ADIP 2018 Fall Project Foreground Extraction- Group 4

Chang-Qi Zhang[a], Chien-Yu Lin[b] and Chen-Hung Hu[c] and
Authors Student ID: 106368002 laboratory: lab107-1 advisor: Lih-Jen Kau[a]
Authors Student ID: 107368003 laboratory: lab107-1 advisor: Lih-Jen Kau[b]
Authors Student ID: 107368001 laboratory: lab107-1 advisor: Lih-Jen Kau[c]

***Abstract*— The foreground extraction methods from previous group. They either use a heavy computation method, Deeplab or manually assign region to achieve human based foreground extraction. In this final project we implement a semi-automatic foreground extraction software which can automatically generate foreground keypoints and provide UI for user select background keypoints.**

I. INTRODUCTION

Traditional image segmentation algorithms, for example, watershed and grabcut. There are differences between these two methods. Watershed initial by assigning both foreground and background keypoints, then based on those points to do gradient search. On the other hand, grabcut initial by giving foreground rectangle region then use Gaussian mixture model, GMM to find foreground region. Due to the watershed algorithm require to assign foreground keypoints on the object which you want to extract, we have to integrate object detection method to get the region of the foreground object. Common object detection, YOLO only give a bounding box of objects. Those bounding boxes usually don't fit perfectly with the foreground as we expect. Therefore, we propose using Openpose[1] to get accurate foreground keypoints on people. In this way, the runtime for our program will be much shorter than using machine learning segmentation technology, DeepLab, Mask RCNN, and SegNet.

II. PROCEDURE METHOD

Fig 1 shows the system structure for our semi-foreground segmentation method. Our system can automatically generate the foreground keypoints when user click on openpose button. The algorithm under the hood is based on OpenPose[1]. For background keypoints, user has to use our user interface to select the background keypoints. Once system get both OpenPose foreground keypoints and user defined background keypoints, we then apply Foreground/Background segmentation algorithm to extract foreground form the input image.

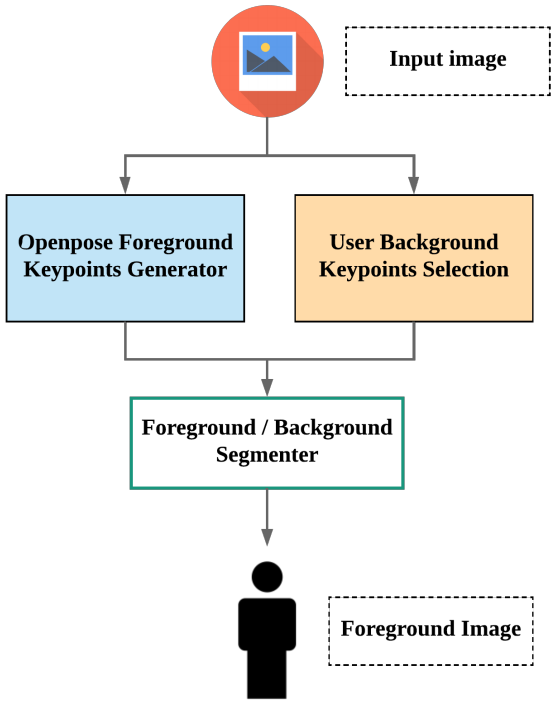

Fig. 1: System structure

A. *Foreground/Background Segmenter*

- Step 1: Use Guassian filtering, with $\sigma = max(image.width, image.height) \times 0.002$;
- Step 2: Get the gradient vector (x y directions) for each channel (r, g, b), Get the length (norm) of each gradient vector for each channel. Sum length for gradient vector in each channel. Add 1 to the sum and divide 1 using this sum. Now we got a priority at this point whose value is in the range (0,1]. For smooth areas the priority value is high, otherwise the priority is low. Gradient in x direction.
- Step 3: Given the labeled keypoints (i.e. foreground or background) and priority image.

  Initialization: For each neighbor (we use 8-neighbor in this project) of the keypoints. Push each neighbor point into priority queque given their priority.

  Propagation: Pop the point with the highest priority. Point with higher priority would be processed first. Push all non-labeled neighbors around this point into priority queue. Set the label of this point to the label of the labeled neighbor. Stop when priority queue is empty. Figure 2e is the result mask, with background in color red, foreground in color green.

