Adv MM Project - Edge detection on Pointillist images

Khang Hui Chua, 2020280442
Kai Wen Yoke, 2020280598

January 10, 2021

Contents

1 Abstract

In this project, we explored traditional and deep learning methods of edge detection on Pointillist images. We found that the combination of preprocessing Pointillist images by reducing its size with pixel area relation and Gaussian blurring before performing Canny edge detection, and then upsizing the edge map and performing line erosion gives the best visual results for traditional methods of edge detection, while conventional methods of edge-preserving denosing such as bilateral and total variational filtering to work poorly. For deep learning based methods, preliminary testing of GAN models that learns directly edge contours from input images found this paradigm suitable for edge detection on Pointillist paintings. Besides, attempting to remove Pointillist styles from images can help to produce better edge detection result.

2 Introduction

In this project, we explore different ways to detect edges and contours in Pointillist images. Pointillism is a technique of painting in which small, distinct dots of colour are applied in patterns to form an image, with notable artists being Georges Seurat and Paul Signac. Notable Pointillist paintings are shown in Figure 1. However, the dotted texture and contrasting dot colours in Pointillist paintings make edge and contour detection very challenging.

Figure 1: Notable Pointillist paintings: (a) Morning, Interior by Maximilien Luce (1890); (b) The Entrance to the Port of Marseille by Signac (1918); (c) Parade de cirque by Seurat (1889)

Note: The notion of a good edge and contour is based mostly on human experience rather than a formal mathematical definition. While there are suggestions on how to quantify the goodness of edge detection, due to limited time, we simply judge visually the quality of detected edges.

2.1 Motivation

Edges, boundaries and contours are important features to detect in the field of computer vision, for purposes such as object detection and segmenting an image into regions of semantic concepts

for scene understanding etc. In the context of artistic works, salient contour extraction on paintings also facilitates the indexing and searching of paintings with a certain motif and content type (e.g. a particular person, flower etc) regardless of the painting style and texture used, especially when it is typical for artists to depict the same subject with different artistic media [1]. For this project, we arbitrarily choose Pointillist paintings to test out various edge and contour detection techniques.

3 Background and related work

Edge detection is a widely researched area in the field of computer vision, and like many other CV areas, the development of edge detection techniques can also be divided into two categories: traditional methods and deep learning based methods.

3.1 Traditional methods of edge detection

Traditional techniques of edge and contour detection have been extensively covered in [2]. Essentially, these techniques can be categorised into local and global operators, combined with preprocessing steps for noise and texture removal. The most representative traditional method of edge detection, the Canny edge detector, is based on a local differential operator, which is fast but not the best performing. Other than differential operators, there are also local contour detection techniques based on statistical approaches and local energy. Nevertheless, local methods only consider a small neighbourhood around a pixel to determine if it belongs to a contour, and neglect contextual and global information. Global methods like contour saliency computation, grouping pixels into contours based on Gestalt principles and active contours typically do better than local methods, albeit with greater computational cost.

3.2 Deep learning based methods of edge detection

The advent of deep learning and CNNs have also greatly improved the performance of edge detection. This is particularly apt since edges and contours are difficult to define mathematically and yet most humans share the same consensus regarding which pixels compose the edges and contours. CNNs have been found to be able to model human visual perception well, and are thus useful for the task of edge detection. Several datasets for learning and benchmarking edge detection have been proposed, with the most prominent one being the BSDS500 dataset [3], which comprises image and ground-truth boundaries hand-drawn by human subjects. CNN-based models like PhotoSketch [4] which learns to draw contours on photographs have achieved state-of-the-art performance. Recently, Art2Contour [1] uses GAN to learn the mapping from artwork images to contour drawings with state-of-the-art results, in an application most similar to ours.

4 Methodology

We pick several representative edge detection algorithms (with source code available and easily accessible online) to test, and also make our own improvisations based on inspiration from related work. We also generate synthetic Pointillist images from normal images for training CNN-based models. Finally, we show a visual comparison of our edge detection results, and discuss what is the key to producing good edge detection on Pointillist images. Below is a summary of the algorithms we tried:

1. Canny Edge Detector [5]
2. Preprocessing methods:
   (a) Filtering: Gaussian filter, Bilateral filter, Total Variational (TV) filter, Wavelet denoising filter
   (b) Resizing: `cv2.resize` with interpolation, Gaussian pyramids
3. CNN models:
   (a) Learn to draw conoturs: PhotoSketch [4]
   (b) Learn to remove Pointillist style: Pix2Pix [6], CycleGAN [7]

5 Experiments with traditional methods of edge detection

For traditional edge detection, due to time limitations and lack of code availability for other local and global edge detection methods, we mainly used the Canny edge detector (OpenCV implementation), with different preprocessing steps to remove texture and noise introduced by the Pointillist dots.

5.1 Canny edge detector

The Canny edge detector [5] is a popular edge detection algorithm proposed in 1986, and works based on local differential operators. It is implemented on OpenCV, with two adjustable parameters, lower and upper thresholds, for determination of edge pixels. OpenCV's Canny edge detector comprises of the following steps:

1. Noise reduction with 5x5 Gaussian kernel.
2. Find intensity gradient of image using horizontal and vertical Sobel kernel.
3. Non-maximum suppression to remove unwanted gradient pixels that do not constitute edges.
4. Hysteresis thresholding to determine which pixels are edge pixels.

