GrabCut and Semantic Segmentation

Tejas Sharma, Jiajun Xu, Youxin Chen, Jacob Irle

December 14, 2017

Introduction

The goal of our project was to implement Grabcut, which is a foreground extraction algorithm, and then to further utilize this tool in Semantic Segmentation, which, when trained with a set of pre-labeled images, can complete GrabCut without any human interaction at all for that class of objects. Our motivation for this project was to be able to quickly detect foreground in images with minimal user interaction.

1 GrabCut

This part of the project was completed by Tejas Sharma and Jiajun Xu

1.1 Outside Resources

- Our grabcut implementation is based on the paper " 'GrabCut' -- Interactive Foreground Extraction using Iterated Graph Cuts" by Carsten Rother, Vladimir Kolmogorov, and Andrew Blake.
- GrabCut is based upon the paper "Graph Cut", by Yuri Boykov and Marie-Pierre Jolly.
- The paper "Implementing GrabCut" by Justin Talbot and Xiaoqian Xu was also crucial in our implementation, as it explained the method in a much simpler manner, which increased our understanding significantly.
- We also used the "PyMaxFlow" library, which is a library for solving min-cut/max-flow in python. This library is based on the method described in "An Experimental Comparison of Min-Cut/Max-Flow Algorithms for Energy Minimization in Vision" by Yuri Boykov and Vladimir Kolmogorov. We initially created our own implementation based on the same paper, but found that it was too slow to be practically used due to being written in pure python. PyMaxFlow creates a wrapper for the C++ version, which makes it a lot faster.
- Our UI is heavily based upon the Python example available in the OpenCV github repo.

1.2 Implementation Details

GrabCut is based upon the Graph Cut algorithm, which is also used for image segmentation. The key difference between the two is that Graph Cut could only be used in black and white images, and required a lot more user interaction.

It uses iterative learning, which greatly reduces the amount of user interaction, but still allows the user to hardcode foreground and background pixels. At a high level, GrabCut attempts to minimize the energy between pixels in an image in order to find an optimal bounding cut. It iteratively improves this cut by recalculating the energy based on its previous classification until convergence.

1.2.1 Gaussian Mixture Models

GrabCut represents the foreground and the background of an image with 2 Gaussian Mixture Models, each of which has K gaussian components. Every pixel of the image belongs to one of the foreground or background component. Each gaussian component stores the mean (μ), the covariance matrix (Σ) and a weight (π). We use these gaussian components to calculate the likelihood that the pixel belongs to either the foreground or the background

We calculate the likelihood a pixel belongs to a GMM using equation 2 from “Implementing GrabCut”, which gives us a likelihood as the function of a pixel m

$$D(m) = -\log \sum_{i=1}^{K} \left[\pi(\alpha_m, i) \frac{1}{\sqrt{\det \Sigma(\alpha_m, i)}} \times \exp\left(\frac{1}{2} \left[z_m - \mu(\alpha_m, i) \right]^T \Sigma(\alpha_m, i)^{-1} \left[z_m - \mu(\alpha_m, i) \right] \right) \right]$$

where $z_m$is the colour of the pixel, and (α_m, i) essentially allows us to uniquely identify the cluster covariance and mean that we are trying to access.

1.2.2 Pixel Classifications

When creating the GMMs, we use the initial user input, which is a simple rectangle that is drawn around the foreground object, to create a set of classifications for the pixels. In “Implementing GrabCut”, they use a trimap of Foreground, Background and Unknown. However, when looking at the example code for our UI, we noticed that it uses four values: Background, Foreground, Potential Background, and Potential Foreground. We decided this made more sense to us, and would make implementation easier, so we also used this method. In the initial step, everything outside the rectangle is classified as Background, and everything inside it is classified as Potential Foreground. To note the difference in this report between the classification Foreground and the regular word foreground, we will be capitalising the classifications.

1.2.3 Graph Cut

To create the graph used in Graph Cut, we represent each pixel as a node in the graph. Each pixel is connected to its 8 adjacent pixels in the image (unless it is on one or more of the borders). We also add two terminal nodes for performing the max flow segmentation, one representing the source (background), and one representing the sink (foreground). The terminal nodes are connected to every pixel.

There are two types of edges in the graph: T-links, which are between a pixel and the two terminal nodes; and N-links, which are between a pixel and its 8 neighbours.

