In [1]:

```
import numpy as np
import pandas as pd
```

In [2]:

```
data=pd.read_csv('../input/fer2013/fer2013.csv')
```

In [3]:

```
width=48
height=48
DataPoints=data["pixels"].tolist()
```

In [4]:

```
print(len(DataPoints[2]))
print(len(DataPoints[0]))
```

```
8581
8287
```

In [5]:

```
X=[]
for i in DataPoints:
    x1 = [int(x) for x in i.split(' ')]
    x1=np.asarray(x1).reshape(width,height)
    X.append(x1.astype('float32'))
X = np.asarray(X)
print(X.shape)
X = np.expand_dims(X, -1)
print(X.shape)
```

```
(35887, 48, 48)
(35887, 48, 48, 1)
```

In [6]:

```
y = pd.get_dummies(data['emotion']).to_numpy()

print(y.shape)
```

```
(35887, 7)
```

In [7]:

```
np.save('fdataX', X)
np.save('flabels', y)
# (0=Angry, 1=Disgust, 2=Fear, 3=Happy, 4=Sad, 5=Surprise, 6=Neutral)
```

In [8]:

```
print("Preprocessing Done")
print("Number of Features: "+str(len(X[0])))
print("Number of Labels: "+ str(len(y[0])))
print("Number of samples in dataset:"+str(len(X)))
```

```
Preprocessing Done
Number of Features: 48
Number of Labels: 7
Number of samples in dataset:35887
```

In [ ]:

```

```

In [9]:

```
import sys,os
import pandas as pd
from sklearn.model_selection import train_test_split
from keras.models import Sequential
from keras.layers import Dense, Dropout, Activation, Flatten
from keras.layers import Conv2D, MaxPooling2D, BatchNormalization
from keras.losses import categorical_crossentropy
from keras.optimizers import Adam
from keras.regularizers import l2
import matplotlib.pyplot as plt
```

In [10]:

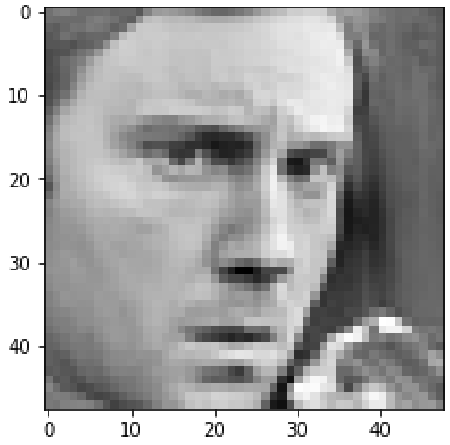
```
labels=7
epochs=100
batch_size=64
# standarisation of x
x=X
x=x-np.mean(x,axis=0)
x/=np.std(x,axis=0)
print(x)
# values in -1 to 1
print(x.shape)
print(X.shape)
```

