hw4

May 5, 2021

1 CSCI-UA 0480-042 Computer Vision

1.1 Homework 4

Enter your name and NetID below.

Name: Yi Yang

NetID:yy2324

The main goals of this assignment include:

- Give an introduction to OpenPose.
- Do body and hand pose estimation using a pretrained OpenPose model.

1.1.1 Part 1: Introduction to OpenPose

OpenPose is a real-time multi-person 2D pose detection system. It jointly detects human body, hand, facial, and foot keypoints (in total 135 keypoints) on single images.

GitHub Repo: https://github.com/CMU-Perceptual-Computing-Lab/openpose

Paper: https://arxiv.org/pdf/1812.08008.pdf

The overall pipeline of the detection system is shown in the figure below.

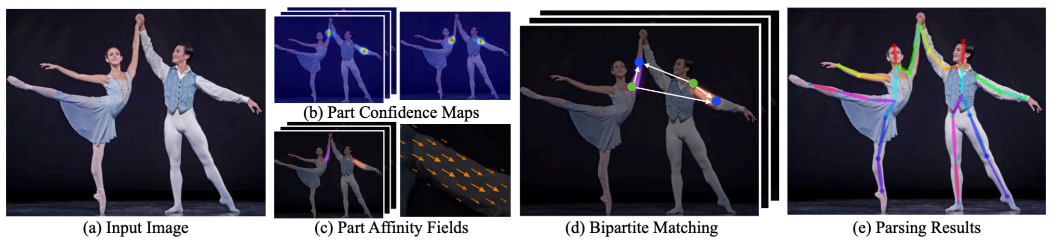


Fig. 2: Overall pipeline. (a) Our method takes the entire image as the input for a CNN to jointly predict (b) confidence maps for body part detection and (c) PAFs for part association. (d) The parsing step performs a set of bipartite matchings to associate body part candidates. (e) We finally assemble them into full body poses for all people in the image.

Given an input image, the model jointly predicts:

(1) **Confidence Maps:** A set of 2d heatmaps of body part locations. Each map is a 2D representation of the belief that a particular body part (neck, shoulder, elbow, etc.) can be located in any given pixel.

(2) **Part Affinity Fields (PAFs):** A set of 2d vector fields that encode body part associations. A pair of body parts form a limb (elbow-shoulder connection). And each PAF is a 2D vector

field that encodes the degree of association as well as the direction from one limb part to another.

In the bipartite matching step, body part candidates are matched to form limbs. Each body part could have multiple candidates (multiple people in the image => elbows at multiple locations). The candidate parts need to be associated with other parts from the same person. All candidate limbs are scored using PAFs, and the optimal association for any given pair of body parts is treated as bipartite graph matching problem to generate limb pairs in the entire image. (See figure below)

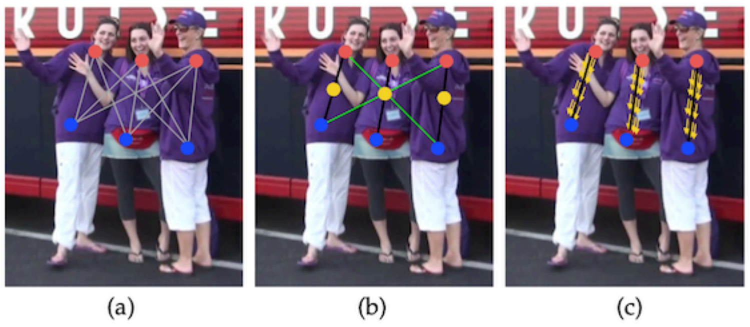


Fig. 5: Part association strategies. (a) The body part detection candidates (red and blue dots) for two body part types and all connection candidates (grey lines). (b) The connection results using the midpoint (yellow dots) representation: correct connections (black lines) and incorrect connections (green lines) that also satisfy the incidence constraint. (c) The results using PAFs (yellow arrows). By encoding position and orientation over the support of the limb, PAFs eliminate false associations.

Once we obtain all the limb candidates, all limb connections that share the same part detection candidates can be assembled into fully body poses of multiple people.

For each question in part 1 (q1-5), please make sure your answer is at most 3-4 lines.

Question 1 Briefly explain the difference between top-down and bottom-up approaches in pose estimation. Is OpenPose bottom-up or top-down?

Hint: See page1 of the paper.

Answer: Top-down approach is very common, such as employing a person detector and perform single-person pose estimation for each detection. Top down approaches can directly leverage existing techniques for single-person pose estimation. However, if the person detector fails, there is no recourse to recovery. Bottom-up approaches offer “robustness to early commitment and have the potential to decouple runtime complexity from the number of people in the image”. They do not use global cues from other parts. OpenPose is bottom-up.

Question 2 Which pretrained network is used in the OpenPose paper to generate features from the image?

Answer: VGG-19.

Question 3 In multiple pose detection, people often crowd together and this can be problematic because it may result in false associations. In the paper they present a naive solution for finding body part associations where this could occur, and they also briefly explain how PAFs address this problem. Describe briefly what the naive solution is and how PAFs avoid this.

Hint: Read Section 3.4 in the paper.

Answer: The naive solution is to detect an additional midpoint between each pair of parts on a limb and check for its incidence between candidate part detections. PAFs preserve both location and orientation information across the region of support of the limb, thus avoiding the problem that may occue with the naive solution.

