3. Facial Keypoint Detection, Complete Pipeline

April 25, 2020

0.1 Face and Facial Keypoint detection

After you've trained a neural network to detect facial keypoints, you can then apply this network to *any* image that includes faces. The neural network expects a Tensor of a certain size as input and, so, to detect any face, you'll first have to do some pre-processing.

1. Detect all the faces in an image using a face detector (we'll be using a Haar Cascade detector in this notebook).
2. Pre-process those face images so that they are grayscale, and transformed to a Tensor of the input size that your net expects. This step will be similar to the `data_transform` you created and applied in Notebook 2, whose job was tp rescale, normalize, and turn any iimage into a Tensor to be accepted as input to your CNN.
3. Use your trained model to detect facial keypoints on the image.

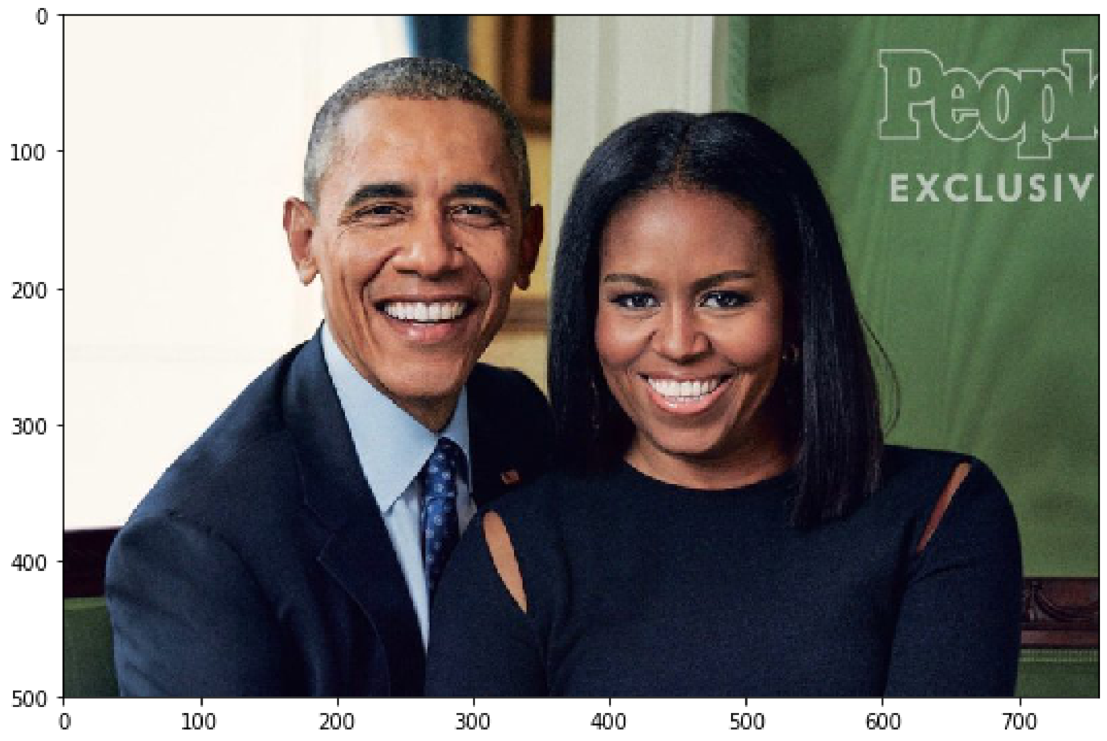
In the next python cell we load in required libraries for this section of the project.

```
In [1]: import numpy as np
        import matplotlib.pyplot as plt
        import matplotlib.image as mpimg
        %matplotlib inline
```

Select an image Select an image to perform facial keypoint detection on; you can select any image of faces in the `images/` directory.

```
In [2]: import cv2
        # load in color image for face detection
        image = cv2.imread('images/obamas.jpg')

        # switch red and blue color channels
        # --> by default OpenCV assumes BLUE comes first, not RED as in many images
        image = cv2.cvtColor(image, cv2.COLOR_BGR2RGB)

        # plot the image
        fig = plt.figure(figsize=(9,9))
        plt.imshow(image)
```

```
Out[2]: <matplotlib.image.AxesImage at 0x7f34f910c358>
```



0.2 Detect all faces in an image

Next, you'll use one of OpenCV's pre-trained Haar Cascade classifiers, all of which can be found in the `detector_architectures/` directory, to find any faces in your selected image.

