CSC420 Project Report

Weimin Huang (1003560522)

Yanna Ding (1003113034)

Peizhi Zhang (1002944987)

ABSTRACT

Image inpainting is the process of filling in damaged or missing parts in an image. In this project, we implemented and compared three approaches to inpaint images: (i) deep learning model, (ii) exemplar-based method using a well-defined patch filling order, and (iii) statistical patch offsets method. We then compared the strength and weakness of the 3 methods. Overall, the deep learning model is the most promising one. At the same time, the exemplar-based methods and statistics patch offset method also have their advantages, such as being more efficient and maintaining the resolution.

1. INTRODUCTION AND LITERATURE REVIEW

Image is an important source of information. However, in practice, images are often corroded by noise such as scratches from old photos, droplets on the camera lens, and time stamps added on pictures. Image inpainting is the art of reconstructing the deteriorated or missing parts in an image or removing the undesired parts added to it. Given an image with a target region that is missing in it, the goal of our project is to fill the pixels of the region with information that could have been in the image in a visually plausible way based on the remaining parts of the image. Image inpainting has various applications including image restoration and photo editing, and these tasks can be challenging due to the complexity of patterns in images. In this project, our main objective is to make imperceptible modifications in an image that appear as realistic and natural to human eyes.

Abundant researches have focused on digital 2D image inpainting, aiming to recover the structure and texture of an image based on known information on the figure. Bertalmio et al.[1] proposed an algorithm that inpaints a given region by prolonging the isophote lines arriving at the contour of the region. They iteratively computed the projection of the smoothness, given by the discretization of the 2D Laplacian, onto the direction of isophote lines. Researchers have also developed exemplar-based synthesis methods for image inpainting. This approach distinguishes a target region to be filled and the source region to gain texture information of the image. The inpainting problem is divided into filling in small patches of equal size. Given a patch in the target region, the algorithm searches for the most similar patch lying entirely in the source region and copies the patch to the target patch. Based on this idea, Criminisi et al.[2] further assigned priorities to patches and confidence levels to pixels. The priority determines the order of patch filling, where the patches lying on some strong edges and surrounded by pixels with high confidence level are visited first.

Recently, researchers have also investigated deep neural networks for image inpainting. Images with target regions missing are passed through deep learning models and trained in an end-to-end fashion to predict the missing regions. Typically, convolutional neural networks (CNN) are used as image context encoders. Yan et al. proposed a CNN that contains a decoder, an encoder, and shift operation that connects that encoder feature of the known region and the decoder feature of missing parts.[3]

2. METHODOLOGY, RESULTS, AND EXPERIMENTS

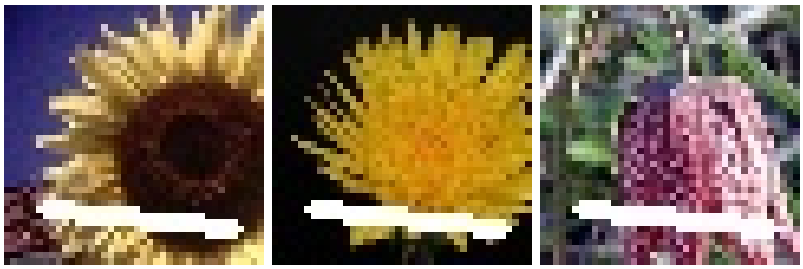
2.1 Deep Learning

We used deep neural networks for image inpainting. We first tried a simple encoder-decoder model, and then improved on it.

2.1.1 Dataset

The dataset we used is the 17 Category Flower Dataset.[?] We cropped and resized the images in the original dataset to 40×40 due to hardware constraints in our systems. We then did a train-validation-test splitting with ratio of $Train : Validation : Test = 6 : 2 : 2$. After the splitting, the number of images for training is 817. Training data consist of images with missing regions, and the target labels consist of corresponding original images. To generate images with missing regions,we drew random lines on the images. We tried two sets of lines. The first set contains a fixed random lines for all images (i.e. all samples have the same random line), with line thickness 3. The second set contains true random lines, with line thickness 5.

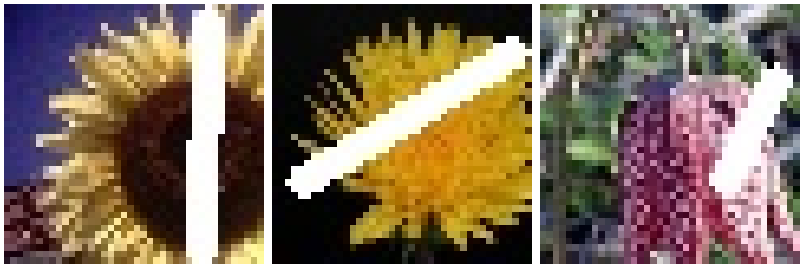
Figures 1 shows images generated by drawing fixed random lines on the images.



