Implementation and Evaluation of Various Inpainting Techniques

Jason Kim, Mikako Inaba

1 Introduction

Image inpainting refers to the problem of filling in unwanted regions of an image. Image inpainting has many applications, from restoring damaged photographs to eliminating watermarks or other artificial image features, but the primary application that we aim to address is the challenge of seamlessly "deleting" objects from an image while minimizing any resulting artifacts in that image. Whether for the purpose of removing background photo-bombers in a romantic photograph or to remove people and cars from a pristine national park vista, object-deletion using inpainting is both a desirable practical tool and also a significant technical challenge. While many inpainting applications focus on removing small features and errors, object-deletion inpainting often involves removing significant portions of an image which increases the difficulty of the problem.

The basic set up for an inpainting problem involves some input image and a mask. A mask refers to a region of the image that needs to be "deleted" or inpainted. Oftentimes, in practice, a mask comes in the form of a black and white image with the same height and width of the original image where white pixels are used to designate all pixels that need to be inpainted and all black pixels designate pixels in the original image that can be kept the way they are. In the case of object-deletion, the mask is used to designate the object that needs to be deleted. See Figure 1.

Figure 1: Example of object deletion using inpainting. On the left is the original image, center is the mask, the result of the object deletion is on the right. Images from Yang et. al. [1]

The primary goal of this project was to implement three methods for inpainting (one diffusion-based, one exemplar-based, and one exemplar-based implementation with some of our own novel additions) and to compare and evaluate their results on object-deletion tasks. In addition, while many traditional inpainting algorithms require a human to manually specify the mask of the object to be deleted, a further goal of this project was to integrate our inpainting algorithms with Mask R-CNN developed by He et. al [2] to automatically recognize objects within an image and delete certain classes (people, dogs, cars, etc).

2 Previous Work

While much work has been done in the field of inpainting and image completion, this task remains a challenging problem. Inpainting methods can largely be categorized in three groups: diffusion-based, exemplar-based, and deep-learning-based methods.

2.1 Diffusion-Based Inpainting

Among the first attempts at implementing inpainting algorithms involved diffusion-based methods. Diffusion-based methods were primarily intended to fill in small holes or narrow lines in an image, usually to reconstruct damaged parts of an image or to cover up small features in the background of an image. Diffusion-based methods generally work by propagating image pixel values or image structures adjacent to the mask [3].

While diffusion-based methods are generally effective at filling in small regions, they become ineffective when used on larger, more textured regions of an image. Because diffusion-based inpainting works by propagating information from adjacent known pixels in an image and cannot synthesize complex textures or image patterns and often create blurred or "paintbrush" artifacts on images [4] as seen in Figure 2. Some diffusion-based inpainting implementations such as Bertalmio et al. [5] attempt to solve this problem by propagating textures based on image gradient information, but such implementations still run into similar problems where textures are "spread over" the mask region in a paintbrush-like fashion.

Figure 2: An example of a common artifact from diffusion-based inpainting where the original mask region is blurred or has a paintbrush effect. Images from Criminisi et. al. [6]

2.2 Exemplar-Based Inpainting

Exemplar or "patch"-based inpainting methods are an alternate method of inpainting that work by considering a neighborhood of known pixels around pixels that need to be inpainted, and then

finding a close match (in pixel values, image gradients, or other features) somewhere else in the image. Once the closest possible "patch" is found, its pixel values are used to fill in the unknown pixel values in the mask [6]. Because exemplar-based inpainting methods use actual image patches, they can often generate good results with convincing textures and image features for even large mask sizes. Some versions of exemplar-based inpainting such as Huang et. al. [4] and Barnes et. al [7] use additional constraints (often in the form of human/user-defined lines or curves) to help the algorithm to recognize certain structures within the image such as columns, roads, or window panes that need to be continued/preserved in the mask region. The primary weakness of these methods are that they are still constrained to information within the image itself and thus cannot generate completely new image features that are nonexistent in the original image [8] (see Figure 3). In addition, exemplar-based inpainting can result in artifacts where the vague outline of the original mask can be seen if the patches used to inpaint are not close enough matches (see Figure 4).

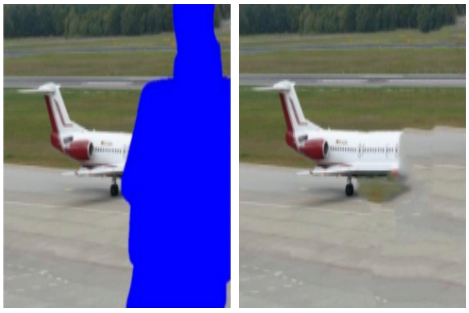


Figure 3: An example of a common artifact from exemplar-based inpainting where the algorithm is unable to generate structures that do not appear elsewhere in the image. Images from Zheng et. al. [8] The PatchMatch [7] algorithm was used for this image.

Figure 4: An example of a common artifact from exemplar-based inpainting where the outline of the original mask is still visible. Images from Zheng et. al. [8] The PatchMatch [7] algorithm was used for this image.

