COMPARISON OF IMAGE RETARGETING ALGORITHMS WITH SEAM CARVING METHOD

TAYLAN MİROĞLU

IŞIK UNIVERSITY
APRIL, 2023

COMPARISON OF IMAGE RETARGETING ALGORITHMS WITH SEAM CARVING METHOD

TAYLAN MİROĞLU
Işık University, School of Graduate Studies, Computer Engineering Master's Program,
2023

This thesis has been submitted to Işık University, Graduate Education Institute for a Master's Degree (MA).

IŞIK UNIVERSITY
APRIL, 2023

IŞIK UNIVERSITY
SCHOOL OF GRADUATE STUDIES
COMPUTER ENGINEERING MASTER'S PROGRAM

COMPARISON OF IMAGE RETARGETING ALGORITHMS WITH SEAM CARVING METHOD

TAYLAN MİROĞLU

APPROVED BY:

Dr. Emine Ekin (Thesis Supervisor)	Işık University
Dr. F. Boray Tek	İTÜ Yapay Zeka ve Veri Mühendisliği Bölümü
Dr. Ayşegül Tüysüz Erman	Işık University

APPROVAL DATE: 25.04.2023

COMPARISON OF IMAGE RETARGETING ALGORITHMS WITH SEAM CARVING METHOD

ABSTRACT

The rise of social media has made sharing photos and pictures more important than ever, both for personal and marketing purposes. This situation also caused the problem of converting the photos taken with the camera in a square format, where the width is higher than the height. To address this need, a recent study explored the use of the Seam Carving method to convert images to a square format while preserving their essential parts. The study compared two algorithms, Greedy and Dijkstra, in terms of processing time and consistency using a supervised image.

The consistency comparison was carried out on five images, three of which were obtained from NRID, and two were created for the study. The five images were used to calculate the average consistency of the Dijkstra algorithm. In addition, 23 more images from NRID were used to compute the average consistency of the Greedy algorithm, resulting in a total of 28 images used in the analysis.

The results showed that the Greedy algorithm had an average consistency that was 6.55% higher than the Dijkstra algorithm based on the five images. Furthermore, the Dijkstra algorithm took an average of 2,347% longer to process than the Greedy algorithm.

The implications of these findings are significant for social media users and marketers alike. The Greedy algorithm can help maintain the essential elements of an image while making it suitable for different social media platforms. The study also highlights the importance of considering processing time when choosing an algorithm to use. Overall, this research demonstrates the potential of the Seam Carving method and provides valuable insights into the choice of algorithm for image manipulation.

Keywords: Seam Carving, Dijkstra, Greedy, Image Retargeting, Image Resizing, Shortest Path

SEAM CARVING YÖNTEMİ İLE GÖRÜNTÜ YENİDEN HEDEFLEME ALGORİTMALARININ KARŞILAŞTIRILMASI

ÖZET

Sosyal medyanın yükselişi, kişisel ve pazarlama amaçları için fotoğraf ve resim paylaşımını daha da önemli hale getirdi. Bu durum aynı zamanda, kamera ile çekilen ve genişliği yüksekliğinden daha fazla olan fotoğrafların kare formata dönüştürülmesi sorununu da beraberinde getirdi. Bu ihtiyacı karşılamak için son zamanlarda bir çalışma, resimleri özgün parçalarını koruyarak kare formata dönüştürmek için Seam Carving yönteminin kullanımını inceledi. Bu çalışmada, süpervize edilmiş bir görüntü üzerinde hem işlem süresi hem de tutarlılık açısından Greedy yaklaşım ve Dijkstra algoritması olmak üzere iki algoritma karşılaştırdı.

Bu araştırmadaki tutarlılık karşılaştırmasında beş görüntü kullanıldı; üç tanesi NRID'den elde edilen ve iki tanesi bu çalışma için özel olarak oluşturulan beş görüntü üzerinde yapıldı. Beş görüntü, Dijkstra algoritmasının ortalama tutarlılığını hesaplamak için kullanıldı. Bunun yanı sıra, NRID'den 23 tane daha görüntü, Greedy algoritmasının ortalama tutarlılığını hesaplamak için kullanıldı. Bu araştırmanın analizinde toplamda 28 görüntü kullanıldı.

Sonuçlar, beş farklı görüntüye dayanarak Greedy algoritmasının ortalama tutarlılığının Dijkstra algoritmasından %6,55 daha yüksek olduğunu gösterdi. Bunun yanı sıra, Dijkstra algoritmasına ait işlem süresinin Greedy algoritmasından %2.347 daha uzun sürdüğü ortaya çıktı.

Bu bulguların sosyal medya kullanıcıları ve pazarlamacılar için önemli sonuçları vardır. Greedy algoritması, bir görüntünün temel öğelerini koruyarak farklı sosyal medya platformlarına uygun hale getirmeye yardımcı olabilir. Bu çalışma, görüntü yeniden boyutlandırma yöntemlerinden olan Seam Carving yönteminde algoritma seçiminde işlem süresinin dikkate alınmasının önemini vurgulamaktadır. Genel olarak, bu araştırma, Seam Carving yönteminin potansiyelini göstermektedir ve görüntü manipülasyonu için algoritma seçimi konusunda değerli bilgiler sağlamaktadır.

Anahtar Kelimeler: Seam Carving, Dijkstra, Greedy, Resim Yeniden Hedefleme, Resim Boyutlandırma, En Kısa Yol

ACKNOWLEDGEMENTS

I would like to thank my family and instructors who have been with me throughout this process.
Also
I would also like to thank my partner Özge Karaboğa, who played an active role in the creation of this study.

Taylan MİROĞLU

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

CHAPTER 1

1. INTRODUCTION

Images are crucial in social media since they let you interact with your audience more. More of you is visible to your fans, who can also follow your actions. Images may convey a message or tell a narrative much more effectively than words ever could (Ellard, 2021).

Most of social media web sites suggest square dimensional images (Arens, 2022), although most of cameras take rectangle dimension photos (Must Photos Always Be Rectangular?, 2020). The proposed of image retargeting algorithms are meet needs of modern digital multimedia technologies improvements and solve different type of devices resolution and aspect ratio with that aim the retargeting process apply displaying images for obtain without distortion on media of these various sizes using different standards. Image retargeting technique resizes an input image to a given target resolution where the aspect ratio changes. For this aim so many retargeting techniques were found. These techniques can be classified with two directions, first one is brute force and second one is content-aware retargeting (Rubinstein et al., 2010). Most well-known brute force approaches are scaling and fixed window cropping and content-aware retargeting approaches are cropping based, segmentation based, patch based, seam carving, warping and multi-operator approaches (Rubinstein et al., 2010).

1.1 Application of Image Retargeting Algorithms

Laplacian transform, and seam carving method with greedy approach and Dijkstra have been applied to an image. Laplacian is a derivative operator; it uses highlights gray level discontinuities and so diminishes one important region with slowly varying gray levels (Abramowitz et al., 2013). Seam carving is one of the most famous image retargeting approaches which decreases an image’s width one pixel at a time with a seam which is a connected path with the crossing of lower energy pixels of an image top to the bottom or left to the right (Avidan et al., 2007). Part of the seam carving retargeting approach involves finding the shortest path, which brings along the greedy and Dijkstra algorithms. The greedy algorithm checks only the first neighbors of the pixel, although the Dijkstra algorithm checks all possible paths like brute force, but it's a faster version of brute force (Sniedovich et al., 2006); I will mention this in the next chapters.

1.2 Contributions

Default greedy approach, directly checks all the possible pixels. This process causes long image retargeting processing time and recalculation of unnecessary paths. For this reason, I made an optimization on the default greedy approach. According to this optimization process, only the 1-pixel width path with the lowest energy sum determined in the previous process, when deleted from the picture, the other paths that were affected are calculated. As a result, instead of the number of paths that will be calculated as the width of the picture in each process according to the default greedy approach, the optimized greedy approach calculates the number of paths as the height of the picture in the worst-case scenario. Also, it's only one path was recalculated in the best-case scenario. This causes the algorithm to use fewer resources and shorten the processing time considerably. I will explain the optimization I made, on the chapter 3.2.4.1 First-level Greedy Approach.

CHAPTER 2

2. LITERATURE SURVEY

Image retargeting has become increasingly important in recent years due to the rise of mobile devices with varying screen sizes and aspect ratios. This technique enables images to be resized without altering the important features or distorting the image's overall quality. Various methods have been proposed for this purpose, such as cropping, scaling, and seam carving.

Seam carving, also known as content-aware image resizing, is a relatively new method that has gained popularity due to its ability to retain the essential features of an image while resizing it. The algorithm identifies and removes the least important seams from the image, resulting in a smaller but visually appealing image. A number of researchers have focused on developing and improving the seam carving algorithm.

One of the most significant contributions to this field was made by Avidan and Shamir (Avidan et al., 2007), who proposed the first seam carving algorithm. Their method uses dynamic programming to identify and remove the least important seams from the image. They demonstrated that their method produced better results than cropping or scaling.

Later, Rubinstein et al. (Rubinstein et al., 2008) proposed a modified version of the algorithm, which introduced a new energy function and made the method more efficient. They showed that their approach produced visually appealing results while reducing computation time.

Another study by Chiang et al. (Chiang et al., 2009) proposed a GPU implementation of the seam carving algorithm. They demonstrated that the use of a GPU significantly improved the speed of the algorithm while maintaining the same level of accuracy.

In recent years, deep learning techniques have also been applied to image retargeting. For example, Wu et al. (Wu et al., 2019) proposed a deep convolutional neural network (CNN) for image retargeting. Their approach learns to map an input image to an output image of a different size while preserving the content of the original image. They demonstrated that their approach achieved better results than traditional methods.

In another study, Singh et al. (Singh et al., 2020) proposed a novel method for image retargeting that uses a convolutional neural network with spatial attention mechanisms. They use a layer in the CNN to resize feature maps of the image, while ensuring that important regions are preserved during the resizing process.

Overall, these studies demonstrate the importance of image retargeting and the various methods that have been proposed to achieve this goal. The development of seam carving algorithms, as well as the recent application of deep learning techniques, have significantly improved the accuracy and speed of image retargeting.

CHAPTER 3

3. METHODS & PROCESSES

3.1 Auxiliaries, Libraries and Language

Since it's powerful in image process Python 3.9 was used as a programming language and PyCharm was my preferred IDE for this research (Muhammad, 2021). A total of 4 Python files, two main .py files, two auxiliary .py files, and a total of four classes were used. Also, this project includes thirty-five class methods, twenty-one necessary methods, and four optional methods in the project. For reaching my aim I created my own methods for whole algorithm steps. Although the internet is used for help at some points, every line except Dijkstra has been specially written by me and I preferred to use built-in libraries of Python as much as I can, instead of using external libraries. Used libraries of the project are Matplotlib, NumPy, DateTime (optional), time(optional), sys, and copy. Also I used the Shapely library in a separate program I wrote to compare the results.

3.2 Steps

After the dataset prepared, the basic process steps of the project are: Selected 3-dimensional colored image is converted to 2-dimensional gray level image and this gray leveled image is used as input for Laplacian transform algorithm. The energy points of the image are determined by this algorithm and marked on a Python list that is the same length as the 2D version of the original image. The process after this step will fork towards two separate points, then they will merge again. One of the processes uses greedy approach for determining shortest path by using energy points, and the other process tries to do same thing with using Dijkstra algorithm. The pixels of the shortest paths of both methods are marked with the RGB color value of 255.0.0, which represents full red. After that these marked points are removed from the image to complete retargeting process.

3.2.1 Preparing the Dataset

Preparing the dataset involved selecting suitable images for the project from the National Tsing Hua University image retargeting dataset (NRID). A total of 26 images with longer widths than heights were selected from a pool of 35. I selected 3 of these 26 images for optimization to reduce the Dijkstra run-time. To optimize these 3 images of the dataset, Adobe Photoshop was used to reduce the width and height of three of the images while maintaining their aspect ratios due to the long runtime of the Dijkstra retargeting algorithm. In addition to these images, a personal photo with the author's brother and two cats was also included, as well as a digital artwork created specifically for this project. These images formed the basis of the dataset used in the project, allowing for accurate and comprehensive testing of the Dijkstra retargeting algorithm. These 5 images (3 of them from NRID, 2 of which I created) were run with both Dijkstra and Greedy. Run-time comparison was made according to the average run-time of 5 pictures. Also, Dijkstra's accuracy was determined from the average accuracy of these 5 images. Apart from this, the remaining 23 images out of 26 selected from NRID were run only with Greedy. The accuracy of these 23 images and the other 5 images was averaged, thus determining Greedy's average accuracy.

3.2.2 3-Dimensional Colored Image to 2-Dimensional Grayscale Image

A color image can be read using several methods and one of these methods is reading as RGB with 3-dimensional list (Raguramanet et al., 2021). The length of the first dimension equals to image’s height, the second one equals to image’s width, and the third one equals three, which presents red, green, and blue values of the pixel. After we achieve RGB values of the pixel, there are three main methods to convert it to greyscale, which are lightness, average, and luminosity (Antoniadis et al., 2022). Green appears around ten times brighter to human eyes than blue does (Brandon, 2019). Researchers in psychology have discovered how differently we interpret the luminance of red, green, and blue through numerous iterations of carefully planned tests (Brandon, 2019). They gave us a unique set of weights to use with our channel averaging to calculate overall luminance (Brandon, 2019). According to this, we sum the RGB values with calculation using the following formula:

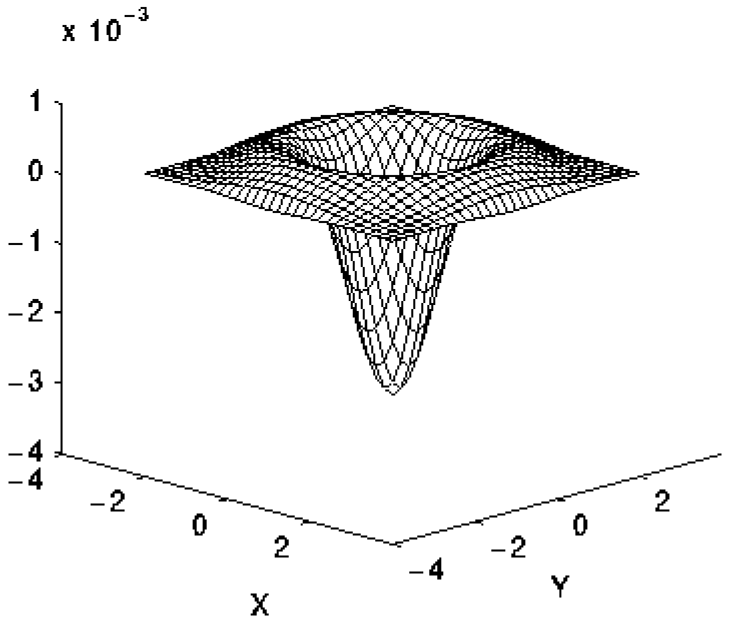
$$Y_{\text{linear}} = 0.2126R_{\text{linear}} + 0.7152G_{\text{linear}} + 0.0722B_{\text{linear}}$$

As a result, we get a 2-dimensional list which refers to a grayscale image.