While the Canny edge detector already starts off with Gaussian blurring preprocessing, we also experiment with other preprocessing methods to see if we can more effectively remove the Pointillist dot texture and noise before applying the Canny edge detector.

5.2 Preprocessing for texture and noise removal

We try out various methods of preprocessing in an attempt to prevent the Pointillist dots from interfering with edge detection. These include smoothing with different filters of different parameters and different ways of resizing the Pointillist image.

5.2.1 Smoothing filters

Our idea was that the Pointillist dots could be similar to image noise such as Gaussian and salt and pepper noise, which could be removed by smoothing filters.

1. **Gaussian filter**: The most common smoothing filter is the Gaussian kernel, but it smooths everything out, including the edges, which results in undesirable edge detection results after Gaussian blurring. Figure 2 shows the edge detection results after Gaussian blurring with kernel size 3 and 5, and standard deviation ranging from 2 to 32. The results are essentially garbage with any parameter, with spurious edges detected due to the dots, indicating that the Pointillist noise do not follow Gaussian distributions.

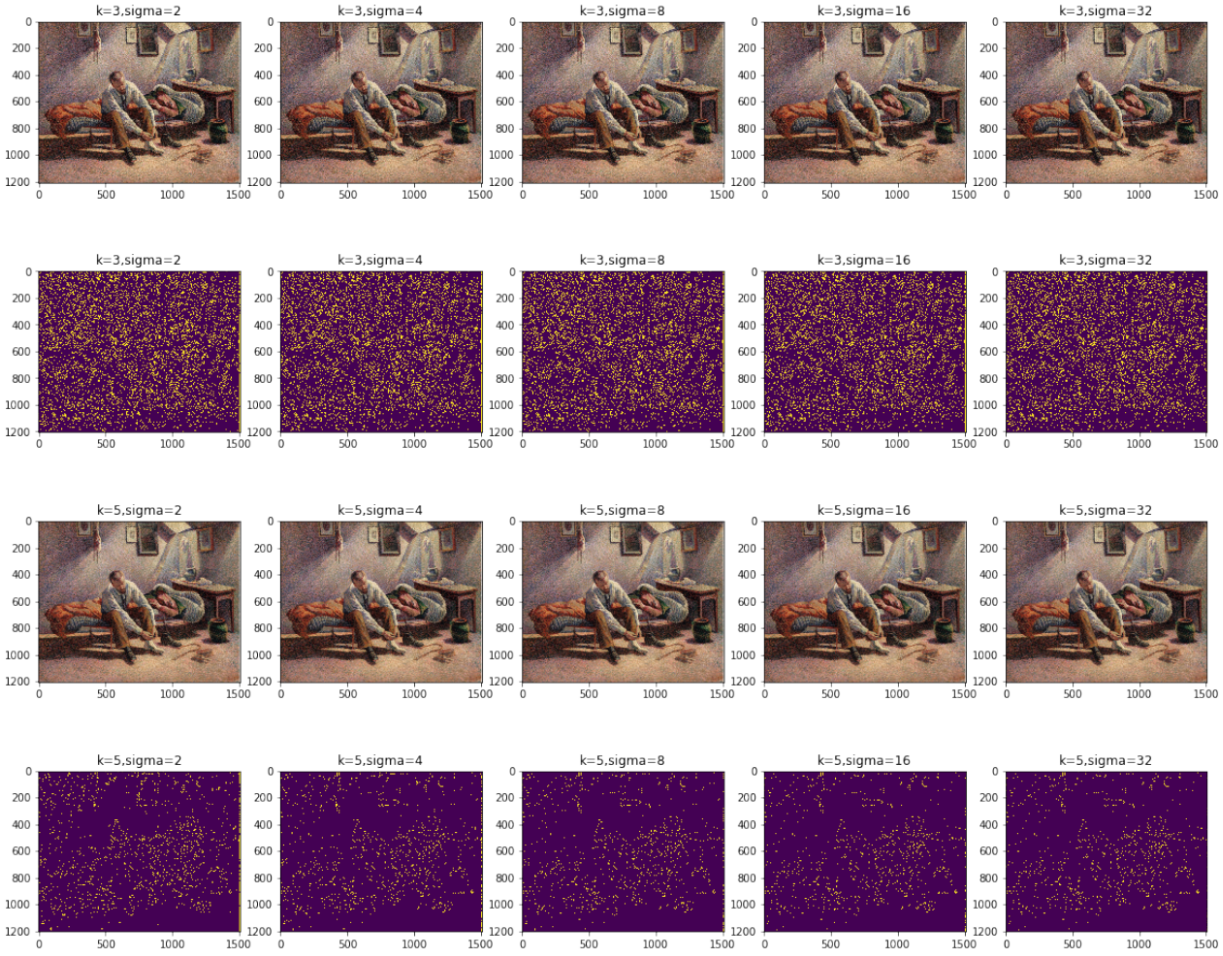

Figure 2: Gaussian blurring with various parameters followed by Canny edge detection

2. **Edge-preserving filter**: The bilateral and total variational (TV) filters are designed to smooth images while attempting to preserve edges. Figure 3 shows the edge detection results after performing bilateral filtering and Figure 4 shows the edge map after TV filtering. The results are rather bad. Although the bilateral filter seems to have smoothened out the image and removed the Pointillist dots quite effectively, the smoothing seems to have gone overboard, and the Canny edge detector fails to detect any edges. For the TV detec-

tor, many spurious edges are still detected. It seems that edge-preserving filters that are meant for normal images are useless for Pointillist images since the true 'edges' are not well defined in the first place due to the dots, so these filters are in fact preserving the wrong edges, resulting in poor edge detection performance.