Question 4 OpenPose network has 2 branches - PAF and confidence map branch. During training, which branch is refined first - PAF or confidence map? What's the intuition behind this?

Hint: See Section 3.2 in the paper.

Answer: PAF is refined first. The intuition behind this is that this way cuts the amount of computations in half. As the author mentioned, "if we look at the PAF channel output, the body part locations can be guessed. However, if we see a bunch of body parts with no other information, we cannot parse them into different people".

Question 5 The loss functions used in training use a binary mask for each pixel. Why is this necessary?

Hint: See Section 3.2 in the paper.

Answer: It is necessary so that it avoids penalizing the true positive predictions during training.

1.1.2 Part 2: Body Pose Estimation

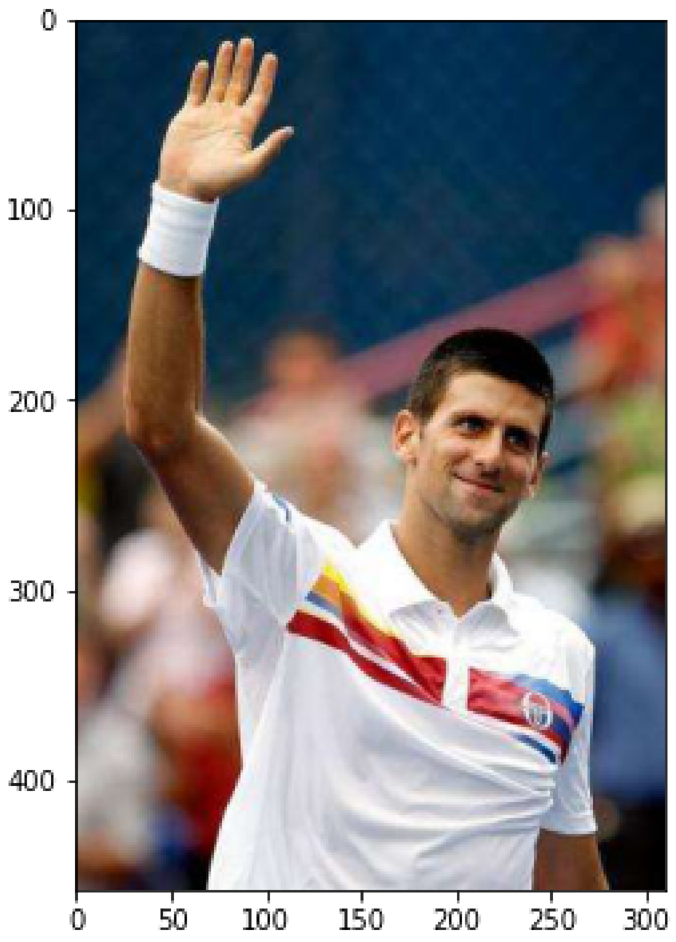
In this part, we'll try to obtain the body keypoints using a pretrained OpenPose model. There's a PyTorch version of OpenPose model which we will be using both in this part and in part 3.

[1]:
```
import cv2
import matplotlib.pyplot as plt

from PIL import Image
from src import util
from src.body import Body, visualize_heatmap, visualize_paf
```

Let's see the image that was picked for pose detection.

[2]:
```
image_path = '../shared/openpose-samples/demo.jpg'

plt.figure(figsize=(6, 6))
plt.imshow(Image.open(image_path))
```

[2]:
```
<matplotlib.image.AxesImage at 0x7fe381619250>
```

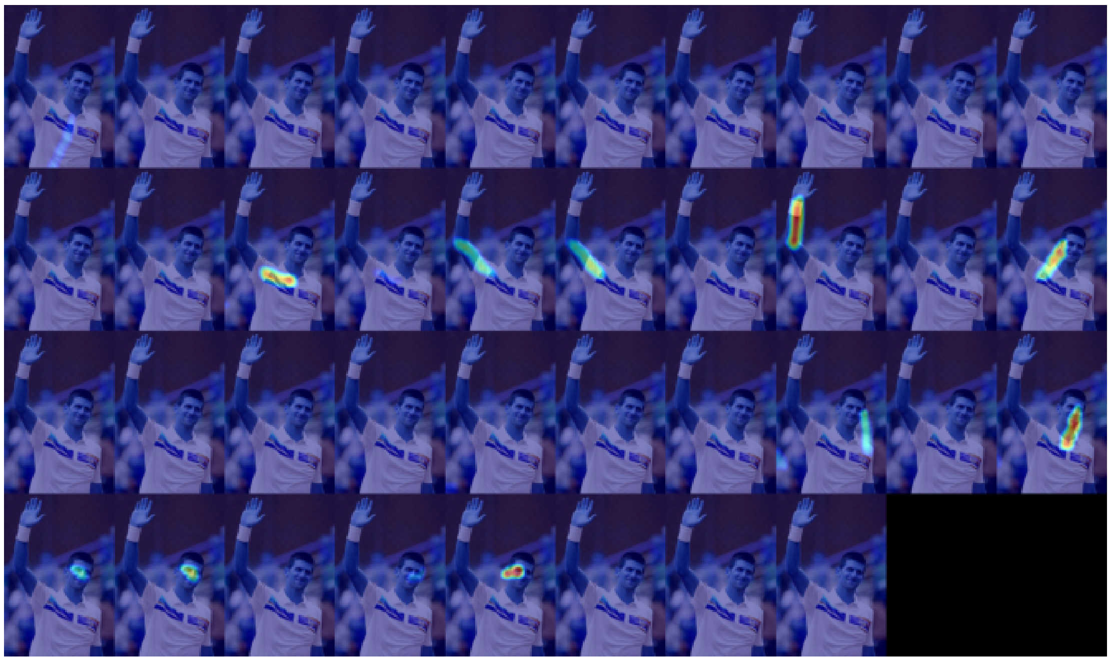


Using the pretrained model, let's generate the PAFs and the confidence maps for this image.

