COMP4102 Project Report
Object Segmentation and Inpainting

Michael Kuang 101000485, Gordon Brown 101002159,
Kevin Sun 101000157, Maxim Kuzmenko 1010002578

April 15 2020

1 Abstract

This project will be to develop an application that will identify and remove detected object(s) from an image, while filling in the background that was in place of said object(s) naturally. The main challenges present with this project are object segmentation, and background interpolation. For object segmentation we need to detect objects within the imagine as well as crop it out of the image precisely. We will explore the various techniques in object detection to determine one that would best fit our application and possibly improve the algorithm to suit our needs.

2 Introduction

The program's objective is to detect objects from a given image, mask that specific object, then inpaint the masked area with the surrounding background such that the selected object is removed and what is left is the background. We implemented different object detection and image inpainting algorithms.

3 Background

The primary new libraries we used in our project are PyTorch, TorchVision, and Scikit-Learn. PyTorch is a machine learning library, TorchVision is a package in PyTorch that contains popular datasets, model architectures, and common image transformations in computer vision. We used these libraries for image recognition, manipulation, and classification in our project. We primarily used these libraries to make use of the pretrained Mask R-CNN model available.

We used the watershed segmentation algorithm and a well known state-of-the-art convolution neural network called Mask R-CNN for the object detection part of the project. We found that the watershed algorithm technique is great for images with few noises but struggle with complex images that are naturally noisy. However, Mask R-CNN works very well with detecting objects and segmenting them. CNNs in general can be configured to be effective object detectors because the model will learn what image features (image filters) are needed to describe the set of data that it will be trained on. Classical algorithmic approaches has one difficulty and it is that in order to segment objects we need to manually tune hyper parameters such that it would work favorably.

We implemented a biharmonic surface completion algorithm as one of the inpainting algorithms. One of the research papers we looked at was the biharmonic and harmonic analyzation by S. B. Damelin and N. S. Hoang from the University of Minnesota [8]. In the paper, they performed numerical experiments for image inpainting using the harmonic and biharmonic functions and go over comparisons between the two.

We found that while the infilling was fast, often times the resulting infill was wrong or could be differentiated from the rest of the image. It would be best used for images where the removed object had a smooth background, simple masks, or when edges are not prevalent. It is not effective at filling in complex images and colors, but it is fast in its computations.

We also implemented an Exemplar-Based Inpainting method as described by Criminisi's paper Object Removal by Exemplar-Based Inpainting. Their method is effective at inpainting complex images with a variety of patterns via the propogation of linear structures, henceforth called isophotes within the image. Their method uses a priority metric to identify which section of the mask to infill. The priority is based on a confidence term and data term which respectively refer to the amount of reliable information surrounding the patch and the strength of isophotes leading into the patch. This results in effective inpainting although at the cost of performance, inpainting a mask in a 512x512 image would result in run time of around 30 minutes.

4 Approach

4.1 Object Detection

Watershed Segmentation The watershed segmentation algorithm is a region-based method that assumes an image as a topographic landscape with ridges and valleys. With that assumption in mind, the elevation values of the landscape are defined by the gray values of the respective pixels or their gradient magnitude. For example, dark areas can be intuitively considered as lower in elevation, and can represent troughs while light areas can be considered as higher in elevation and can represent hills and ridges. The watershed transform decomposes the image into catchment basins, these are locations which all the water will drain to. For each local minimum, a catchment basin comprises all points whose path of steepest descent terminates at this minimum, consequently this will separate pixels to a region or a watershed [1]. Basically, if we flood the surface of the image from its minima, and we prevent the merge of the waters coming from different sources, we partition the image into catchment basins and watershed lines.

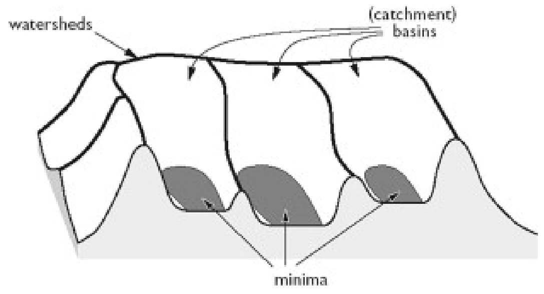

Figure 1: This figure depicts how watershed decomposes regions of images into certain minimums [1]

The watershed transform will assign each pixel to a particular region, but because the image is viewed as a topographic landscape, we can see that there will inherently be problems with noisy images or local irregularities in the gradient image which can lead to over-segmentation. The basic algorithm (Meyer's flooding algorithm) is defined as:

1. A set of markers (the minimas), pixels where the flooding shall start, are chosen. Each is given a different label.

2. The neighboring pixels of each marked area are inserted into a priority queue with a priority level corresponding to the gradient magnitude of the pixel.

3. The pixel with the highest priority level is extracted from the priority queue. If the neighbors of the extracted pixel that have already been labeled all have the same label, then the pixel is labled with their label. All non-marked neighbors that are not yet in the priority queue are put into the priority queue.

4. Redo step 3 until the priority queue is empty.

Once the algorithm terminates, the non-labeled pixels are the watershed lines and thus we have segmented the image with respect to the watershed lines.