(a) Gaussian blurred (b) x direction gradient (c) y direction gradient (d) Priority Image, with black = 0, white = 1 (e) result mask, with background in color red, foreground in color green.

Fig. 2: Foreground/Background Segmente result figures

B. Openpose Foreground Keypoints Generator

Figure 4 shows the convolutional architecture of Openpose. In this structure, it use multiple stages for the training progress. Each stage is locally supervised after compute loss with original using an intermediate loss layer that prevents vanishing gradients during training.[2] the model for our program use six stages for training progress.

The results of foreground keypoints extraction show in Figure 3. As you can see, the keypoints which generated by OpenPose are pointing precisely. Also, it is able to deal with multi people scenario (Figure 3d and Figure 3e).

(a) Test set image 24. (b) Test set image 14. (c) Test set image 21.

(d) Test set image 27. (e) Test set image 29.

Fig. 3: Openpose foreground keypoints results

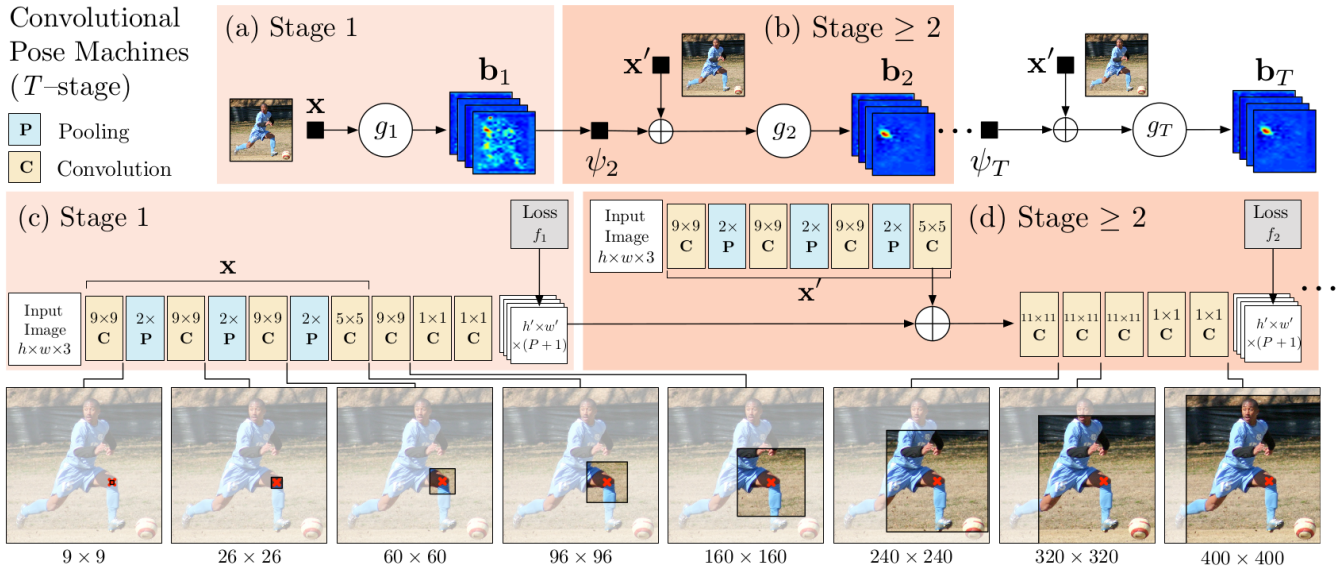

Fig. 4: Openpose network architecture[2]

III. SOFTWARE ARCHITECTURE

Figure 5 shows the software structure for our program. Our program use QT as main thread, it is able to do file control. In manual mode, it can read the mouse event for user defined foreground and background keypoints. With intelligence mode, our program will active Openpose to get human body foreground keypoints.