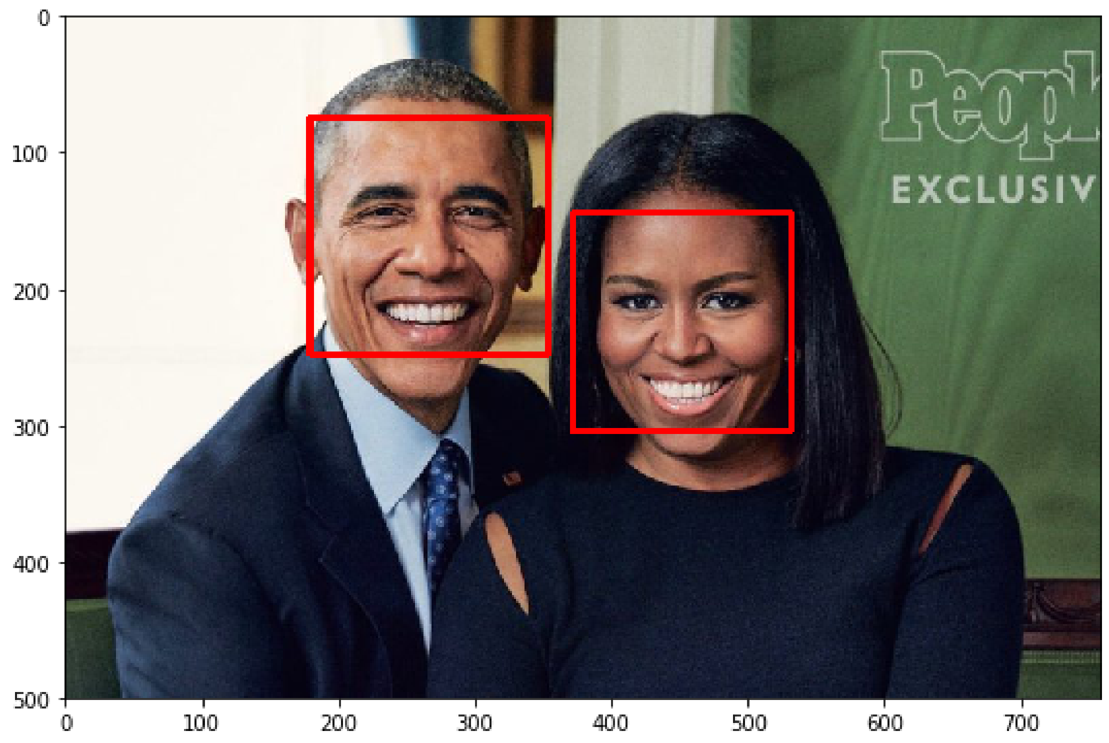
In the code below, we loop over each face in the original image and draw a red square on each face (in a copy of the original image, so as not to modify the original). You can even add eye detections as an *optional* exercise in using Haar detectors.

An example of face detection on a variety of images is shown below.

```
In [3]: # load in a haar cascade classifier for detecting frontal faces
        face_cascade = cv2.CascadeClassifier('detector_architectures/haarcascade_frontalface_def

        # run the detector
        # the output here is an array of detections; the corners of each detection box
        # if necessary, modify these parameters until you successfully identify every face in a
        faces = face_cascade.detectMultiScale(image, 1.2, 2)

        # make a copy of the original image to plot detections on
        image_with_detections = image.copy()

        # loop over the detected faces, mark the image where each face is found
```

```
         for (x,y,w,h) in faces:
             # draw a rectangle around each detected face
             # you may also need to change the width of the rectangle drawn depending on image re
             cv2.rectangle(image_with_detections,(x,y),(x+w,y+h),(255,0,0),3)

         fig = plt.figure(figsize=(9,9))

         plt.imshow(image_with_detections)
```

```
Out[3]: <matplotlib.image.AxesImage at 0x7f34f90a3f60>
```



0.3 Loading in a trained model

Once you have an image to work with (and, again, you can select any image of faces in the `images/` directory), the next step is to pre-process that image and feed it into your CNN facial keypoint detector.

First, load your best model by its filename.

```
In [11]: import torch
         from models import Net

         net = Net()

         ## TODO: load the best saved model parameters (by your path name)
## You'll need to un-comment the line below and add the correct name for *your* saved m
net.load_state_dict(torch.load('saved_models/keypoints_model_3.pt'))

## print out your net and prepare it for testing (uncomment the line below)
net.eval()
```

```
Out[11]: Net(
           (conv1): Conv2d(1, 32, kernel_size=(5, 5), stride=(1, 1))
           (conv2): Conv2d(32, 64, kernel_size=(3, 3), stride=(1, 1))
           (conv3): Conv2d(64, 128, kernel_size=(3, 3), stride=(1, 1))
           (conv4): Conv2d(128, 256, kernel_size=(2, 2), stride=(1, 1))
           (pool): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
           (fc1): Linear(in_features=36864, out_features=1000, bias=True)
           (fc2): Linear(in_features=1000, out_features=1000, bias=True)
           (fc3): Linear(in_features=1000, out_features=136, bias=True)
           (drop1): Dropout(p=0.1)
           (drop2): Dropout(p=0.2)
           (drop3): Dropout(p=0.3)
           (drop4): Dropout(p=0.4)
           (drop5): Dropout(p=0.5)
           (drop6): Dropout(p=0.6)
         )
```

0.4 Keypoint detection

Now, we'll loop over each detected face in an image (again!) only this time, you'll transform those faces in Tensors that your CNN can accept as input images.

