AI model training for Happy and Sad Image recognition

Accuracy Improved 8/10

Data is consist of 5000 images for each happy and sad trained class, and average of 250 test images for each class

In [1]:
```
import pandas as pd
import numpy as np
from sklearn import datasets, linear_model
from sklearn.model_selection import train_test_split
from matplotlib import pyplot as plt
import numpy as np
import glob
import cv2
```

Converting images to array (Train Data Set)

In [2]:

```
#Preparing Training data

path_1 = glob.glob("../input/newdata/Newdata/train/happy/*.*")
cv1_img = []
for img in path_1:
    n =  cv2.imread(img)
    n = cv2.resize(n,(100,100))
    cv1_img.append(n)

array1 = np.asarray(cv1_img)
print(array1.shape)


path_2 = glob.glob("../input/newdata/Newdata/train/sad/*.*")
cv2_img = []
for img in path_2:
    n =  cv2.imread(img)
    n = cv2.resize(n,(100,100))
    cv2_img.append(n)

array2 = np.asarray(cv2_img)
print(array2.shape)
```

```
(5000, 100, 100, 3)
(5000, 100, 100, 3)
```

Converting images to array (Test Data Set)

In [3]:

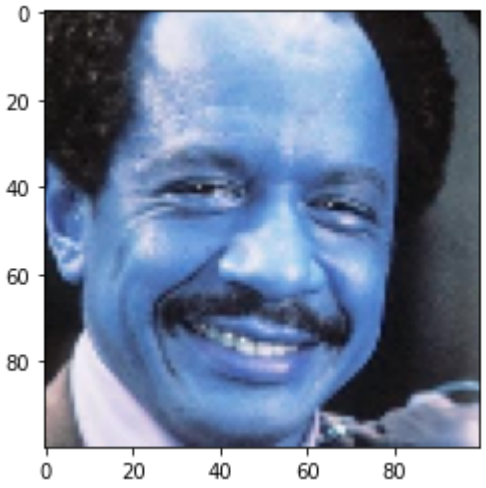
```
#Preparing Testing data


path_1 = glob.glob("../input/newdata/Newdata/test/happy/*.*")
cv3_img = []
for img in path_1:
    n =  cv2.imread(img)
    n = cv2.resize(n,(100,100))
    cv3_img.append(n)

array3 = np.asarray(cv3_img)
print(array3.shape)


path_2 = glob.glob("../input/newdata/Newdata/test/sad/*.*")
cv4_img = []
for img in path_2:
    n =  cv2.imread(img)
    n = cv2.resize(n,(100,100))
    cv4_img.append(n)

array4 = np.asarray(cv4_img)
print(array4.shape)
```

```
(248, 100, 100, 3)
(248, 100, 100, 3)
```

Previewing an image

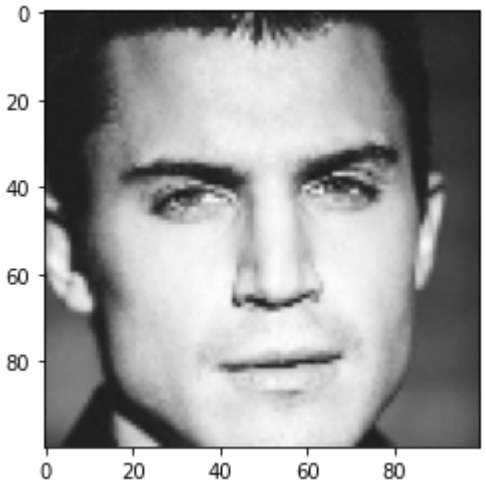
In [4]:
```
plt.imshow(array1[15])
```

Out[4]:
```
<matplotlib.image.AxesImage at 0x7fbe8a22d790>
```



In [5]:
```
plt.imshow(array2[17])
```

Out[5]:
```
<matplotlib.image.AxesImage at 0x7fbe781e6090>
```



In [6]:
```
xtr = np.concatenate((array1, array2))
xts = np.concatenate((array3, array4))

xtr.shape, xts.shape
```

Out[6]:
```
((10000, 100, 100, 3), (496, 100, 100, 3))
```

Setting Targets for Train dataset

In [7]:
```
H = np.zeros(len(array1))
S = np.ones(len(array2))

print('Happy labels =',H, 'Sad Labels =',S)
len(H),len(S)
```

```
Happy labels = [0. 0. 0. ... 0. 0. 0.] Sad Labels = [1. 1. 1. ... 1.
1. 1.]
```

