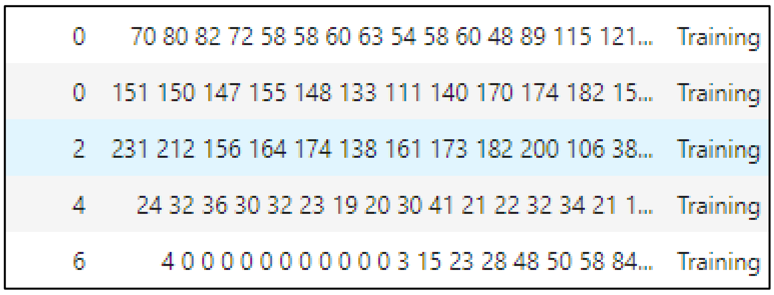
Name: - Prekshita Vasudeo Patil
Registration Number: - 20MAI0073

Assignment -1
Assignment Name: Deep learning

Dataset

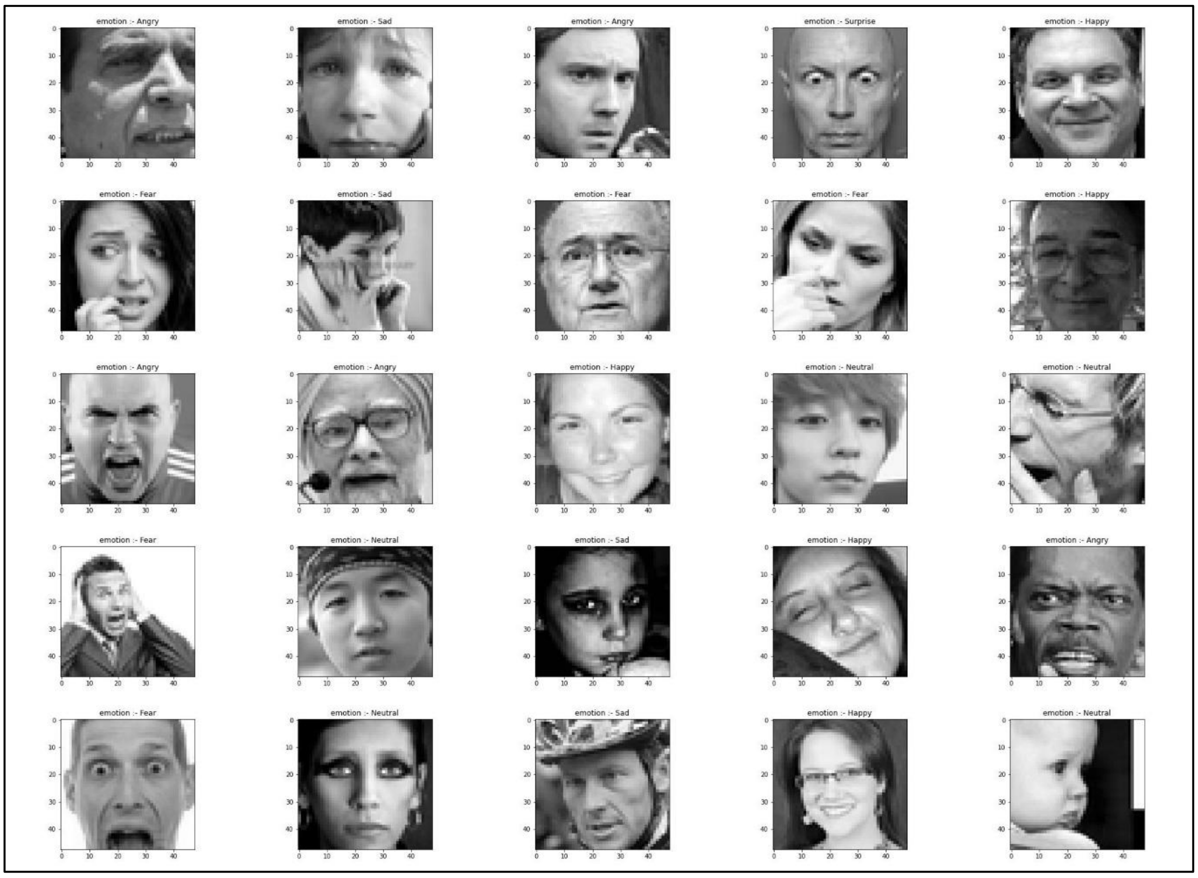
1) The data consists of 48x48 pixel grayscale images of faces. The faces have been automatically registered so that the face is more or less centred and occupies about the same amount of space in each image.
2) The task is to categorize each face based on the emotion shown in the facial expression into one of seven categories (0=Angry, 1=Disgust, 2=Fear, 3=Happy, 4=Sad, 5=Surprise, 6=Neutral). Fer2013.csv contains two columns, "emotion" and "pixels".
3) The "emotion" column contains a numeric code ranging from 0 to 6, inclusive, for the emotion that is present in the image.
4) The "pixels" column contains a string surrounded in quotes for each image. The contents of this string a space-separated pixel values in row major order. The dataset consists of 28,709 examples.