Mask Regional Convolution Neural Network (Mask R-CNN) Classical image segmentation techniques are generally used for specific images that have inherent known properties. For example, the watershed segmentation algorithm works well with images that can be viewed as a topographic landscape, like medical images. However, the algorithm struggles when we want to detect people in a family photo or a dog. Another problem is that there are many hyper-parameters that need to be tuned for each task at hand and thus it is not a good choice to use watershed algorithm for a general set of images. In that case, convolution neural networks (CNN) can be used to detect objects in an image. Mask R-CNN is an extension of Faster R-CNN in which it adds a branch for predicting an object mask in parallel with the existing branch for bounding box recognition [2]. At a high level, Mask R-CNN consists of 4 main modules: the backbone, region proposal network (RPN), region of interest classifier and bounding box regressor, and segmentation masks. The backbone is a standard CNN that serves as a feature extractor. A CNN is a machine learning model that convolves input images with a multitude of image filters to extract features of an image. Then a fully connected layer is added at the end of the model in order to classify the image. The model will learn what the image filters should be through back-propagation and gradient descent, which is the training method that calculates the error and loss from prediction, back propagate the loss to each layer and update the weights (in this case the image filters are the weights at each layer) [7]. To understand how a CNN works, we explain the basic idea of how a neural network learns to become a classifier. A neural network in the most basic sense can be described as a linear function defined by $y = w^T x + b$ where w is the weight vector, x is the input and b is the bias. Intuitively, we understand that the dot product between two vectors is the similarity between them and thus the output of the function tells us how similar the weight matrix is with respect to the inputs. Initially, the weight vectors of a neural network at each layer are randomly assigned, and the job of the model is the learn the correct values for the weight vectors such that the output given the input is as close as possible to the expected label. For images, we use a CNN where the weight vectors are actually a set image filters. The output after convolving the image with these image filters produce feature maps that are then used as features in a neural network to classify the image. The model learns what image filters are necessary to separate a given dataset into their respective classes in order to minimize the cost function.

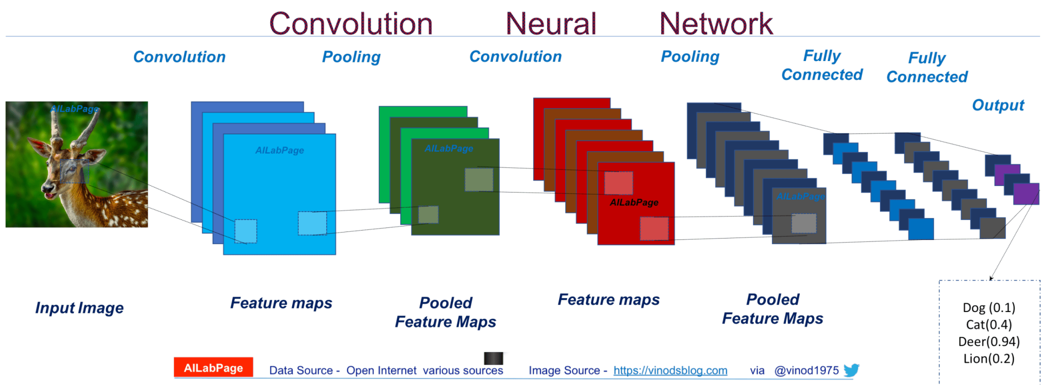

Figure 2: A depiction of a CNN workflow taken from [4]

The RPN is a simple neural network in which it convolves the image to find areas that contain objects. Specifically, the RPN convolves defined locations that are called *anchors* which are boxes distributed over the image area as shown in figure 3.

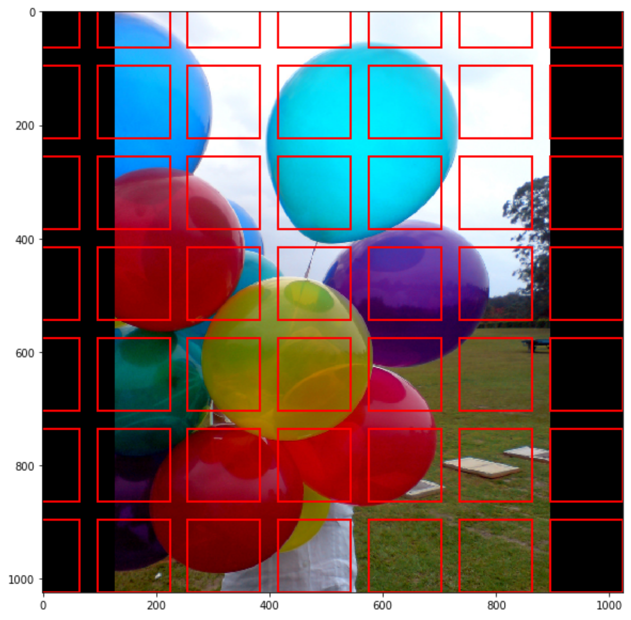

Figure 3: Image showing 49 anchors taken from [3]

In practice, there are many anchors with different sizes and aspect ratios, and they overlap to cover as much of the image as possible . The RPN takes in the results of the backbone feature maps as input (unlike how it was drawn in figure 3) and outputs the anchor and one of two classes associated with it: foreground or background. The foreground class implies that there is likely an object in that box. There may be many anchors that detect the same object in their box and many of those may not have the correct centering of the object, so the RPN estimates the change in x, y, height and width of these set of anchors to produce an anchor box that fits the object better [3].