0.4.1 TODO: Transform each detected face into an input Tensor

You'll need to perform the following steps for each detected face: 1. Convert the face from RGB to grayscale 2. Normalize the grayscale image so that its color range falls in [0,1] instead of [0,255] 3. Rescale the detected face to be the expected square size for your CNN (224x224, suggested) 4. Reshape the numpy image into a torch image.

Hint: The sizes of faces detected by a Haar detector and the faces your network has been trained on are of different sizes. If you find that your model is generating keypoints that are too small for a given face, try adding some padding to the detected `roi` before giving it as input to your model.

You may find it useful to consult to transformation code in `data_load.py` to help you perform these processing steps.

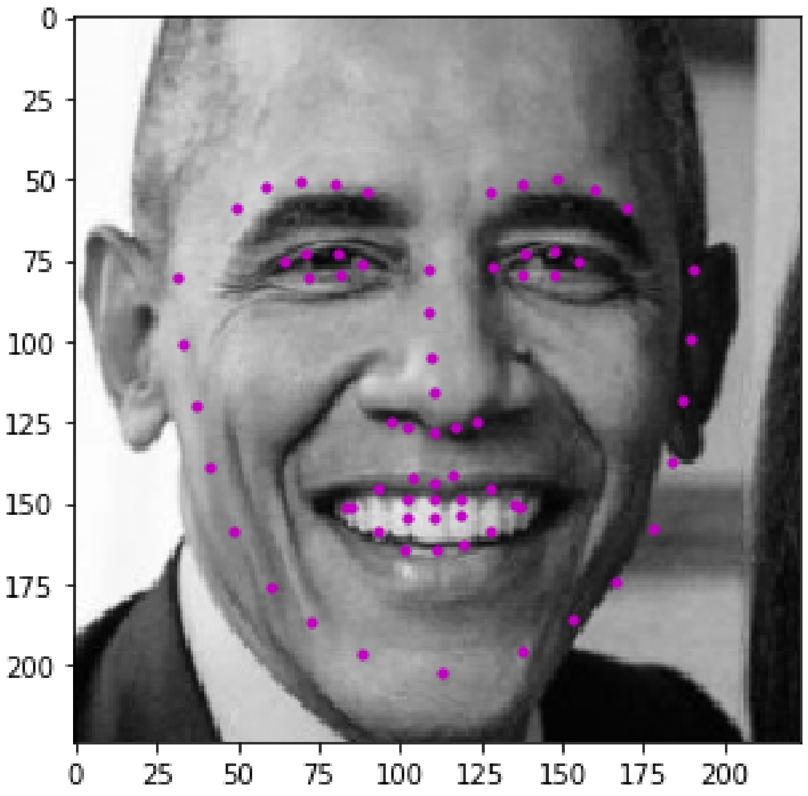
0.4.2 TODO: Detect and display the predicted keypoints

