Exploring Digital Image Inpainting Techniques for Artwork Restoration

COMP 558 Final Report

Nick Morrison, Declan Giltz, Sagar Nandeshwar

1. Abstract

In this paper we cover three digital image inpainting algorithms: two procedural methods (Telea and Naive strokes) and one data driven (Partial Convolution). We have implemented both Telea and Partial Convolution. We compare our methods to each other and expert implementations - our Telea is compared to openCV Telea and Naive Strokes inpainting, and our Partial Convolution model to the NVIDIA design - and found that our implementations fared quite well, even despite minimal training with regards to our Partial Convolution model. We did these comparisons with a dataset of paintings from 50 famous artists, with a future goal of using our techniques to aid in the restoration process of damaged paintings.

2. Background

Inpainting is the process of restoring an image by strategically filling areas that have been damaged or obscured. This technique has been the basis of art restoration for centuries and now composes a fundamental aspect of computer vision with the prevalence of digital images. It has applications in graphics, image restoration, and photography.

Modern inpainting originates from the restoration of physical art. We find evidence of this idea as early as the Renaissance in which damaged paintings from the Middle Ages were retouched by trained artists [1]. In the 1800's, Italian art director Pietro Edwards was the first to emphasize respect for the artist's original intentions during the conservation process [2]. Then in 1930, Helmhut Ruhemann brought the idea to light at a technical art conference in Rome. His approach to physical inpainting required the artist to consider the painting as a whole when repairing small regions [3]. In this way, a unified practice of art restoration was established.

Naturally, the idea of inpainting was applied to the world of photography and film. Frames of a movie could be retouched to remove specks of dust or other noisy artifacts. Images that were cracked or ruined with age could be repaired to their former glory. As our media transitioned to being primarily online, images could be easily corrupted with overlaid text or poor scanning. Thus the need for digital inpainting arose [1].

3. Digital Inpainting Techniques

Digital inpainting techniques have evolved significantly over the past few decades, so there is great variety in motivation and implementation. While none of these techniques perfectly recreate the original, they have achieved much success in creating a visually pleasing image.

Convolution Based Techniques: The most basic method involves simply convolving the image with a filter to diffuse known information into regions of unknown as presented by Oliveira in 2001 [10]. While this method is quite fast, it has no concept of preserving contours, only works on small regions, and requires significant human intervention [4].

PDE Based Techniques: PDE based techniques solve a partial differential equation to propagate color inside the missing region. The goal is to preserve adjacent isophote directions which are curves that connect points of equal brightness. This idea was first introduced by Bertalmio and Sapiro. Their method attempts to maintain structure in the missing region by preventing crossing isophotes [1]. Telea later improved this method by using a Fast Marching Method to order the propagation of pixels [4]. Other models such as the Total Variation model and the Curvature Driven Diffusion model by Chen use slight variations that allow for inpainting of thicker regions [9]. While these methods are effective, isophote estimation and color propagation both are subject to diffusion which can lead to blurring in the inpainted region.

Exemplar Based Techniques: Criminisi was the first to define this patch-wise filling instead of a pixel wise filling. This technique is ideal for large regions because it copies and pastes patches from the source image which preserves texture [12]. Zhou and Kelly later improved this method by adding local optimization to improve consistency [13].

Deep Learning Techniques: Recently, there have been several deep learning based approaches to inpainting that make use of Big Data. One example is Context Encoder from 2016, which was the first GAN based algorithm. This technique allows for a deeper semantic understanding of the entire image using a channel-wise fully connected layer [7]. Following this breakthrough, many other approaches emerged including MSNPS which added a texture network and GLCIC which avoids the computationally expensive fully connected layer by using dilated convolutions. There are also copy-and-paste based methods such as ShiftNet and DeepFill V1. Most recently, PartialConv has achieved impressive results on images masked with multiple irregular holes and it is one of the algorithms we implemented [8].

4. Inspiration

Ferdinand Deger, a researcher at the Norwegian Color and Visual Computing Laboratory, was similarly interested in the applications of Digital Inpainting for artwork restoration. He compiled

a survey in which he naively implemented 8 popular PDE, convolution, and exemplar based approaches. He found that while people preferred manual inpainting to all these digital techniques, the exemplar methods seemed to be favored. He postulated that these techniques could be useful as a guide for restoration, a quick visualization for museums or online databases [6]. We were inspired to investigate further into how inpainting algorithms can be implemented and improved for helping restoration.