Description of Data



0	70 80 82 72 58 58 60 63 54 58 60 48 89 115 121...	Training
0	151 150 147 155 148 133 111 140 170 174 182 15...	Training
2	231 212 156 164 174 138 161 173 182 200 106 38...	Training
4	24 32 36 30 32 23 19 20 30 41 21 22 32 34 21 1...	Training
6	4 0 0 0 0 0 0 0 0 0 0 0 0 3 15 23 28 48 50 58 84...	Training

- There are 3 columns emotion, pixels and usage
- Where there are 6 emotions which are Angry, Disgust, Fear, Happy, Sad, Surprise, Neutral.
- Figures are in numeric values.
- The data will be classified into 3 categories which are Training, Public-Test, Private-Test
    - Hence, the data will be filtered in accordingly.
    - Training will be used for training the data.
    - Testing will be used for testing the data.
- The images which are presented in data.



Explanation and what is done

```python
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
import keras
from keras.models import Sequential
from keras.layers import Dense, Dropout, Activation, Flatten
from keras.layers import Conv2D, MaxPooling2D, BatchNormalization,AveragePooling2D
from keras.losses import categorical_crossentropy
from keras.optimizers import Adam
from keras.regularizers import l2
from keras.utils import np_utils
import tensorflow as tf
```

Importing Libraries such as pandas, numpy, matplotlib, seaborn , Tensorflow and keras which will be useful.

```python
read=pd.read_csv('fer2013.csv')
```

Reading Data present in the file . The file is a csv file (comma separated value) and storing it

----

```python
read.head()
```

This will help us to display first 5 values in the data and the output for the following will be:-

```
   emotion                                            pixels     Usage
0        0  70 80 82 72 58 58 60 63 54 58 60 48 89 115 121...  Training
1        0  151 150 147 155 148 133 111 140 170 174 182 15...  Training
2        2  231 212 156 164 174 138 161 173 182 200 106 38...  Training
3        4  24 32 36 30 32 23 19 20 30 41 21 22 32 34 21 1...  Training
4        6  4 0 0 0 0 0 0 0 0 0 0 0 0 3 15 23 28 48 50 58 84...  Training
```

