Project Development Phase

Sprint 2

Date	**17 November 2022**
Team ID	PNT2022TMID42615
Project Name	Natural Disaster Intensity Analysis and Classification using Artificial Intelligence

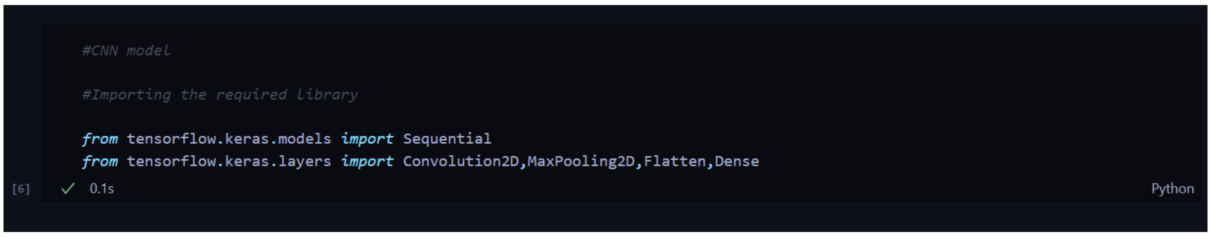
CNN Model Building:

AI model (CNN – Convolution Neural Network) building involves training the model, testing the model and saving the model.

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.

1.Importing the libraries:

We can start building a model by importing a required libraries like *Sequential* from *tensorflow.keras.models* module and required CNN layers such as *Convolution layer, Maxpooling layer, Flatten layer, Dense layer* from *tensorflow.keras.layers* module.

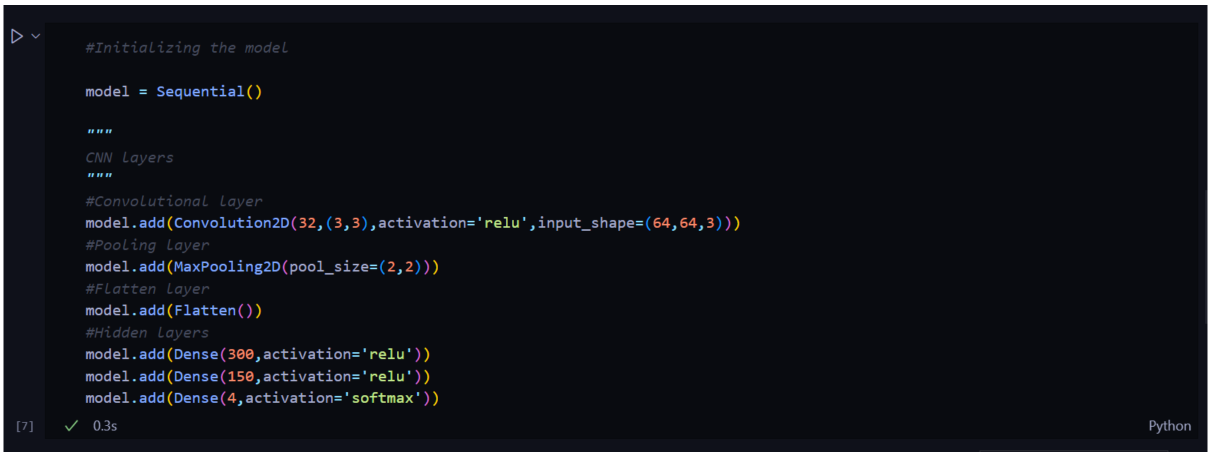


```
#CNN model

#Importing the required library

from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Convolution2D,MaxPooling2D,Flatten,Dense
```

*Importing the Sequential class from tensorflow.keras.models module and Convoltion2D, Maxpooling2D, Flatten, Dense classes from tensorflow.keras.layers module

1. Initializing the model:

After importing the required libraries, we have to initialize the model with Convolution layer, Maxpooling layer, Flatten layer, and Dense (Hidden) layers.