We build the N-links as follows:

- Each pixel is connected to its neighbours with the weight function

$$N(m, n) = \frac{1}{dist(m, n)} e^{-\beta \| z_m - z_n \|^2}$$

- $\beta = \left(2 \left\langle (z_m - z_n)^2 \right\rangle \right)^{-1}$, where $\langle\rangle$ represents an expectation over an image sample.
- The *dist* factor simply denotes whether the pixels are diagonal neighbours or immediate neighbours as noted in “Implementing GrabCut”, since the method in the original GrabCut paper favoured diagonal edges.

We build the T-links between the terminal nodes and the pixels by doing the following:

- First, we store the maximum weight created in the N-links. This will be used when a pixel has been clasified as Foreground or Background.
- If a pixel is classified as Foreground, we assign its foreground T-link to be the maximum weight, and its background T-link to be 0
- If a pixel is classified as Background, we assign its foreground T-link to be 0, and its background T-link to be the maximum weight
- If a pixel is classified as Potential Background or Potentail Foreground, we use the $D(m)$ function referenced earlier and assign its foreground T-link to be $D_{bgd}(m)$ and its background T-link to be $D_{fgd}(m)$.

After building the links, we run a max cut algorithm on it, and use the results to tell us which parts are foreground and background. Pixels that have been marked as Foreground or Background by the user are unchanged, and pixels that are not are updated to Potential Foreground and Potential Background based on the cut. These values are then fed back into the GMMs to iteratively run until convergence.

1.3 GrabCut Results

The following are the results of running our GrabCut implementation on some images that were used in the original papers.

The first is an image of a llama, which we decided worked pretty well without any secondary user input.

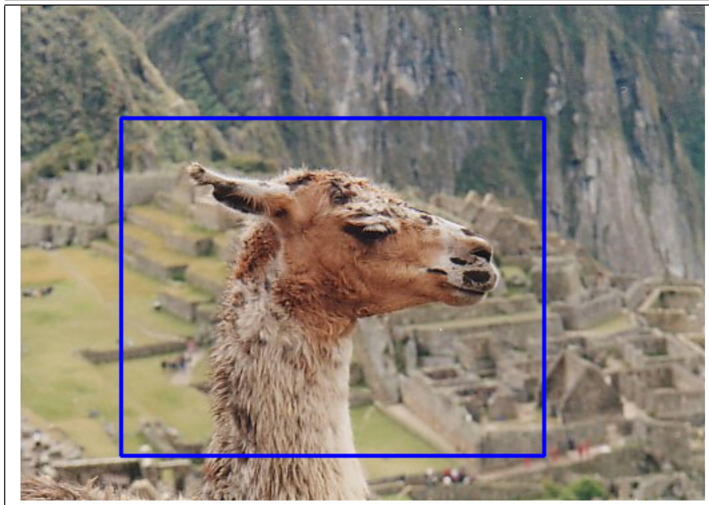
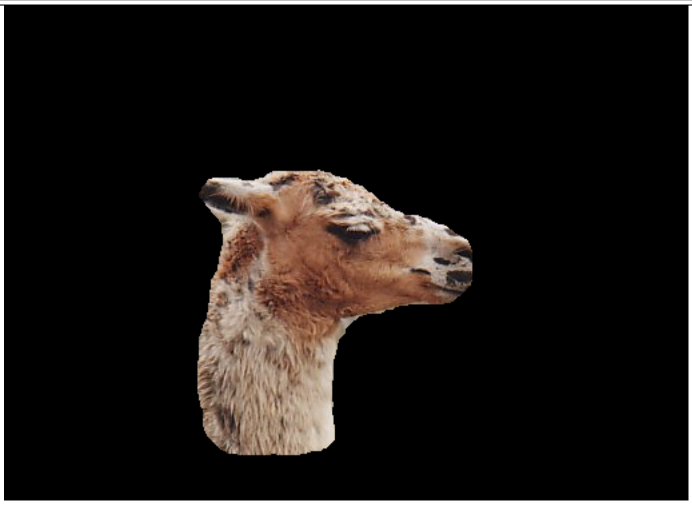
Original Image	GrabCut Output
	

Table 1: Llama Segmentation

The next is an image of Lena Söderberg, which allowed us to show off the ability for users to mark certain areas as Foreground or Background that were incorrectly classified by GrabCut.

