CSC420 Project Report - Image Inpainting

Team: asWeCan
Binwei Chen 1003065201
Yifan Liu 1002928102
Chenghao Wang 1003812966

Abstract

Image inpainting is the task of reconstructing damaged, deteriorating, or partially missed images, and filling the missing parts with both “visually and semantically plausible appeal” is the goal of image inpainting. The produced image should be plausible enough that any human eye will not be able to tell this is a processed image. The inpainting technique has various usages and applications, it is most often used to recover damaged portions of old photos or remove any unwanted objects from an image.

In our project, we decided to implement a general image inpainting program based on Criminisi’s paper in 2004, a program that can be used to remove small scratches and any unwanted object given the correct mask. We also adjusted parameters as well as related calculations to perform a comparative analysis based on the outputs over a set of testing images.

By comparing the outputs of different types of image input with various configuration settings, our algorithm shows promising results when dealing with structure and texture. However, it requires configuration re-adjustment every time running the program with a new image or mask and the efficiency of the program is low due to the brutal force match algorithm. Moreover, when the original input image lacks the information of unique or non-repetitive parts, using a deep-learning method with feature recognition ability will generate a better result.

Introduction and Literature Review

Image inpainting as a technique can be traced back to the Renaissance when artists were trying to repair deteriorated paintings. In 21th century, computer scientists took this technique into processing digital images on a computer. Nowadays there are numerous different ways to solve image inpainting, we divided them into two main categories, the traditional approach with mathematical models[1,2] and a more novel approach using a deep learning method with CNN or GAN[5,6,7].

Figure 1. Image inpainting example: restoration of an old, damaged photo[2]

The deep learning method works by building neural networks to solve the problem, by using a suitable architecture and training it against a large data set so that it will try to understand the type and the structures of images and capture the critical information from images. Finally, it will predict what the missing pixels should look like and fill the missing area. On the other hand, the main idea behind the traditional approach is based on the structural and textural nature of the image, by examining the information around the boundaries and using the remaining part on the image to fill the missing parts. In our project we mainly focused on the traditional approach of image inpainting.

Back to the traditional approach, the most intuitive way to fill a missing area is to draw the continuation of the lines on the boundary of the remaining image with the same color. Bertalmio introduces the method of plotting continuing lines based isophote lines around the structure around the boundary[2,4]. This method works very well on small scratches and the image better to be smooth, noiseless, and low texture. When filling in a high noise and texture image with a large block of missing area, this method will fail because the lines in the image can be irregular and non-continuous.

A solution to this problem is the texture synthesis approach. Many natural photos contain areas with “pure” textures, which are defined as repetitive two-dimensional textural patterns. For example, images of sea, sky, and forest are all images with “pure” textures.

These images usually contain rich texture content that allow algorithms to synthesize texture from the remaining image. Some exemplar-based techniques were introduced to effectively and cheaply generate textures[8,9,10]. One is a copy and paste texture synthesis that gradually fills the missing area from the boundary into the center. However this approach also has some disadvantages, the most important one is that it only focuses on the texture but loses the linear structures of the image, the resulting image is likely to have discrete boundaries and distorted lines.

An exemplar-based method was introduced in *Region Filling and Object Removal by Exemplar-Based Inpainting*[1], which presents a novel and efficient approach that combines the advantages of the above two approaches. The exemplar-based method calculates a reasonable filling order based on the linear structure around the boundary to ensure the correct structure propagation and then uses the copy-and-paste texture synthesis approach to fill in the missing area.

The algorithm from the paper is applied in three main steps:

1. Computing patch priorities:

   When filling a missing area, the existing structure should be continued and extended and hence boundary points of the missing area are considered the top priority, which we call the ***fill-front***. Among the points from the fill-front, the ones with a continuation of image structures should obtain a higher priority. Another factor is the amount of the extra information around the boundary points, with more information around the point it is easier to synthesize the correct texture. Thus these points should also obtain a higher priority. Combining the two factors together, the overall priority is calculated as $P(p)=C(p)D(p)$, where p is the point from the fill-front.