(a) image_0143.jpg (b) image_0490.jpg (c) image_0851.jpg

Figure 1: Set 1: Fixed random masks

Figures 2 shows images with irregular masks generated by drawing true random lines on the images.



(a) image_0143.jpg (b) image_0490.jpg (c) image_0851.jpg

Figure 2: Set 2: True random masks

2.1.2 Model 1: Simple Encoder-Decoder

Hiton et al.[4] proposed that deep autoencoder network can be used to reduce the dimensionality of data and then reconstruct the high-dimensional data matrix, resulting in better performance than principal components analysis. This is the basis of encoder-decoder models. Missing regions such as scratches in images can be viewed as noise on image. Then we can pass the images to an encoder-decoder model to recover the noise, leading to a complete image with missing regions filled. The design of our encoder-decoder model comes from a Blog where a model is use to denoise a grayscale image.[5] We used a similar design, but we constructed the network so that it takes in and outputs 3-channel RGB images.

Model 1 is a simple encoder-decoder network that we constructed. It consists of 4 encoder blocks and 4 decoder blocks. Each encoder block contains a convolutional operation and a max-pool operation to reduce the image size and add in non-linearity.

The loss function we used is mean squared error (MSE), which can be expressed as $\frac{1}{N}\sum(y_{pred} - y_{ground})^2$. It captures the difference between output and groundtruth on the pixel level. y is the normalized pixel value expressed as $y_{ground} = \frac{intensity}{255}$, so that the values are all between 0 and 1. N is the total number of pixels.

After convolution and deconvolution, a sigmoid activation is applied to convert the values to a range between 0 and 1. In testing phase where we get the predicted output, we multiply the output feature map by multiplying 255. $intensity = y_{pred} \times 255$.

Figure 3 shows the architecture of the model. There are 4 encoder layers and 4 decoder layers. After each encoder layer, a maxpooling operation is applied to reduce the dimension and add in non-linearity.

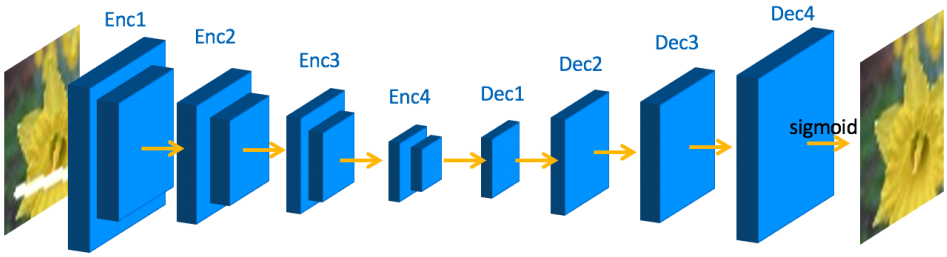


Figure 3: Simple Encoder-Decoder

The following table shows the information for each layer.

Layer	Size	Maxpool	Activation
Enc1	(64,40,40)	(2,2)	relu
Enc2	(32,20,42)	(2,2)	relu
Enc3	(16,10,10)	(2,2)	relu
Enc4	(8,5,5)	(2,2)	relu
Dec1	(8,5,5)	N/A	relu
Dec2	(16,10,10)	N/A	relu
Dec3	(32,20,42)	N/A	relu
Dec4	(64,40,40)	N/A	relu
Out	(3,40,40)	N/A	sigmoid

Figure 4 shows the result of an image for testing when we used model 1.

(a) Original Image (b) Masked Image (c) Output Image

Figure 4: image_0030

It can be seen that the output of our Model 1 is a blurred version of the input, which is not desirable. The reason is that the information about the original image resolution is lost after passing the image through the network. The convolution operations can have a blurring affect and do not successfully reconstruct the pixel values. To solve the problem we revisited some state-of-the-art architectures, including U-Net,[6] where a shortcut connection is concatenated after decoding and encoding. We modified our Model 1 and added some U-Net like shortcut connections, resulting in architecture 2 as in the next section.

2.1.3 Model 2: U-net like Encoder-Decoder

In model 2 we used U-net like connections to concatenat the layers that have the same size after encoding and decoding, and the architecture is shown in Figure 5.