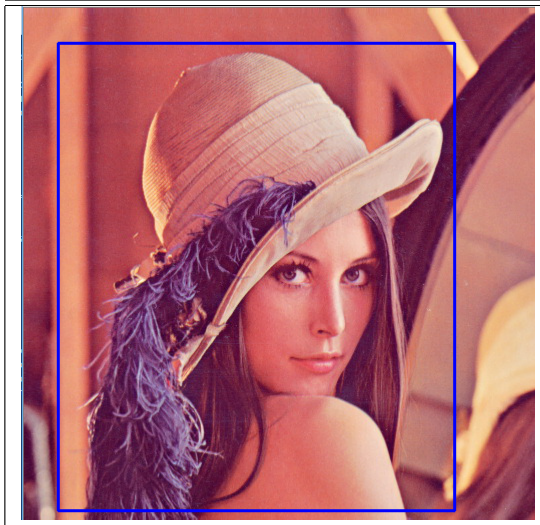
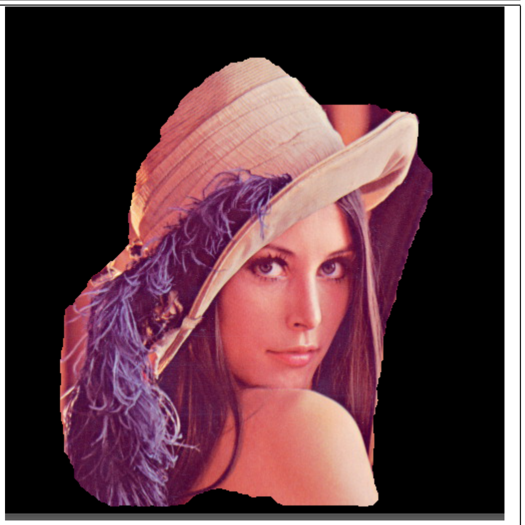
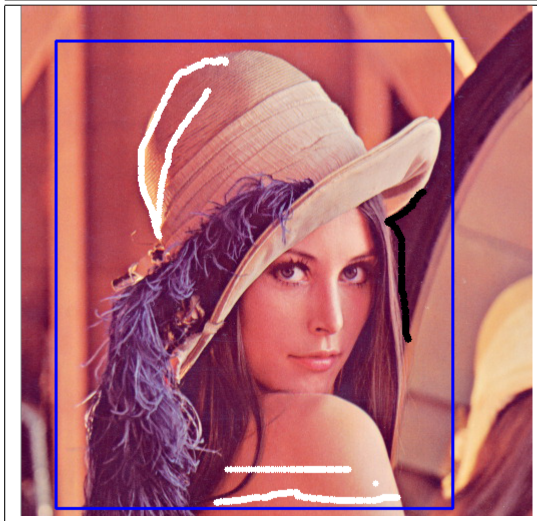
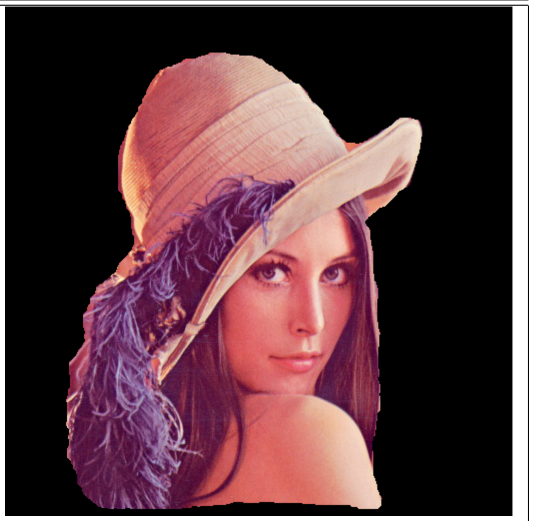
Original Image	Initial GrabCut Output
	
User Touchup	**Final GrabCut Output**
	

Table 2: Lena Segmentation

Finally, we have an image of a soldier. We were expecting this image to be really hard due to the similarity of colours between the foreground and background. Indeed, the initial result was not great. However, even with such similar colours, after some user input we had a pretty good result.

