hw4

April 28, 2022

1 CSGI-GA 3033-076 Vision meets ML

1.1 Homework 4

Enter your name and NetID below.

Name: Shuvadeep Saha

NetID: ss15592

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.

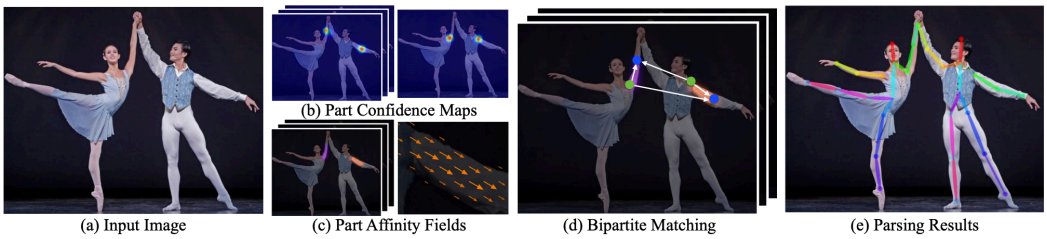


Fig. 2: Overall pipeline. (a) Our method takes the entire image as the input for a CNN to jointly predict (b) confidence maps for body part detection and (c) PAFs for part association. (d) The parsing step performs a set of bipartite matchings to associate body part candidates. (e) We finally assemble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

(a) (b) (c)

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.

Question 1 Briefly explain in 2-3 lines the difference between top-down and bottom-up approaches in pose estimation. Is open pose bottom-up or top-down?

Answer: In the top-down approach, each person is identified first and then the joints are estimated. Whereas, in bottom-up approach, all the joints are identified first followed by the person that belongs to.

The bottom-up approach is faster as compared to the top-down approach.

Open Pose is bottom-up approach.

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

Answer: VGG-19(the first 10 layers) pretrained network is used in the OpenPose paper to generate features from the image.

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.

Answer: To associate parts, we need a confidence measure of the association for each pair of body part detections, i.e., they should belong to the same person.

The Naive Solution measures association between parts by detecting an additional midpoint between each pair of parts on a limb and check for its incidence between candidate part detections. If the incidence constraint is satisfied the part connection is considered correct. A potential problem With Naive Solution is when people crowd together these mid points are likely to support false associations. Which may arise because of the following limitations of mid-point incidence : 1) Encodes only position(not orientation i.e. the direction of limb) 2) The region of a limb is reduced to a single point. PAFs avoid the problems in naive solution since, PAFs preserve both location and orientation information accross the region(not a single point) of support of the limb. PAFs are 2D vector fields for each limb, the 2D vector encodes the direction that points from one part of limb to the other. Each type of limb has a corresponding PAF joining its two associated body parts.

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?

Answer: At first, PAF is refined, followed by the confidence maps, Intuitively, 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?

Answer: The binary mask is used to avoid 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.

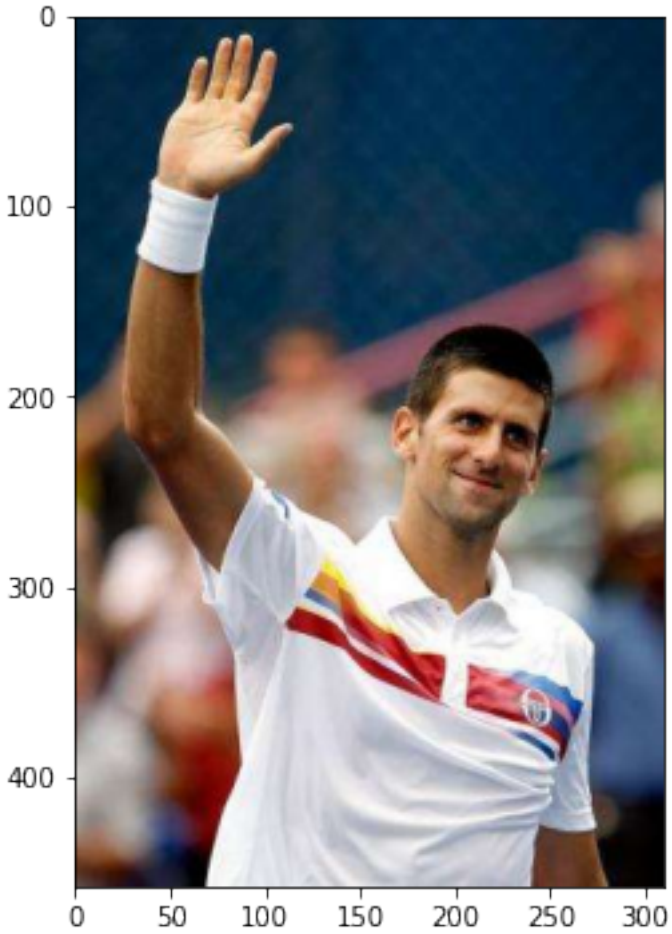
[1]:
```
import cv2
import matplotlib.pyplot as plt

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

%matplotlib inline
```

```
/opt/conda/envs/py37/lib/python3.7/site-packages/tqdm/auto.py:22: TqdmWarning:
IProgress not found. Please update jupyter and ipywidgets. See
https://ipywidgets.readthedocs.io/en/stable/user_install.html
  from .autonotebook import tqdm as notebook_tqdm
```

