

## Sprint – III

### Model building

<b>Date</b>	<b>14 November2022</b>
<b>Team ID</b>	<b>PNT2022TMID40634</b>
<b>Project Name</b>	<b>Natural Disasters Intensity Analysisand Classification using Artificial Intelligence</b>

### Extract zip file

ZIP is an archive file format that supports lossless data compression. By lossless compression, we mean that the compression algorithm allows the original data to be perfectly reconstructed from the compressed data.

```
#extract zip file

!unzip '/content/drive/MyDrive/IBM/dataset.zip'
```

[2]

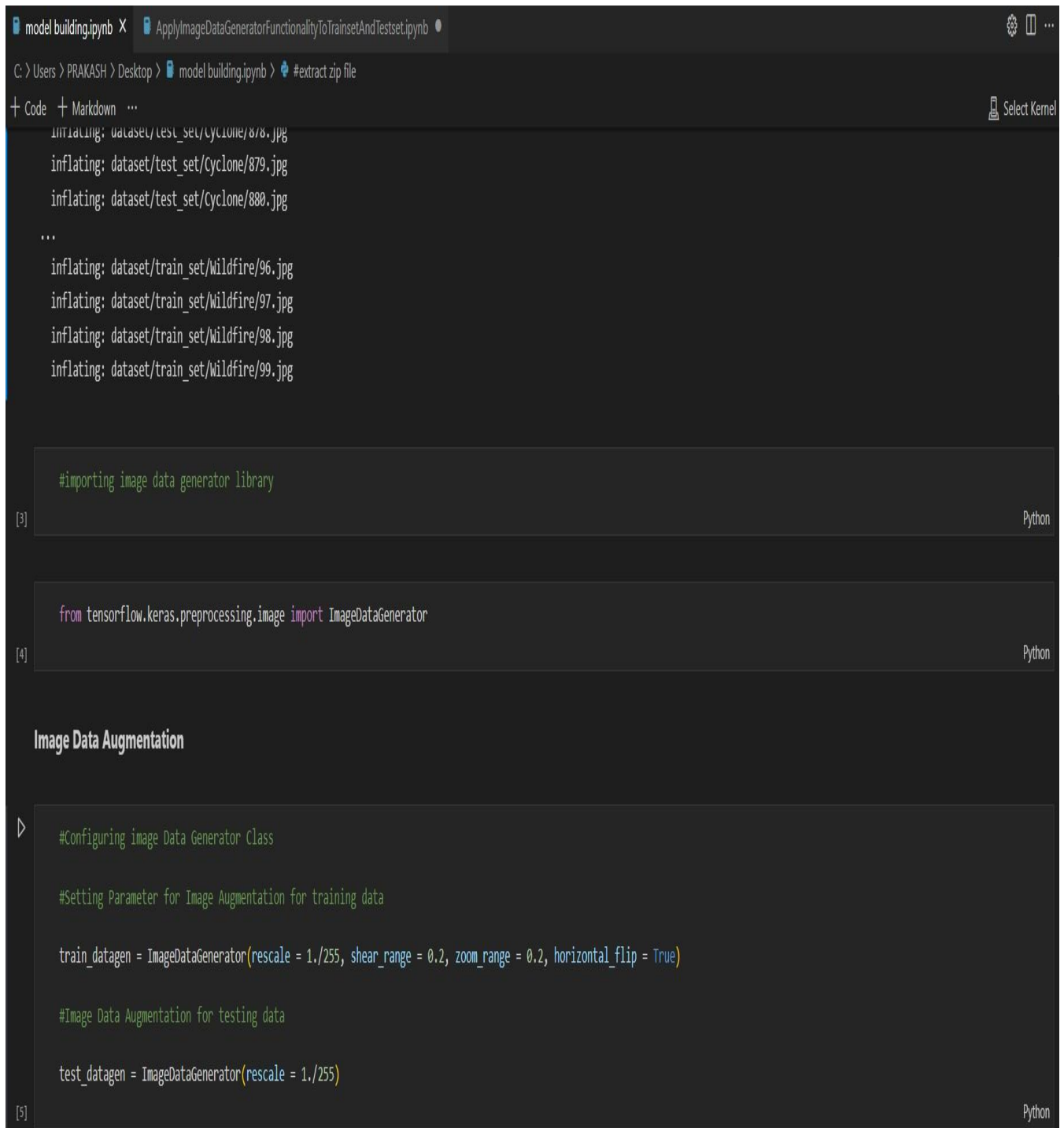
Python

... Output exceeds the [size limit](#). Open the full output data [in a text editor](#)