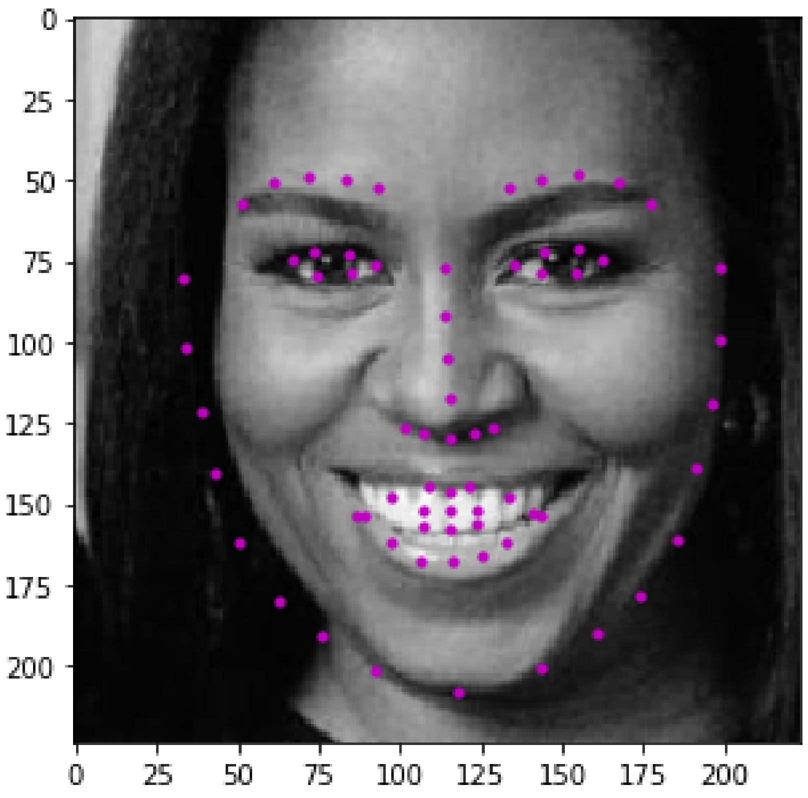
After each face has been appropriately converted into an input Tensor for your network to see as input, you can apply your `net` to each face. The ouput should be the predicted the facial keypoints. These keypoints will need to be "un-normalized" for display, and you may find it helpful to write a helper function like `show_keypoints`. You should end up with an image like the following with facial keypoints that closely match the facial features on each individual face:

```
In [13]: key = [None] * 136;
         i=1
         j=i+1
         def show_all_keypoints(image, keypoints):
             """
             Visuzlizing the image and the keypoints on it.
             """
             plt.figure(figsize=(10,5))
             i=1
             j=i+1
             keypoints = keypoints.data.numpy()
             keypoints = keypoints * 70.0 + 100 # Becuase of normalization, keypoints won't be p


             while i<j:
                  key[i]=keypoints
                  i=i+1
             keypoints = np.reshape(keypoints, (68, -1)) # reshape to 2 X 68 keypoint for the fa

             image = image.numpy()
             image = np.transpose(image, (1, 2, 0))  # Convert to numpy image shape (H x W x C)
             image = np.squeeze(image)
             plt.imshow(image, cmap='gray')
             plt.scatter(keypoints[:, 0], keypoints[:, 1], s=40, marker='.', c='m')

         import numpy as np
         from torch.autograd import Variable
         image_copy = np.copy(image)

         # loop over the detected faces from your haar cascade
         for (x,y,w,h) in faces:
             roi = cv2.cvtColor(image_copy, cv2.COLOR_RGB2GRAY)
             # Select the region of interest that is the face in the image
             roi = roi[y:y + int(1.2 * h), x - int(0.12 * w):x + int(1.1 * w)]

             ## TODO: Convert the face region from RGB to grayscale
             #roi = cv2.cvtColor(roi, cv2.COLOR_RGB2GRAY)
             ## TODO: Normalize the grayscale image so that its color range falls in [0,1] inste
             roi = roi/255.0
             ## TODO: Rescale the detected face to be the expected square size for your CNN (224
             roi=cv2.resize(roi, (224, 224))
             ## TODO: Reshape the numpy image shape (H x W x C) into a torch image shape (C x H
             roi = np.reshape(roi, (1, 1, 224, 224))
             ## TODO: Make facial keypoint predictions using your loaded, trained network
             roi_torch = Variable(torch.from_numpy(roi))

             roi_torch = roi_torch.type(torch.FloatTensor)
             keypoints = net(roi_torch)
```

```
        ## TODO: Display each detected face and the corresponding keypoints
        show_all_keypoints(roi_torch.squeeze(0), keypoints)

    key = np.array(key)
    key=key.flatten()
    key1=key[1]
    key1=key1.reshape(-1,2)
    print(key1.shape)
```

```
(68, 2)
```