Similar to the RPN, the ROI classifier generates two outputs: the specific class of the object (person, car, dog) and the bounding box. In order to classify an object from the outputs of the RPN, we first need to perform ROI pooling. Generally, CNN classifiers require fixed input sizes, however the the anchors produced by the RPN are all variable in size. ROI pooling fixes this problem by cropping and resizing the anchors to a fixed size, thus enabling a classifier to handle the image classification of this region of interest [3].

The last part is the segmentation mask and is the defining difference between Faster R-CNN and Mask R-CNN. This branch is a CNN that takes the regions selected by the ROI classifier and generates masks for them. The generated masks are low resolution (28x28 pixels) soft masks meaning they are represented by float numbers rather than binary numbers. Therefore, each pixel holds more detail and during the prediction phase, the soft mask is scaled up to the size of the bounding box to produce the final mask [3].

K-Means K-Means is a simple clustering algorithm that clusters similar colours together based on the number of provided cluster centers.[11] In image processing, this method may be used for colour quantization, in hopes that it would represent a colour body representative of the boundary of the objects of interest. K-means works by partitioning n data points into k clusters. Afterwards, each of the n data points will be designated to a cluster with the nearest mean. Data points inside a specific cluster would be considered more similar to each other than points outside that cluster and belong to other clusters. In our case, we want to find clusters such that we find objects of interest in the image. A way to do this is to cluster the pixel intensities of an RGB image; thus, after applying K-Means clustering, pixels that belong to a specific cluster will be more similar in colour that pixels outside the cluster. In our method, we additionally used blurring and OpenCV contour detection functions in order to increase the performance of object detection using this method.

Canny Edge Detector Canny edge detection is an multi-stage algorithm that can detect fine edges with noise suppressed at the same time.[12] First, this method applies noise reduction in order to remove spurious edges and to keep larger edges. Afterwards, it uses a gradient calculation step, in which it finds edge intensity and direction by using Sobel operators; this finds edge intensity for both the horizontal and vertical components of the image. At this step the edges are now highlighted. However, these edges vary in thickness; some are thin, some are large. To solve this problem, non-maximum suppression is applied in order to thin the edges. After this step, we may find that the edges still vary in intensity, so, double thresholding and edge tracking by hysteresis is then applied in order to transform all pixels above a certain intensity to the same intensity, and to get rid of pixels below a certain intensity. Ultimately, this results in detecting major contours in the image without unwanted edges that smaller have details. By infilling the insides of the major contours, we can then extract a mask that would hopefully result in detecting important/major objects in the original image.

4.2 Inpainting

Biharmonics An application of smooth surface completion is smooth image inpainting. In smooth image inpainting, there is a smooth image which is known in a neighbourhood outside of a region while the data inside is missing. The goal of image inpainting is to extend the function over the region in such a way that the extension over the missing region is not noticeable with human eyes. In image inpainting, an inpainting scheme is considered linear order if for any smooth test image, as the diameter of the inpainting region shrinks to 0, one has:

$$|u - u_o| = O(d^2), \tag{1}$$

where u is the image obtained from the inpaint.

The biharmonics equation is a 4th order partial differential equation that is relevant in applied mechanics, such as as elasticity and fluids [10]. The implemented biharmonics algorithm works by first separating the mask into independent regions: a dilated mask using the binary dilation function, and a labeled mask where two pixels are considered connected when they are neighbors and have the same values. The labeled mask is then iterated over and used to generate a new matrix, and then the right hand side of the matrix is computed using as a sum of the known matrix columns. Solving this linear system results in obtaining the masked points, which can then be substituted for their inpainted versions.

In the research paper [8], the authors implemented the biharmonics equation by solving the Poisson equation:

$$\Delta u = f_1 u_s = g_1 \tag{2}$$

which is then reduced to:

$$Au = (f_1 - \frac{g_1}{h^2}) \tag{3}$$

where g is a vector containing the boundary values of the image u, and A is a tri-diagonal matrix; that is all nonzero elements are on the main diagonal and the first diagonals above and below the main diagonal.

One of the pros of the biharmonic algorithm is that while it may not always be accurate in the infill, it is extremely fast. With images of over 1024x768 pixels, computations will often take under a minute to finish. If the image mask does not require complex filling, biharmonic can be efficient and effective for infilling.

Exemplar-Based Image Inpainting The core of this Exemplar-Based Inpainting algorithm is the use of isophotes in the image sampling process. [9] Prior to Criminisi's technique, researchers used texture synthesis as a way to fill large image regions with a repetitive 2D pattern. Example based techniques are used to effectively generate and identify textures by sampling and copy regions from the source. While these techniques are effective when the background is uniform they struggle with real world pictures where the background is not a uniform pattern. The issue with natural photos is that boundaries between the mask and the source will be composed of multiple textures instead of one prevailing pattern.