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

[2]:
```
image_path = 'images/demo.jpg'

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

[2]:
```
<matplotlib.image.AxesImage at 0x7fc4b7486f50>
```



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/model/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?

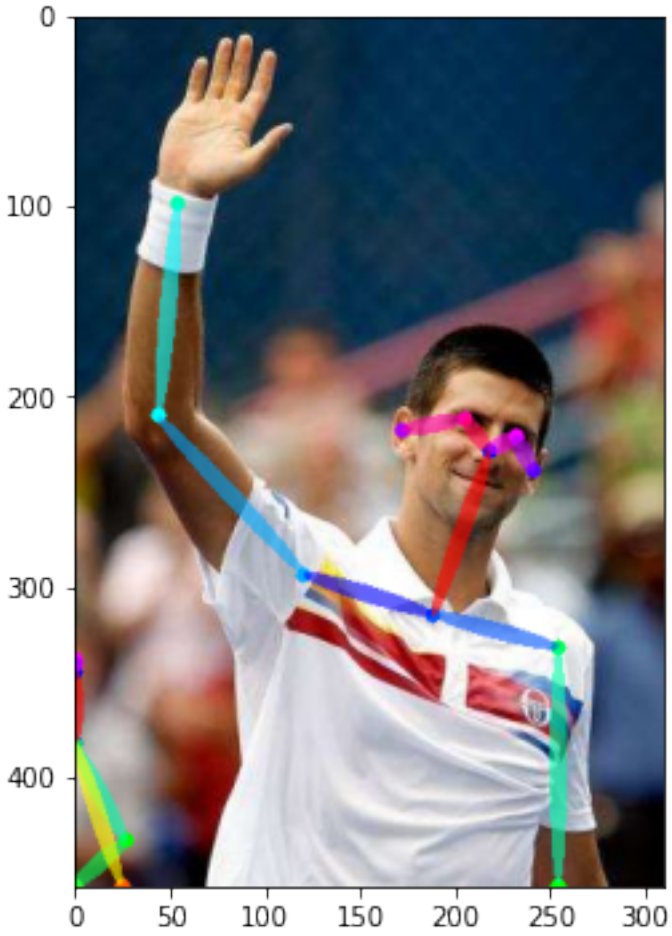
[5]:
```
visualize_heatmap(image_path, heatmap_avg)
```

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]])
```