5. Getting Started

Our project aims to compare three inpainting algorithms when applied to digital images of paintings as a way to simulate art restoration. We selected the Telea method and the Naive-Stroke method by Bartalmio and Shapiro, which are both procedural, as well as the PartialConv method which is data-driven. While the Telea and NS algorithms consider mainly local neighborhoods while inpainting, the CNN approach uses a more comprehensive look at the picture which is more similar to how a trained artist would inpaint.

Dataset: We selected a database called "Best Artworks of All Time" from Kaggle which can be found here. Although the title is debatable, it contains large digital portfolios from 50 influential painters throughout history as well as a brief history of each artist.

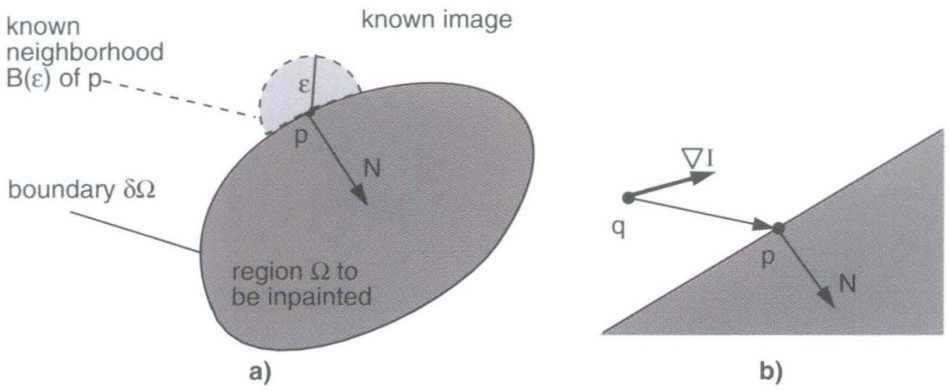
GUI: With these inpainting algorithms at our disposal, we set up an interface for digital art restoration. The result is a GUI that users can interact with to select a painter of their choice and generate a painting from their portfolio. This painting is then artificially "damaged" using a mask that overlays small circular or linear regions onto the image and the user can select their inpainting algorithm of choice to "repair" the image. Our goal is that eventually users could input an image of a painting that is damaged in real life and then use this generator to generate plausible reconstruction of the missing regions.

(Further information on the GUI and operation is available in the readme file)

6. Algorithms

Telea Algorithm: The Telea algorithm for inpainting (created by Alexander Telea in the 90s), is a procedural inpainting method that utilizes the Fast Marching Method (FMM) devised by Seithan to inpaint unknown pixels in a "natural" way. It is advantageous to other inpainting methods due to two main factors: simplicity of implementation and speed of computation. Our python implementation of this method is entirely based upon the pseudocode and implementation details provided in Telea's original paper.

Put simply, this method uses a band to propagate smoothness along the gradient using known pixels in the neighborhood. Given an area to inpaint, our band - called the Narrow Band - is exactly the boundary of this area. The unknown regions are then treated as level sets and the Fast Marching Method is used to propagate information inwards in an orderly fashion.



To inpaint a pixel *p*, one looks at the neighborhood *B* of known pixels within a certain radius. The intensity that is then assigned to this pixel is calculated as a weighted sum of these points, based on three factors, as described in Telea's paper.

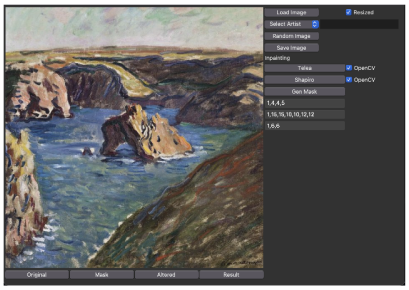
Fast Marching Method :
The Fast Marching Method (FMM) decides the order in which pixels are inpainted. Pixels closest to the known area are painted first which mimics real inpainting. This method solves the Eikonal equation which gives us a distance map denoting how far each pixel in the image to be inpainted is from the border.

The FMM does this by creating a band of pixels that separates the known from the unknown pixel areas, this corresponds to the inpainting boundary. To initialize this, each pixel stores three values: the distance to boundary value, the intensity at that pixel, and a flag that denotes it as on, outside, or inside the band. Pixels are inpainted iteratively by 'selecting' (pop the heapqueue) the point in the band with minimum T, iterating over the four quadrant of the band neighborhood and inpainting (if the neighbor is unknown), and then updating the T value for the neighbor.