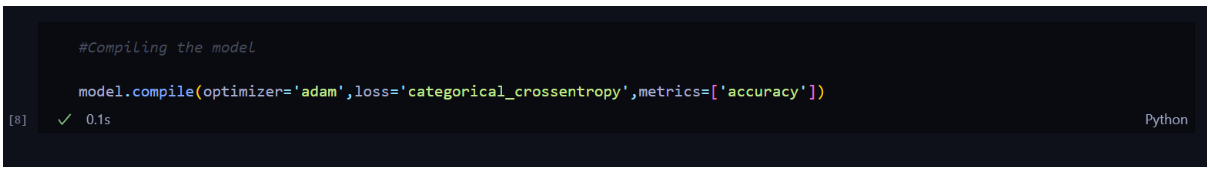
```python
#Initializing the model

model = Sequential()

"""
CNN layers
"""
#Convolutional layer
model.add(Convolution2D(32,(3,3),activation='relu',input_shape=(64,64,3)))
#Pooling layer
model.add(MaxPooling2D(pool_size=(2,2)))
#Flatten layer
model.add(Flatten())
#Hidden layers
model.add(Dense(300,activation='relu'))
model.add(Dense(150,activation='relu'))
model.add(Dense(4,activation='softmax'))
```

[7] ✓ 0.3s Python

*Initializing the CNN model with various layers

3.Compiling the model:

Compilation process of an AI model is the process of transforming and optimizing AI execution from its development form to its deployment form.



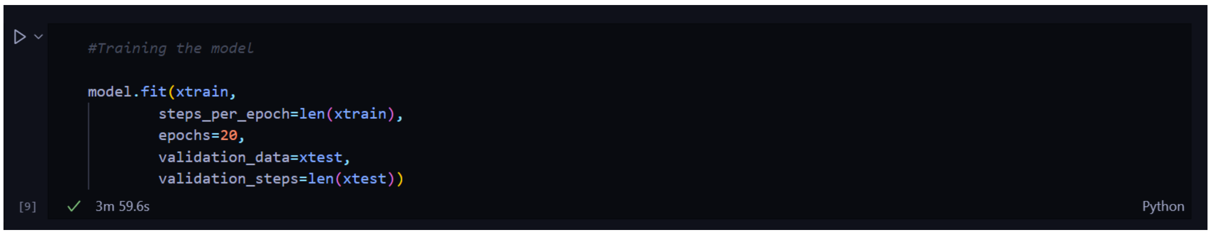
```python
#Compiling the model

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

[8] ✓ 0.1s Python

*Compiling the model

4. Training the model:

To develop an AI model, it should be trained with training data. The selection of training dataset is very important process in developing an AI model. In order to get a accurate prediction, it should be trained with good quality train dataset.



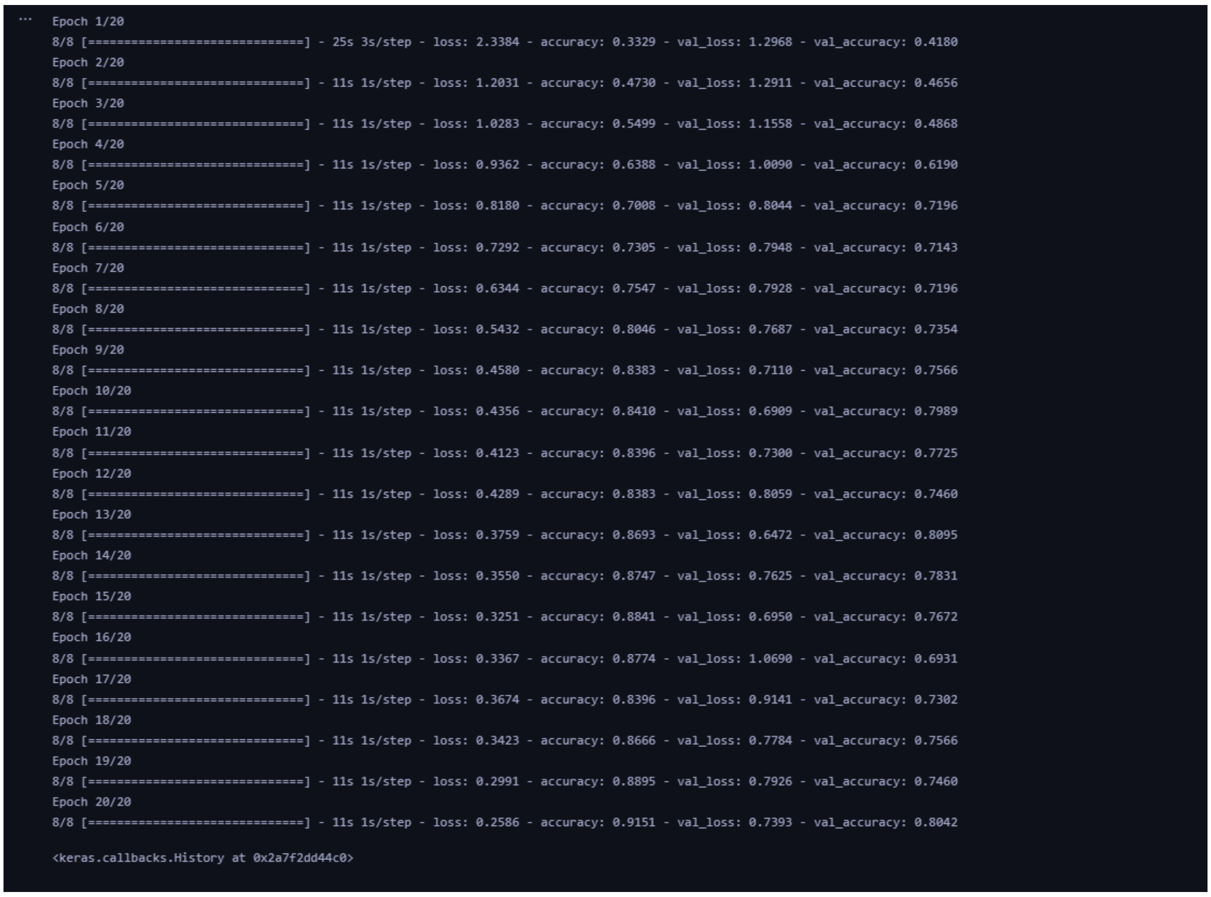
```python
#Training the model

