

## Project Development Phase Sprint - II

Date	05 November 2022
Team ID	PNT2022TMID29513
Project Name	Natural Disasters Intensity Analysis And Classification Using Artificial Intelligence
Maximum Marks	4 Marks

### Building the CNN Model for Natural Disaster Classification, Training and Validating it, and Testing results

Link:

[https://dataplatform.cloud.ibm.com/analytics/notebooks/v2/6cbeecfd-214b-4a41-aea0-dcd3dafabe61/view?access\\_token=cc793da694f128bd71a83af2dd03af6db746baa06c11850fce55b299b697b05a](https://dataplatform.cloud.ibm.com/analytics/notebooks/v2/6cbeecfd-214b-4a41-aea0-dcd3dafabe61/view?access_token=cc793da694f128bd71a83af2dd03af6db746baa06c11850fce55b299b697b05a)

The screenshot displays the IBM Watson Studio interface. The top navigation bar includes the IBM Watson Studio logo, a search bar, and user account information (Raakesh Kumar C's Account, Dallas). The main workspace shows a Jupyter Notebook titled "Disaster Classification". The notebook code is as follows:

```
In [27]: #Model Fitting - training and validation
history=model.fit_generator(x_train,steps_per_epoch=len(x_train),epochs=20,validation_data=x_val,validation_steps=len(x_val))
#steps_per_epoch = no of train images/batch size
#validation_steps = no of test images/batch size
```

The output of the notebook shows the training and validation progress over 20 epochs. The results are as follows:

Epoch	Time	Loss	Accuracy	Val Loss	Val Accuracy
Epoch 1/20	384s	1.1158	0.5341	1.0172	0.6244
Epoch 2/20	380s	0.7026	0.7391	0.6189	0.7862
Epoch 3/20	378s	0.6090	0.7785	0.5414	0.8032
Epoch 4/20	378s	0.5252	0.8082	0.5541	0.8133
Epoch 5/20	375s	0.4850	0.8205	0.5834	0.7805
Epoch 6/20	376s	0.4537	0.8305	0.4234	0.8450
Epoch 7/20	378s	0.4294	0.8437	0.4307	0.8529
Epoch 8/20	379s	0.4033	0.8524	0.3810	0.8767
Epoch 9/20	379s	0.3695	0.8631	0.3842	0.8586
Epoch 10/20	383s	0.3419	0.8760	0.3953	0.8620
Epoch 11/20	380s	0.3305	0.8728	0.4521	0.8541
Epoch 12/20	383s	0.3426	0.8731	0.4412	0.8529
Epoch 13/20	378s	0.3359	0.8763	0.4537	0.8541
Epoch 14/20	384s	0.3157	0.8834	0.4474	0.8439
Epoch 15/20	380s	0.2992	0.8967	0.5975	0.8213

The right sidebar shows the "Data" section with a "Files" tab. It indicates that files can be uploaded and provides a link to "Cyclone\_Wildfi...ake\_Database.zip".

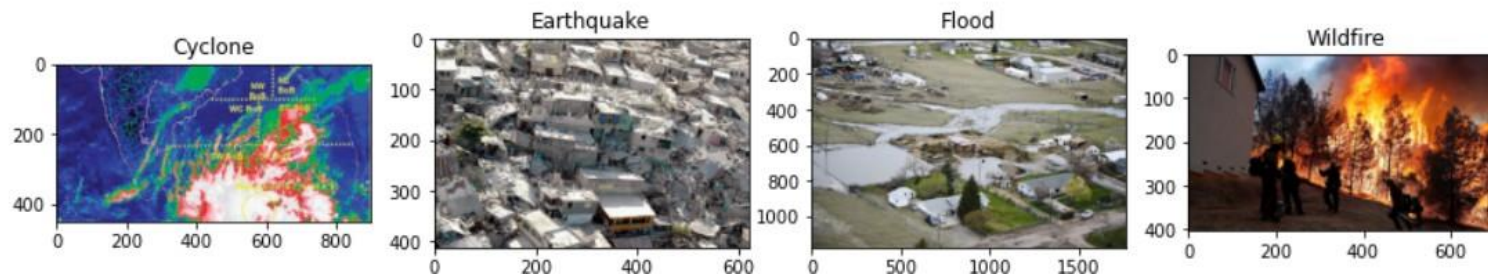
## 1. Indexing Disaster Classes

```
In [19]: #Classes of Disasters  
x_train.class_indices
```

```
Out[19]: {'Cyclone': 0, 'Earthquake': 1, 'Flood': 2, 'Wildfire': 3}
```