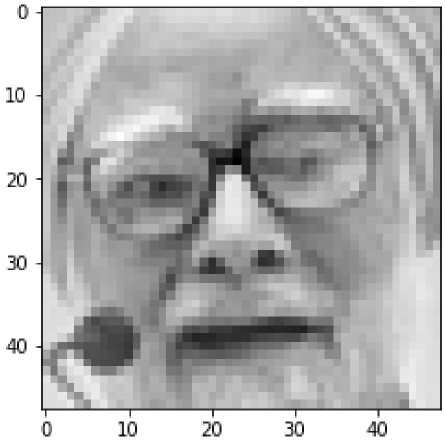
```
[[[[-0.60648495]
   [-0.4562006 ]
   [-0.40110716]
   ...
   [-0.76583207]
   [-0.901028  ]
   [-0.9487176 ]]

  [[-0.6577653 ]
   [-0.67946464]
   [-0.6925164 ]
   ...
   [-0.7071234 ]
   [-0.7820999 ]
   [-0.90432435]]

  [[-0.8304327 ]
   [-0.8934978 ]
   [-0.734429  ]
   ...
   [-0.7875835 ]
   [-0.72186905]
   [-0.85738075]]

  ...

  [[-0.33435795]
   [-0.6523918 ]
   [-0.937419  ]
   ...
   [-0.526588  ]
   [-0.7380636 ]
   [-0.9091638 ]]

  [[-0.51195467]
   [-0.43779588]
   [-0.46622527]
   ...
   [-0.10086355]
   [-0.5583586 ]
   [-0.870607  ]]

  [[-0.51134855]
   [-0.5642179 ]
   [-0.3995129 ]
   ...
   [-0.08714531]
   [-0.05788792]
   [-0.41197315]]]


 [[[ 0.37588486]
   [ 0.40796128]
   [ 0.41422534]
   ...
   [ 0.20136307]
   [ 0.29842654]
   [ 0.01435708]]

  [[ 0.39234453]
   [ 0.41382545]
   [ 0.457275  ]
   ...
   [ 0.12816645]
   [ 0.32547316]
   [ 0.23515376]]

  [[ 0.41038084]
   [ 0.45646253]
   [ 0.56443846]
   ...
   [-0.02222032]
   [ 0.11624809]
   [ 0.35951388]]

  ...

  [[ 0.8852279 ]
   [ 0.90315676]
   [ 0.0683608 ]
   ...
   [ 0.93312514]
   [ 0.91320026]
   [ 0.90503156]]

  [[ 0.8856582 ]
   [ 0.8920534 ]
   [ 1.0251336 ]
   ...
   [ 0.94489074]
   [ 0.8747153 ]
   [ 0.9183194 ]]

  [[ 0.86072004]
   [ 0.8548148 ]
   [ 0.8874503 ]
   ...
   [ 1.0353719 ]
   [ 0.889124  ]
   [ 0.8830088 ]]]


 [[[ 1.3461267 ]
   [ 1.1733618 ]
   [ 0.52711755]
   ...
   [-0.8663199 ]
   [-1.0988761 ]
   [-1.2534881 ]]

  [[ 1.3447697 ]
   [ 0.736843  ]
   [ 0.44463992]
   ...
   [-1.0741447 ]
   [-0.9936588 ]
   [-1.1126161 ]]
  [[ 1.1843537 ]
   [ 0.51896065]
   [ 0.57717246]
   ...
   [-1.0554606 ]
   [-1.1471822 ]
   [-1.0909262 ]]

  ...

  [[ 1.5515996 ]
   [ 1.6240207 ]
   [ 1.7107102 ]
   ...
   [-0.7203553 ]
   [-0.16204132]
   [ 0.39756432]]

  [[ 1.6159425 ]
   [ 1.6899631 ]
   [ 1.7389464 ]
   ...
   [-0.44944832]
   [-0.110523  ]
   [ 0.6011339 ]]

  [[ 1.6663384 ]
   [ 1.7037005 ]
   [ 1.715694  ]
   ...
   [-0.31939024]
   [-0.04509046]
   [ 0.47673994]]]


 ...


