4. Fun with Keypoints

May 5, 2024

0.1 Facial Filters

Using your trained facial keypoint detector, you can now do things like add filters to a person's face, automatically. In this optional notebook, you can play around with adding sunglasses to detected face's in an image by using the keypoints detected around a person's eyes. Check out the `images/` directory to see what other .png's have been provided for you to try, too!

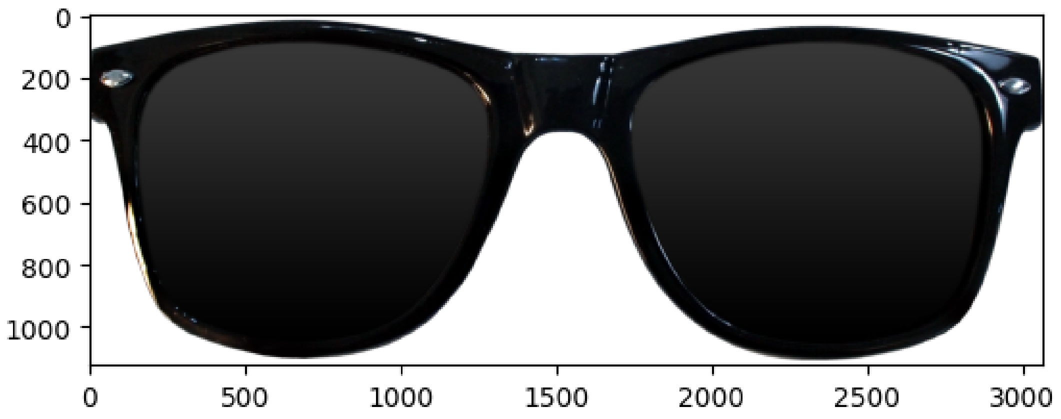
Let's start this process by looking at a sunglasses .png that we'll be working with!

[1]:
```
# import necessary resources
import matplotlib.image as mpimg
import matplotlib.pyplot as plt
import numpy as np
import pandas as pd
import os
import cv2
```

[2]:
```
# load in sunglasses image with cv2 and IMREAD_UNCHANGED
sunglasses = cv2.imread('images/sunglasses.png', cv2.IMREAD_UNCHANGED)

# plot our image
plt.imshow(sunglasses)

# print out its dimensions
print('Image shape: ', sunglasses.shape)
```

```
/usr/local/lib/python3.10/dist-packages/numpy/core/getlimits.py:518:
UserWarning: The value of the smallest subnormal for <class 'numpy.float64'>
type is zero.
  setattr(self, word, getattr(machar, word).flat[0])
/usr/local/lib/python3.10/dist-packages/numpy/core/getlimits.py:89: UserWarning:
The value of the smallest subnormal for <class 'numpy.float64'> type is zero.
  return self._float_to_str(self.smallest_subnormal)
```

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



0.2 The 4th dimension

You'll note that this image actually has *4 color channels*, not just 3 as your avg RGB image does. This is due to the flag we set `cv2.IMREAD_UNCHANGED`, which tells this to read in another color channel.

Alpha channel It has the usual red, blue, and green channels any color image has, and the 4th channel represents the **transparency level of each pixel** in the image; this is often called the **alpha** channel. Here's how the transparency channel works: the lower the value, the more transparent, or see-through, the pixel will become. The lower bound (completely transparent) is zero here, so any pixels set to 0 will not be seen; these look like white background pixels in the image above, but they are actually totally transparent.

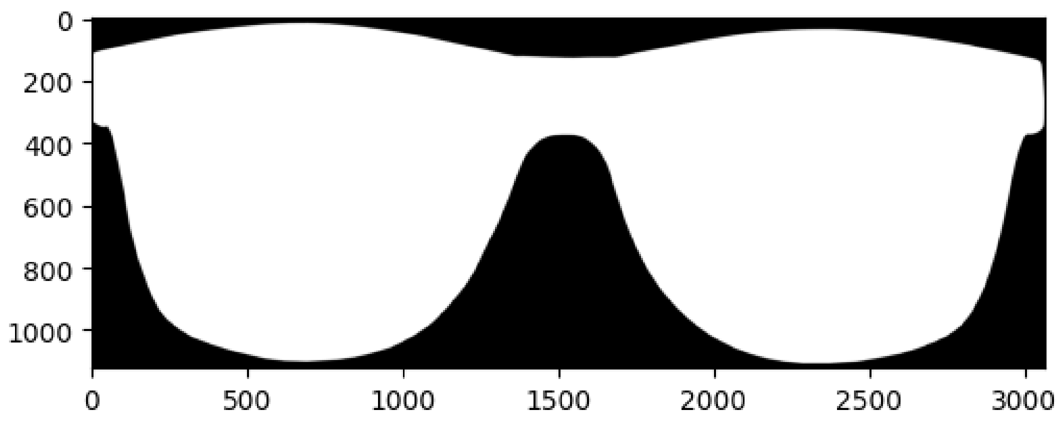
This transparent channel allows us to place this rectangular image of sunglasses on an image of a face and still see the face area that is technically covered by the transparent background of the sunglasses image!