[3]:
```
image = cv2.imread(image_path)
body_estimation = Body('../shared/openpose-models/body_pose_model.pth')
heatmap_avg, paf_avg = body_estimation.get_heatmap_paf(image)
```

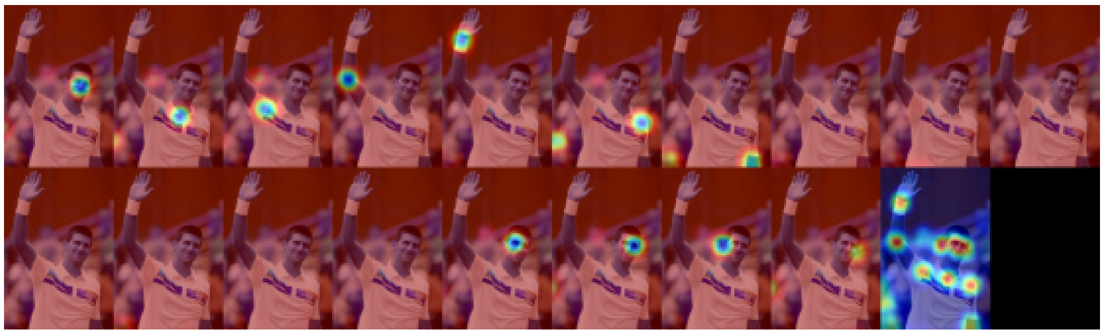
Question 6 Can you show the PAFs? In the resulting plot, you'll see 38 subplots (19 pairs). In each pair, the first and second plots are heatmaps representing the x and y dimensions of the PAF vector field respectively.

[4]:
```
visualize_paf(image_path, paf_avg)
```



Question 7 Can you show the heatmap?

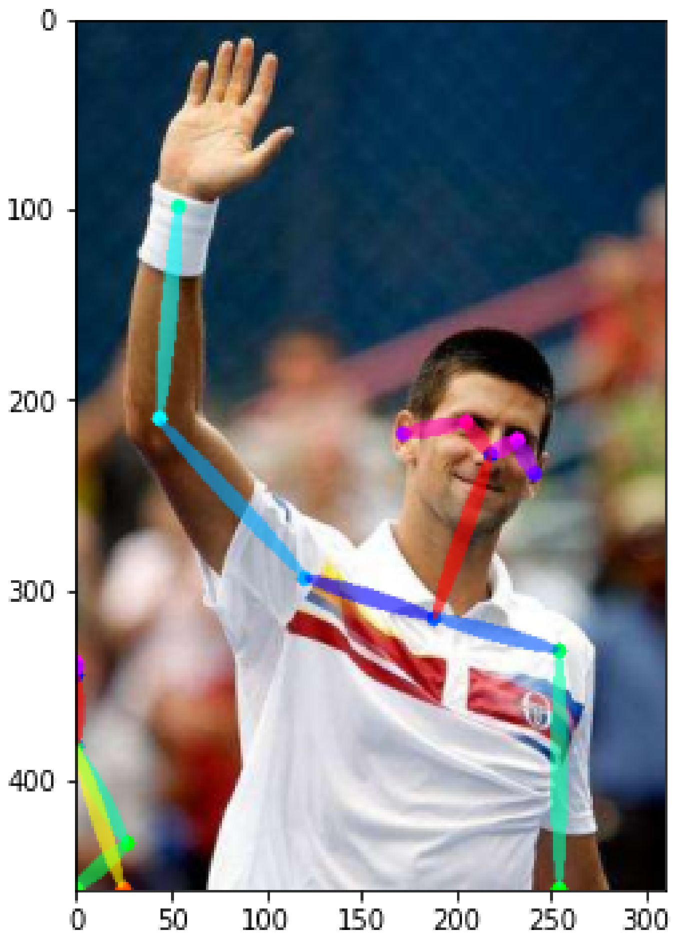
[5]:
```
visualize_heatmap(image_path, heatmap_avg)
```



Question 8 Using the PAFs and the heatmap, generate the parsing results and show the output.