   C(p), which we call the ***confidence*** term, is computed as the sum of confidence values of each pixel from the patch centred at point p divided by the area of the patch, i.e. the average confidence value over the patch. Note that the confidence value of each pixel is initialized as 1 for presented pixels from the original image and 0 for any missing pixels to be filled. D(p), which we call the ***data*** term, is computed as the norm of the dot product of the isophote vector of the patch p and unit normal vector at the point p, divided by a normalization factor (255 for a gray-scale image). Note that the isophote vector is calculated as the maximum gradient from the patch rotated by 90 degrees.

2. Propagating texture and structure information:

   After locating the priority point, the patch surrounding the point should be filled first. The patch will be filled from another path extracted from the remaining image.

By looking for the most similar path with the existing information ensures the propagation of the texture and structure. The degree of similarity between two patches is compared by the sum of squared differences of the filled pixel values between two patches, we call the value ***patch difference***.

3. Updating confidence values

After a patch has been filled the confidence value is updated for the newly filled pixels. It measures the amount of reliable information for the next filling process, it could also be seen as the amount of confidence we have for the patches on the boundary. This also follows human intuition, as the filling proceeds into the inner area of the missing region we are less and less confident about the correctness of the information we have.

In fig.2, it demonstrates the expected ordering of filling using the algorithm proposed. The program prioritizes filling the fill-front points in the middle as they have greater values of the ***data term*** because of the boundary line between two different colored blocks. Plus, all the points share the same ***confidence term*** value as other fill-front points because they form a perfect circle. Therefore, the filling starts from there and moves toward the center.

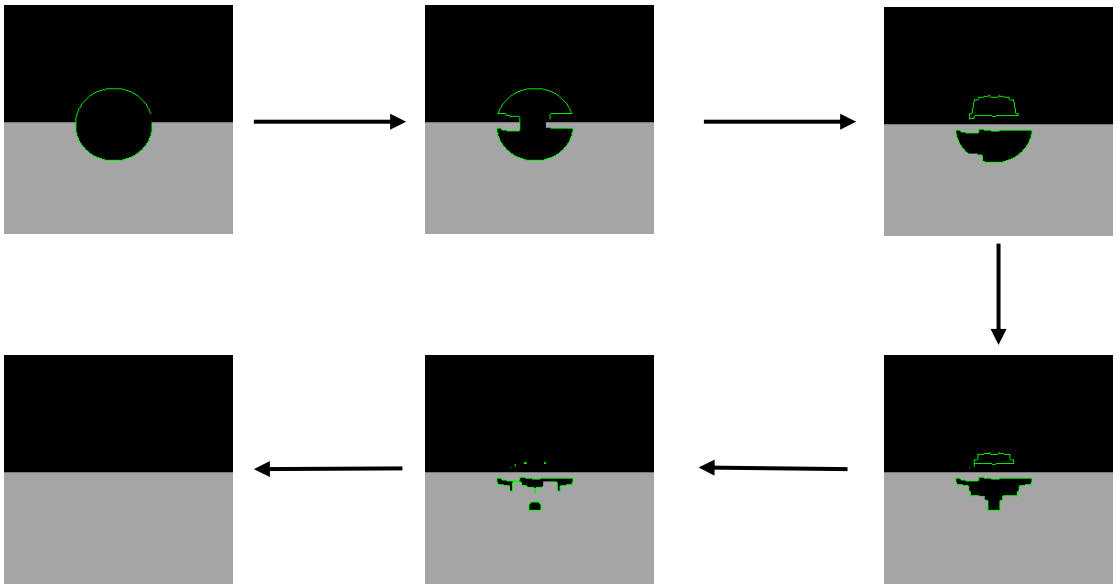

Figure 2. Ordering of the filling by using the exemplar-based patch synthesis[1]

Methodology

About the program:

In this project, we design and implement a program to perform image inpainting tasks. The **input files** of the program are the original (undamaged) RGB image and a corresponding

binary mask image, as in fig.2. The unwanted area to be removed has no limitation of size or position, but the overall shape of the mask image has to be the same as the original image. The mask being binary means each pixel is either in white (255) or in black (0) colors. The program can locate the missing part and fill it. The **output** will be the restored image after the process, with the same shape as the input image, as the fig.3 in the example shown below.