Archive: /content/drive/MyDrive/IBM/dataset.zip  
replace dataset/readme.txt? [y]es, [n]o, [A]ll, [N]one, [r]ename: yes  
inflating: dataset/readme.txt  
replace dataset/test\_set/Cyclone/867.jpg? [y]es, [n]o, [A]ll, [N]one, [r]ename: yes  
inflating: dataset/test\_set/Cyclone/867.jpg  
replace dataset/test\_set/Cyclone/868.jpg? [y]es, [n]o, [A]ll, [N]one, [r]ename: yes  
inflating: dataset/test\_set/Cyclone/868.jpg  
replace dataset/test\_set/Cyclone/869.jpg? [y]es, [n]o, [A]ll, [N]one, [r]ename: yes  
inflating: dataset/test\_set/Cyclone/869.jpg  
replace dataset/test\_set/Cyclone/870.jpg? [y]es, [n]o, [A]ll, [N]one, [r]ename: y  
inflating: dataset/test\_set/Cyclone/870.jpg  
replace dataset/test\_set/Cyclone/871.jpg? [y]es, [n]o, [A]ll, [N]one, [r]ename: yes  
inflating: dataset/test\_set/Cyclone/871.jpg  
replace dataset/test\_set/Cyclone/872.jpg? [y]es, [n]o, [A]ll, [N]one, [r]ename: y  
inflating: dataset/test\_set/Cyclone/872.jpg  
replace dataset/test\_set/Cyclone/873.jpg? [y]es, [n]o, [A]ll, [N]one, [r]ename: y  
inflating: dataset/test\_set/Cyclone/873.jpg  
replace dataset/test\_set/Cyclone/874.jpg? [y]es, [n]o, [A]ll, [N]one, [r]ename: ALL yes  
inflating: dataset/test\_set/Cyclone/874.jpg  
inflating: dataset/test\_set/Cyclone/875.jpg  
inflating: dataset/test\_set/Cyclone/876.jpg  
inflating: dataset/test\_set/Cyclone/877.jpg  
inflating: dataset/test\_set/Cyclone/878.jpg  
inflating: dataset/test\_set/Cyclone/879.jpg