 [[[-1.2492701 ]
   [-1.2339463 ]
   [-1.2289833 ]
   ...
   [-0.37644184]
   [-0.02307674]
   [ 1.5382094 ]]

  [[-1.2316625 ]
   [-1.2261096 ]
   [-1.2231894 ]
   ...
   [-0.09963987]
   [ 0.26324993]
   [ 1.6564384 ]]

  [[-1.2112764 ]
   [-1.2309879 ]
   [-1.2055868 ]
   ...
   [ 0.22014467]
   [ 0.4790152 ]
   [ 1.699327  ]]

  ...

  [[-1.4282134 ]
   [-1.2088482 ]
   [-0.88649344]
   ...
   [ 0.94604295]
   [ 0.8491978 ]
   [ 0.91771823]]
  [[-1.4185144 ]
   [-1.2610359 ]
   [-0.95059824]
   ...
   [ 0.82869583]
   [ 0.7467623 ]
   [ 0.77875775]]

  [[-1.2288523 ]
   [-1.2864043 ]
   [-1.1895299 ]
   ...
   [ 0.5321745 ]
   [ 0.24925108]
   [-0.0184002 ]]]


 [[[-1.0916058 ]
   [-1.0981494 ]
   [-1.0784603 ]
   ...
   [-0.6653443 ]
   [-0.8144694 ]
   [-0.91214514]]

  [[-1.0851356 ]
   [-1.1018722 ]
   [-1.0715685 ]
   ...
   [-0.60587615]
   [-0.7820999 ]
   [-0.95333415]]

  [[-1.063853  ]
   [-1.0809923 ]
   [-1.0400449 ]
   ...
   [-0.6345109 ]
   [-0.74688745]
   [-0.98029935]]

  ...

  [[-0.17090829]
   [-0.09593542]
   [-0.07168449]
   ...
   [-1.0045472 ]
   [-1.0708765 ]
   [-1.0740906 ]]

  [[-0.19717698]
   [-0.14649557]
   [-0.09657223]
   ...
   [-1.0046018 ]
   [-1.0573754 ]
   [-1.0862932 ]]

  [[-0.30994394]
   [-0.26013947]
   [-0.19563752]
   ...
   [-1.0032226 ]
   [-1.0688871 ]
   [-1.0975518 ]]]


 [[[-1.2250141 ]
   [-1.283327  ]
   [-1.2540704 ]
```

```
   ...
   [-0.06241743]
   [-0.25802144]
   [-0.40013075]]

  [[-1.2560837 ]
   [-1.2261096 ]
   [-1.2358245 ]
   ...
   [-0.08698396]
   [-0.25942498]
   [-0.34071153]]

  [[-1.321844  ]
   [-1.3184854 ]
   [-1.2947248 ]
   ...
   [-0.12426874]
   [-0.25902823]
   [-0.2673712 ]]

  ...

  [[-1.2521906 ]
   [-1.2973753 ]
   [-1.2684351 ]
   ...
   [-0.7461909 ]
   [-0.63565964]
   [-0.24945639]]

  [[-1.2926033 ]
   [-1.2863663 ]
   [-1.3075047 ]
   ...
   [ 0.13152629]
   [ 0.733967  ]
   [ 0.9817565 ]]

  [[-1.2792034 ]
   [-1.2990742 ]
   [-1.3042097 ]
   ...
   [ 0.9837619 ]
   [ 1.0938833 ]
   [ 1.098839  ]]]]
(35887, 48, 48, 1)
(35887, 48, 48, 1)
```

