Improving Interactive Image Segmentation with Image Preprocessing

Richard Du

Adviser: Jia Deng

Independent Work Report, Spring 2019

Abstract

Interactive image segmentation uses user input to iteratively refine the object segmentation mask. In this paper, we investigate ways to improve interactive image segmentation models. We experiment with various approaches of using the output from non-interactive object segmentation models to improve the performance of the GrabCut [3] algorithm. We also simulate user input for evaluation purposes and investigate the effect of different user input formats on model performance. By using the output of Mask R-CNN [1] to preprocess the image, and then applying the GrabCut algorithm, we are able to significantly reduce the number of iterations required to achieve 0.85 and 0.9 IOU on the COCO [2] dataset. Our result holds for all user input formats.

1. Introduction

Datasets drive progress in machine learning research. A large, high quality dataset often boosts the performance of existing models, and sometimes allows us to train novel model architectures that require new forms of input data. Thus, constructing new datasets and finding ways to streamline the annotation of new datasets is highly impactful.

One new input form is that of 3D shape data. 3D shape data is particularly relevant today for several reasons. First of all, many industry applications, such as autonomous vehicles, require 3D input. Second of all, many of the latest computer vision models, such as PointNet, require 3D data. While 3D shape datasets such as ShapeNet and ModelNet exist, they are limited in their size as well as the number of object categories that they contain. Thus, creating large-scale 3D object datasets with more categories will be important in advancing computer vision research.

With this in mind, the goal of my Independent Work Seminar was to create an application to convert a photograph of an object into a 3D model of that object. This complex task was broken down into a pipeline of tasks. The first step in this pipeline was to take a photograph of an object and extract the object segmentation mask, i.e. extract all of the pixels that belong to that specific object. This is illustrated in Figure 1.

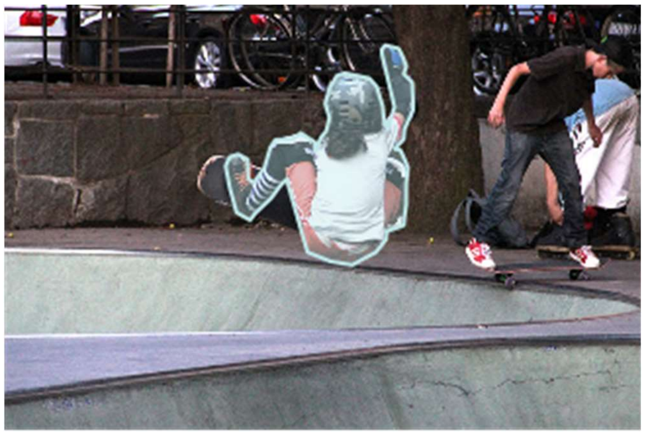

Figure 1: An example of an object segmentation mask

My project focused on improving existing interactive object segmentation models. Specifically, I worked on improving the GrabCut model using some modern object segmentation models.

2. Project Background

Formally, the specification of the project was as follows.

Inputs:

- An RGB image: an array of size (3, H, W)
- User input: at each step, we have access to the input of a user, who is helping us refine our segmentation mask. The format of the user input is up to us to define.

Outputs:

- An object segmentation mask: more formally, this is a (H, W) array that specifies for each pixel in the image whether or not the pixel belongs to the object of interest.

Additionally, we constrain our task by only focusing on one object per image. This simpler task is what most of the existing research has focused on. Good performance on this task generalizes to segmenting multiple objects in an image. To extract multiple objects per image, we simply use the same image but change the object of interest.

When we incorporate user input into the segmentation algorithm, this is known as interactive object segmentation, which is a different problem from regular object segmentation. In addition to having different inputs, the two types of problems have different goals. Non-interactive object segmentation presumes that there will be no user input, and then tries to maximize the segmentation accuracy. On the other hand, interactive object segmentation's main goal is to achieve a certain level of segmentation accuracy, while minimizing the amount of user input required.

3. Related Work

At a high level, interactive segmentation models work as follows.

Input: An image is fed into the interactive segmentation model. If there are multiple objects in the image, the object of interest will be indicated.

Initialization/Iteration: We repeat the following steps until we reach a desired accuracy OR we reach a predefined maximum number of iterations:

1. User input is either simulated or requested from the real user. In the first iteration, there will sometimes be no user input.
2. The interactive segmentation model takes the previous iteration's object mask, as well as the user input, and outputs a refined object mask. In the first iteration, there is no previous object mask.

This is illustrated in Figure 2.

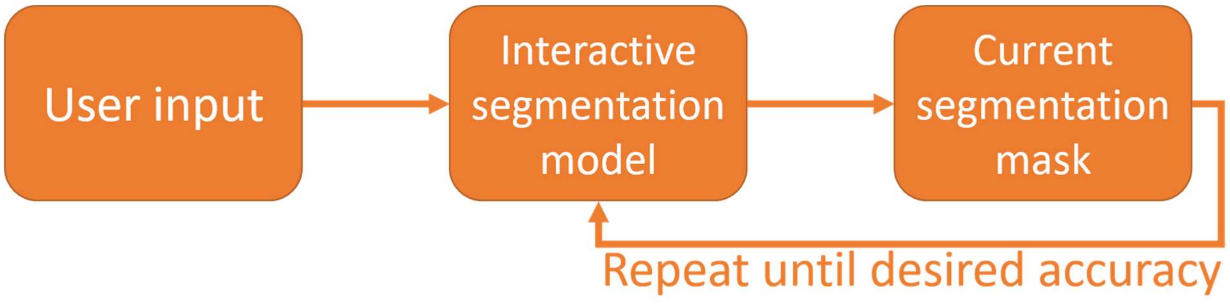
User input
Interactive segmentation model
Current segmentation mask
Repeat until desired accuracy

Figure 2 A visualization of a general interactive segmentation model

3.1 GrabCut

GrabCut [3] is an iterative image segmentation model. At a high level, it works as follows:

1. The user draws a rectangle around the object of interest. This defines the foreground region (inside the rectangle) and the background region (outside the rectangle).
2. The color distributions of the foreground and the background of the image are estimated with a Gaussian Mixture Model.
3. The overall quality of the image segmentation is defined by an energy function. The energy function is iteratively minimized until convergence, and this is the initial mask.
4. The following steps are repeated until the desired accuracy is achieved:
    a. The user can edit the image by labelling some of the pixels of the image. They can label the pixels to be definitely within the object, or definitely outside the object.
    b. The mask is refined using this new information from the user

There are some additional steps, such as border matting and incorporating the detected edges into the mask.

At each step, the algorithm has an internal array that gives each pixel one of 4 labels: probably in the object, probably not in the object, definitely in the object, and definitely not in the object. Users inform the algorithm of which points are definitely in/not in the object, and this information is used to refine the segmentation mask.

3.2 Mask R-CNN

Mask R-CNN [1] is a non-interactive object segmentation model. It simply takes in an image and outputs a segmentation mask of all the objects in the image. For this project, we do not need to know the details of how it is implemented.