# Importing image data generator library/Image data Augmentation

Keras Image Data Generator is used for getting the input of the original data and further, it makes the transformation of this data on a random basis and gives the output resultant containing only the data that is newly transformed.



```
model building.ipynb X ApplyImageDataGeneratorFunctionalityToTrainsetAndTestset.ipynb
C:\Users\> PRAKASH > Desktop > model building.ipynb > #extract zip file
+ Code + Markdown ...
initialing: dataset/test_set/cyclone/878.jpg
inflating: dataset/test_set/Cyclone/879.jpg
inflating: dataset/test_set/Cyclone/880.jpg
...
inflating: dataset/train_set/Wildfire/96.jpg
inflating: dataset/train_set/Wildfire/97.jpg
inflating: dataset/train_set/Wildfire/98.jpg
inflating: dataset/train_set/Wildfire/99.jpg

# importing image data generator library
[3] Python

from tensorflow.keras.preprocessing.image import ImageDataGenerator
[4] Python

Image Data Augmentation

#Configuring Image Data Generator Class

#Setting Parameter for Image Augmentation for training data

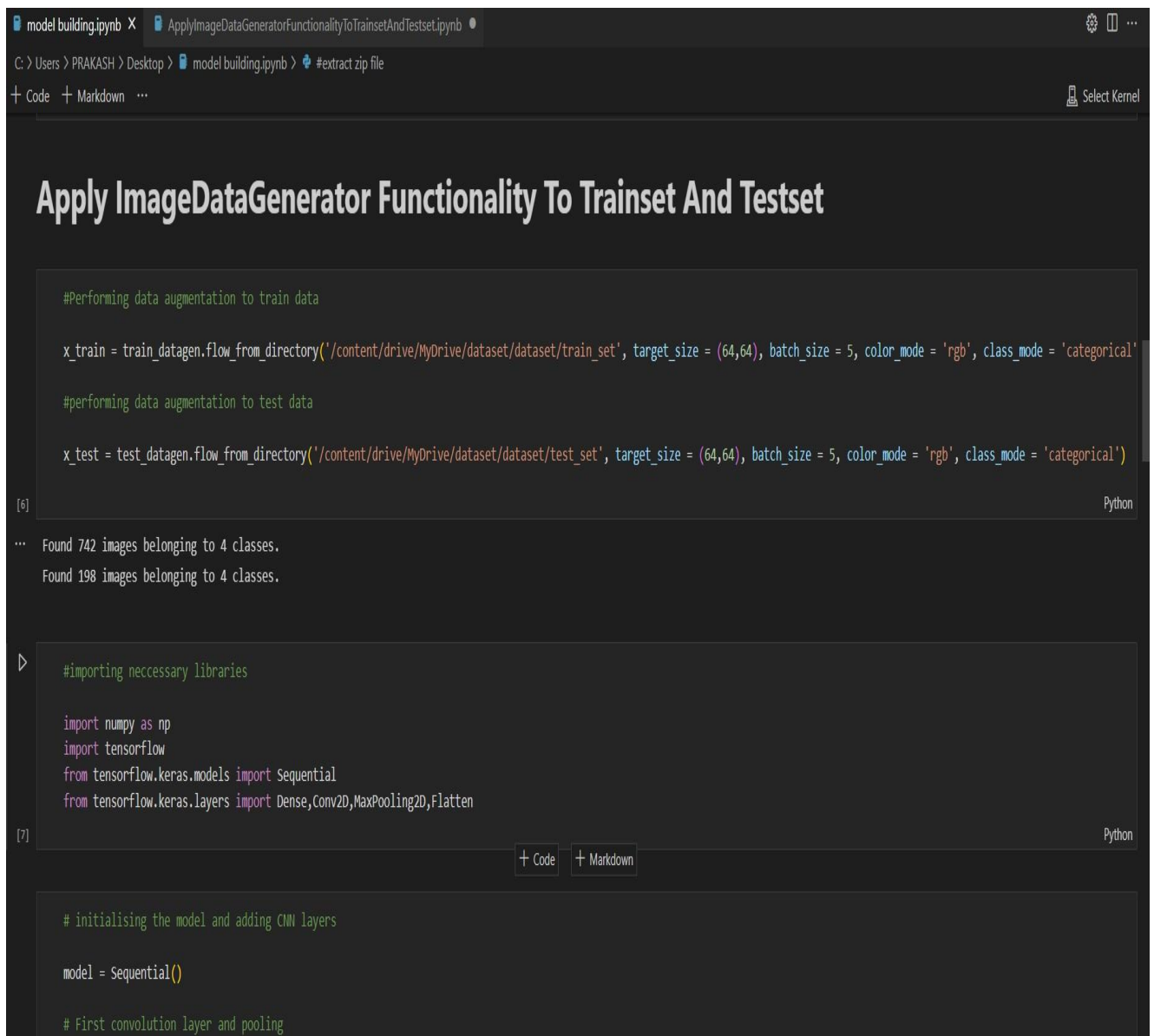
train_datagen = ImageDataGenerator(rescale = 1./255, shear_range = 0.2, zoom_range = 0.2, horizontal_flip = True)

#Image Data Augmentation for testing data

test_datagen = ImageDataGenerator(rescale = 1./255)
[5] Python
```

## Apply Image Data Generator Functionality to train set and test set

You probably encountered a situation where you try to load a dataset but there is not enough memory in your machine. As the field of machine learning progresses, this problem becomes more and more common. Today this is already one of the challenges in the field of vision where large datasets of images and video files are processed



The screenshot shows a Jupyter Notebook with two tabs: 'model building.ipynb' and 'ApplyImageDataGeneratorFunctionalityToTrainsetAndTestset.ipynb'. The active tab is 'ApplyImageDataGeneratorFunctionalityToTrainsetAndTestset.ipynb'. The notebook's title bar shows the file path 'C:\Users\> PRAKASH > Desktop > model building.ipynb' and a file icon. The notebook content includes a title 'Apply ImageDataGenerator Functionality To Trainset And Testset' and two code cells. The first cell, labeled '[6]', contains code for data augmentation on train and test sets. The second cell, labeled '[7]', contains code for importing necessary libraries. The notebook interface includes a 'Select Kernel' button in the top right corner and a 'Python' label at the bottom right of each code cell.

```
#Performing data augmentation to train data

x_train = train_datagen.flow_from_directory('/content/drive/MyDrive/dataset/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/drive/MyDrive/dataset/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.