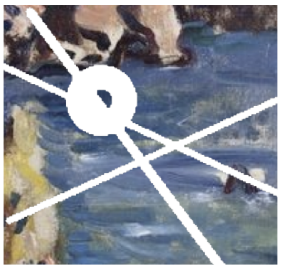
Our Implementation:
As previously mentioned, the implementation is mostly based upon pseudocode provided in Telea's original paper, but some details are different. First, the flags and distances are stored using two arrays, F and T respectively. The narrow band is maintained with python's heapqueue, and is simply a list. Everything is initialized as described by Telea, including the precomputation

step calculating the distance field of known pixels in the neighborhood around the band, but after this all distances are set to negatives, this has no effect on the processing as the Eikonal solve is independent of sign as long as all inputs are negative or positive. The precomputation is done by reversing flags and running FMM (without inpainting). Our algorithm runs less than or around 3 seconds for most if not all images, and overall is less than 300 lines of code.

Results:

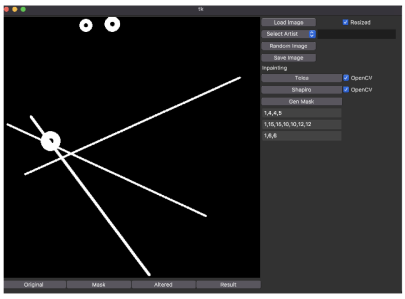


Original Painting



ROI of Masked Original

Our Telea



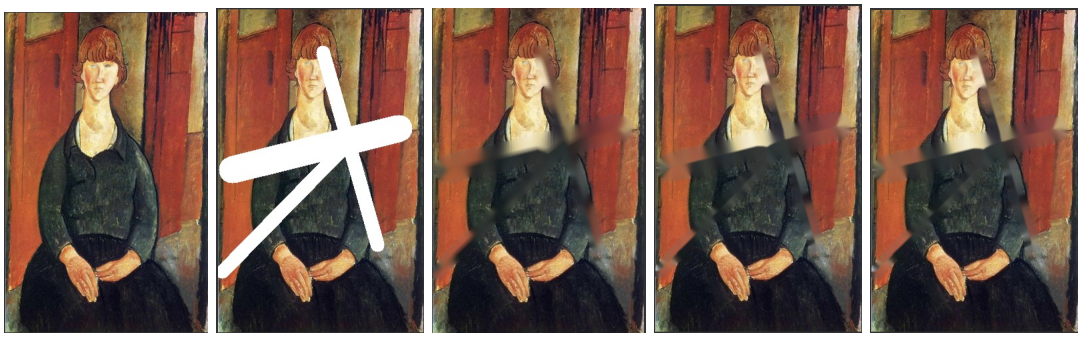
Mask

Open CV Bartalmio Sapiro

Open CV Telea

We can see that our implementation is almost identical to the openCV Telea method, and is only slightly more blurry than NS. This becomes worse as the thickness of the region to inpaint increases however, but we can see that the openCV methods also struggle.

Original *Altered* *Our Telea* *openCV Telea* *openCV NS*



Naive Stroke Method:

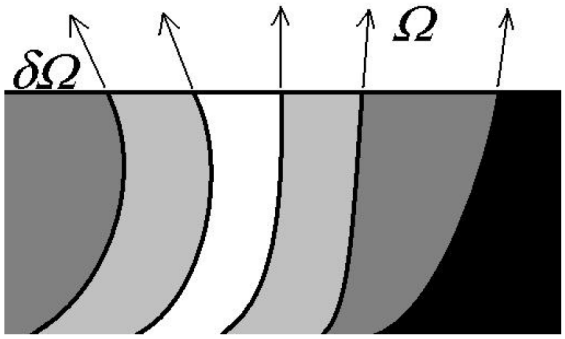
Description
This algorithm from the paper "Navier-Stokes, Fluid Dynamics, and Image and Video Inpainting" by Bertalmio [1]. The Algorithm is essentially based on the Partial Differential Equations method which incorporates concepts from classical fluid dynamics. The idea here is to treat the image intensity as a 'stream function' and the Laplacian of the image intensity act as vorticity of the fluid which is transported from the exterior (boundary of the region) into the region to be inpainted by a vector field defined by the stream function. The resulting algorithm is designed to continue isophote lines (equal intensity gray level lines) while matching gradient vectors at the boundary of the inpainting region.