Criminisi cites various other techniques, such as Harrison's and Zalesny's, which are both ruled out as effective means of inpainting due to excessive blurring in the final image. Criminisi's demonstrates how the two prior methods are negatively influenced by noise resulting in blurry or inaccurate images.

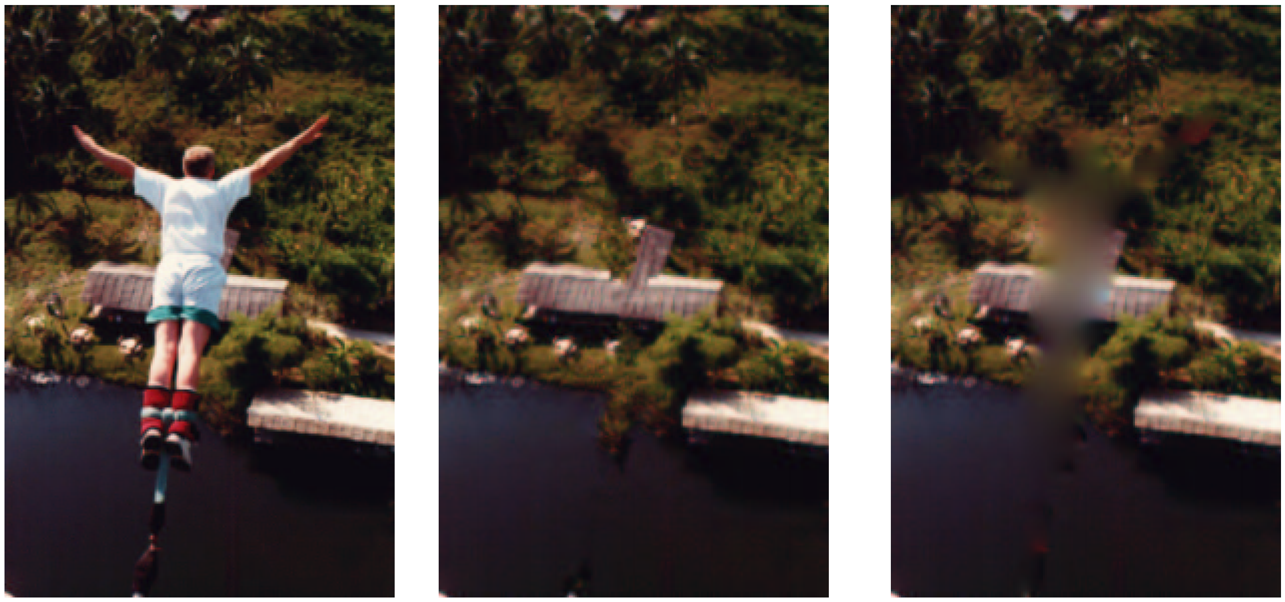

Figure 4: A Comparison of Criminisi's technique to traditional inpainting [9]

Regarding the specifics of Criminisi's algorithm a user provides a target region (or a mask) to be removed and filled. The source region is the remainder of the image. Following this as with other exemplar-based algorithms the size of the template window must be specified, for this project we went with the default 9x9 window, but Criminisi advises that it be set to slightly larger than the largest texture element in your image. Once these parameters are set the rest of the processes is autonomous. Each pixels to be filled is given a color and a confidence value which reflects our confidence in the pixels color value. As the algorithm progresses patches, the size of our window will be given a temporary priority value, which determines the order in which they are filled. With this the algorithm follows three steps until all pixels are filled:

1. Computing patch priorities For each possible patch to fill we must compute it's priority to determine which patch we are most confident in filling. This is done using two criterias: a confidence term and a data term.

$$P(p) = C(p)D(p) \tag{4}$$

Where C is the confidence term and D is the data term as define below:

$$C(p) = \frac{\Sigma q \epsilon p(C(q))}{|p|} \tag{5}$$

$$D(p) = \frac{|\Delta I_p^{\perp} * n_p|}{\alpha} \tag{6}$$

Confidence is the sum of the confidence of all the pixels in the surrounding area. —p— is the size of the patch around p, q is a pixel around p. Data is a function of the strength of the isophotes (patterns) hitting the front between the mask and source. α is some normalization factor, we used 255 as suggested in the paper, n_p is a unit vector orthogonal to the front between the point P and the source.

2. Propagating texture and structure information Once we calculate all of the priorities we find the patch with the highest priority and fill it with the data from the source region. traditionally this step is done using diffusion which results in blurry images. Instead we propagate images by direct sampling with the source image.

$$\Psi_{q*} = \arg \min \psi_q \epsilon \Phi d(\Psi_{q*}, \Psi_q) \tag{7}$$

Where $d(\Psi_{q*}, \Psi_q)$ is simply the sum of squared differences between the already filled pixels in the two patches. Ψ_{q*} and Ψ_q are the patches around p^* and q. and Φ is the source space in the image. Essentially we search through the entirety of the already filled in image to find the best fitting patch to copy in. This ensures that the selected path to use has a high rate of conformity with the existing image.