[6]:
```
candidate, subset = body_estimation.get_parsing_results(heatmap_avg, paf_avg)

canvas = util.draw_bodypose(image, candidate, subset)

fig = plt.figure(figsize=(6, 6))
plt.imshow(canvas[:, :, [2, 1, 0]])
```

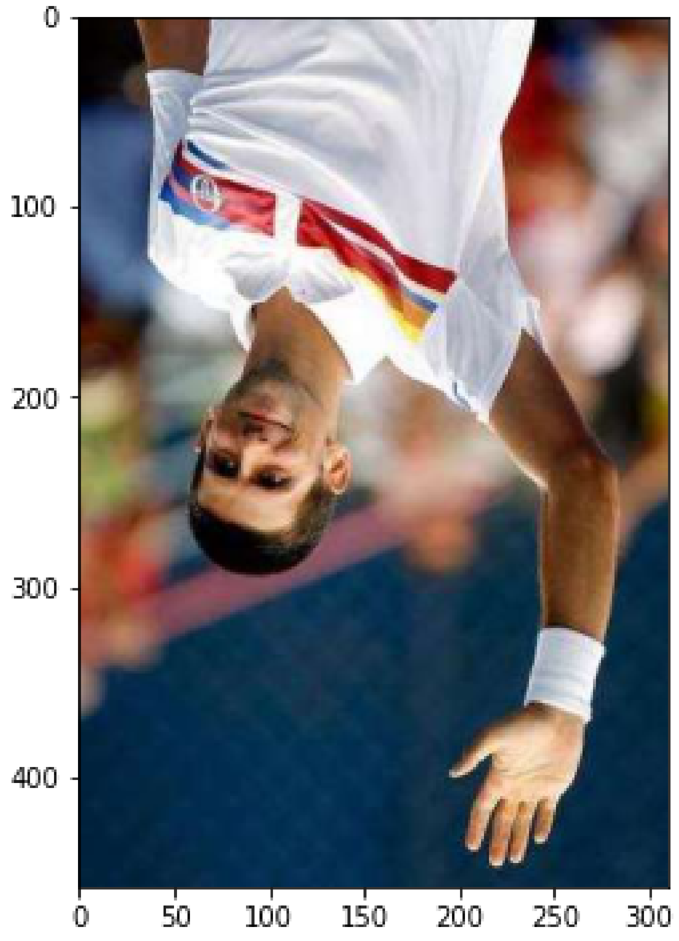
```
[6]: <matplotlib.image.AxesImage at 0x7fe369405c10>
```



Let's now check if the model works on a 180-degree rotated image.

```
[7]: image_path = '../shared/openpose-samples/demo_180.jpg'

     plt.figure(figsize=(6, 6))
     plt.imshow(Image.open(image_path))
```

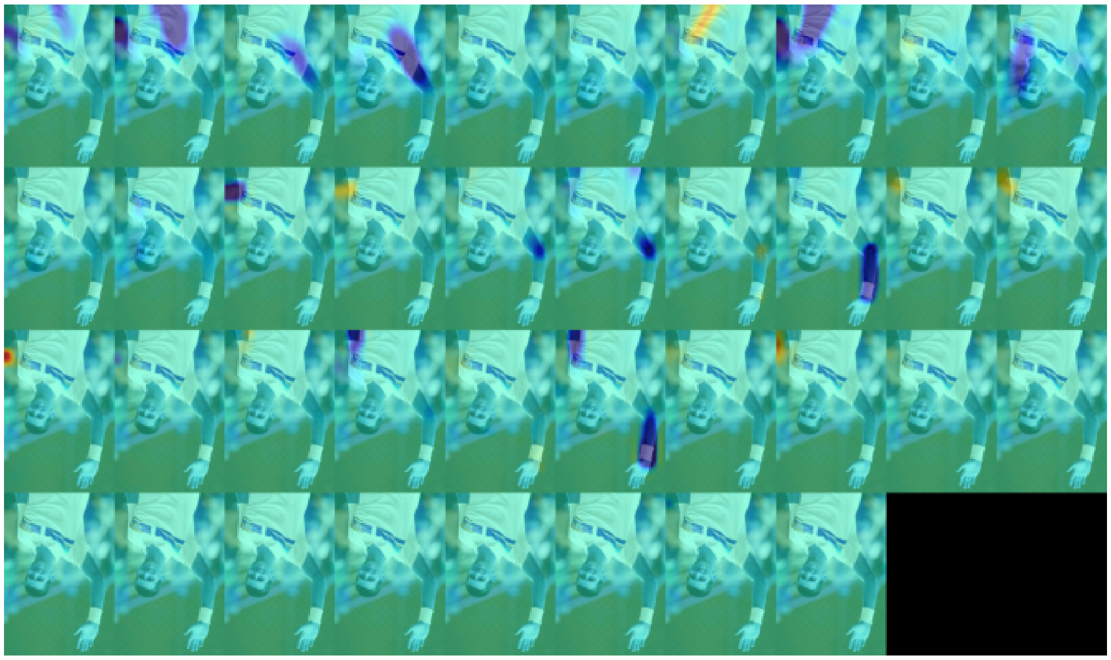
```
[7]: <matplotlib.image.AxesImage at 0x7fe36937fad0>
```



```
[8]: image = cv2.imread(image_path)
     heatmap_avg, paf_avg = body_estimation.get_heatmap_paf(image)
```