## 2. Sample Plot for each of the Classes

```
In [20]: #Sample Plot for each of the Classes  
from skimage import io  
f=['/home/wsuser/work/dataset/train/Cyclone/1.jpg', '/home/wsuser/work/dataset/train/Earthquake/0.jpg', '/home/wsuser/work/dataset/train/Flood/0.jpg', '/home/wsuser/work/dataset/train/Wildfire/0.jpg']  
class_names=['Cyclone', 'Earthquake', 'Flood', 'Wildfire']  
x, axarr = plt.subplots(1,4,figsize=(15,15))  
for i in range(4):  
    axarr[i].imshow(io.imread(f[i]))  
    axarr[i].title.set_text(class_names[i])
```



### 3. CNN Model Architecture

```
In [21]: model=Sequential()
```

```
In [22]: #Input Convolution Layer
model.add(Convolution2D(32,kernel_size=(3,3),input_shape=(299,299,3),strides=(1,1),activation='relu'))
model.add(MaxPooling2D(pool_size=(2,2)))
#Convolution Layer 2
model.add(Convolution2D(64,kernel_size=(3,3),input_shape=(299,299,3),strides=(1,1),activation='relu'))
model.add(MaxPooling2D(pool_size=(2,2)))
model.add(Dropout(0.3))
#Convolution Layer 3
model.add(Convolution2D(32,kernel_size=(3,3),input_shape=(299,299,3),strides=(1,1),activation='relu'))
model.add(MaxPooling2D(pool_size=(2,2)))
model.add(Dropout(0.3))
#Flattening of Output
model.add(Flatten())
#FCN or Dense Layer
model.add(Dense(units=256,kernel_initializer="random_uniform",activation="relu"))
model.add(Dropout(0.4))
#Output Layer
model.add(Dense(units=4,activation="softmax"))
```

### 4. Summary of the Model

```
model.summary()
```

Model: "sequential"

Layer (type)	Output Shape	Param #
conv2d (Conv2D)	(None, 297, 297, 32)	896
max_pooling2d (MaxPooling2D)	(None, 148, 148, 32)	0
conv2d_1 (Conv2D)	(None, 146, 146, 64)	18496
max_pooling2d_1 (MaxPooling2D)	(None, 73, 73, 64)	0
dropout (Dropout)	(None, 73, 73, 64)	0
conv2d_2 (Conv2D)	(None, 71, 71, 32)	18464
max_pooling2d_2 (MaxPooling2D)	(None, 35, 35, 32)	0
dropout_1 (Dropout)	(None, 35, 35, 32)	0
flatten (Flatten)	(None, 39200)	0
dense (Dense)	(None, 256)	10035456
dropout_2 (Dropout)	(None, 256)	0
dense_1 (Dense)	(None, 4)	1028

Total params: 10,074,340

## 5. Compiling the Model

```
In [24]: #Compiling the Model
model.compile(loss="categorical_crossentropy",optimizer="adam",metrics=["accuracy"])
```

## 6. Training and Validating the Model

```
In [27]: #Model Fitting - training and validation
history=model.fit_generator(x_train,steps_per_epoch=len(x_train),epochs=20,validation_data=x_val,validation_steps=len(x_val),
#steps_per_epoch = no of train images/batch size
#validation_steps = no of test images/batch size
Epoch 1/20
310/310 [=====] - 384s 1s/step - loss: 1.1158 - accuracy: 0.5341 - val_loss: 1.0172 - val_accuracy: 0.6244
Epoch 2/20
310/310 [=====] - 380s 1s/step - loss: 0.7026 - accuracy: 0.7391 - val_loss: 0.6189 - val_accuracy: 0.7862
Epoch 3/20
310/310 [=====] - 378s 1s/step - loss: 0.6090 - accuracy: 0.7785 - val_loss: 0.5414 - val_accuracy: 0.8032
Epoch 4/20
310/310 [=====] - 378s 1s/step - loss: 0.5252 - accuracy: 0.8082 - val_loss: 0.5541 - val_accuracy: 0.8133
Epoch 5/20
310/310 [=====] - 375s 1s/step - loss: 0.4850 - accuracy: 0.8205 - val_loss: 0.5834 - val_accuracy: 0.7805
Epoch 6/20
310/310 [=====] - 376s 1s/step - loss: 0.4537 - accuracy: 0.8305 - val_loss: 0.4234 - val_accuracy: 0.8450
Epoch 7/20
310/310 [=====] - 378s 1s/step - loss: 0.4294 - accuracy: 0.8437 - val_loss: 0.4307 - val_accuracy: 0.8529
Epoch 8/20
310/310 [=====] - 379s 1s/step - loss: 0.4033 - accuracy: 0.8524 - val_loss: 0.3810 - val_accuracy: 0.8767
Epoch 9/20
310/310 [=====] - 379s 1s/step - loss: 0.3695 - accuracy: 0.8631 - val_loss: 0.3842 - val_accuracy: 0.8586
Epoch 10/20
310/310 [=====] - 383s 1s/step - loss: 0.3419 - accuracy: 0.8760 - val_loss: 0.3953 - val_accuracy: 0.8760
```