3.2.3 Laplacian Transform

To apply the Laplacian filter, we get the grayscale image from the previous method. The Laplacian I(x,y) of an image with pixel intensity J(x,y) is ;

$$I(x,y) = \partial 2J / \partial 2x + \partial 2J / \partial 2y$$



The x and y axes are marked in standard deviation

Figure 3.2.3.1 The 2-D Laplacian function (researchgate.net)

For obtaining Laplacian image, I compared two commonly used discrete approximation Laplacian filters then I decided to use first of them since its more appropriate (Abramowitz et al., 2013). Related Laplace filters are shown in Table 3.2.3.1.

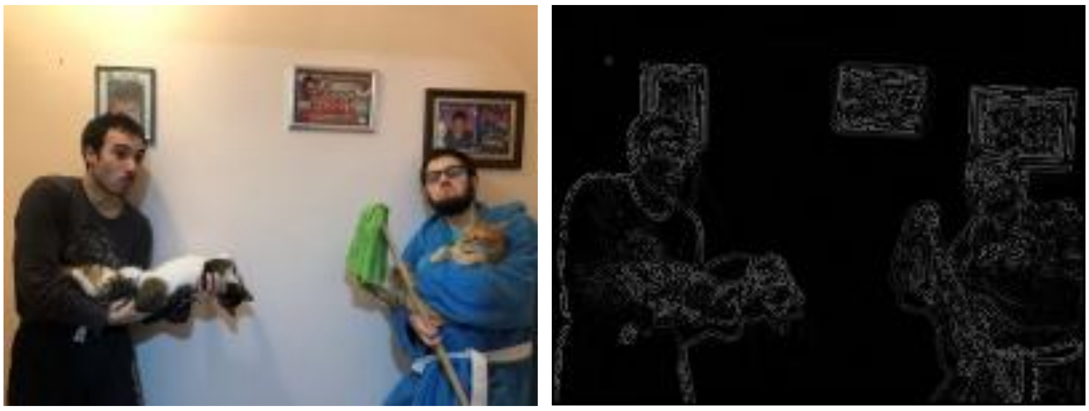
Table 3.2.3.1 Compared Laplacian filters

1	1	1
1	-8	1
1	1	1

This was used for the project

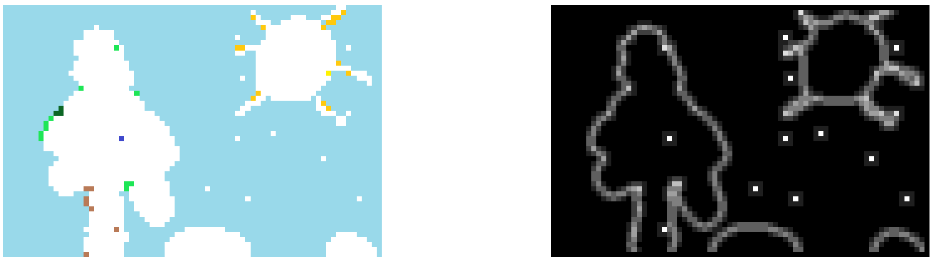
0	1	0
1	-4	1
0	1	0

Most important parts of the image can be detectable easily by computers when comparing the original image (Haralick et al., 1993).



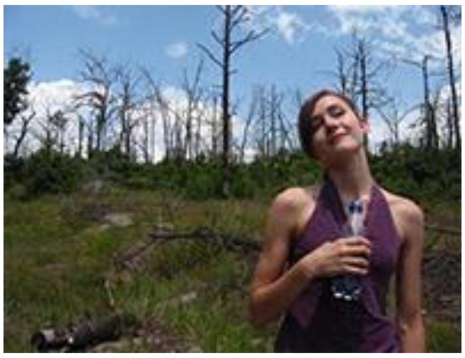
The original photo　　Laplacian filter applied photo

Figure 3.2.3.2 The original and Laplacian filtered photos



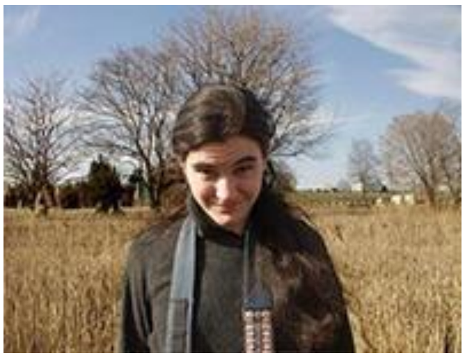
The original image　　Laplacian filter applied image

Figure 3.2.3.3 The original and Laplacian filtered images


The original image

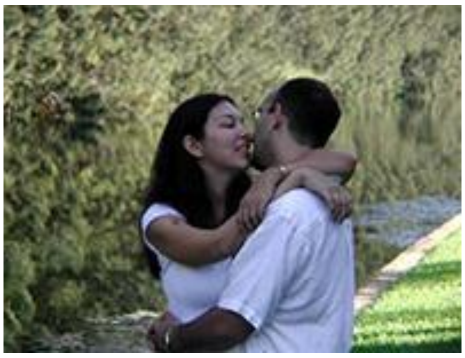
Laplacian filter applied image

Figure 3.2.3.4 The original and Laplacian filtered images in NRID (*ours_11_aaa*)


The original image

Laplacian filter applied image

Figure 3.2.3.5 The original and Laplacian filtered images in NRID (*ours_14_aaa*)


The original image

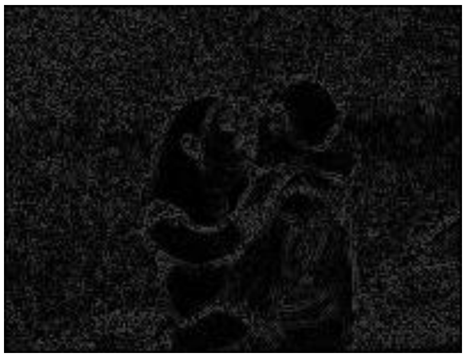

Laplacian filter applied image

Figure 3.2.3.6 The original and Laplacian filtered images in NRID (*ours_16_aaa*)

3.2.4 Finding The Lowest Energy Points – The Shortest Path

After we apply the Laplacian filter to the grayscale image, we get all the energy points of the pixels. Based on these energy points, the sum of the lowest energy point route is calculated. There are several algorithms to calculate the shortest path (Baum, 2020). I compare 2 different algorithms which are first-level greedy approach search and Dijkstra.

3.2.4.1 First-level Greedy Approach

The algorithm starts from all of the top pixels separately, then it’s going down by comparing the energy points of their first neighbors which are left, center, and right choosing the lowest energy pixels. It can be exactly under or right cross under or left cross under till I reach from the first row to the last row of an image.

After that calculation, the algorithm gets multiple routes as many as the number of the image width. Total energy points were calculated for each route while routes were created. A new row is added at the bottom of the image and the total points that are calculated for each start point pixel are written on that new row.

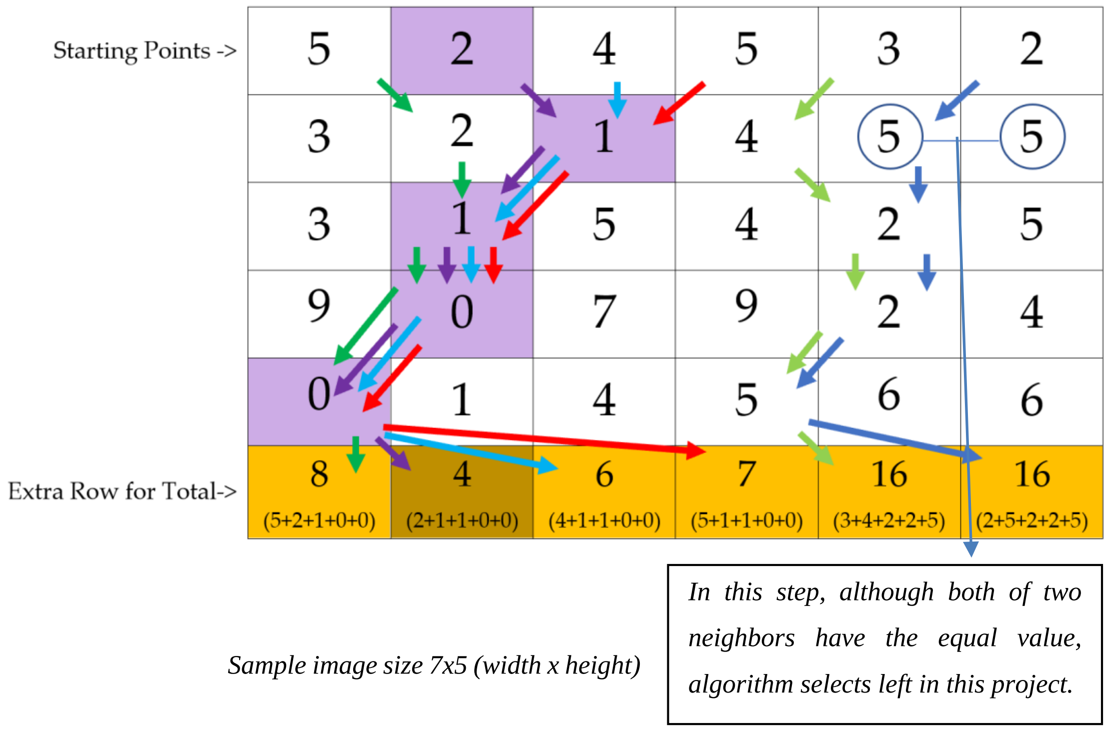


Figure 3.2.4.1.1 Greedy approach

The extra row contains the total scores of all routes and according to this information; the starting point with the lowest total energy point will be selected. The selected route will be saved in a variable so it can be used to mark the selected route with a red marker.

After the initial marking is complete, the width of the image can be reduced by 1 pixel. If the image wants to be reduced by more than 1 pixel, the algorithm will re-search for energy points for specific routes since I made an optimization in the greedy approach instead of using classical greedy approach. Based on this optimization, when a path marked/removed, the algorithm detects which routes had been affected and calculate only these paths' energy points, this optimized greedy approach is working ~91,93% faster than classical greedy approach.

Table 3.2.4.1.1 Comparison of Optimized and Classical Greedy Approach

Greedy Approach without Optimization			Greedy Approach with Optimization	
Image Name	Run Time		Image Name	Run Time
ours_11_aaa	0:02:06		ours_11_aaa	0:00:11
ours_14_aaa	0:02:06		ours_14_aaa	0:00:09
ours_16_aaa	0:02:05		ours_16_aaa	0:00:10
TOTAL	0:06:14		TOTAL	0:00:30
AVERAGE	0:02:04		AVERAGE	0:00:10
			PERCENTAGE	-~91.93%

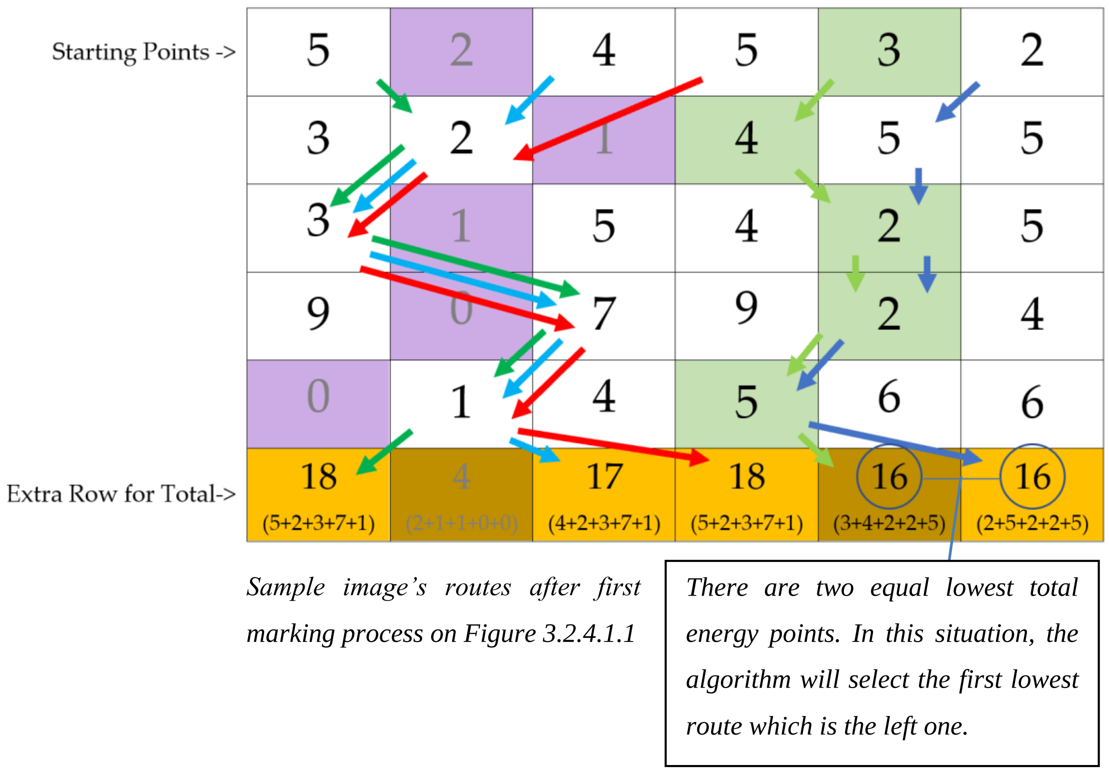
Also, algorithm will skip the marked pixels while searching for the lowest energy point. This means:

- First, the algorithm detects which routes should be recalculated (*DetermineRecalculationPixels*). To do that, it checks the previous selected/marked pixels that was stored in a variable (*usedPixels*), and starts searching in the calculated paths (*routeAndSumOfEnergyPoints*) on the previous 1-pixel image retargeting process. If it finds that the pixel was already used in the previous processes, it adds it to a list (*recalculationStartPixels*) for the recalculation process.
- After recalculation paths were determined, the algorithm starts to find the lowest energy point path without using selected/marked pixels (*usedPixels*): If the algorithm determines that the left neighbor pixel is marked red, it goes left in the same row and checks if it is marked or not. This search process continues until the algorithm finds an unmarked pixel. If there is no unmarked pixel on

left, the algorithm acts like there is no pixel on the left side and tries to compare center and right neighbors.