3. Updating confidence values Once will fill the selected region around p, we must update the confidence values around said region:

$$C(q) = C(p^*) \forall q \epsilon \psi_{p^*} \tag{8}$$

This update rule causes confidence to decay as you go further towards the center of the target region in the mask. This is desired as in general pixels near the edge of the mask are easier to predict than those towards the center.

Compared to Biharmonic and other approaches mentioned by Criminisi this algorithm was very effective disparity a long run time. For future work it would be necessary to determine the source of our long computation time (compared with Criminisi supposed ¡1 minute run time).

5 Results

5.1 Object Detection

Watershed Results As mentioned before, watershed algorithm assumes a topographic landscape and as such it works well with segmenting objects with images that have those intrinsic properties. However, it fails with complex images that have noisy elements and are non-topographic in nature. We testing the algorithm on two different images as shown in figure 5 and figure 6 using the exact same parameters. As you can see, the coins are segmented well because the image is simple with few noise and can be naturally viewed in a topographic setting. On the other hand, it fails to segment the person and dog objects in the landscape image in figure 6.

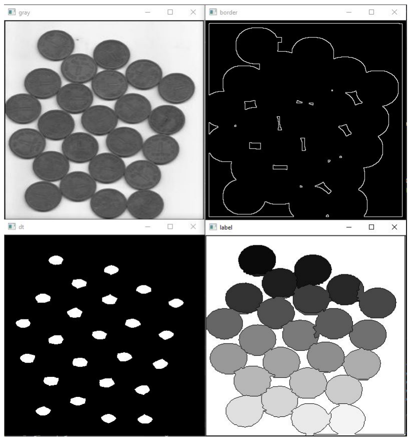

Figure 5: Intermediate results and the final segmented objects using watershed segmentation algorithm on an image with coins

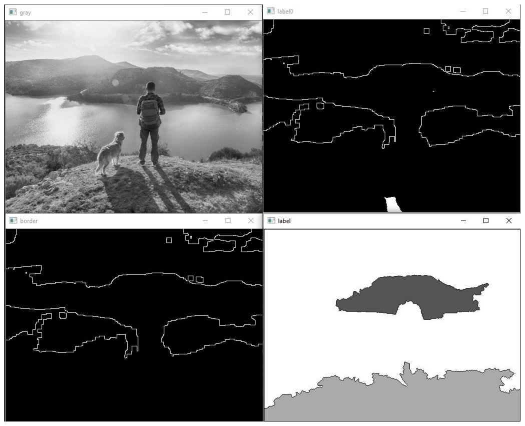

Figure 6: Shows the intermediate results and the final segmented objects using watershed segmentation algorithm on a landscape type image with a person and a dog

Mask R-CNN Results The Mask R-CNN was able to precisely produce masks for classes that it was trained on. The masks were well defined as can be seen in figure 7. One error to note is that it misclassified a patch of vegetation as a car and this is one of the downsides to using machine learning algorithms for image segmentation. Despite all this, we found that the Mask R-CNN is the best method to detect objects and generate the corresponding masks for these objects.

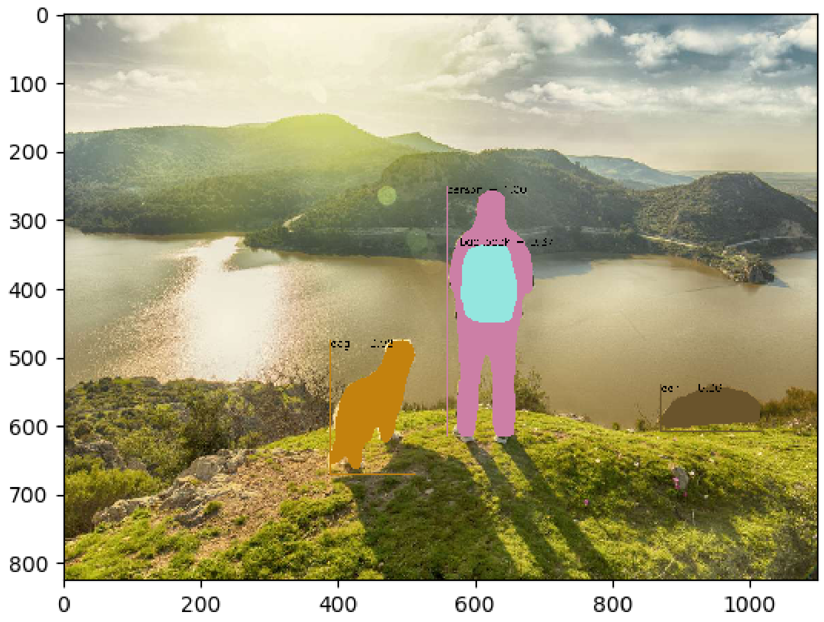

Figure 7: Masks predicted from the Mask R-CNN, the outputs include a dog, backpack, person and a incorrect classified car

K-Means Results We found that this method performs with considerable accuracy when there are distinct colour differences between objects of interest and the background. However, other general objection detection algorithms such as RNN and Waterfall outperform this method when it comes to object detection in a similar color environment; many undesirable/irrelevant detection may also result.