```
#importing neccessary libraries

import numpy as np
import tensorflow
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense,Conv2D,MaxPooling2D,Flatten
```

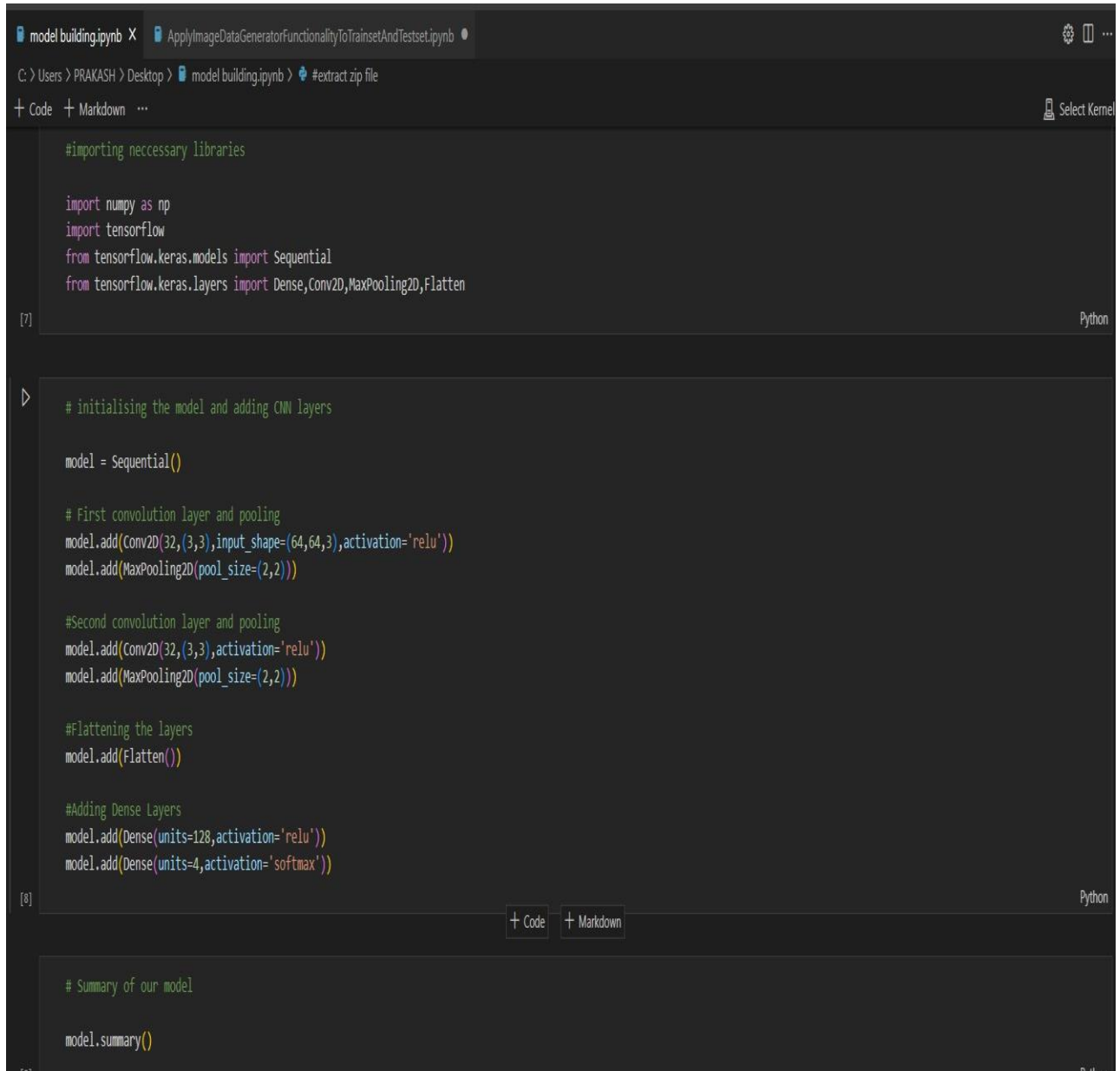
```
# initialising the model and adding CNN layers

model = Sequential()