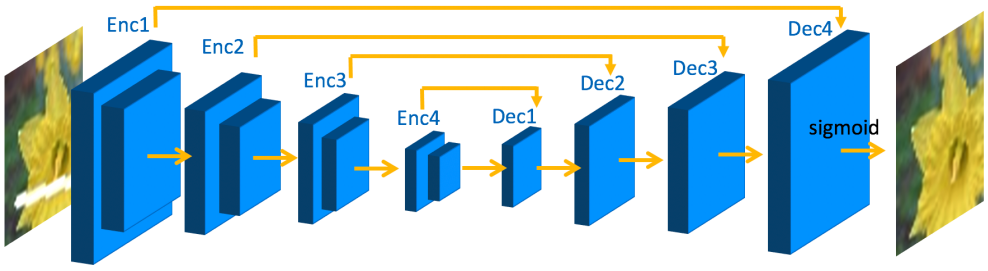


Figure 5: U-net like Encoder-Decoder

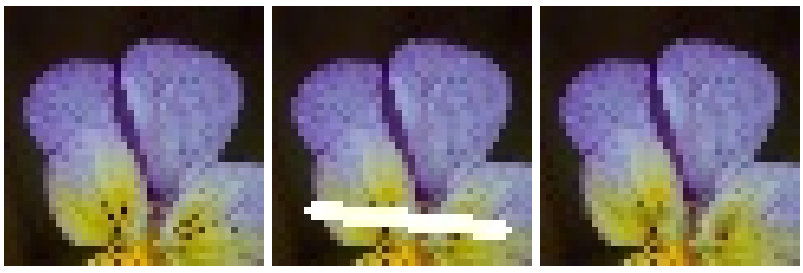
Figure 6-8 are the results of Model 2 tested on images with fixed random masks. It can be seen that the model successfully inpaints the images.

(a) Original Image (b) Masked Image (c) Output Image

Figure 6: Fixed random masks - image_0030.jpg

(a) Original Image (b) Masked Image (c) Output Image

Figure 7: Fixed random masks - image_0196.jpg



(a) Original Image (b) Masked Image (c) Output Image

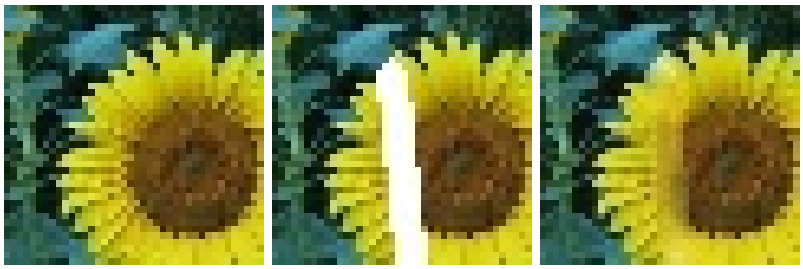
Figure 8: Fixed random masks - image_0344.jpg

2.1.4 Limitation

We also tested Model 2 on images with true random masks and bigger mask size. The results are shown in Figures 9-12. It can been seen that also it inpaints the general contour of the image, however, the missing regions filled are slightly blurred and not very coherent with the rest of the images.

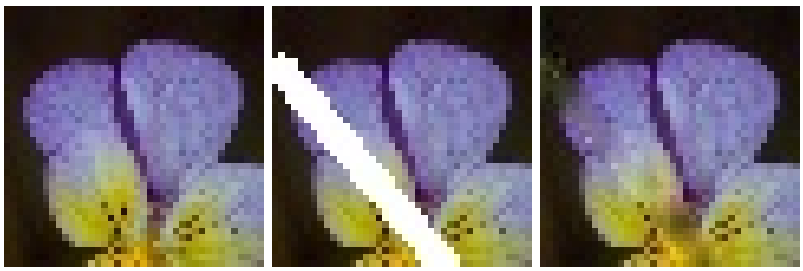
(a) Original Image (b) Masked Image (c) Output Image

Figure 9: True random masks - image_0030.jpg



(a) Original Image (b) Masked Image (c) Output Image

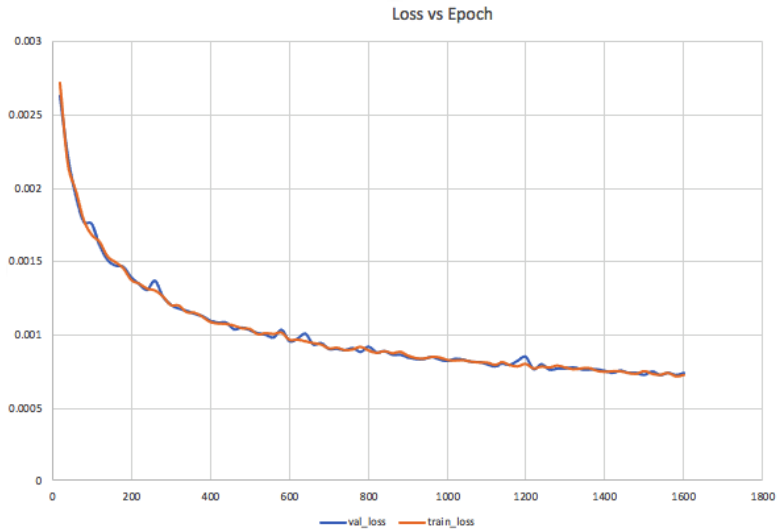
Figure 10: True random masks - image_0196.jpg