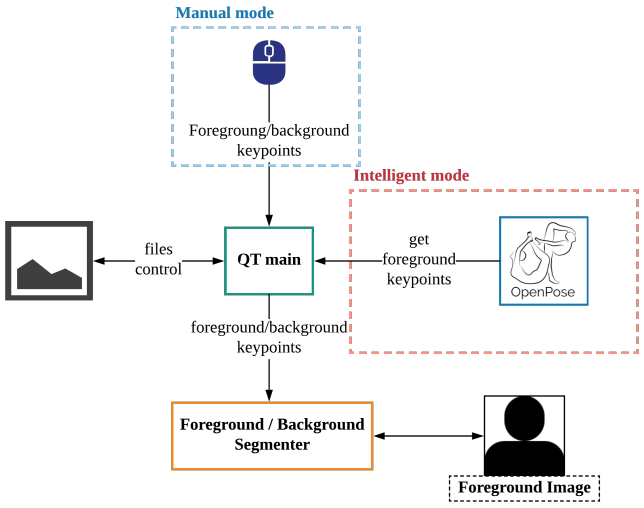

Fig. 5: Software structure

IV. RESULTS AND COMPARE

For evaluating the testing results, we used Dell Precision 5520 laptop with 8 cores Intel Xeon CPU E3-1505M v6 @ 3.00GHz, Quadro M1200 GPU and 32GB RAM. Running on Ubuntu 16.04 operating system.

For evaluation the results, we are using following equations 1, 2 and 3.

$$Precision = \frac{True\ positives}{True\ positives + False\ positives} \quad (1)$$

$$Recall = \frac{True\ positives}{True\ positives + False\ negatives} \quad (2)$$

$$Fmeasure = \frac{2 \times Precision}{Precision + Recall} \quad (3)$$

IMG	Precision	Recall	F measure	MAE	Runtime(ms)
1	0.9151	0.8036	0.8558	0.1287	4415
2	0.9346	0.5528	0.6947	0.1010	1764
3	0.9845	0.9980	0.9912	0.0091	1329
4	0.9832	0.9582	0.9705	0.0246	1622
5	0.9705	0.9944	0.9823	0.0186	1260
6	0.8739	0.9999	0.9327	0.0598	1123
7	0.8231	0.7952	0.8089	0.1654	1688
8	0.8125	0.9391	0.8712	0.1315	1485
9	0.5943	0.9994	0.7453	0.0940	1674
10	0.9031	0.9133	0.9082	0.0899	1376
11	0.9326	0.8997	0.9158	0.0389	1542
12	0.9308	0.8498	0.8884	0.0469	1783
13	0.9597	0.9591	0.9594	0.0183	1527
14	0.6748	0.9508	0.7894	0.1202	1666
15	0.7340	0.7223	0.7281	0.1123	2098
16	0.9710	0.6350	0.7679	0.1504	2121
17	0.9900	0.9999	0.9949	0.0077	1775
18	0.9595	0.9148	0.9366	0.0214	1605
19	0.9220	0.7015	0.7968	0.0449	1703
20	0.9212	0.9865	0.9527	0.0030	1632
21	0.9027	0.9627	0.9317	0.0486	2398
22	0.9229	0.8287	0.8733	0.1099	1779
23	0.9569	0.9937	0.9749	0.0146	1631
Avg.	0.8944	0.8851	0.8813	0.0678	1782

TABLE I: Testing results with test dataset.

Our results from Table I, precision and recall are not very good compare with pure machine learning segmentation technology, DeepLab or Mask RCNN, but our runtime is much faster than either DeepLab or Mask RCNN. Typical runtime for DeepLab and Mask RCNN are approximate 20+ seconds with Nvidia Titan X GPU machine. On the other hand, our method can be running in real time, if we execute program on the powerful desktop instead running normal laptop.

Following Figure 6, 7, 8 are our execution results with test dataset.