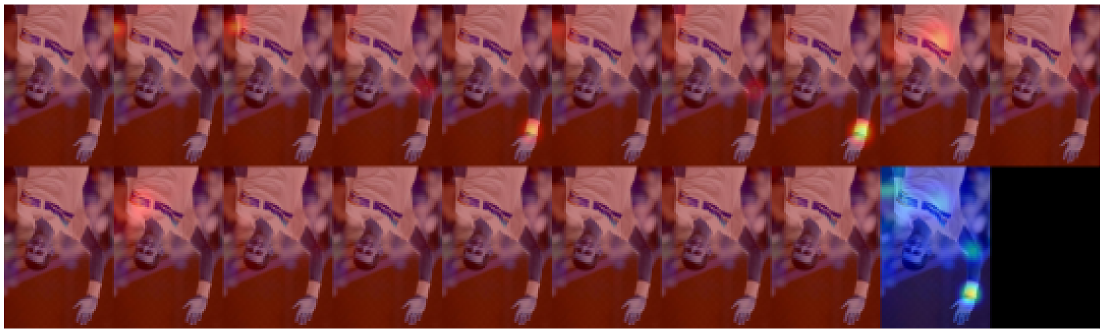
Question 9 Visualize the Part Affinity Fields.

```
[9]: visualize_paf(image_path, paf_avg)
```



Question 10 Show the heatmap.

[10]:
```
visualize_heatmap(image_path, heatmap_avg)
```

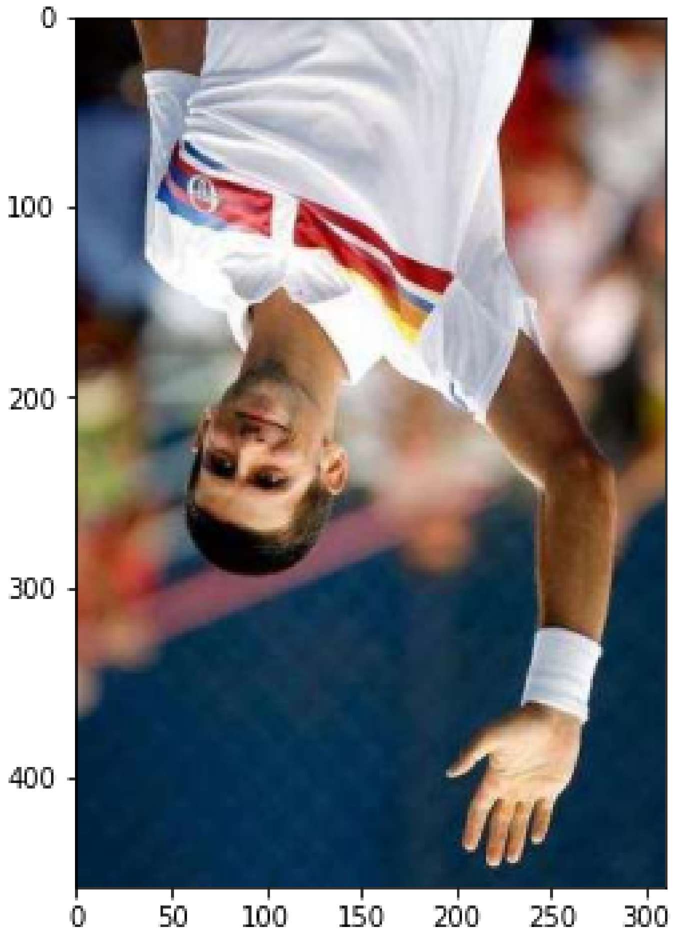


Question 11 Using the PAFs and the confidence maps, generate the parsing results and visualize it.

[11]:
```
candidate, subset = body_estimation.get_parsing_results(heatmap_avg, paf_avg)

canvas = util.draw_bodypose(image, candidate, subset)

fig = plt.figure(figsize=(6, 6))
```

```
plt.imshow(canvas[:, :, [2, 1, 0]])
```

[11]:
```
<matplotlib.image.AxesImage at 0x7fe36db4aed0>
```



Question 12 Does the model work on the rotated image? If not, at which stage does it fail?

Answer: No. It fails at the PAF stage.

