1 No Cameraman Left Behind

Members: Jeffrey Li, Sindhura Mente, Amar Mohanty, Sriya Reddi

2 Project Summary

In this project, we address a universal issue in taking group pictures: leaving out the photographer. Having access to a source image of only the photographer and a target image of the rest of the group, we propose a program that can incorporate both parties into one realistic final output using open-source software, namely NumPy, OpenCV, and PyTorch. The image processing pipeline involves three major steps: (1) image segmentation of person(s), (2) information extraction from the source image, and (3) gradient domain blending. From qualitative tests, our program works well on images with people standing in front of a solid background and with people standing next to each other on the horizontal axis. The final product rivals the results manually produced using Adobe Photoshop software. The program performs suboptimally on images with complex backgrounds, with people standing at different locations in the foreground, and images taken at different angles. Future work can be done to improve the robustness of our program and to combat more edge cases.

3 Goals & Objectives

The objective for this project is two-fold. Firstly, we want to design a neural network architecture for the detection and labeling of person(s) in images. Achieving a mask with a good score will allow us to detect and locate the person(s) in the source image and target image. To achieve this, we modify the U-Net Convolutional Neural Network architecture for our project, given its high performance on small image datasets.

Secondly, we want to achieve reasonable output images as our final product. In other words, we want the final images to look natural and realistic. Some factors we need to consider include changes in exposure, lighting, shadows, and positions. We work with several assumptions to minimize the effects of these external variables, including having pictures in the same setting (foreground and background), taking pictures at the same angle, and aligning people's positions.

4 Related Works

Segmentation can be accomplished by thresholding, edge-based/boundary-based, region-based, energy-based, hybrid (region and boundary based). Methods such as gradient operator, second derivative operator and optimal edge detector can be used for edge-based segmentation. Methods such as region splitting and merging, region growing and clustering can be used for region-based segmentation. Energy-based segmentation can use methods like active contour, graph normalized cut, graph cut, local variation [1].

U-Net is a convolutional network architecture originally developed by Ronneberger et al. for the segmentation of biomedical images [2]. U-Net's name derives from the symmetric nature of its two paths. The first path is the contracting path which follows the typical architecture of a convolutional network. The second path is the expanding path which enables precise localization using transposed convolutions. Through this symmetry, U-Net can achieve very good segmentation performance even with small datasets [2].

5 Approach

5.1 Produce Ground Truth Images

In our ground truth image, we want the person from the source image to be placed onto the target image. To facilitate this, we use copy-paste and smoothing techniques from Adobe Photoshop software. This resulting image will be used as an image for comparison for the final product of our program.

5.2 Image Segmentation of People

The goal of the image segmentation model is to generate a mask of both the target and source image. We trained an encoder-decoder model from the Segmentations Models Python Library [3. The encoder was a pre-trained encoder called "EfficientNetB3" and the decoder was a U-Net model. We chose the U-Net model due to its speed and accuracy with image segmentation [4]. We could have used a more advanced model, but due to the simplicity of the task and our limited computer resources, we decided to go with a more traditional model. We chose a Binary Cross Entropy for the loss function and the Adam Optimizer due their high performance with image segmentation tasks.

To train the model, we used the SuperVise.ly human segmentation dataset which contained 2,667 images of individual humans and their associated mask. Since the images varied in sizes, we used the Albumentations library to resize all the images to 192x256. Additionally, to improve the performance of the model, we transformed the images so that there were changes to color and flipped the images when training the model. Since the model would output a mask by 192x256, we had to warp the mask to the original input image size. Once the model was trained, we found that we could reliably generate a mask for the person in the image. For our application, we generated a mask for both the target image and the source image.

5.3 Finding Source Height and Width

The input to the next part of the pipeline is the full size mask of both the source and target images. The trained Neural Network detects and segments the person in the source and target images and provides a mask. The key consideration though is that both masks are the same size as the original image. Because of this we are not able to simply perform gradient domain blending between the images. The problem with directly doing gradient blending is that the people will not be aligned when placed next to each other, and since the masks contain the majority of the backgrounds there will be huge discrepancies if blended. It will be apparent that the people were not next to each other when the picture was taken. To overcome this we would need to determine where the person is in the mask based on their edges.