---

```
Epoch 11/20
310/310 [=====] - 380s 1s/step - loss: 0.3305 - accuracy: 0.8728 - val_loss: 0.4521 - val_ac
curacy: 0.8541
Epoch 12/20
310/310 [=====] - 383s 1s/step - loss: 0.3426 - accuracy: 0.8731 - val_loss: 0.4412 - val_ac
curacy: 0.8529
Epoch 13/20
310/310 [=====] - 378s 1s/step - loss: 0.3359 - accuracy: 0.8763 - val_loss: 0.4537 - val_ac
curacy: 0.8541
Epoch 14/20
310/310 [=====] - 384s 1s/step - loss: 0.3157 - accuracy: 0.8834 - val_loss: 0.4474 - val_ac
curacy: 0.8439
Epoch 15/20
310/310 [=====] - 380s 1s/step - loss: 0.2992 - accuracy: 0.8967 - val_loss: 0.5975 - val_ac
curacy: 0.8213
Epoch 16/20
310/310 [=====] - 380s 1s/step - loss: 0.2774 - accuracy: 0.8999 - val_loss: 0.4088 - val_ac
curacy: 0.8597
Epoch 17/20
310/310 [=====] - 384s 1s/step - loss: 0.2534 - accuracy: 0.9109 - val_loss: 0.3894 - val_ac
curacy: 0.8643
Epoch 18/20
310/310 [=====] - 379s 1s/step - loss: 0.2779 - accuracy: 0.9015 - val_loss: 0.4040 - val_ac
curacy: 0.8665
Epoch 19/20
310/310 [=====] - 379s 1s/step - loss: 0.2430 - accuracy: 0.9093 - val_loss: 0.4831 - val_ac
curacy: 0.8529
Epoch 20/20
310/310 [=====] - 381s 1s/step - loss: 0.2490 - accuracy: 0.9073 - val_loss: 0.4113 - val_ac
curacy: 0.8801
```

## 7. Saving the Model as .h5 file and json file

```
In [28]: len(x_train)
```

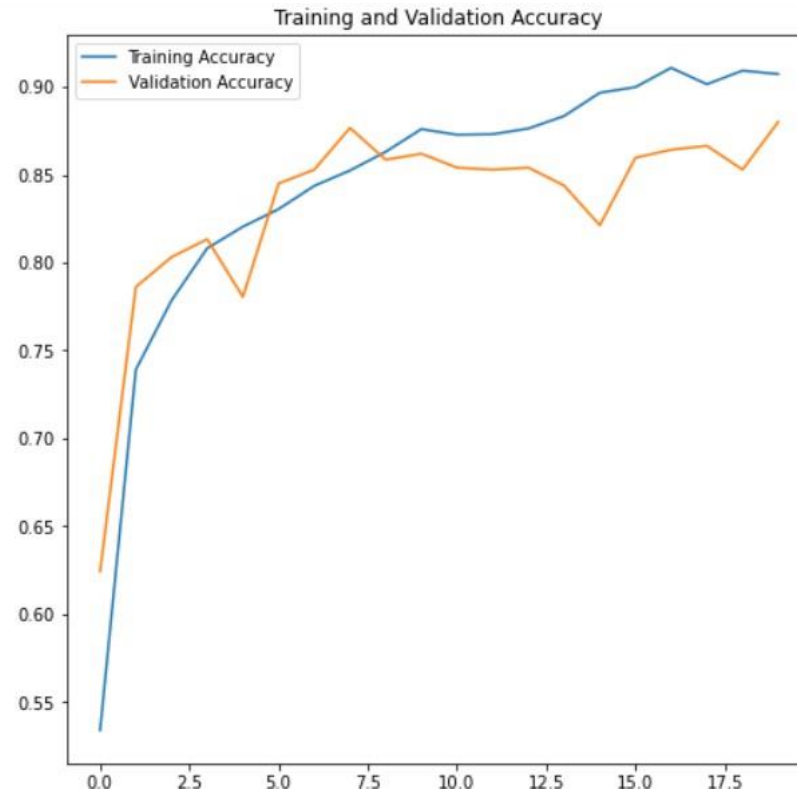
```
Out[28]: 310
```

```
In [29]: #saving the Model
model.save('Disaster_Classifier.h5')
model_json=model.to_json()
with open("model-bw.json","w") as json_file:
    json_file.write(model_json)
```

