capturing user image

while running this code if the user press space bar, the image will be captured. if esc is pressed means the camera window will be closed

In [36]:
```
import PIL.Image
import os
import cv2
img_counter =0
face_cascade = cv2.CascadeClassifier('haarcascade_frontalface_default.xml')
cap = cv2.VideoCapture(0)
path = r"C:\Users\aksha\OneDrive\Desktop\mini_project\baseData\testing"
while True:
    ret,img = cap.read()
    if not ret:
        print("failed to grab frame")
        break
    gray = cv2.cvtColor(img , cv2.COLOR_BGR2GRAY)
    faces = face_cascade.detectMultiScale(gray , 1.1 , 4)
    for (x , y , w , h) in faces:
        cv2.rectangle(img , (x,y) , (x+w, y+h) , (255,0,0) ,2)

    cv2.imshow('img',img)
    k = cv2.waitKey(1)
    if k==256 or 27:
        print('esc..pressed.....')
        break
    elif k==32 or 32:

    # SPACE pressed
        img_name = "opencv_frame_{}.png".format(img_counter)
        cv2.rectangle(img , (x,y) , (x+w, y+h) , (0,255,0) ,2)


        cv2.imwrite(os.path.join(path ,'pic.jpg'), img)
        print("{} written!".format(img_name))
        break

cap.release()
cv2.destroyAllWindows()
```

```
opencv_frame_0.png written!
```

upload the captured image in the firebase and retrieve the link of the image

In [37]:
```
#we have taken a training and test dataset for person with hair and person not having hair
```

In [38]:
```
#cnn module is created to check wether the ticket is given to valid person or not
```

importing necessary libraries

In [1]:
```
import tensorflow as tf
import matplotlib.pyplot as plt
import os
import cv2
import numpy as np
from tensorflow.keras.preprocessing.image import ImageDataGenerator
from tensorflow.keras.preprocessing import image
from tensorflow.keras.optimizers import RMSprop,Adam
```

In [2]:
```
img = image.load_img(r"C:\Users\aksha\OneDrive\Desktop\mini_project\baseData\training\faketicket\1.jpg")
```

In [3]:
```
plt.imshow(img)
```

Out[3]:
```
<matplotlib.image.AxesImage at 0x22b1a2b9d30>
```

In [4]:
```
#preprocessing image height and width
```

In [5]:
```
cv2.imread("C:\\Users\\aksha\\OneDrive\\Desktop\\mini_project\\baseData\\training\\valid_ticket(nothaving hair)/1.jpg")
```

Out[5]:
```
array([[[242, 238, 233],
        [242, 238, 233],
        [244, 238, 233],
        ...,
        [244, 238, 227],
        [243, 237, 226],
        [243, 237, 226]],

       [[242, 238, 233],
        [242, 238, 233],
        [243, 238, 232],
        ...,
        [244, 238, 227],
        [243, 237, 226],
        [243, 237, 226]],

       [[242, 238, 233],
        [242, 238, 233],
        [242, 238, 232],
        ...,
        [244, 238, 227],
        [243, 237, 226],
        [243, 237, 226]],

       ...,

       [[ 62,  53,  50],
        [ 61,  57,  56],
        [ 61,  56,  55],
        ...,
        [ 54,  50,  49],
        [ 55,  50,  51],
        [ 55,  50,  51]],

       [[ 60,  55,  54],
        [ 59,  54,  53],
        [ 59,  54,  53],
        ...,
        [ 55,  51,  50],
        [ 54,  49,  50],
        [ 54,  49,  50]],

       [[ 60,  55,  54],
        [ 59,  54,  53],
        [ 59,  54,  53],
        ...,
        [ 55,  51,  50],
        [ 54,  49,  50],
        [ 54,  49,  50]]], dtype=uint8)
```

In [6]:
```
train = ImageDataGenerator(rescale=1/255)
validation = ImageDataGenerator(rescale = 1/255)
```

training and validation dataset

In [7]:
```
train_dataset = train.flow_from_directory("C:\Users\aksha\OneDrive\Desktop\mini_project\baseData\training",
                                          target_size = (200,200),
                                          batch_size = 3,
                                          class_mode='binary')
validation_dataset =  train.flow_from_directory("C:\Users\aksha\OneDrive\Desktop\mini_project\baseData\validation",
                                          target_size = (200,200),
                                          batch_size = 3,
                                          class_mode='binary')
```

```
Found 1660 images belonging to 2 classes.
Found 249 images belonging to 2 classes.
```