4. Approach

In this paper, we improve the GrabCut algorithm by improving the initialization of the algorithm. By improving this initial object segmentation, we are able to substantially reduce the number of iterations required to achieve 0.85 and 0.9 IOU.

The key idea is to incorporate the predictions of other object segmentation models in order to improve GrabCut's initial object segmentation. Our most successful result comes from preprocessing the image using the output of Mask R-CNN.

5. Implementation

The next few sections go into detail on each component of the interactive segmentation model and evaluation framework.

5.1 User Input

There are many ways to specify how the user should interact with the segmentation model. Good user input formats should have the following properties:

1. They are robust to errors and imprecisions that are normal when working with real users.
2. They have the capacity to express both positive and negative regions, i.e. they should be able to inform the segmentation mask of regions that should be in the mask, as well as regions that are incorrectly in the mask
3. They are intuitive for non-sophisticated users

Our goal is to achieve a good segmentation as quickly as possible, meaning there is a trade-off between how much information the user input can convey, and how much time and skill it takes to enter the input.

The approach I settled on after surveying the literature was to have the user label pixels in the image as either positive or negative pixels. Positive pixels are pixels that should belong in the mask, while negative pixels are pixels that should not belong in the mask. I experimented with two ways of labelling these pixels.

5.1.1 Line segments

Here, the user draws straight lines to label positive and negative regions, as shown in Figure 3. The red pixels represent negative labels, while the green pixels represent positive labels.

5.1.2 Points

The user simply clicks points that are positive or negative, as shown in Figure 4. Again, red and green represent negative and positive points, respectively.

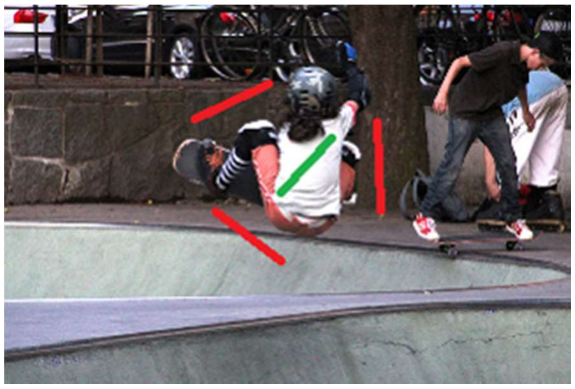

Figure 3 Line input example

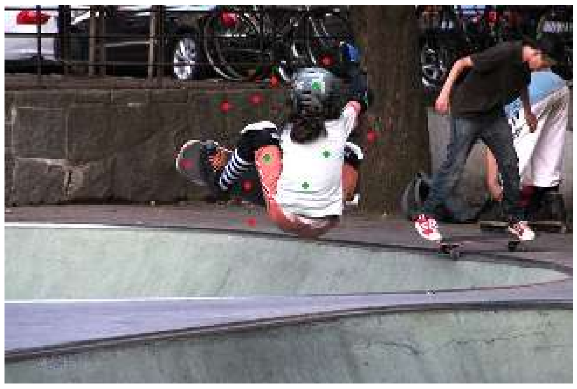

Figure 4 Point input example

While line segments will give a lot more information, they are also more time consuming to enter than points. Thus, depending on the performance, as well as the time required for both forms of inputs, we can determine the optimal input format.

After determining the user input format, we also need to define one unit of user effort. This is so that we can evaluate the model's performance. We define one unit of user effort to be either 1 point or 1 line segment on the image.

5.2 User Input Simulation

User input was required for both training and testing purposes. While I could have used a service such as Amazon Turk to get real user input, the timeframe and resources of this project precluded its use. Instead, I simulated the user input.

The procedure to simulate the user input was as follows:

1. We input the actual mask, as well as the predicted mask
2. Taking these masks, we create a false positive (FP) region as well as a false negative (FN) region. The false negative region is the region inside the actual mask, but not inside the predicted mask. Similarly, the false positive region is the region inside the predicted mask, but not inside the actual mask.
3. We compare the area of the false positive region with the false negative region.
4. We select the larger region. The idea is that the model will gain more information from the larger region, because there is more improvement possible.
5. From this larger region, we either:
    a. If the input format is a point: sample a point from the region. This will be the simulated user input.
    b. If the input format is a line: Sample two points from the region. Then, we draw a line connecting the two points.
        i. Because the FP/FN regions may not be convex, it is possible that the line will also contain points that are not within the region. Thus, we take the intersection of this line with the region.

This procedure is illustrated in Figure 5.