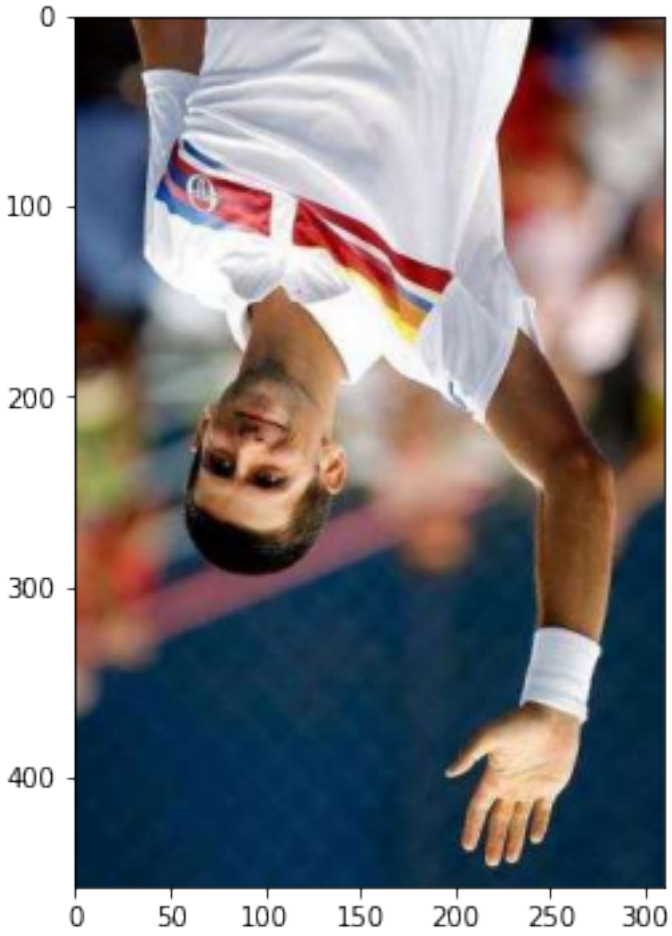
[6]: <matplotlib.image.AxesImage at 0x7fc49f2719d0>