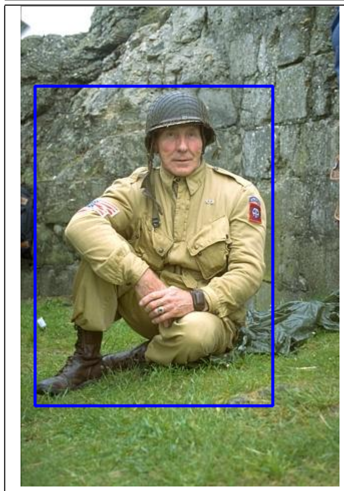
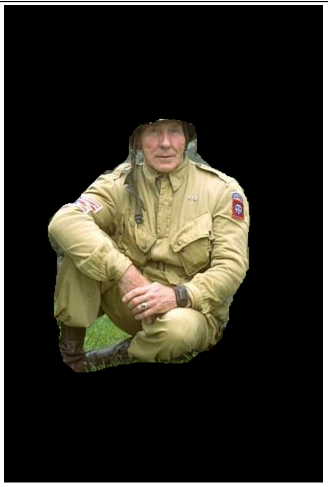
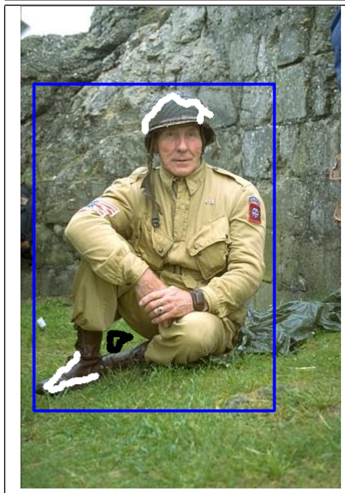
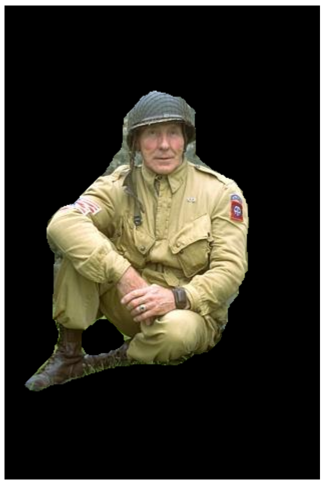
Original Image	Initial GrabCut Output
	
User Touchup	Final GrabCut Output
	

Table 3: Lena Segmentation

2 Semantic Segmentation

This part of the project was completed by Youxin Chen and Jacob Irle

2.1 Outside Resources

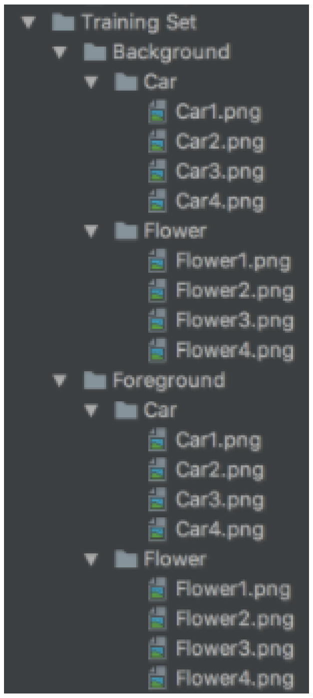
- Our Semantic Segmentation is based on "Semantic Segmentation using GrabCut" by Christopher Göring, Björn Fröhlich, and Joachim Denzler.
- The metric we use to decide the similarity between two patches is Hu Moments, based on the paper "Visual Pattern Recognition by Moment Invariants" by Ming-Kuei Hu.

2.2 Implementation Details

Semantic Segmentation uses GrabCut in order to complete GrabCut without any user interaction. It does this by essentially training a pair of Gaussian Mixture Models with a set of sample images of a particular class of object (e.g. cars, flowers), and then using those models in the GrabCut algorithm to get the foreground.

2.2.1 Training the GMMs

The assumption that we make while implementing Semantic Segmentation is that all objects of a particulat class can be modeled using Gaussian Mixture Models. So, in order to bootstrap the algorithm, we manually label the background and foreground of some images of certain classes. A simplified training set can be seen below.



```
▼ Training Set
  ▼ Background
    ▼ Car
        Car1.png
        Car2.png
        Car3.png
        Car4.png
    ▼ Flower
        Flower1.png
        Flower2.png
        Flower3.png
        Flower4.png
  ▼ Foreground
    ▼ Car
        Car1.png
        Car2.png
        Car3.png
        Car4.png
    ▼ Flower
        Flower1.png
        Flower2.png
        Flower3.png
        Flower4.png
```

Figure 1: Simplified Training Set

We use photoshop to generate the corresponding foreground and background images. Those pixtures are saved as png files, with the irrelevant section having zero opacity.