Mathematical relationship between inpainting and NS equation from fluid mechanics:
We first establish the problem of image inpainting problem in mathematical terms**:**
Assume:

Ω will be referred to as the inpainting region.
I original damaged image
I* refers to the solution where I* is same image I but with the region Ω
The process of finding the appropriate I* is called an inpainting problem.

Now, the inpainting algorithm extends edges from the boundary of Ω, connect these extended edges, and then fill in intra-region accordingly. (This is basically the general ides of PDE based image inpainting algorithm)

Next we define isophotes as level lines of equal gray levels.



The direction of the isophotes can be given by

$$\nabla^{\perp} I$$

From course lecture we know that the smoothness can be interpreted as,

$$\Delta I$$

Now the gray levels in the intra-regions are simply determined by the smoothness in Ω and the gray levels on the boundary of Ω. This creates our solution constraints for the inpainting problem. The solution I* satisfies

$$\nabla^{\perp} I^* \cdot \nabla \Delta I^* = 0$$

In the Bertalmio algorithm, we propagate ΔI in the direction of $\nabla \perp I$ from the boundary of Ω and when all information is propagated we have a solution I*.

Relationship with Fluid mechanics:

In Fluid mechanics Newtonian flow obey the Navier-Stokes equations,

$$v_t + v \cdot v = -\nabla p + \nu \Delta v$$

$$\nabla \cdot v = 0$$

where

v is the velocity vector,

p is the pressure

ν is the viscosity.

For 2D flows, we introduce a stream function Ψ, so

$$\nabla \perp \Psi = v$$

(Eliminates p in and identically satisfies the divergence free condition in)

Letting $\omega = \nabla \times v$, the vorticity, we obtain the vorticity-stream function formulation for the Navier Stokes equations:

$$\omega_t + v \cdot \nabla \omega = \nu \Delta \omega$$

When viscosity is nearly zero, $\nu \approx 0$, we have the steady state solution of approaching,

$$v \cdot \nabla \omega = \nabla \perp \Psi \cdot \nabla \Delta \Psi \approx 0$$

This equation is similar to image inpainting criteria that we created before. Replacing Ψ with an image matrix I. So, we compare the image inpainting and fluid mechanics as follows

Fluid dynamics	Image processing
stream function Ψ	Image intensity I
fluid velocity $\vec{v} = \nabla^{\perp}\Psi$	isophote direction $\nabla^{\perp} I$
vorticity $\omega = \Delta\Psi$	smoothness $w = \Delta I$
viscosity ν	anisotropic diffusion ν

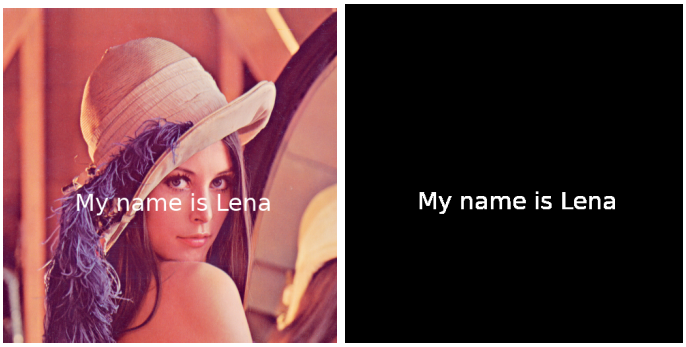
Algorithm:

As stated, the algorithm makes use of the surrounding regions of the edges (color information) and inpainting the interior of the region to be painted.It travels across the region and propagates color information from the outer region to the interior of the patch.

(We assume that the edges are continues so we can apply PDE)

The fill-in is done in such a way that the isophote lines arriving at the unknown region's boundary are completed inside, which allows the smooth continuation of information towards the center of the unknown region. The algorithm tries to preserve gradients (e.g. edge) and at the same time continue to propagate color information in smooth regions. The authors set up a partial differential equation (PDE) to update image intensities inside the region with the above constraints. The image smoothness information is estimated by the image Laplacian, and it is propagated along the isophotes (contours of equal intensities).