model.fit(xtrain,
        steps_per_epoch=len(xtrain),
        epochs=20,
        validation_data=xtest,
        validation_steps=len(xtest))
```

[9] ✓ 3m 59.6s Python

*Training the model

5.Fitting the model

Training and fitting the model with 20 epochs.



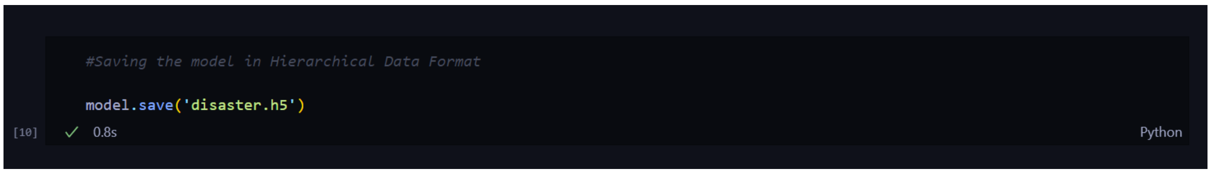
```
... Epoch 1/20
    8/8 [==============================] - 25s 3s/step - loss: 2.3384 - accuracy: 0.3329 - val_loss: 1.2968 - val_accuracy: 0.4180
    Epoch 2/20
    8/8 [==============================] - 11s 1s/step - loss: 1.2031 - accuracy: 0.4730 - val_loss: 1.2911 - val_accuracy: 0.4656
    Epoch 3/20
    8/8 [==============================] - 11s 1s/step - loss: 1.0283 - accuracy: 0.5499 - val_loss: 1.1558 - val_accuracy: 0.4868
    Epoch 4/20
    8/8 [==============================] - 11s 1s/step - loss: 0.9362 - accuracy: 0.6388 - val_loss: 1.0090 - val_accuracy: 0.6190
    Epoch 5/20
    8/8 [==============================] - 11s 1s/step - loss: 0.8180 - accuracy: 0.7008 - val_loss: 0.8044 - val_accuracy: 0.7196
    Epoch 6/20
    8/8 [==============================] - 11s 1s/step - loss: 0.7292 - accuracy: 0.7305 - val_loss: 0.7948 - val_accuracy: 0.7143
    Epoch 7/20
    8/8 [==============================] - 11s 1s/step - loss: 0.6344 - accuracy: 0.7547 - val_loss: 0.7928 - val_accuracy: 0.7196
    Epoch 8/20
    8/8 [==============================] - 11s 1s/step - loss: 0.5432 - accuracy: 0.8046 - val_loss: 0.7687 - val_accuracy: 0.7354
    Epoch 9/20
    8/8 [==============================] - 11s 1s/step - loss: 0.4580 - accuracy: 0.8383 - val_loss: 0.7110 - val_accuracy: 0.7566
    Epoch 10/20
    8/8 [==============================] - 11s 1s/step - loss: 0.4356 - accuracy: 0.8410 - val_loss: 0.6909 - val_accuracy: 0.7989
    Epoch 11/20
    8/8 [==============================] - 11s 1s/step - loss: 0.4123 - accuracy: 0.8396 - val_loss: 0.7300 - val_accuracy: 0.7725
    Epoch 12/20
    8/8 [==============================] - 11s 1s/step - loss: 0.4289 - accuracy: 0.8383 - val_loss: 0.8059 - val_accuracy: 0.7460
    Epoch 13/20
    8/8 [==============================] - 11s 1s/step - loss: 0.3759 - accuracy: 0.8693 - val_loss: 0.6472 - val_accuracy: 0.8095
    Epoch 14/20
    8/8 [==============================] - 11s 1s/step - loss: 0.3550 - accuracy: 0.8747 - val_loss: 0.7625 - val_accuracy: 0.7831
    Epoch 15/20
    8/8 [==============================] - 11s 1s/step - loss: 0.3251 - accuracy: 0.8841 - val_loss: 0.6950 - val_accuracy: 0.7672
    Epoch 16/20
    8/8 [==============================] - 11s 1s/step - loss: 0.3367 - accuracy: 0.8774 - val_loss: 1.0690 - val_accuracy: 0.6931
    Epoch 17/20
    8/8 [==============================] - 11s 1s/step - loss: 0.3674 - accuracy: 0.8396 - val_loss: 0.9141 - val_accuracy: 0.7302
    Epoch 18/20
    8/8 [==============================] - 11s 1s/step - loss: 0.3423 - accuracy: 0.8666 - val_loss: 0.7784 - val_accuracy: 0.7566
    Epoch 19/20
    8/8 [==============================] - 11s 1s/step - loss: 0.2991 - accuracy: 0.8895 - val_loss: 0.7926 - val_accuracy: 0.7460
    Epoch 20/20
    8/8 [==============================] - 11s 1s/step - loss: 0.2586 - accuracy: 0.9151 - val_loss: 0.7393 - val_accuracy: 0.8042