## 8. Plots for training vs validation accuracies and losses

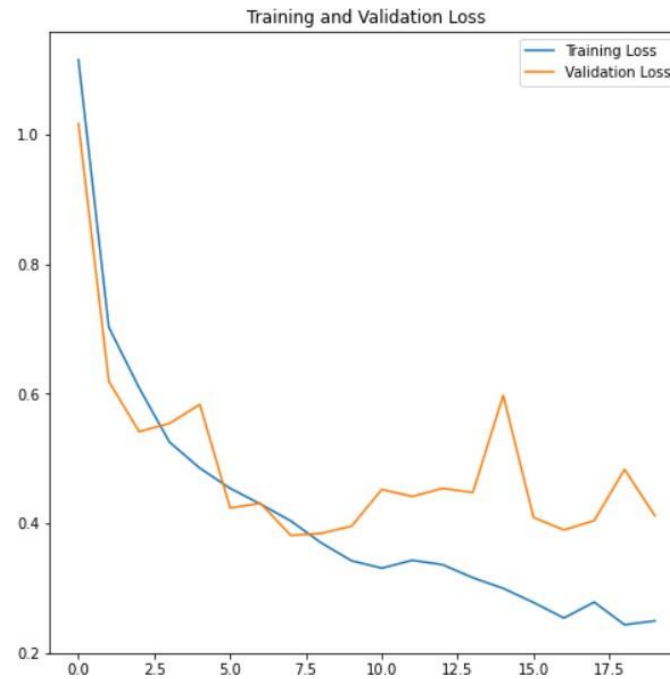
```
In [30]: #Training and Validation Accuracy Plots
epochs_range = range(20)

plt.figure(figsize=(8, 8))
plt.plot(epochs_range, history.history['accuracy'], label='Training Accuracy')
plt.plot(epochs_range, history.history['val_accuracy'], label='Validation Accuracy')
plt.legend()
plt.title('Training and Validation Accuracy')
plt.show()
```





```
In [31]: #Training and Validation Loss Plot
plt.figure(figsize=(8, 8))
plt.plot(epochs_range, history.history['loss'], label='Training Loss')
plt.plot(epochs_range, history.history['val_loss'], label='Validation Loss')
plt.legend()
plt.title('Training and Validation Loss')
plt.show()
```



## 9. Testing the CNN Model with test data

```
In [35]: #Testing the CNN Model with test data
test_generator=test_datagen.flow_from_directory(r"/home/wsuser/work/dataset/test",
                                                target_size=(299,299),
                                                batch_size=447,
                                                color_mode='rgb',
                                                class_mode='categorical')
```