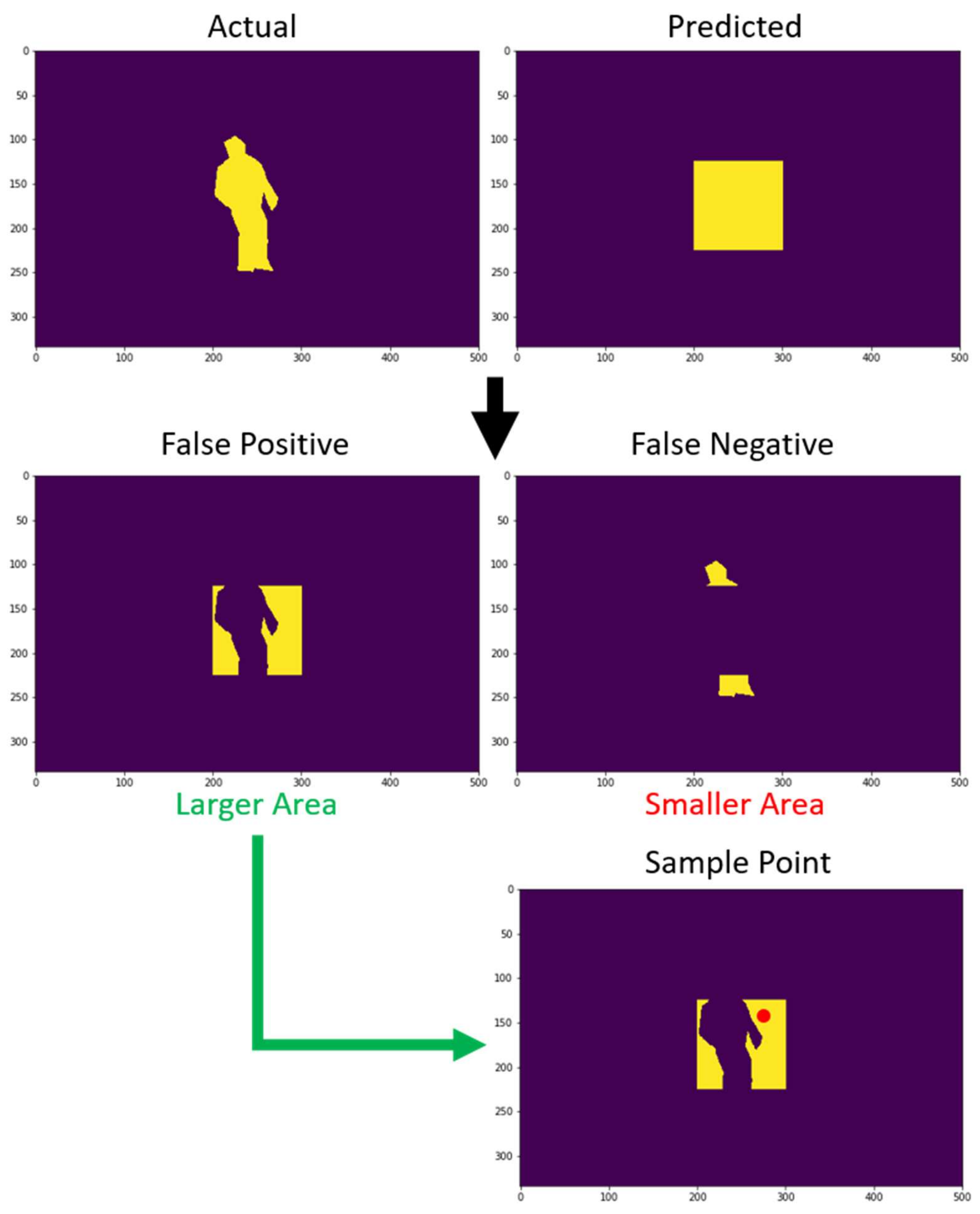

Figure 5 An example of simulating user input. Here the user input format is a point.

5.3 Evaluation Framework

For this project, we used the COCO dataset [2], a standard dataset that consists of images with object annotations. Another advantage of the COCO dataset is that there is a large number of object categories.

Before discussing the evaluation framework, we first discuss our evaluation metric, which is IOU (intersection over union). The IOU is a standard metric for evaluating the quality of a segmentation mask given the ground truth segmentation mask. It's computed by computing the area of the intersection of the ground truth with the predicted segmentation mask, as well as the area of the union of the two masks. The ratio of these two areas is the IOU metric. This is shown in Figure 6.

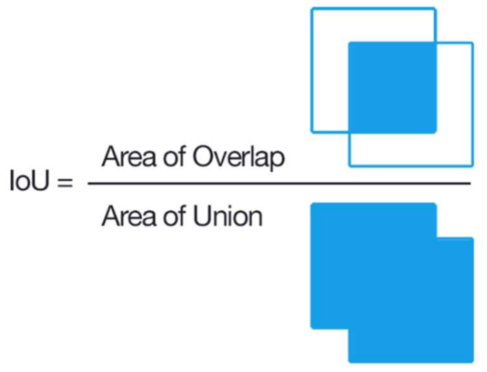

Figure 6 Diagram by Adrian Rosebrock, CC BY-SA 4.0

The performance of our model is measured **how many iterations are required to achieve certain IOU levels** (the lower the better). Generally, we set these IOU targets to be 0.85 and 0.9.

Our evaluation framework works as follows:

1. For a large number of images N:
    a. Most images have multiple objects, and so we need to determine how we will choose the object of interest. A logical choice, and the convention that we use, is to simply pick the largest object, which we define to be the object with the largest number of pixels in the image.

b. Once we pick the largest object, we repeat the following steps for up to T iterations:
    i. If the algorithm requires user input, then we simulate it.
    ii. We take in the previous segmentation mask, as well as the user input, and then compute the current segmentation mask.
    iii. We compute our evaluation metric, IOU. Then we store the metric as well as the current segmentation mask.

2. At the end of this, we compute the average IOU across all images for each iteration.

For most of my experiments, I used N = 500 or N = 1000. While more images is always better, I found that this gave me fairly consistent results, while not taking too much compute time. I generally set T = 20 when the user input was in the form of lines, and T = 60 when the user input was in the form of points. T was selected so that the models would reach > 0.9 IOU. This is because I found that after 0.9, the IOU metric was no longer particularly informative about the actual accuracy of the segmentation mask. This was because the segmentation masks were polygons, and so they weren't perfectly accurate. Often masks that appeared to be more accurate than the ground truth label would have IOUs no larger than 0.9. Thus, I felt that 0.9 IOU was a reasonable threshold at which to end the iterations.

6. Results

These results are written in chronological order of the experiments I ran. Initially, I used lines as the input format, and so the initial results below are all with line inputs.

Furthermore, unless stated, all of these results were evaluated using the evaluation framework discussed above.

6.1 Baseline Model

The baseline model that I used was the GrabCut model, which was the benchmark for the rest of the other models. To implement the GrabCut model, I used the OpenCV library for Python, which has a built in GrabCut method.

The GrabCut method, as mentioned above, requires a bounding box around the object of interest in order to initialize the first mask. This is unlike the rest of the user input, and so this was implemented separately.

To simulate the user input for the bounding box, I took the ground truth bounding box from the COCO dataset for that specific object. While the bounding box was based on the ground truth polygon mask, and thus wasn't perfectly accurate, a real user would also not be able to draw a perfectly accurate bounding box.

As you can see in Figure 7, the performance was quite good. We plot the 25th, 50th, and 75th percentile metrics for the IOU at each iteration, as well as the average IOU at each iteration. From this plot, we can see that the median IOU reaches 0.9 after 7 iterations, which is quite fast. On the other hand, the 25th percentile takes 12 iterations before the 0.9 is reached.

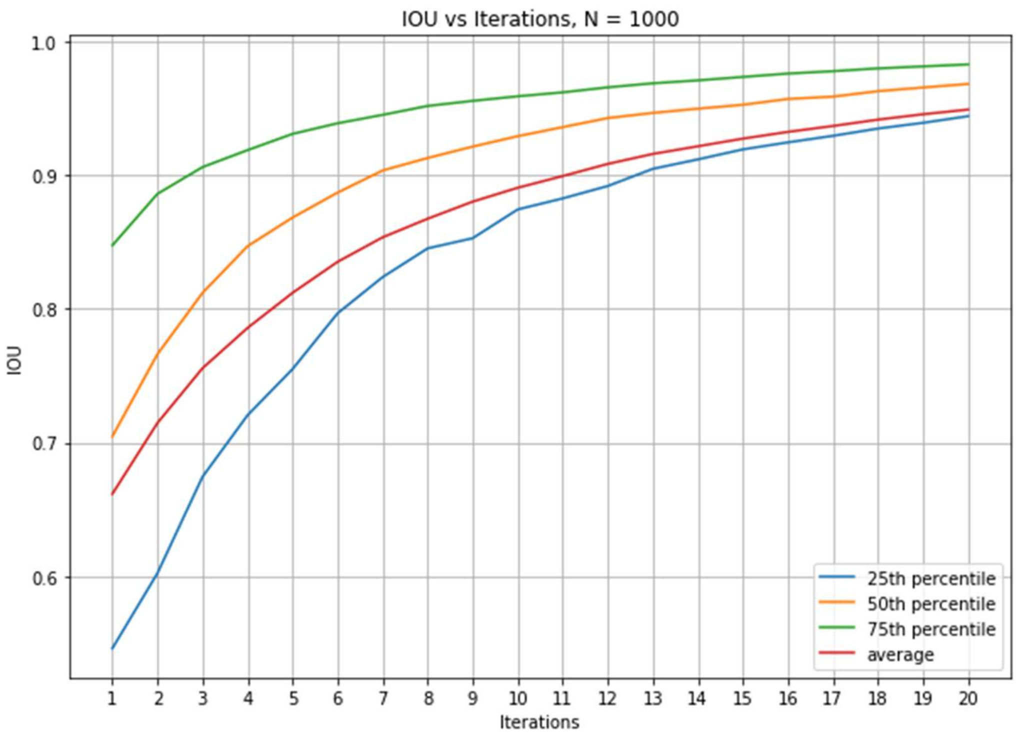

Figure 7 GrabCut baseline model performance.