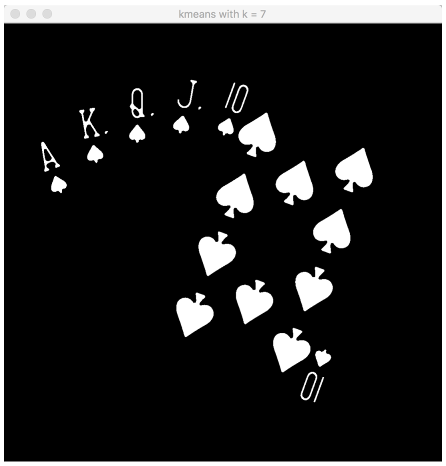

Figure 8: Mask achieved by running K-means with 7 clusters (k=7) on a deck of cards. Accurate mask was achieved due to distinct color differences in the image

Canny Edge Detector Results We found that while this method does accurately return image edges/contours, it fails when it comes to filling the inside of the contours in order to produce a mask because of various small discontinuities along the contours. The upshot is that this method does successfully showcase contours of desired objects, as seen in the figure below.

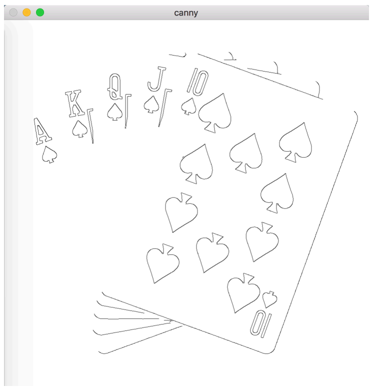

Figure 9: Canny image contours

5.2 Inpainting

Exemplar-Based Image Inpainting Results One downside of this Criminisi's algorithm is the long compute time we experienced. A standard 512x512 image would take around 30 minutes to infill, which compared to Criminisi's paper is terrible: he cited times in the seconds, where we experienced times in the tens of minutes. It is possible however that we implemented something incorrectly resulting in this disparity.

Criminisi's inpainting algorithm works very well at working with images that have strong patterns. This can be seen in figure 10. The sharp line between the black and gray sections was preserved.

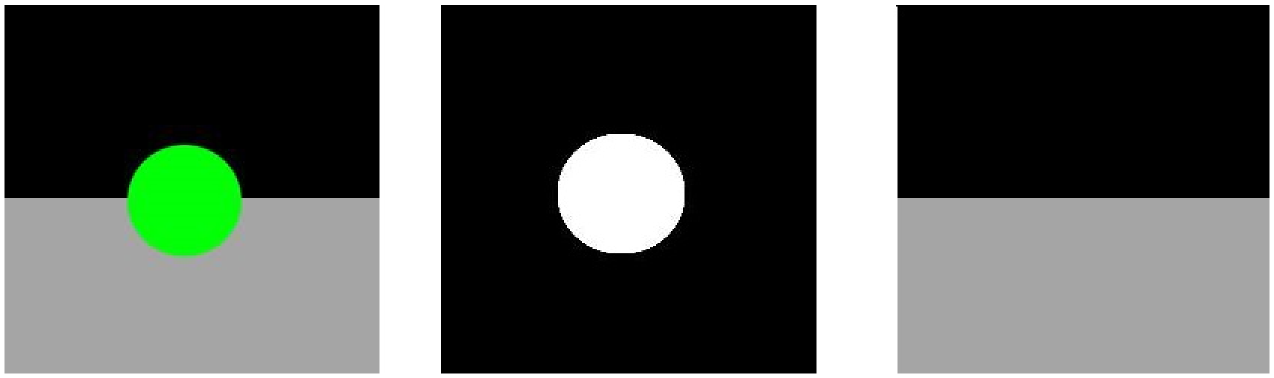

Figure 10: Simple object removal, source, mask, final image

In Figure 11 we see a different story where the shore is incorrectly extended into the lake. This is due to the noise involved with the shoreline. However we paint a fairly convincing final image outside of the extended shoreline. This is contrasted with the image we saw earlier in figure 4 where there was unnatural blurring.