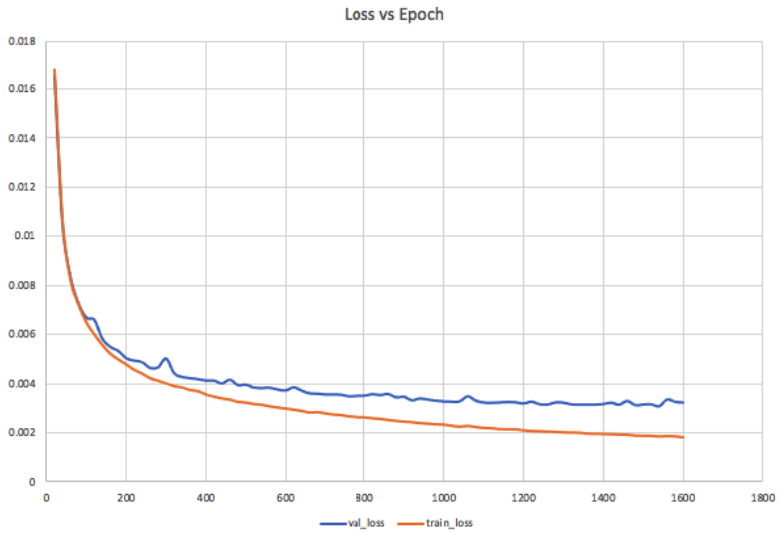
(a) Original Image (b) Masked Image (c) Output Image

Figure 11: True random masks - image_0344.jpg

We looked at the loss while training for images with fixed and true random mask respectively. It can be seen that when fixed mask results in a lower mean squared loss finally. Also, overfitting in fixed mask images are minimal. True random mask data is more proned to overfitting. It is hard to encode and decode image features when the masks are at varying locations. The model for images with true random mask is harder to train and optimize, resulting in images that are not perfectly inpainted.



(a) Fixed random mask



(b) True random mask

Figure 12: Loss vs Epoch

2.2 Exemplar Based Method

2.2.1 Algorithm

We implemented an exemplar-based method introduced by Criminisi et al.[2] Given an image I with a masked region $\Omega \subset I$, the algorithm fills in the pixels in Ω in some well-defined order. The overall task is to complement the missing region using similar pixels in the source region (*i.e.*, $I - \Omega$). By copying pixel values directly to the target area, we synthesize the texture of the original image; and by filling in the target region in an edge-driven order, we synthesize the linear structures in the original image. The input of the algorithm consists of:

1. An image I with size $m \times n$.
2. A mask M with size $m \times n$. The mask is an image with black background and white pixels specifying the corresponding target region in I.

3. A square patch's side length k, where k is odd. During the algorithm, we copy and paste patches Ψ_p centering at some pixel $p \in I$.

The effective information available for guessing the target pixel values lies in the source region. The patches that are filled earlier become part of the source region and determine the pixel values filled later. Therefore the filling order is crucial. To achieve a desirable order, we gives high priority to patches that: (i) contain more effective information and (ii) are on some linear structures in I.

To account for (i), we keep track of a confidence value $C(p) \in [0, 1]$ for each pixel $p \in I$. $C(p)$ has a positive relation with the number of known pixels surrounding p. Patches with relatively more pixels in $I - \Omega$ contains more effective information than others. Favouring patches with high $C(p)$ can better leverage the provided pixel values in $I - \Omega$.

To measure (ii), we store and update a data term $D(p) \in [0, 1]$ for $p \in I$. Specifically,