    <keras.callbacks.History at 0x2a7f2dd44c0>
```

*Fitting the model

As we see, the accuracy increases with increase in epoch steps

6. Saving the model

After training the model, we can save it for future use. Saving our AI model in a hierarchy file (.h5 file).



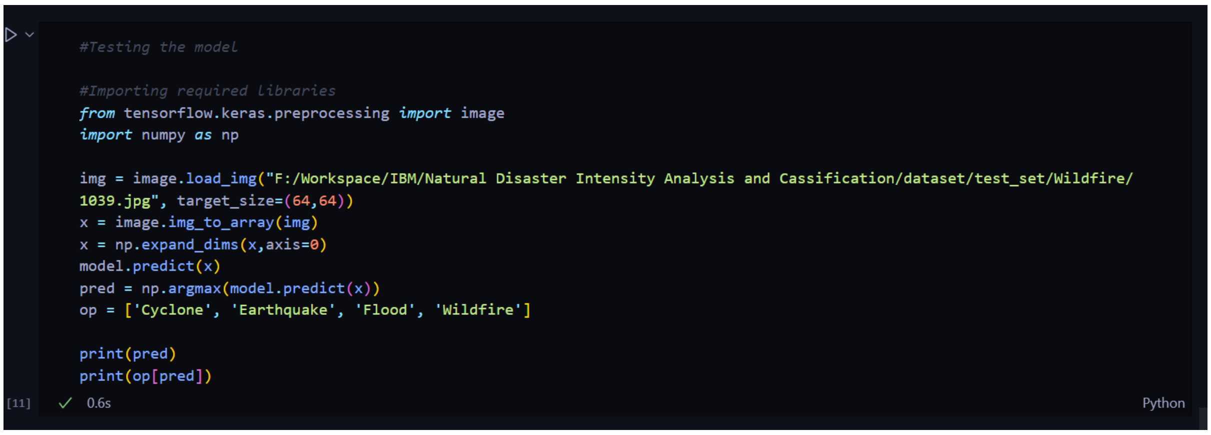
```
#Saving the model in Hierarchical Data Format