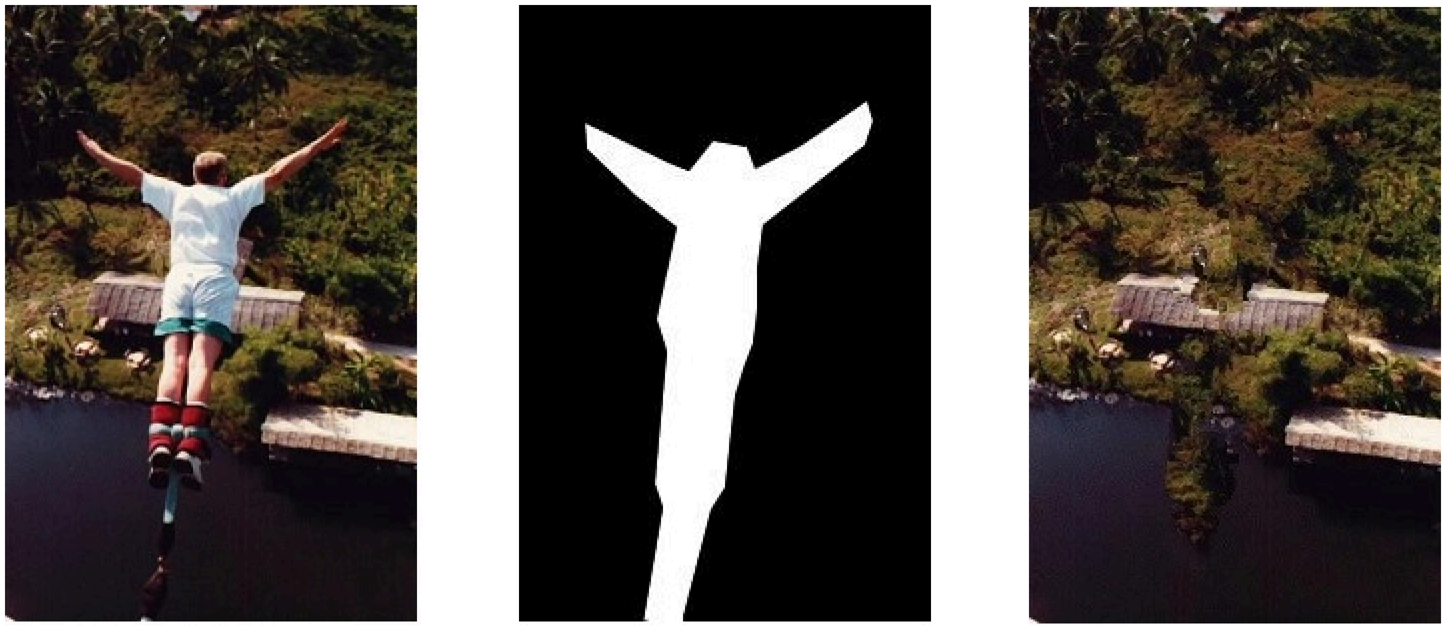

Figure 11: Complex object removal, source, mask, final image

The removal of the person in Figure 12 is very convincing due to the sharp contrasts and lines between the person and the water. In this image the shoreline is a much sharper line allowing for the infill algorithm to correctly infill without unnaturally extending the shore.

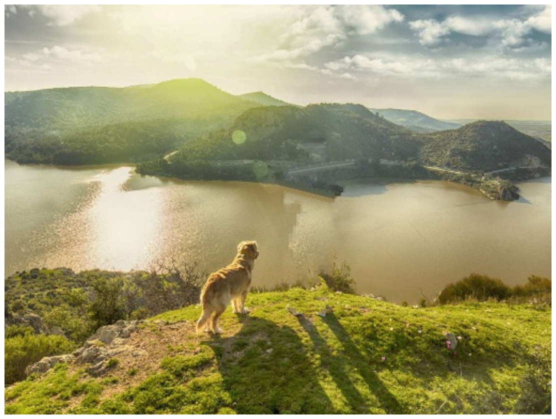

Figure 12: Criminisi's exemplar-based inpainting using the person mask produced from the Mask R-CNN

The removal of the person in figure 13 is almost a success. Due to the lack of patterns in the persons shirt it was unable to detect patterns within it and incorrectly filled in part of the bag with water. With a more ideal shirt it clear that the algorithm would be much more successful at inpainting the remainder of the person.

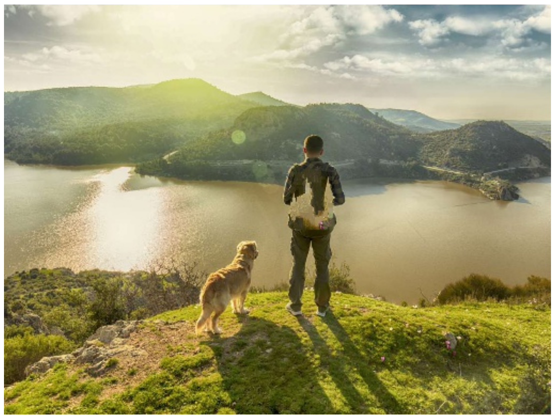

Figure 13: Criminisi's exemplar-based inpainting using the bag mask produced from the Mask R-CNN

Biharmonic Results Using the biharmonic algorithm to infill images is most effective when the image mask covers very smooth portions of the image; that is the parts covered are also very similar in color and do not contain edges or variations. In cases like these, the infill becomes accurate, only being noticeable in some cases. This is because the biharmonic algorithm effectively smooths the mask into a variety of colors based on its neighbors. When the mask is over a smooth image, a smooth infill will look natural and proper.