Figure 3. Examples of the input image[3] and mask

Figure4. Example of output

Behind the program:

We implement the image inpainting program mainly based on the algorithm introduced from the paper, *Object Removal by Exemplar-Based Inpainting*[1]. Two reasons behind the decision of choosing the traditional method over neural networks are: 1) The traditional method is overall more generalized than deep-learning methods, which matches with the goal of the project, a general image inpainting program. 2) Training an image-inpainting neural network takes a huge dataset of trainable images, which we hardly have the access to, plus training the network costs much more extra time. Thus, due to the time constraint and the nature of the project, we decide to use the traditional approach for the implementation instead of applying the neural network approach.

After the initial implementation, we run a few test examples with it and then are able to identify an obvious insufficiency of the program. The example output in fig.5 clearly illustrates the issue.

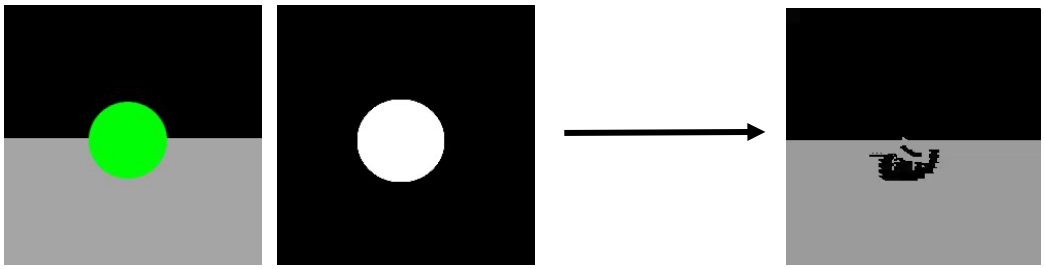

Figure 5. The input image[2], mask, and output filled image

In the fig.5, it shows an input image and its mask, two blocks of different colors stacked together with a hole in the middle at the boundary line. Theoretically, the output image would be a color block stacked together without the hole. However, the initial output is rather unexpected. Then we realise that the reason for polluting the grey area with black color traces back to the bottom pixels of the hole. Since we only consider filled pixels when comparing patches, the very bottom patch is just a block with the lower half filled with grey color and patches satisfying such a condition yield the same patch similarity. Hence, the first satisfied patch ends up being the chosen one and it is the first patch lying on the boundary line. Consequently, the other half is filled with black color.

Therefore, we add one more factor when comparing the patches, the Euclidean distance of the two patches, we call this value ***patch distance***. We also introduce a weight for each factor and compare the sum of two weighted factors to decide the similarity of patches. In this project, we use a weight in the range of 0 to 1, and 1 by default.

Experiments & Results

Testing Set

We search through academic journals[1,2] and online web pages[3], and eventually collect ten pairs of image-masks. The testing images (fig.4) range from simple geometric shapes to real-life photographs of landscapes and people. All the masks (fig.5) are created by either simply creating Numpy Array and output as an image file or manually drawing layers using Microsoft Paint 3D.