- If the algorithm determines that the center neighbor pixel is marked red, it acts like there is no center neighbor and tries to compare left and right neighbors.
- If the algorithm determines that the right neighbor pixel is marked red, it goes right in the same row and checks if it is marked or not. This search process continues until the algorithm finds an unmarked pixel. If there is no unmarked pixel on right, the algorithm acts like there is no pixel on the right side and tries to compare left and center neighbors.

Total energy point calculations are made based on this method and override the old total energy points on the last extra row of image.



Sample image's routes after first marking process on Figure 3.2.4.1.1

There are two equal lowest total energy points. In this situation, the algorithm will select the first lowest route which is the left one.

Figure 3.2.4.1.2 Greedy approach (Next Step)

These processes are repeated until the algorithm reaches the desired retarget number. When the marking process is completed, a method removes the marked pixels from the original image. I will mention the marking and removing process with details in the Chapter 3.2.5.

3.2.4.2 Dijkstra Algorithm

In 1959, Edsger Dijkstra, a Dutch computer scientist, presented an algorithm that may be used with a weighted graph. The graph must embrace a non-negative value on each of its edges in order to be considered directed or undirected. This algorithm was given his name, "Dijkstra's Algorithm" (Tyagi, 2020).

The energy points of the image we already calculated will be used as weight or cost in Dijkstra as we used in the greedy approach. Based on these weights, Dijkstra will determine the shortest path on the directed graph which represent the Laplacian filtered 2-dimensional list of the image. As I mention on Chapter 2, first Dijkstra calculates the lowest energy path. After Dijkstra calculates the lowest energy path, the path is saved in a variable. Unlike our other algorithm, this 1 pixel-wide path is deleted directly from the Laplacian filtered image and the original image, without waiting for the desired other pixel width to be reduced. As in the other algorithm, the relevant path is recorded to be marked with a red marker, but no marking is made at that time: When the desired shrink pixel width is reached, marking with red will be performed. In addition to this situation, since the 2-dimensional list given to Dijkstra's algorithm for the second shortest path selection process will now have a width of 1 pixel less, the new shortest path of the Dijkstra won't give us the actual right way for us to mark on the original image. Because now the pixel indexes have been changed. At this point, we will provide editing via a helper pre-process method (adjustDijPaths) before saving the shortest path in the second loop to our list. I will explain this method in the next chapter. For each pixel wide to be deleted, the algorithm will re-search for energy points for all routes but will skip the marked pixels while searching for the lowest energy point. Because these pixels are already deleted from the Laplacian image, which includes the energy points (weights) and Dijkstra calculates the shortest path using the Laplacian image. A method for marking the used pixels from the original and the Laplacian image follows the deletion process. In the next chapter, I will go into more detail on marking.

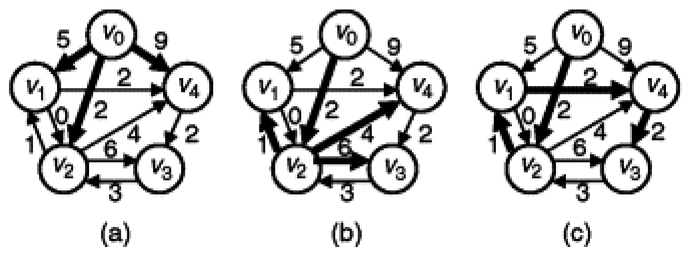


Figure 3.2.4.2.1 The logic of Dijkstra Algorithm (sciencedirect.com)

Table 3.2.4.2.1 The logic of Dijkstra Algorithm (sciencedirect.com)

vertex	Predecessors					Shortest-Path Estimates				
	v_0	v_1	v_2	v_3	v_4	v_0	v_1	v_2	v_3	v_4
non	NIL	NIL	NIL	NIL	NIL	0	∞	∞	∞	∞
v_0	NIL	v_0	v_0	NIL	v_0	0	5	2	∞	9
v_2	NIL	v_2	v_0	v_2	v_2	0	3	2	8	6
v_1	NIL	v_2	v_0	v_2	v_1	0	3	2	8	5
v_4	NIL	v_2	v_0	v_4	v_1	0	3	2	7	5
v_3	NIL	v_2	v_0	v_4	v_1	0	3	2	7	5

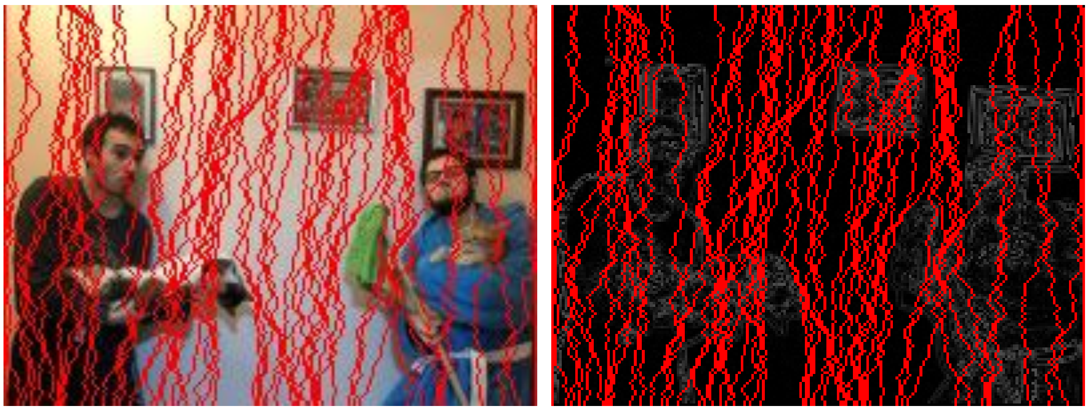
3.2.5 Marking Red – Removing Pixels

Both two algorithms work differently; although the greedy approach deletes the selected pixels from the original and the Laplacian image when all the desired width is finished, Dijkstra’s part deletes the selected pixels right after the 1-pixel wide path detected. However, marking with red process works the same way with a minor difference for both two algorithms.

3.2.5.1 Process Sequence for First-level Greedy Approach

When the lowest energy (weight) path detected by greedy approach, this path is added to a list variable (*usedPixels*). Before the next shortest path is detected, the algorithm passes the currently selected path to a method (*paintRedForGreedySearch*) for marking with red. This method loops through the image’s height and change in the third depth of the image with using each selected path’s value. The length of the third depth of the image is three and these represent red, green, and blue values. Since we would like to mark it red along the selected path, we change the [0] to 255, [1] to 0, and [2] to 0. As a result, the selected path’s pixels’ colors on the image are changed to red color.

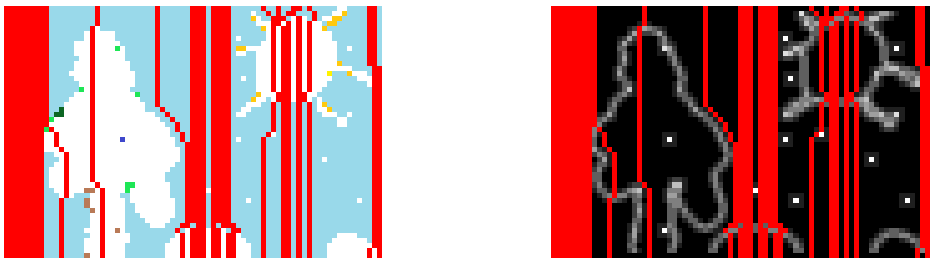
After all of the desired pixel width loops are finished and all the selected paths are marked as red, the list which includes all these selected paths (*usedPixels*) is sent to another method (*removeMarkedPixels*) to retarget (resize) the image. The most important part of this method is; to present and match the same list indexes of the image and the selected path list, the method sorts each row of the selected pixels list as descending. Then it starts removing the first index of the sorted list; which is caused the pixels to be started to delete from the end of each row of the image. If we don't do that sorting before the removing process, when a pixel from near of the start of the row is deleted, the whole selected pixel list's indexes show the wrong (shifted) indexes on the resized image.



Red marks on the original photo

Red marks on the Laplacian filter applied photo

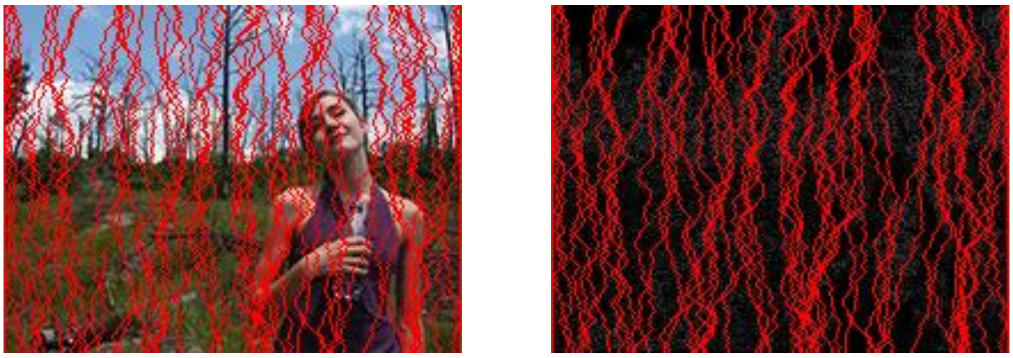
Figure 3.2.5.1.1 Red marks on the original and Laplacian filtered photos with using greedy approach



Red marks on the original image

Red marks on the Laplacian filter applied image

Figure 3.2.5.1.2 Red marks on the original and Laplacian filtered images with using greedy approach



Red marks on the original image | Red marks on the Laplacian filter applied image

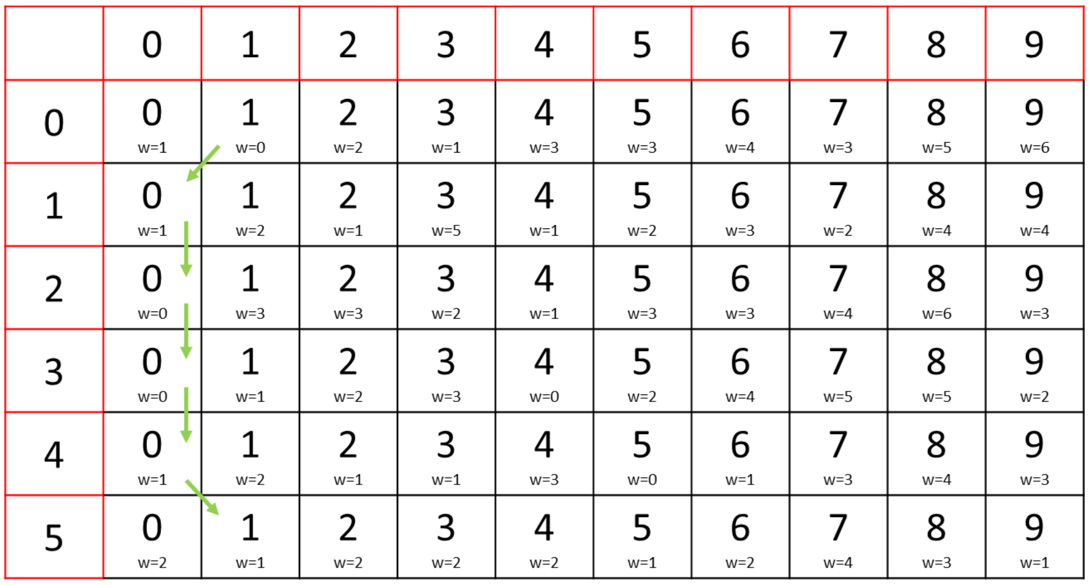
Figure 3.2.5.1.3 Red marks on the original and Laplacian filtered images in NRID (*ours_11_aaa*) with using greedy approach

3.2.5.2 Process Sequence for Dijkstra Algorithm

The path that has the lowest energy is added to a list variable (*dijPaths*) when it is found by the Dijkstra algorithm. Before the algorithm passes the found shortest path to the method that deletes these pixels, firstly the found shortest path is sent to another pre-process algorithm (*adjustDijPaths*) to adjust the indexes of the path. Contrary to the greedy approach, this kind of operation is needed because, unlike the greedy approach, before each shortest path detection loop in the Dijkstra process, the path detected in the previous loop is deleted and the new shortest path proceeds over the image reduced to 1 pixel width. The way the harmonization (adjustment) process works is as follows:

- The method behaves as if all indexes are affected as much as possible. That means it adds the number of the previously selected shortest paths to all currently selected path indexes.
- Then it tries to find how many pixels of the previous shortest paths don't affect the current shortest path and subtract 1 for each one.
- To find which pixels don't affect, a loop starts from the length of the previously selected paths and goes through by subtracting 1.
- In this loop, it checks if the selected path index is bigger or equal to the currently selected path index. If it's true, subtract 1 from the currently selected path index.