Implementation:
Because of the complexity of the algorithm, we have used OpenCV inpaint() function with the flag INPINT_NS, to restore the given picture.



The above two are input (marked image and mark) for OpenCV

Left is the output of Telea implementation
Right side is the output for Naive Stroke Implementation.
(The above results are produced by OpenCV library)

Image Inpainting (for Irregular Holes) Using Partial Convolutions

Description:
Partial Convolution is one of the most modern, Machine Learning (or Data Driven), approaches towards image inpainting problems. Developed by NVIDIA research, it is a deep learning technique to inpaint (damaged/marked) images by infilling holes with plausible pixel predictions. The main structure is based on U-Net architecture with partial convolution layers, and the training is performed using six different loss functions.

Partial Convolution:
In the context of image, the standard convolution layer has only one input (image), whereas in partial convolution, there are two inputs, a mask and an image. The image layers are generated by first calculating the element-wise product of the previous mask and image layer, followed by a convolution. However, the mask layers are generated in a similar way as a normal convolution.

U-Net architecture
U-Net structure (could be depicted as shape of 'U') is a method where the image dimensions undergo not only progressive reduction (using encoders layers) but also the U-Net recovers the image back to its original size using progressive expansion (using decoders layers) unlike linear CNN where the dimensions of the layers are progressively reduced. (The general purpose of CNN is to produce an output, usually, a vector, representing a class prediction, therefore it doesn't need to go to expansion layers). The normal convolutional layers are replaced with partial convolutional layers.

To prevent information loss, layers from the encoder section are concatenated with corresponding layers on the decoder section. The output of a U-Net is an image with the same dimension as the input image. (The output image is modified (in painted) image)

Partial Convolution function (From the paper):
For image:
Let W be the convolution filter weights for the convolution filter and b its corresponding bias. X are the feature values (pixels values) for the current convolution (sliding) window and M is the corresponding binary mask.

$$x' = \begin{cases} \mathbf{W}^T(\mathbf{X} \odot \mathbf{M})\frac{\text{sum}(\mathbf{1})}{\text{sum}(\mathbf{M})} + b, & \text{if sum}(\mathbf{M}) > 0 \\ 0, & \text{otherwise} \end{cases}$$

The scaling factor sum(1)/sum(M) applies appropriate scaling to adjust for the varying amount of valid (unmasked) inputs

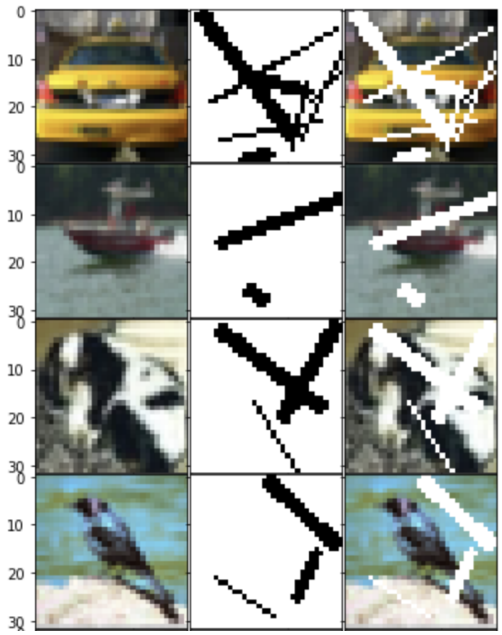
For mark:
If the convolution was able to condition its output on at least one valid input value, then we mark that location to be valid. This is expressed as:

$$m' = \begin{cases} 1, & \text{if } \text{sum}(\mathbf{M}) > 0 \\ 0, & \text{otherwise} \end{cases}$$

Loss function:
The methods have an extremely complex loss function that covers both per-pixel reconstruction accuracy as well as composition (i.e., how smoothly the predicted hole values transition into their surrounding context)

$$\mathcal{L}_{total} = \mathcal{L}_{valid} + 6\mathcal{L}_{hole} + 0.05\mathcal{L}_{perceptual} + 120(\mathcal{L}_{style_{out}} + \mathcal{L}_{style_{comp}}) + 0.1\mathcal{L}_{tv}$$

Datasets:
The training dataset is taken from Cifar10, which consists of 60000 32x32 color images (they are categorized in 10 classes, with 6000 images per class but we don't care about their label) There are 50000 training images and 10000 test images.
We use this dataset as it is relatively small. The implemented model has 16 partial convolution layers, so it wasn't possible for us to train the model with high resolution pictures. We have created a function that creates random marks (region to be inpaint) on the subset (batch) of the dataset. As stated above, we used both Images and marks to train the model.