Found 447 images belonging to 4 classes.

```
In [36]: x_test, y_test = test_generator.__getitem__(0)
```

```
In [37]: y_test
```

```
Out[37]: array([[1., 0., 0., 0.],
               [0., 1., 0., 0.],
               [0., 0., 0., 1.],
               ...,
               [0., 0., 0., 1.],
               [0., 0., 0., 1.],
               [0., 0., 1., 0.]], dtype=float32)
```

```
In [38]: #predicting the labels of test data
y_pred = model.predict(x_test)
```

```
In [39]: y_pred = np.argmax(y_pred,axis=1)
```

```
In [40]: y_pred
```

```
Out[40]: array([0, 1, 3, 3, 1, 3, 2, 1, 1, 0, 0, 2, 3, 0, 2, 3, 3, 1, 1, 2, 2, 3,
                2, 0, 1, 3, 1, 3, 0, 1, 3, 0, 1, 3, 0, 1, 3, 2, 2, 1, 3, 1, 1, 0,
                2, 3, 2, 3, 2, 1, 3, 1, 2, 0, 1, 3, 0, 3, 3, 0, 0, 2, 0, 2, 0, 1,
                3, 1, 3, 0, 0, 0, 2, 1, 0, 1, 1, 0, 2, 2, 1, 0, 1, 0, 3, 3, 3, 2,
                2, 1, 1, 2, 2, 1, 1, 3, 1, 2, 3, 3, 1, 3, 0, 0, 1, 1, 1, 0, 0, 1,
                1, 2, 1, 0, 0, 1, 2, 2, 1, 2, 3, 1, 1, 2, 2, 1, 0, 1, 1, 1, 2, 1,
                3, 1, 0, 3, 2, 1, 2, 1, 3, 2, 2, 0, 1, 2, 0, 1, 1, 3, 0, 1, 0, 1,
                3, 1, 2, 1, 1, 1, 0, 1, 0, 3, 2, 0, 3, 0, 0, 0, 1, 0, 0, 2, 3, 2,
                0, 0, 1, 0, 0, 2, 2, 1, 0, 3, 1, 1, 1, 2, 0, 3, 1, 2, 3, 2, 0, 0,
                3, 1, 2, 1, 3, 3, 2, 0, 0, 2, 3, 1, 2, 2, 3, 1, 3, 1, 0, 0, 3, 1,
                3, 0, 1, 2, 2, 3, 1, 2, 2, 1, 1, 2, 1, 0, 1, 1, 2, 2, 2, 1, 0, 2,
                2, 3, 0, 1, 1, 3, 1, 0, 2, 2, 3, 0, 0, 3, 1, 0, 1, 1, 1, 1, 2, 0,
                3, 2, 0, 0, 3, 2, 3, 1, 1, 0, 1, 1, 2, 3, 1, 2, 0, 3, 3, 3, 1, 2,
                2, 2, 2, 2, 3, 3, 2, 1, 1, 1, 1, 3, 2, 3, 2, 1, 2, 2, 3, 2, 3, 2,
                2, 1, 3, 2, 2, 1, 1, 2, 0, 1, 2, 2, 3, 1, 2, 1, 2, 1, 2, 1, 3, 2,
                3, 2, 2, 3, 1, 3, 1, 3, 1, 0, 1, 2, 2, 2, 3, 0, 0, 2, 3, 3, 3, 1,
                2, 1, 3, 1, 1, 2, 0, 3, 2, 2, 0, 3, 1, 1, 1, 1, 1, 0, 1, 2, 0, 3,
                2, 0, 2, 2, 0, 1, 3, 3, 3, 2, 2, 2, 1, 1, 2, 0, 3, 1, 2, 1, 1, 1,
                2, 0, 3, 1, 2, 0, 2, 1, 3, 2, 3, 3, 1, 3, 2, 2, 1, 0, 3, 0, 0, 1,
                3, 3, 2, 2, 0, 1, 0, 2, 1, 2, 0, 1, 2, 1, 1, 3, 2, 3, 3, 1, 1, 1,
                3, 3, 0, 0, 3, 3, 2])
```

```
In [41]: y_test = np.argmax(y_test, axis=1)
```



```
In [42]: y_test
Out[42]: array([0, 1, 3, 3, 1, 3, 2, 1, 1, 0, 0, 2, 3, 0, 2, 3, 3, 2, 1, 3, 2, 3,
                0, 0, 1, 3, 1, 3, 0, 1, 3, 0, 1, 3, 0, 1, 3, 1, 2, 1, 3, 1, 1, 0,
                2, 3, 1, 3, 2, 1, 3, 0, 2, 0, 1, 3, 2, 3, 3, 0, 0, 2, 0, 2, 0, 1,
                3, 1, 3, 0, 0, 0, 2, 1, 0, 1, 1, 0, 2, 2, 1, 0, 1, 0, 3, 3, 3, 2,
                2, 1, 1, 2, 2, 2, 1, 3, 1, 2, 3, 3, 1, 3, 0, 0, 1, 1, 1, 0, 0, 3,
                1, 2, 1, 0, 0, 1, 2, 2, 1, 2, 3, 1, 1, 2, 2, 2, 0, 1, 2, 1, 1, 0,
                3, 1, 0, 3, 2, 1, 2, 1, 3, 2, 2, 3, 1, 2, 0, 1, 3, 2, 3, 1, 0, 1,
                3, 3, 3, 1, 1, 1, 0, 1, 0, 3, 2, 0, 3, 0, 0, 0, 2, 0, 0, 2, 3, 2,
                0, 0, 1, 0, 0, 2, 2, 1, 0, 3, 1, 1, 1, 2, 0, 3, 1, 3, 3, 2, 0, 0,
                3, 1, 2, 1, 3, 3, 2, 0, 1, 1, 3, 1, 2, 0, 1, 1, 3, 3, 0, 0, 3, 0,
                3, 0, 2, 2, 2, 3, 1, 2, 2, 1, 1, 2, 1, 0, 1, 1, 2, 2, 2, 1, 0, 2,
                2, 3, 0, 2, 1, 3, 1, 0, 2, 1, 3, 0, 0, 3, 0, 0, 1, 0, 1, 1, 2, 0,
                3, 2, 1, 0, 3, 2, 3, 1, 1, 0, 1, 1, 1, 3, 1, 2, 0, 3, 3, 3, 1, 2,
                3, 2, 2, 1, 3, 3, 3, 1, 1, 1, 1, 3, 2, 3, 1, 1, 2, 3, 3, 2, 3, 2,
                2, 1, 3, 2, 2, 1, 1, 1, 0, 1, 2, 2, 3, 1, 2, 1, 2, 1, 2, 1, 3, 1,
                3, 2, 2, 3, 1, 3, 1, 3, 1, 0, 1, 2, 2, 2, 3, 0, 0, 2, 3, 0, 3, 1,
                2, 2, 3, 1, 1, 2, 0, 3, 2, 2, 0, 3, 1, 0, 0, 1, 1, 0, 1, 2, 0, 3,
                2, 0, 2, 2, 0, 1, 3, 3, 3, 1, 2, 2, 1, 1, 2, 0, 3, 1, 2, 1, 1, 1,
                2, 0, 3, 2, 2, 0, 0, 1, 3, 2, 3, 3, 1, 3, 2, 2, 1, 0, 3, 0, 0, 1,
                3, 3, 2, 2, 0, 1, 0, 2, 1, 1, 0, 1, 2, 1, 1, 3, 1, 3, 3, 1, 1, 1,
                3, 3, 0, 0, 3, 3, 2])
```