# First convolution layer and pooling
```

# Importing necessary libraries/Initializing the model and adding CNN layers

TensorFlow is a popular deep learning framework. In this tutorial, you will learn the basics of this Python library and understand how to implement these deep, feed-forward artificial neural networks with it.



The screenshot shows a Jupyter Notebook with two open tabs: 'model building.ipynb' and 'ApplyImageDataGeneratorFunctionalityToTrainsetAndTestset.ipynb'. The active tab is 'model building.ipynb', which displays the following code in a dark-themed editor:

```
#importing necessary libraries

import numpy as np
import tensorflow
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, Conv2D, MaxPooling2D, Flatten

# 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'))

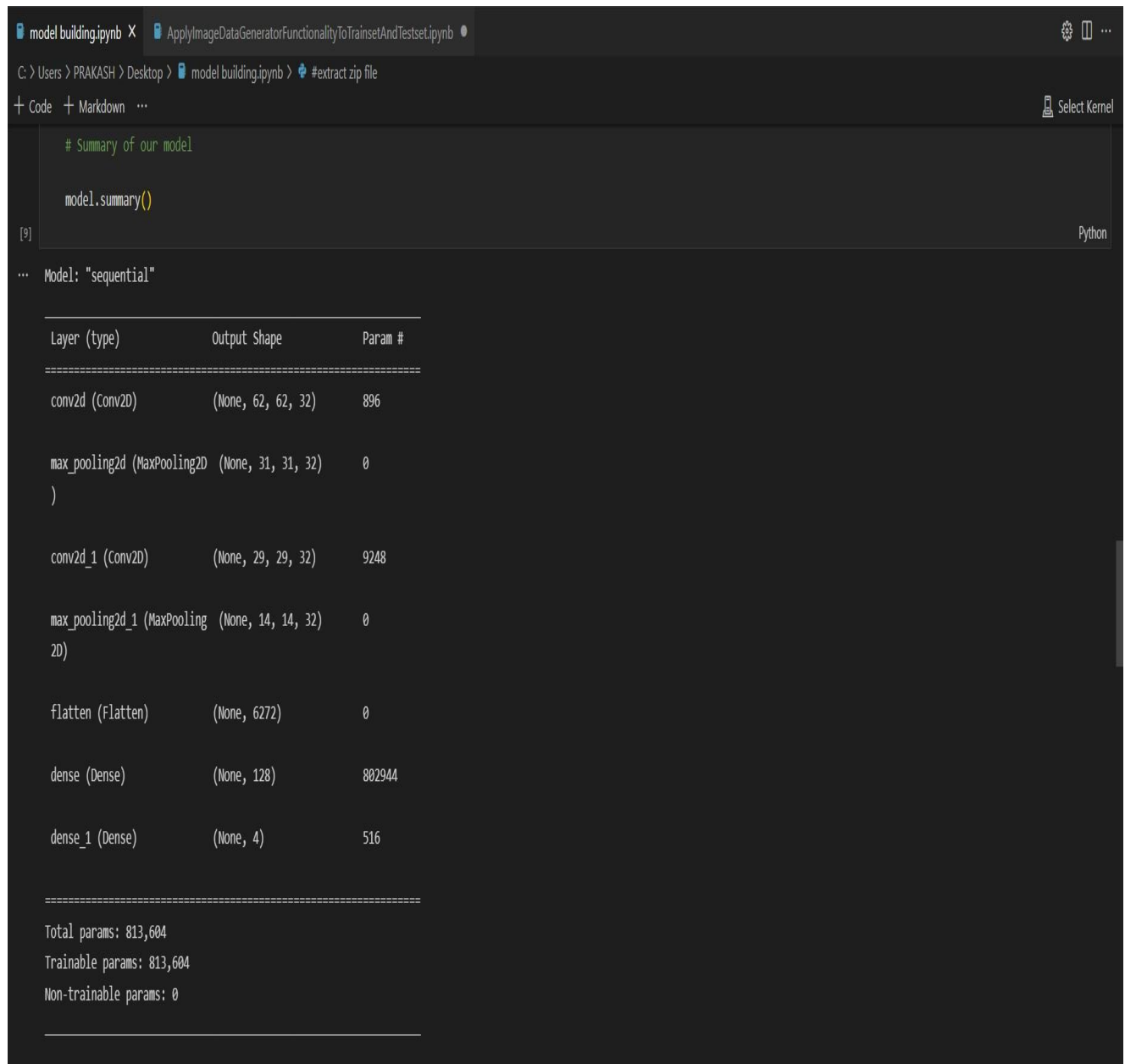
# Summary of our model

model.summary()
```

The code is organized into three sections, each with a comment header. The first section imports necessary libraries (numpy, tensorflow, and keras models and layers). The