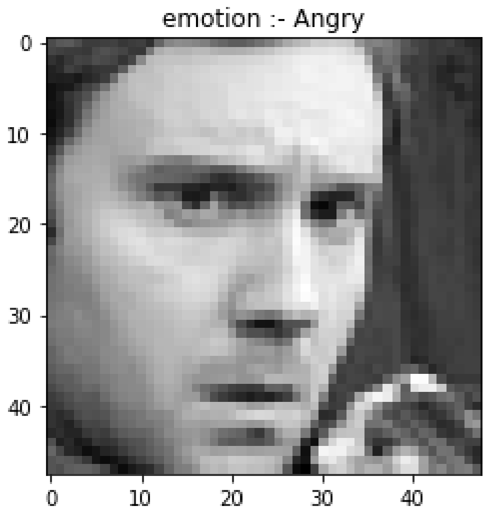
```
pixels = read['pixels']
```

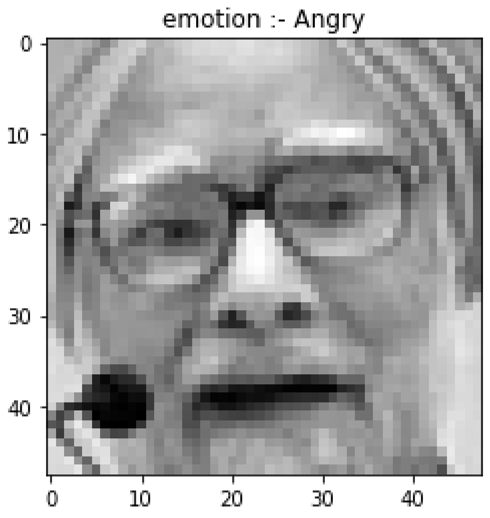
```
X = np.zeros((pixels.shape[0], 48*48))
for ix in range(X.shape[0]):
    p = pixels[ix].split(' ')
    for iy in range(X.shape[1]):
        X[ix, iy] = int(p[iy])
x = X
```

```
emotions = {0:'Angry', 1:'Disgust', 2:'Fear', 3:'Happy', 4:'Sad', 5:'Surprise', 6:'Neutral'}
```

```
for ix in range(25):
    plt.figure(ix)
    plt.imshow(x[ix].reshape((48, 48)), interpolation='none', cmap='gray')
    zz=emotions[read['emotion'][ix]]
    plt.title('emotion :- '+zz)
#     plt.savefig(str(ix)+'.jpg')
    plt.show()
```

Using pixels from data we are using it to visualize the data with the help of imshow from matplotlib and getting the emotions stored in dictionary(which was created by us) the images that are created are like:-





It also has the label that describes the type of emotions which are presented as title in the figure.

```
for index, row in read.iterrows():
    val=row['pixels'].split(" ")
    if 'Training' in row['Usage']:
        xtrain.append(np.array(val,'float32'))
        ytrain.append(row['emotion'])
    elif 'PublicTest' in row['Usage']:
        xtest.append(np.array(val,'float32'))
        ytest.append(row['emotion'])
```

Classifying the data into training and testing with the help of the dataframe that was created.

```
num_features = 64
num_labels = 7
batch_size = 64
epochs = 15
width, height = 48, 48
le=1e-3
```

Specifying the properties that are needed for model.

```
xtrain = np.array(xtrain,'float32')
ytrain = np.array(ytrain,'float32')
xtest = np.array(xtest,'float32')
ytest = np.array(ytest,'float32')
```

Converting the type into float32.

```
ytrain=np_utils.to_categorical(ytrain, num_classes=num_labels)
ytest=np_utils.to_categorical(ytest, num_classes=num_labels)
```

Converting into categories which will be form the data like [1,0,0,0,0,0] something like that where one represents the present emotions in the data set

----

```
xtrain -= np.mean(xtrain, axis=0)
xtrain /= np.std(xtrain, axis=0)
xtest -= np.mean(xtest, axis=0)
xtest /= np.std(xtest, axis=0)
xtrain = xtrain.reshape(xtrain.shape[0], 48, 48, 1)
xtest = xtest.reshape(xtest.shape[0], 48, 48, 1)
```

Transforming the data so that it should not be in skewed manner with the help of mean and standard deviation and reshaping it in 48 x 48 so the data becomes like

```
array([[[[-0.6098866 ],
         [-0.4592209 ],
         [-0.40325198],
         ...,
         [-0.7694696 ],
         [-0.90518403],
         [-0.95160526]],

        [[-0.66049284],
         [-0.68162924],
         [-0.694159  ],
         ...,
         [-0.7112763 ],
         [-0.7862608 ],
         [-0.90819967]],
```

```
model = Sequential()
```

Using Sequential as the model

First layer

```
model.add(Conv2D(64,(5,5),input_shape=(xtrain.shape[1:])))
model.add(Conv2D(64,(5, 5)))
model.add(BatchNormalization())
model.add(MaxPooling2D(pool_size=(2,2)))
model.add(Activation("relu"))
```

Second layer

```
model.add(Conv2D(128,(5, 5)))
model.add(Conv2D(128,(5, 5)))
model.add(BatchNormalization())
model.add(MaxPooling2D(pool_size=(2,2)))
model.add(Activation("relu"))
```

Third layer

```
model.add(Conv2D(256,(3, 3)))
model.add(Conv2D(256,(3, 3)))
model.add(BatchNormalization())
model.add(MaxPooling2D(pool_size=(2,2)))
model.add(Activation("relu"))
```