Out[7]:
```
(5000, 5000)
```

Setting targets for test Dataset

In [8]:

```
HP = np.zeros(len(array3))
SD = np.ones(len(array4))

print('Happy Labels =', HP, 'Sad Labels =',SD)
len(HP),len(SD)
```

```
Happy Labels = [0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.
0. 0. 0. 0. 0. 0.
 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.
0.
 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.
0.
 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.
0.
 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.
0.
 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.
0.
 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.
0.
 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.
0.
 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.
0.
 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.
0.
 0. 0. 0. 0. 0. 0. 0. 0.] Sad Labels = [1. 1. 1. 1. 1. 1. 1. 1. 1. 1.
1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1.
 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1.
1.
 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1.
1.
 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1.
1.
 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1.
1.
 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1.
1.
 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1.
1.
 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1.
1.
 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1.
1.
 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1.
```

```
1.
 1. 1. 1. 1. 1. 1. 1. 1.]
```

Out[8]:

```
(248, 248)
```

Combining Data for Happy and Sad

In [9]:

```
ytr = np.concatenate((H,S))
yts = np.concatenate((HP,SD))

ytr.shape, yts.shape
```

Out[9]:

```
((10000,), (496,))
```

In [10]:

```
from sklearn.model_selection import train_test_split
xtr,xval,ytr,yval = train_test_split(xtr,ytr,test_size = 0.2, shuffle = T
rue)
xtr.shape,xval.shape,ytr.shape,yval.shape
```