The left column represents the original image (from dataset), the central column represent the mark (input to Partial convolution) and the right column represent the marked_image (input to Partial convolution)

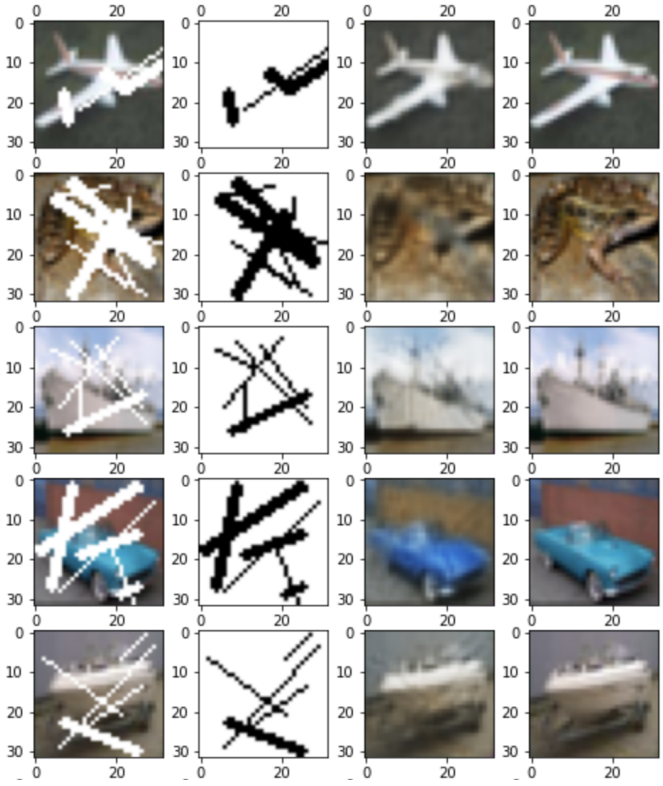
Modification:

- For loss function we have taken RELU for encoder and leaky RELU for decoder. This is done to make implementation and computation relatively easier.
- We have conducted only 4 epochs which training with Image dimensions of 32 * 32

In contrast the actual model was built in 50 epochs with multiple batches with several data resources both for images and marks (both high resolution). It is estimated that their team took 10 days to train the model with a supercomputer with strong GPU support.

The Research team at Nvidia have completely not disclosed their entire implementation to the public. Therefore, we took help from the following GitHub repository to complete the model. I have added reference in codes

- https://github.com/NVIDIA/partialconv (This is the main codes given by the author, used for the overall structural review)
- https://github.com/MathiasGruber/PConv-Keras (for keras implementation of Partial Convolution layers and all cnn_helper methods)
- https://stanford.edu/~shervine/blog/keras-how-to-generate-data-on-the-fly (for ML model implementation: Keras implementation for data generators)
- COMP 551: Applied Machine Learing (activation function, RELU instead of L_total for simplicity, comparison function and Model compile arguments)



the first column represent the the marked input image, the second column represent the mark, the third column represent the output of model, the last column represent the original image

7. Discussion and Conclusion:
Our implementation of the Telea algorithm suffered from two things in comparison to the openCV method, first, it takes longer to run, and two, we have slightly different color values for each inpainted pixel (color channel value generally different by no more than 2 compared to openCV implementation) which leads to a greater blurring effect. However, these are minor details, and in general our Telea works just as well if not better than the openCV implementation.

Compared to the Naive Stroke's model in openCV, our Telea implementation suffers more. The Naive Stroke's is again faster, but also there is a notable difference in the amount of blurring of the inpainted region; NS inpainting yields a sharper contrast with less smoothing. This can harm NS inpainting for some images, but in general leads to a better result.

In general, as the thickness of the area to be inpainted increased, our Telea method performed worse. Blurring increased in step with mask thickness, and as said in the original paper, Telea's algorithm is primarily for regions 3 to 15 pixels thick. With time, modifications could be made to the algorithm to allow for better performance on larger areas, for instance changing the algorithm to be also dependent on image depth information (less thick -> less smoothing, more thick -> more smoothing as band progresses).