creating the model

In [8]:
```
model = tf.keras.models.Sequential([tf.keras.layers.Conv2D(16,(3,3),activation='relu',input_shape=(200,200,3)),
                                    tf.keras.layers.MaxPool2D(2,2),
                                    #
                                    tf.keras.layers.Conv2D(32,(3,3),activation='relu'),
                                    tf.keras.layers.MaxPool2D(2,2),
                                    #
                                    tf.keras.layers.Conv2D(64,(3,3),activation='relu',input_shape=(200,200,3)),
                                    tf.keras.layers.MaxPool2D(2,2),

                                    #flatten
                                    tf.keras.layers.Flatten(),
                                    #Dense layers
                                    tf.keras.layers.Dense(512,activation='relu'),
                                    #
                                    tf.keras.layers.Dense(1,activation='sigmoid')])
```

compiling the model

In [9]:
```
model.compile(loss='binary_crossentropy',
              optimizer = RMSprop(lr=0.001),
              metrics = ['accuracy'])
```

```
C:\Users\aksha\anaconda3\lib\site-packages\keras\optimizer_v2\rmsprop.py:135: UserWarning: The `lr` argument is deprecated, use `learning_rate` ins
tead.
  super(RMSprop, self).__init__(name, **kwargs)
```

In [10]:
```
history = model.fit(train_dataset,
                    steps_per_epoch = 3,
                    epochs = 50,
                    validation_data = validation_dataset)
```

```
Epoch 1/50
3/3 [==============================] - 6s 3s/step - loss: 0.7312 - accuracy: 0.7143 - val_loss: 0.5536 - val_accuracy: 0.7208
Epoch 2/50
3/3 [==============================] - 6s 3s/step - loss: 0.3618 - accuracy: 0.8889 - val_loss: 0.5201 - val_accuracy: 0.7208
Epoch 3/50
3/3 [==============================] - 7s 3s/step - loss: 0.5781 - accuracy: 0.7778 - val_loss: 0.5389 - val_accuracy: 0.7883
Epoch 4/50
3/3 [==============================] - 7s 3s/step - loss: 0.3096 - accuracy: 0.8889 - val_loss: 1.1761 - val_accuracy: 0.6542
Epoch 5/50
3/3 [==============================] - 6s 3s/step - loss: 0.3973 - accuracy: 0.8889 - val_loss: 1.2679 - val_accuracy: 0.5875
Epoch 6/50
3/3 [==============================] - 6s 3s/step - loss: 0.9783 - accuracy: 0.6667 - val_loss: 1.3004 - val_accuracy: 0.5458
Epoch 7/50
3/3 [==============================] - 7s 3s/step - loss: 1.1494 - accuracy: 0.6667 - val_loss: 0.4079 - val_accuracy: 0.7708
Epoch 8/50
3/3 [==============================] - 6s 3s/step - loss: 0.1610 - accuracy: 1.0000 - val_loss: 0.5829 - val_accuracy: 0.7167
Epoch 9/50
3/3 [==============================] - 6s 3s/step - loss: 0.1816 - accuracy: 1.0000 - val_loss: 0.5461 - val_accuracy: 0.7625
Epoch 10/50
3/3 [==============================] - 6s 3s/step - loss: 0.8719 - accuracy: 0.7778 - val_loss: 0.5471 - val_accuracy: 0.7250
Epoch 11/50
3/3 [==============================] - 6s 3s/step - loss: 0.2896 - accuracy: 0.8889 - val_loss: 0.4512 - val_accuracy: 0.8000
Epoch 12/50
3/3 [==============================] - 6s 3s/step - loss: 0.4829 - accuracy: 0.6667 - val_loss: 0.8853 - val_accuracy: 0.5917
Epoch 13/50
3/3 [==============================] - 6s 3s/step - loss: 0.4635 - accuracy: 0.7778 - val_loss: 0.4059 - val_accuracy: 0.8250
Epoch 14/50
3/3 [==============================] - 6s 3s/step - loss: 0.0352 - accuracy: 1.0000 - val_loss: 0.6736 - val_accuracy: 0.7500
Epoch 15/50
3/3 [==============================] - 6s 3s/step - loss: 0.3185 - accuracy: 0.7778 - val_loss: 0.4718 - val_accuracy: 0.7667
Epoch 16/50
3/3 [==============================] - 6s 3s/step - loss: 0.2590 - accuracy: 0.8889 - val_loss: 0.4133 - val_accuracy: 0.7833
Epoch 17/50
3/3 [==============================] - 6s 3s/step - loss: 1.2129 - accuracy: 0.5556 - val_loss: 0.4326 - val_accuracy: 0.8250
Epoch 18/50
3/3 [==============================] - 6s 3s/step - loss: 0.5596 - accuracy: 0.4444 - val_loss: 0.6377 - val_accuracy: 0.6625
Epoch 19/50
3/3 [==============================] - 6s 3s/step - loss: 0.1808 - accuracy: 1.0000 - val_loss: 0.7888 - val_accuracy: 0.6417
Epoch 20/50