Out[10]:

```
((8000, 100, 100, 3), (2000, 100, 100, 3), (8000,), (2000,))
```

Normalzing the data

In [11]:

```
xtrnorm = xtr/255
xvalnorm = xval/255
xtsnorm = xts/255
```

Building, Compiling and Training the AI Model

In [12]:

```
# Applying CNN model
from keras import layers , models
import tensorflow as tf
tf.random.set_seed(2)

model = models.Sequential()

model.add(layers.Conv2D(32,3,activation = 'relu', input_shape = (100,100,
3)))
model.add(layers.MaxPooling2D(3,3))
model.add(layers.Conv2D(64,3,activation = 'relu'))
model.add(layers.MaxPooling2D(3,3))
model.add(layers.Conv2D(90,3,activation = 'relu'))
model.add(layers.MaxPooling2D(3,3))
model.add(layers.Flatten())
model.add(layers.Dense(16,activation = 'relu'))
model.add(layers.Dense(1, activation = 'sigmoid'))
model.compile(optimizer = 'adam' , loss = 'binary_crossentropy', metrics
= ['acc'])
model_hist = model.fit(xtr,ytr,epochs = 5, validation_data = (xval,yval),
verbose = 1)
model.fit(xtrnorm,ytr,epochs = 20, validation_data = (xvalnorm, yval))
```

```
User settings:

   KMP_AFFINITY=granularity=fine,verbose,compact,1,0
   KMP_BLOCKTIME=0
   KMP_DUPLICATE_LIB_OK=True
   KMP_INIT_AT_FORK=FALSE
   KMP_SETTINGS=1
   KMP_WARNINGS=0

Effective settings:

   KMP_ABORT_DELAY=0
   KMP_ADAPTIVE_LOCK_PROPS='1,1024'
   KMP_ALIGN_ALLOC=64
   KMP_ALL_THREADPRIVATE=128
   KMP_ATOMIC_MODE=2
   KMP_BLOCKTIME=0
   KMP_CPUINFO_FILE: value is not defined
   KMP_DETERMINISTIC_REDUCTION=false
   KMP_DEVICE_THREAD_LIMIT=2147483647
   KMP_DISP_NUM_BUFFERS=7
   KMP_DUPLICATE_LIB_OK=true
   KMP_ENABLE_TASK_THROTTLING=true
   KMP_FORCE_REDUCTION: value is not defined
   KMP_FOREIGN_THREADS_THREADPRIVATE=true
   KMP_FORKJOIN_BARRIER='2,2'
   KMP_FORKJOIN_BARRIER_PATTERN='hyper,hyper'
   KMP_GTID_MODE=3
   KMP_HANDLE_SIGNALS=false
   KMP_HOT_TEAMS_MAX_LEVEL=1
   KMP_HOT_TEAMS_MODE=0
   KMP_INIT_AT_FORK=true
   KMP_LIBRARY=throughput
   KMP_LOCK_KIND=queuing
   KMP_MALLOC_POOL_INCR=1M
   KMP_NUM_LOCKS_IN_BLOCK=1
   KMP_PLAIN_BARRIER='2,2'
   KMP_PLAIN_BARRIER_PATTERN='hyper,hyper'
   KMP_REDUCTION_BARRIER='1,1'
   KMP_REDUCTION_BARRIER_PATTERN='hyper,hyper'
```

```
   KMP_SCHEDULE='static,balanced;guided,iterative'
   KMP_SETTINGS=true
   KMP_SPIN_BACKOFF_PARAMS='4096,100'
   KMP_STACKOFFSET=64
   KMP_STACKPAD=0
   KMP_STACKSIZE=8M
   KMP_STORAGE_MAP=false
   KMP_TASKING=2
   KMP_TASKLOOP_MIN_TASKS=0
   KMP_TASK_STEALING_CONSTRAINT=1
   KMP_TEAMS_THREAD_LIMIT=4
   KMP_TOPOLOGY_METHOD=all
   KMP_USE_YIELD=1
   KMP_VERSION=false
   KMP_WARNINGS=false
   OMP_AFFINITY_FORMAT='OMP: pid %P tid %i thread %n bound to OS proc
set {%A}'
   OMP_ALLOCATOR=omp_default_mem_alloc
   OMP_CANCELLATION=false
   OMP_DEFAULT_DEVICE=0
   OMP_DISPLAY_AFFINITY=false
   OMP_DISPLAY_ENV=false
   OMP_DYNAMIC=false
   OMP_MAX_ACTIVE_LEVELS=1
   OMP_MAX_TASK_PRIORITY=0
   OMP_NESTED: deprecated; max-active-levels-var=1
   OMP_NUM_THREADS: value is not defined
   OMP_PLACES: value is not defined
   OMP_PROC_BIND='intel'
   OMP_SCHEDULE='static'
   OMP_STACKSIZE=8M
   OMP_TARGET_OFFLOAD=DEFAULT
   OMP_THREAD_LIMIT=2147483647
   OMP_WAIT_POLICY=PASSIVE
   KMP_AFFINITY='verbose,warnings,respect,granularity=fine,compact,1,
0'

2022-01-14 18:24:02.930001: I tensorflow/core/common_runtime/process_u
til.cc:146] Creating new thread pool with default inter op setting: 2.
Tune using inter_op_parallelism_threads for best performance.
2022-01-14 18:24:03.897757: I tensorflow/compiler/mlir/mlir_graph_opti
```

```