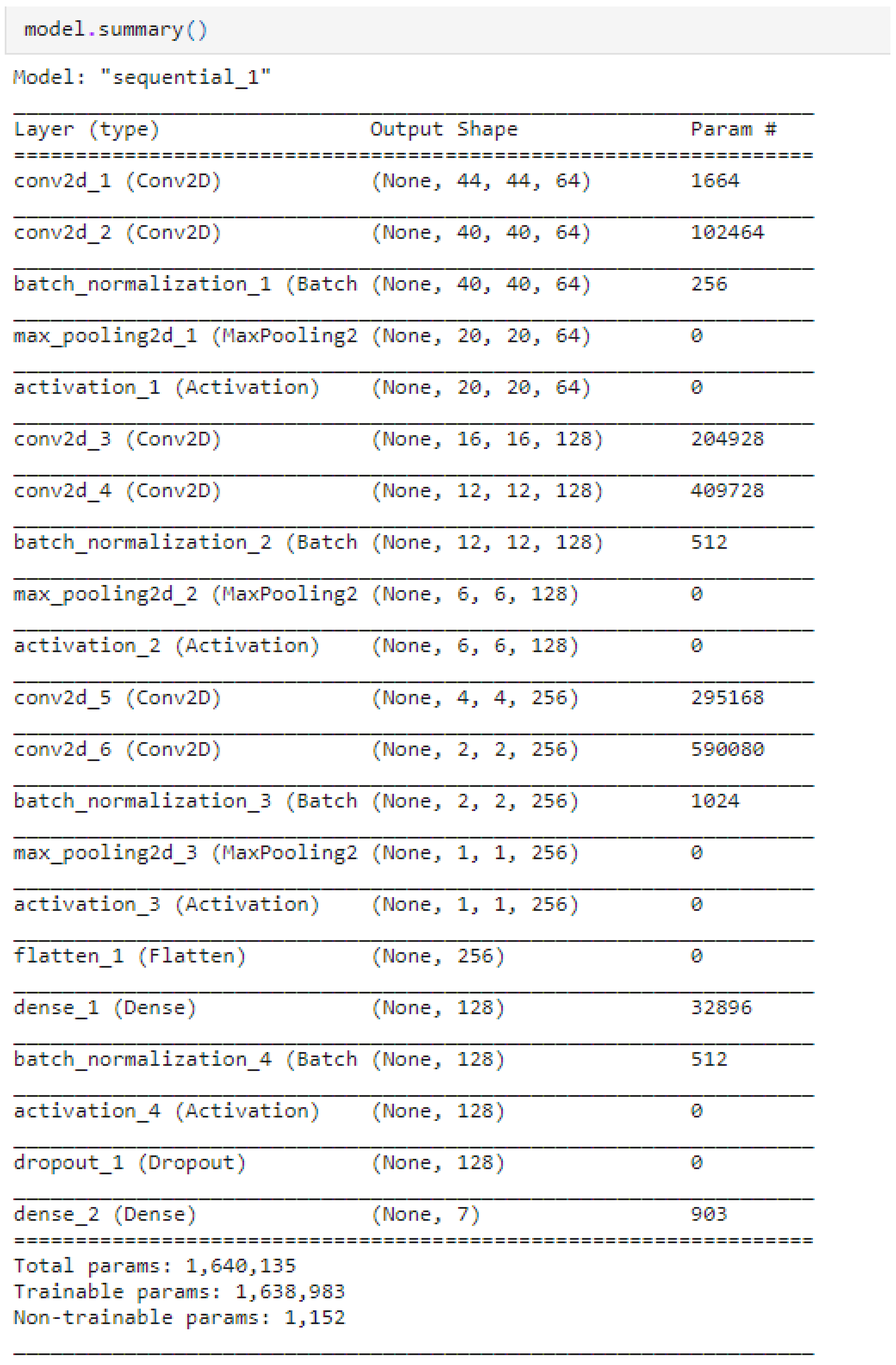
Fourth layer

```
model.add(Flatten())
model.add(Dense(128))
model.add(BatchNormalization())
model.add(Activation("relu"))
model.add(Dropout(0.25))
model.add(Dense(7, activation='softmax'))
```

Then we added layers such as Convolutional 2d having 64 neurons initially and stride as 5*5 and giving the input shape for the model later to make model more accurate we add layers such as batch-normalization and maxpooling2d which will take the maximum number in 2*2 matrix and will do until the end. The neurons gradually increased till third layer they almost got double in the next stage.

Finally, in the fourth layer we added flatten layer as the dense layer doesn't accept high dimensional arrays.