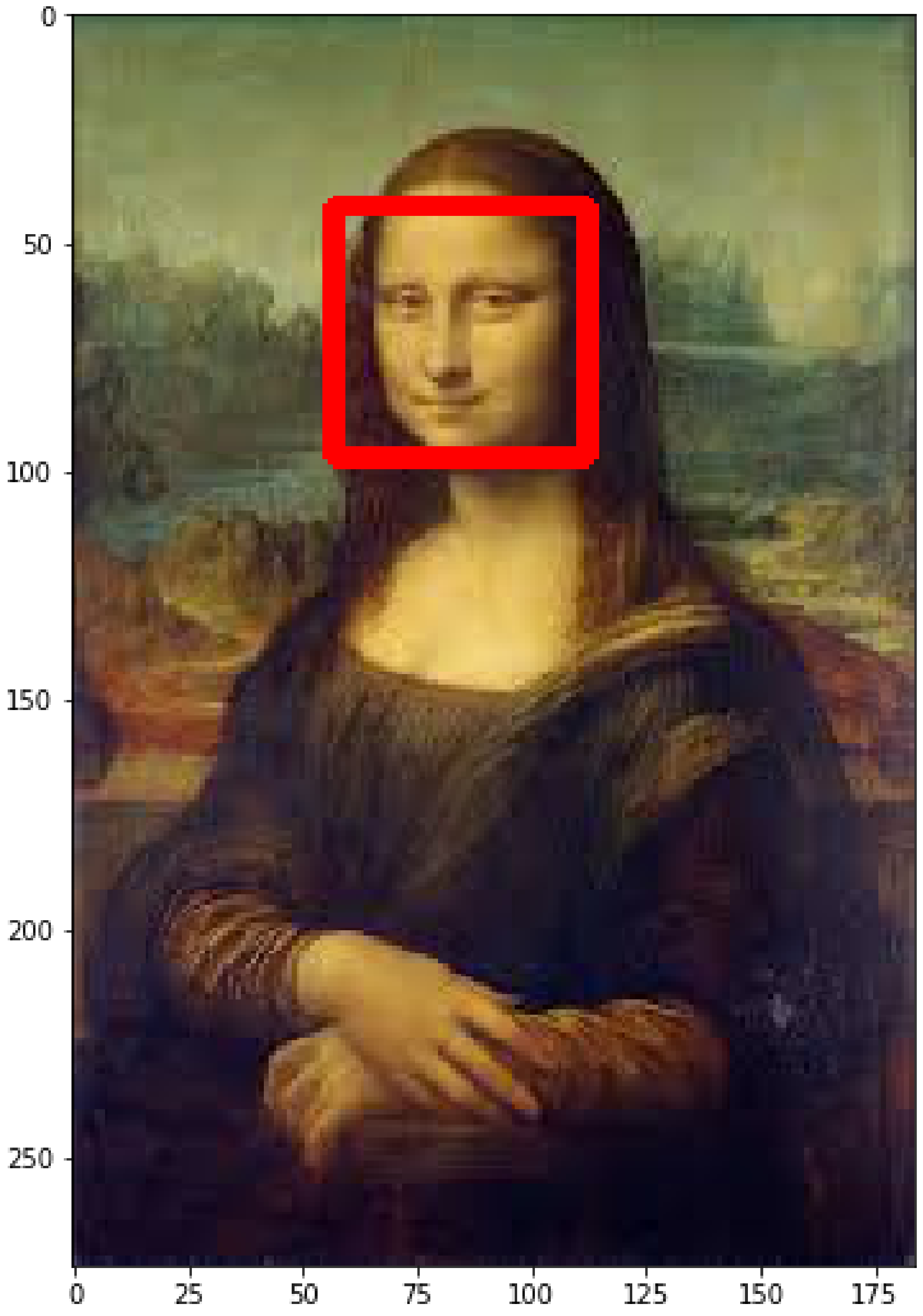


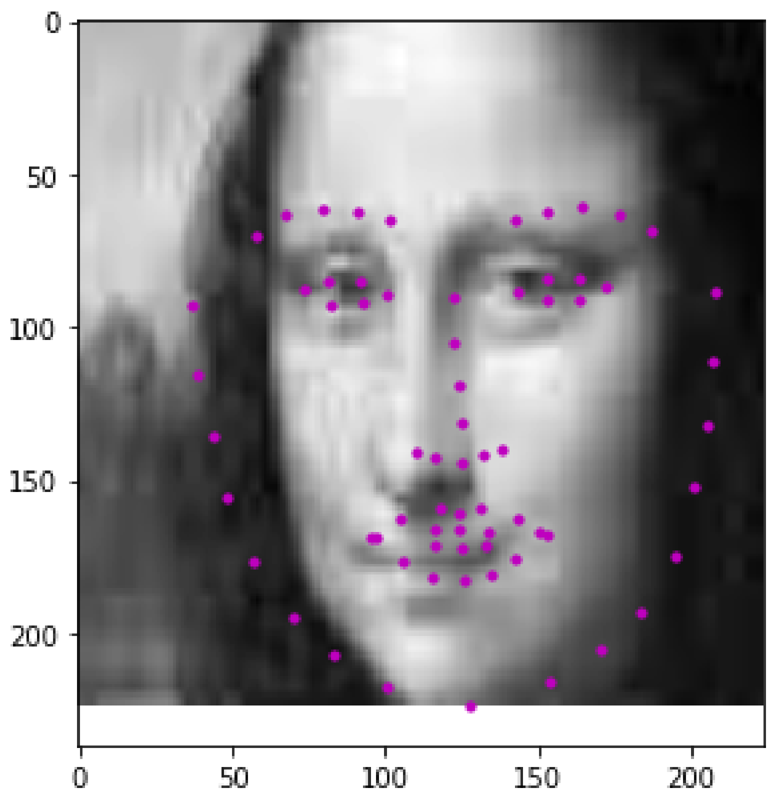


```
In [14]: import cv2
         # load in color image for face detection
         image = cv2.imread('images/mona_lisa.jpg')

         # switch red and blue color channels
         # --> by default OpenCV assumes BLUE comes first, not RED as in many images
         image = cv2.cvtColor(image, cv2.COLOR_BGR2RGB)

         # plot the image
         fig = plt.figure(figsize=(9,9))
         plt.imshow(image)
         # load in a haar cascade classifier for detecting frontal faces
         face_cascade = cv2.CascadeClassifier('detector_architectures/haarcascade_frontalface_de

         # run the detector
         # the output here is an array of detections; the corners of each detection box
         # if necessary, modify these parameters until you successfully identify every face in a
         faces = face_cascade.detectMultiScale(image, 1.3, 4)

         # make a copy of the original image to plot detections on
         image_with_detections = image.copy()
```

```
# loop over the detected faces, mark the image where each face is found
for (x,y,w,h) in faces:
    # draw a rectangle around each detected face
    # you may also need to change the width of the rectangle drawn depending on image r
    cv2.rectangle(image_with_detections,(x,y),(x+w,y+h),(255,0,0),3)

fig = plt.figure(figsize=(9,9))

plt.imshow(image_with_detections)
from torch.autograd import Variable

image_copy = np.copy(image)

# loop over the detected faces from your haar cascade
for (x,y,w,h) in faces:

    # Select the region of interest that is the face in the image
    roi = image_copy[y:y + int(1.0 * h), x - int(0.2 * w):x + int(1.1 * w)]
    #plt.imshow(roi)

    ## TODO: Convert the face region from RGB to grayscale
    roi = cv2.cvtColor(roi, cv2.COLOR_RGB2GRAY)
    #plt.imshow(roi, cmap = 'gray')

    ## TODO: Normalize the grayscale image so that its color range falls in [0,1] inste
    roi = roi / 255.
    #plt.imshow(roi, cmap = 'gray')

    ## TODO: Rescale the detected face to be the expected square size for your CNN (224
    roi = cv2.resize(roi, (224, 224))
    #plt.imshow(roi, cmap = 'gray')

    ## TODO: Reshape the numpy image shape (H x W x C) into a torch image shape (C x H
    roi = np.expand_dims(roi, 0)
    roi = np.expand_dims(roi, 0) # (1, 1, 224, 224)
    # roi = np.reshape(roi, (1, 1, 224, 224)) # Option 2
    #print(roi.shape)

    ## TODO: Make facial keypoint predictions using your loaded, trained network
    ## perform a forward pass to get the predicted facial keypoints
    roi_torch = Variable(torch.from_numpy(roi)) # Converting numpy to torch variable
    #print(roi_torch.shape)
    roi_torch = roi_torch.type(torch.FloatTensor)
    keypoints = net(roi_torch) # Forward pass

    ## TODO: Display each detected face and the corresponding keypoints
    show_all_keypoints(roi_torch.squeeze(0), keypoints)
```