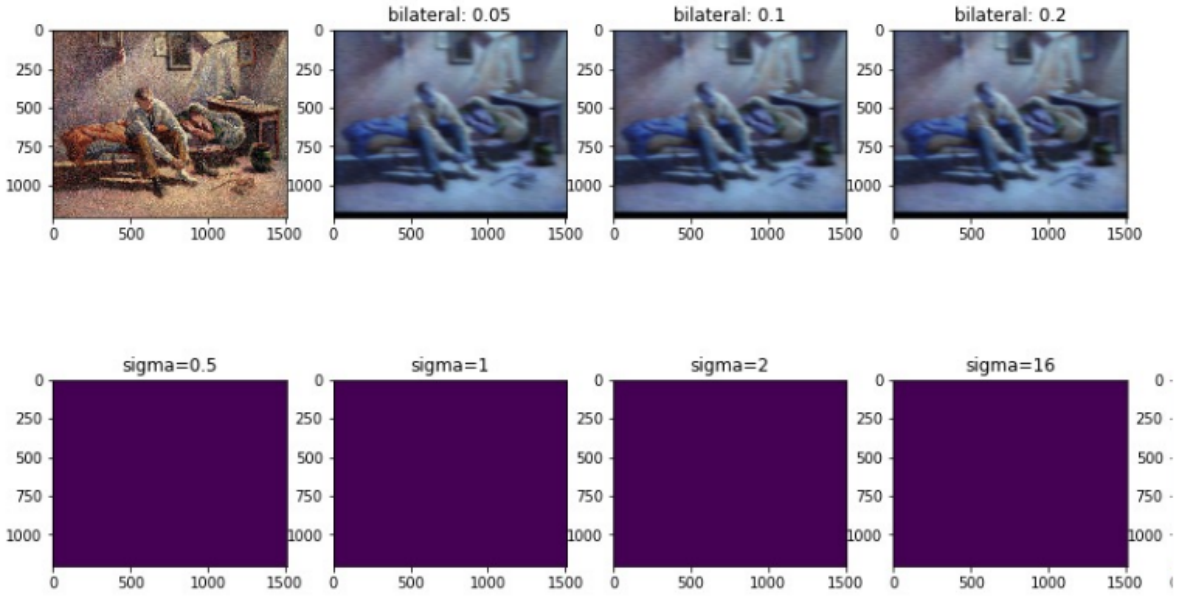

Figure 3: Bilaterail filter followed by Canny edge detection

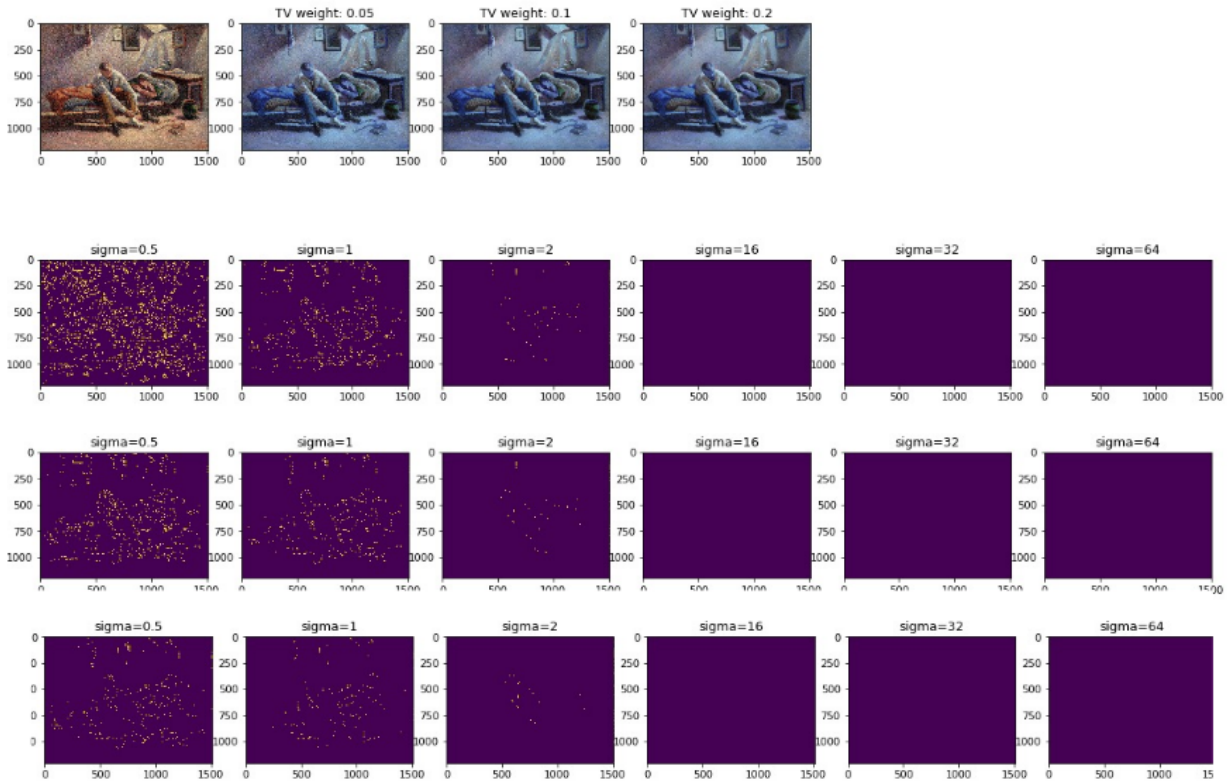

Figure 4: Total variational filtering followed by Canny edge detection

3. **Wavelet denoising filter**: The wavelet denoising filter works in the wavelet domain representation of the image. Again, conventional denoising filters like this do not work well on Pointillist images, as shown in Figure 5.