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

[7]:
```
image_path = 'images/demo_180.jpg'

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

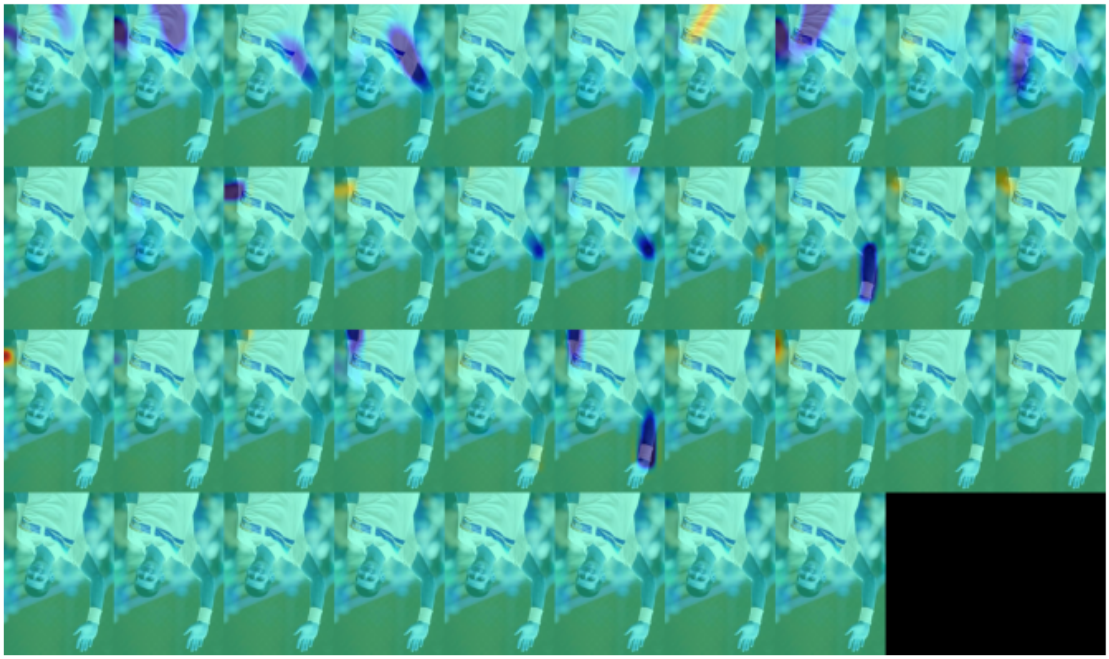
[7]: <matplotlib.image.AxesImage at 0x7fc49f1f7a90>



[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.

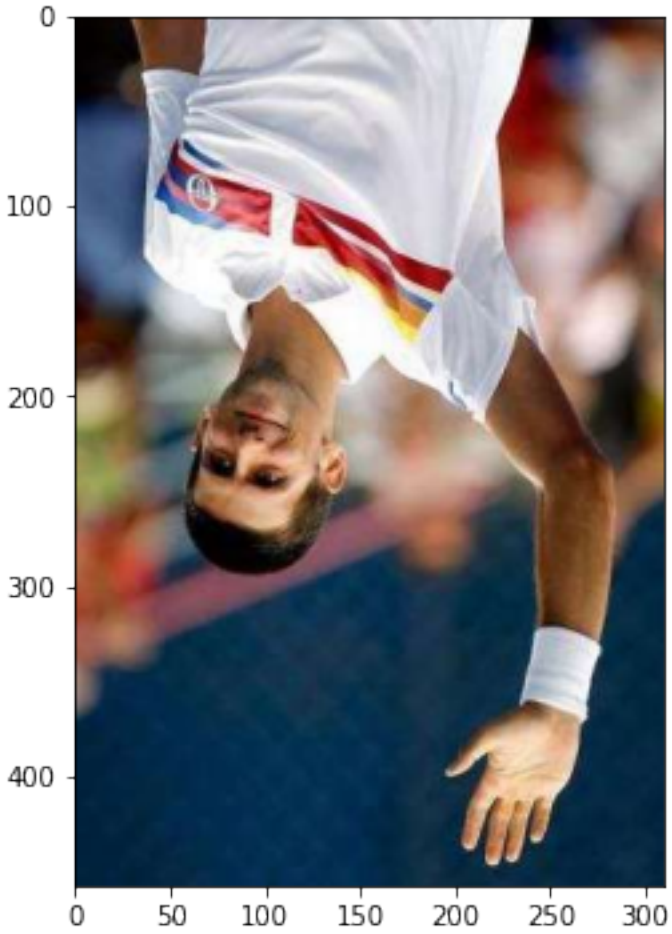
[10]:
```
visualize_heatmap(image_path, heatmap_avg)
```

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 0x7fc49f2d3c10>



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

Answer: The PAFs and the heatmap for the joints generated for the rotated image have incorrect predictions. This failure primarily occurs at the PAF generation stage itself.

1.1.3 Part 3: Hand Pose Estimation

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

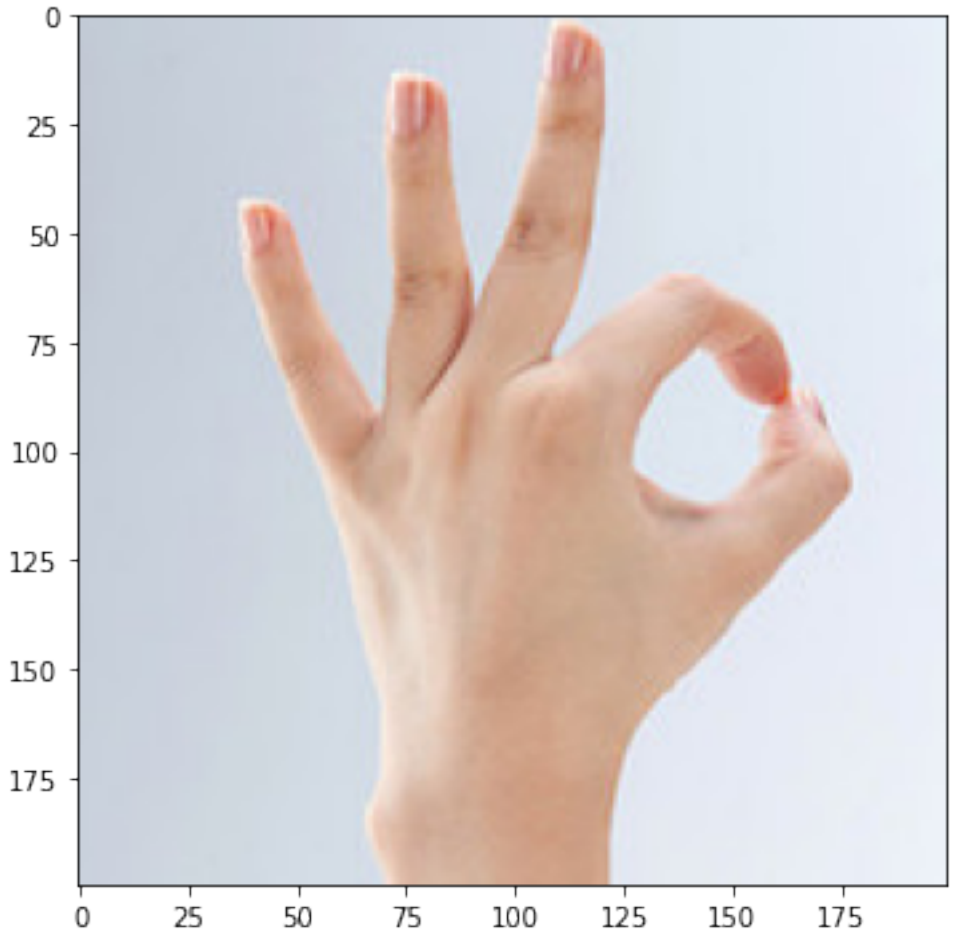
[12]:
```
import cv2
import numpy as np
import matplotlib.pyplot as plt