0
50
100
150
200
250
0
25
50
75
100
125
150
175


0
50
100
150
200
250
0
25
50
75
100
125
150
175

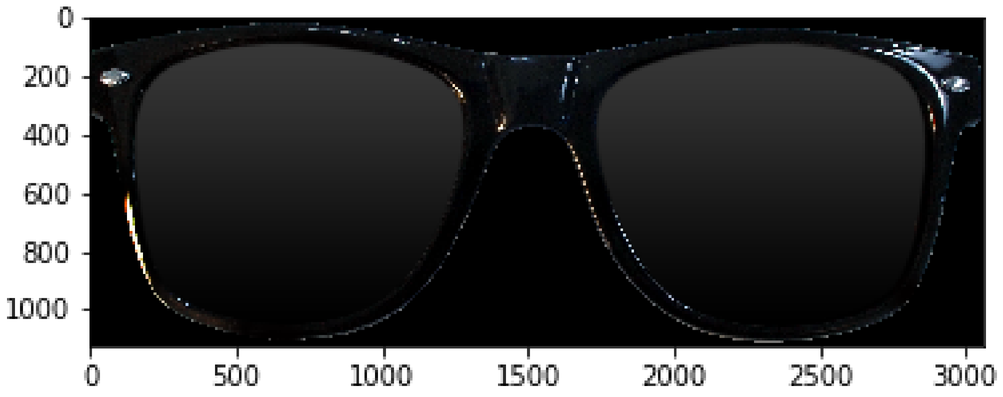


```
In [15]: import matplotlib.image as mpimg
         import matplotlib.pyplot as plt
         import numpy as np
         import pandas as pd
         import os
         import cv2
         sunglasses = cv2.imread('images/sunglasses.png', cv2.IMREAD_UNCHANGED)

         # plot our image


         # print out its dimensions
         print('Image shape: ', sunglasses.shape)
         alpha_channel = sunglasses[:,:,3]
         print ('The alpha channel looks like this (black pixels = transparent): ')
         plt.imshow(alpha_channel, cmap='gray')
         values = np.where(alpha_channel != 0)
         print ('The non-zero values of the alpha channel are: ')
         print (values)
         plt.imshow(sunglasses)
```

```
Image shape:  (1123, 3064, 4)
```