Lastly, we have added 7 neurons in dense this states that we have 7 outputs or we expected the outcome in those 7 emotions.



```python
model.summary()
```

```
Model: "sequential_1"
_________________________________________________________________
Layer (type)                 Output Shape              Param #   
=================================================================
conv2d_1 (Conv2D)            (None, 44, 44, 64)        1664      
_________________________________________________________________
conv2d_2 (Conv2D)            (None, 40, 40, 64)        102464    
_________________________________________________________________
batch_normalization_1 (Batch (None, 40, 40, 64)        256       
_________________________________________________________________
max_pooling2d_1 (MaxPooling2 (None, 20, 20, 64)        0         
_________________________________________________________________
activation_1 (Activation)    (None, 20, 20, 64)        0         
_________________________________________________________________
conv2d_3 (Conv2D)            (None, 16, 16, 128)       204928    
_________________________________________________________________
conv2d_4 (Conv2D)            (None, 12, 12, 128)       409728    
_________________________________________________________________
batch_normalization_2 (Batch (None, 12, 12, 128)       512       
_________________________________________________________________
max_pooling2d_2 (MaxPooling2 (None, 6, 6, 128)         0         
_________________________________________________________________
activation_2 (Activation)    (None, 6, 6, 128)         0         
_________________________________________________________________
conv2d_5 (Conv2D)            (None, 4, 4, 256)         295168    
_________________________________________________________________
conv2d_6 (Conv2D)            (None, 2, 2, 256)         590080    
_________________________________________________________________
batch_normalization_3 (Batch (None, 2, 2, 256)         1024      
_________________________________________________________________
max_pooling2d_3 (MaxPooling2 (None, 1, 1, 256)         0         
_________________________________________________________________
activation_3 (Activation)    (None, 1, 1, 256)         0         
_________________________________________________________________
flatten_1 (Flatten)          (None, 256)               0         
_________________________________________________________________
dense_1 (Dense)              (None, 128)               32896     
_________________________________________________________________
batch_normalization_4 (Batch (None, 128)               512       
_________________________________________________________________
activation_4 (Activation)    (None, 128)               0         
_________________________________________________________________
dropout_1 (Dropout)          (None, 128)               0         
_________________________________________________________________
dense_2 (Dense)              (None, 7)                 903       
=================================================================
Total params: 1,640,135
Trainable params: 1,638,983
Non-trainable params: 1,152
_________________________________________________________________
```

We printed the summary of the model which describes each and every layer and its execution.

So we can say at a glimpse that Conv2d will frame 44*44*64 and the output will be in 7 categorical values.

```
adam_optimizer=keras.optimizers.Adam(learning_rate=le)
```

Declaring properties of adam optimizer that will be used during compiling the model with specific properties.

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

The model is compiled with categorical_crossentropy as we are classifying different types of categories, the optimizer adam as we stated and the metrics is accuracy what we needed for the model.

```
model.fit(xtrain,ytrain,batch_size=batch_size,epochs=epochs)
```

```
Epoch 1/15
28709/28709 [==============================] - 375s 13ms/step - loss: 1.7088 - accuracy: 0.3289
Epoch 2/15
28709/28709 [==============================] - 520s 18ms/step - loss: 1.3978 - accuracy: 0.4594
Epoch 3/15
28709/28709 [==============================] - 417s 15ms/step - loss: 1.2690 - accuracy: 0.5151
Epoch 4/15
28709/28709 [==============================] - 371s 13ms/step - loss: 1.1944 - accuracy: 0.5462
Epoch 5/15
28709/28709 [==============================] - 373s 13ms/step - loss: 1.1359 - accuracy: 0.5702
Epoch 6/15
28709/28709 [==============================] - 374s 13ms/step - loss: 1.0910 - accuracy: 0.5874
Epoch 7/15
28709/28709 [==============================] - 374s 13ms/step - loss: 1.0435 - accuracy: 0.6043
Epoch 8/15
28709/28709 [==============================] - 373s 13ms/step - loss: 0.9954 - accuracy: 0.6247
Epoch 9/15
28709/28709 [==============================] - 374s 13ms/step - loss: 0.9475 - accuracy: 0.6444
Epoch 10/15
28709/28709 [==============================] - 374s 13ms/step - loss: 0.8895 - accuracy: 0.6658
Epoch 11/15
28709/28709 [==============================] - 371s 13ms/step - loss: 0.8340 - accuracy: 0.6867
Epoch 12/15
28709/28709 [==============================] - 375s 13ms/step - loss: 0.7748 - accuracy: 0.7093
Epoch 13/15
28709/28709 [==============================] - 372s 13ms/step - loss: 0.7128 - accuracy: 0.7348
Epoch 14/15
28709/28709 [==============================] - 373s 13ms/step - loss: 0.6455 - accuracy: 0.7620
Epoch 15/15
28709/28709 [==============================] - 377s 13ms/step - loss: 0.5785 - accuracy: 0.7857
<keras.callbacks.callbacks.History at 0x1ff87786608>
```