In [11]:

```
for x1 in range(2):
   plt.figure(x1)
   plt.imshow(x[x1], interpolation='none', cmap='gray')
plt.show()
```





In [12]:

```
X_train, X_test, y_train, y_test = train_test_split(x, y, test_size=0.1)
np.save('X_test',X_test)
np.save('y_test',y_test)
```

CNN MODEL

conv2d since 2d kernel used in convolution

In [13]:

```
model=Sequential()
n=64
# no of neurons

model.add(Conv2D(n,kernel_size=(3,3),activation='relu',input_shape=(width, height, 1)))
model.add(Conv2D(n,kernel_size=(3,3),activation='relu',padding='same'))
# Same padding means the size of output feature-maps are the same as the input feature-ma
ps
model.add(BatchNormalization())
model.add(MaxPooling2D(pool_size=(2, 2), strides=(2, 2)))
model.add(Dropout(0.5))

model.add(Conv2D(2*n,kernel_size=(3,3),activation='relu',padding='same'))
model.add(BatchNormalization())
model.add(Conv2D(2*n,kernel_size=(3,3),activation='relu',padding='same'))
model.add(BatchNormalization())
model.add(MaxPooling2D(pool_size=(2, 2), strides=(2, 2)))
model.add(Dropout(0.5))

model.add(Conv2D(2*2*n,kernel_size=(3,3),activation='relu',padding='same'))
model.add(BatchNormalization())
model.add(Conv2D(2*2*n,kernel_size=(3,3),activation='relu',padding='same'))
model.add(BatchNormalization())
model.add(MaxPooling2D(pool_size=(2, 2), strides=(2, 2)))
model.add(Dropout(0.5))

model.add(Conv2D(2*2*2*n,kernel_size=(3,3),activation='relu',padding='same'))
model.add(BatchNormalization())
model.add(Conv2D(2*2*2*n,kernel_size=(3,3),activation='relu',padding='same'))
model.add(BatchNormalization())
model.add(MaxPooling2D(pool_size=(2, 2), strides=(2, 2)))
model.add(Dropout(0.5))


model.add(Flatten())

model.add(Dense(2*2*2*n, activation='relu'))
model.add(Dropout(0.4))

model.add(Dense(2*2*n, activation='relu'))
model.add(Dropout(0.4))
model.add(Dense(2*n, activation='relu'))
model.add(Dropout(0.4))

model.add(Dense(labels,activation='softmax'))

model.summary()

# There are mainly two types of non-trainable weights:

# The ones that you have chosen to keep constant when training. This means that keras won
't update these weights during training at all.
# The ones that work like statistics in BatchNormalization layers. They're updated with m
ean and variance, but they're not "trained with backpropagation".
```

```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #   
=================================================================
conv2d (Conv2D)              (None, 46, 46, 64)        640       
_________________________________________________________________
conv2d_1 (Conv2D)            (None, 46, 46, 64)        36928     
_________________________________________________________________
batch_normalization (BatchNo (None, 46, 46, 64)        256       
_________________________________________________________________
max_pooling2d (MaxPooling2D) (None, 23, 23, 64)        0         
_________________________________________________________________
dropout (Dropout)            (None, 23, 23, 64)        0         
_________________________________________________________________
conv2d_2 (Conv2D)            (None, 23, 23, 128)       73856     
_________________________________________________________________
batch_normalization_1 (Batch (None, 23, 23, 128)       512       
_________________________________________________________________
conv2d_3 (Conv2D)            (None, 23, 23, 128)       147584    
_________________________________________________________________
batch_normalization_2 (Batch (None, 23, 23, 128)       512       
_________________________________________________________________
max_pooling2d_1 (MaxPooling2 (None, 11, 11, 128)       0         
_________________________________________________________________
dropout_1 (Dropout)          (None, 11, 11, 128)       0         
_________________________________________________________________
conv2d_4 (Conv2D)            (None, 11, 11, 256)       295168    
_________________________________________________________________
batch_normalization_3 (Batch (None, 11, 11, 256)       1024      
_________________________________________________________________
conv2d_5 (Conv2D)            (None, 11, 11, 256)       590080    
_________________________________________________________________
batch_normalization_4 (Batch (None, 11, 11, 256)       1024      
_________________________________________________________________
max_pooling2d_2 (MaxPooling2 (None, 5, 5, 256)         0         
_________________________________________________________________
dropout_2 (Dropout)          (None, 5, 5, 256)         0         
_________________________________________________________________
conv2d_6 (Conv2D)            (None, 5, 5, 512)         1180160   
_________________________________________________________________
batch_normalization_5 (Batch (None, 5, 5, 512)         2048      
_________________________________________________________________
conv2d_7 (Conv2D)            (None, 5, 5, 512)         2359808   
_________________________________________________________________
batch_normalization_6 (Batch (None, 5, 5, 512)         2048      
_________________________________________________________________
max_pooling2d_3 (MaxPooling2 (None, 2, 2, 512)         0         
_________________________________________________________________
dropout_3 (Dropout)          (None, 2, 2, 512)         0         
_________________________________________________________________
flatten (Flatten)            (None, 2048)              0         
_________________________________________________________________
dense (Dense)                (None, 512)               1049088   
_________________________________________________________________
dropout_4 (Dropout)          (None, 512)               0         
_________________________________________________________________
dense_1 (Dense)              (None, 256)               131328    
```

```
_________________________________________________________________
dropout_5 (Dropout)          (None, 256)               0
_________________________________________________________________
dense_2 (Dense)              (None, 128)               32896
_________________________________________________________________
dropout_6 (Dropout)          (None, 128)               0
_________________________________________________________________
dense_3 (Dense)              (None, 7)                 903
=================================================================
Total params: 5,905,863
Trainable params: 5,902,151
Non-trainable params: 3,712
_________________________________________________________________
```

In [14]:

```
model.compile(loss=categorical_crossentropy,
              optimizer=Adam(lr=0.001, beta_1=0.9, beta_2=0.999, epsilon=1e-7),
              metrics=['accuracy'])
```