We then load all the pictures, and only use pixels with an opacity value greater than 200.

After doing this, we feed all foreground pixels of a class to one GMM, and the background pixels to another GMM. After creating the GMMs, we finetune the number of components down to 5. We first train a series of GMMs with a number of different components, and run those GMMs on a validation of the same class.

2.2.2 Creating a Segmentation

With the GMMs trained, we feed our test images to them, and label them as either foreground or background based on the following formula

$$\alpha_i^* = \underbrace{argmax}_{\alpha_i \in \{\text{fgd,bgd}\}} \; p\left(z_i | \alpha_i, k_i, \theta\right), \forall i \in \{1, \ldots, N\}$$

Where N is the number of pixels, z_i is the i-th pixel, and θrepresents the GMMS we trained for each class.

With this, we get an initial segmentation, which together with the GMMs and GrabCut, can be used to optimize the segmentation that will later be used for classification.

2.2.3 Classification

After GrabCut returns the optimized segmentation, we find the class which has the minimum distance to the segment. This is implemented by taking the distance between the Hu-moments of the segment and a given training set of images. The function shown below describes how we calculate the distance of a segment to a given class.

$$dist_h(\alpha_c, Z_c) = \underset{i=1\ldots N_c}{min} M(h(\alpha_c), h(\alpha^{i,c}))$$

In order for this function to make sense, we must have the Hu moments of the training images and the segment. We can obtain the Hu moments of an image by first finding the central moments of an image. The central moments of an image are defined below, where M_{xx} describes an already calculated moment

$$\begin{aligned}
\mu_{00} &= M_{00}\\
\mu_{01} &= 0\\
\mu_{10} &= 0\\
\mu_{11} &= M_{11} - \overline{x}M_{01} = M_{11} - \overline{y}M_{10}\\
\mu_{20} &= M_{20} - \overline{x}M_{10}\\
\mu_{02} &= M_{02} - \overline{x}M_{01}\\
\mu_{21} &= M_{21} - 2\overline{x}M_{11} - \overline{y}M_{20} + 2\overline{x}^2 M_{01}\\
\mu_{12} &= M_{12} - 2\overline{y}M_{11} - \overline{x}M_{02} + 2\overline{y}^2 M_{10}\\
\mu_{30} &= M_{30} - 3\overline{x}M_{20} + 2\overline{x}^2 M_{10}\\
\mu_{03} &= M_{03} - 3\overline{7}M_{02} + 2\overline{y}^2 M_{01}
\end{aligned}$$

After the central moments are found, we can use them to calculate the Hu moments which are used in our distance calculation. There are 7 invariant Hu moments which we will use in order to calculate the distance.

$$\begin{aligned}
I_1 &= \eta_{20} + \eta_{02}\\
I_2 &= (\eta_{20} - \eta_{02})^2 + 4\eta_{11}^2\\
I_3 &= (\eta_{30} - 3\eta_{12})^2 + (3\eta_{21} - \eta_{03})^2\\
I_4 &= (\eta_{30} + \eta_{12})^2 + (\eta_{21} + \eta_{03})^2\\
I_5 &= (\eta_{30} - 3\eta_{12})(\eta_{30} + \eta_{12})\left[(\eta_{30} + \eta_{12})^2 - 3(\eta_{21} - \eta_{03})^2\right] + (3\eta_{21} - \eta_{03})(\eta_{21} + \eta_{03})\left[3(\eta_{30} + \eta_{12})^2 - (\eta_{21} + \eta_{03})^2\right]\\
I_6 &= (\eta_{20} - \eta_{02})\left[(\eta_{30} + \eta_{12})^2 - 3(\eta_{21} + \eta_{03})^2\right] + 4\eta_{11}(\eta_{30} + \eta_{12})(\eta_{21} + \eta_{03})\\
I_7 &= (3\eta_{21} - \eta_{03})(\eta_{30} + \eta_{12})\left[(\eta_{30} + \eta_{12})^2 - 3(\eta_{21} - \eta_{03})^2\right] - (\eta_{30} - 3\eta_{12})(\eta_{21} + \eta_{03})\left[3(\eta_{30} + \eta_{12})^2 - (\eta_{21} + \eta_{03})^2\right]
\end{aligned}$$