model.save('disaster.h5')
```

[10] ✓ 0.8s Python

*Saving the model in 'disasterh5'

7. Testing the model

Loading the model from the tensorflow.keras.models and predicting the result with test dataset.

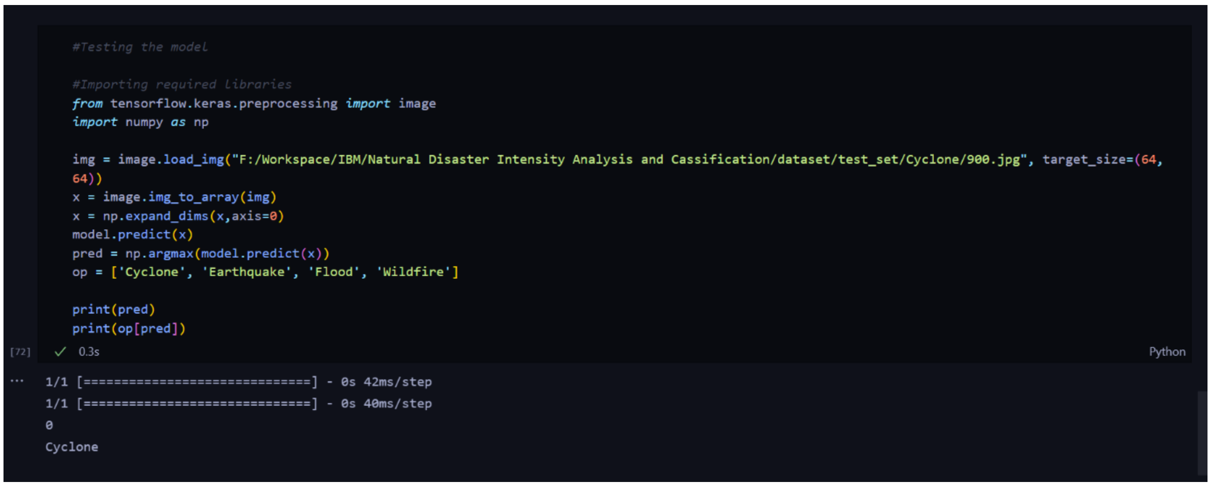


```
#Testing the model

#Importing required libraries
from tensorflow.keras.preprocessing import image
import numpy as np

img = image.load_img("F:/Workspace/IBM/Natural Disaster Intensity Analysis and Cassification/dataset/test_set/Wildfire/
1039.jpg", target_size=(64,64))
x = image.img_to_array(img)
x = np.expand_dims(x,axis=0)
model.predict(x)
pred = np.argmax(model.predict(x))
op = ['Cyclone', 'Earthquake', 'Flood', 'Wildfire']