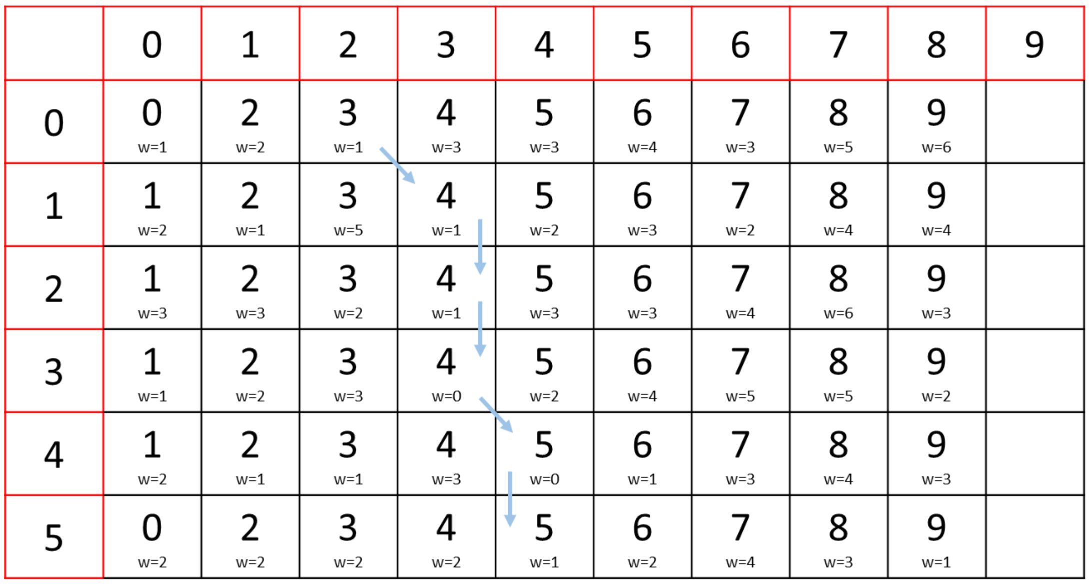
- After the loop is completed, the adjusted currently selected path is added to the list variable which stores all the shortest paths.



	0	1	2	3	4	5	6	7	8	9
0	0 w=1	1 w=0	2 w=2	3 w=1	4 w=3	5 w=3	6 w=4	7 w=3	8 w=5	9 w=6
1	0 w=1	1 w=2	2 w=1	3 w=5	4 w=1	5 w=2	6 w=3	7 w=2	8 w=4	9 w=4
2	0 w=0	1 w=3	2 w=3	3 w=2	4 w=1	5 w=3	6 w=3	7 w=4	8 w=6	9 w=3
3	0 w=0	1 w=1	2 w=2	3 w=3	4 w=0	5 w=2	6 w=4	7 w=5	8 w=5	9 w=2
4	0 w=1	1 w=2	2 w=1	3 w=1	4 w=3	5 w=0	6 w=1	7 w=3	8 w=4	9 w=3
5	0 w=2	1 w=1	2 w=2	3 w=2	4 w=2	5 w=1	6 w=2	7 w=4	8 w=3	9 w=1

Dijkstra selected [1, 0, 0, 0, 0, 1] as the lowest weighted path on the first loop. It will mark [1, 0, 0, 0, 0, 1].

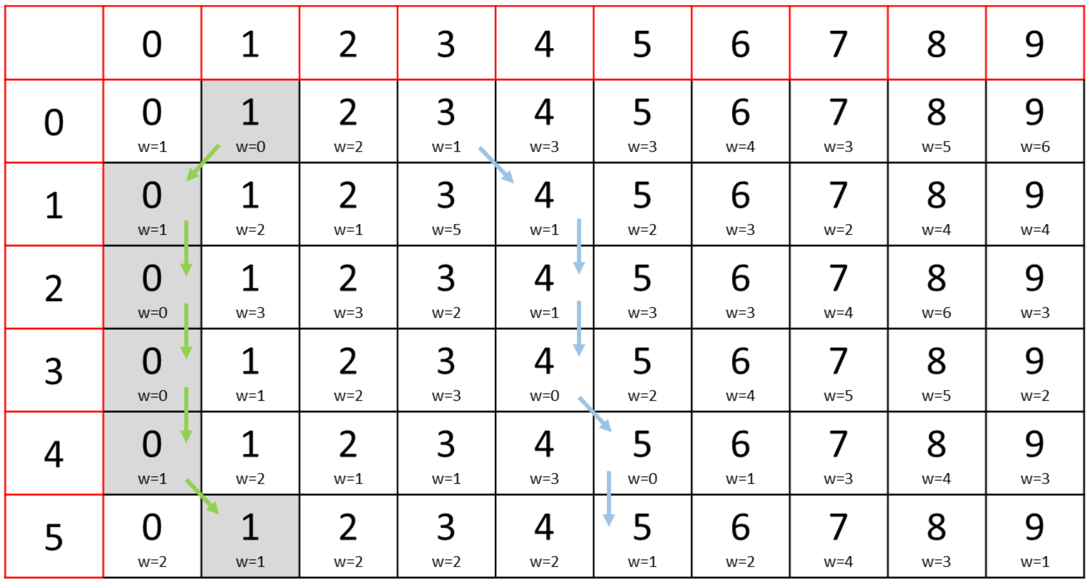
Figure 3.2.5.2.1 Why algorithm needs adjusting process before marking pixels –Step 1



	0	1	2	3	4	5	6	7	8	9
0	0 w=1	2 w=2	3 w=1	4 w=3	5 w=3	6 w=4	7 w=3	8 w=5	9 w=6	
1	1 w=2	2 w=1	3 w=5	4 w=1	5 w=2	6 w=3	7 w=2	8 w=4	9 w=4	
2	1 w=3	2 w=3	3 w=2	4 w=1	5 w=3	6 w=3	7 w=4	8 w=6	9 w=3	
3	1 w=1	2 w=2	3 w=3	4 w=0	5 w=2	6 w=4	7 w=5	8 w=5	9 w=2	
4	1 w=2	2 w=1	3 w=1	4 w=3	5 w=0	6 w=1	7 w=3	8 w=4	9 w=3	
5	0 w=2	2 w=2	3 w=2	4 w=2	5 w=1	6 w=2	7 w=4	8 w=3	9 w=1	

Dijkstra selected [2, 3, 3, 3, 4, 4] as the lowest weighted path from 1-pixel wide deleted image. If it directly marked red without adjusting, it will mark [2, 3, 3, 3, 4, 4] although it should mark [3, 4, 4, 4, 5, 5] on the original image.

Figure 3.2.5.2.2 Why algorithm needs adjusting process before marking pixels –Step 2

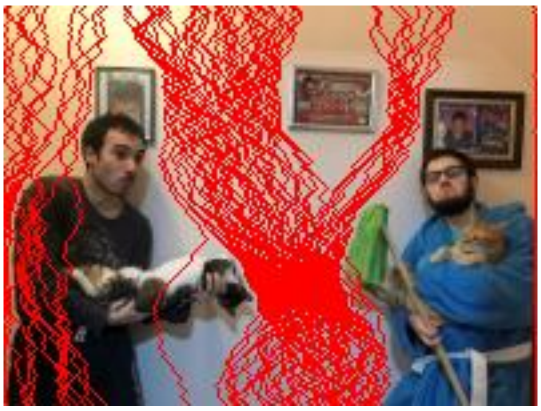


	0	1	2	3	4	5	6	7	8	9
0	0 w=1	1 w=0	2 w=2	3 w=1	4 w=3	5 w=3	6 w=4	7 w=3	8 w=5	9 w=6
1	0 w=1	1 w=2	2 w=1	3 w=5	4 w=1	5 w=2	6 w=3	7 w=2	8 w=4	9 w=4
2	0 w=0	1 w=3	2 w=3	3 w=2	4 w=1	5 w=3	6 w=3	7 w=4	8 w=6	9 w=3
3	0 w=0	1 w=1	2 w=2	3 w=3	4 w=0	5 w=2	6 w=4	7 w=5	8 w=5	9 w=2
4	0 w=1	1 w=2	2 w=1	3 w=1	4 w=3	5 w=0	6 w=1	7 w=3	8 w=4	9 w=3
5	0 w=2	1 w=1	2 w=2	3 w=2	4 w=2	5 w=1	6 w=2	7 w=4	8 w=3	9 w=1

If we see the marked original image, we can easily see that list must be shifted before marking again.

Figure 3.2.5.2.3 Why algorithm needs adjusting process before marking pixels –Step 3

Then, the algorithm sends the currently chosen both adjusted and raw (not adjusted) paths to a method (*completeRetargetProcessForDijkstra*) for removing the chosen path from both the original image and the Laplacian image before the next shortest path is found. This method also paints red using the adjusted selected path. For painting red, the method uses the same way with greedy approach.


Red marks on the original photo

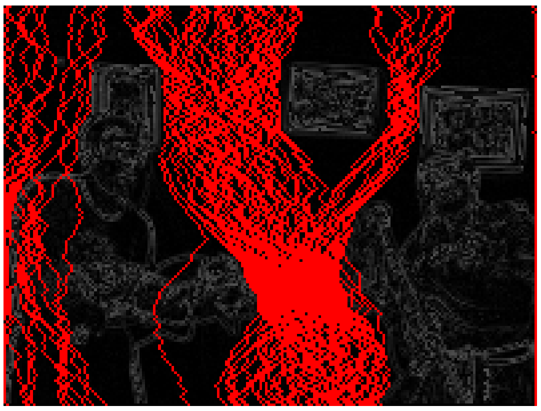
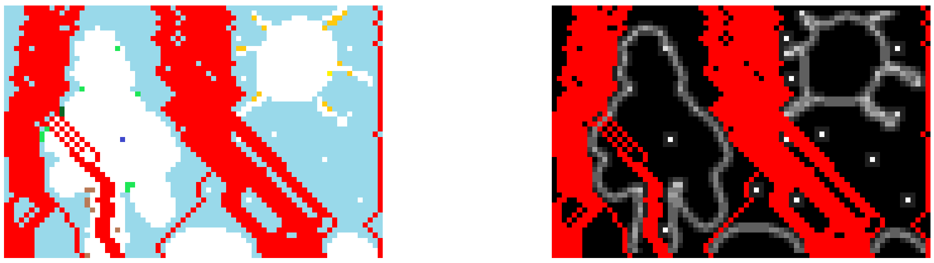
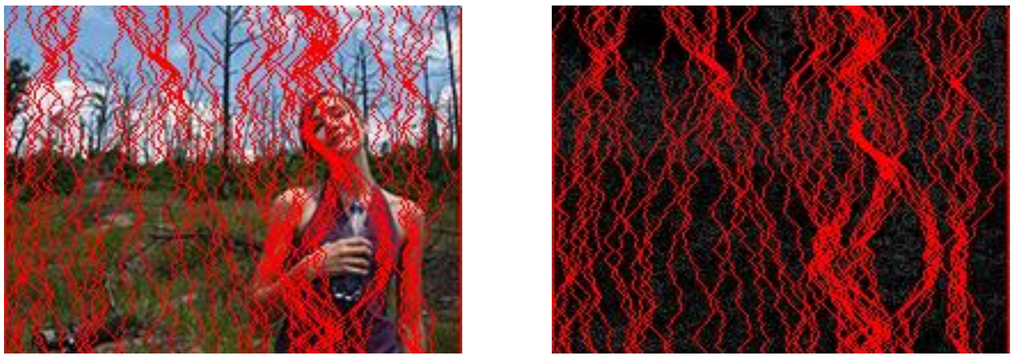

Red marks on the Laplacian filter applied photo

Figure 3.2.5.2.4 Red marks on the original and Laplacian filtered photos with using Dijkstra algorithm



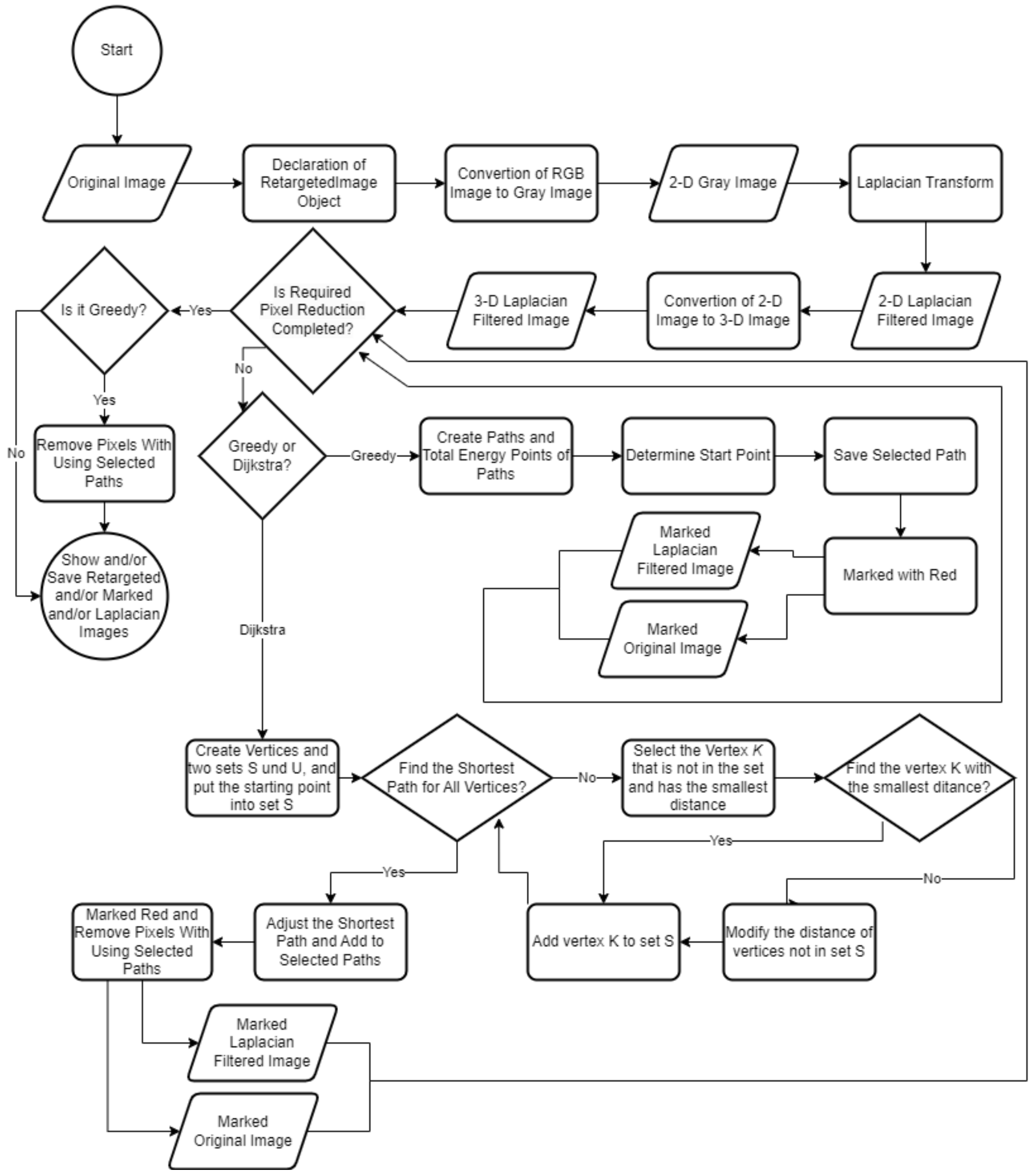
Red marks on the original image Red marks on the Laplacian filter applied image