6.2 Combining GrabCut with Mask R-CNN

After implementing the baseline model, I found that the main problem was that the algorithm had a slow start. Looking at the graph, we can see that the first iteration has rather low IOU (25th and 50th percentile IOUs are approximately 0.55 and 0.7, and average IOU is approximately 0.66). Thus, if we could improve the first iteration, this would dramatically shorten the number of iterations required to reach our goals of 0.85 and 0.9 IOU. A clear way to do this would be to combine GrabCut with a non-interactive object segmentation model, in order to use it as a prior for the GrabCut algorithm.

Thus, I decided to use the Mask R-CNN model, which was a state of the art object segmentation model in 2017. Note that while Mask R-CNN is an excellent non-interactive object segmentation model, it does not accept user input, and thus cannot improve its object segmentation predictions.

How it works is not particularly important – we treat it as a black box that simply takes an image and outputs the segmentation masks of all the objects in the image.

Once we have the Mask R-CNN image segmentation mask, we locate the object mask that corresponds to the object that we are interested in. This can be accomplished in any of the following ways:

- Iterate over all of the masks outputted by the Mask R-CNN model, and find the one with the largest IOU with the object that we are interested in.
- Ask the user to input a point in the object of interest, and return the mask that contains this point.

Once we have the object segmentation mask that is predicted by Mask R-CNN, there are many ways to combine it with GrabCut as a prior. Here are the approaches I experimented with.

6.2.1 Direct Initialization

In the first step, GrabCut produces an initial object segmentation mask of the object. Thus, the most straightforward approach to incorporate Mask R-CNN was to simply take the output mask predicted by Mask R-CNN, and substitute it as the first mask used by the GrabCut algorithm.`

As mentioned above, GrabCut labels each pixel as either definitely foreground/background or probably foreground/background. Thus, I used the object mask of Mask R-CNN and assigned all of them to be probably foreground points. I then let the algorithm run as usual.

The idea is that this would jump start the GrabCut algorithm, and the algorithm could simply work on improving this initialization. The IOU of just using the Mask R-CNN algorithm was > 80%, and so it was expected that this would dramatically improve the initial couple of iterations for the GrabCut algorithm.

Surprisingly, this approach was completely ineffective. After initializing the GrabCut algorithm with the Mask R-CNN algorithm, the IOU of the first iteration is very high, which is promising. However, in the second iteration, the algorithm essentially overwrites the progress made, reverting to the performance of the first iteration in the original GrabCut algorithm. Thus, this actually increased the number of iterations needed. An example of this is in Figure 8.

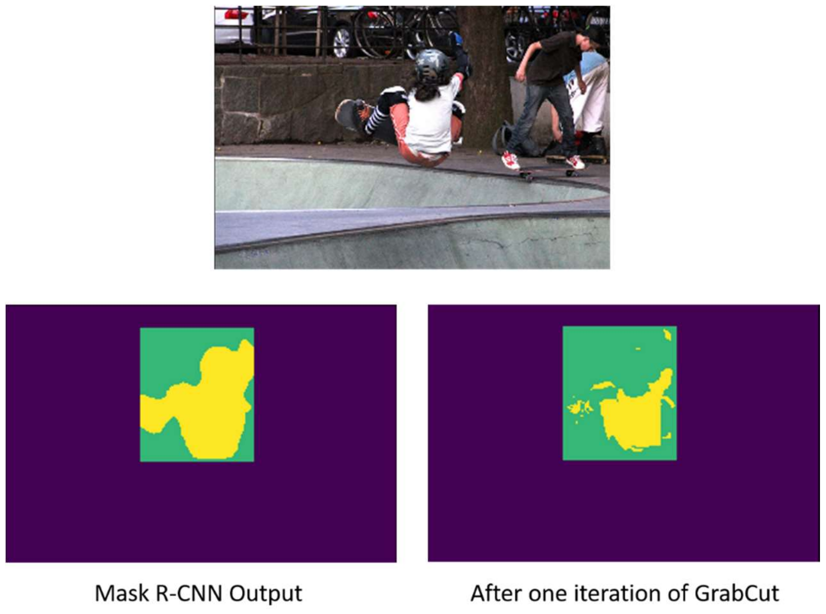

Figure 8 An example of a failed initialization. The GrabCut overwrites the Mask R-CNN output. Legend: purple = background, green = probably background, yellow = probably object.

I believed that the reason why a basic initialization was ineffective was because GrabCut creates its segmentation by minimizing an energy function that is based on the color distributions of the foreground and background. Thus, in our basic initialization method, we do not actually change this energy function, as the image remains unchanged. As a result, we would expect the initialization to do nothing, as it has not changed any of the internal state variables that the algorithm uses.

Thus, this first attempt to use Mask R-CNN was ineffective.

6.2.2 Use the output of Mask R-CNN as a 'ground truth'

At a high level, the problem with the first initialization approach was that it clashed with the GrabCut algorithm, and as a result, the algorithm rejected the information from the Mask R-CNN output. With this in mind, the second initialization approach was designed to work more closely with the algorithm. The approach was to use the output of Mask R-CNN as the 'ground truth'. Specifically, we performed the following steps:

1. Compute the output mask of the Mask R-CNN network
2. Take this output mask and 'pretend' it is the ground truth for 10 iterations:
    a. In the first iteration, we take a bounding box around the Mask R-CNN network's output and use it as the rectangle to initialize.
    b. At each iteration, we refine the current segmentation mask by simulating the user input, using the Mask R-CNN output as the ground truth.
    c. We refine the mask using the GrabCut algorithm.
3. At the end of these 10 iterations, the resulting mask is our initial mask.
4. We discard the Mask R-CNN output and use the ground truth mask again. Now the process continues like it normally does for 20 iterations.

Notice that there is a difference between the user input in step 2 and the simulated user input in step 4. In step 2, we never ask for an external user's input. We simply try to emulate what a user would do in order to refine the GrabCut algorithm's prediction and mold it into the output mask of the Mask R-CNN network. Essentially, we are trying to get the GrabCut algorithm's segmentation mask to be close to the Mask R-CNN output, before we actually start using the user's input to refine our segmentation mask.