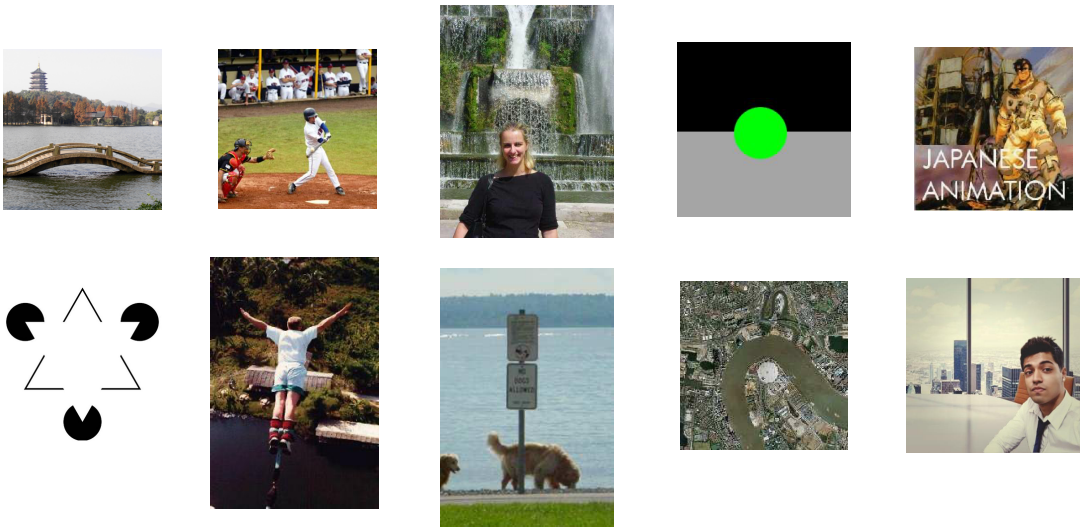

Figure 4. Testing image set

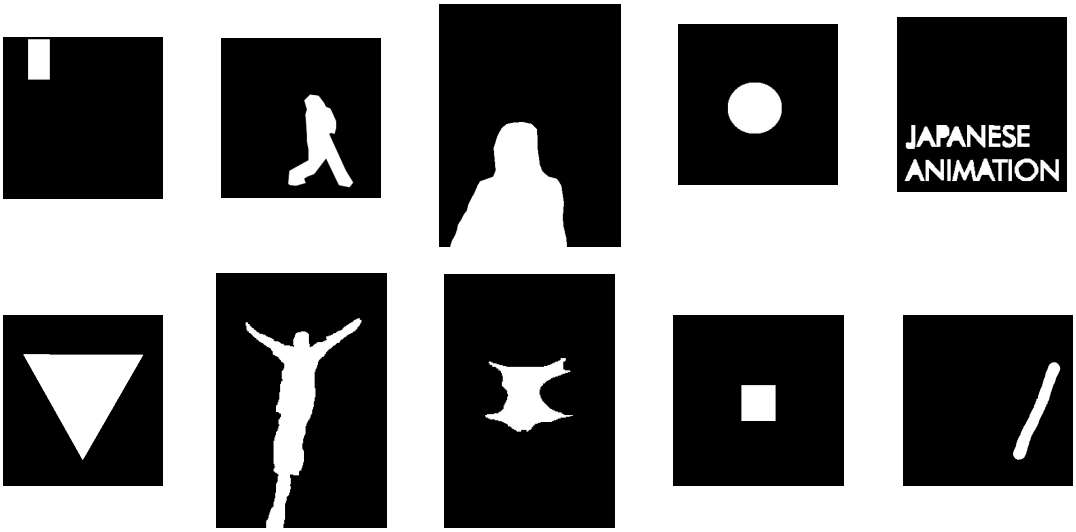

Figure 5. Matching mask set

Testing Runs:

There are three configurable parameters in the program, patch-size, patch difference weight, and patch distance weight. Besides the patch size 9, proposed from the paper[1], we also test the effect of increasing and decreasing it, 13 and 5. The configuration of each parameters that we test are listed below and we run each of the testing image using all the possible configuration combinations:

- patch size: 5, 9, 13
- patch difference weight: 0.5, 1
- patch distance weight: 0, 0.5, 1

Testing results:

We examine and compare the output of all test runs and manually pick out the best result for each testing image (see next two pages). Optimum Parameter configuration is displayed in such format:

{patch size} - {patch difference weight} - {patch distance weight}

img1: 9 - 1 - 0

img2: 9 - 1 - 0

img3: 9 - 1 - 1

img4: 9 - 1 - 1

img5: 9 - 1 - 0.5

img6: 9 - 1 - 1

img7: 9 - 1 - 0.5

img8: 9 - 1 - 0.5

img9: 9 - 1 - 0.5

img10: 9 - 1 - 0.5

Figure 6. Best results for each of the testing image with the corresponding configuration

Analysis:

- From the unpicked results, it is believable that increasing the patch size from 9 to 13 does not have much improvement on the output. Instead, it makes certain high-frequency textures more discrete, which is expected as larger patches implies more pixels filled in one step, and the outcome effect is obviously less smooth. On the other hand, a pixel size of 5 does not explicitly improve the quality of outputs nor degrade it. However, it takes much longer time to proceed with an image, which severely decreases the efficiency of the program.

- Geometric shapes, like img4 from fig.6, tend to require large weight on the patch distance, meaning to enforce the program to find patches from the surrounding. The intuitive reasoning behind this result is, that the shape tends to be continuous when the patches are close. For instance, a straight line fragment tends to be extended by another straight line. However, the drawback of overweighting on the distance causes another issue. For example, in the img6, the circles are far away from each other and hence to fill the missing part of a circle the program may need to go to the other side of the picture to locate the matching patch. But it instead uses a closer patch to fill the circle and yield a less ideal output.

- For some test cases, like img1, 2, 5, 7, and 9, the outputs are decent and relatively believable. The only major disadvantage is the low efficiency due to brutal force matching.

- For high-frequency images, like img3 , due to the complexity and nature patterns underlying the images, such as water waves or clouds, a more delicate balance of the patch difference and the distance is required. Nature patterns are usually periodic and hence overweighting the distance value could cause the pattern to be discrete.

- The program by far is not able to generate decent outputs, not even close, for img10, a missing human face. It is because of the limitation of the traditional image inpainting method, that it can not reproduce information that is missed from the filled areas since it borrows from the present pixels. In img10, there is only one human face in the image and every part of the face is unique, such as the right eye, mouth, and nose. Thus, deep-learning methods will be more suitable for this type of task.

Result Comparison with Neural Networks:

Our algorithm also has some disadvantages, the first problem is that this algorithm only uses the existing information in the remaining image to fill up the missing area, if there does not exist an extra patch in the image that is similar to the patch we want to fill our

algorithm will likely to fail. This algorithm works better on the low frequency and repetitive texture dominated images because they usually contain the extra redundant information to complete the inpainting.

For image inpainting algorithms using neural networks, this would not be a problem, since they could use the information from a large data set of images with various types. The deep learning algorithms can also recognize the high-level feature of the image, and use these features to direct the generation of the missing area. Insead of learning features from the image, our algorithm works on the structure and the texture of the image. However our algorithm doesn't need to train the entire data set before any inpating start.

Figure 7. Inpainted by our program vs. by deep learning methods [3]

The example result shows this property, the deep learning methods successfully recognize the human face and generate a plausible face as a result. Our algorithm on the other hand couldn't fill up the missing area because there is no extra left face on the remaining image, unless we consider adding another step involving rotation or other types of transformation to the patch, but that will greatly increase the run time and the decrease the efficiency of the algorithm.

Conclusion

We implemented an algorithm that can remove objects from the image with a properly sized mask in a visually plausible manner and seems reasonable to the human eye. Our

approach uses an exemplar-based method with a priority term that determines the filling order. The priority term takes both the confidence level of the information around the pixel and the existence of any linear structure on the boundary into consideration. The algorithm for comparing the similarity of two patches is calculating the weighted sum of the norm of Sum of Square Difference and the Euclidean distance between two patches. This algorithm can be used to restore a damaged image as well as removing large objects from the image. However, if a significant or special area of the image is missing the results will not be promising, using a deep learning image inpainting algorithm will be a better choice.