$$D(p) = \frac{|\nabla I^\top \cdot n_p|}{\alpha} \qquad (*)$$

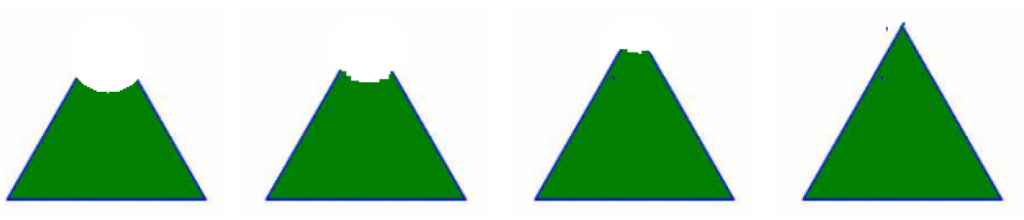
$\nabla I^\top = [I_y(p), -I_x(p)]$ denotes the isophote at p, which is orthogonal to the image gradient $[I_x(p), I_y(p)]$. $n_p = \left[\frac{M_y(p)}{\sqrt{M_x(p)^2+M_y(p)^2}}, -\frac{M_x(p)}{\sqrt{M_x(p)^2+M_y(p)^2}}\right]$ denotes the normal to the contour $\delta\Omega$ of Ω at p. α is a normalization term, which is equal to 255 when I a gray-scale image.$D(p)$ is maximized when p is at the continuation of some linear structures (*e.g.* edges).

The algorithm works as follows:

1. Initialize $C(p)$ as $C(p) = 1$ for $p \in I - \Omega$ and $C(p) = 0$ for $p \in \Omega$.
2. Identify the fill front $\delta\Omega$. We applied the filter $[-1, 1]$ on the mask to locate the vertical edges of the target and filter $[-1, 1]^\top$ for locating horizontal edges.
3. Compute the priority term $P(p) = C(p)D(p)$ for $p \in \delta\Omega$, where $C(p) = \frac{\sum_{q \in \Psi_p \cap I-\Omega} C(q)}{|\Psi_p|}$ and $D(p)$ is as defined in $(*)$. Note that $C(p)$ is average over the confidence levels of surrounding pixels in the source region.
4. Pick the patch $\Psi_{\hat{p}}$ centering at the pixel with the highest priority. Theoretically, $\Psi_{\hat{p}}$ has the largest number of known pixels and a linear structure passing nearly orthogonal to the contour $\delta\Omega$ passes through $\Psi_{\hat{p}}$.
5. Find the patch $\Psi_{\hat{q}} \subset I - \Omega$ that minimizes $d(\Psi_q, \Psi_{\hat{p}})$, where d is the sum of the squared differences between the already-filled pixel values in Ψ_q and $\Psi_{\hat{p}}$. Copy the values from $\Psi_{\hat{q}}$ to $\Psi_{\hat{p}}$. Update the confidence value for $p \in \Psi_{\hat{p}} \cap \Omega$ as $C(p) = C(\hat{p})$.
6. Repeat step 2 to 5 until all pixels in the target region are filled.

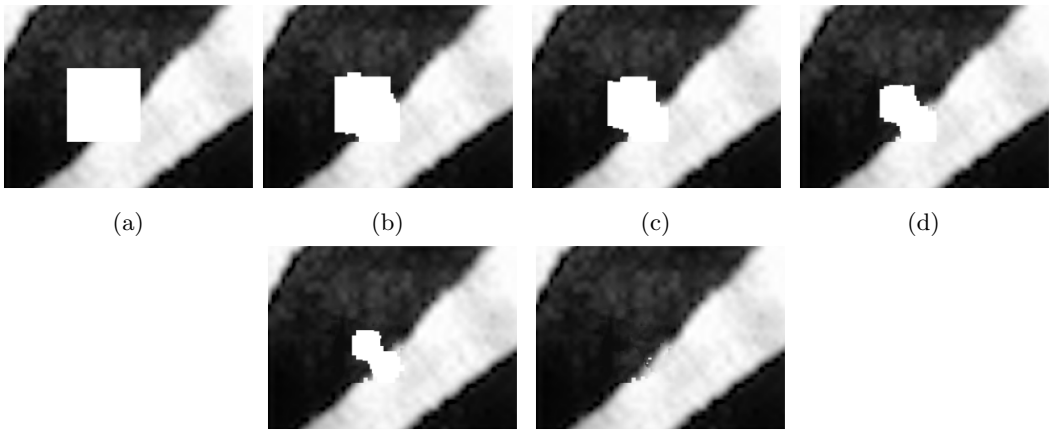
2.2.2 Results

The following examples demonstrate the filling process of this algorithm. Note that the data term forces early recovery of the region near edges, which yields natural continuation of edges. The scripts to generate these results are included in `./exemplar-based/README.txt`.



(a) (b) (c) (d)

Figure 13: Inpaint image "triangle.png" with patch size $k = 5$. Figure (a) shows the input image, where the top of the triangle is masked by a circle. Figures (b)-(c) are captured in the middle of the filling process. Figure (d) is the final result.