The results were not quite as good as the original GrabCut algorithm. While the initial IOU was higher, the rate of increase of the IOU was lower than in the GrabCut algorithm, and so after 20 iterations, the final IOU was actually not as high as we would have expected.

After examining several examples, I found that this approach had some disadvantages, which is possibly what resulted in the lower rate of growth in the IOU. Specifically, in order to mold the initial segmentation mask into the Mask R-CNN segmentation mask, we had to label points as either inside or outside the object. Because the Mask R-CNN output mask was not actually the ground truth, the result was that many points were mislabelled. Furthermore, these points were labeled as definitely in the foreground/background, and thus they persist until the user relabels them. Thus, these mislabelled points significantly hinder the progress of the GrabCut algorithm. Another problem is these mislabelled points create small holes and discontinuities in the segmentation mask. Because the ultimate application of these object segmentation masks is to be used as a ground truth for other algorithms, these artifacts would make them less useful as a ground truth.

6.2.3 Preprocessing the image using the Mask R-CNN output

The second approach failed because it created discontinuities, and the initialization process hindered the algorithm in subsequent steps. The third approach sought to fix the problems of the first two approaches by modifying the image by changing the color space. Specifically:

1. We take the output mask of the Mask R-CNN model.
2. Then, we take all of the pixels of the original image that are **not** in the Mask R-CNN object mask, and then we transform these pixels by darkening them. Specifically, we multiply their RGB values by $\alpha < 1$, i.e. $(R, G, B) \rightarrow (\alpha * R, \alpha * G, \alpha * B)$.

3. After transforming the image, we run the usual GrabCut algorithm with the usual initialization and iterations.

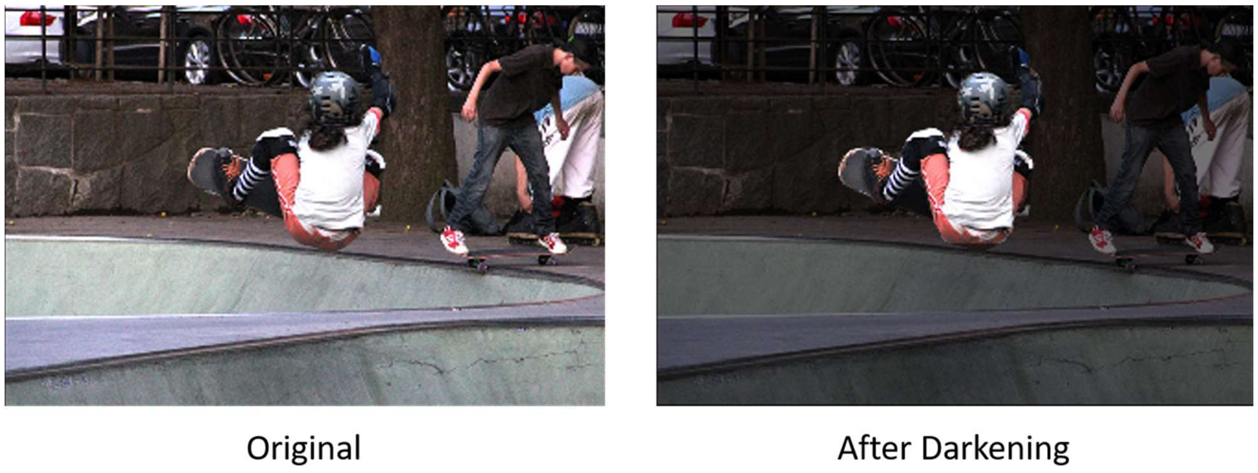

Figure 9 The result of the transformation. Here, α = 0.45.

This is shown in Figure 9. The idea behind this approach is that because the GrabCut algorithm estimates the color space of the foreground and background, by darkening the pixels that are in background of the Mask R-CNN output, this will increase the distance between the foreground and background's color spaces. This will tend to induce the GrabCut algorithm to predict something closer to the Mask-CNN output. Furthermore, this fixes the problem with the second approach, because it doesn't lead to any mislabelled points.

This approach has a parameter alpha, and thus, in order to evaluate this approach, we had to do a hyper parameter search across alphas, meaning we evaluated the performance on different alpha values. A smaller alpha meant we darkened the background more. We tried $\alpha = 0.4, 0.6, 0.8$.

This results of this approach can be seen in Figure 10, which plots the average IOU vs number of iterations for all 3 alphas that we tried, as well as the original GrabCut algorithm. The best performing model was $\alpha = 0.4$. At the 0.85 IOU level, the improved model requires approximately 2 iterations, which is a 50% decrease from 4 iterations. At the 0.9 IOU level, the improved model requires 5 iterations, which is a 28% decrease from 7. Thus, we can see that there has been a substantial improvement in performance.