To accomplish this task we drew a bounding box around both the source person and the target person. We did this through OpenCV's `find_contours()` and `regionprops()` which allowed us to extract the top left and bottom right corner pixels of the bounding box around each person. From this we are able to calculate the location of the top right and bottom left corner of the box. We used OpenCV's `rectangle()` to visualize this and found this package performed well in identifying and drawing a tight bounding box around the person. This is favorable because it includes very little of the background which is important when placing our source person in the target image to reduce conflicts between the backgrounds.

The next step is to determine key pixel locations in the source and target images.

5.4 Placement Center in Target Image

Important Note: These next two sections are for the purpose of calculating the values needed in the next stage of the pipeline (Gradient Domain Blending) to actually place the source person in the target image. The next part of the pipeline expects the pixel location in the target image where the center of the source person needs to be placed. It also expects the cropped source image to be placed into the target [5][6].

In order to do this, the first location to find was the bottom left corner of the bounding box of the target person, basically the ending of the right foot (when viewed from their perspective), since we are always inserting the source person to the right of the target person (again from their perspective). We used the pixel coordinates calculated in the previous step to extract this value.

Next, we determined the height and width of the source person to find their center point. We approximated this using the pixel coordinates of their bounding box. We derived the following equations to do this:

$$X = bottom\_right\_x - (Width/2) - padding$$

$$Y = bottom\_right\_y + (Height/2)$$

where bottom_right_x is the x coordinate of the bottom right corner of the target person's right foot (can also be thought of as the bottom left corner of the bounding box), width is the width of the source person in pixels, bottom_right_y is the y coordinate of the bottom right corner of the target person's right foot (can also be thought of as the bottom left corner of the bounding box), height is the height of the source person in pixels, and padding is a pixel value to have some distance between the two people so they don't overlap [7][8].

We send this calculated center point to the Gradient Domain Blending step of the pipeline.

5.5 Cropping Source Image

As previously mentioned we want to avoid blending the entire mask of the source image with the target image because their backgrounds will look unnatural and will not yield visually appealing results. So we send a cropped part of the image containing the source person to the next step in the pipeline. We achieved this by simply taking the bounding box of the source image and cropping it accordingly.

5.6 Gradient Domain Blending

In image blending, we want to create an output image from the composition of two different and separate images. The ultimate goal is to seamlessly incorporate and blend an object from the source image onto a target image. Using a simple copy-paste technique, or alpha blending, we would achieve non-ideal results. The regions of the source image bordering the region of the target image would be too obvious to the human eye, due to the drastic changes in image intensity.

We resort to gradient domain blending for this segment of the project. The gradients capture properties of an image, including shape, shading, texture, and illumination. We work with the gradients to preserve these features from the source image and to effectively blend the source image region onto the target image. The output image will appear more natural compared to the result from alpha blending, because we blend the gradients to ensure image properties are integrated smoothly.

We use OpenCV's `seamlessClone()` function to achieve our project goals [9][10]. The function requires source image, target image, source mask, and pixel coordinates for placing source image. We modify the source image and source mask by cropping them so it will only contain the source object (person). Additionally, to avoid border gradient being artificially interpreted as null, we use a rectangular bounding box to increase the white space of the mask [11]. Furthermore, we change the color of the mask from black to gray to avoid large contrasts that would result in poor blending. We pass the cv2.NORMAL_CLONE argument to preserve the gradient of the source image in the target image [11].

6 Results

From qualitative tests, our program works well on images with people standing in front of a solid background and with people standing next to each other on the horizontal axis. These results are understandable. Because of the assumptions we asserted at the beginning of the project (see Goals Objective), we successfully reduced the effects of external variables in our final output. Since the background is solid and remains unchanged for both the source image and target image, gradient domain blending can easily blend and smooth the two images. Using a gray and white mask mitigates the risks for obvious blending distortions. Additionally, having the two people aligned on the horizontal axis makes it easier to place the source image person and target image person next to each other without further transformations.