## 10. Generating Classification Report with F1 Score

```
In [44]: import keras.backend as K
def accuracy(y_true, y_pred):
    '''Calculates the mean accuracy rate across all predictions for binary
    classification problems.
    '''
    return K.mean(K.equal(y_true, K.round(y_pred)))
```

```
In [45]: #Classification report with Accuracy (F1 Score) for each Class
print("CNN Disaster Classification Model Accuracy on test set: {:.4f}".format(accuracy(y_test, y_pred)))
print(classification_report(y_test, y_pred))
```

CNN Disaster Classification Model Accuracy on test set: 0.8881

	precision	recall	f1-score	support
0	0.94	0.88	0.91	94
1	0.86	0.88	0.87	136
2	0.82	0.90	0.85	108
3	0.97	0.89	0.93	109
accuracy			0.89	447
macro avg	0.90	0.89	0.89	447
weighted avg	0.89	0.89	0.89	447

## 11. Weighted Accuracy of the model

```
In [50]: #Weighted Accuracy of the model
accu = np.count_nonzero(np.equal(y_pred,y_test))/x_test.shape[0]
print("Accuracy: {} %".format(accu*100))

Accuracy: 88.81431767337807 %
```

## 12. Confusion Matrix for test data

```
In [50]: #Weighted Accuracy of the model
accu = np.count_nonzero(np.equal(y_pred,y_test))/x_test.shape[0]
print("Accuracy: {} %".format(accu*100))

Accuracy: 88.81431767337807 %
```

```
In [51]: classes = list(x_train.class_indices.keys())
```

```
In [53]: #Confusion matrix for test data Classification
import pandas as pd
df_cmatrix = pd.DataFrame(confusion_matrix(y_test, y_pred),index=classes, columns=classes)
sns.set(font_scale=1.0)
fig,ax = plt.subplots(figsize=(16,12))
sns.heatmap(df_cmatrix, annot=True,annot_kws={"size": 15},fmt='2g')
```