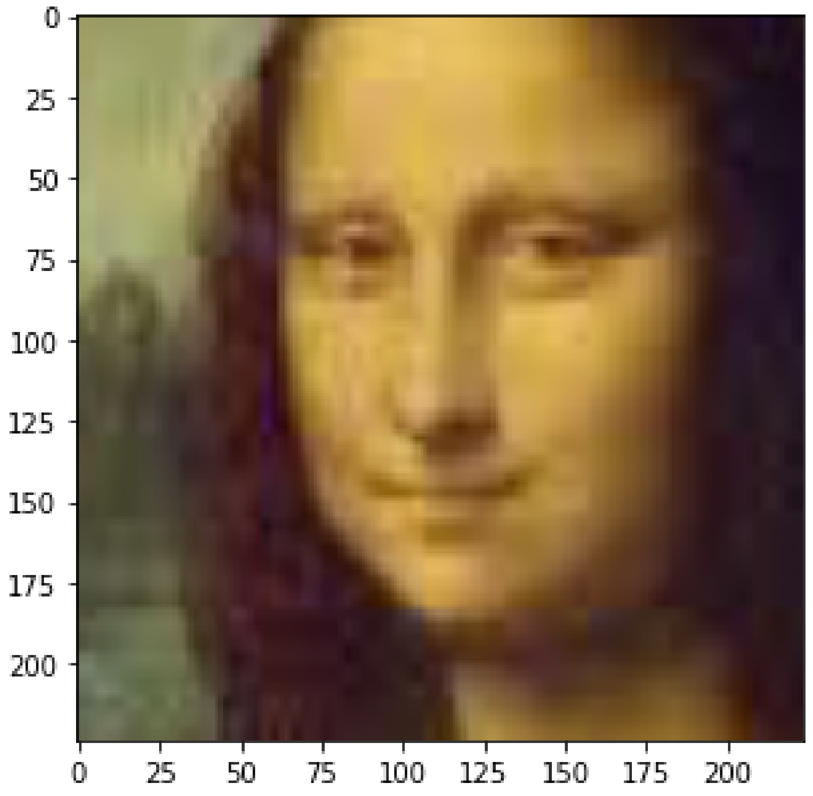
```
The alpha channel looks like this (black pixels = transparent):
The non-zero values of the alpha channel are:
(array([  17,   17,   17, ..., 1109, 1109, 1109]), array([ 687,  688,  689, ..., 2376, 2377, 237
```

```
Out[15]: <matplotlib.image.AxesImage at 0x7f348497dd30>
```



```
In [16]: image = cv2.imread('images/mona_lisa.jpg')

         # switch red and blue color channels
         # --> by default OpenCV assumes BLUE comes first, not RED as in many images
         image = cv2.cvtColor(image, cv2.COLOR_BGR2RGB)
         image = image[y:y + int(1.2 * h), x - int(0.2 * w):x + int(1.1 * w)]
         image=cv2.resize(image,(224,224))
         #image=np.reshape(image,(1,1,224,224))
         # plot the image
         fig = plt.figure(figsize=(5,5))
         plt.imshow(image)
```

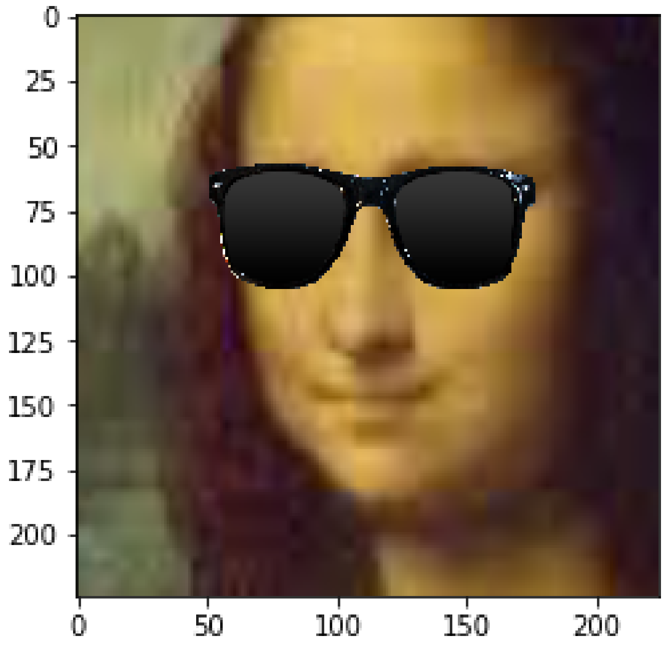
```
Out[16]: <matplotlib.image.AxesImage at 0x7f34848ecf28>
```



```
In [17]: import numpy as np

         image_copy = np.copy(image)
         #key=key.numpy()
         #print(key)
         # top-left location for sunglasses to go
         # 17 = edge of left eyebrow
         x = int(key1[17, 0]) #205
         y = int(key1[17, 1]) #120
         #print(x,y)
         # height and width of sunglasses
         # h = length of nose
         h = int(abs(key1[27,1] - key1[34,1])) #55
         # w = left to right eyebrow edges
         w = int(abs(key1[17,0] - key1[26,0])) #135
         #print(x,y,h,w)
         # read in sunglasses
         sunglasses = cv2.imread('images/sunglasses.png', cv2.IMREAD_UNCHANGED)
         # resize sunglasses
         new_sunglasses =  cv2.resize(sunglasses, (w, h), interpolation = cv2.INTER_CUBIC)
         #roi_color = cv2.cvtColor(image_copy, cv2.COLOR_RGB2GRAY)
         # get region of interest on the face to change
```

```
        roi_color = image[y:y+h,x:x+w]
        #print(roi_color)
        # find all non-transparent pts
        #roi_color=cv2.resize(roi_color, (224, 224))
        #roi_color = np.reshape(roi_color, (1, 1, 224, 224))
        ind = np.argwhere(new_sunglasses[:,:,3] > 0)
        #print(ind.shape)
        # for each non-transparent point, replace the original image pixel with that of the neu
        for i in range(3):
            roi_color[ind[:,0],ind[:,1],i] = new_sunglasses[ind[:,0],ind[:,1],i]
        # set the area of the image to the changed region with sunglasses
        image[y:y+h,x:x+w] = roi_color


        # display the result!
        plt.imshow(image)
```

```
Out[17]: <matplotlib.image.AxesImage at 0x7f34847b4ba8>
```