In **Figure 1**, we use a solid background. The image on the left is the ground truth produced using Adobe Photoshop software. The image on the right is our output result. Simple Photoshop copy-paste and smoothing techniques place the source person perfectly left of the target person, but does not accommodate for shadows and looks very rigid and unnatural. Our program successfully places the target person and accommodates for the gradients and lighting changes. We see minor shading blemishes on the right arm of the target person and blending distortions near the bottom of the wall.

Figure 1: Solid Background. Left to Right: Source image, target image, program output, and ground truth.

In **Figure 2**, we use a complex window background. The positioning of the target person on the left and the gradient blending on the solid floor are good. However, we see blending distortions in the window and parts of the wall due from using the larger mask.

Figure 2: Complex Window Background. Left to Right: Source image, target image, and program output.

Figure 3: Window Background and Sitting. Source image, target image, and program output.

In **Figure 3**, we use seated person(s) and a complex window background. The positioning of the target person on the left and the gradient blending on the solid floor are good. However, we see blending distortions in the window and the wooden structure.

Contrastly, the program performs suboptimally on images with complex backgrounds, with people standing at different locations in the foreground, and images taken at different angles. Since we are using a larger mask to avoid artificially induced null gradients, portions of the source image outside of the source person will be incorporated onto the target image. Secondly, our program does not accommodate changes in location beyond a horizontal translation, because it cannot resize people. This means that if the source/target person(s) is far away from the target/source person(s), there will be substantial differences in sizes of the people when they are placed next to each other. Similarly, the program does not accommodate changes in the vertical direction.

In **Appendix A**, we use a complex wall background. The positioning of the target person on the left and the gradient blending on the solid floor and wall are good. However, we see obvious blending distortions with the poster. Additionally, the target person appears blended with the poster and results in a 'ghost' image. In **Appendix B**, the source person and the target person are not standing on the same horizontal plane. As a result, the source person appears much larger than the target person. In **Appendix C**, the source image was taken at a high angle and the target image was taken at a low angle. As a result, the output image keeps the angle (and size) in which both people were taken.

7 Conclusion and Future Plans

The developed algorithm successfully creates an image with the source person placed appropriately in the target image. Given an input of two images where the cameraman is included in one of them, the algorithm merges both the images to create an image that includes everyone. This ensures that no one gets left behind in the picture. There are a number of potential additions that can be made to broaden the scope of the results produced, such as:

- If both the images are taken from different distances, the source person would need to be resized to create a visually appealing result.
- If both the images are taken from different angles the source persons' angle would need to be adjusted to match the angle in the target image.
- There are few instances where the lighting and shadows of the source person do not match the target image. This can be rectified with an enhanced/modified blending algorithm.
- The mask created can sometimes include the background. When working with non-solid or different backgrounds there can be potential visual discrepancies in the background.

8 Sources

1 https://www.researchgate.net/publication/354846947_A_Comprehensive_Review_of_Image_Segmentation_Techniques

2 https://arxiv.org/abs/1505.04597

3 https://github.com/qubvel/segmentation_models.pytorch

4 https://lmb.informatik.uni-freiburg.de/people/ronneber/u-net/

5 https://stackoverflow.com/questions/61403755/get-the-real-bounding-box-of-a-rectangle-shaped-mask

6 https://github.com/nikhilroxtomar/Semantic-Segmentation-Mask-to-Bounding-Box/blob/main/mask_to_bbox.py

7 https://www.geeksforgeeks.org/python-opencv-cv2-rectangle-method/

8 https://stackoverflow.com/questions/60869306/how-to-simple-crop-the-bounding-box-in-python-opencv

9 https://docs.opencv.org/3.4/df/da0/group__photo__clone.html

10 https://learnopencv.com/seamless-cloning-using-opencv-python-cpp/

11 https://stackoverflow.com/questions/49008854/using-seamlessclone-in-opencv-python-produces-image-with

A Complex Wall Background

Figure 4: Left to Right: Source image, target image, and program output.

B Standing on Different Planes

Figure 5: Left to Right: Source image, target image, and program output.

C Different Camera Angle

Figure 6: Left to Right: Source image, target image, and program output.