In [15]:

```
model.fit(np.array(X_train), np.array(y_train),
          batch_size=batch_size,
          epochs=100,
          verbose=1,
          shuffle=True)
```

```
Epoch 1/100
505/505 [==============================] - 17s 25ms/step - loss: 2.1634 - accuracy: 0.202
8
Epoch 2/100
505/505 [==============================] - 12s 24ms/step - loss: 1.7729 - accuracy: 0.262
2
Epoch 3/100
505/505 [==============================] - 12s 24ms/step - loss: 1.6302 - accuracy: 0.348
6
Epoch 4/100
505/505 [==============================] - 12s 24ms/step - loss: 1.5014 - accuracy: 0.409
6
Epoch 5/100
505/505 [==============================] - 12s 24ms/step - loss: 1.4120 - accuracy: 0.449
2
Epoch 6/100
505/505 [==============================] - 12s 24ms/step - loss: 1.3585 - accuracy: 0.481
0
Epoch 7/100
505/505 [==============================] - 12s 24ms/step - loss: 1.3073 - accuracy: 0.511
8
Epoch 8/100
505/505 [==============================] - 12s 24ms/step - loss: 1.2615 - accuracy: 0.526
3
Epoch 9/100
505/505 [==============================] - 12s 25ms/step - loss: 1.2411 - accuracy: 0.534
4
Epoch 10/100
505/505 [==============================] - 12s 24ms/step - loss: 1.2166 - accuracy: 0.550
8
Epoch 11/100
505/505 [==============================] - 12s 24ms/step - loss: 1.1789 - accuracy: 0.568
0
Epoch 12/100
505/505 [==============================] - 12s 24ms/step - loss: 1.1559 - accuracy: 0.573
5
Epoch 13/100
505/505 [==============================] - 12s 24ms/step - loss: 1.1348 - accuracy: 0.579
2
Epoch 14/100
505/505 [==============================] - 12s 24ms/step - loss: 1.0968 - accuracy: 0.598
8
Epoch 15/100
505/505 [==============================] - 12s 24ms/step - loss: 1.0701 - accuracy: 0.608
```

```
5
Epoch 16/100
505/505 [==============================] - 12s 24ms/step - loss: 1.0406 - accuracy: 0.619
2
Epoch 17/100
505/505 [==============================] - 12s 24ms/step - loss: 1.0397 - accuracy: 0.623
7
Epoch 18/100
505/505 [==============================] - 12s 24ms/step - loss: 1.0091 - accuracy: 0.632
7
Epoch 19/100
505/505 [==============================] - 12s 24ms/step - loss: 0.9697 - accuracy: 0.645
8
Epoch 20/100
505/505 [==============================] - 12s 24ms/step - loss: 0.9566 - accuracy: 0.652
6
Epoch 21/100
505/505 [==============================] - 12s 24ms/step - loss: 0.9341 - accuracy: 0.658
4
Epoch 22/100
505/505 [==============================] - 12s 24ms/step - loss: 0.9176 - accuracy: 0.667
6
Epoch 23/100
505/505 [==============================] - 12s 24ms/step - loss: 0.9104 - accuracy: 0.674
9
Epoch 24/100
505/505 [==============================] - 12s 24ms/step - loss: 0.8880 - accuracy: 0.673
9
Epoch 25/100
505/505 [==============================] - 12s 24ms/step - loss: 0.8568 - accuracy: 0.691
4
Epoch 26/100
505/505 [==============================] - 12s 24ms/step - loss: 0.8467 - accuracy: 0.693
4
Epoch 27/100
505/505 [==============================] - 12s 24ms/step - loss: 0.8145 - accuracy: 0.705
0
Epoch 28/100
505/505 [==============================] - 12s 24ms/step - loss: 0.7996 - accuracy: 0.712
9
Epoch 29/100
505/505 [==============================] - 12s 24ms/step - loss: 0.7874 - accuracy: 0.718
2
Epoch 30/100
505/505 [==============================] - 12s 24ms/step - loss: 0.7690 - accuracy: 0.721
0
Epoch 31/100
505/505 [==============================] - 12s 24ms/step - loss: 0.7617 - accuracy: 0.724
9
Epoch 32/100
505/505 [==============================] - 12s 24ms/step - loss: 0.7412 - accuracy: 0.738
0
Epoch 33/100