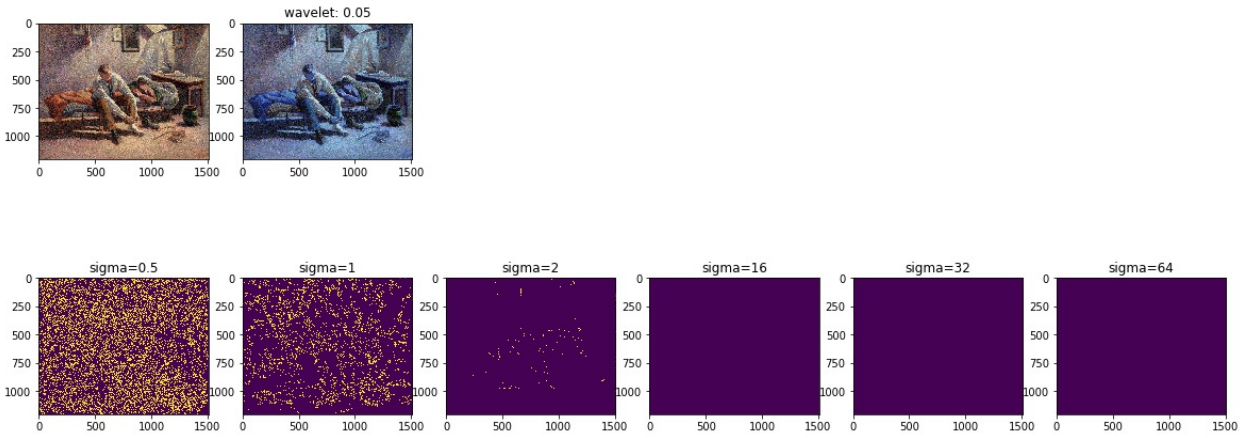

Figure 5: Wavelet denoising followed by Canny edge detection

5.2.2 Resizing and interpolation

Our idea is that similar to blurring, shrinking the Pointillist images will reduce the interference of the dots and make the contours and edges of the main subject depicted more defined, and thus making edge detection easier for the Canny edge detector.

1. **Resizing to different scales, with different interpolation**: Using OpenCV's `resize` function, we shrink the Pointillist images to 0.2 and 0.1 of its original dimensions, and applied different interpolation methods: `cv2.INTER_NEAREST, cv2.INTER_LINEAR, cv2.INTER_AREA, cv2.INTER_CUBIC`, with results shown in Figure 6. Indeed, greater extent of shrinking results in better edge detection, as less noise is captured. The type of interpolation method also affects edge detection performance significantly, and we see that `cv2.INTER_AREA` results in much better performance than the rest in Figure 6. According to OpenCV documentation, `cv2.INTER_AREA` is typically the best interpolation method for decimation as it gives moire'-free results. We also tried adding Gaussian blur after resizing, and then performing Canny edge detection, with results shown in Figure 7.