3/3 [==============================] - 6s 3s/step - loss: 0.2485 - accuracy: 0.8889 - val_loss: 0.4340 - val_accuracy: 0.7792
Epoch 21/50
3/3 [==============================] - 6s 3s/step - loss: 0.3021 - accuracy: 0.7778 - val_loss: 0.4480 - val_accuracy: 0.7792
Epoch 22/50
3/3 [==============================] - 6s 3s/step - loss: 0.6679 - accuracy: 0.7778 - val_loss: 0.4662 - val_accuracy: 0.7542
Epoch 23/50
3/3 [==============================] - 6s 3s/step - loss: 0.5882 - accuracy: 0.6667 - val_loss: 0.4488 - val_accuracy: 0.7833
Epoch 24/50
3/3 [==============================] - 6s 3s/step - loss: 0.5247 - accuracy: 0.8889 - val_loss: 0.4604 - val_accuracy: 0.7917
Epoch 25/50
3/3 [==============================] - 6s 3s/step - loss: 0.1967 - accuracy: 0.8889 - val_loss: 0.4513 - val_accuracy: 0.7917
Epoch 26/50
3/3 [==============================] - 6s 3s/step - loss: 0.4507 - accuracy: 0.7778 - val_loss: 0.4225 - val_accuracy: 0.8042
Epoch 27/50
3/3 [==============================] - 6s 3s/step - loss: 0.1662 - accuracy: 1.0000 - val_loss: 0.4186 - val_accuracy: 0.8042
Epoch 28/50
3/3 [==============================] - 6s 3s/step - loss: 0.2005 - accuracy: 0.8889 - val_loss: 0.6146 - val_accuracy: 0.7583
Epoch 29/50
3/3 [==============================] - 6s 3s/step - loss: 0.2913 - accuracy: 0.8889 - val_loss: 0.4693 - val_accuracy: 0.7750
Epoch 30/50
3/3 [==============================] - 6s 3s/step - loss: 0.2778 - accuracy: 0.8889 - val_loss: 0.6162 - val_accuracy: 0.7083
Epoch 31/50
3/3 [==============================] - 6s 3s/step - loss: 0.6337 - accuracy: 0.7778 - val_loss: 0.4764 - val_accuracy: 0.7667
Epoch 32/50
3/3 [==============================] - 6s 3s/step - loss: 0.9338 - accuracy: 0.5556 - val_loss: 0.4367 - val_accuracy: 0.7833
Epoch 33/50
3/3 [==============================] - 6s 3s/step - loss: 0.7248 - accuracy: 0.6667 - val_loss: 0.4748 - val_accuracy: 0.8000
Epoch 34/50
3/3 [==============================] - 6s 3s/step - loss: 0.2931 - accuracy: 0.7778 - val_loss: 0.4346 - val_accuracy: 0.8125
Epoch 35/50
3/3 [==============================] - 6s 3s/step - loss: 0.2392 - accuracy: 0.8889 - val_loss: 0.4326 - val_accuracy: 0.7833
Epoch 36/50
3/3 [==============================] - 6s 3s/step - loss: 0.1859 - accuracy: 0.8889 - val_loss: 0.4642 - val_accuracy: 0.7875
Epoch 37/50
3/3 [==============================] - 6s 3s/step - loss: 0.4742 - accuracy: 0.8889 - val_loss: 0.4112 - val_accuracy: 0.7917
Epoch 38/50
3/3 [==============================] - 6s 3s/step - loss: 0.3053 - accuracy: 0.7778 - val_loss: 0.5427 - val_accuracy: 0.7792
Epoch 39/50
3/3 [==============================] - 6s 3s/step - loss: 0.2930 - accuracy: 0.8889 - val_loss: 0.4605 - val_accuracy: 0.7583
Epoch 40/50
3/3 [==============================] - 6s 3s/step - loss: 0.6178 - accuracy: 0.7778 - val_loss: 0.4432 - val_accuracy: 0.8125
Epoch 41/50
3/3 [==============================] - 6s 3s/step - loss: 0.3273 - accuracy: 0.7778 - val_loss: 0.4988 - val_accuracy: 0.7958
Epoch 42/50
3/3 [==============================] - 6s 3s/step - loss: 0.4485 - accuracy: 0.7778 - val_loss: 0.4605 - val_accuracy: 0.8083
Epoch 43/50
3/3 [==============================] - 6s 3s/step - loss: 0.5875 - accuracy: 0.7778 - val_loss: 0.4088 - val_accuracy: 0.8042
Epoch 44/50
3/3 [==============================] - 6s 3s/step - loss: 0.1709 - accuracy: 1.0000 - val_loss: 0.4604 - val_accuracy: 0.8083
Epoch 45/50
3/3 [==============================] - 6s 3s/step - loss: 0.2994 - accuracy: 0.8889 - val_loss: 0.7771 - val_accuracy: 0.6833
Epoch 46/50
3/3 [==============================] - 6s 3s/step - loss: 0.1337 - accuracy: 1.0000 - val_loss: 1.2457 - val_accuracy: 0.6167
Epoch 47/50
3/3 [==============================] - 6s 3s/step - loss: 0.7776 - accuracy: 0.6667 - val_loss: 0.4202 - val_accuracy: 0.8167
Epoch 48/50
3/3 [==============================] - 6s 3s/step - loss: 0.5518 - accuracy: 0.6667 - val_loss: 0.4289 - val_accuracy: 0.8042
Epoch 49/50
3/3 [==============================] - 6s 3s/step - loss: 0.3158 - accuracy: 0.7778 - val_loss: 0.4188 - val_accuracy: 0.7917
Epoch 50/50
3/3 [==============================] - 6s 3s/step - loss: 0.3969 - accuracy: 0.8667 - val_loss: 0.4099 - val_accuracy: 0.8208
```