Then we fitted the model by specifying the input as xtrain,ytrain,batch-size and epochs where batch-size is that how much inputs will be considered in a batch which can be anything and epochs we have specified 15 which was declared ahead . Epoch stands for the number of time program will run and improve itself.

```
from sklearn.metrics import confusion_matrix
from sklearn.metrics import classification_report
```

Importing confusion matrix and classification report.

```
y_predict=model.predict_classes(xtest)
```

Predicting the model as we want something which can help to calculate confusion matrix and classification report.

```
confusion_matrix(y_predict,ytest)
```

Printing confusion matrix and saying which are true positive and true negative. There will be 7 X 7 matrix as there are 7 categories.

```
array([[263,  20,  77,  47, 137,  18,  79],
       [ 14,  24,   4,   3,   7,   1,   4],
       [ 70,   8, 241,  39, 129,  62,  77],
       [ 25,   0,  26, 692,  44,  24,  57],
       [ 47,   4,  61,  24, 211,   7,  81],
       [  9,   0,  31,  19,  12, 294,  12],
       [ 39,   0,  56,  71, 113,   9, 297]], dtype=int64)
```

So, the output that we got using confusion matrix is stated above.

```
classification_report(y_predict,ytest)
```

We used classification report because it will detail output of each and every output also it will be calculating different types of score such as f1 score, precision, recall, support and accuracy

So, the output what we got is.

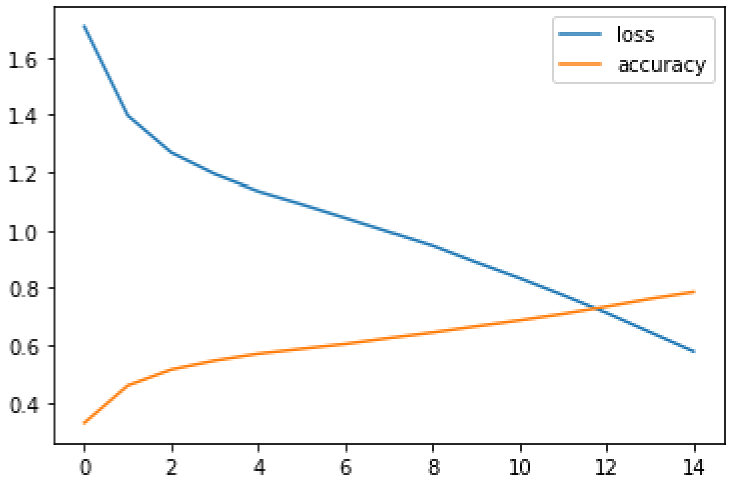
```
              precision    recall  f1-score   support

           0       0.56      0.41      0.47       641
           1       0.43      0.42      0.42        57
           2       0.49      0.38      0.43       626
           3       0.77      0.80      0.79       868
           4       0.32      0.49      0.39       435
           5       0.71      0.78      0.74       377
           6       0.49      0.51      0.50       585

    accuracy                           0.56      3589
   macro avg       0.54      0.54      0.53      3589
weighted avg       0.57      0.56      0.56      3589
```

```
metrics=pd.DataFrame(model.history.history)
```

Storing the result that we obtained in each and every epoch

```
metrics.plot()
```

```
<AxesSubplot:>
```