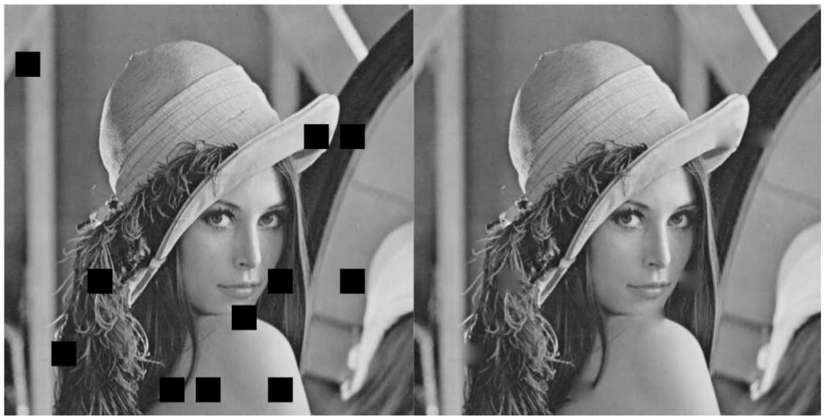

Figure 14: Left: Damaged Image. Right: Biharmonic Inpainting

If one is not paying enough attention or if the image is small enough, certain unnatural colors and effects will hardly be noticeable. In the above example, the masks over the lady's hair are the most apparent to look unnatural, but because each individual mask is small, as well as the image not being large itself, it can be difficult to locate the errors if one weren't told that there were errors, or if they did not see the original or damaged image.

However, if the mask is over an object with complex colors, edges, or shapes, such as a person's face, a house, or an animal, the algorithm will not be able to accurately fill in the mask with what we would want it to.

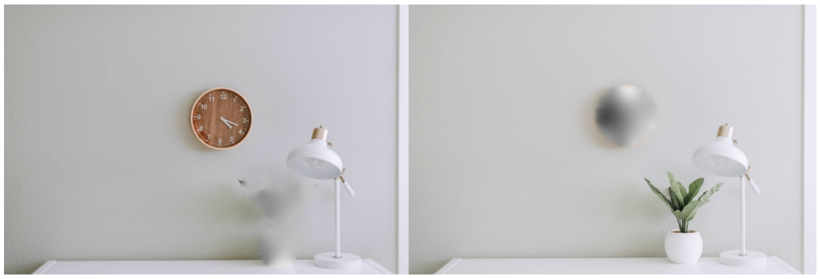

Figure 15: Left: Infill on Potted Plant. Right: Infill on Clock

In the above two examples, the algorithm filled in the mask with the correct colors, however there is still visible distortion and colors that are off from the rest of the surroundings. With more complex images, colors, and objects, the biharmonic method may be less likely to succeed.

6 Conclusion

Overall, the project can be considered a success because we researched implemented multiple object segmentation and image inpainting algorithms. Some of these methods may have advantages over others, but also have some disadvantages. It is important to measure the pros and cons of each method to determine when it would each method should be used for the appropriate situation. With further research, development, and implementations, others can further improve these algorithms or develop new ones, and this type of idea can be developed into a full scale project.

7 List of Work

Equal work was performed by all project members.

8 GitHub Page

The GitHub page for our project can be found at https://github.com/Krusso/COMP4102TermProject

9 Appendix

References

[1] Visual Computing for Medicine, 2013, Bernhard Preim Charl Botha
`https://www.sciencedirect.com/topics/computer-science/watershed-segmentation`

[2] Mask R-CNN, 2017, Kaiming He, Georgia Gkioxari, Piotr Dollar, Ross Girshick
`https://arxiv.org/abs/1703.06870`

[3] Splash of Color: Instance Segmentation with Mask R-CNN and TensorFlow, 2018, Waleed Abdulla

https://engineering.matterport.com/splash-of-color-instance-segmentation-with-mask-r-cnn-and-tensorflow-7c761e238b46

[4] Deep Learning Introduction to Convolutional Neural Networks, 2018, V Sharma
https://vinodsblog.com/2018/10/15/everything-you-need-to-know-about-convolutional-neural-networks/

[5] Watershed (image processing), 2020,
https://en.wikipedia.org/wiki/Watershed_(image_processing)

[6] IMAGE SEGMENTATION AND MATHEMATICAL MORPHOLOGY, 2010, Serge Beucher
http://www.cmm.mines-paristech.fr/~beucher/wtshed.html

[7] Part 2: Gradient descent and backpropagation, 2018, Tobias Hill
https://towardsdatascience.com/part-2-gradient-descent-and-backpropagation-bf90932c066a

[8] On Surface Completion and Image Inpainting by Biharmonic Functions: Numerical Aspects, 2018, S. B. Damelin and N. S. Hoang.
https://www.hindawi.com/journals/ijmms/2018/3950312/

[9] Object Removal by Exemplar-Based Inpainting, 2003, A. Criminisi and P. Perez
https://www.microsoft.com/en-us/research/wp-content/uploads/2016/02/criminisi_cvpr2003.pdf

[10] Biharmonics Equation, Science Direct
https://www.sciencedirect.com/topics/engineering/biharmonic-equation

[11] Introduction to Image Segmentation with K-Means Clustering, Towards Data Science https://towardsdatascience.com/introduction-to-image-segmentation-with-k-means-clustering-83fd0a9e2fc3

[12] Canny Edge Detection. OpenCV opencv-python-tutroals.readthedocs.io/en/latest/py_tutorials/py_imgproc/py_canny/py_canny.html