(a) (b) (c) (d)

(e) (f)

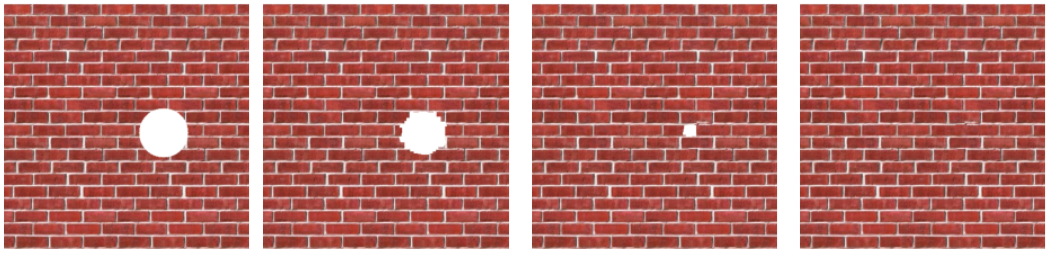
Figure 14: Inpaint image "stripes.png" with patch size $k = 7$. Figure (a) shows the input image, where the white square specifies the target region. Figures (b)-(e) show patches near the edge are filled first. Figure (f) is the final output.

(a) (b)

Figure 15: Inpaint image "flower1.jpg" with patch size $k = 3$. Figure (a) shows the input image, and figure (b) is the output.

(a) (b)

Figure 16: Inpaint image "flower2.jpg" with patch size $k = 3$. Figure (a) shows the input image, and figure (b) is the output.



(a) (b) (c) (d)

Figure 17: Inpaint image "brick.png" with patch size $k = 9$. Figure (a) shows the input image, and figure (d) is the output.

2.2.3 Limitations

1. The algorithm requires careful tuning of k. It is not intuitive to choose the best k. For example, if we set k to 9 when recovering the triangle, we would get the following result, which has an extra part at the top of the triangle. So an inappropriate k value could lead to copying undesired pixel values.

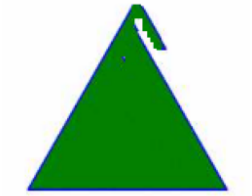


Figure 18: patch size $k = 9$

2. This exemplar-based method draws upon existing pixel values. If a to-be-inpainted region is supposed to contain pixels not in the source area, then this algorithm is not going to generate a satisfying result.

3. The method is incapable of recovering nonlinear structures well.

2.3 Statistical Patch Offsets

We also tried a third method, introduced in the paper "statistical patch offsets for image completion."[7]

2.3.1 Intuition

We observed that often we can use similar patches around the hole to fill the hole. Where are the patches then? What are their directions and distances? So the key is to calculate patch offsets to find similar patches around the hole. We can calculate in a statistical way: we split the image into small patches, and try to find the most similar match for each patch and we calculate the offsets for each pair of these matches. Then we calculate the statistics and try to find the dominant offsets. And then we can use the dominant offsets to find the similar patches to fill the hole.

2.3.2 Algorithm

I implemented statistics of patch offsets for image completion by Kaiming He and Jian Sun

Step 1: Matching Similar Patches in the known region and obtain their offsets.[8]

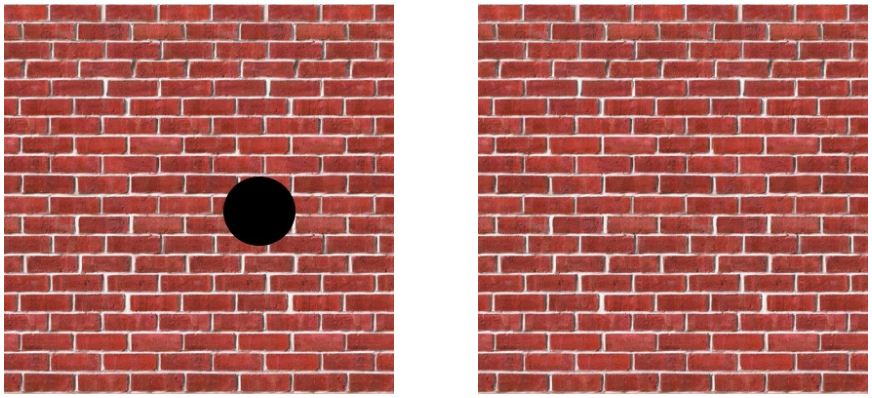
for each patch P in the known region, we compute its offset s to its most similar patch, which is measured by sum of squared differences between two patches.

Step 2: Given all the offsets s(x), we compute their statistics by a 2-d histogram h(u, v)

The core of this algorithm is to use patch offsets and their statistics to model image characteristics and patterns.

Step 3: combining shifted images via optimization For optimization, I used a graphcut algorithm as discussed in the paper.[9][10][11]

2.3.3 Results



(a) (b)

Figure 19

For simple images such as these, the result is almost perfect.

