CS229 Final Project
One Click: Object Removal

Ming Jiang Nicolas Meunier

December 12, 2008

1 Introduction

In this project, our goal is to come up with an algorithm that can automatically detect the contour of an object selected by the user and remove this object from the image by replacing it with a plausible estimate of the background. The algorithm mainly consists of two steps of tasks. The first one is to detect the contour of an object in an image. Given a pixel in the image chosen by the user, the algorithm should be able to define a mask surrounding the entire object containing the selected pixel. We do not need to find the boundaries of all objects in the image but just to find a mask that contains at least all the pixels of an object that we would like to remove. The second important task is to remove properly this object from the image. For this task, given an entire mask containing pixels to be replaced, we aim to produce an algorithm that can precisely replace the object by a visually pleasant estimate of the background behind it.

2 Foreground Detection

Our entire goal being to remove an object from an image, we first need to be able to define a mask that represent our object and that we would like to remove. In other terms, given a simple click by the user, we need to figure out what object he wants to remove and automatically define a mask that surrounds this object. As we want to work on the widest possible class of objects, we decided not to implement any kind of object detection that would limit our algorithm to the learnt objects. We finally came up with the idea to define the mask by preprocessing our image in superpixels (cluster of coherent pixels) and then asking the user to choose one of them that is inside the object he wants to remove. It then remains the task to define all the superpixels that are inside the object. For the superpixel extraction, we used the method of Ren and Malik with the available code at `http://www.cs.sfu.ca/~mori/research/superpixels/code/` and we then developed the following algorithm based on a simplified segmentation classifier.

2.1 Principles of the algorithm

In order to compute the entire mask, we decided to build a classifier that learns if a superpixel is part of the foreground (so is potentially part of an object we would like to remove) or part of the background (and then we don't want to add this superpixel to the mask as it could contain important information for the image inpainting task). This is a simple segmentation problem where we only have to distinguish between two possibilities for a superpixel.

To build this classifier, we first need to extract features from the superpixel that will help us distinguish between background and foreground. The features we used in our algorithm contain some color statistics over the superpixels, texture information based on different filterings (wavelet filters), geometry information about the superixels and their spatial location in the image. Then

we implemented two learning methods based on these features to classify the superpixels: logistic regression and SVM. We used groundtruth information about the foreground and background of images as our training data (around 15000 groundtruth superpixels taken from 100 images). After training, we evaluate our algorithm on a separate set of testing data (around 15000 superpixels) and produce the estimated foreground mask for each image. First the mask contains all predicted foreground superpixels. Then based on the user's selection and adjacency information of the superpixels, we create the final mask which contains only the connected components around the user selection. Figure 1 illustrates the algorithm process. You can find another example in Figure 8.

Figure 1: Process of generating foreground mask

2.2 Comparison and use of the different classifiers

We used two classifiers for learning: logistic regression and SVM (with different kernels). To compare the performance of these two methods, we used the following three measures:

1. Accuracy: the rate of correct predictions made by the model over a data set.
2. Precision: the number of true positives (i.e. the number of items correctly labeled as belonging to the class) divided by the total number of elements labeled as belonging to the class (i.e. the sum of true positives and false positives).
3. Recall: the number of true positives divided by the total number of elements that actually belong to the class (i.e. the sum of true positives and false negatives).

There are two big issues that are worth noticing for our algorithm. One is that there will be many more negative data (i.e. the superpixels belonging to background) than positive data (i.e. the superpixels belonging to foreground). That means we cannot judge the performance of our classifiers only by looking at its accuracy. As our goal is to detect the foreground objects, it is more important to focus on the precision and recall measures than accuracy. Table 1 gives the comparison result between logistic regression and SVM with a linear kernel for a non-biased decision threshold.

Table 1: Comparison between logistic regression and SVM

	Logistic	SVM
Accuracy	0.9016	0.8804
Precision	0.6414	0.7555
Recall	0.4662	0.4655

The other important issue is that for the image inpainting algorithm, we can tolerate a mask bigger than the original object. That means we can sacrifice the accuracy and precision to some extent for a higher recall. To change the recall, we can adjust the threshold parameter T of

boundary decision. We will label foreground superpixels with a probability/margin larger than T and background otherwise. Figure 2 illustrates the precision-recall curve.