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 = 'images/hand.jpg'

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

[13]:
```
<matplotlib.image.AxesImage at 0x7fc49681eb10>
```



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

[14]:
```
image = cv2.imread(image_path)  # B,G,R order
```

```
hand_model = Hand('../shared/model/hand_pose_model.pth')
heatmap = hand_model.get_heatmap(image)
```

Question 13 Show the heatmap.

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

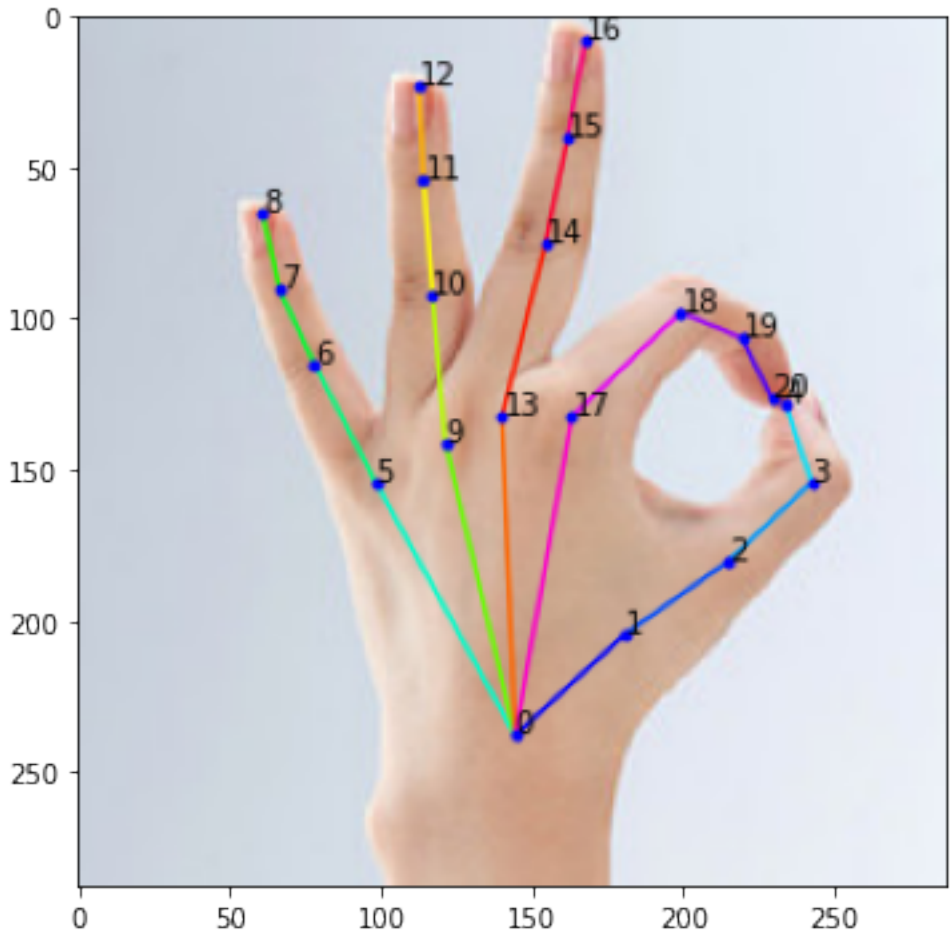
Question 14 Using the heatmap, generate the keypoints (peaks).