Let's check out the alpha channel of our sunglasses image in the next Python cell. Because many of the pixels in the background of the image have an alpha value of 0, we'll need to explicitly print out non-zero values if we want to see them.

[3]:
```
# print out the sunglasses transparency (alpha) channel
alpha_channel = sunglasses[:,:,3]
print ('The alpha channel looks like this (black pixels = transparent): ')
plt.imshow(alpha_channel, cmap='gray')
```

```
The alpha channel looks like this (black pixels = transparent):
```

[3]:
```
<matplotlib.image.AxesImage at 0x7f65f7d33010>
```



[4]:
```
# just to double check that there are indeed non-zero values
# let's find and print out every value greater than zero
values = np.where(alpha_channel != 0)
print ('The non-zero values of the alpha channel are: ')
print (values)
```

```
The non-zero values of the alpha channel are:
(array([  17,   17,   17, …, 1109, 1109, 1109]), array([ 687,  688,  689, …,
2376, 2377, 2378]))
```

Overlaying images This means that when we place this sunglasses image on top of another image, we can use the transparency channel as a filter:

- If the pixels are non-transparent (alpha_channel > 0), overlay them on the new image

Keypoint locations In doing this, it's helpful to understand which keypoint belongs to the eyes, mouth, etc., so in the image below we also print the index of each facial keypoint directly on the image so you can tell which keypoints are for the eyes, eyebrows, etc.,

It may be useful to use keypoints that correspond to the edges of the face to define the width of the sunglasses, and the locations of the eyes to define the placement.

Next, we'll load in an example image. Below, you've been given an image and set of keypoints from the provided training set of data, but you can use your own CNN model to generate keypoints for *any* image of a face (as in Notebook 3) and go through the same overlay process!

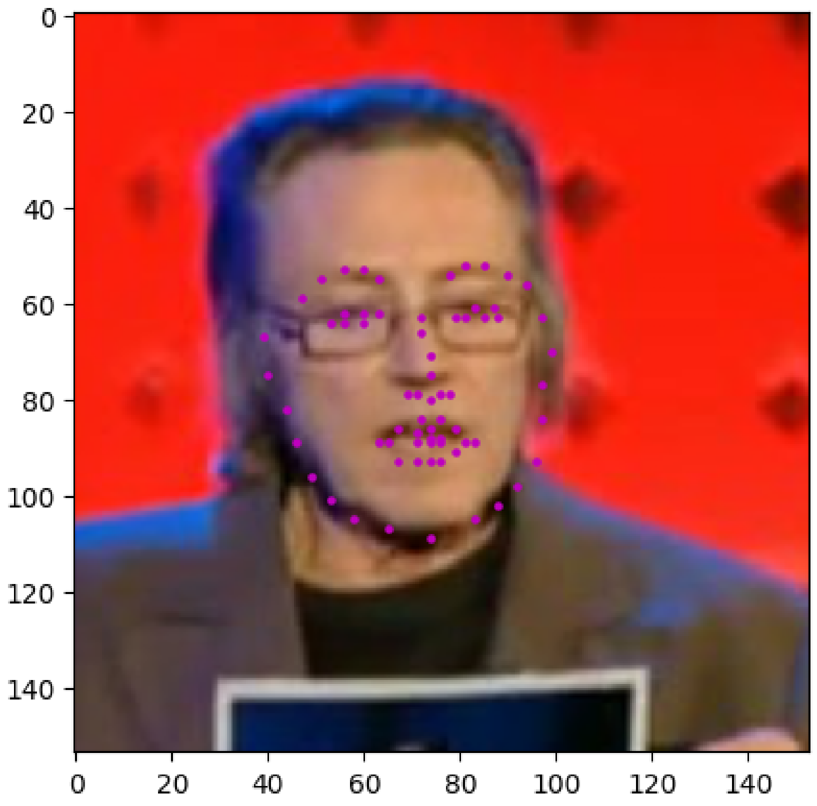
[5]:
```
# load in training data
key_pts_frame = pd.read_csv('data/training_frames_keypoints.csv')

# print out some stats about the data
print('Number of images: ', key_pts_frame.shape[0])
```

```
Number of images:  3462
```