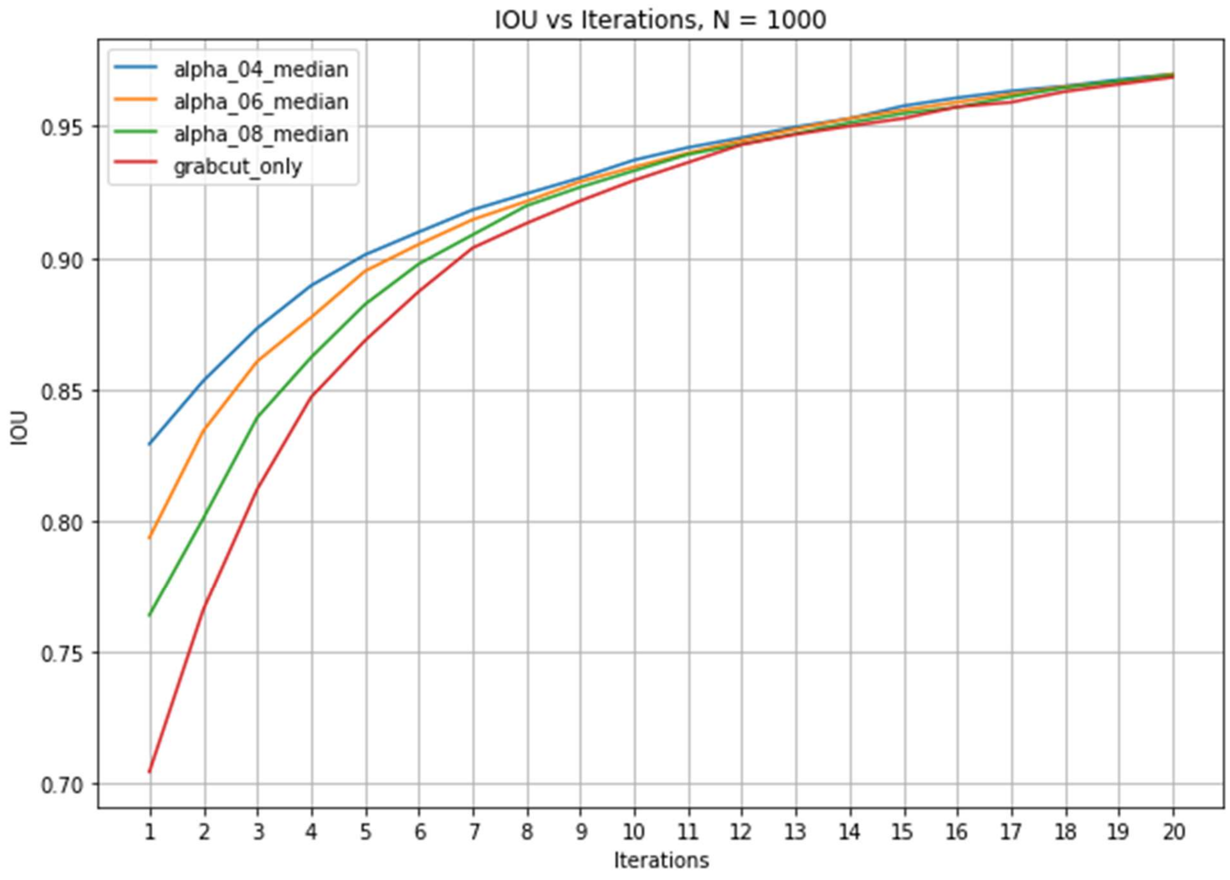

Figure 10 Performance of the Mask R-CNN + GrabCut model and the baseline GrabCut model

I chose to darken the points in the background, and specifically I chose to accomplish this by multiplying the R, G, and B values all by the same constant. However, there are a lot of different ways to transform the colors of the points. Some other possibilities that could be explored would be changing the HSV values (e.g. by changing the saturation), or by computing the distribution of colors in the image, and using that to determine individual R, G, B multipliers for each color channel.

6.2.4 Ensemble approach

We have shown that using Mask R-CNN as a prior will improve the performance of the GrabCut algorithm. Thus, the next logical step is to see if we can push this improvement even further by combining the predictions of multiple models. Before, when we would darken the pixels that were not in the predicted Mask R-CNN mask, we were essentially coloring the pixels based on our confidence that they were in the object mask. Now, with multiple models, we can do the same thing, but at a more granular level. Specifically:

1. We take 4 different architectures, all under the Mask R-CNN umbrella of object segmentation models. Each model has been trained separately, and will give different predictions for the object segmentation mask.
2. As before, each model will output an object segmentation mask over the entire image, and so for each model, we first pick the object mask that corresponds to the object of interest.
3. Now, we combine these object segmentation masks by adding them together, as shown in Figure 11. The result is an array that is the same size of the image, where the value of each element in the array represents the number of models that classified the corresponding pixel in the image as inside the object of interest. This represents our confidence that the pixel belongs in the segmentation mask.
4. We darken the pixels in the image by different amounts, depending on the confidence in the pixel. This step has a vector of alpha parameters $(\alpha_0, \alpha_1, \alpha_2, \alpha_3, \alpha_4)$, where α_i represents how much we darken the pixel if there are i models that believe the pixel is in the object. While we can pick alpha however we want, we generally will want $\alpha_0 < \alpha_1 < \alpha_2 < \alpha_3 < \alpha_4$.

5. After modifying the image, we let the GrabCut algorithm proceed as usual.

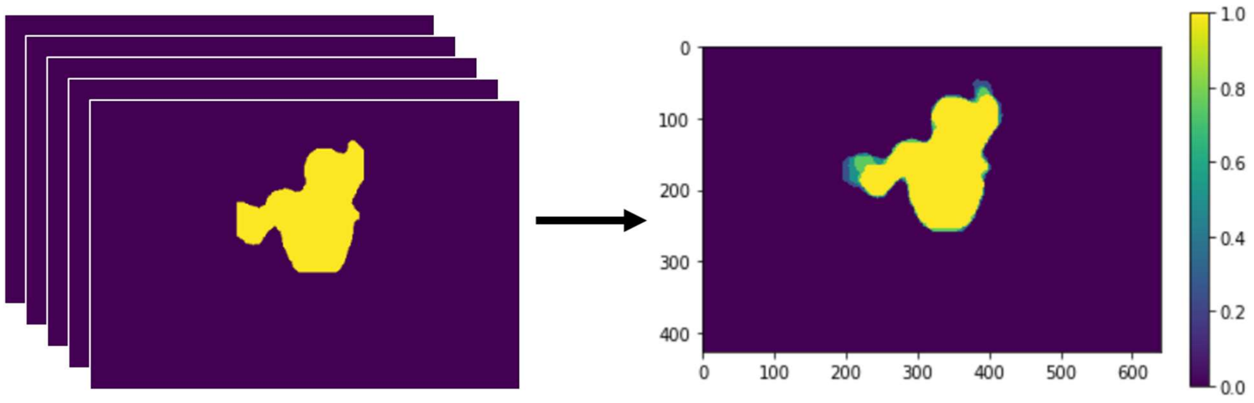

Figure 11 we combine several predictions into one.

For this approach, we also had to do a hyperparameter search. Because the dimension of the hyperparameters was higher (there are 4 possible alpha parameters), the search space is much larger. Thus, we used some heuristics to pick the alphas so that we wouldn't have to search through as many possibilities. When only using the output of 1 model, $\alpha = 0.4$ performed quite well, and so we decided to keep $\alpha_0 = 0.4$, as α_0 represents the darkening applied to the pixels outside of any of the output masks. Similarly, $\alpha_4 = 1$ to keep the results comparable to the single model approach. For $(\alpha_1, \alpha_2, \alpha_3)$, we constrained them to increase uniformly, and so the three sequences we tried were $(\alpha_1, \alpha_2, \alpha_3) = (0.55, 0.7, 0.85), (0.7, 0.8, 0.9), (0.85, 0.9, 0.95)$. The larger the difference between the alphas, the less we care about the pixels that are less confident. In an ideal scenario, we would be able to test more alpha values.

The results of the ensemble method approach are shown below in Figure 12. We compare the different hyper parameters with the result of the basic GrabCut model. As we can see, it performs noticeably better than the GrabCut model, but has essentially the same performance as the $\alpha = 0.4$ model in the previous section.

