*#extract zip file*

!unzip '/content/dataset.zip'

In [ ]:

*#importing image data generator library*

In [21]:

**from** tensorflow.keras.preprocessing.image **import** ImageDataGenerator

Image Data Augmentation

In [22]:

*#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)

**Apply ImageDataGenerator Functionality To Trainset And Testset**

In [23]:

*#Performing data augmentation to train data*

x\_train **=** train\_datagen**.**flow\_from\_directory('/content/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/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.

In [25]:

*#importing neccessary libraries*

**import** numpy **as** np

**import** tensorflow

**from** tensorflow.keras.models **import** Sequential

**from** tensorflow.keras.layers **import** Dense,Conv2D,MaxPooling2D,Flatten

In [26]:

*# 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'))

In [27]:

*# Summary of our model*

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

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

In [28]:

*# Compiling the model*

model**.**compile(optimizer**=**'adam', loss**=**'categorical\_crossentropy', metrics**=**['accuracy'])

In [29]:

*# Fitting the model*

model**.**fit\_generator(generator**=**x\_train,steps\_per\_epoch**=**len(x\_train),epochs**=**20,validation\_data**=**x\_test,validation\_steps**=**len(x\_test))

/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 1/20

149/149 [==============================] - 42s 272ms/step - loss: 1.1632 - accuracy: 0.5216 - val\_loss: 0.9812 - val\_accuracy: 0.5354

Epoch 2/20

149/149 [==============================] - 38s 252ms/step - loss: 0.8483 - accuracy: 0.6712 - val\_loss: 0.7859 - val\_accuracy: 0.7374

Epoch 3/20

149/149 [==============================] - 39s 263ms/step - loss: 0.6883 - accuracy: 0.7278 - val\_loss: 0.8899 - val\_accuracy: 0.6970

Epoch 4/20

149/149 [==============================] - 41s 279ms/step - loss: 0.6571 - accuracy: 0.7170 - val\_loss: 1.0388 - val\_accuracy: 0.6111

Epoch 5/20

149/149 [==============================] - 38s 253ms/step - loss: 0.5828 - accuracy: 0.7655 - val\_loss: 0.7886 - val\_accuracy: 0.7525

Epoch 6/20

149/149 [==============================] - 39s 265ms/step - loss: 0.5124 - accuracy: 0.8113 - val\_loss: 0.9449 - val\_accuracy: 0.6616

Epoch 7/20

149/149 [==============================] - 38s 252ms/step - loss: 0.4475 - accuracy: 0.8208 - val\_loss: 0.9295 - val\_accuracy: 0.7626

Epoch 8/20

149/149 [==============================] - 37s 253ms/step - loss: 0.5198 - accuracy: 0.8208 - val\_loss: 1.0729 - val\_accuracy: 0.7172

Epoch 9/20

149/149 [==============================] - 39s 261ms/step - loss: 0.4103 - accuracy: 0.8423 - val\_loss: 1.0310 - val\_accuracy: 0.6768

Epoch 10/20

149/149 [==============================] - 42s 280ms/step - loss: 0.4223 - accuracy: 0.8491 - val\_loss: 0.7108 - val\_accuracy: 0.7929

Epoch 11/20

149/149 [==============================] - 38s 254ms/step - loss: 0.4170 - accuracy: 0.8544 - val\_loss: 0.8419 - val\_accuracy: 0.7121

Epoch 12/20

149/149 [==============================] - 40s 268ms/step - loss: 0.3207 - accuracy: 0.8841 - val\_loss: 0.7221 - val\_accuracy: 0.8030

Epoch 13/20

149/149 [==============================] - 38s 249ms/step - loss: 0.3373 - accuracy: 0.8585 - val\_loss: 0.9803 - val\_accuracy: 0.7525

Epoch 14/20

149/149 [==============================] - 43s 287ms/step - loss: 0.3147 - accuracy: 0.8922 - val\_loss: 1.3861 - val\_accuracy: 0.6667

Epoch 15/20

149/149 [==============================] - 40s 267ms/step - loss: 0.2967 - accuracy: 0.8841 - val\_loss: 1.0562 - val\_accuracy: 0.7626

Epoch 16/20

149/149 [==============================] - 39s 261ms/step - loss: 0.2683 - accuracy: 0.9003 - val\_loss: 0.9182 - val\_accuracy: 0.8182

Epoch 17/20

149/149 [==============================] - 40s 272ms/step - loss: 0.2650 - accuracy: 0.8976 - val\_loss: 1.0180 - val\_accuracy: 0.7677

Epoch 18/20

149/149 [==============================] - 39s 264ms/step - loss: 0.2661 - accuracy: 0.9164 - val\_loss: 0.8409 - val\_accuracy: 0.7929

Epoch 19/20

149/149 [==============================] - 38s 256ms/step - loss: 0.2089 - accuracy: 0.9313 - val\_loss: 1.0649 - val\_accuracy: 0.7677

Epoch 20/20

149/149 [==============================] - 37s 251ms/step - loss: 0.2323 - accuracy: 0.9111 - val\_loss: 0.9940 - val\_accuracy: 0.7879

Out[29]:

In [34]:

*# Save the model*

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

model\_json **=** model**.**to\_json()

**with** open("model-bw.json", "w") **as** json\_file:

json\_file**.**write(model\_json)

In [35]:

*# Load the saved model*

**from** tensorflow.keras.models **import** load\_model

**from** tensorflow.keras.preprocessing **import** image

model **=** load\_model('disaster.h5')

In [46]:

x\_train**.**class\_indices

Out[46]:

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

In [56]:

*# taking image as input*

img **=** image**.**load\_img('/content/dataset/test\_set/Flood/1003.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 22ms/step

Flood

In [59]:

*# input 2*

img **=** image**.**load\_img('/content/dataset/test\_set/Wildfire/1065.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 27ms/step

Wildfire