2.3 Deep-Learning-Based Inpainting

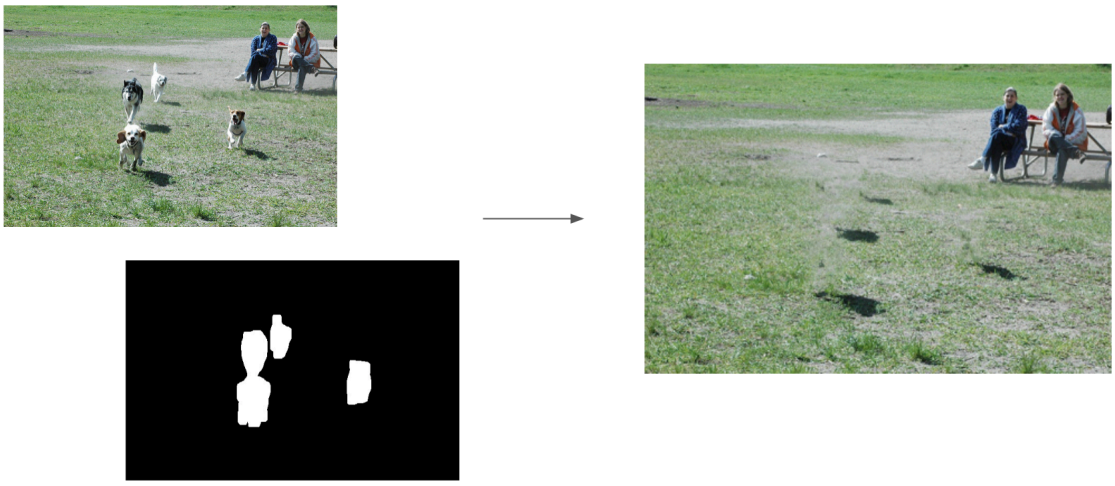
Deep-Learning-based inpainting methods have the significant advantage that they draw on large data sets that allow them to construct or fill in image features that are not present in an image itself, yet would be understood by humans to exist (see Figure 3). Pathak et. al [9] first made an attempt at inpainting using a convolutional neural network, using adversarial loss inspired by Generative Adversarial Networks. Since then, much work has been done to refine this method. Iizuka et. al [10], for example used two discriminators for adversarial training to make inpainting more locally and globally consistent. Zeng et. al [8] combined CNN approaches with an iterative approach to inpainting by generating a confidence map of inpainted pixels and retaining only high-confidence pixels after each iteration until the entire mask is filled.

3 Design and Implementation

3.1 System Overview

The goal of our project was to build a system to remove objects, specified by a mask, and seamlessly continue the background where the objects originally were. Our system takes an image, a mask, the patch size, and the inpainting method as the input and outputs the original image with the mask removed.

First, a mask must be created to define the object or the region to be removed. Then, the specified inpainting algorithm is used to fill in those regions. The implementation of our 3 inpainting methods are detailed below. Figure 5 shows sample inputs and the corresponding output. Note that the system only inpaints the mask such that artifacts of the removed objects such as the shadows of the dogs remain in the output image.



```
inpaint(img, mask, 3, EXEMPLAR)
```

Figure 5: Example of an input image and mask and the image outputted by our inpainting system using Exemplar-Based Inpainting. The last 2 function parameters specify the patch size and the inpainting method to be used.

3.2 Creating the Mask

Our system does not specify the method for creating the mask. Past papers have created random masks or created an interface to outline the region to remove. The only requirement is that the mask has a pixel value of 255 (white) for regions to be removed and a pixel value of 0 (black) elsewhere.

For our experiments, we performed object detection using a model pre-trained on the MS COCO data set. This implementation of Mask R-CNN mostly follows the original paper [11], but resizes images and uses a different learning rate. There are 81 different output classes, ranging from people and dogs to donuts and airplanes.

For a given image, the model outputs bounding boxes and masks for each detected object, along with confidence scores for the class. Figure 6 shows an example of the detection of 4 dogs, 2 people, and a bench. These colored masks were used as the mask input to our system.

Figure 6: Example of an output from the pre-trained Mask R-CNN model.

We found that the outputted masks often excluded some parts of the detected object. For example, looking at the person on the left, identified by the green mask, a part of her right ear and shoulder and parts of her shoes are not covered by the mask. So, we modified the masks by adding padding around the masks before passing them into our system. Figure 7 shows the modified version of the masks in Figure 6. Note that unlike the colored masks overlaying the original image, the mask is black everywhere except where the dogs are in the input image.

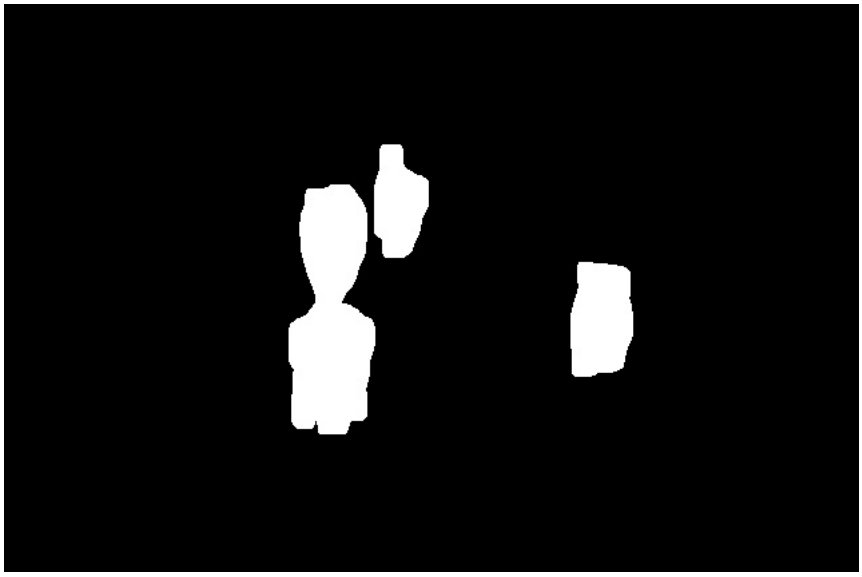


Figure 7: Example of an input mask. Each dog mask was padded 5 pixels around the outline of the mask.

3.3 Inpainting

Given the original image and the padded masks, the following inpainting methods were implemented to determine pixel values for the mask regions.