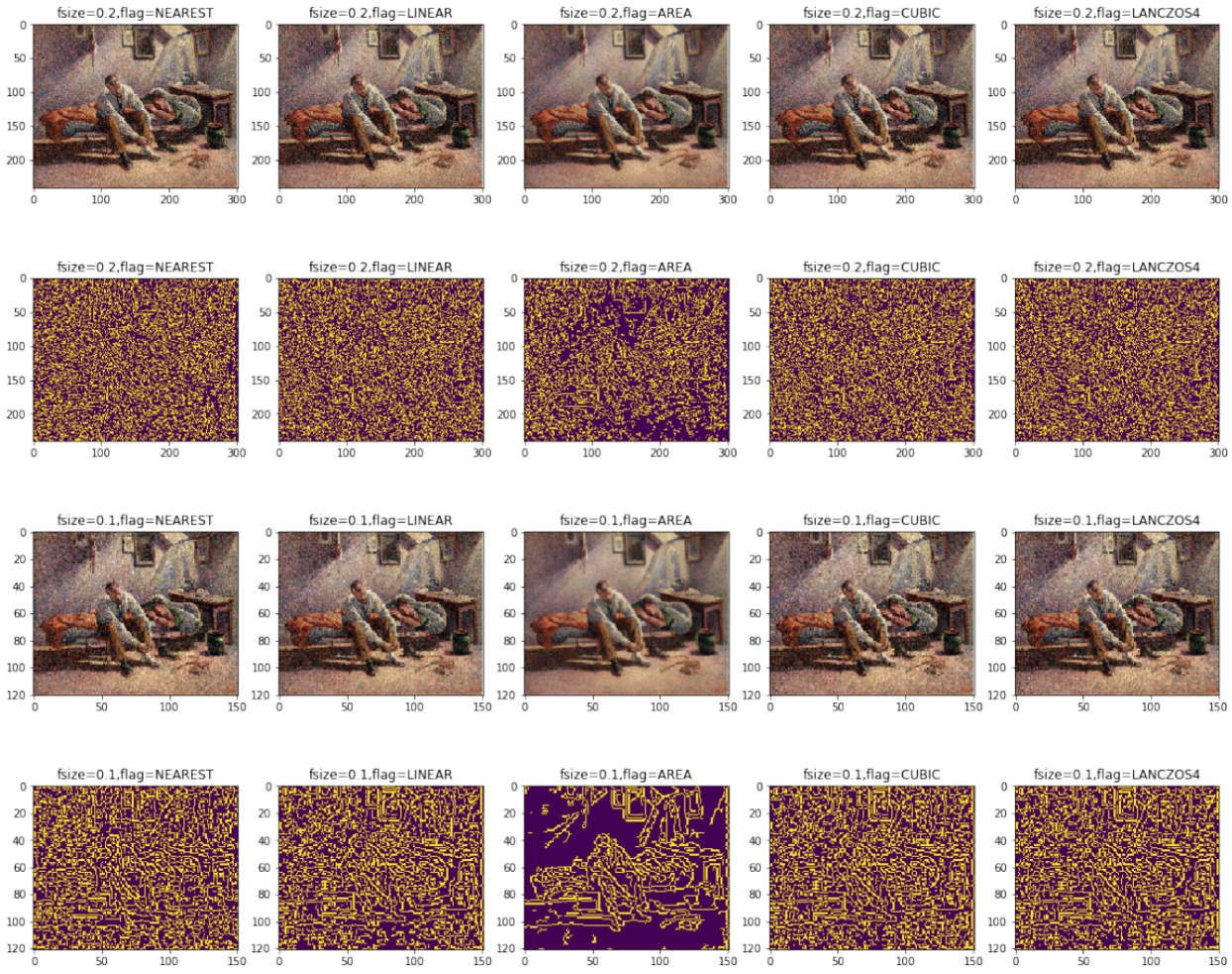

Figure 6: Resizing to different scales, across different interpolation methods, followed by Canny edge detection

The combination of resizing and Gaussian blur before Canny edge detection (Figure 7) seems to give the better results than solely resizing (Figure 6).

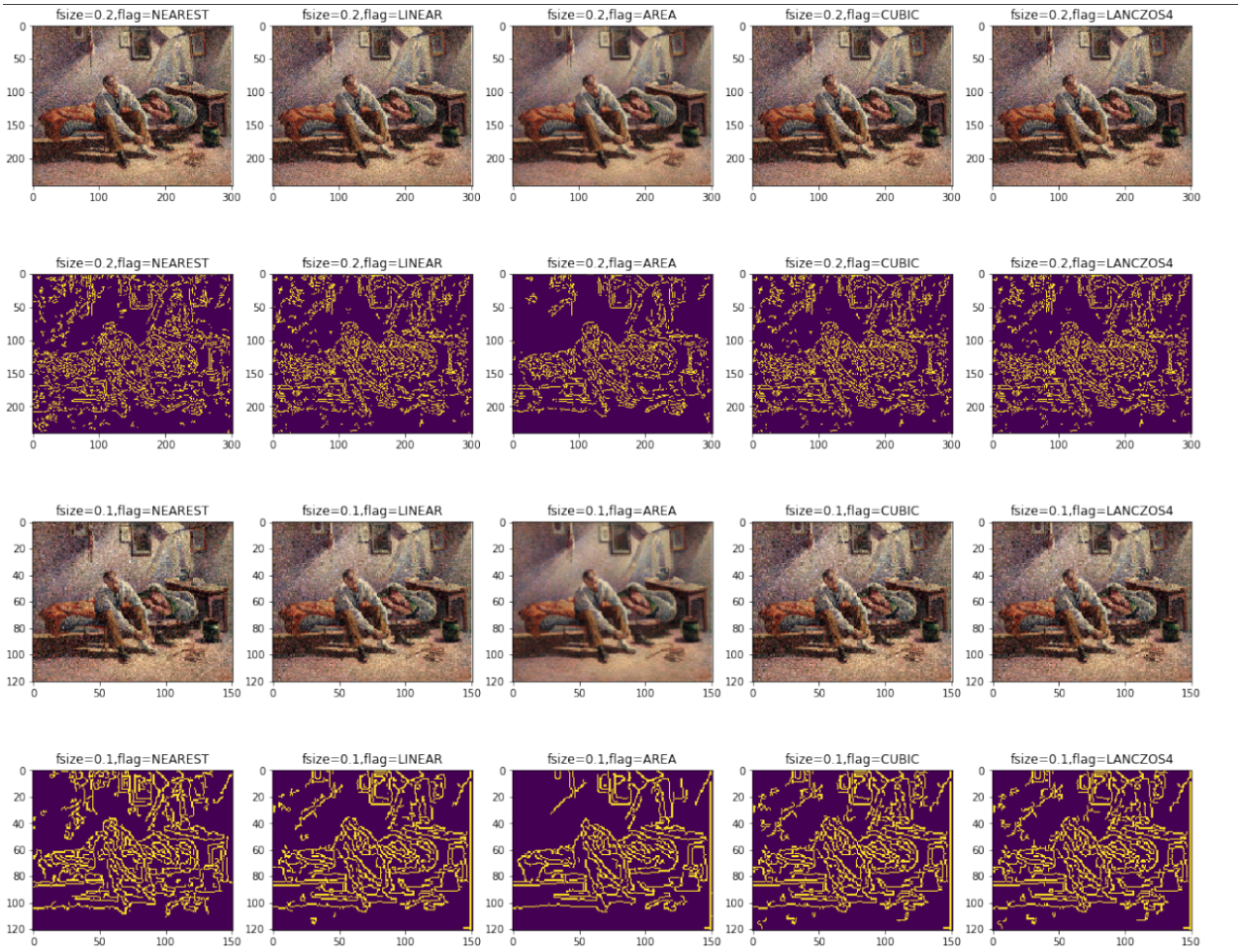

Figure 7: Resizing to different scales, across different interpolation methods, followed by Gaussian smoothing, then Canny edge detection