print(pred)
print(op[pred])
```

[11] ✓ 0.6s Python

*Testing the model with test dataset

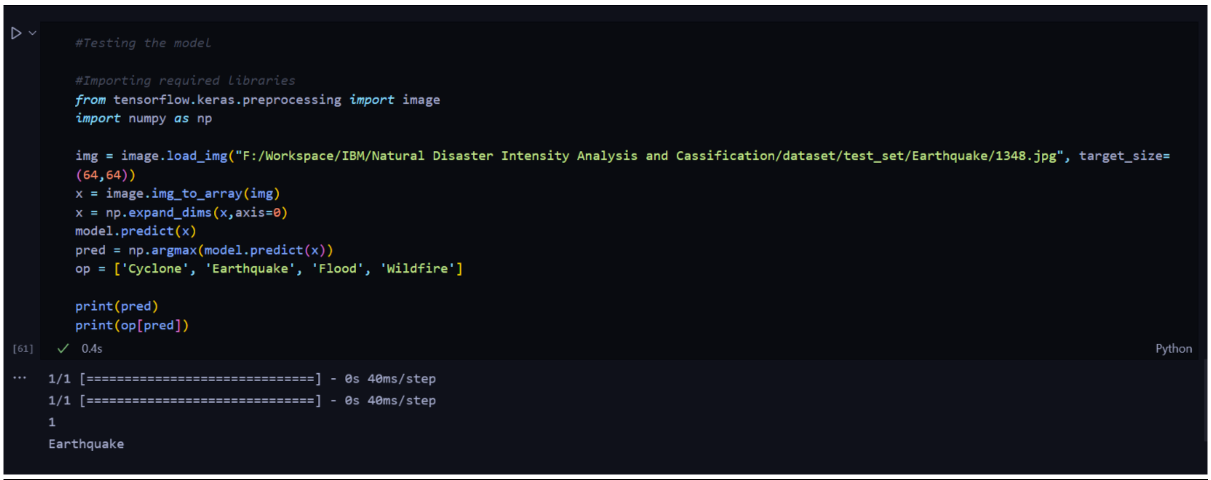
Prediction results:



```
#Testing the model

#Importing required libraries
from tensorflow.keras.preprocessing import image
import numpy as np

img = image.load_img("F:/Workspace/IBM/Natural Disaster Intensity Analysis and Cassification/dataset/test_set/Cyclone/900.jpg", target_size=(64,
64))
x = image.img_to_array(img)
x = np.expand_dims(x,axis=0)
model.predict(x)
pred = np.argmax(model.predict(x))
op = ['Cyclone', 'Earthquake', 'Flood', 'Wildfire']

print(pred)
print(op[pred])
```

[72] ✓ 0.3s Python

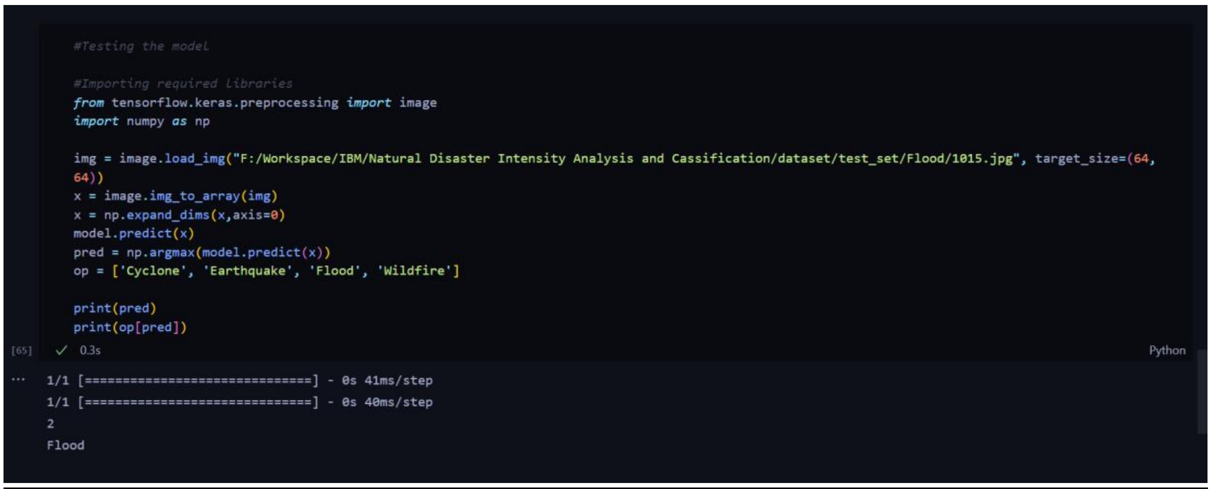
```
1/1 [==============================] - 0s 42ms/step
1/1 [==============================] - 0s 40ms/step
0
Cyclone
```