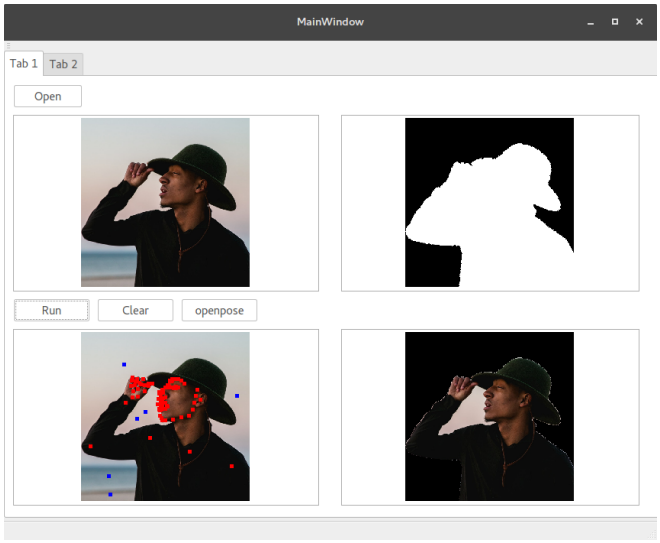

Fig. 6: GUI result with test image 3

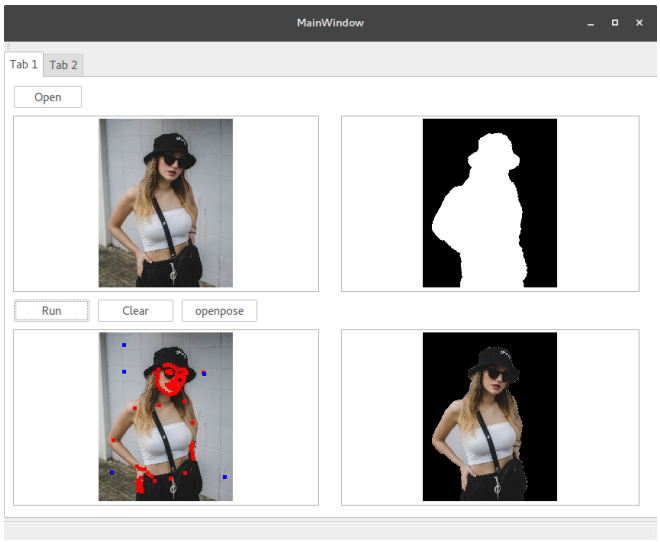

Fig. 7: GUI result with test image 4

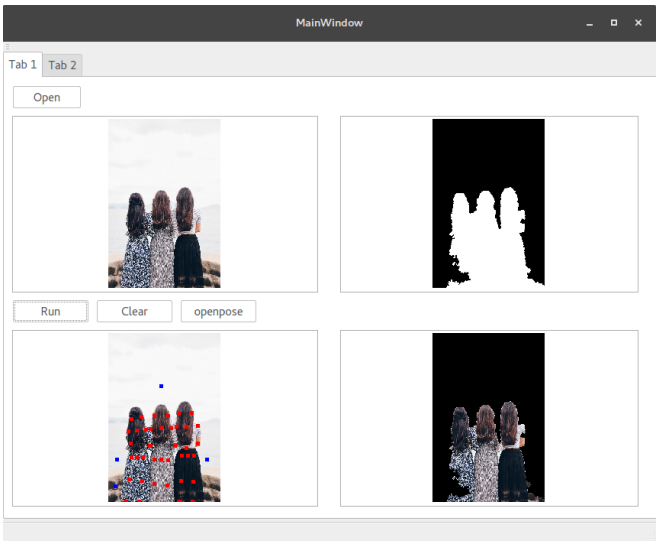

Fig. 8: GUI result with test image 20

V. PROJECT CONTRIBUTION

- Chang-Qi Zhang[a]: Creating and testing library for Openpose foreground keypoints extraction, PPT author and report author.
- Chen-Hung Hu[b]: QT user interface and doing software libraries integration from other authors[ac].
- Chien-Yu Lin[c]: implementing foreground/background Segmenter algorism, helping author[b] to do system integration and project software structure design.

REFERENCES

[1] Zhe Cao and Gines Hidalgo and Tomas Simon and Shih-En Wei and Yaser Sheikh, OpenPose: realtime multi-person 2D pose estimation using Part Affinity Fields
[2] Shih-En Wei, Varun Ramakrishna, Takeo Kanade and Yaser Sheikh, Convolutional Pose Machines
[3] Concurrent computation of topological watershed on shared memory parallel machines
[4] The Morphological Approach to Segmentation - The Watershed Transformation (S. Beucher and F. Meyer)
[5] Topographic distance and watershed lines
[6] Watershed Approaches For Color Image Segmentation
[7] Watershed wiki page
[8] How to code a nice user-guided foreground extraction algorithm? (Addendum)