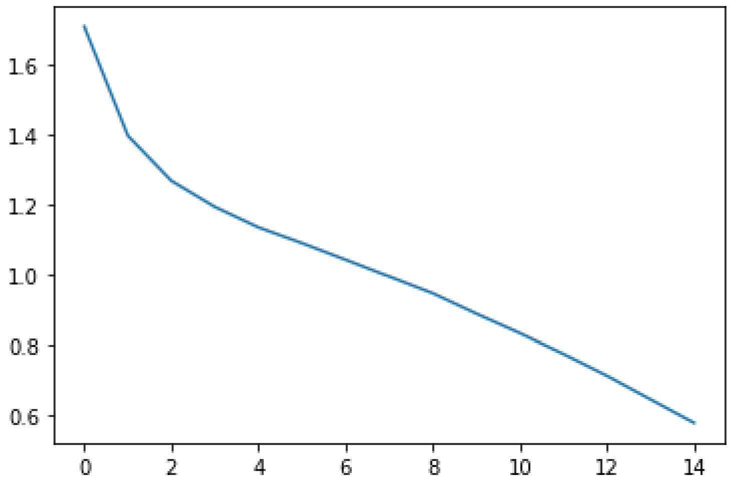


What result we obtained is represented in the line graph for the number of epochs

----

```
metrics.loss.plot()
```

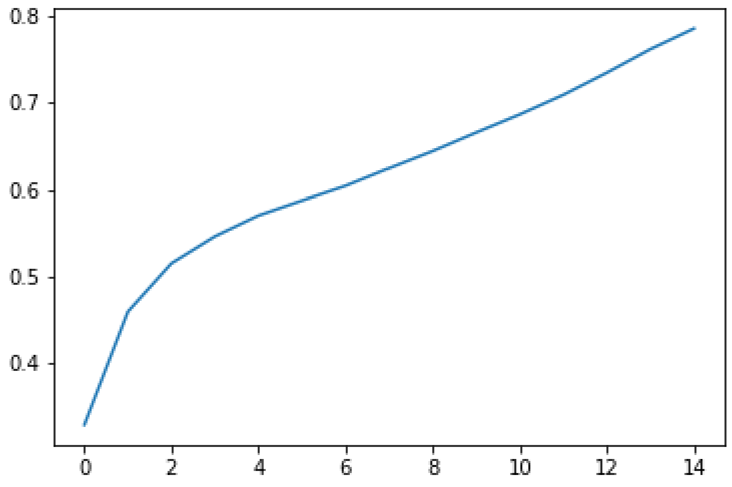
```
<AxesSubplot:>
```



Plotting loss alone. We can observe that loss decreased as the number of epoch increased.

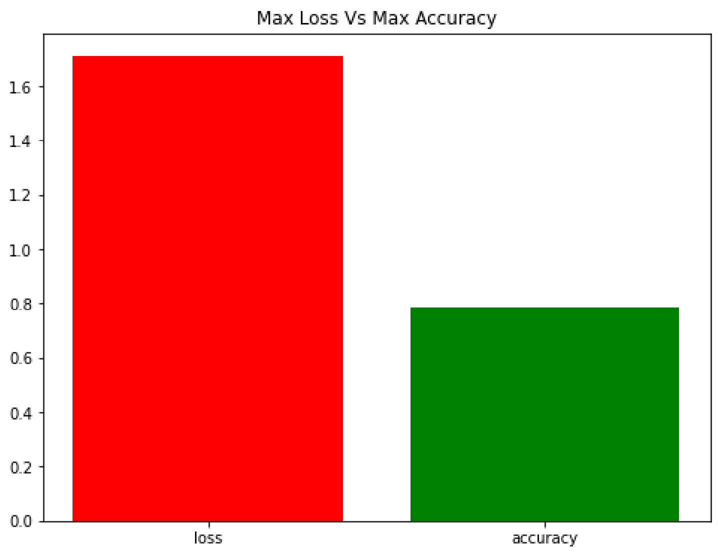
```
metrics.accuracy.plot()
```

```
<AxesSubplot:>
```



Plotting accuracy alone. So we can see accuracy increased as the number of epoch increased.

```
plt.figure(figsize=(8,6))
plt.bar(x=['loss','accuracy'],height=[max(metrics.loss),max(metrics.accuracy)],color=['red','green'])
plt.title('Max Loss Vs Max Accuracy')
plt.show()
```



Plotting the max loss and max accuracy that we obtained.