second section initializes the model and adds CNN layers, including two convolutional layers with pooling, a flattening layer, and two dense layers. The third section provides a summary of the model. The notebook interface includes a file explorer on the left, a command prompt at the top showing the current directory, and a toolbar with options like '+ Code' and '+ Markdown'.

## Summary of our model

The model summary gives us a fine visualization of our model and the aim is to provide complete information that is not provided by the print statement.



```
# Summary of our model

model.summary()
```

[9] Python

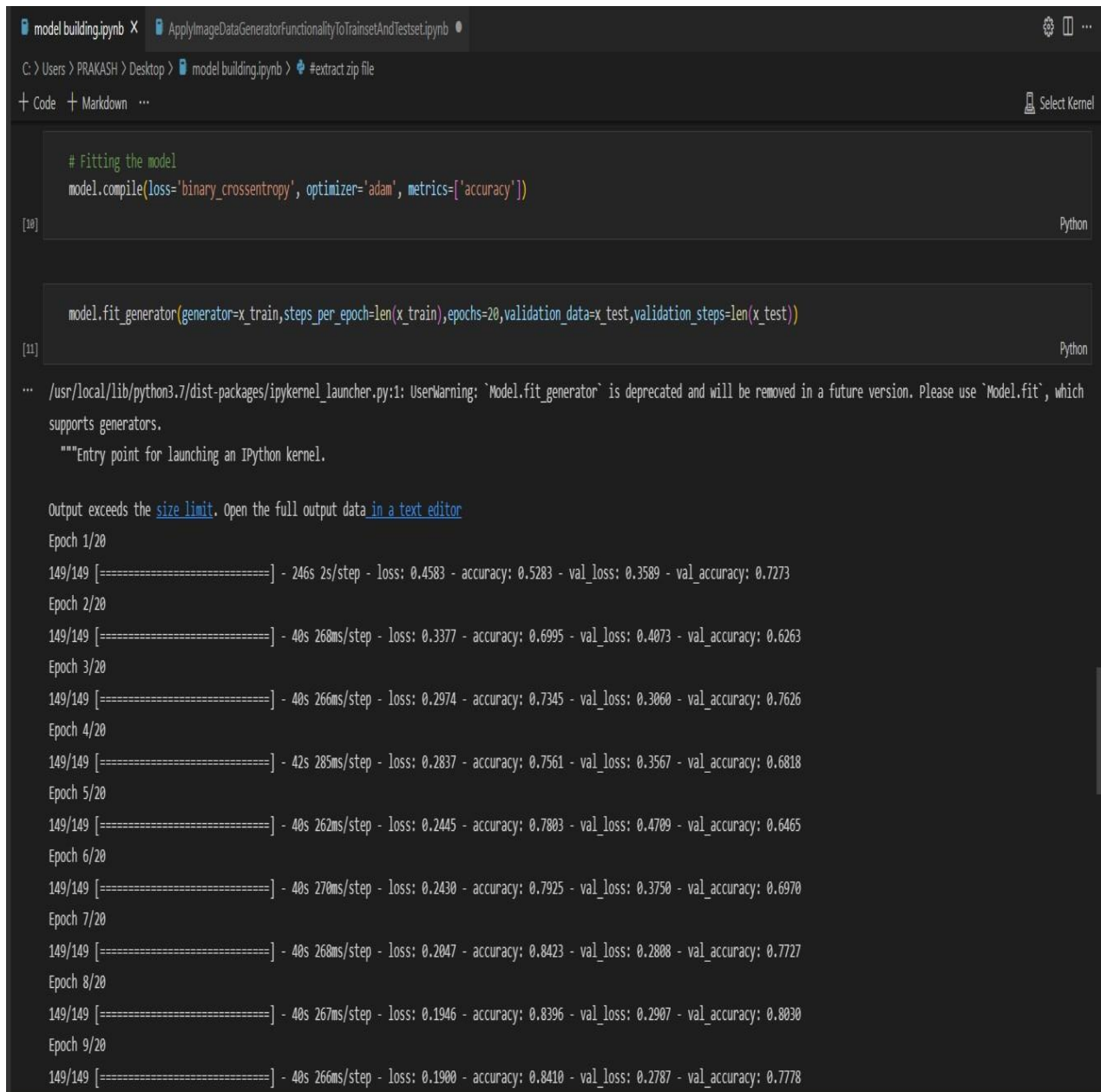
... Model: "sequential"

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  
=====

## Fitting the model

We'll define the Keras sequential model and add a one-dimensional convolutional layer. Input shape becomes as it is confirmed above We'll add Dense, MaxPooling1D, and Flatten layers into the model. The output layer contains the number of output classes and 'SoftMax' activation.