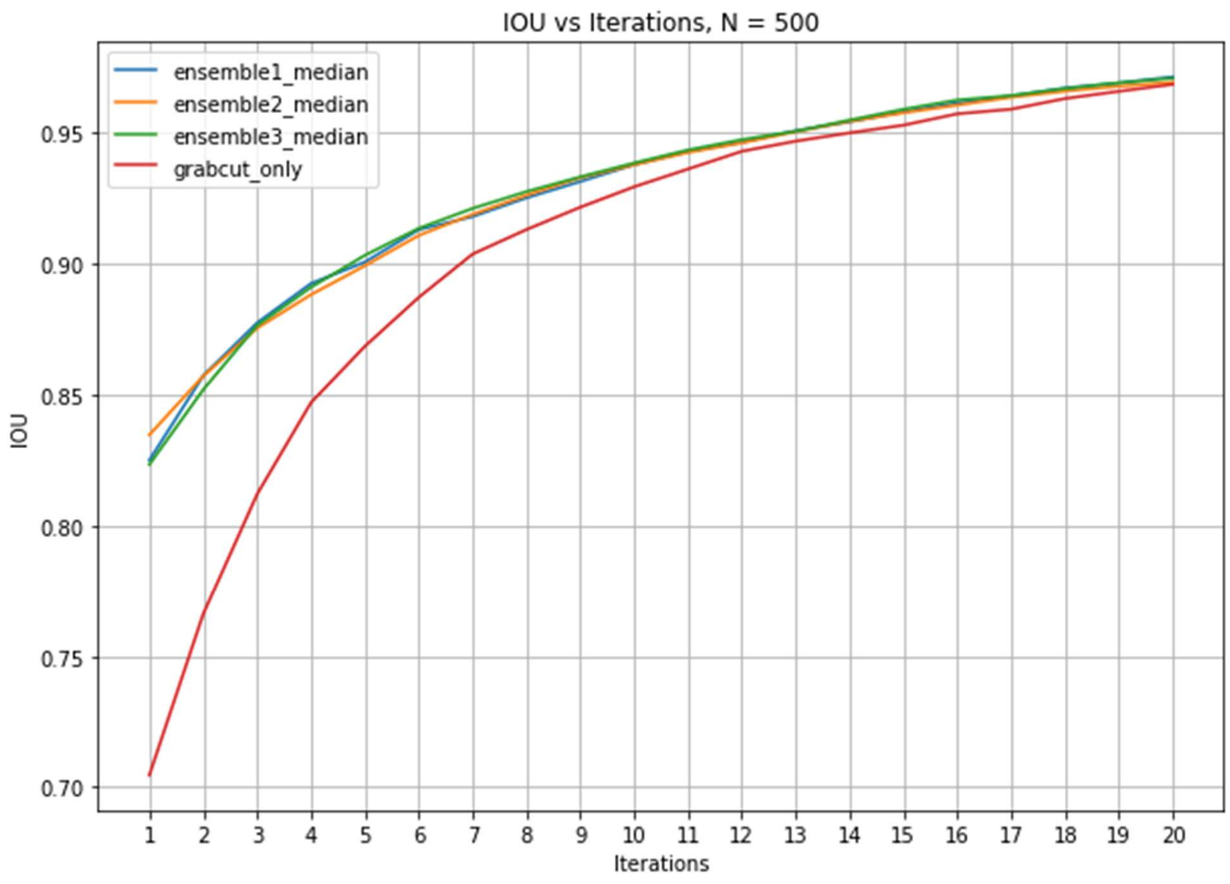

Figure 12 Performance of 3 different ensembles (the difference is in the alpha parameters)

We would expect the ensemble method to be at least as good as using a single model, as averaging multiple segmentation masks should lead to a better prediction. However, there was not a noticeable difference in performance between these ensemble methods and the single model approach. This could have been because of the alpha parameters that were chosen. A more likely explanation is we are reaching diminishing returns by only focusing on the initialization part of the GrabCut algorithm.

6.3 User Input with Points

After trying various approaches with line segments, I decided to use points as the user input format, to see how much the performance would degrade.

In Figure 13, we can see the baseline performance of the GrabCut model when points are the input format. As you can see, there is a very substantial degradation in performance, with points requiring twice as many iterations to achieve the same level of performance. This is to be expected, as points contain far less information than line segments.

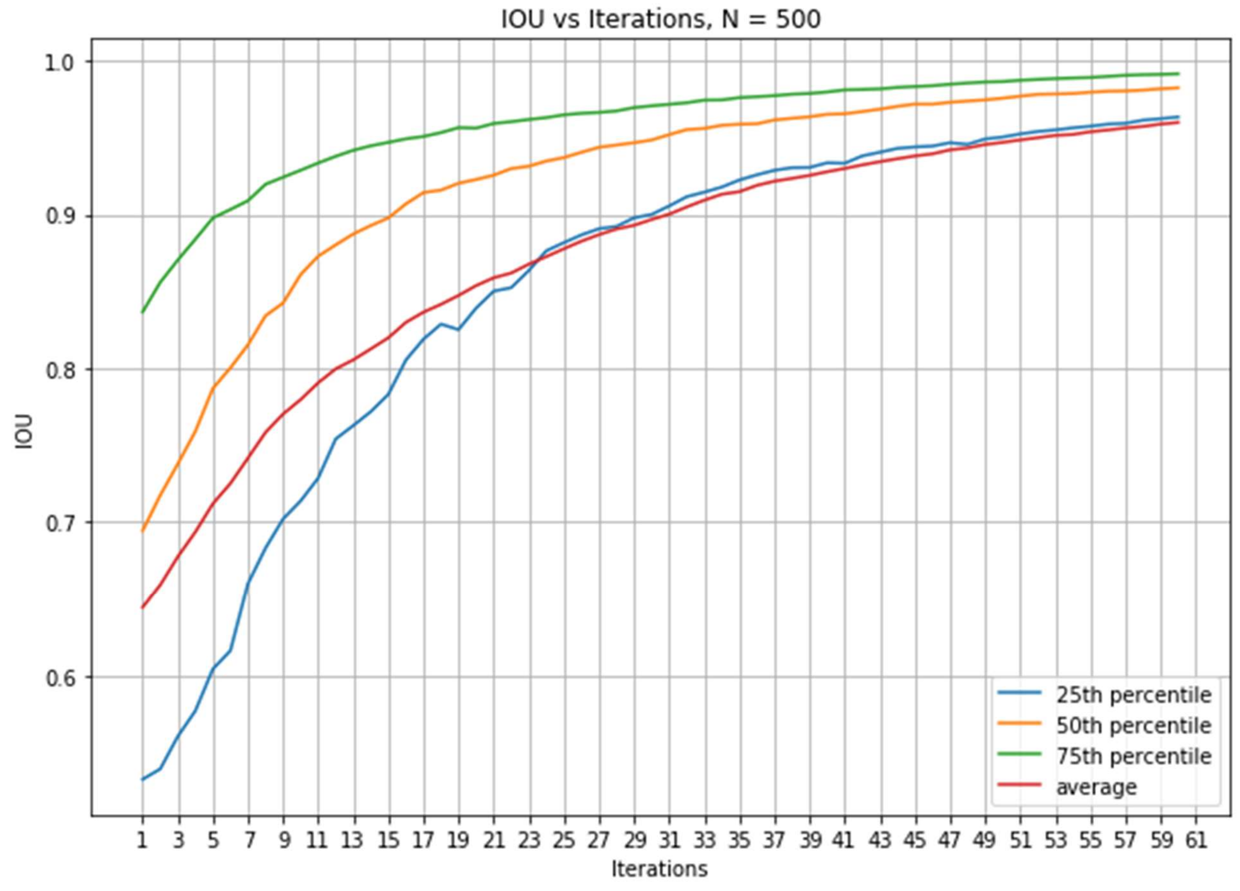

Figure 13 Notice that the performance degrades significantly

The other result of using points is that the initialization of the mask becomes much more important, because the rate of IOU increase is lower, and thus a good initialization can allow us to achieve the same IOU in far less steps.