1.1.3 Part 3: Hand Pose Estimation

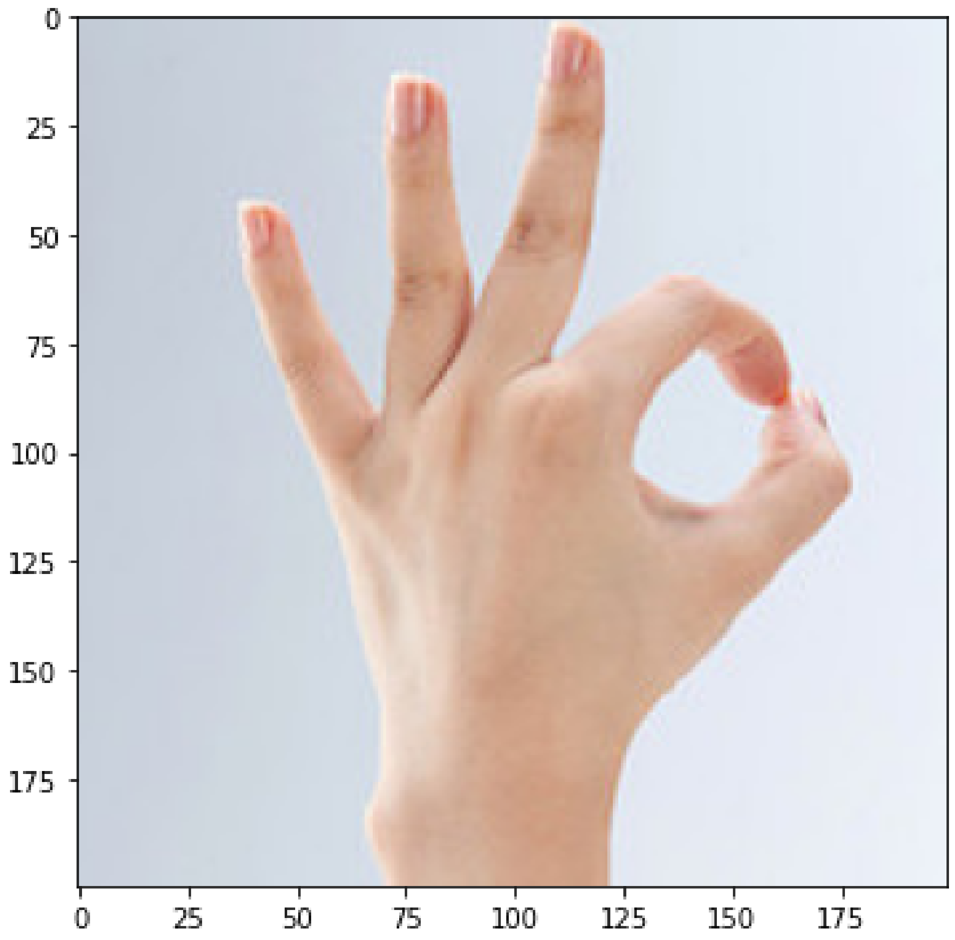
Given a hand image, let's try to estimate the hand pose.

[12]:
```
import cv2
import numpy as np
import matplotlib.pyplot as plt

from PIL import Image
from src import util
from src.hand import Hand, visualize_heatmap

%matplotlib inline
```

Here's the sample image we're going to use.

```
[13]: image_path = '../shared/openpose-samples/hand.jpg'

      plt.figure(figsize=(6, 6))
      plt.imshow(Image.open(image_path))
```

```
[13]: <matplotlib.image.AxesImage at 0x7fe361951c50>
```



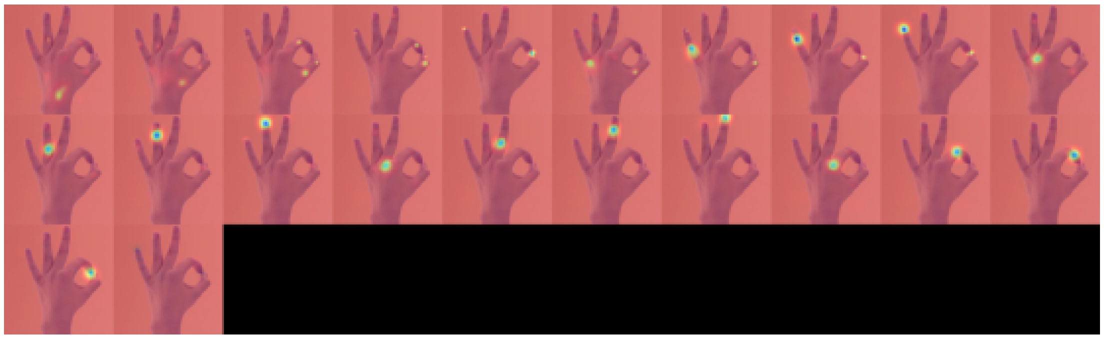
Let's generate the heatmap using a pretrained hand pose estimation model.

```
[16]: image = cv2.imread(image_path)

      hand_model = Hand('../shared/openpose-models/hand_pose_model.pth')
```

```
heatmap = hand_model.get_heatmap(image)
```

Question 13 Show the heatmap.

```
[15]: visualize_heatmap(image_path, heatmap)
```