```
# Fitting the model
model.compile(loss='binary_crossentropy', optimizer='adam', metrics=['accuracy'])

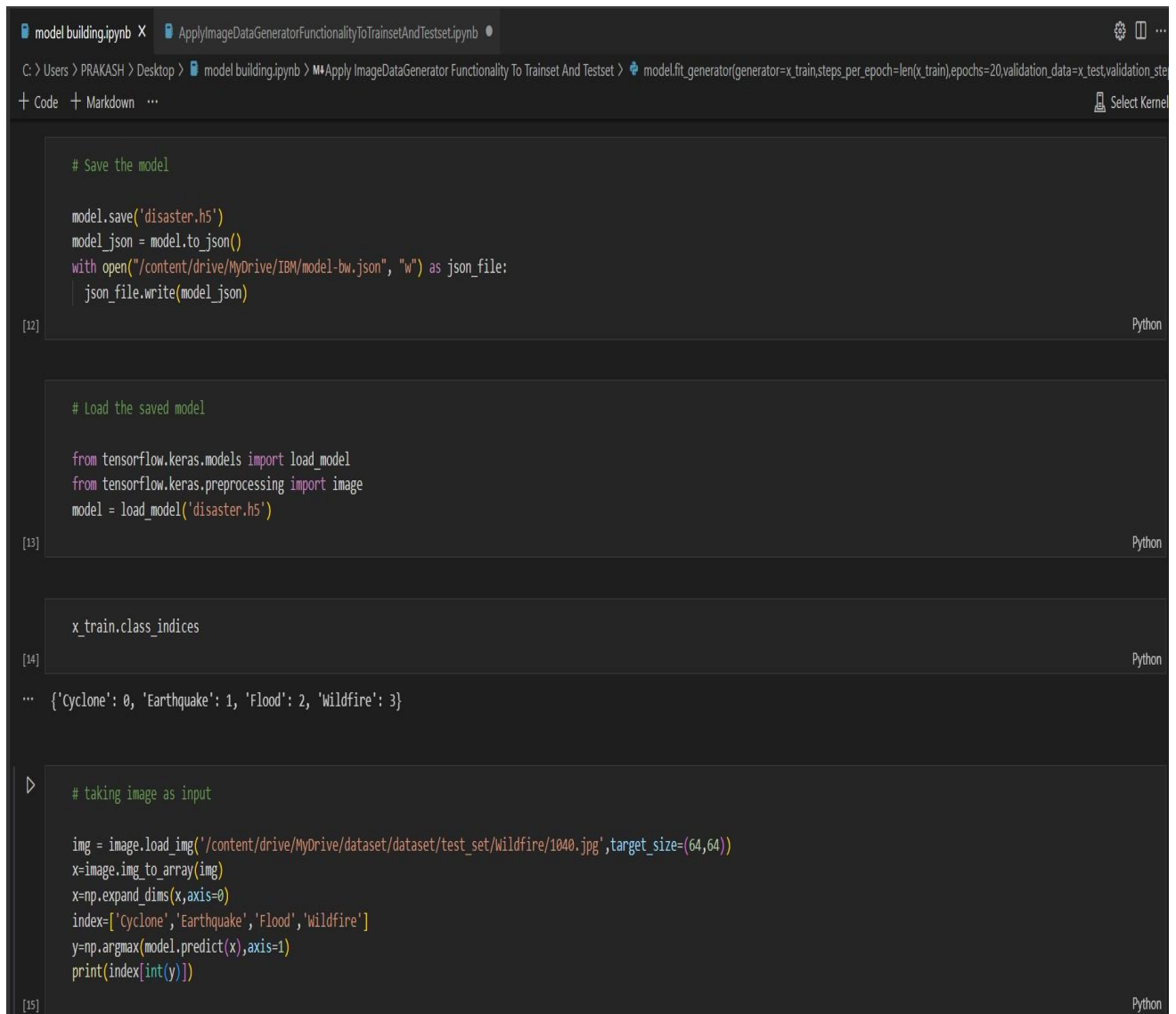
model.fit_generator(generator=x_train, steps_per_epoch=len(x_train), epochs=20, validation_data=x_test, validation_steps=len(x_test))
```

Output exceeds the [size limit](#). Open the full output data [in a text editor](#)

Epoch 1/20  
149/149 [=====] - 246s 2s/step - loss: 0.4583 - accuracy: 0.5283 - val\_loss: 0.3589 - val\_accuracy: 0.7273  
Epoch 2/20  
149/149 [=====] - 40s 268ms/step - loss: 0.3377 - accuracy: 0.6995 - val\_loss: 0.4073 - val\_accuracy: 0.6263  
Epoch 3/20  
149/149 [=====] - 40s 266ms/step - loss: 0.2974 - accuracy: 0.7345 - val\_loss: 0.3060 - val\_accuracy: 0.7626  
Epoch 4/20  
149/149 [=====] - 42s 285ms/step - loss: 0.2837 - accuracy: 0.7561 - val\_loss: 0.3567 - val\_accuracy: 0.6818  
Epoch 5/20  
149/149 [=====] - 40s 262ms/step - loss: 0.2445 - accuracy: 0.7803 - val\_loss: 0.4709 - val\_accuracy: 0.6465  
Epoch 6/20  
149/149 [=====] - 40s 270ms/step - loss: 0.2430 - accuracy: 0.7925 - val\_loss: 0.3750 - val\_accuracy: 0.6970  
Epoch 7/20  
149/149 [=====] - 40s 268ms/step - loss: 0.2047 - accuracy: 0.8423 - val\_loss: 0.2808 - val\_accuracy: 0.7727  
Epoch 8/20  
149/149 [=====] - 40s 267ms/step - loss: 0.1946 - accuracy: 0.8396 - val\_loss: 0.2907 - val\_accuracy: 0.8030  
Epoch 9/20  