Fast Marching Method. The Fast Marching Method [3], created by Telea in 2004, works by defining a "band" which refers to the outermost pixels of the mask. The Fast Marching Method inpaints pixels one at a time in order of the distance of pixels in the mask to the original band (the original outline of the mask at the start of the algorithm). Thus, the algorithm proceeds by inpainting pixels on the outermost edge of the mask and works its way into the interior of the mask until all the pixels are inpainted. On each iteration, the pixel on the band with the smallest distance to the original band (i.e. the outermost pixel in the mask) is removed from the band and all of its neighbors within the mask are inpainted and added to the band (i.e. the boundary of the mask is marched inward).

Inpainting of a single pixel in the Fast Marching Method works by taking a weighted sum of pixels in the neighborhood of a pixel (defined by $B(\epsilon)$ see Figure 8). For all points q that are in the neighborhood $B(\epsilon)$ of pixel p, we calculate the vector $\mathbf{r}$ from the pixel q to the pixel p and the image gradient ∇I at q. We also calculate the weight we use in our weighted sum for the pixel q as $w(p,q) = \text{dir} \times \text{dst} \times \text{lev}$ where:

$$\text{dir}(p,q) = \frac{p-q}{\|p-q\|} \cdot N(p)$$

$$\text{dst}(p,q) = \frac{d_0^2}{\|p-q\|^2}$$

$$\text{lev}(p,q) = \frac{T_0}{1+|T(p)-T(q)|}$$

$\text{dir}(p,q)$ refers to the directional component of the weight and adds greater preference for pixels that lie along the normal vector ($N(p)$) of the pixel p to the band (see Figure 8). $\text{dst}(p,q)$ refers to the geometric distance component of the weight which decreases the weight for pixels in the neighborhood that are farther from the pixel p. $\text{lev}(p,q)$ refers to the level set distance component which weighs pixels close to the contour through p heavier than farther pixels. Note that $T(p)$ and $T(q)$ refer to the distance of pixels p and q respectively from the original band. T_0 and d_0 are reference distances which in practice are set to the interpixel distance (i.e. 1).

Finally, we can inpaint the pixel p, setting its new value to:

$$I(p) = \frac{\sum_{q \in B(p)} w(p,q)[I(q) + \nabla I(q)\mathbf{r}]}{\sum_{q \in B(p)} w(p,q)}$$

where $I(q)$ refers to the pixel value for q (Note that for color images, we run the inpainting calculations separately for each of the three channels).

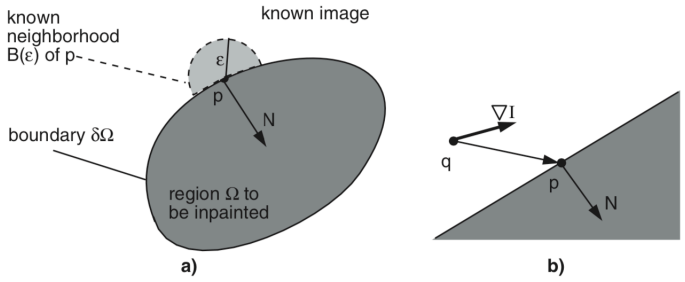


Figure 8: Diagram of inpainting in the Fast Marching Method for pixel p. A neighborhood with radius ϵ is used to calculate the new value for the pixel p. For each pixel q in the neighborhood that lies outside of the mask (Ω), we calculate the image gradient at q (∇I), the vector from q to p ($\mathbf{r}$) and the normal vector $N(p)$ for the pixel p with respect to the band. Diagram by Telea [3].

Exemplar-Based Inpainting. Criminisi et al [6] proposed an exemplar-based image inpainting algorithm that defines a priority term for each pixel on the band of the mask and inpaints patches of pixels at a time. Each iteration, the patch to be inpainted, defined by its center pixel, is determined

using a priority function. $P(p)$ the priority of a pixel p on the band is defined as the product of the confidence and data term of p, $C(p)D(p)$ where:

$$C(p) = \frac{\sum_{q \in \Psi_p \cap (\mathcal{I} - \Omega)} C(q)}{|\Psi_p|}$$

$$D(p) = \frac{|\nabla I_p^{\perp} \cdot n_p|}{\alpha}$$

See Figure 9 for definitions of each variable. Initially, $C(p) = 1$ for all known pixels and $C(p) = 0$ elsewhere (pixels in the mask). The confidence term is a measure of the reliability of the pixels in the neighborhood of p. The data term is a measure of isophotes hitting the band at p. Together, the priority term prioritizes patches with more known pixels and linear structures.

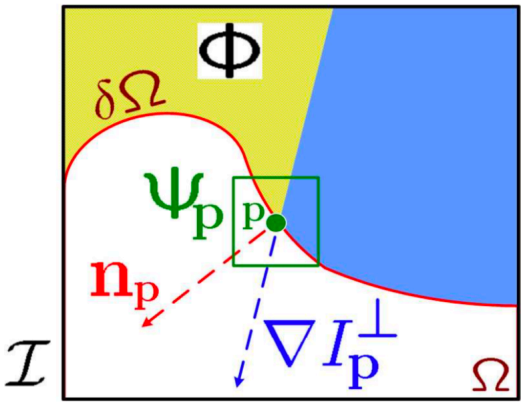


Figure 9: Diagram for calculating the priority of pixel p. The green box labelled Ψ_p outlines the pixels that are a part of the patch of p. $\mathcal{I}$ denotes the input image, Φ denotes the known pixels, Ω denotes the mask region, and $\delta\Omega$ denotes the band. n_p is the unit vector orthogonal to the band at p and $\nabla I_p^{\perp}$ is the direction and intensity of the isophote at p. α is a normalization term, in our case 255. Diagram from Criminisi et al [6].

First, we find the band pixel with the highest priority p'. Then, we find the exemplar, the patch in Φ (see Figure 9) that is most similar to the patch centered at p' where the distance between the two patches is defined as the sum of square differences of the known pixel values. The pixel values from the exemplar are used to fill in the unknown values of the patch centered at p'. Finally, the confidence terms for all newly inpainted pixels are updated. This process is repeated until all the pixels have been filled in. Note that the band and priority terms change every iteration.