After these are obtained we can then calculate the distance between two sets of hu moments as such:

$$M(H, H') = \sum_{i=1,\ldots,7} \left| \frac{\operatorname{sign}(H_i)}{\log|H_i|} - \frac{\operatorname{sign}(H_i')}{\log|H_i'|} \right|$$

Now that we have a way to calculate the distance between two sets of hu moments, we cans ee that the distance between a segment and a given class is described as the hu moment distance between the segment and a given class, which allows us to determine which class the segment belongs to by finding the class that gives us a minimum distance from the segment.

2.3 Results

2.3.1 Creating a segmentation

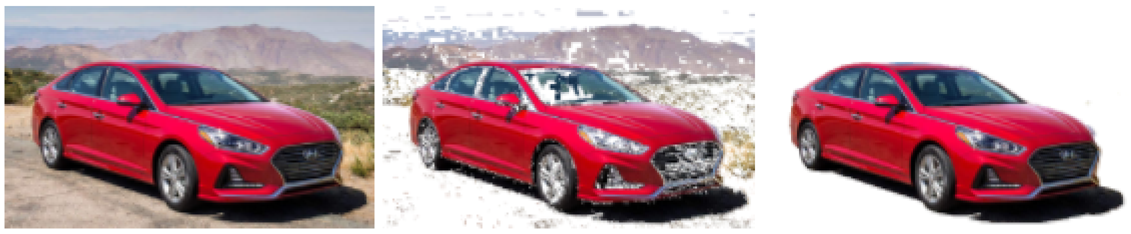


Figure 2: Semantic Segmentation on Car (Initial Image / After Initial Segmentation / After GrabCut Optimization

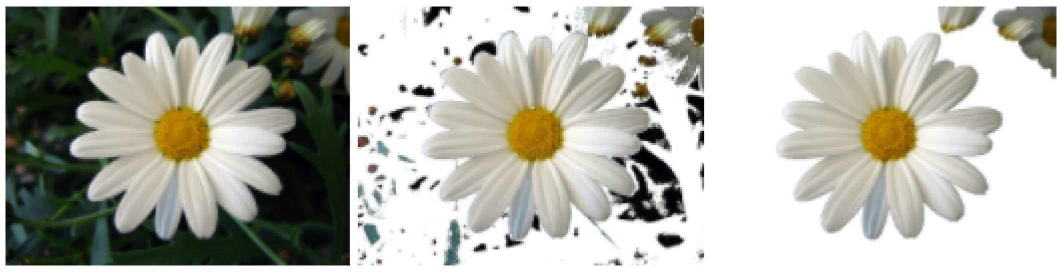


Figure 3: Semantic Segmentation on Flower (Initial Image / After Initial Segmentation / After GrabCut Optimization

2.3.2 Classification

For the below image, the results were:

- Distance to Car class: 0.0256119028253
- Distance to Flower class: 0.0311897888659

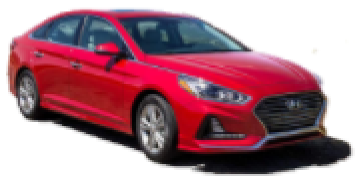


Figure 4: Car Classification

For the below image, the results were:

- Distance to Car class: 0.00651206406006
- Distance to Flower class: 0.00531785182462

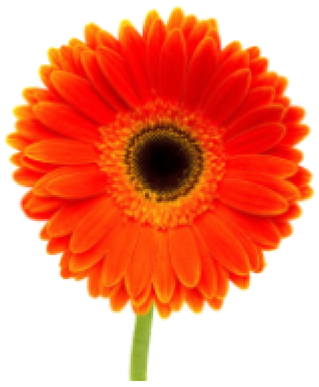


Figure 5: Flower Classification

As you can see, the classifying functionality of C-Grabcut works as intended, although the difference between the two distances is not as extreme as we initially thought it would be. This could be a result of our training set being fairly small for each class (4 images), which was due to the fact that we needed to manually construct the background and foreground images for each class. If we were able to automate that process, we may have been able to build stronger training sets that would yield more explicit results.