracy: 0.6873 - val_loss: 0.6263 - val_accuracy: 0.7525
Epoch 3/20
149/149 [==============================] - 34s 225ms/step - loss: 0.6689 - accuracy: 0.7318 - val_loss: 0.7319 - val_accuracy: 0.7273
Epoch 4/20
149/149 [==============================] - 33s 217ms/step - loss: 0.5827 - accuracy: 0.7574 - val_loss: 0.7686 - val_accuracy: 0.7424
Epoch 5/20
149/149 [==============================] - 33s 219ms/step - loss: 0.5061 - accuracy: 0.8100 - val_loss: 0.5469 - val_accuracy: 0.8030
Epoch 6/20
149/149 [==============================] - 34s 226ms/step - loss: 0.4730 - accuracy: 0.8315 - val_loss: 0.5556 - val_accuracy: 0.8182
Epoch 7/20
149/149 [==============================] - 32s 218ms/step - loss: 0.4642 - accuracy: 0.8221 - val_loss: 0.5224 - val_accuracy: 0.8283
Epoch 8/20
149/149 [==============================] - 32s 217ms/step - loss: 0.4213 - accuracy: 0.8288 - val_loss: 0.6842 - val_accuracy: 0.8030
Epoch 9/20
149/149 [==============================] - 33s 223ms/step - loss: 0.3917 - accuracy: 0.8544 - val_loss: 0.6540 - val_accuracy: 0.7727
Epoch 10/20
149/149 [==============================] - 33s 219ms/step - loss: 0.3245 - accuracy: 0.8827 - val_loss: 0.8957 - val_accuracy: 0.7475
Epoch 11/20
149/149 [==============================] - 32s 216ms/step - loss: 0.3467 - accuracy: 0.8747 - val_loss: 0.5863 - val_accuracy: 0.8283
Epoch 12/20
149/149 [==============================] - 32s 217ms/step - loss: 0.3061 - accuracy: 0.8787 - val_loss: 0.7613 - val_accuracy: 0.7980
Epoch 13/20
149/149 [==============================] - 33s 220ms/step - loss: 0.2523 - accuracy: 0.9137 - val_loss: 0.7057 - val_accuracy: 0.7980
Epoch 14/20
149/149 [==============================] - 32s 216ms/step - loss: 0.2450 - accuracy: 0.9272 - val_loss: 0.7239 - val_accuracy: 0.8030
Epoch 15/20
149/149 [==============================] - 32s 218ms/step - loss: 0.2441 - accuracy: 0.9164 - val_loss: 0.6528 - val_accuracy: 0.8182
Epoch 16/20
149/149 [==============================] - 33s 222ms/step - loss: 0.2148 - accuracy: 0.9111 - val_loss: 0.8139 - val_accuracy: 0.7929
```

```
Epoch 17/20
149/149 [==============================] - 33s 219ms/step - loss: 0.2063 - accuracy: 0.9299 - val_loss: 0.8902 - val_accuracy: 0.7879
Epoch 18/20
149/149 [==============================] - 34s 228ms/step - loss: 0.1718 - accuracy: 0.9407 - val_loss: 0.8917 - val_accuracy: 0.7980
Epoch 19/20
149/149 [==============================] - 34s 232ms/step - loss: 0.1728 - accuracy: 0.9340 - val_loss: 1.5961 - val_accuracy: 0.6717
Epoch 20/20
149/149 [==============================] - 34s 225ms/step - loss: 0.1809 - accuracy: 0.9299 - val_loss: 0.7846 - val_accuracy: 0.8182
<keras.callbacks.History at 0x7f6a5c2fd310>
```

Save the model

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

```
model.save('disaster.h5')
model_json=model.to_json()
with open("model-bw.json","w") as json_file:
  json_file.write(model_json)
```

Test the model

Evaluation is a process during the development of the model to check whether the model is the best fit for the given problem and corresponding data.

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

```
img = image.load_img(r"/content/drive/MyDrive/dataset/dataset/test_set/Earthquake/1328.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 19ms/step
Earthquake
```