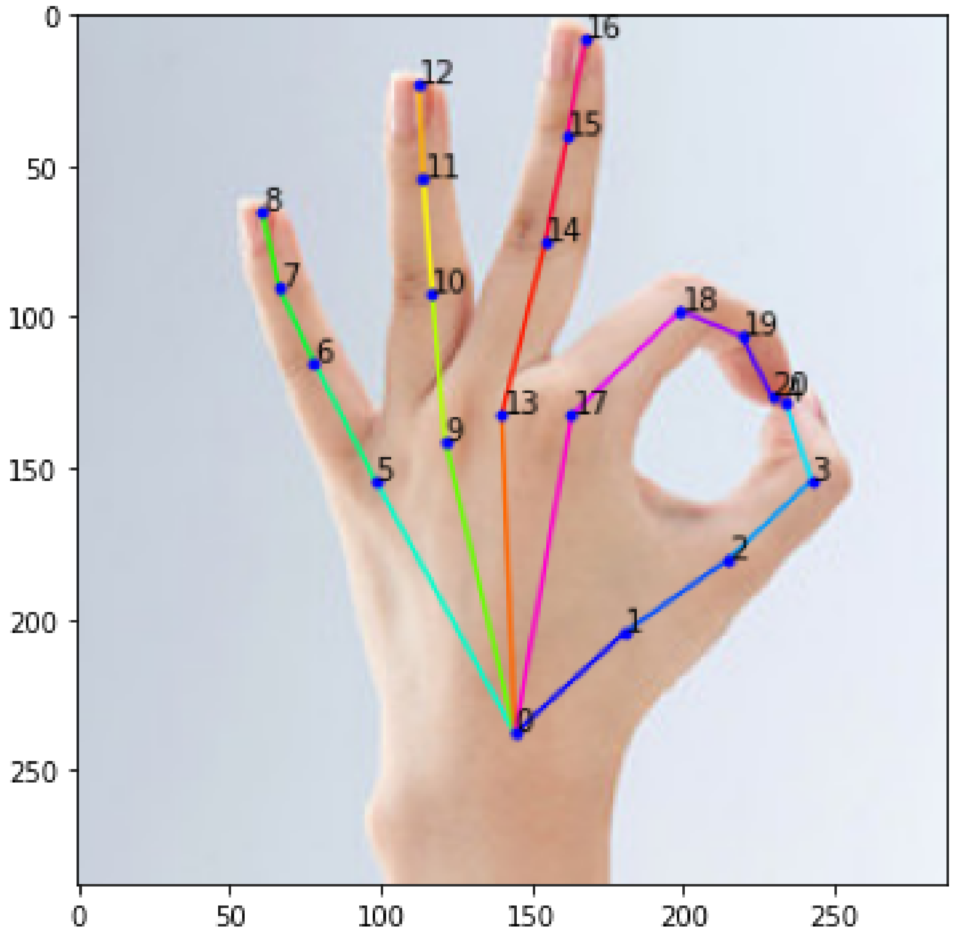
Question 14 Using the heatmap, generate the keypoints (peaks).

```
[17]: peaks = hand_model.get_peaks(heatmap)
```

Question 15 Show the final results.

```
[18]: canvas = util.draw_handpose(image, peaks, show_number=True)

      plt.figure(figsize=(6, 6))
      plt.imshow(canvas[:, :, [2, 1, 0]])
```

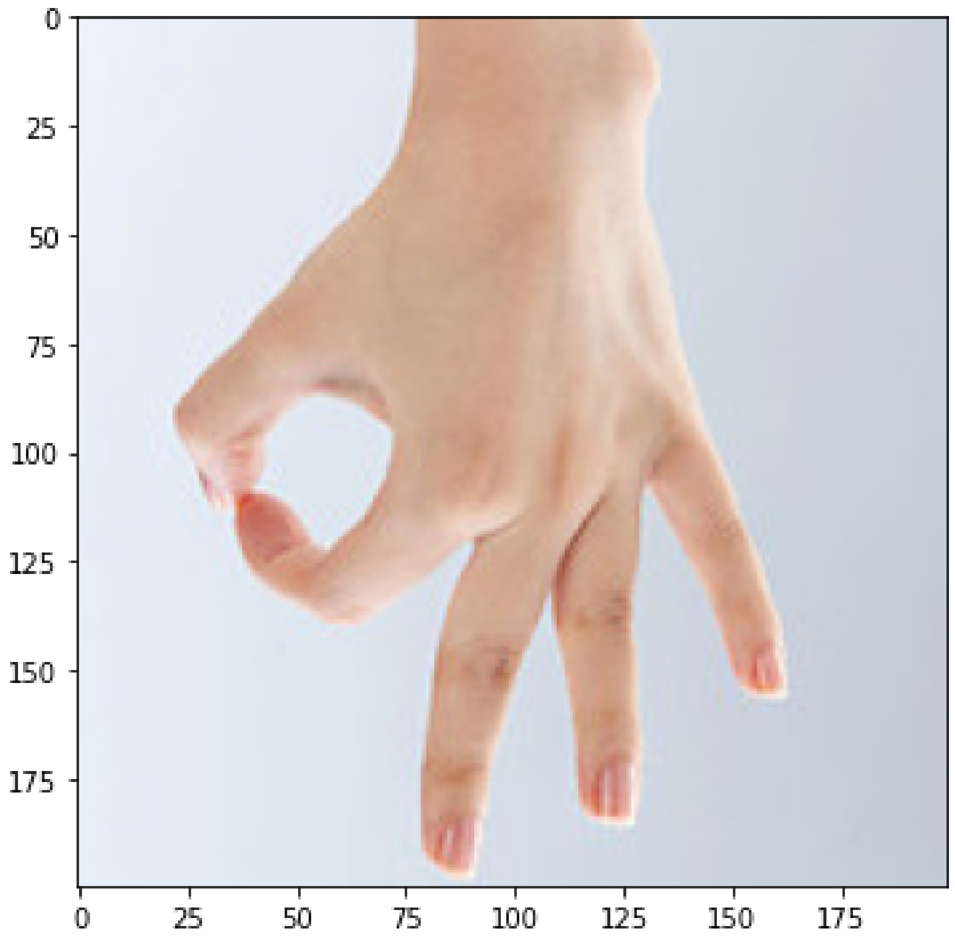
```
[18]: <matplotlib.image.AxesImage at 0x7fe360159e10>
```



Just like before, let's try the hand pose model on a flipped image.

```
[19]: image_path = '../shared/openpose-samples/hand_180.jpg'

      plt.figure(figsize=(6, 6))
      plt.imshow(Image.open(image_path))
```

```
[19]: <matplotlib.image.AxesImage at 0x7fe3600d3b50>
```