505/505 [==============================] - 12s 24ms/step - loss: 0.7190 - accuracy: 0.738
8
Epoch 34/100
505/505 [==============================] - 12s 24ms/step - loss: 0.7062 - accuracy: 0.751
6
Epoch 35/100
505/505 [==============================] - 12s 24ms/step - loss: 0.7074 - accuracy: 0.746
5
Epoch 36/100
505/505 [==============================] - 12s 24ms/step - loss: 0.6705 - accuracy: 0.764
0
Epoch 37/100
505/505 [==============================] - 12s 24ms/step - loss: 0.6503 - accuracy: 0.771
7
Epoch 38/100
505/505 [==============================] - 12s 24ms/step - loss: 0.6550 - accuracy: 0.769
6
Epoch 39/100
505/505 [==============================] - 12s 24ms/step - loss: 0.6435 - accuracy: 0.772
```

```
6
Epoch 40/100
505/505 [==============================] - 12s 24ms/step - loss: 0.6363 - accuracy: 0.776
9
Epoch 41/100
505/505 [==============================] - 12s 24ms/step - loss: 0.6178 - accuracy: 0.783
8
Epoch 42/100
505/505 [==============================] - 12s 24ms/step - loss: 0.6008 - accuracy: 0.789
6
Epoch 43/100
505/505 [==============================] - 12s 24ms/step - loss: 0.5841 - accuracy: 0.793
3
Epoch 44/100
505/505 [==============================] - 12s 24ms/step - loss: 0.5720 - accuracy: 0.795
2
Epoch 45/100
505/505 [==============================] - 12s 24ms/step - loss: 0.5696 - accuracy: 0.798
9
Epoch 46/100
505/505 [==============================] - 12s 24ms/step - loss: 0.5484 - accuracy: 0.808
9
Epoch 47/100
505/505 [==============================] - 12s 24ms/step - loss: 0.5494 - accuracy: 0.810
0
Epoch 48/100
505/505 [==============================] - 12s 24ms/step - loss: 0.5426 - accuracy: 0.813
1
Epoch 49/100
505/505 [==============================] - 12s 24ms/step - loss: 0.5392 - accuracy: 0.811
8
Epoch 50/100
505/505 [==============================] - 12s 24ms/step - loss: 0.5321 - accuracy: 0.813
5
Epoch 51/100
505/505 [==============================] - 12s 24ms/step - loss: 0.5021 - accuracy: 0.825
5
Epoch 52/100
505/505 [==============================] - 12s 24ms/step - loss: 0.5024 - accuracy: 0.827
5
Epoch 53/100
505/505 [==============================] - 12s 24ms/step - loss: 0.4906 - accuracy: 0.833
0
Epoch 54/100
505/505 [==============================] - 12s 24ms/step - loss: 0.4764 - accuracy: 0.838
3
Epoch 55/100
505/505 [==============================] - 12s 24ms/step - loss: 0.4667 - accuracy: 0.841
4
Epoch 56/100
505/505 [==============================] - 12s 24ms/step - loss: 0.4570 - accuracy: 0.846
8
Epoch 57/100
505/505 [==============================] - 12s 24ms/step - loss: 0.4560 - accuracy: 0.846
1
Epoch 58/100
505/505 [==============================] - 12s 24ms/step - loss: 0.4496 - accuracy: 0.846
0
Epoch 59/100
505/505 [==============================] - 12s 24ms/step - loss: 0.4462 - accuracy: 0.850
1
Epoch 60/100
505/505 [==============================] - 12s 24ms/step - loss: 0.4299 - accuracy: 0.854
6
Epoch 61/100
505/505 [==============================] - 12s 24ms/step - loss: 0.4096 - accuracy: 0.860
3
Epoch 62/100
505/505 [==============================] - 12s 24ms/step - loss: 0.4132 - accuracy: 0.859
2
Epoch 63/100
505/505 [==============================] - 12s 24ms/step - loss: 0.4116 - accuracy: 0.860
```

```
3
Epoch 64/100
505/505 [==============================] - 12s 24ms/step - loss: 0.3950 - accuracy: 0.866
5
Epoch 65/100
505/505 [==============================] - 12s 24ms/step - loss: 0.3878 - accuracy: 0.867
3
Epoch 66/100
505/505 [==============================] - 12s 24ms/step - loss: 0.3939 - accuracy: 0.868
2
Epoch 67/100