mization_pass.cc:185] None of the MLIR Optimization Passes are enabled
(registered 2)
```

```
Epoch 1/5
250/250 [==============================] - 24s 89ms/step - loss: 1.019
5 - acc: 0.5616 - val_loss: 0.6390 - val_acc: 0.6555
Epoch 2/5
250/250 [==============================] - 22s 89ms/step - loss: 0.591
0 - acc: 0.6789 - val_loss: 0.5314 - val_acc: 0.7230
Epoch 3/5
250/250 [==============================] - 22s 88ms/step - loss: 0.476
0 - acc: 0.7751 - val_loss: 0.4443 - val_acc: 0.8030
Epoch 4/5
250/250 [==============================] - 21s 86ms/step - loss: 0.406
4 - acc: 0.8167 - val_loss: 0.4048 - val_acc: 0.8190
Epoch 5/5
250/250 [==============================] - 22s 87ms/step - loss: 0.353
9 - acc: 0.8428 - val_loss: 0.4008 - val_acc: 0.8300
Epoch 1/20
250/250 [==============================] - 23s 91ms/step - loss: 0.680
7 - acc: 0.5428 - val_loss: 0.6502 - val_acc: 0.6400
Epoch 2/20
250/250 [==============================] - 21s 85ms/step - loss: 0.588
3 - acc: 0.6952 - val_loss: 0.5083 - val_acc: 0.7640
Epoch 3/20
250/250 [==============================] - 22s 88ms/step - loss: 0.480
5 - acc: 0.7753 - val_loss: 0.4346 - val_acc: 0.8085
Epoch 4/20
250/250 [==============================] - 22s 87ms/step - loss: 0.422
1 - acc: 0.8080 - val_loss: 0.4259 - val_acc: 0.8020
Epoch 5/20
250/250 [==============================] - 21s 85ms/step - loss: 0.393
5 - acc: 0.8285 - val_loss: 0.3843 - val_acc: 0.8295
Epoch 6/20
250/250 [==============================] - 22s 88ms/step - loss: 0.367
0 - acc: 0.8421 - val_loss: 0.3773 - val_acc: 0.8375
Epoch 7/20
250/250 [==============================] - 22s 88ms/step - loss: 0.342
0 - acc: 0.8560 - val_loss: 0.3988 - val_acc: 0.8190
Epoch 8/20
250/250 [==============================] - 22s 89ms/step - loss: 0.324
2 - acc: 0.8670 - val_loss: 0.3579 - val_acc: 0.8435
Epoch 9/20
```

```
250/250 [==============================] - 21s 85ms/step - loss: 0.313
6 - acc: 0.8673 - val_loss: 0.3364 - val_acc: 0.8565
Epoch 10/20
250/250 [==============================] - 22s 89ms/step - loss: 0.290
8 - acc: 0.8801 - val_loss: 0.3259 - val_acc: 0.8675
Epoch 11/20
250/250 [==============================] - 21s 86ms/step - loss: 0.272
6 - acc: 0.8869 - val_loss: 0.3262 - val_acc: 0.8605
Epoch 12/20
250/250 [==============================] - 22s 88ms/step - loss: 0.255
0 - acc: 0.8964 - val_loss: 0.3342 - val_acc: 0.8625
Epoch 13/20
250/250 [==============================] - 22s 88ms/step - loss: 0.242
8 - acc: 0.9010 - val_loss: 0.3292 - val_acc: 0.8635
Epoch 14/20
250/250 [==============================] - 21s 85ms/step - loss: 0.223
2 - acc: 0.9095 - val_loss: 0.3217 - val_acc: 0.8735
Epoch 15/20
250/250 [==============================] - 22s 87ms/step - loss: 0.212
9 - acc: 0.9154 - val_loss: 0.3259 - val_acc: 0.8730
Epoch 16/20
250/250 [==============================] - 22s 88ms/step - loss: 0.198
5 - acc: 0.9220 - val_loss: 0.4022 - val_acc: 0.8395
Epoch 17/20
250/250 [==============================] - 22s 88ms/step - loss: 0.184
4 - acc: 0.9298 - val_loss: 0.3285 - val_acc: 0.8705
Epoch 18/20
250/250 [==============================] - 21s 85ms/step - loss: 0.164
3 - acc: 0.9371 - val_loss: 0.3413 - val_acc: 0.8690
Epoch 19/20
250/250 [==============================] - 22s 88ms/step - loss: 0.146
6 - acc: 0.9459 - val_loss: 0.3567 - val_acc: 0.8685
Epoch 20/20
250/250 [==============================] - 22s 87ms/step - loss: 0.131
4 - acc: 0.9529 - val_loss: 0.3978 - val_acc: 0.8595
```