In Figure 14, we compare the Mask R-CNN initialization approach with $\alpha = 0.4$ with the baseline GrabCut model. We can see that this time, there is an even more substantial effect on the performance. At the 0.85 IOU level, the Mask R-CNN approach requires 3 iterations, which is a 66% decrease from 9 iterations. At the 0.9 IOU level, the Mask R-CNN approach requires 10 iterations on average, which is a 33% decrease from 15 iterations. Thus, we can see that the benefit of the initialization is larger with a simpler user input format.

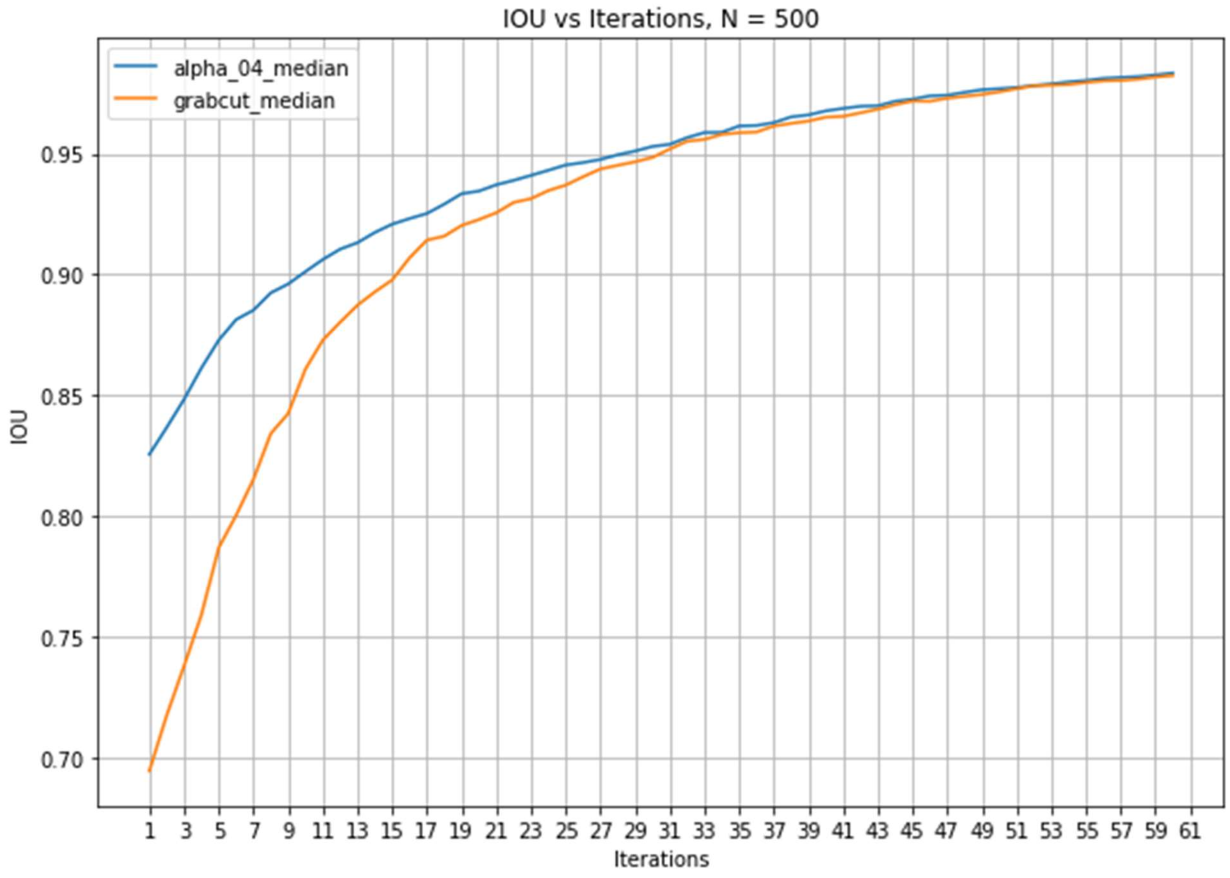

Figure 14 The performance increase is more dramatic when using points as the input format

7. Conclusions + Next Steps

In this project, we were able to substantially improve the GrabCut algorithm by combining it with non-interactive object segmentation models. We experimented with several approaches of combining these non-interactive models with the GrabCut algorithm, and did some hyperparameter searching to find the optimal hyperparameter values. We also experimented with and simulated different forms of user input, and created an evaluation framework to compare performance.

From our experiments, we are able to come to the following conclusions:

- GrabCut has a slow start that decreases its performance
- This slow start can be remedied by incorporating non-interactive object segmentation methods, such as Mask R-CNN
- Not all approaches to incorporating the output of Mask R-CNN are effective, but using the output to pre-process the input image before running the GrabCut algorithm works well
- Ensemble approaches are not substantially more effective than just using one model, and this implies that past a certain point, improving the initialization algorithm has diminishing returns.
- The performance increase from incorporating Mask R-CNN is larger when points are used instead of line segments as the user input

Our experiments provided insight into the shortcomings of the GrabCut algorithm, and also provide us with effective ways to combine non-interactive segmentation models to boost the performance of interactive segmentation models.

It is important to note that our performance is not as good as modern deep learning approaches. However, we were able to create a system with solid performance that is fairly easy to implement and sufficient for most purposes.

There are several directions that could be explored for next steps. We found that preprocessing the image had a substantial impact on performance, and so it is worth seeing if this could generalize to other interactive object segmentation models. Specifically, we can turn the Mask R-CNN initialization as an image preprocessing module, incorporate it into other interactive object segmentation models, and evaluate whether this boosts performance.

We could also do a more thorough hyperparameter search with cross validation. This would help probably improve performance further.

8. Acknowledgements

I wanted to thank Professor Jia Deng for his guidance during this semester. His insights led to several of the techniques that I tried, and his experience in this field helped me quickly get up to speed with the existing research. Overall, I had a fruitful and enjoyable IW experience.

Appendix

Code

All of the code is in a local git repository at the moment. However, it will soon be on GitHub. Feel free to email me for access to the repository.

Honor Code

I pledge my honor that I have not violated the Honor Code.

Richard Du

References

[1] He, K., Gkioxari, G., Dollár, P., & Girshick, R. (2017). Mask r-cnn. In Proceedings of the IEEE international conference on computer vision (pp. 2961-2969).

[2] Lin, T. Y., Maire, M., Belongie, S., Hays, J., Perona, P., Ramanan, D., ... & Zitnick, C. L. (2014, September). Microsoft coco: Common objects in context. In European conference on computer vision (pp. 740-755). Springer, Cham.

[3] Rother, C., Kolmogorov, V., & Blake, A. (2004, August). Grabcut: Interactive foreground extraction using iterated graph cuts. In ACM transactions on graphics (TOG) (Vol. 23, No. 3, pp. 309-314). ACM.