505/505 [==============================] - 12s 24ms/step - loss: 0.3805 - accuracy: 0.872
3
Epoch 68/100
505/505 [==============================] - 12s 24ms/step - loss: 0.3756 - accuracy: 0.874
0
Epoch 69/100
505/505 [==============================] - 12s 24ms/step - loss: 0.3620 - accuracy: 0.877
3
Epoch 70/100
505/505 [==============================] - 12s 24ms/step - loss: 0.3716 - accuracy: 0.876
4
Epoch 71/100
505/505 [==============================] - 12s 24ms/step - loss: 0.3758 - accuracy: 0.876
6
Epoch 72/100
505/505 [==============================] - 12s 24ms/step - loss: 0.3471 - accuracy: 0.885
5
Epoch 73/100
505/505 [==============================] - 12s 24ms/step - loss: 0.3544 - accuracy: 0.884
5
Epoch 74/100
505/505 [==============================] - 12s 24ms/step - loss: 0.3305 - accuracy: 0.888
9
Epoch 75/100
505/505 [==============================] - 12s 24ms/step - loss: 0.3492 - accuracy: 0.884
0
Epoch 76/100
505/505 [==============================] - 12s 24ms/step - loss: 0.3296 - accuracy: 0.889
2
Epoch 77/100
505/505 [==============================] - 12s 24ms/step - loss: 0.3278 - accuracy: 0.890
2
Epoch 78/100
505/505 [==============================] - 12s 24ms/step - loss: 0.3204 - accuracy: 0.894
5
Epoch 79/100
505/505 [==============================] - 12s 24ms/step - loss: 0.3154 - accuracy: 0.896
4
Epoch 80/100
505/505 [==============================] - 12s 24ms/step - loss: 0.3187 - accuracy: 0.897
3
Epoch 81/100
505/505 [==============================] - 12s 24ms/step - loss: 0.3081 - accuracy: 0.895
3
Epoch 82/100
505/505 [==============================] - 12s 24ms/step - loss: 0.3015 - accuracy: 0.902
0
Epoch 83/100
505/505 [==============================] - 12s 24ms/step - loss: 0.3133 - accuracy: 0.896
4
Epoch 84/100
505/505 [==============================] - 12s 24ms/step - loss: 0.3032 - accuracy: 0.900
0
Epoch 85/100
505/505 [==============================] - 12s 24ms/step - loss: 0.2890 - accuracy: 0.908
9
Epoch 86/100
505/505 [==============================] - 12s 24ms/step - loss: 0.2868 - accuracy: 0.905
3
Epoch 87/100
505/505 [==============================] - 12s 24ms/step - loss: 0.2933 - accuracy: 0.905
```

```
1
Epoch 88/100
505/505 [==============================] - 12s 24ms/step - loss: 0.3006 - accuracy: 0.901
7
Epoch 89/100
505/505 [==============================] - 12s 24ms/step - loss: 0.2760 - accuracy: 0.908
4
Epoch 90/100
505/505 [==============================] - 12s 24ms/step - loss: 0.2838 - accuracy: 0.907
3
Epoch 91/100
505/505 [==============================] - 12s 24ms/step - loss: 0.2870 - accuracy: 0.907
7
Epoch 92/100
505/505 [==============================] - 12s 24ms/step - loss: 0.2792 - accuracy: 0.911
1
Epoch 93/100
505/505 [==============================] - 12s 24ms/step - loss: 0.2523 - accuracy: 0.917
0
Epoch 94/100
505/505 [==============================] - 12s 24ms/step - loss: 0.2515 - accuracy: 0.915
5
Epoch 95/100
505/505 [==============================] - 12s 24ms/step - loss: 0.2564 - accuracy: 0.917
4
Epoch 96/100
505/505 [==============================] - 12s 24ms/step - loss: 0.2632 - accuracy: 0.916
3
Epoch 97/100
505/505 [==============================] - 12s 24ms/step - loss: 0.2525 - accuracy: 0.918
0
Epoch 98/100
505/505 [==============================] - 12s 24ms/step - loss: 0.2446 - accuracy: 0.919
8
Epoch 99/100
505/505 [==============================] - 12s 24ms/step - loss: 0.2474 - accuracy: 0.921
5
Epoch 100/100
505/505 [==============================] - 12s 24ms/step - loss: 0.2422 - accuracy: 0.922
0
```