Out[12]:

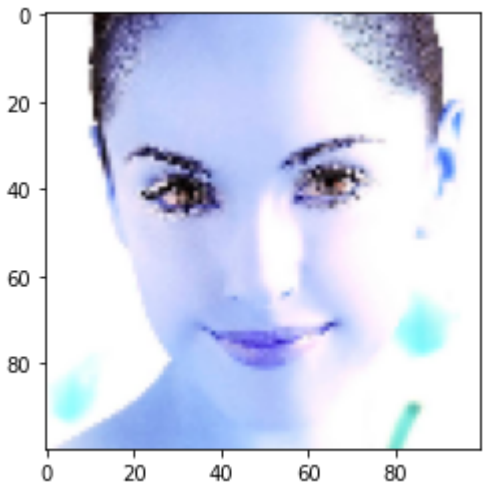
```
<keras.callbacks.History at 0x7fbd946ab890>
```

Evaluating the model--Predicting an input image

In [13]:
```
#Predicting model for class "0" which is happy

#print(model.predict_classes(xts[[1500]]))
print(model.predict(xts[[3]]))
plt.imshow(xts[3])
```

```
[[0.]]
```

Out[13]:
```
<matplotlib.image.AxesImage at 0x7fbda9f0cb50>
```



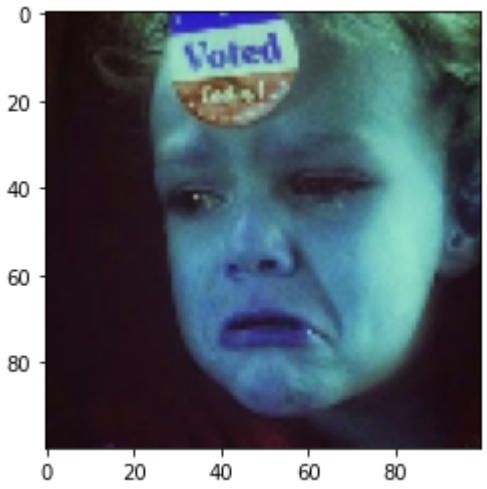
In [14]:

```
#Predicting model for class "1" which is sad

print(model.predict(xts[[480]]))
plt.imshow(xts[480])
```

```
[[1.]]
```

Out[14]:

```
<matplotlib.image.AxesImage at 0x7fbda16ee510>
```



In [15]:

```
##Summary for CNN model
model.summary()
```

```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d (Conv2D)              (None, 98, 98, 32)        896
_________________________________________________________________
max_pooling2d (MaxPooling2D) (None, 32, 32, 32)        0
_________________________________________________________________
conv2d_1 (Conv2D)            (None, 30, 30, 64)        18496
_________________________________________________________________
max_pooling2d_1 (MaxPooling2 (None, 10, 10, 64)        0
_________________________________________________________________
conv2d_2 (Conv2D)            (None, 8, 8, 90)          51930
_________________________________________________________________
max_pooling2d_2 (MaxPooling2 (None, 2, 2, 90)          0
_________________________________________________________________
flatten (Flatten)            (None, 360)               0
_________________________________________________________________
dense (Dense)                (None, 16)                5776
_________________________________________________________________
dense_1 (Dense)              (None, 1)                 17
=================================================================
Total params: 77,115
Trainable params: 77,115
Non-trainable params: 0
_________________________________________________________________
```

Evaluating Model

In [16]:
```
#Evaluating CNN model
ev1 = model.evaluate(xtsnorm,yts)
ev1
```

```
16/16 [==============================] - 0s 22ms/step - loss: 0.4059 -
acc: 0.8649
```

Out[16]:
```
[0.40588024258613586, 0.8649193644523621]
```

In [17]:
```
histo = model_hist.history
histo.keys()
```

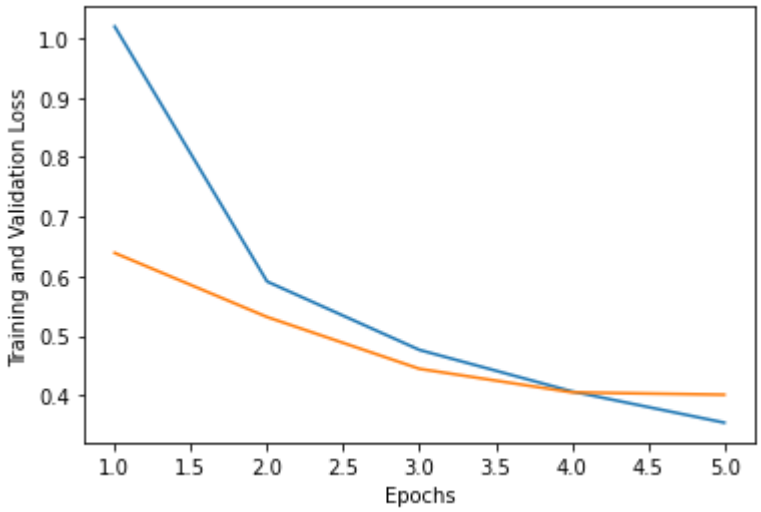
Out[17]:
```
dict_keys(['loss', 'acc', 'val_loss', 'val_acc'])
```

In [18]:

```
epochs = np.arange(1,len(histo['val_acc'])+1)
import matplotlib.pyplot as plt
plt.xlabel("Epochs")
plt.ylabel("Training and Validation Loss")
plt.plot(epochs,histo["loss"])
plt.plot(epochs,histo["val_loss"])
```