2. **Gaussian pyramid construction**: We implement Gaussian pyramid downsampling and upsampling using OpenCV's `cv2.pyrDown()`, `cv2.pyrUp()`, with five levels, with Canny edge detection on each level as shown in Figure 8. Each level reduces the image size by four using convolution with a Gaussian kernel (as opposed to interpolation for `cv2.resize`). We observe similar to resizing that the lower the resolution, the better the edge detection performance. Also, upsampling from the smallest level is not useful as the true edges are already lost.

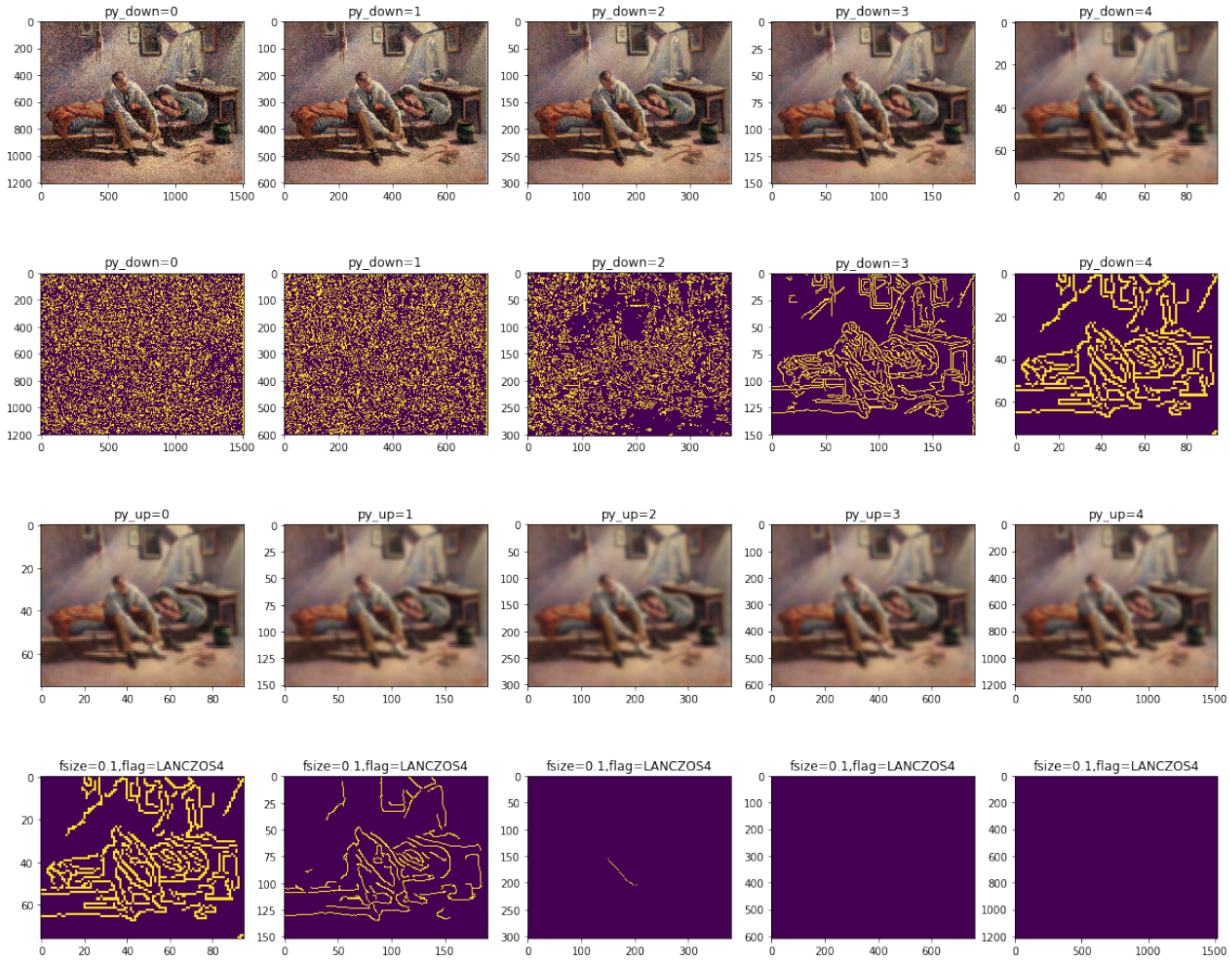

Figure 8: Downsampling and upsampling with Gaussian pyramids, followed by Canny edge detection

3. **Upsizing edge maps**: After downsizing the images and obtaining the edge maps, we should still upsize the edge map to the original image size using `cv2.resize`, and then apply erosion using `cv2.erode` to reduce the thickness of the edge lines.

5.3 Conclusion on traditional edge detection methods

It seems like shrinking the Pointillist image to a small enough size with either `cv2.resize + cv2.INTER_AREA` or `cv2.pyrDown` yields the most significant effect on Canny edge detection, especially when combined with Gaussian smoothing, while smoothing filters alone are insufficient to remove Pointillist dots and texture. Figure 9 shows that the combination of resizing, Gaussian smoothing and Canny edge detection, followed by upsizing and erosion, performs reasonably well across three Pointillist paintings.