For complex images, the result is far from perfect but acceptable. The reason probably is that the offsets are irregular, making the image patterns implied by offset statistics more difficult to find

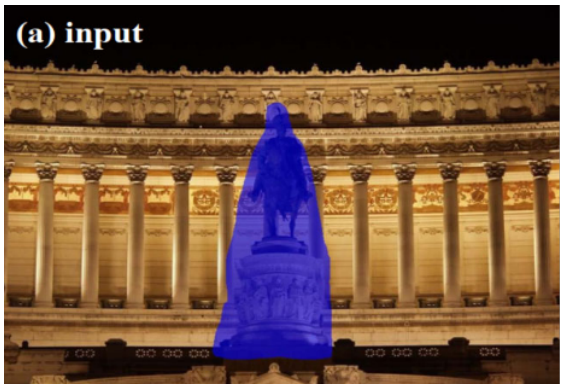
We have to fine tune parameters to make those complicated images have better performance. For example,

(a)

(b)

Figure 20

The above left image was inpainted at K = 60. When we let k = 90, the above right image results and it's much better compared to the left image.



(a)

(b)

Figure 21

Another set of those failed images is presented here. Here K = 90, but the failure persists.

2.3.4 Improvements

What if we used a different method to describe a patch? The original algorithm used intensity values to find the most similar patch. I used SIFT matching to generate the keypoints and the descriptors to find the simialr patches. The resulting algorithm has a much better performance, leading to improved results.

For running my code, please refer to statisticalPatchOffsets/README.txt

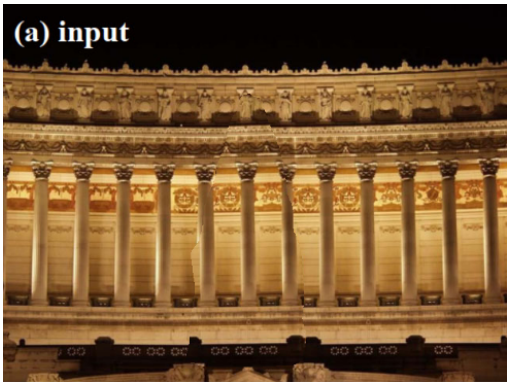


Figure 22

2.4 Comparison

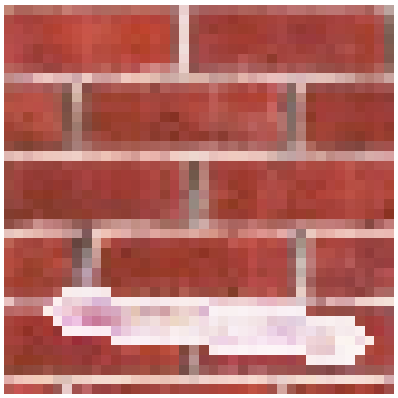
After working on these 3 approaches, we compared the pros and cons of them. The deep inpainting approach seems to be the most promising algorithm (Figures 6-8), as it considers the global semantics of an image. The filled regions looks visually and semantically coherent for images with fixed random masks. The two examplar-based methods work reasonably well for images that have simple patterns, such as the stripes image (Figure 14) and the brick image (Figure 17). However, when it comes to more complex images such as flowers, the examplar-based methods tend not to produce results that are very coherent. (Figure 15) In general, the exemplar-based method is less flexible than a well-trained neural network model for the following reasons:

1. Exemplar-based method is sensitive to the shape of the mask, as the method relies on the image gradients to synthesize the existing image structures. If the target region is not placed appropriately, one could expect weird results.

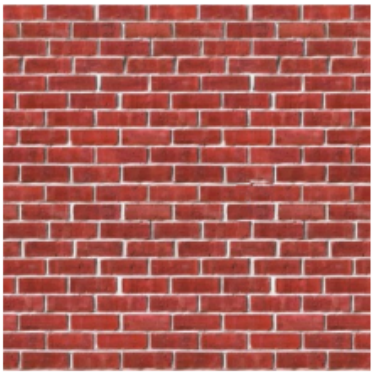
2. The output of exemplar-based methods also depends on the patch size being used, as demonstrated in section 3.3.3.

3. A well-trained neural network learns the features of a category of images and does not rely heavily on the input image itself.

However, there also are some drawbacks of the deep inpainting approach. Model trained on one dataset is not applicable for another dataset without retraining or fine-tuning. The following figure shows the output of passing a brick image to a model only trained on the flowers dataset. While the exemplar-based method successfully recovers the structures and textures, for the brick image, the neural network model fails to capture the pattern of the original image. The reason is that the brick image has very different characteristics from the images in the flower dataset. Neural networks learn the pattern in a particular set and require that training and testing samples have similar statistics.