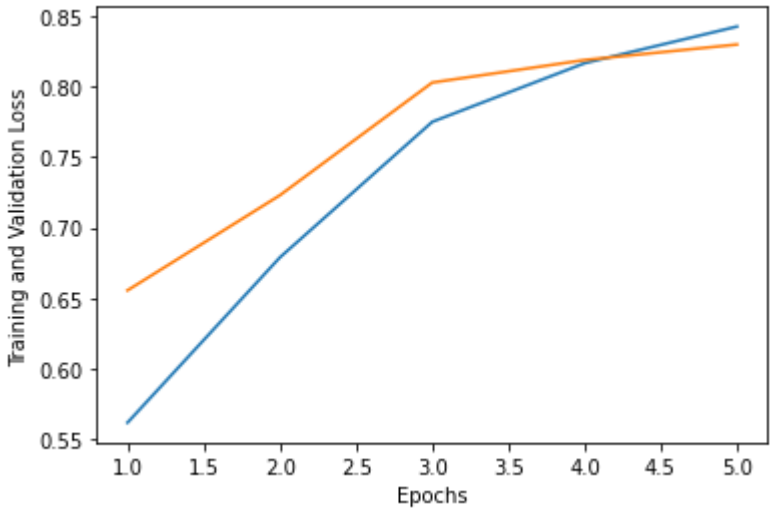
Out[18]:

```
[<matplotlib.lines.Line2D at 0x7fbda16ae110>]
```



In [19]:
```
epochs = np.arange(1,len(histo['val_acc'])+1)
import matplotlib.pyplot as plt
plt.xlabel("Epochs")
plt.ylabel("Training and Validation Loss")
plt.plot(epochs,histo["acc"])
plt.plot(epochs,histo["val_acc"])
```

Out[19]:
```
[<matplotlib.lines.Line2D at 0x7fbda1fc97d0>]
```



In [20]:
```
# plotting Accuracy
#plt.bar(['Cnn Acc','Ann Acc'],[ev1[1],ev2[1]])
```

In [21]:

```
path_5 = glob.glob("../input/accuracycheck/*.*")
cv_img = []
for img in path_5:
    n =  cv2.imread(img)
    n = cv2.resize(n,(100,100))
    cv_img.append(n)

array5 = np.asarray(cv_img)
print(array5.shape)
```

```
(10, 100, 100, 3)
```

Happy = 0

Sad = 1

Accuracy Results = 8/10

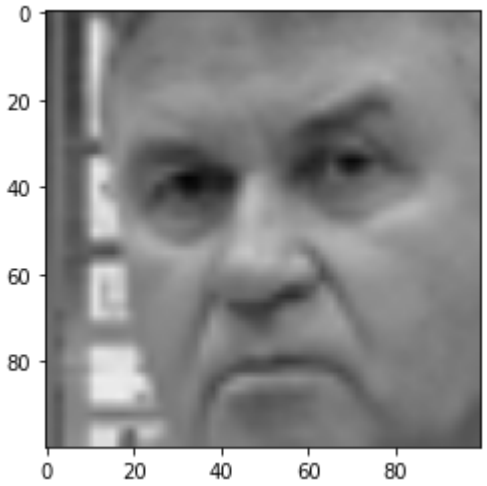
In [22]:

```
print(model.predict(array5[[9]]))
plt.imshow(array5[9])
```

```
[[1.]]
```

Out[22]:

```
<matplotlib.image.AxesImage at 0x7fbda0f03790>
```



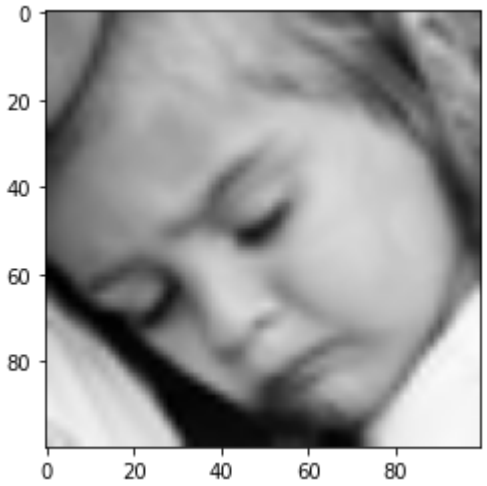
In [23]:

```
print(model.predict(array5[[8]]))
plt.imshow(array5[8])
```

```
[[1.]]
```