149/149 [=====] - 40s 266ms/step - loss: 0.1900 - accuracy: 0.8410 - val\_loss: 0.2787 - val\_accuracy: 0.7778

## Save the model/Load the saved model/Taking image as input

The Saved Model format is another way to serialize models. Models saved in this format can be restored using and are compatible with TensorFlow Serving. The Saved Model goes into detail about how to serve/inspect the Saved Model. The section below illustrates the steps to save and restore the model.



```
model building.ipynb X ApplyImageDataGeneratorFunctionalityToTrainsetAndTestset.ipynb
C:\Users\PRAKASH\Desktop> model building.ipynb > M*Apply ImageDataGenerator Functionality To Trainset And Testset > model.fit_generator(generator=x_train,steps_per_epoch=len(x_train),epochs=20,validation_data=x_test,validation_steps=10)
+ Code + Markdown ... Select Kernel

# Save the model

model.save('disaster.h5')
model_json = model.to_json()
with open("/content/drive/MyDrive/IBM/model-bw.json", "w") as json_file:
    json_file.write(model_json)

[12] Python

# Load the saved model

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

[13] Python

x_train.class_indices

[14] Python

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

# taking image as input

img = image.load_img('/content/drive/MyDrive/dataset/dataset/test_set/Wildfire/1040.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)])

[15] Python
```