HSL Exemplar-Based Inpainting Our final inpainting implementation was an attempt at improving the base exemplar-based inpainting approach. The primary difference we incorporated was the conversion of the input image into the HSL (hue, saturation, lightness) color space before

executing the exemplar-based inpainting algorithm. After the algorithm is complete, we convert the image back into RGB values. The motivation behind using HSL as opposed to RGB was to help the algorithm find "better" patches to use for inpainting. While standard exemplar-based inpainting finds patches based on their closeness in raw RGB values to the surrounding pixels of an unknown region, HSL exemplar-based inpainting find patches based on their closeness in values that are perceptually more meaningful (hue, saturation, lightness). The goal is to find patches so that the final inpainted region looks less "patched" together with pixels chosen from regions of the image that may not be properly representative of the brightness/lighting conditions of the region that needs to be inpainted. Instead, HSL allows the algorithm to take factors like varied lighting conditions in different parts of the image into account when selecting patches to use for inpainting.

4 Analysis

4.1 Data Set

To evaluate our inpainting algorithms, we chose 10 images from the COCO validation set. When choosing the images, we considered many characteristics such as the number of objects, the size of the objects (salient, background objects, etc), the background (natural, repeating structures, etc), and the type of object (animal, food, vehicle, etc). Figure 10 shows the 10 images and the corresponding masks used. Removal of the airplane from the gray sky and the two airplanes from the blue sky were meant to be "easier" examples while removal of the banana and the donuts were meant to be "harder" tasks.

Figure 10: The 10 images (some repeated) from the COCO validation set chosen to evaluate our inpainting methods. The corresponding masks, generated using Mask R-CNN are shown below each image.

4.2 Human Evaluation

To evaluate the implemented algorithms, we had 5 people rank the inpainted images. The outputs were presented in 15 sets of 4 images. Each set contained an image (from the set of images in Figure 10) inpainted using the Fast Marching Method with a patch size of 7, Exemplar-Based Inpainting with a patch size of 7, Exemplar-Based Inpainting with a patch size of 13, and HSL Exemplar-Based Inpainting with a patch size of 7. Each of these methods will be abbreviated as FMM, Exemplar, Exemplar_13, and HSL, respectively. The order of the images were randomized and labelled $1-4$ such that there were no indicators of which algorithm was used and the outputs from the different methods were presented in a different order each set. The evaluators were asked to rank the images in each set from best to worst in terms of how seamlessly the object(s) were removed.

All the rankings given by the evaluators are summarized in Table 1. Only 4 sets of images received the same rankings from everyone, demonstrating the subjective nature of the quality of an inpainting algorithm. As such, there was no clear "best" inpainting algorithm, also evidenced by the close average rankings in Table 2. However, FMM consistently did the worst, with an average ranking of 3.92 (ranking of 4 in all but 3 out of 75 evaluations). Note that analyzing rankings is difficult as the rankings do not take into account if one image was much better or worse than another image or if the two were similar in quality.

Figure 11: Example of a set of images (Set 13) presented to evaluators to be ranked. The top left is HSL, the top right is Exemplar, the bottom left is Exemplar_13, and the bottom right is FMM. The evaluators all gave the same ranking of HSL, Exemplar, Exemplar_13, FMM (best to worst). The full evaluation set contained 15 such sets. See Figure 10 for the original image before the motorcycle was removed.

Set	Rankings	Set	Rankings
1	[Exemplar, Exemplar_13, HSL, FMM] * 5	10	[Exemplar, Exemplar_13, HSL, FMM] * 2 [HSL, Exemplar_13, Exemplar, FMM] [Exemplar_13, HSL, Exemplar, FMM] [HSL, Exemplar, Exemplar_13, FMM]
2	[Exemplar, Exemplar_13, HSL, FMM] * 2 [Exemplar, HSL, Exemplar_13, FMM] [HSL, Exemplar_13, Exemplar, FMM] [Exemplar_13, Exemplar, HSL, FMM]	11	[HSL, Exemplar_13, Exemplar, FMM] * 3 [Exemplar_13, HSL, Exemplar, FMM] [Exemplar_13, Exemplar, HSL, FMM]
3	[Exemplar_13, HSL, Exemplar, FMM] * 3 [HSL, Exemplar_13, Exemplar, FMM] * 2	12	[HSL, Exemplar, Exemplar_13, FMM] * 2 [HSL, Exemplar_13, Exemplar, FMM] [HSL, Exemplar, FMM, Exemplar_13] [Exemplar, HSL, Exemplar_13, FMM]
4	[HSL, Exemplar, Exemplar_13, FMM] * 2 [Exemplar, HSL, Exemplar_13, FMM] * 2 [Exemplar_13, HSL, Exemplar, FMM]		
5	[Exemplar, Exemplar_13, HSL, FMM] * 4 [FMM, Exemplar, Exemplar_13, HSL]	13	[HSL, Exemplar, Exemplar_13, FMM] * 5
6	[Exemplar_13, HSL, Exemplar, FMM] * 3 [Exemplar_13, Exemplar, HSL, FMM] [HSL, Exemplar, Exemplar_13, FMM]	14	[Exemplar_13, HSL, Exemplar, FMM] * 5
7	[Exemplar_13, Exemplar, HSL, FMM] * 5	15	[Exemplar_13, HSL, Exemplar, FMM] * 4 [HSL, Exemplar_13, Exemplar, FMM]
8	[Exemplar_13, Exemplar, HSL, FMM] * 4 [Exemplar_13, HSL, Exemplar, FMM]		
9	[Exemplar, Exemplar_13, HSL, FMM] * 3 [Exemplar, HSL, Exemplar_13, FMM] [Exemplar, FMM, HSL, Exemplar_13]		

Table 1: A summary of the rankings, from best to worst, given by the evaluators for each set of images. The number after the asterisk denotes the number of evaluators that reported that particular ranking.

