```
#extract zip file

!unzip '/content/dataset.zip'
#importing image data generator library
from tensorflow.keras.preprocessing.image import ImageDataGenerator
```

Image Data Agumentation

```
#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 Image Data Generator Functionality To Train And Test Database

```
#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.
```

```
#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
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()
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
_________________________________________________________________
```

```
# Compiling the model

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

```
# 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)
# Load the saved model

from tensorflow.keras.models import load_model
from tensorflow.keras.preprocessing import image
model = load_model('disaster.h5')
x_train.class_indices
{'Cyclone': 0, 'Earthquake': 1, 'Flood': 2, 'Wildfire': 3}
# 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
# 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
```