[16]:
```
peaks = hand_model.get_peaks(heatmap)
```

Question 15 Show the final results.

[17]:
```
canvas = util.draw_handpose(image, peaks, show_number=True)

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

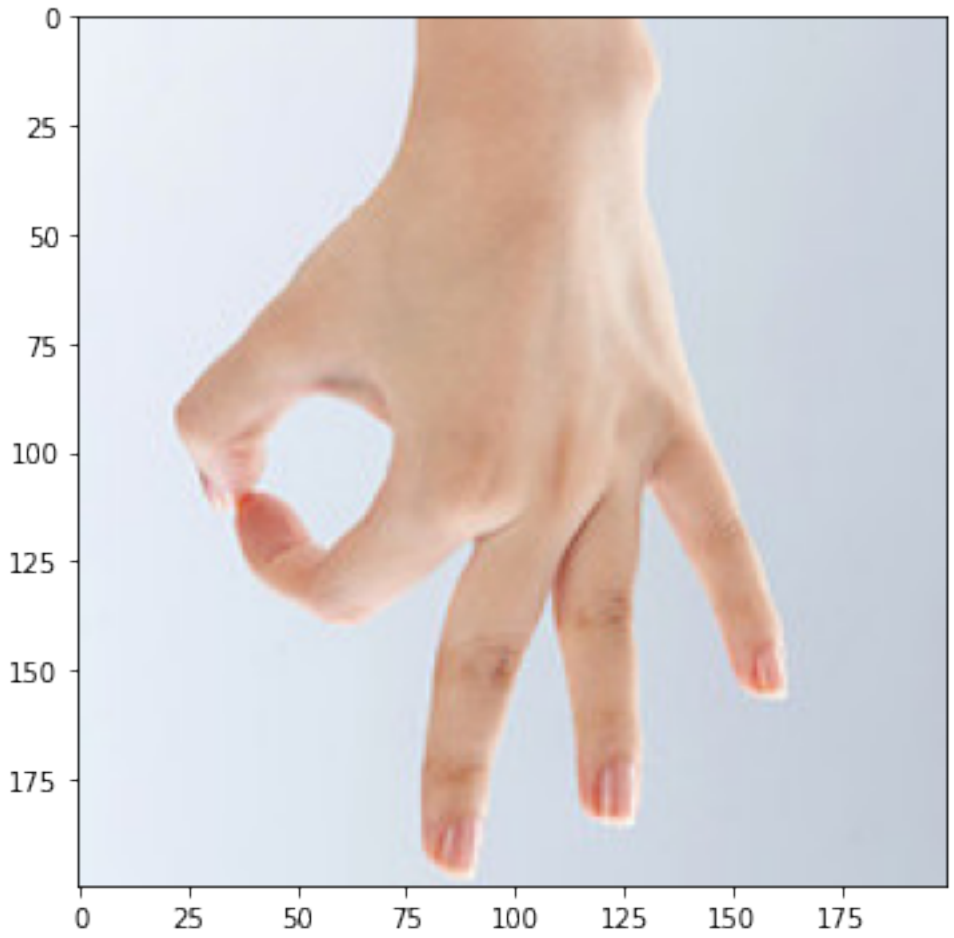
[17]:
```
<matplotlib.image.AxesImage at 0x7fc49500f150>
```



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

[18]:
```
image_path = 'images/hand_180.jpg'