Method	Average Ranking
Exemplar_13	1.87
Exemplar	2.08
HSL	2.13
FMM	3.92

Table 2: The average ranking for each of the evaluated methods.

4.3 Quantitative Evaluation

Due to the inherently subjective nature of the quality of inpainting results and object deletion in particular, there is usually not a lot of available quantitative evaluation that can be done when assessing different inpainting techniques. Many previous inpainting papers rely mostly on qualitative comparison of inpainting results in order to assess the improvements made with newer inpainting techniques. While flawed, we do want to introduce one quantitative form of evaluation of our various inpainting techniques to get some form of quantitative comparison. The underlying principle is to create a random mask on an image and inpaint that mask region. Then, compare the inpainted pixel values to the original image's pixel values and determine how close the inpainting result is to the original image. The masks used for our 10 images are shown in Figure 12.

While this method can produce some indication as to the quality of an inpainting method by assessing its ability to reproduce the original image, there are a number of flaws we must first point out. For one, if the random mask happens to cover an object in the image, there is significantly less chance that any inpainting method would be able to accurately reproduce that object. Thus, this method works better when the random mask covers some background or uniform texture. In addition, the average "closeness" of pixel values of the inpainted region to the original image is not always a reliable indicator of the quality of inpainting. In particular, diffusion-based inpainting methods tend to be over-exaggerated in their effectiveness when assessed using this quantitative approach since they use surrounding pixel values to inpaint the mask region. In other words, the overall closeness of pixel values does not always correspond to realistic recreation of textures and patterns within an image region.

Figure 12: For quantitative evaluation, a random circular mask of radius 25 pixels was created for each of our 10 images. We then compare the pixel values of the original image and the pixel values generated by each inpainting algorithm.

The results shown in Figure 13 and Figure 14 show somewhat suprising results. For 9 out of 10 of

the images, the Fast Marching Method produced an inpainted region with both the lowest mean pixel difference and the lowest total pixel difference between the inpainted region and the original image. On the other hand, HSL had the highest mean and total pixel differences for all ten images. Again, this could be attributed to the fact that diffusion-based methods use their surrounding pixel values to inpaint the mask region. In adddition, since HSL-exemplar uses HSL values instead of RGB values to find patches, it is possible that the actual RGB pixel values do not match up quite as well as for the other algorithms. In addition, the mask regions as seen in Figure 12 mostly cover background/most homogeneous areas of the image which FMM excels at.

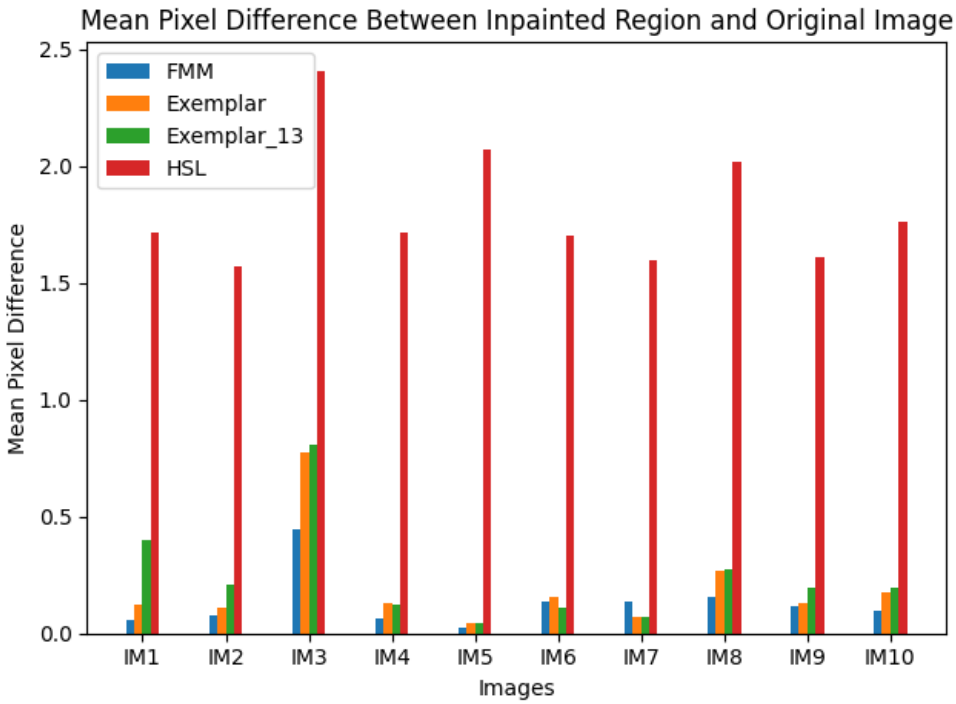


Figure 13: For quantitative evaluation, a random circular mask of radius 25 pixels was created for each of our 10 images. We then compare the pixel values of the original image and the pixel values generated by each inpainting algorithm.