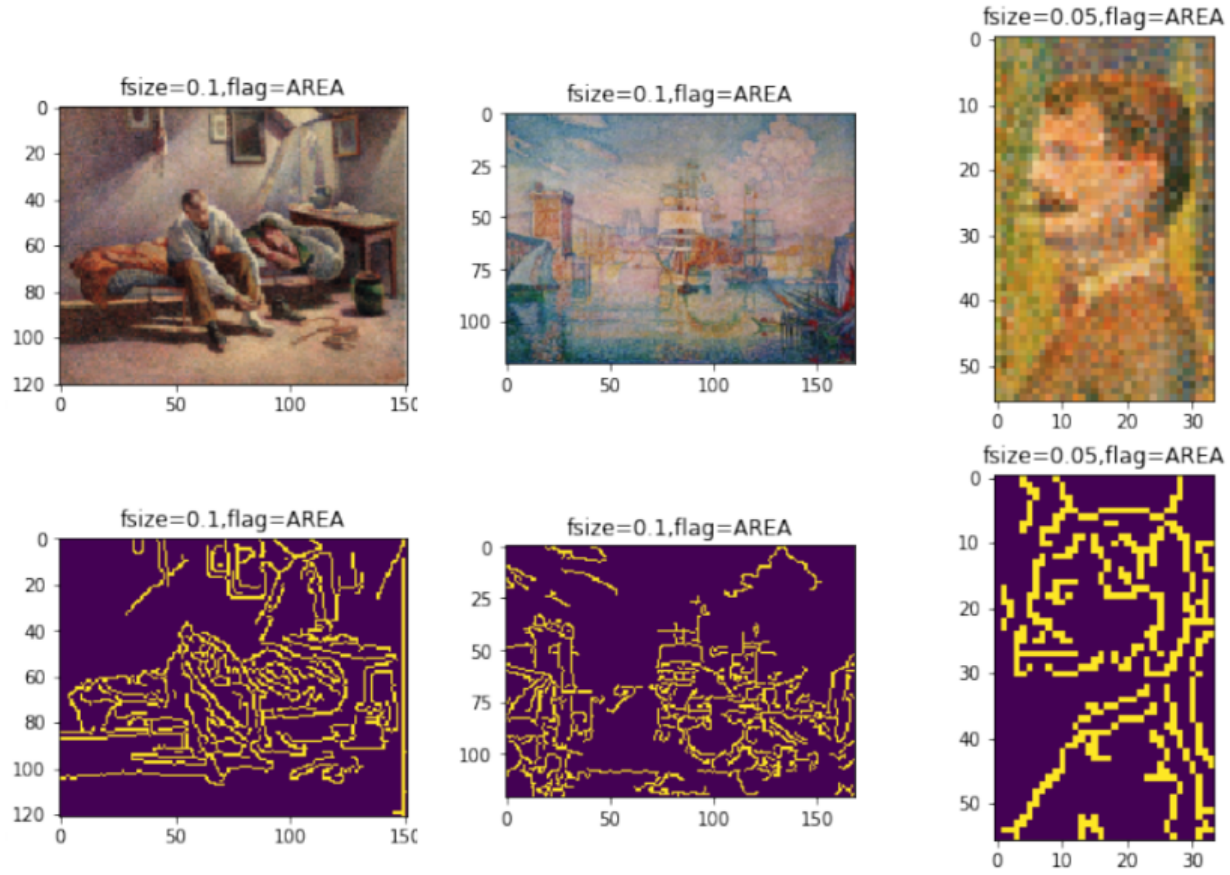

Figure 9: Results of edge detection on Pointillist paintings, resized, Gaussian blurred then Canny edge detection applied

While Canny edge detection with preprocessing performs quite well on Pointillist images by visual inspection, there are still some spurious edges not belonging to the main subject of interest. We subsequently try deep learning based methods to see if we can achieve better results.

6 Experiments with CNN-based methods of edge detection

For CNN-based methods of edge detection, we explored two paradigms: (1) directly learning the edges from the input Pointillist images, and (2) removing the Pointillist style from input images via 'inverse' style transfer and then applying Canny edge detection.

6.1 Synthetic Pointillist image generation

We generate synthetic Pointillist images from normal images to generate ground truth and corresponding Pointillist versions image pairs for CNN training. We used 1000 source images from ImageNet [8] which depicted main subjects in the centre, and approximately 7000 scene images from the Van Gogh dataset [9], and applied our Pointillist style rendering algorithm to these 8000 images. Example Pointillist image and ground truth pairs are shown in Figure 10, with the left images being from ImageNet and the right ones from the Van Gogh dataset.