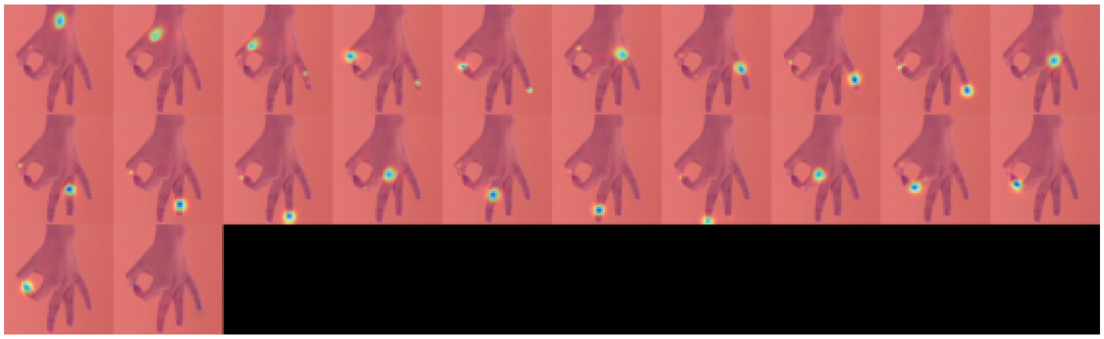


Now let's generate the heatmap by passing the image through the model.

[20]:
```
image = cv2.imread(image_path)

hand_model = Hand('../shared/openpose-models/hand_pose_model.pth')
heatmap = hand_model.get_heatmap(image)
```

Question 16 Show the heatmap.

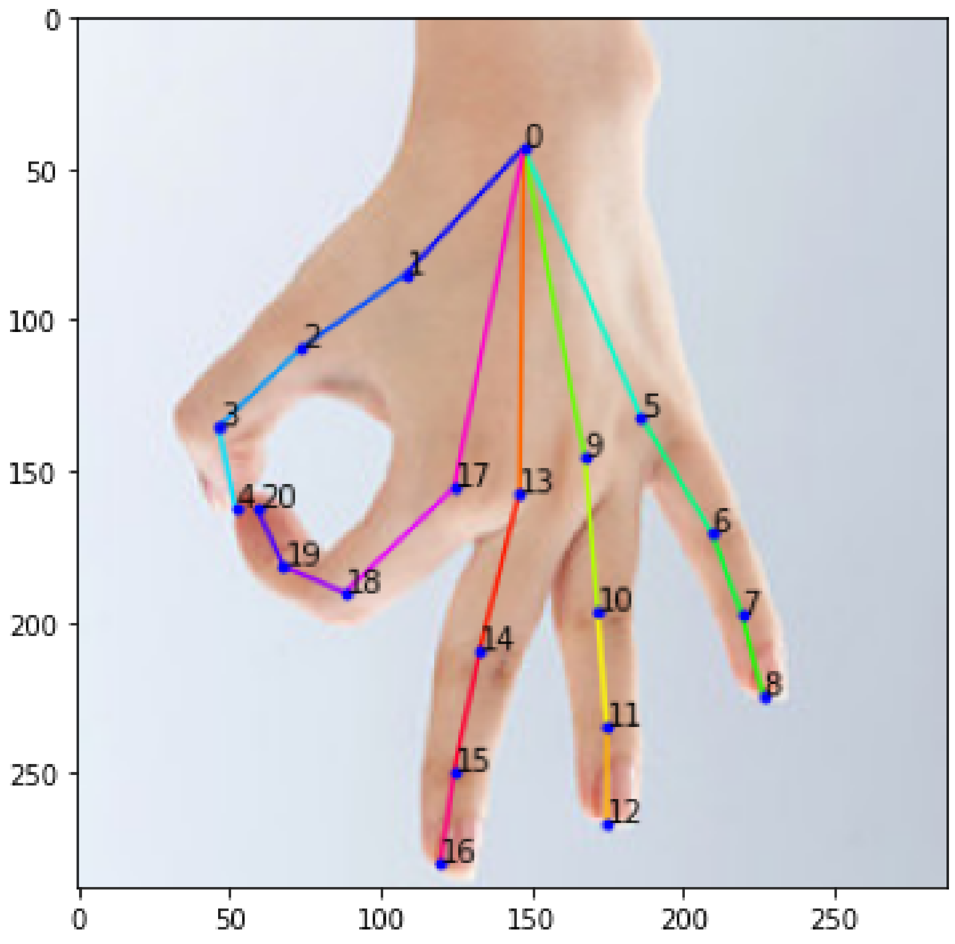
[21]:
```
visualize_heatmap(image_path, heatmap)
```



Question 17 Generate the keypoints and show the final output.

[22]:
```
peaks = hand_model.get_peaks(heatmap)

canvas = util.draw_handpose(image, peaks, show_number=True)

plt.figure(figsize=(6, 6))
plt.imshow(canvas[:, :, [2, 1, 0]])
```

[22]:
```
<matplotlib.image.AxesImage at 0x7fe35ff2d710>
```



Question 18 Does the handpose model work on the flipped image? If it doesn't, in which stage does it fail?

Answer: Yes, it does work.

```
[ ]:
```