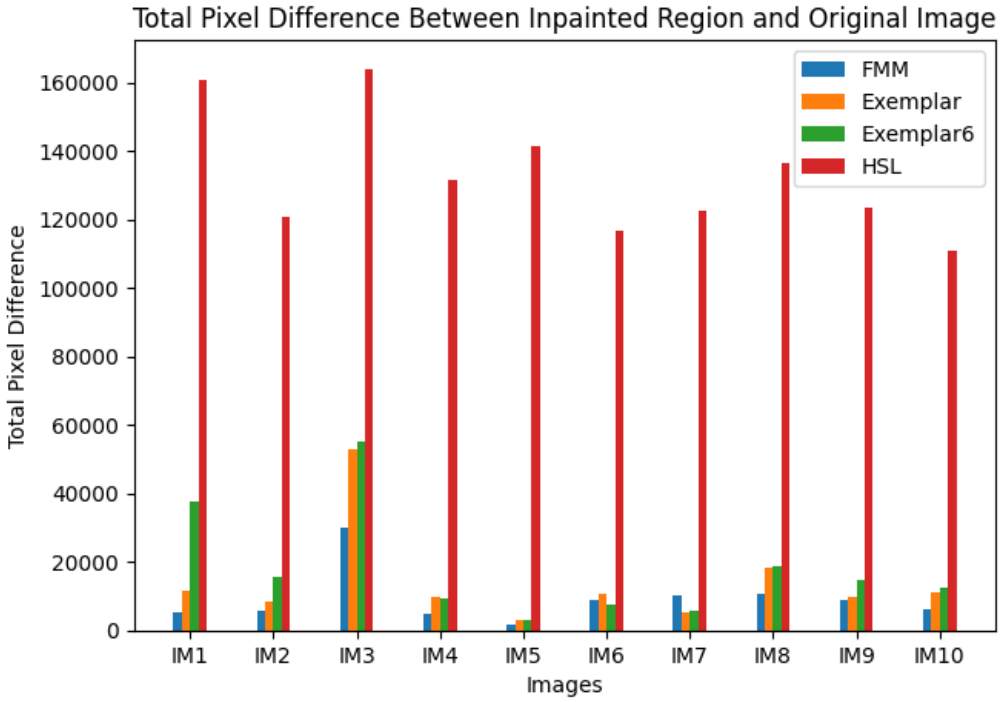


Figure 14: For quantitative evaluation, a random circular mask of radius 25 pixels was created for each of our 10 images. We then compare the pixel values of the original image and the pixel values generated by each inpainting algorithm.

Nevertheless, it is important to point out that just because FMM excels quantitatively does not equate to good performance in the subjective realm of inpainting. For example, in Figure 15, even though the quantitative results for IM8 show that FMM produced the best results, a visual analysis of the inpainting results show that FMM produced a blurry inpainted region while every other method, including HSL, produced a more textured, grassy area more accurately resembling the original image. While the quantitative results would lead one to believe that HSL performed poorly, the results in Figure 15 instead show HSL performing on-par or better than every other method.

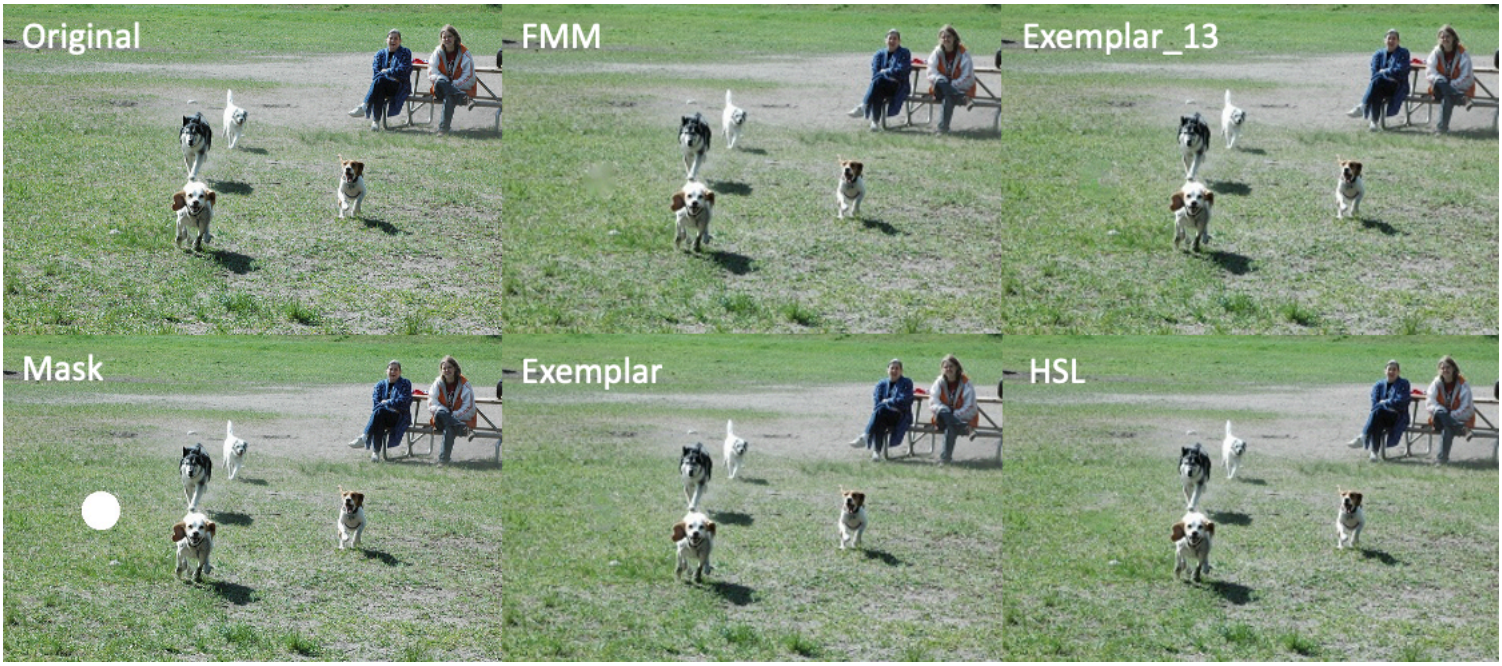


Figure 15: Image results for quantitative evaluation of IM8. The original image, image with mask region covered in white, and inpainted results for FMM, exemplar, exemplar13, and HSL are shown.