Code

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
import keras
from keras.models import Sequential
from keras.layers import Dense, Dropout, Activation, Flatten
from keras.layers import Conv2D, MaxPooling2D,
BatchNormalization,AveragePooling2D
from keras.losses import categorical_crossentropy
from keras.optimizers import Adam
from keras.regularizers import l2
from keras.utils import np_utils
import tensorflow as tf
tf.__version__
read=pd.read_csv('fer2013.csv')
read.head()
pixels = read['pixels']
X = np.zeros((pixels.shape[0], 48*48))
for ix in range(X.shape[0]):
    p = pixels[ix].split(' ')
    for iy in range(X.shape[1]):
        X[ix, iy] = int(p[iy])
x = X
emotions = {0:'Angry', 1:'Disgust', 2:'Fear', 3:'Happy', 4:'Sad',
5:'Surprise', 6:'Neutral'}
for ix in range(25):
    plt.figure(ix)
    plt.imshow(x[ix].reshape((48, 48)), interpolation='none', cmap='gray')
    zz=emotions[read['emotion'][ix]]
    plt.title('emotion :- '+zz)
#     plt.savefig(str(ix)+'.jpg')
    plt.show()
read["emotion"].unique()
read["Usage"].unique()
xtrain,ytrain,xtest,ytest=[],[],[],[]
for index, row in read.iterrows():
    val=row['pixels'].split(" ")
    if 'Training' in row['Usage']:
        xtrain.append(np.array(val,'float32'))
        ytrain.append(row['emotion'])
    elif 'PublicTest' in row['Usage']:
        xtest.append(np.array(val,'float32'))
        ytest.append(row['emotion'])
len(xtrain)
len(xtest)
len(ytrain)
len(ytest)
num_features = 64
num_labels = 7
batch_size = 64
epochs = 15
width, height = 48, 48
```

```
le=1e-3
xtrain = np.array(xtrain,'float32')
ytrain = np.array(ytrain,'float32')
xtest = np.array(xtest,'float32')
ytest = np.array(ytest,'float32')
ytrain=np_utils.to_categorical(ytrain, num_classes=num_labels)
ytest=np_utils.to_categorical(ytest, num_classes=num_labels)
xtrain
xtest
ytrain
ytest
xtrain -= np.mean(xtrain, axis=0)
xtrain /= np.std(xtrain, axis=0)
xtest -= np.mean(xtest, axis=0)
xtest /= np.std(xtest, axis=0)
xtrain = xtrain.reshape(xtrain.shape[0], 48, 48, 1)
xtest = xtest.reshape(xtest.shape[0], 48, 48, 1)
xtrain
xtest
model = Sequential()
print(model)
model.add(Conv2D(64,(5,5),input_shape=(xtrain.shape[1:])))
model.add(Conv2D(64,(5, 5)))
model.add(BatchNormalization())
model.add(MaxPooling2D(pool_size=(2,2)))
model.add(Activation("relu"))
model.add(Conv2D(128,(5, 5)))
model.add(Conv2D(128,(5, 5)))
model.add(BatchNormalization())
model.add(MaxPooling2D(pool_size=(2,2)))
model.add(Activation("relu"))
model.add(Conv2D(256,(3, 3)))
model.add(Conv2D(256,(3, 3)))
model.add(BatchNormalization())
model.add(MaxPooling2D(pool_size=(2,2)))
model.add(Activation("relu"))
model.add(Flatten())
model.add(Dense(128))
model.add(BatchNormalization())
model.add(Activation("relu"))
model.add(Dropout(0.25))
model.add(Dense(7, activation='softmax'))
model.summary()
adam_optimizer=keras.optimizers.Adam(learning_rate=le)
model.compile(loss=categorical_crossentropy,optimizer=adam_optimizer,metric
s=['accuracy'])
model.fit(xtrain,ytrain,batch_size=batch_size,epochs=epochs)
from sklearn.metrics import confusion_matrix
from sklearn.metrics import classification_report
y_predict=model.predict_classes(xtest)
y_predict
ytest=np.argmax(ytest,axis=1)
cm=confusion_matrix(y_predict,ytest)
cm
cr = classification_report(y_predict,ytest)
print(cr)
metrics=pd.DataFrame(model.history.history)
```

```
metrics.plot()
metrics.loss.plot()
metrics.accuracy.plot()
plt.figure(figsize=(8,6))
plt.bar(x=['loss','accuracy'],height=[max(metrics.loss),max(metrics.accurac
y)],color=['red','green'])
plt.title('Max Loss Vs Max Accuracy')
plt.show()
```