Figure 3.2.5.2.5 Red marks on the original and Laplacian filtered images with using Dijkstra algorithm



Red marks on the original image Red marks on the Laplacian filter applied image

Figure 3.2.5.2.6 Red marks on the original and Laplacian filtered images in NRID (*ours_11_aaa*) with using Dijkstra algorithm



Start
Original Image
Declaration of RetargetedImage Object
Convertion of RGB Image to Gray Image
2-D Gray Image
Laplacian Transform
2-D Laplacian Filtered Image
Convertion of 2-D Image to 3-D Image
3-D Laplacian Filtered Image
Is Required Pixel Reduction Completed?
Yes
Is it Greedy?
Yes
No
Remove Pixels With Using Selected Paths
Show and/or Save Retargeted and/or Marked and/or Laplacian Images
No
Greedy or Dijkstra?
Greedy
Create Paths and Total Energy Points of Paths
Determine Start Point
Save Selected Path
Marked with Red
Marked Laplacian Filtered Image
Marked Original Image
Dijkstra
Create Vertices and two sets S und U, and put the starting point into set S
Find the Shortest Path for All Vertices?
No
Select the Vertex K that is not in the set and has the smallest distance
Find the vertex K with the smallest ditance?
Yes
No
Add vertex K to set S
Modify the distance of vertices not in set S
Yes
Adjust the Shortest Path and Add to Selected Paths
Marked Red and Remove Pixels With Using Selected Paths
Marked Laplacian Filtered Image
Marked Original Image

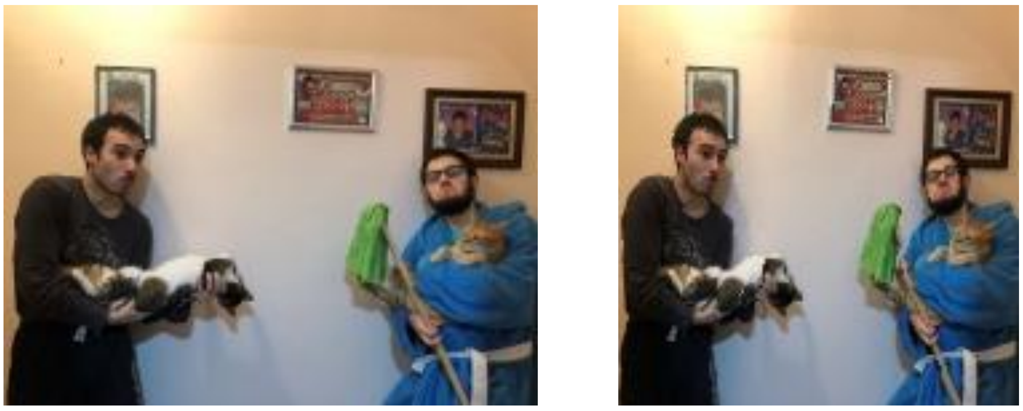
Figure 3.2.5.2.7 Flowchart of the Algorithm

CHAPTER 4

4. RESULTS & COMPARISON

4.1 Visual Results

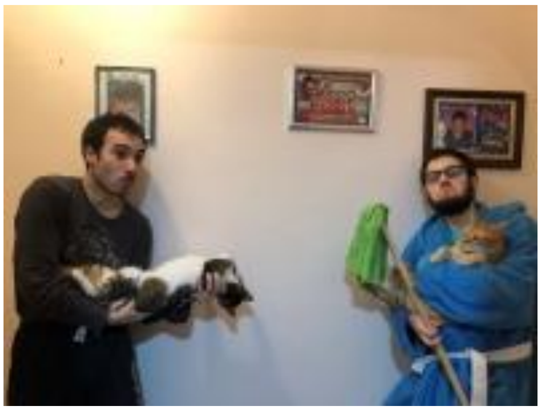
As a result, although both algorithms shrink the images to a square shape, they give very different results visually due to the methods they use and the shortest paths they choose. Especially when we look at the red-marked pictures given by the greedy approach, we can observe that as a human being the red line pass over many objects that we can define as "important". As a result, pixels that we don't want to be removed from the picture are also removed, causing the components of important objects to deteriorate, although not as much as cropping or stretching.



The original photo

Retargeted photo with greedy approach

Figure 4.1.1 The original and retargeted photo with greedy approach


The original photo

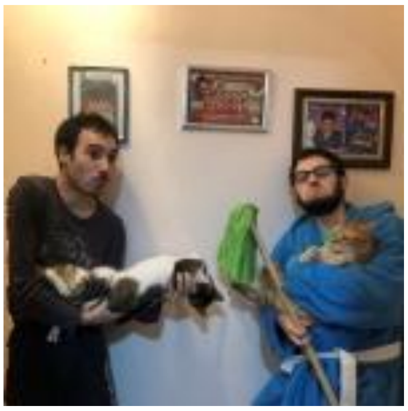
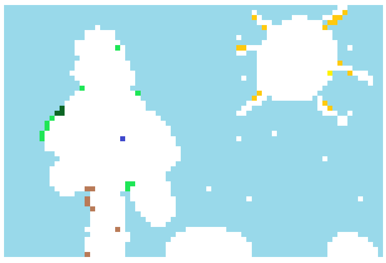

Retargeted photo with Dijkstra algorithm

Figure 4.1.2 The original and retargeted photo with Dijkstra algorithm


The original image

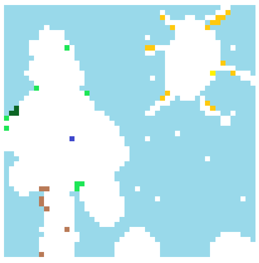
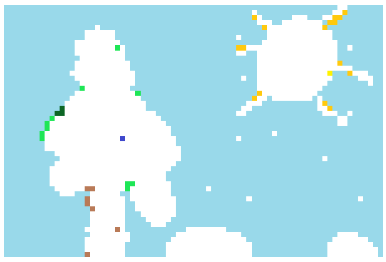

Retargeted image with greedy approach

Figure 4.1.3 The original and retargeted image with greedy approach


The original image

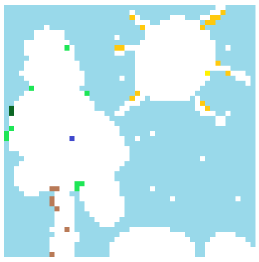
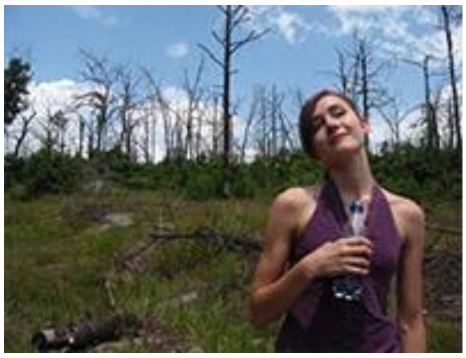

Retargeted image with Dijkstra algorithm

Figure 4.1.4 The original and retargeted image with Dijkstra algorithm


The original image

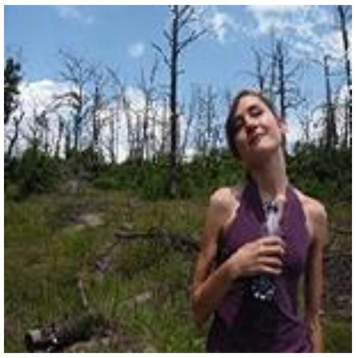
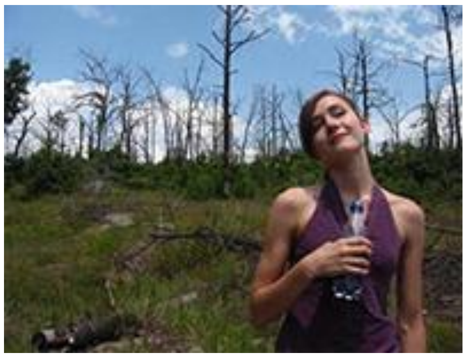

Retargeted image with Greedy algorithm

Figure 4.1.5 The original and retargeted image with Greedy algorithm in NRID (*ours_11_aaa*)


The original image

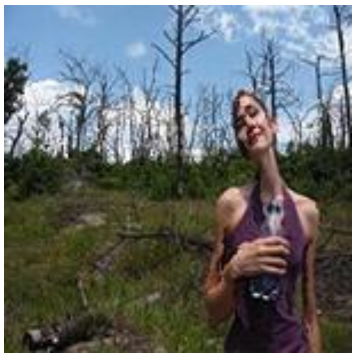
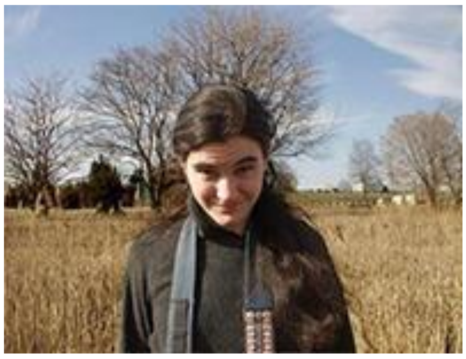

Retargeted image with Dijkstra algorithm

Figure 4.1.6 The original and retargeted image with Dijkstra algorithm in NRID (*ours_11_aaa*)


The original image

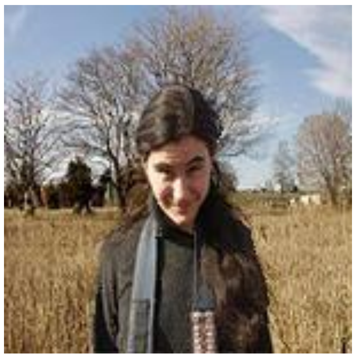
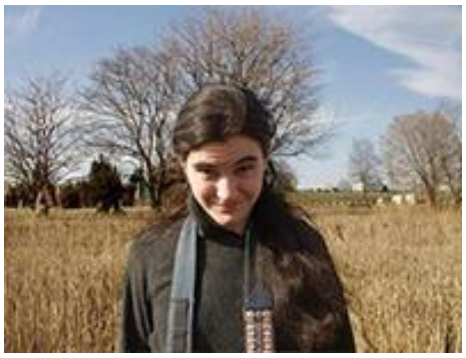
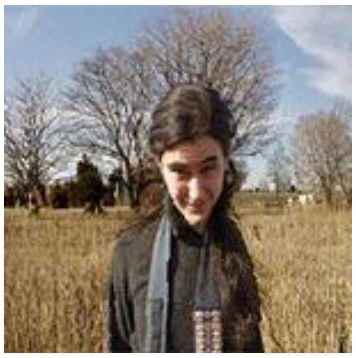
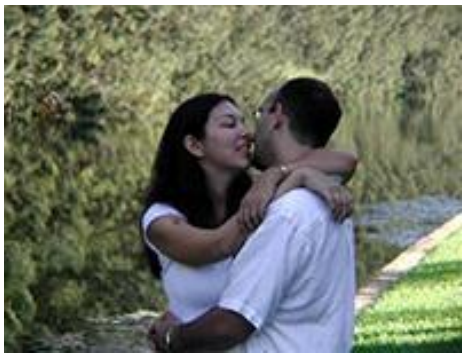
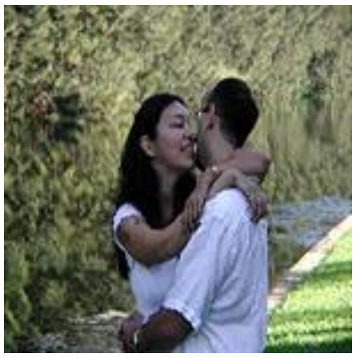

Retargeted image with Greedy algorithm

Figure 4.1.7 The original and retargeted image with Greedy algorithm in NRID (*ours_14_aaa*)

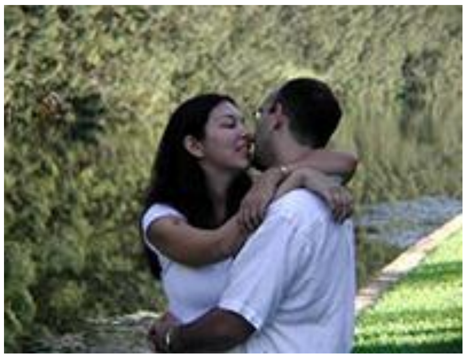
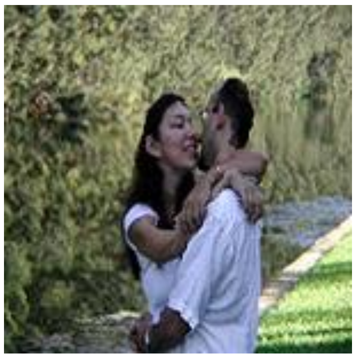
The original image Retargeted image with Dijkstra algorithm

Figure 4.1.8 The original and retargeted image with Dijkstra algorithm in NRID (*ours_14_aaa*)

The original image Retargeted image with Greedy algorithm

Figure 4.1.9 The original and retargeted image with Greedy algorithm in NRID (*ours_16_aaa*)

The original image Retargeted image with Dijkstra algorithm

Figure 4.1.10 The original and retargeted image with Dijkstra algorithm in NRID (*ours_16_aaa*)

4.2 Intersection over Union (IoU)

Intersection over Union (IoU) is a widely used evaluation metric in computer vision, particularly in the field of object detection and segmentation (Redmon et al., 2016). IoU measures the similarity between two bounding boxes or regions of interest by calculating the overlap between them. The IoU score ranges from 0 to 1, where 1 indicates a perfect overlap between the two regions, and 0 indicates no overlap at all (Everingham et al., 2010).

The IoU score is calculated as follows:

IoU = (Area of Intersection) / (Area of Union)

Where the Area of Intersection is the overlap between the two regions, and the Area of Union is the total area covered by both regions.

IoU is commonly used as an evaluation metric for object detection models, where it is used to measure the accuracy of the model's predictions (Ronneberger et al., 2015). A high IoU score indicates that the predicted bounding box closely matches the ground truth bounding box.