Out[23]:

```
<matplotlib.image.AxesImage at 0x7fbda1713a90>
```



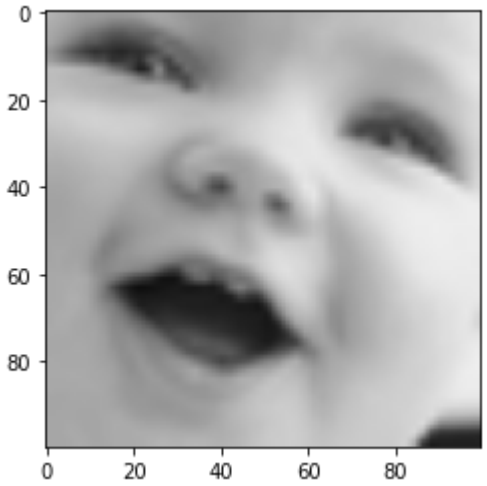
In [24]:

```
print(model.predict(array5[[7]]))
plt.imshow(array5[7])
```

```
[[0.]]
```

Out[24]:

```
<matplotlib.image.AxesImage at 0x7fbda154f450>
```



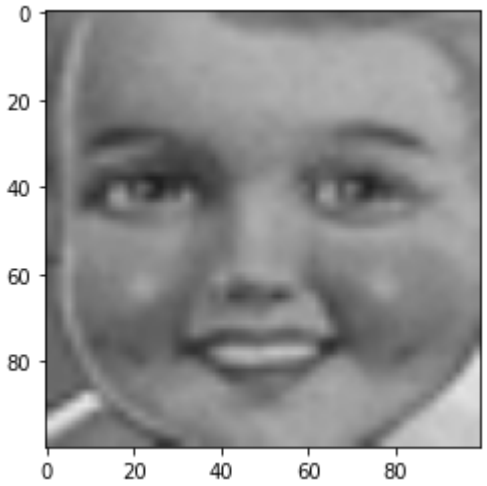
In [25]:

```
print(model.predict(array5[[6]]))
plt.imshow(array5[6])
```

```
[[0.]]
```

Out[25]:

```
<matplotlib.image.AxesImage at 0x7fbda14c5550>
```



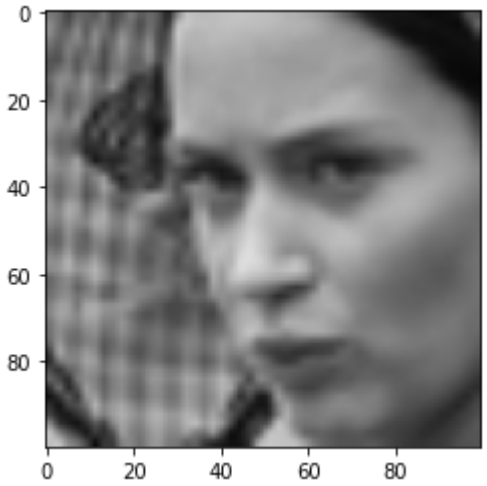
In [35]:

```
print(model.predict(array5[[5]]))
plt.imshow(array5[5])
```

```
[[1.4958497e-23]]
```

Out[35]:

```
<matplotlib.image.AxesImage at 0x7fbda10dd350>
```



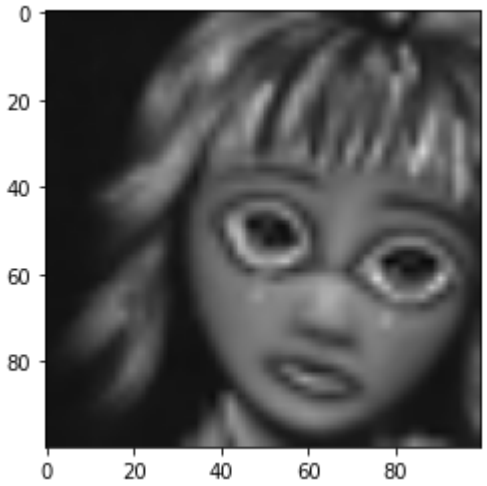
In [27]:

```
print(model.predict(array5[[4]]))
plt.imshow(array5[4])
```

```
[[1.]]
```