```
#Testing the model

#Importing required libraries
from tensorflow.keras.preprocessing import image
import numpy as np

img = image.load_img("F:/Workspace/IBM/Natural Disaster Intensity Analysis and Cassification/dataset/test_set/Earthquake/1348.jpg", target_size=
(64,64))
x = image.img_to_array(img)
x = np.expand_dims(x,axis=0)
model.predict(x)
pred = np.argmax(model.predict(x))
op = ['Cyclone', 'Earthquake', 'Flood', 'Wildfire']

print(pred)
print(op[pred])
```

[61] ✓ 0.4s Python

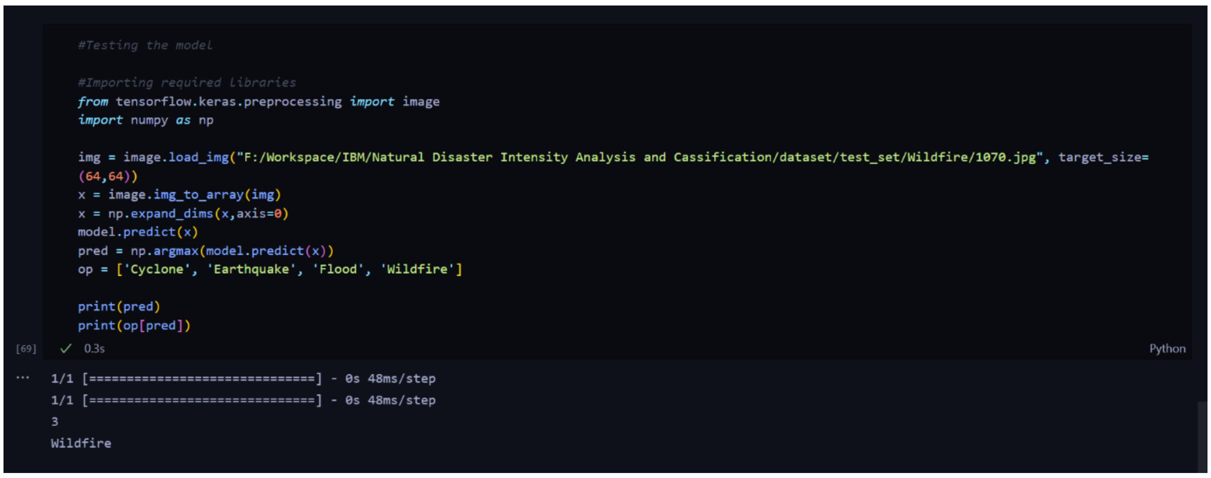
```
1/1 [==============================] - 0s 40ms/step
1/1 [==============================] - 0s 40ms/step
1
Earthquake
```



```python
#Testing the model

#Importing required libraries
from tensorflow.keras.preprocessing import image
import numpy as np

img = image.load_img("F:/Workspace/IBM/Natural Disaster Intensity Analysis and Cassification/dataset/test_set/Flood/1015.jpg", target_size=(64,
64))
x = image.img_to_array(img)
x = np.expand_dims(x,axis=0)
model.predict(x)
pred = np.argmax(model.predict(x))
op = ['Cyclone', 'Earthquake', 'Flood', 'Wildfire']

print(pred)
print(op[pred])
```

[65] ✓ 0.3s Python

```
1/1 [==============================] - 0s 41ms/step
1/1 [==============================] - 0s 40ms/step
2
Flood
```



```python
#Testing the model

#Importing required libraries
from tensorflow.keras.preprocessing import image
import numpy as np

img = image.load_img("F:/Workspace/IBM/Natural Disaster Intensity Analysis and Cassification/dataset/test_set/Wildfire/1070.jpg", target_size=
(64,64))
x = image.img_to_array(img)
x = np.expand_dims(x,axis=0)
model.predict(x)
pred = np.argmax(model.predict(x))
op = ['Cyclone', 'Earthquake', 'Flood', 'Wildfire']

print(pred)
print(op[pred])
```

[69] ✓ 0.3s Python

```
1/1 [==============================] - 0s 48ms/step
1/1 [==============================] - 0s 48ms/step
3
Wildfire
```

Note: Prediction is done with selecting random images in different classes. Even though the model is well trained with dataset, it couldn't achieve the maximum accuracy of level 1. It can be resolved by trained the model with more and more data.