Figure 4.2.1 Supervised photo

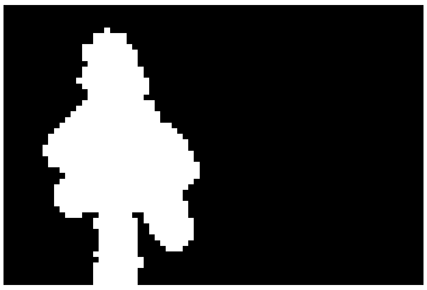


Figure 4.2.2 Supervised image

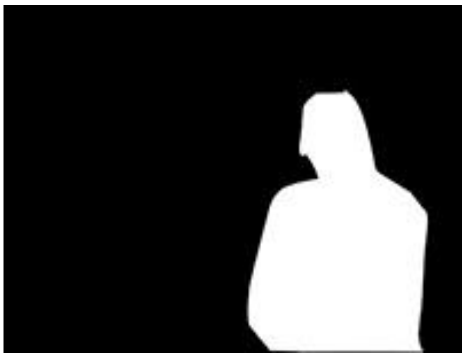


Figure 4.2.3 Supervised image in NRID (*ours_11_aaa*)

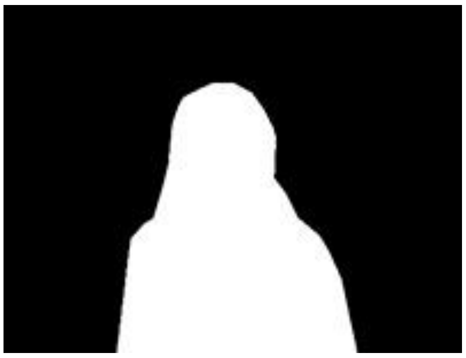


Figure 4.2.4 Supervised image in NRID (*ours_14_aaa*)

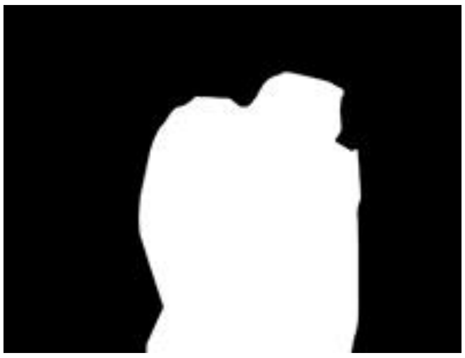
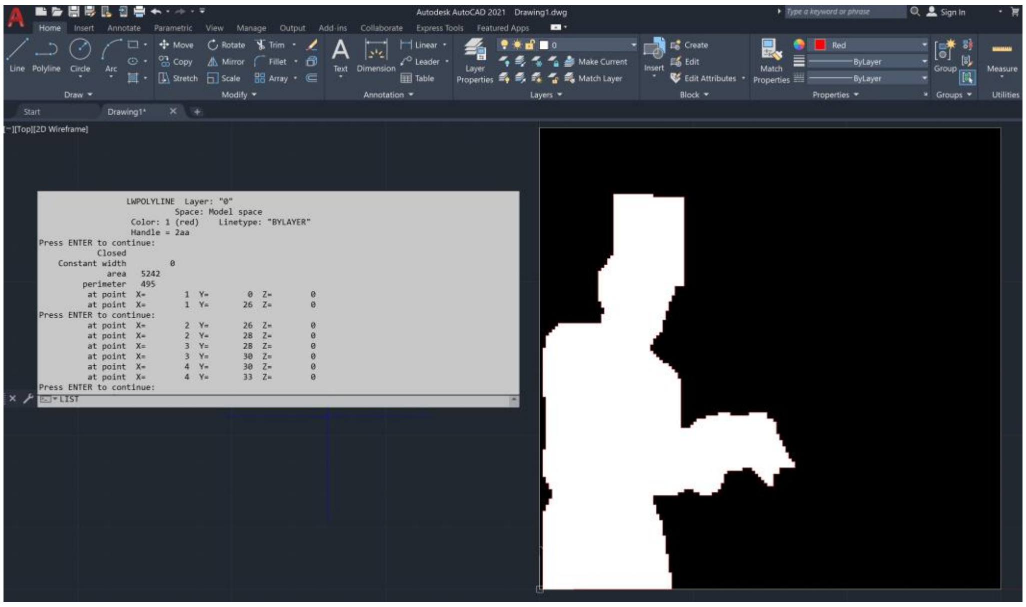


Figure 4.2.5 Supervised image in NRID (*ours_16_aaa*)

Both of supervised image and the retargeted image (by either the greedy approach or Dijkstra) are given to another Python file (*comparer.py*) which can be run separately from the main image retargeting program. In this study, supervised images were used to create polygons in AutoCAD 2021. The coordinates of these polygons were then exported and utilized in a Python file. The Shapely library was employed to recreate the polygons in Python, and to calculate the intersection, union, and IoU of the polygons. These calculations were essential for evaluating the accuracy of the important objects in the images. In addition, the Matplotlib library was used to plot the polygons, providing a visual representation of the results. Overall, the combination of AutoCAD, Python, Shapely, and Matplotlib enabled thorough analysis of the polygons and the accuracy of their representation in the images.



```
                 LWPOLYLINE  Layer: "0"
                          Space: Model space
                  Color: 1 (red)    Linetype: "BYLAYER"
                  Handle = 2aa
Press ENTER to continue:
            Closed
    Constant width        0
               area   5242
          perimeter   495
           at point  X=        1  Y=        0  Z=        0
           at point  X=        1  Y=       26  Z=        0
Press ENTER to continue:
           at point  X=        2  Y=       26  Z=        0
           at point  X=        2  Y=       28  Z=        0
           at point  X=        3  Y=       28  Z=        0
           at point  X=        3  Y=       30  Z=        0
           at point  X=        4  Y=       30  Z=        0
           at point  X=        4  Y=       33  Z=        0
Press ENTER to continue:
LIST
```

Figure 4.2.6 Creating Polylines in AutoCAD for Getting Coordinates

Table 4.2.1 IoU of first-level greedy approach photo

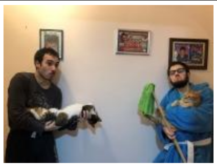
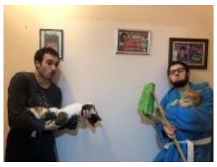
Retarget method:	First-level greedy approach	
Image dimensions:	150 x 200	
Total pixels:	30,000	
	Intersection	4125
	Union	6331
	IoU	~0.6515

Table 4.2.2 IoU of Dijkstra photo

Retarget method:	Dijkstra	
Image dimensions:	150 x 200	
Total pixels:	30,000	
	Intersection	4668
	Union	6454
	IoU	~0.7232

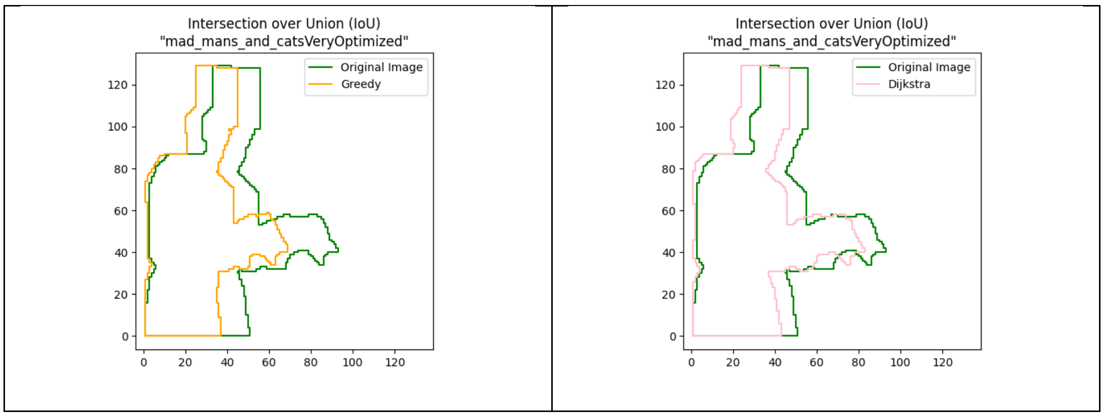


Figure 4.2.7 Visual representation of IoU of photo

Table 4.2.3 IoU of first-level greedy approach image

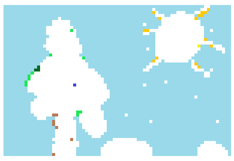
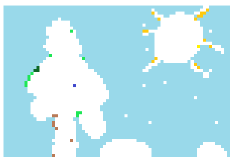
Retarget method:	First-level greedy approach	
Image dimensions:	50 x 75	
Total pixels:	3,750	
	Intersection	653
	Union	732
	IoU	~0.8920

Table 4.2.4 IoU of Dijkstra image

Retarget method:	Dijkstra	
Image dimensions:	50 x 75	
Total pixels:	3,750	
	Intersection	569
	Union	792
	IoU	~0.7184

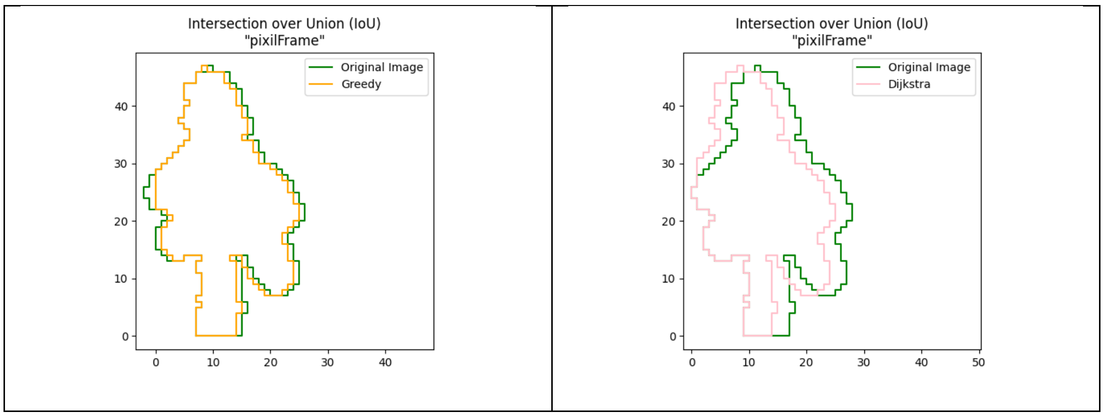


Figure 4.2.8 Visual representation of IoU of image

Table 4.2.5 IoU of first-level greedy approach image in NRID (*ours_11_aaa*)

Retarget method:	First-level greedy approach	
Image dimensions:	152 x 200	
Total pixels:	30,400	
	Intersection	4945
	Union	6488
	IoU	~0.7621

Table 4.2.6 IoU of Dijkstra image in NRID (*ours_11_aaa*)

Retarget method:	Dijkstra	
Image dimensions:	152 x 200	
Total pixels:	30,400	
	Intersection	3387
	Union	7070
	IoU	~0.4791

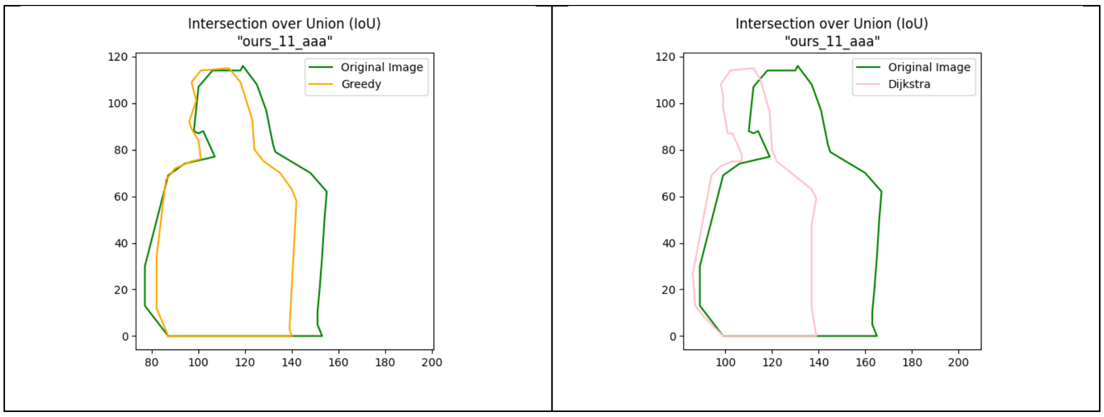


Figure 4.2.9 Visual representation of IoU of image in NRID (*ours_11_aaa*)

Table 4.2.7 IoU of first-level greedy approach image in NRID (*ours_14_aaa*)

Retarget method:	First-level greedy approach	
Image dimensions:	152 x 200	
Total pixels:	30,400	
	Intersection	6258
	Union	8337
	IoU	~0.7507

Table 4.2.8 IoU of Dijkstra image in NRID (*ours_14_aaa*)

Retarget method:	Dijkstra	
Image dimensions:	152 x 200	
Total pixels:	30,400	
	Intersection	5302
	Union	8600
	IoU	~0.6164

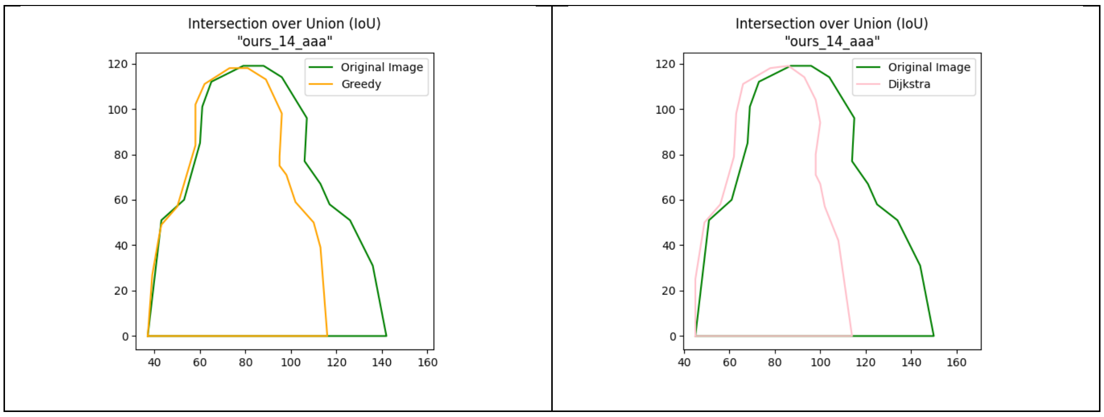


Figure 4.2.10 Visual representation of IoU of image in NRID (*ours_14_aaa*)