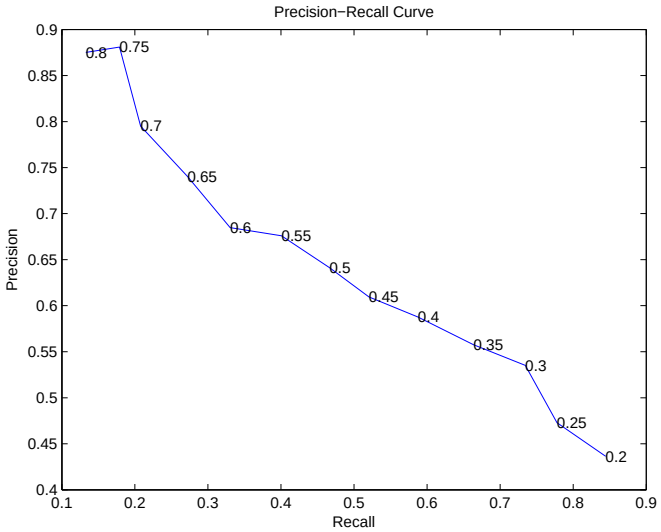

Figure 2: Precision-Recall Curve

Finally we choose a threshold parameter 0.3 as opposed to the normal 0.5 for testing in our algorithm, which produces a precision of 53.51% and a recall of 73.45%.

3 Image Inpainting

For the inpainting part itself, we now have a mask that has been filled in by the previous algorithm and that we want to remove from the image. The mask is considered as a missing region of the image that we want to fill in such a way that the result will seem visually natural. We do not intend to restore any hidden object in our project, just to replace an object by a plausible background. Many algorithms have been developed over the last years, using:

1. statistical-based methods - given the texture surrounding the mask, this approach tries to extract interesting statistics for those texture and then reproduce them inside the mask area. The typical problem of this kind of approach is that it works well when we are only working with textures.

2. PDE-based methods - given information surrounding the mask, we try to propagate it inside the mask using a diffusion process. This process is generally simulated by solving a partial differential equation (PDE). A problem of this approach is that the mask generally has to be quite thin and the information quite smooth to give a visual natural solution. So, when the missing region is highly textured or corresponds to a large object, those methods would lead to a blurry solution.

3. Examplar-based methods - This is the most known successful techniques for inpainting purpose. These methods fill the mask by copying content from the known part of the image, thus considering that the missing information inside the mask can be found elsewhere in the image.

For this project, we decided to use an examplar-based inpainting approach and to improve it by using machine learning to define optimal metrics used in the algorithm. The challenge here is to be

able to fill the removed object by a plausible background that fits well with the rest of the image. We want it to preserve edges inside the region and we want it to use appropriate information from the background to do so. For this purpose, we implemented the algorithm in [1].

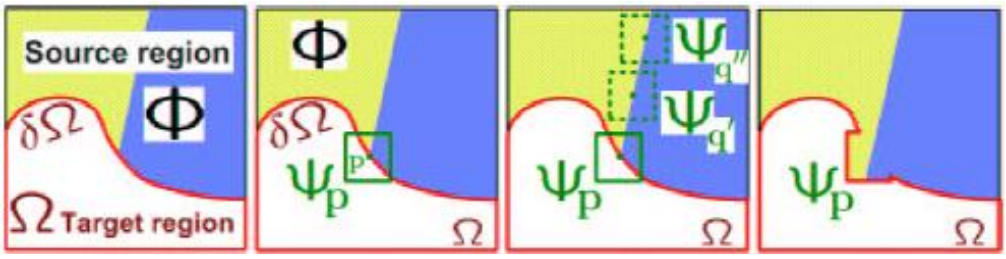

Figure 3: principle of the algorithm

We have a mask Ω that we want to remove from the image. We will progressively fill the mask with information coming from the rest of the image. To do so, we will use a template window that will find the closest patch in the image (outside the mask) to a given patch ψ_p on the border of our mask. This closest patch in the image will be used to fill the missing information of the border patch inside the mask. To compute this closest patch, we will define our own metric that is defined in the next section.