In [11]:
```
history.history.keys()
```

Out[11]:
```
dict_keys(['loss', 'accuracy', 'val_loss', 'val_accuracy'])
```

model summary

In [20]:
```
model.summary()
```

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d (Conv2D)             (None, 198, 198, 16)      448

 max_pooling2d (MaxPooling2D  (None, 99, 99, 16)       0
 )

 conv2d_1 (Conv2D)           (None, 97, 97, 32)        4640

 max_pooling2d_1 (MaxPooling  (None, 48, 48, 32)       0
 2D)

 conv2d_2 (Conv2D)           (None, 46, 46, 64)        18496

 max_pooling2d_2 (MaxPooling  (None, 23, 23, 64)       0
 2D)

 flatten (Flatten)           (None, 33856)             0

 dense (Dense)               (None, 512)               17334784

 dense_1 (Dense)             (None, 1)                 513

=================================================================
Total params: 17,358,881
Trainable params: 17,358,881
Non-trainable params: 0
_________________________________________________________________
```

In [21]:
```
validation_dataset.class_indices
```

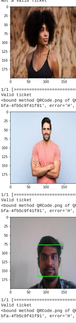
Out[21]:
```
{'faketicket': 0, 'valid_ticket(nothaving hair)': 1}
```

In [ ]:
```

```

In [ ]:
```

```

In [29]:
```
dir_path = r"C:\Users\aksha\OneDrive\Desktop\mini_project\baseData\testing"
for i in os.listdir(dir_path):
    img = image.load_img(dir_path+"/"+i,target_size = (200,200))
    plt.imshow(img)
    plt.show()

    X = image.img_to_array(img)
    X = np.expand_dims(X,axis=0)
    val = model.predict(X)
    print(val)
    if val == 0:
        print("Valid ticket")
        import pyqrcode
        import png
        from pyqrcode import QRCode

        # String which represents the QR code
        s = "https://firebasestorage.googleapis.com/v0/b/mini-6e8f2.appspot.com/o/pic.jpg?alt=media&token=211fc9b5-21ef-4a1b-8bfa-8fa4f5b3f41f91"

        # Generate QR code
        url = pyqrcode.create(s)

        # Create and save the svg file naming "myqr.svg"
        url.svg("myqr1.svg", scale = 8)

        # Create and save the png file naming "myqr.png"
        url.png('myqr1.png', scale = 8)
        print(url.png)
    else:
        print("Not a Valid Ticket ")
```

```
1/1 [==============================] - 1s 1s/step
Not a Valid Ticket
```



```
1/1 [==============================] - 0s 64ms/step
Valid ticket
<bound method QRCode.png of QRCode(content='https://firebasestorage.googleapis.com/v0/b/mini-6e8f2.appspot.com/o/pic.jpg?alt=media&token=211fc9b5-21ef-4a1b-8bfa-8fa4f5b3f41f91', error='e', version=11, mode='binary')>
```



```
1/1 [==============================] - 0s 67ms/step
Valid ticket
<bound method QRCode.png of QRCode(content='https://firebasestorage.googleapis.com/v0/b/mini-6e8f2.appspot.com/o/pic.jpg?alt=media&token=211fc9b5-21ef-4a1b-8bfa-8fa4f5b3f41f91', error='e', version=11, mode='binary')>
```



```
1/1 [==============================] - 0s 75ms/step
Valid ticket
<bound method QRCode.png of QRCode(content='https://firebasestorage.googleapis.com/v0/b/mini-6e8f2.appspot.com/o/pic.jpg?alt=media&token=211fc9b5-21ef-4a1b-8bfa-8fa4f5b3f41f91', error='e', version=11, mode='binary')>
```

In [ ]:
```

```