Table 4.2.9 IoU of first-level greedy approach image in NRID (*ours_16_aaa*)

Retarget method:	First-level greedy approach	
Image dimensions:	152 x 200	
Total pixels:	30,400	
	Intersection	7285
	Union	10461
	IoU	~0.6964

Table 4.2.10 IoU of Dijkstra image in NRID (*ours_16_aaa*)

Retarget method:	Dijkstra	
Image dimensions:	152 x 200	
Total pixels:	30,400	
	Intersection	5949
	Union	10809
	IoU	~0.5504

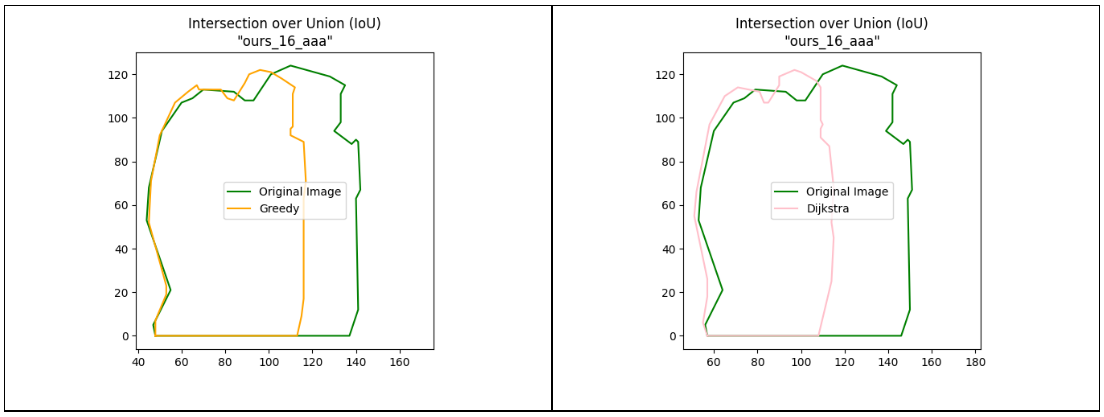


Figure 4.2.11 Visual representation of IoU of image in NRID (*ours_16_aaa*)

Table 4.2.11 IoU of greedy approach

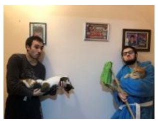
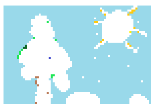
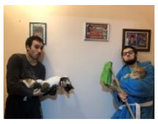
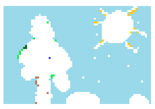
					
Retarget method:	First-level greedy approach				
Image dimensions:	150 x 200	50 x 75	152 x 200	152 x 200	152 x 200
Accuracy	0.6515	0.8920	0.7621	0.7507	0.6964

Table 4.2.12 IoU of Dijkstra

					
Retarget method:	Dijkstra				
Image dimensions:	150 x 200	50 x 75	152 x 200	152 x 200	152 x 200
Accuracy	0.7232	0.7184	0.4791	0.6164	0.5504

4.3 Mean Results and Comparison

When it comes to accuracy and execution time, the greedy algorithm outperforms Dijkstra's algorithm. Moreover, it can be said by observing in a supervised manner that the greedy technique is more accurate on a variety of colored backgrounds compared to images with a single or fairly similar colored background. Based on this inference, we can put forward the thesis that the reason why Dijkstra's algorithm gives worse accuracy is related to the detection of important objects in the picture and the evaluation of the values of energy points, rather than the algorithm itself.

Table 4.3.1 Mean accuracy (IoU) of the all the image

	Average results of all the images*	
Retarget method:	Greedy approach	Dijkstra algorithm
Image dimensions:	150 x 200 and 50 x 75	
Accuracy (IoU)	0.6598	0.6175

* The IoU average of 5 images for Dijkstra, and the IoU average of 23 additional images of these 5 images (28 images in total) for Greedy

The Dijkstra algorithm gives worse mean accuracy than the greedy approach, as well as works much slower than the greedy approach since the time complexity of Dijkstra is O(E*logV) and first-level greedy approach is O(N^m). To understand and compare easier, I run the code on exact same powerful computer. The machine has Intel i9-10900K CPU, 64GB 4880MHz RAM, RTX 3090 24GB graphic card.

Table 4.3.2 Run time and accuracy comparison of greedy approach and Dijkstra algorithm on photo

Retarget method:	Greedy approach	Dijkstra algorithm
Image dimensions:	150 x 200	150 x 200
Accuracy	0.6515	0.7232
Time duration	0:00:07.538000	15:03:24.744456

Table 4.3.3 Run time and accuracy comparison of greedy approach and Dijkstra algorithm on image

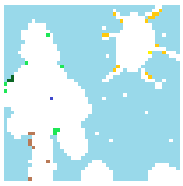
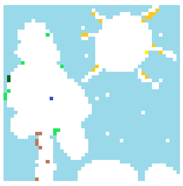
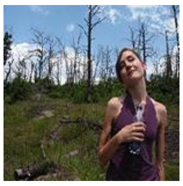
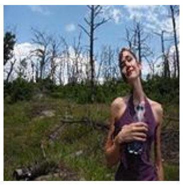
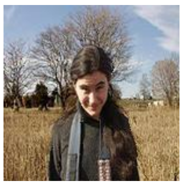
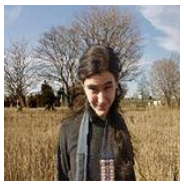
		
Retarget method:	Greedy approach	Dijkstra algorithm
Image dimensions:	50 x 75	50 x 75
Accuracy	0.8920	0.7184
Time duration	0:00:00.401485	0:09:10.639831

Table 4.3.4 Run time and accuracy comparison of greedy approach (*v1 & v2*) and Dijkstra algorithm on the image in NRID (*ours_11_aaa*)

		
Retarget method:	Greedy approach	Dijkstra algorithm
Image dimensions:	152 x 200	152 x 200
Accuracy (IoU)	0.7622	0.4791
Time duration (*Greedy v2**)	0:00:11.481114	14:15:04.679344
Time duration (*Greedy v1***)	0:02:06.419997	

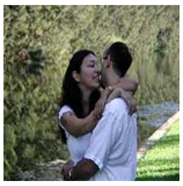
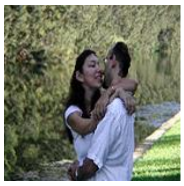
** Optimized Greedy Algorithm* *** Classic Greedy Algorithm*

Table 4.3.5 Run time and accuracy comparison of greedy approach (*v1 & v2*) and Dijkstra algorithm on the image in NRID (*ours_14_aaa*)

		
Retarget method:	Greedy approach	Dijkstra algorithm
Image dimensions:	152 x 200	152 x 200
Accuracy (IoU)	0.7507	0.6165
Time duration (*Greedy v2**)	0:00:09.344086	14:14:45.383034
Time duration (*Greedy v1***)	0:02:06.141000	

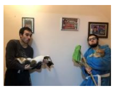
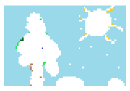
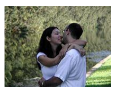
** Optimized Greedy Algorithm* *** Classic Greedy Algorithm*

Table 4.3.6 Run time and accuracy comparison of greedy approach (*v1 & v2*) and Dijkstra algorithm on the image in NRID (*ours_16_aaa*)

		
Retarget method:	Greedy approach	Dijkstra algorithm
Image dimensions:	152 x 200	152 x 200
Accuracy (IoU)	0.6964	0.5504
Time duration (*Greedy v2**)	0:00:10.062097	14:22:53.173311
Time duration (*Greedy v1***)	0:02:05.993500	

** Optimized Greedy Algorithm* *** Classic Greedy Algorithm*

Table 4.3.7 Run time and accuracy comparison of greedy approach (*v2*) and Dijkstra algorithm on the mean of all the images

	Average results of all the images		
Retarget method:	Greedy approach		Dijkstra algorithm
Image dimensions:	150 x 200 | 50 x 75 | 152 x 200  		
		Change	
Accuracy*	0.658	+6.55%	0.6175
Time duration** (milliseconds) (hh:mm:ss)	1,726,797 00:28:46	+ 2,347%	42,261,000 11:44:21

* The IoU average of 5 images for Dijkstra, and the IoU average of 23 additional images of these 5 images (28 images in total) for Greedy
** The run-time average of 5 images. Optimized Greedy run-time used.

First-level greedy approach gives ~6.55% better accuracy than the Dijkstra algorithm, as well as spends much less time; to be exact, the Dijkstra algorithm spends ~2,347% more time.

4.4 Visual Comparison with Other Methods

There are a lot of image-resizing methods, and crop & stretch are the most popular of these today. I use the same photo and image to show the differences between seam carving method.

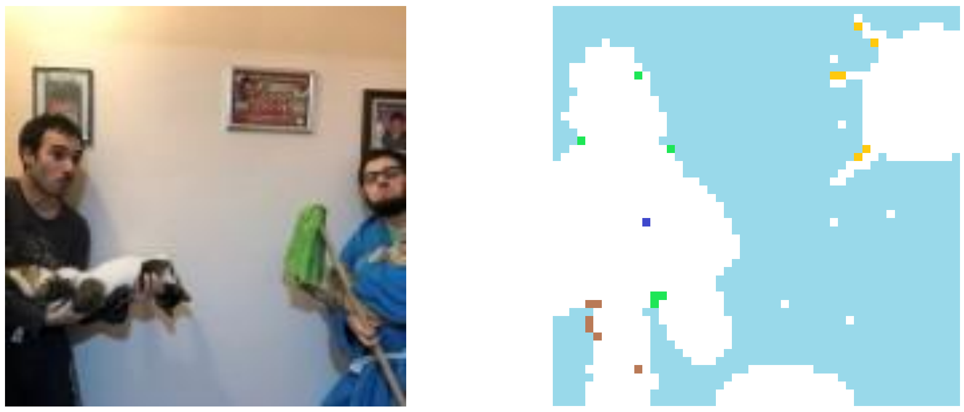


Figure 4.4.1 Cropped photo and image

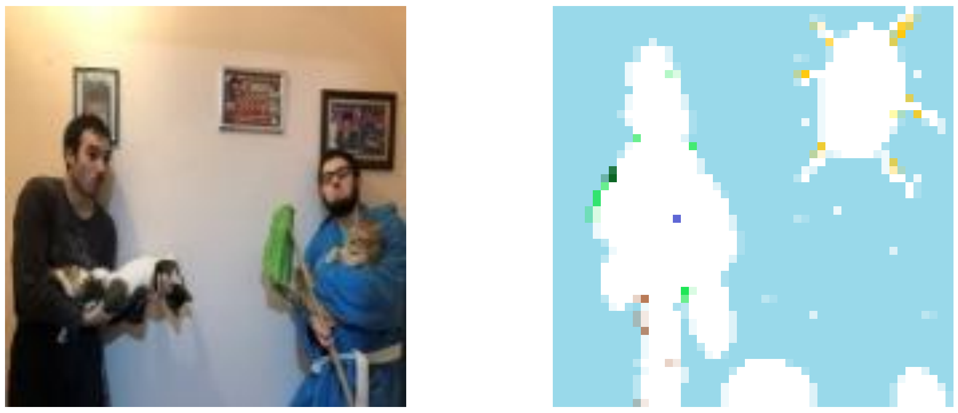
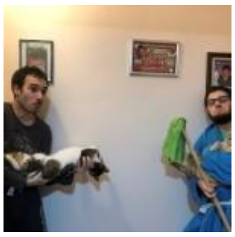
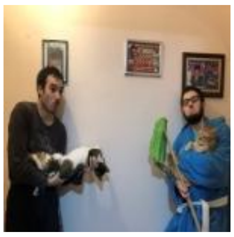


Figure 4.4.2 Stretched photo and image


Cropped


Stretched

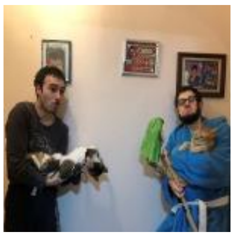
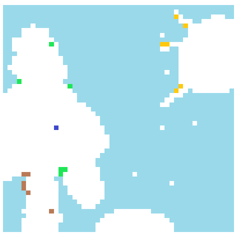
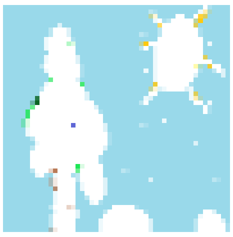

Greedy seam carving

Figure 4.4.3 Cropped vs Stretched vs Greedy seam carving of photo


Cropped


Stretched

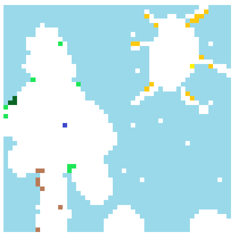
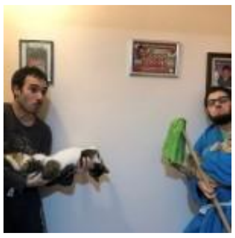
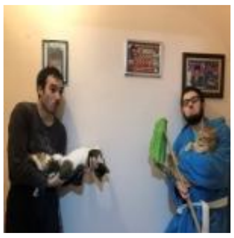