On the other hand, it is true that in some cases, FMM does indeed outperform every other algorithm as the quantitative data suggests. In the results for IM2, for example, FMM produces the best quantitative results and also produces an inpainted region that best matches the original image. All other methods show clear square-like artifacts from the patching, exemplar-based algorithms that they rely on.

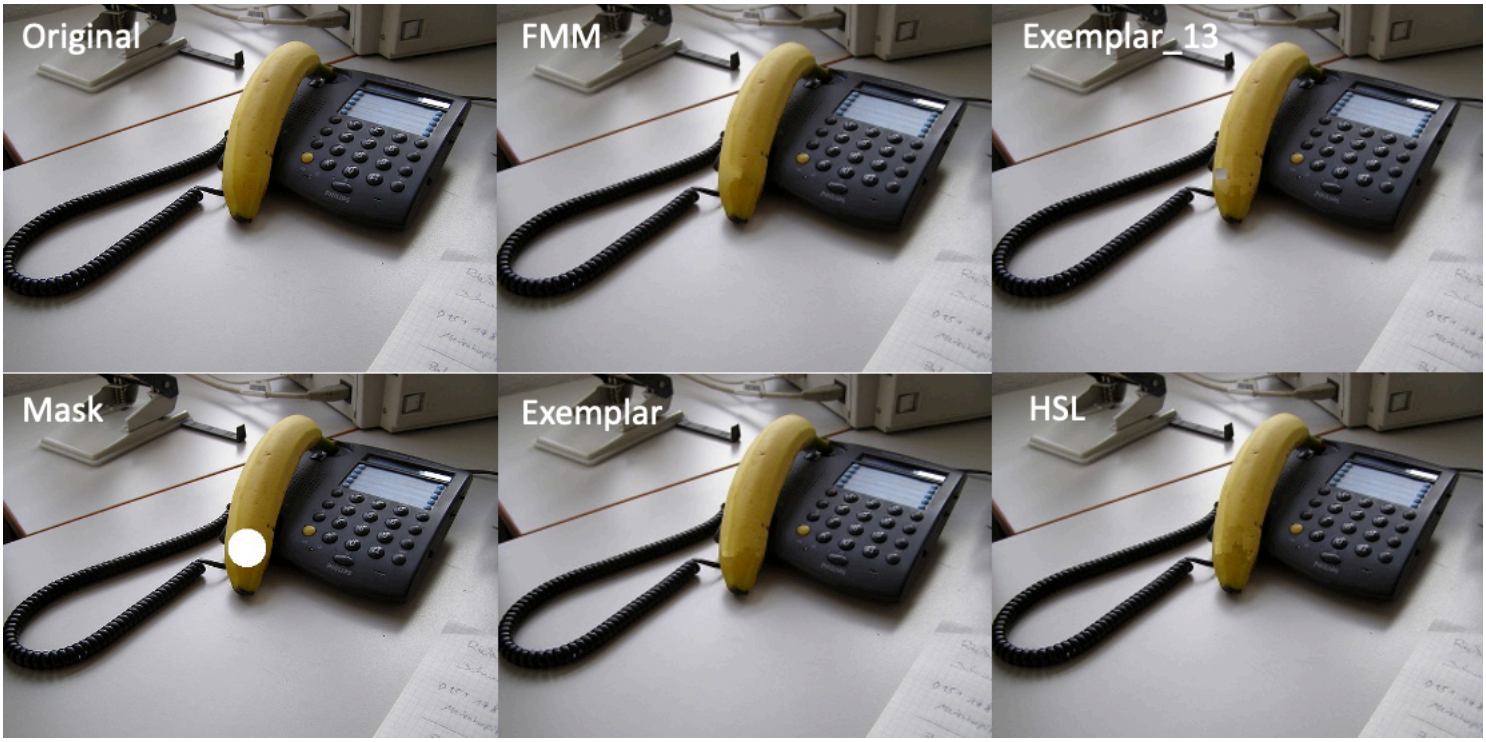


Figure 16: Image results for quantitative evaluation of IM2. The original image, image with mask region covered in white, and inpainted results for FMM, exemplar, exemplar13, and HSL are shown.

4.4 Discussion

The results from our human and quantitative evaluation emphasize the difficulty of comparing inpainting algorithms. Oftentimes, the quality of the method is dependent on the mask and its surroundings.

Effect of Patch Size. Generally, Exemplar and Exemplar_13 had similar results, both in terms of metrics from Section 4.3 and evaluator rankings.

A bigger patch size seemed to work best when the mask covered parts of the image with repeated structures. In such cases, the bigger patches took a larger surrounding area into account such that the patches were aware of the presence of these repeating structures. Figure 17 shows examples of the bigger patch size improving the inpainting results. The benefits of considering a larger patch are especially clear in the upper two images. Unlike the inpainted image on the right, the image on the left is able to recreate the grid structure.

Figure 17: Example of outputs from Exemplar-Based Inpainting using a patch size of 7 (right) and 13 (left). Exemplar_13 was ranked higher than Exemplar by all the evaluators (see Set 7 (bottom) and 15 (top) in Table 1).

A smaller patch size yielded better results in other cases where the bigger patch sizes led to more noticeable square artifacts in the output image. Figure 18 shows examples of the smaller patch size leading to better inpainting results. The smaller patch sizes are better able to reconstruct the texture of the grass in the upper images and the texture of the door and wall in the bottom images.

Figure 18: Example of outputs from Exemplar-Based Inpainting using a patch size of 7 (right) and 13 (left). Exemplar was ranked higher than Exemplar_13 by all the evaluators (see Set 1 (top) and 9 (bottom) in Table 1).

RGB vs HSL Exemplar-Based Inpainting. Overall, the outputs from Exemplar-Based Inpainting using the RGB and HSL color spaces were similar in terms of evaluator rankings. The metrics from Section 4.3 show that using the HSL color space led to higher mean and total pixel differences. However, it is important to note that this metric is imperfect and was based on distance measured using the RGB color space.