Out[27]:

```
<matplotlib.image.AxesImage at 0x7fbda142d750>
```



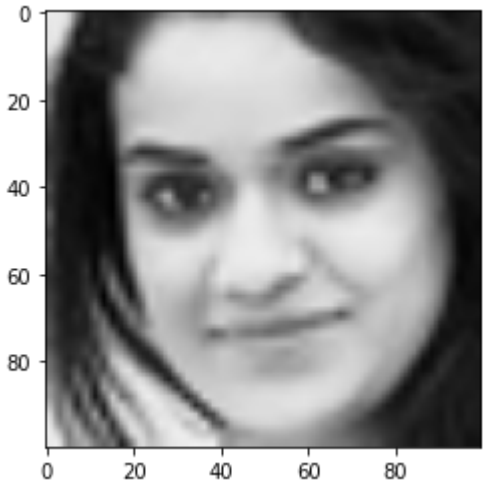
In [28]:

```
print(model.predict(array5[[3]]))
plt.imshow(array5[3])
```

```
[[0.]]
```

Out[28]:

```
<matplotlib.image.AxesImage at 0x7fbda13a1190>
```



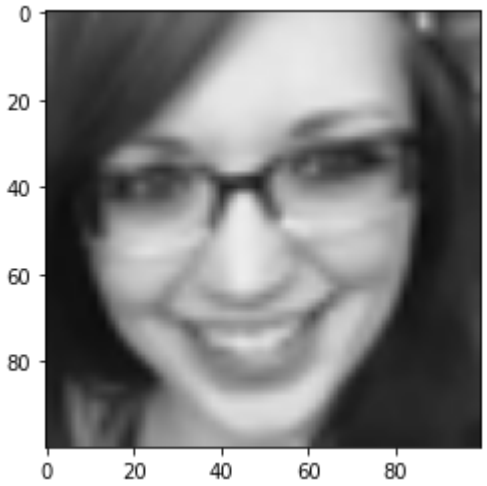
In [29]:

```
print(model.predict(array5[[2]]))
plt.imshow(array5[2])
```

```
[[0.]]
```

Out[29]:

```
<matplotlib.image.AxesImage at 0x7fbda130f090>
```



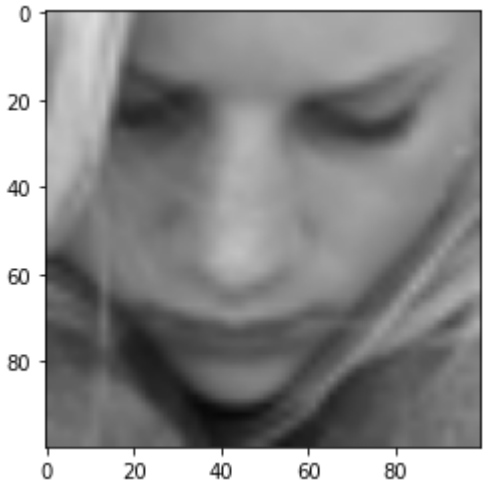
In [30]:

```
print(model.predict(array5[[1]]))
plt.imshow(array5[1])
```

```
[[0.]]
```

Out[30]:

```
<matplotlib.image.AxesImage at 0x7fbda128b650>
```



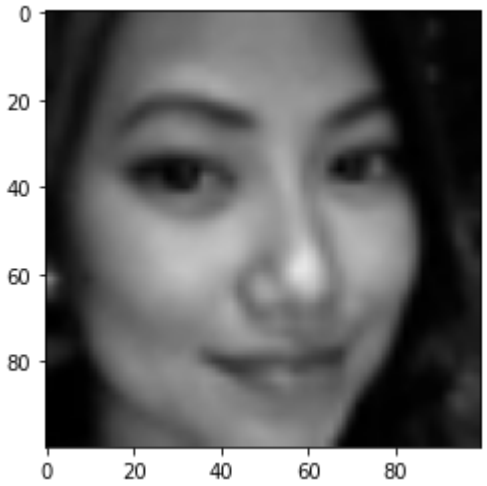
In [31]:

```
print(model.predict(array5[[0]]))
plt.imshow(array5[0])
```

```
[[0.]]
```

Out[31]:

```
<matplotlib.image.AxesImage at 0x7fbda12080d0>
```



Accuracy 8/10