Greedy seam carving

Figure 4.4.4 Cropped vs Stretched vs Greedy seam carving of image


Cropped


Stretched

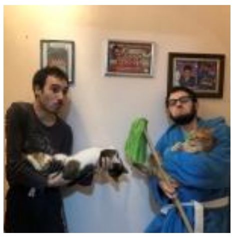

Dijkstra seam carving

Figure 4.4.5 Cropped vs Stretched vs Dijkstra seam carving of photo

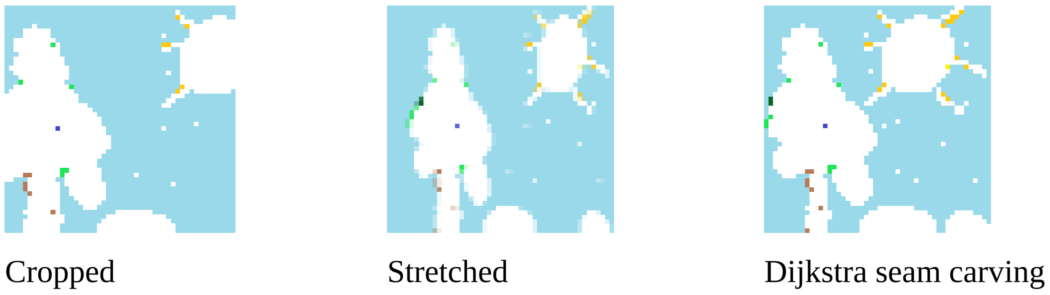

Cropped Stretched Dijkstra seam carving

Figure 4.4.6 Cropped vs Stretched vs Dijkstra seam carving of image

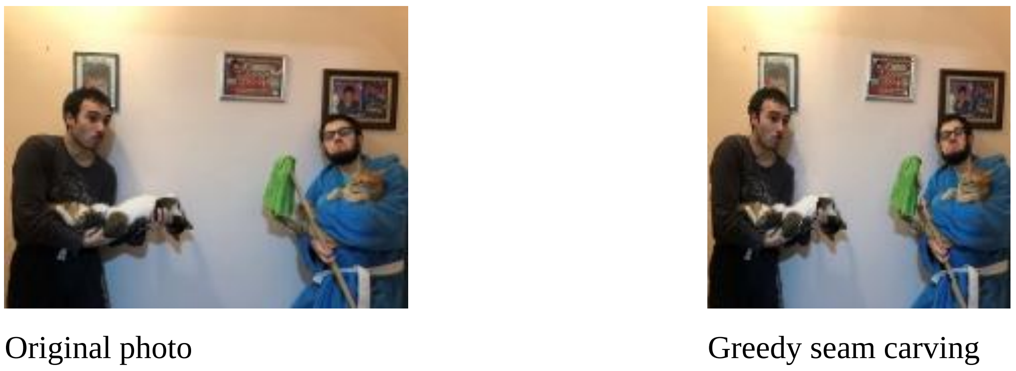

Original photo Greedy seam carving

Figure 4.4.7 Cropped vs Stretched vs Greedy seam carving of photo

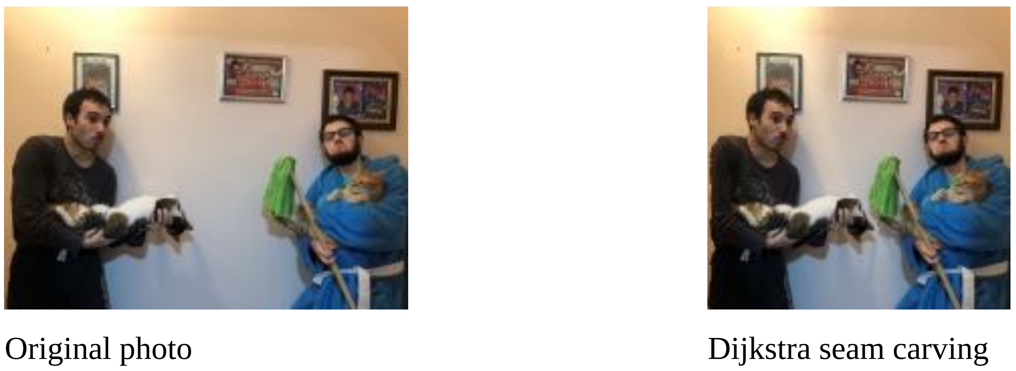
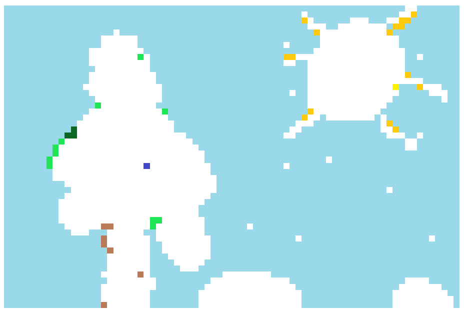
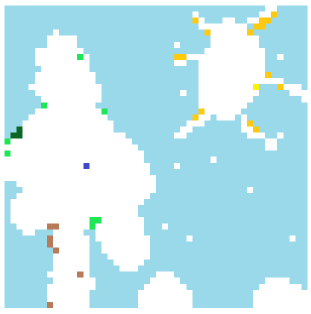

Original photo Dijkstra seam carving

Figure 4.4.8 Cropped vs Stretched vs Dijkstra seam carving of photo

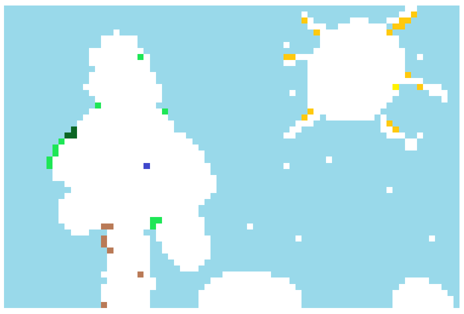


Original image

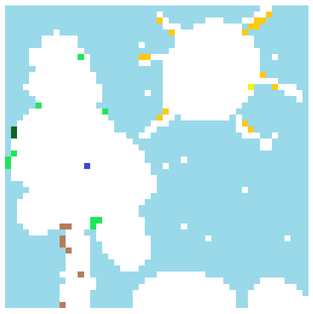


Greedy seam carving

Figure 4.4.9 Cropped vs Stretched vs Greedy seam carving of image

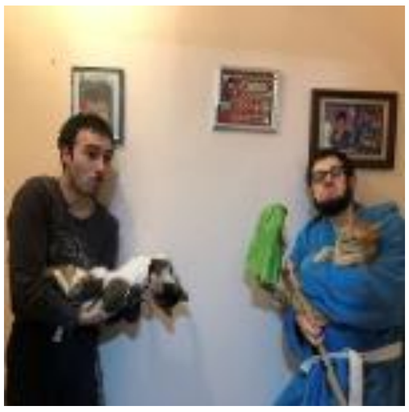


Original image

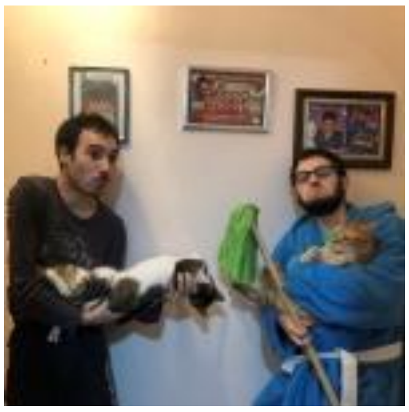


Dijkstra seam carving

Figure 4.4.10 Cropped vs Stretched vs Dijkstra seam carving of image

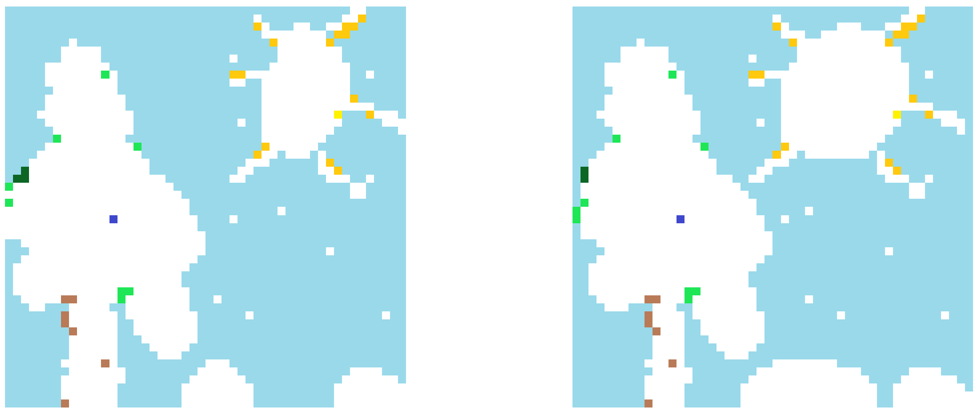


Greedy seam carving



Dijkstra seam carving

Figure 4.4.11 Greedy vs Dijkstra seam carving of photo



Greedy seam carving Dijkstra seam carving

Figure 4.4.12 Greedy vs Dijkstra seam carving of image

CHAPTER 5

5. CONCLUSION AND RECOMMENDATIONS

Based on this dataset I used, the Greedy approach provides higher accuracy than Dijkstra when averaging all of the images. The Greedy approach is more advantageous when using the seam carving method as the image retargeting method, since it achieves a ~6.55% better result than the Dijkstra and the Dijkstra algorithm has a ~2,347% longer processing time.

In order to develop the program, it is necessary to improve the shortest path algorithm. There can be several ways to do this. As used in the Greedy approach, in the Dijkstra algorithm, only the pixels affected in the previous iteration can be calculated, rather than going through the entire template in each iteration. Similarly, based on the width of the picture, the maximum distance it can travel from the selected pixel can be calculated, and in this way only the energy points of the necessary places can be calculated. Since Dijkstra is one of the best-known shortest path finding algorithms (Tyagi, 2020), there may be no need to use another method. Only the development of the algorithm can be achieved in a way that is more adaptable to this situation. Thus, the Dijkstra algorithm can be optimized for the seam carving method and its duration can be shortened.

In addition, a different approach can be taken for the detection of energy points in the image. The Laplacian filter we applied has difficulty in accurately identifying the important object in the foreground in complex and colorful images with a mixed background. Today, especially deep learning techniques provide great success in detecting important objects in the picture. First, the important objects in the picture are detected with deep learning, and after the Laplacian filter is applied, an algorithm can be developed to not use the pixels where the important objects are while performing image retargeting, no matter how low the energy points of those regions are.

REFERENCES

Abramowitz, M., Stegun, I. A. (2013). Laplace Transforms. *Ch. 29 in Handbook of Mathematical Functions with Formulas, Graphs, and Mathematical Tables* (9th Printing). New York: Dover.

Antoniadis, P. (2022, November 6). How to Convert an RGB Image to a Grayscale. Retrieved November 10, 2022 from https://www.baeldung.com/cs/convert-rgb-to-grayscale

Arens, E. (2022, June 05). Always up-to-date guide to social media image sizes. Retrieved October 30, 2022, from https://sproutsocial.com/insights/social-media-image-sizes-guide

Avidan, S., & Shamir, A. (2007). Seam carving for content-aware image resizing. *In ACM Transactions on Graphics.* Association for Computing Machinery (ACM). doi: 10.1145/1276377.1276390

Baum, J. (2020, 7 February). 5 Ways to Find the Shortest Path in a Graph. Retrieved November 15, 2022 from https://betterprogramming.pub/5-ways-to-find-the-shortest-path-in-a-graph-88cfefd0030f

Brandon. (2019, 14 November). How to Convert an RGB Image to Grayscale. Retrieved November 10 , 2022 from https://e2eml.school/convert_rgb_to_grayscale.html

Chen-Kuo Chiang, Shu-Fan Wang, Yi-Ling Chen, & Shang-Hong Lai. (2009, November). Fast JND-Based Video Carving With GPU Acceleration for Real-Time Video Retargeting. IEEE Transactions on Circuits and Systems for Video Technology, vol. 19, pp. 1588–1597. doi: 10.1109/tcsvt.2009.2031462

Ellard, N. (2021, 24 February). The Importance of Images in Social Media Part 1. Retrieved October 30, 2022, from https://www.urbanelement.com/insights/the-importance-of-images-in-social-media

Everingham, M., Van Gool, L., Williams, C. K., Winn, J., & Zisserman, A. (2010, June). The PASCAL Visual Object Classes (VOC) Challenge. *International Journal of Computer Vision*, pp. 303-308. doi: 10.1007/s11263-009-0275-4

Haralick, R. M., & Shapiro, L. G. (1991). *Computer and Robot Vision: v.1*. London: Addison Wesley.

Muhammad, A. (2021, 2 October). Which programming language to use for Image Processing. Retrieved November 1, 2022, from https://medium.com/@abdullah1621997/which-programming-language-to-use-for-image-processing-864d0110e695

Must Photos Always Be Rectangular? (2020, 19 September). Retrieved October 30, 2022, from https://www.moneymakerphotography.com/must-photos-always-rectangular

Raguraman, P., Meghana, A., Navya, Y., Karishma, Sk., & Iswarya, S. (2021, January). Color Detection of RGB Images Using Python and OpenCv. *International Journal of Scientific Research in Computer Science, Engineering and Information Technology*, pp. 109–112. doi: https://doi.org/10.32628/cseit217119

Redmon, J., Divvala, S., Girshick, R., & Farhadi, A. (2016). You only look once: Unified, real-time object detection. Proceedings of the IEEE conference on computer vision and pattern recognition, Las Vegas, 27 June- 30 June 2016, pp. 779-788, IEEE.

Ronneberger, O., Fischer, P., & Brox, T. (2015). U-net: Convolutional networks for biomedical image segmentation. Proceedings of the International Conference on Medical image computing and computer-assisted intervention, Munich, 5 October- 9 October 2015, pp. 234-241. Springer, Cham.

Rubinstein, M., Gutierrez, D., Sorkine, O., & Shamir, A. (2010). A comparative study of image retargeting. *Presented at the ACM SIGGRAPH Asia 2010 papers*. doi: 10.1145/1882262.1866186

Rubinstein, M., Shamir, A., & Avidan, S. (2008, August). Improved seam carving for video retargeting. ACM Transactions on Graphics, pp. 1-9. doi: 10.1145/1360612.1360615

Singh, V. V., & Kambhamettu, C. (2020). Feature Map Retargeting to Classify Biomedical Journal Figures. *In Advances in Visual Computing*, pp. 728–741. Berlin: Springer International Publishing. doi: 10.1007/978-3-030-64559-5_58

Sniedovich, M. (2006). Dijkstra's algorithm revisited: the dynamic programming connexion. *Control and Cybernetics Vol. 35*, pp. 599-620. http://eudml.org/doc/209437

Tyagi, N. (2020, 14 December). Dijkstra's Algorithm: The Shortest Path Algorithm. Retrieved November 18, 2022 from https://www.analyticssteps.com/blogs/dijkstras-algorithm-shortest-path-algorithm

Wu, J., Xie, R., Song, L., & Liu, B. (2019). Deep Feature Guided Image Retargeting. Proceedings of the 2019 IEEE Visual Communications and Image Processing (VCIP), Sydney, 1 December- 4 December, pp. 1-4. doi: 10.1109/vcip47243.2019.8966008

CURRICULUM VITAE