Because the HSL color space explicitly takes lightness into consideration, HSL Exemplar-Based Inpainting did better where Exemplar-Based Inpainting chose patches with lightness that did not match its surrounding lightness. Figure 18 shows an example of the benefits of using the HSL color

space. Focusing on the region within the red box, we can see that the clouds in the image on the right match the lightness of the surrounding clouds better than in the image on the left.

Figure 19: Example of outputs from Exemplar-Based Inpainting based on the RGB (left) and HSL (right) color spaces. The red rectangle roughly outlines the mask used.

Strengths. The main strength of our HSL Exemplar-Based Inpainting algorithm is that it does not require the computational and time resources that deep learning methods do. Nonetheless, for the wide range of images and masks evaluated, the method works well. Overall, HSL Exemplar-Based Inpainting produces outputs just as good as Exemplar-Based Inpainting with the added benefit of considering lightness.

Weaknesses. Because our method is a modification of Exemplar-Based Inpainting, it also borrows its weaknesses. The algorithm's "knowledge" is constrained to the known patches in the images and therefore the method cannot infer new pixels or features. This method also looks at patches independently such that repetitive structures are harder to inpaint well. This can be mitigated by using larger patch sizes, but larger patches also lead to more marked square artifacts.

5 Conclusion and Future Work

In this project, we successfully implemented and evaluated three separate inpainting methods: Fast Marching Diffusion-Based Inpainting, Exemplar-Based Inpainting, and HSL Exemplar-Based Inpainting. Although the Fast Marching Method performed well in purely quantitative results, the visual results of FMM inpainting fell short of all the other inpainting methods. HSL Exemplar-Based Inpainting was a novel inpainting method that we introduced in this project. While we hoped that using the HSL color space would lead to better visual results in inpainting, the overall conclusion we reached was HSL mostly worked on-par with normal exemplar-based inpainting and was, at best, incremental in its improvement.

In the future, it may be worth looking into other ways to incorporate other spaces other than RGB into inpainting algorithms such as LMS or XYZ color spaces to see if they would produce markedly different results. In addition, incorporating a way to dynamically allow an exemplar-based inpainting algorithm to choose its patch size could help lead to better results as we noticed that changing the patch size led to marked improvements on some images but not others. Finally, while we did not make deep-learning-based inpainting methods a focus of this project, there are a lot of possible areas of further research in with incorporating deep learning into inpainting algorithms as well.

6 Additional Information

6.1 Links

The link to the GitHub repository with pre-trained Mask R-CNN model used to create masks: https://github.com/matterport/Mask_RCNN

The link to our Google Drive folder with all the images used in our analysis: https://drive.google.com/drive/folders/1gzOkHt1Ehr9XNNjXxjuZ1sjwBRc9YwAP

The sets under 'Labeled Sets' were labeled by the inpainting method used for convenience. The sets under 'Randomized Sets' were the sets of images sent to the evaluators. Output from our quantitative analysis and the original 10 images from the COCO data set were also included.

6.2 Code

inpaint.py: This is the main file used to run the implemented inpainting pipeline. There are 5 arguments: the mask path, the image path, the epsilon value, the method, and the output image path. The mask path specifies the file path to the mask with pixel values of 255 for regions to be inpainted and pixel values of 0 elsewhere. The image path specifies the image to overlay the mask on. The image and the mask must have the same dimensions. The epsilon value specifies the half-width of the patch size. The three options for the method are 'FMM', 'EXEMPLAR', and 'HSL'. The output path specifies the file path to write the inpainted image to.

Depending on the method, inpaint.py calls functions from inpaint_fmm.py, inpaint_exemplar.py, or inpaint_hsl.py. Each of these functions implement the Fast Marching Method, Exemplar-Based Inpainting, and HSL Exemplar-Based Inpainting. All the inpainting code was implemented by us. No external libraries besides cv2 (for reading and writing images and calculating image gradients), numpy, copy (for making deep copies), and heapq were used.

References

(1) Yang, C.; Lu, X.; Lin, Z.; Shechtman, E.; Wang, O.; Li, H. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2017, pp 6721–6729.

(2) He, K.; Gkioxari, G.; Dollar, P.; Girshick, R. In *Proceedings of the IEEE International Conference on Computer Vision (ICCV)*, 2017.

(3) Telea, A. *Journal of graphics tools* **2004**, *9*, 23–34.

(4) Huang, J.-B.; Kang, S. B.; Ahuja, N.; Kopf, J. *ACM Transactions on graphics (TOG)* **2014**, *33*, 1–10.

(5) Bertalmio, M.; Vese, L.; Sapiro, G.; Osher, S. *IEEE transactions on image processing* **2003**, *12*, 882–889.

(6) Criminisi, A.; Pérez, P.; Toyama, K. *IEEE Transactions on image processing* **2004**, *13*, 1200–1212.

(7) Barnes, C.; Shechtman, E.; Finkelstein, A.; Goldman, D. B. *ACM Trans. Graph.* **2009**, *28*, 24.

(8) Zeng, Y.; Lin, Z.; Yang, J.; Zhang, J.; Shechtman, E.; Lu, H. *arXiv preprint arXiv:2005.11742* **2020**.

(9) Pathak, D.; Krahenbuhl, P.; Donahue, J.; Darrell, T.; Efros, A. A. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2016, pp 2536–2544.

(10) Iizuka, S.; Simo-Serra, E.; Ishikawa, H. *ACM Transactions on Graphics (ToG)* **2017**, *36*, 1–14.

(11) He, K.; Gkioxari, G.; Dollár, P.; Girshick, R. In *Proceedings of the IEEE international conference on computer vision*, 2017, pp 2961–2969.