We believe the advancement of machine learning in inpainting will outperform classical image inpainting methods. This is because even with a very small model (small training set and few epochs) the partial convolution model was able to detect and remove the marks. However, it does take significantly longer time to train the model as compared to no significant time required for classical methods. Due to complexity and computational limitations we have to modify the partial convolution model, however even with that many significant modifications we observe that the model was successful in identifying the marks and restoring the original image from those marks. However, If we notice carefully, when we compare the third column with the fourth column from the result, we notice that there is some color distortion from the original image. This may be caused by relatively small training of either the model or the loss function.

In terms of art restoration, the procedural approaches seem to fare well as quick renderings to fix small damaged regions. However, these areas are still relatively blurry and often miss global structures. Therefore, data-driven models seem to be more promising as they take in a more holistic view of the image similar to manual inpainting.

9. Contribution
Declan G. coded the GUI, the Telea implementation and wrote those sections.
Nicholas M. wrote the majority of the report and did research on inpainting techniques.
Sagar N. coded the Partial Convolution implementation and wrote that section and the Naive Strokes section.

References

[1] Bertalmio and Shapiro (2000) Image Inpainting, Proceedings of the 27th Annual Conference on Computer Graphics and Interactive Techniques, 417–424, DOI: 10.1145/344779.344972 https://ieeexplore.ieee.org/document/990497

[2] Darrow, Elizabeth Jane (2000) Pietro Edwards and the restoration of the public pictures of Venice, 1778-1819: necessity introduced these arts. https://digital.lib.washington.edu/researchworks/handle/1773/6223

[3] Cardinali, Marco. (2017). Technical Art History and the First Conference on the Scientific Analysis of Works of Art (Rome, 1930). History of Humanities. 2. 221-243. 10.1086/690580.

[4] Alexandru Telea (2004) An Image Inpainting Technique Based on the Fast Marching Method, Journal of Graphics Tools, 9:1, 23-34, DOI: 10.1080/10867651.2004.10487596

[5] Jireh Jam et al (2021) A comprehensive review of past and present image inpainting methods, Computer Vision and Image Understanding, Volume 203, 103147, ISSN 1077-3142, https://www.sciencedirect.com/science/article/pii/S1077314220301661

[6] https://www.labri.fr/perso/bugeau/JBAMI14/slides/F_Deger.pdf

[7] Deepak Pathak, Philipp Krahenbuhl, Jeff Donahue, Trevor Darrell, and Alexei A. Efros, "Context Encoders: Feature Learning by Inpainting," *Proc. International Conference on Computer Vision and Pattern Recognition* (*CVPR*), 2016.

[8] Guilin Liu, Fitsum A. Reda, Kevin J. Shih, Ting-Chun Wang, Andrew Tao, and Bryan Catanzaro, "Image Inpainting for Irregular Holes Using Partial Convolution," *Proc. European Conference on Computer Vision* (*ECCV*), 2018.

[9] Chan and J. Shen (2000). "Mathematical Models for Local Deterministic Inpaintings." Technical Report CAM OG-ll, Image Processing Research Group, UCLA.

[10] M.M. Oliveira, B. Bowen, R. McKenna, and Y.-S. Chang, "Fast Digital Image Inpainting," Proceedings of the International Conference on Visualization, Imaging and Image Processing (VIIP 2001), pp. 261–266, 2001.

[11] Connelly Barnes, Eli Shechtman, Adam Finkelstein, and Dan B Goldman. "PatchMatch: A Randomized Correspondence Algorithm for Structural Image Editing."*ACM Transactions on Graphics (Proc. SIGGRAPH) 28(3)*, August 2009.

[12] Criminisi A, Perez P, Toyama K (2004) Region filling and object removal by exemplar based image inpainting. IEEE Trans Image Process 13(9):1200–1212

[13] Shroff, M., Bombaywala, M.S.R. A qualitative study of Exemplar based Image Inpainting. *SN Appl. Sci.* 1, 1730 (2019). https://doi.org/10.1007/s42452-019-1775-7

[14] "Image Inpainting for Irregular Holes Using Partial Convolutions" Guilin Liu, Fitsum A. Reda, Kevin J. Shih, Ting-Chun Wang, Andrew Tao, Bryan Catanzaro. Published at ECCV 2018 https://arxiv.org/abs/1804.07723