Contributions

Binwei Chen:

- Implemented the image inpainting program based on the initial structure
- Tested and debugged the final image inpainting program
- Collected images and masks for the test image set
- Collected and analysed program outputs

Yifan Liu:

- Worked on literature review looking for potential pros and cons for each approach
- Built the initial outline and program structure at the early-stage
- Assisted with implementation of the image inpainting program
- Tested and debugged the final image inpainting program
- Collected images and masks for the test image set
- Helped with the analysis of results

Chenghao Wang:

- Collected images and masks for the test image set
- Helped with literature review looking for results example for each approach
- Tested and debugged the final image inpainting program
- Worked on the presentation slides

References

[1]A. Criminisi, P. Perez and K. Toyama, "Region Filling and Object Removal by Exemplar-Based Image Inpainting", IEEE Transactions on Image Processing, vol. 13, no. 9, pp. 1200-1212, 2004. Available: 10.1109/tip.2004.833105 [Accessed 15 December 2020].

[2] M. Bertalmio, G. Sapiro, V. Caselles and C. Ballester, "Image inpainting", *Proceedings of the 27th annual conference on Computer graphics and interactive techniques - SIGGRAPH '00*, 2000. Available: 10.1145/344779.344972 [Accessed 16 December 2020].

[3]J. Khan, "Guide to Image Inpainting: Using machine learning to edit and correct defects in photos", *Heartbeat*, 2019. [Online]. Available: https://heartbeat.fritz.ai/guide-to-image-inpainting-using-machine-learning-to-edit-and-correct-defects-in-photos-3c1b0e13bbd0. [Accessed: 16- Dec- 2020].

[4] M. Bertalmio, A. Bertozzi and G. Sapiro, "Navier-stokes, fluid dynamics, and image and video inpainting", *Proceedings of the 2001 IEEE Computer Society Conference on Computer Vision and Pattern Recognition. CVPR 2001*. Available: 10.1109/cvpr.2001.990497 [Accessed 18 December 2020].

[5] D. Pathak, P. Krahenbuhl, J. Donahue, T. Darrell and A. Efros, "Context Encoders: Feature Learning by Inpainting", *Openaccess.thecvf.com*, 2020. [Online]. Available: https://openaccess.thecvf.com/content_cvpr_2016/html/Pathak_Context_Encoders_Feature_CVPR_2016_paper.html. [Accessed: 18- Dec- 2020].

[6] R. Yeh, T. Yian Lim, A. Schwing, M. Hasegawa-Johnson and M. Do, "Semantic Image Inpainting With Deep Generative Models", *Openaccess.thecvf.com*, 2020. [Online]. Available: https://openaccess.thecvf.com/content_cvpr_2017/html/Yeh_Semantic_Image_Inpainting_CVPR_2017_paper.html. [Accessed: 18- Dec- 2020].

[7] Y. Wang, X. Tao, X. Qi, X. Shen and J. Jia, "Image Inpainting via Generative Multi-column Convolutional Neural Networks", *arXiv.org*, 2020. [Online]. Available: https://arxiv.org/abs/1810.08771. [Accessed: 18- Dec- 2020].

[8] A. Efros and W. Freeman, "Image quilting for texture synthesis and transfer", *Proceedings of the 28th annual conference on Computer graphics and interactive techniques - SIGGRAPH '01*, 2001. Available: 10.1145/383259.383296 [Accessed 18 December 2020].

[9] A. Efros and T. Leung, "Texture synthesis by non-parametric sampling", *Proceedings of the Seventh IEEE International Conference on Computer Vision*, 1999. Available: https://www2.eecs.berkeley.edu/Research/Projects/CS/vision/papers/efros-iccv99.pdf. [Accessed 18 December 2020].

[10] W. Freeman, *International Journal of Computer Vision*, vol. 40, no. 1, pp. 25-47, 2000. Available: 10.1023/a:1026501619075 [Accessed 18 December 2020].