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

[18]:
```
<matplotlib.image.AxesImage at 0x7fc495037d90>
```

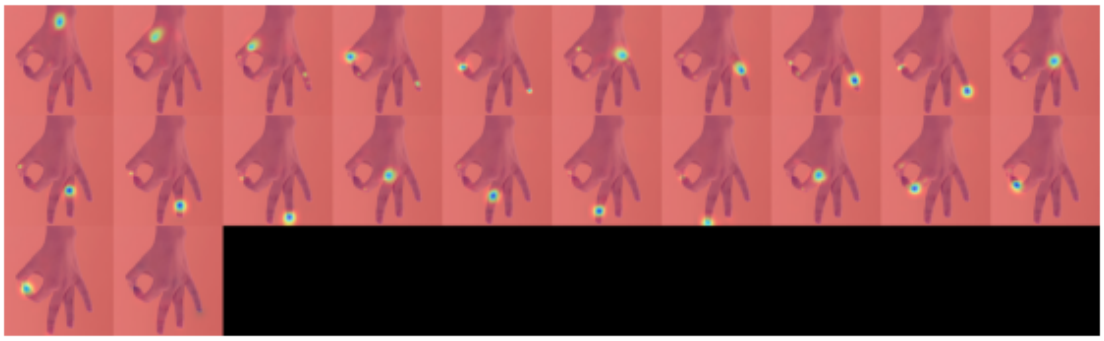


Now let's generate the heatmap by passing the image through the model.

[19]:
```
image = cv2.imread(image_path)  # B,G,R order

hand_model = Hand('../shared/model/hand_pose_model.pth')
heatmap = hand_model.get_heatmap(image)
```

Question 16 Show the heatmap.

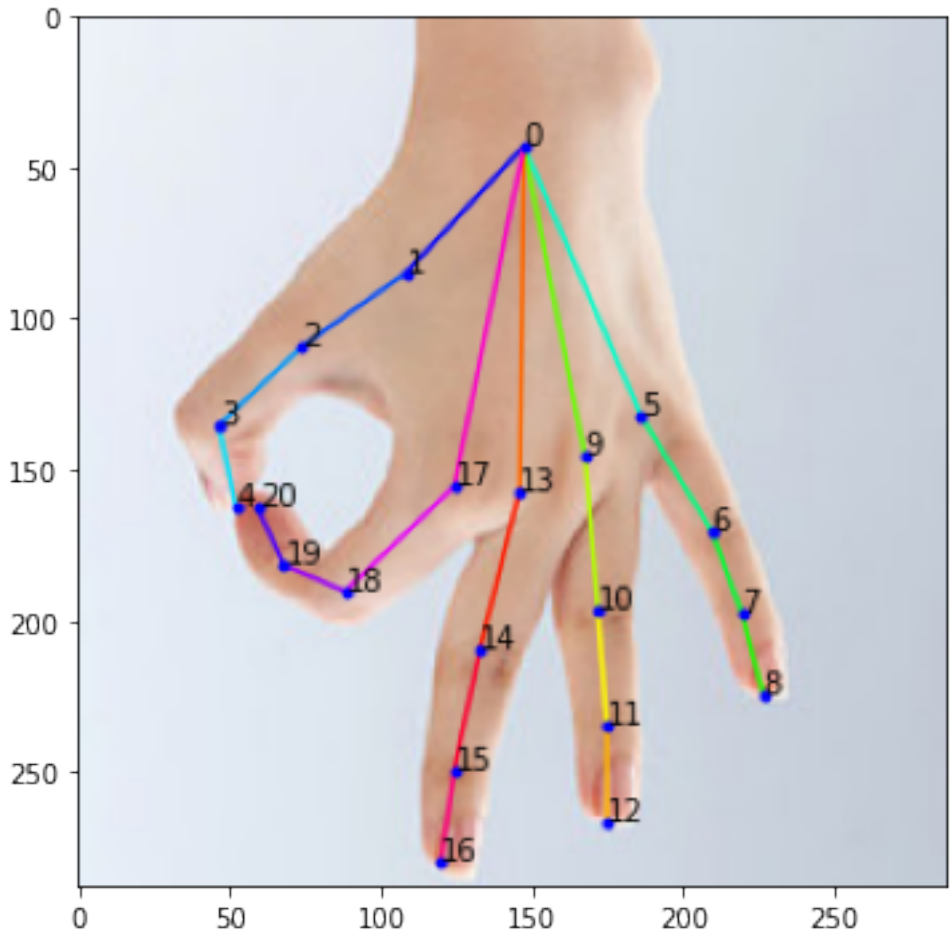
[20]:
```
visualize_heatmap(image_path, heatmap)
```



Question 17 Generate the keypoints and show the final output.

[21]:
```
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]])
```

[21]:
```
<matplotlib.image.AxesImage at 0x7fc494e3b1d0>
```



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

Answer: Yes the handpose model works on the flipped image as well.