[6]:
```
# helper function to display keypoints
def show_keypoints(image, key_pts):
    """Show image with keypoints"""
    plt.imshow(image)
    plt.scatter(key_pts[:, 0], key_pts[:, 1], s=20, marker='.', c='m')
```

[7]:
```
# a selected image
n = 120
image_name = key_pts_frame.iloc[n, 0]
image = mpimg.imread(os.path.join('data/training/', image_name))
key_pts = key_pts_frame.iloc[n, 1:].values
key_pts = key_pts.astype('float').reshape(-1, 2)

print('Image name: ', image_name)

plt.figure(figsize=(5, 5))
show_keypoints(image, key_pts)
plt.show()
```

```
Image name:  Christopher_Walken_01.jpg
```



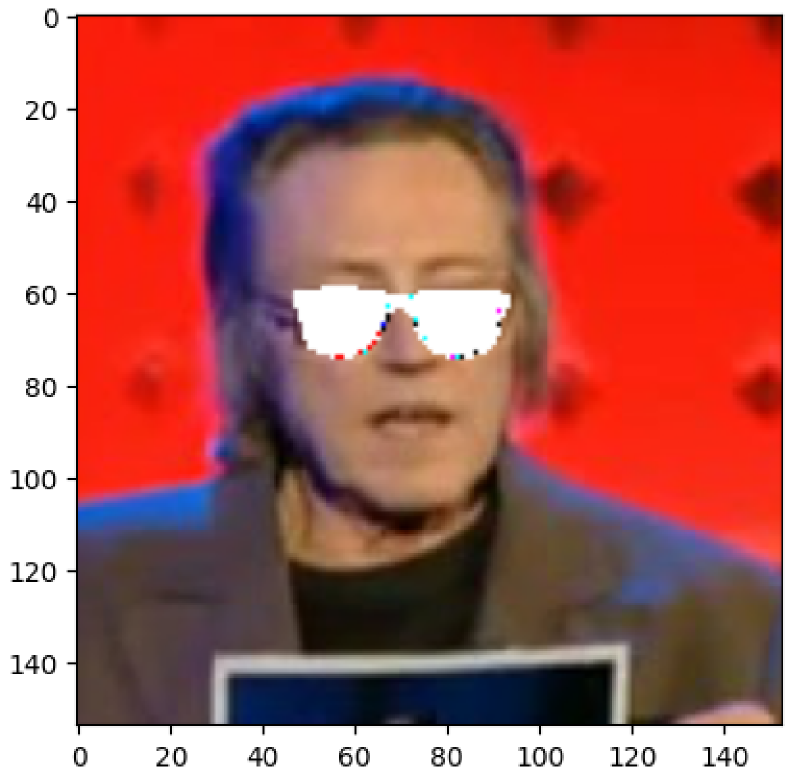
Next, you'll see an example of placing sunglasses on the person in the loaded image.

Note that the keypoints are numbered off-by-one in the numbered image above, and so `key_pts[0,:]` corresponds to the first point (1) in the labeled image.

```
[8]: # Display sunglasses on top of the image in the appropriate place

     # copy of the face image for overlay
     image_copy = np.copy(image)

     # top-left location for sunglasses to go
     # 17 = edge of left eyebrow
     x = int(key_pts[17, 0])
     y = int(key_pts[17, 1])

     # height and width of sunglasses
     # h = length of nose
     h = int(abs(key_pts[27,1] - key_pts[34,1]))
     # w = left to right eyebrow edges
     w = int(abs(key_pts[17,0] - key_pts[26,0]))

     # 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)

     # get region of interest on the face to change
     roi_color = image_copy[y:y+h,x:x+w]

     # find all non-transparent pts
     ind = np.argwhere(new_sunglasses[:,:,3] > 0)

     # for each non-transparent point, replace the original image pixel with that of␣
      ↪the new_sunglasses
     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_copy[y:y+h,x:x+w] = roi_color


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

```
Clipping input data to the valid range for imshow with RGB data ([0..1] for
floats or [0..255] for integers).
```

```
[8]: <matplotlib.image.AxesImage at 0x7f65f72da2c0>
```



Further steps Look in the `images/` directory to see other available .png's for overlay! Also, you may notice that the overlay of the sunglasses is not entirely perfect; you're encouraged to play around with the scale of the width and height of the glasses and investigate how to perform image rotation in OpenCV so as to match an overlay with any facial pose.

```
[ ]:
```