(a) neural network result



(b) exemplar-based result

To make the neural network work on other images, Re-training or fine-tuning on a corresponding dataset is required for applying to different datasets. However, due to the nature of machine learning, the deep inpainting approach is very time consuming. It will take even more time if we train it on images with higher resolution. Also, it is hard to find a large dataset for all images.

Also, when we apply our deep inpainting model to image data with true random masks (without a fixed mask location and with a varying mask size), the results look blurred. It seems that the decoder do not successfully re-construct the information after several convolution layers. Potential solution would be to investigate in more complex models or to try other training strategies for this model (such as data augmentation). In comparison, in the exemplar-based method and in the statistics patch offsets method, we are using the exact original pixel values to fill in the target region, therefore, we don't face the problem of getting blurred image regions.

3. CONCLUSION

We implemented three approaches for image inpainting. Overall, we have found approaches that are successful in filled in the missing region of images. The examplar-based method and the statistics offset method perform well in processing simple images. For example, the statistical patch offset algorithm had nearly perfect performance at the brick images.17 The deep learning approach, when trained on a particular set of images, can produce results that are very visually coherent in images with a more complex pattern, such as flowers. Deep learning models are good at capturing the global semantics of an image and reproduce the contours of complex structures in the original images. When it comes to complex images, examplar-based methods seem to fail and difficult to obtain unanimous results through parameter fine tuning even after we improved the statistical method. Overall, the deep learning approach seems to be the most promising method, since a well-designed neural network can not only infer on the local statistics of the images, but also is understand the global semantics of an entire image.

Unfortunately, it's very hard to train a neural network; it requires time, hardware, and large number of images. To make the deep inpainting approach work well in images with different size, different themes, and different masked regions, we will need to devote much time in adjusting the model architectures and optimization strategies.

4. AUTHORS' CONTRIBUTIONS

We each implemented an algorithm, namely. Weimin Huang implemented the deep learning algorithm and improved the first simple model using U-net like connections in the paper "U-Net: Convolutional Networks for Biomedical Image Segmentation". Yanna Ding implemented the exemplar based method based on the paper "Region Filling and Object Removal by Exemplar-Based Image Inpainting.". Peizhi Zhang implemented "Statistics of Patch Offsets for Image Completion". We worked together to compare the results of these 3 algorithms. Finally, we cooperated to write the report.

REFERENCES

[1] Bertalmio, M., Sapiro, G., Caselles, V., and Ballester, C., "Image inpainting," in [*Proceedings of the 27th Annual Conference on Computer Graphics and Interactive Techniques*], *SIGGRAPH '00*, 417–424, ACM Press/Addison-Wesley Publishing Co., USA (2000).

[2] Criminisi, A., Perez, P., and Toyama, K., "Region filling and object removal by exemplar-based image inpainting," *IEEE Transactions on Image Processing* **13**(9), 1200–1212 (2004).

[3] Yan, Z., Li, X., Li, M., Zuo, W., and Shan, S., "Shift-net: Image inpainting via deep feature rearrangement," in [*Proceedings of the European Conference on Computer Vision (ECCV)*], (September 2018).

[4] Hinton, G. E. and Salakhutdinov, R. R., "Reducing the dimensionality of data with neural networks," *Science* **313**, 504 – 507 (2006).

[5] Rath, S. R., "Autoencoder neural network: Application to image denoising." https://debuggercafe.com/autoencoder-neural-network-application-to-image-denoising/ (2020).

[6] Ronneberger, O., Fischer, P., and Brox, T., "U-net: Convolutional networks for biomedical image segmentation." https://arxiv.org/abs/1505.04597 (2020).

[7] He, K. and Sun, J., "Statistics of patch offsets for image completion," 16–29 (10 2012).

[8] He, K. and Sun, J., "Computing nearest-neighbor fields via propagation-assisted kd-trees," *2012 IEEE Conference on Computer Vision and Pattern Recognition* (2012).

[9] Boykov, Y. and Kolmogorov, V., "An experimental comparison of min-cut/max- flow algorithms for energy minimization in vision," *IEEE Transactions on Pattern Analysis and Machine Intelligence* **26**(9), 1124–1137 (2004).

[10] Boykov, Y., Veksler, O., and Zabih, R., "Fast approximate energy minimization via graph cuts," *IEEE Transactions on Pattern Analysis and Machine Intelligence* **23**(11), 1222–1239 (2001).

[11] Kolmogorov, V. and Zabih, R., "What energy functions can be minimized via graph cuts?," *IEEE Transactions on Pattern Analysis and Machine Intelligence* **26**(2), 147–159 (2004).