Out[15]:

```
<tensorflow.python.keras.callbacks.History at 0x7f4efc064690>
```

100 epochs batch size:64

In [16]:

```
fer_json = model.to_json()
with open("fer.json", "w") as json_file:
    json_file.write(fer_json)
model.save_weights("fer.h5")
print("Saved model to disk")
```

```
Saved model to disk
```

In [17]:

```
y_predicted=model.predict(X_test)
print(y_predicted.shape)
```

```
(3589, 7)
```

In [18]:

```
import joblib
yh = y_predicted.tolist()
yt = y_test.tolist()
count = 0
predy=[]
```

```
truey=[]
for i in range(len(y_test)):
    yy = max(yh[i])
    yyt = max(yt[i])
    predy.append(yh[i].index(yy))
    truey.append(yt[i].index(yyt))
    if(yh[i].index(yy)== yt[i].index(yyt)):
        count+=1

acc = (count/len(y_test))*100
print(acc)
np.save('truey', truey)
np.save('predy', predy)

joblib.dump(model, "data_transformer.joblib")
model.save('model.h5')
```

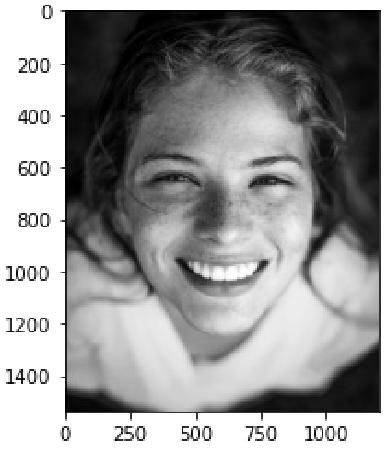
```
65.42212315408192
```

> **67% accurate on test data**

In [59]:

```
from PIL import Image
import cv2
from urllib.request import urlopen
from PIL import Image




img = cv2.imread("../input/test-happy/t2.jpg")

print(img.shape)
img = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
print(img.shape)
plt.imshow(img,cmap="gray")
plt.show()
```

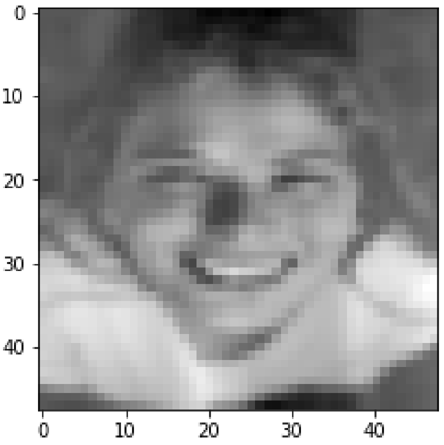
```
(1536, 1200, 3)
(1536, 1200)
```



In [60]:

```
from skimage.transform import resize
from skimage import data
image = np.array(img)
image=resize(image, (48, 48,1))
x12=image
x12=x12-np.mean(x12,axis=0)
x12/=np.std(x12,axis=0)
# print(x)
```

```
plt.imshow(x12,cmap="gray")
plt.show()
```



In [61]:

```
arr=model.predict(np.array([image]))

c=0
index=0
max=0.00
for x in arr[0]:
    if x > max:
        max=x
        index=c
    c+=1

print(index)
# (0=Angry, 1=Disgust, 2=Fear, 3=Happy, 4=Sad, 5=Surprise, 6=Neutral)
```

```
3
```

saving model for future usage

In [68]:

```
import os
os.chdir(r'/kaggle/working')
from IPython.display import FileLink
FileLink(r'model.h5')
```

Out[68]:

model.h5