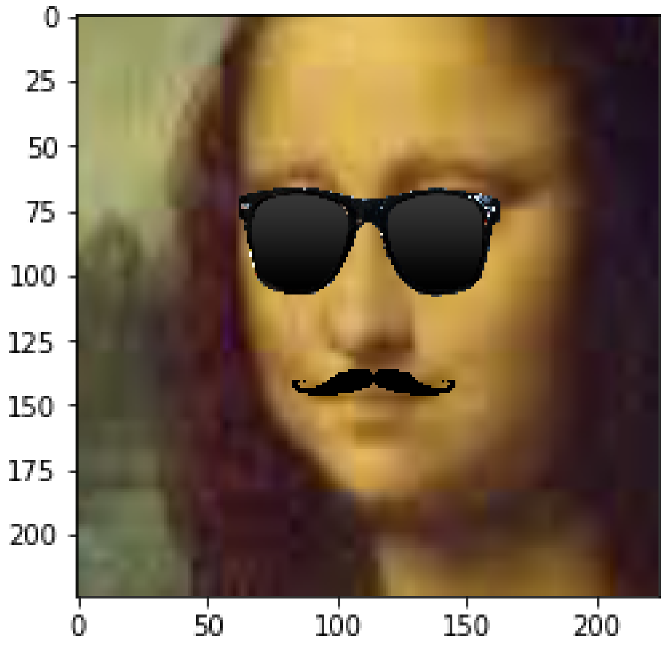
```
In [397]: import matplotlib.image as mpimg
          import matplotlib.pyplot as plt
          import numpy as np
          import pandas as pd
          import os
          import cv2
          moustache = cv2.imread('images/moustache.png', cv2.IMREAD_UNCHANGED)
```

```
# plot our image


# print out its dimensions
print('Image shape: ', moustache.shape)
alpha_channel = moustache[:,:,3]
print ('The alpha channel looks like this (black pixels = transparent): ')
plt.imshow(alpha_channel, cmap='gray')
values = np.where(alpha_channel != 0)
print ('The non-zero values of the alpha channel are: ')
print (values)
plt.imshow(moustache)

import numpy as np

image_copy = np.copy(image)
#key=key.numpy()
#print(key)
# top-left location for sunglasses to go
# 17 = edge of left eyebrow
x = int(key1[6, 0]) #205
y = int(key1[3, 1]) #120
#print(x,y)
# height and width of sunglasses
# h = length of nose
h = int(abs(key1[34,1] - key1[63,1])) #55
# w = left to right eyebrow edges
w = int(abs(key1[37,0] - key1[46,0])) #135
#print(x,y,h,w)
# read in sunglasses
moustache = cv2.imread('images/moustache.png', cv2.IMREAD_UNCHANGED)
# resize sunglasses
new_moustache =  cv2.resize(moustache, (w, h), interpolation = cv2.INTER_CUBIC)
#roi_color = cv2.cvtColor(image_copy, cv2.COLOR_RGB2GRAY)
# get region of interest on the face to change
roi_color = image[y:y+h,x:x+w]
#print(roi_color)
# find all non-transparent pts
#roi_color=cv2.resize(roi_color, (224, 224))
#roi_color = np.reshape(roi_color, (1, 1, 224, 224))
ind = np.argwhere(new_moustache[:,:,3] > 0)
#print(ind.shape)
# for each non-transparent point, replace the original image pixel with that of the ne
for i in range(3):
    roi_color[ind[:,0],ind[:,1],i] = new_moustache[ind[:,0],ind[:,1],i]
# set the area of the image to the changed region with sunglasses
image[y:y+h,x:x+w] = roi_color
```

```
# display the result!
plt.imshow(image)
```

```
Image shape:  (480, 960, 4)
The alpha channel looks like this (black pixels = transparent):
The non-zero values of the alpha channel are:
(array([105, 105, 105, ..., 374, 374, 374]), array([384, 385, 386, ..., 800, 801, 802]))
```

```
Out[397]: <matplotlib.image.AxesImage at 0x7ffa954bf0b8>
```



```
In [ ]:
```

```
In [ ]:
```