For every patch ψ_p around the mask we define a priority order in which we will fill the mask based on the parameters defined in figure 3. We modified the original priority function in [1] using different features, like presence of edges around a patch, and presence of different kind of segmentation, using the algorithm in [3] and the Stair Vision Library [2].

3.1 Comparing patch and propagating information inside the mask

If we want to choose a correct patch to replace part of the mask, we need a way to compare patches. In the original algorithm, only color difference is used to compute distinction between different patches. We purpose here to use a learning algorithm to define a new metric to pick up patches for replacing. For two given patches, we can define a small set of different features that are some characteristics of their difference:

1. the sum of absolute difference of each color channel
2. the sum of square difference of each color channel
3. the number of pixels with a different segmentation
4. the difference of entropy in the segmentation

We then have a set of features $S = \{F_i\}_{1 \leq i \leq n}$ and we can define a function that permits us to compare two patches as follows :

$$f(P_1, P_2) = \sum_{i=1}^{n} w_i F_i(P_1, P_2)$$

For a given starting patch around the mask we can select a potential replacing patch and run our inpainting algorithm starting with this patch. We then score the final result comparing its color and segmentation with the original image. For each patch tried as a potential solution we can then attribute a score. By this method, we can then create a training database on which to run a learning algorithm (here a logistic regression) to maximize our final metric. The results of this approach gives almost equal importance to the different features and with the learnt metric, it permits to avoid reconstructing unwanted objects inside the mask as it could sometimes happen with the original algorithm. This is why using the segmentation features to compare patches was useful in our approach. You can see some examples of our inpainting algorithm in figure 6 and 9.

Figure 4: Original image

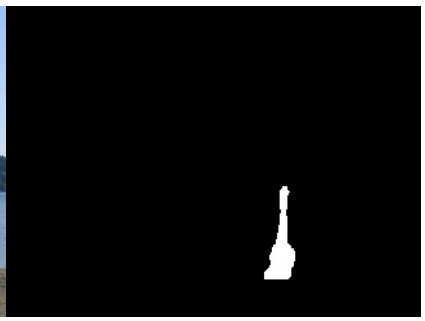

Figure 5: Mask

Figure 6: Inpainted image

Figure 7: Original image

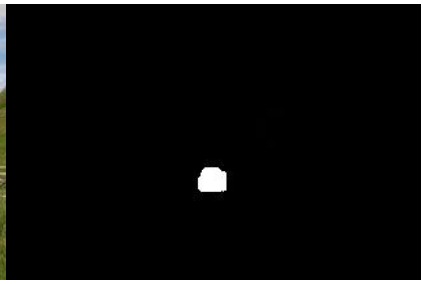

Figure 8: Mask

Figure 9: Inpainted image

4 Conclusion

We have presented an algorithm to detect the contour of an object selected by the user and remove it from the image in a visually pleasant way. For foreground detection, we applied and compared logistic regression and SVM for the learning process, and also adjusted the threshold parameter for decision boundary to trade off between the precision and recall rate. For image inpainting, our results always do at least as good as the original algorithm and improves it in some difficult cases where the original algorithm fails. We think that improving the results can not be done without using a more global approach, or without having access to higher level information, concerning context or objects surrounding the mask.

Acknowledgements

We would like to thank John Bauer, Geremy Heitz, Stephen Gould and Daphne Koller for their contribution to the inpainting algorithm. We would also like to thank Professor Andrew Ng and the TAs for their advice and suggestions.

References

[1] A.Criminisi and P.Perez. Object removal by exemplar-based inpainting. *Computer Vision and Pattern Recognition*, 2003.

[2] Stephen Gould, Andrew Y. Ng, and Daphne Koller. The stair vision library, http://ai.stanford.edu/ sgould/svl. 2008.

[3] Stephen Gould, Jim Rodgers, David Cohen, Gal Elidan, and Daphne Koller. Multi-class segmentation with relative location prior. *International Journal of Computer Vision (IJCV)*, 2008.