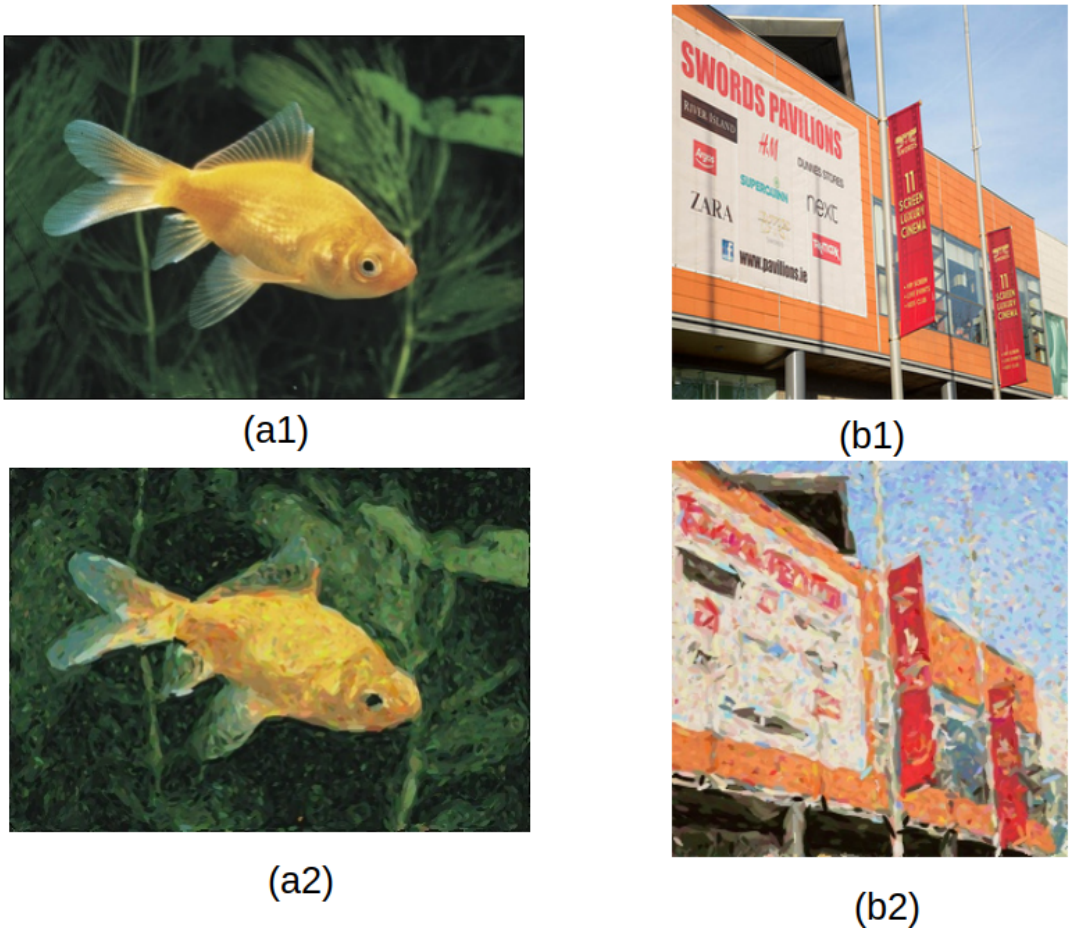

Figure 10: Original image and corresponding generated Pointillist style

6.1.1 Pointillism rendering algorithm

In the area of non-photorealistic rendering, there are various ways of simulating artistic styles, such as through stroke-based rendering, mosaicking and tiling, region-based techniques and image filtering [10]. For synthetic Pointillist image generation from normal images, we used the stroke-based rendering implementation by [11]using OpenCV. The idea is to first generate a colour palette coherent with the original image which are also vibrant and distinct, and then simulating different stroke size, length and direction to achieve a Pointillist effect.

6.2 Learning edges directly from input images

Several works such as [1] [4] found CNNs useful in learning object contour shapes from images, and [1] even showed that their GAN-based contour detection algorithm works on paintings (see Figure 11).

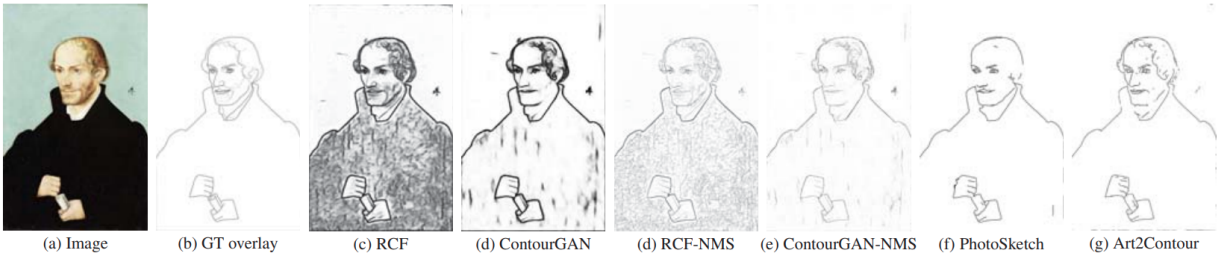

Figure 11: CNN-based contour detection results on paintings [1]

However, these algorithms rely on a large dataset of image pairs with their ground-truth contours hand-drawn by humans. While there exists photographs and object boundary/contour datasets such as the BSDS500 [3], there is no such dataset for Pointillist paintings, and we also do not have to time to annotate all our training set. Therefore, training from scratch is not feasible for our project scope and instead, we run pretrained models available online to see how they perform on Pointillist paintings that they were not trained on.

6.2.1 PhotoSketch (pretrained) [4]

PhotoSketch is a state-of-the-art contour generation GAN-based model. It directly generates the contours without requiring any pre or post-processing of the input image. The authors provide their pretrained model online, and we tested it on real and synthetic Pointillist images to observe its performance. Figure 12 shows the performance of PhotoSketch on the three famous Pointillist paintings, showing excellent performance on (a) *Morning, Interior*, while failing on the other two paintings. This is understandable since it can also be visually observed that the other two paintings are more corrupted by the Pointillist style, and the model has not been trained on such images before.

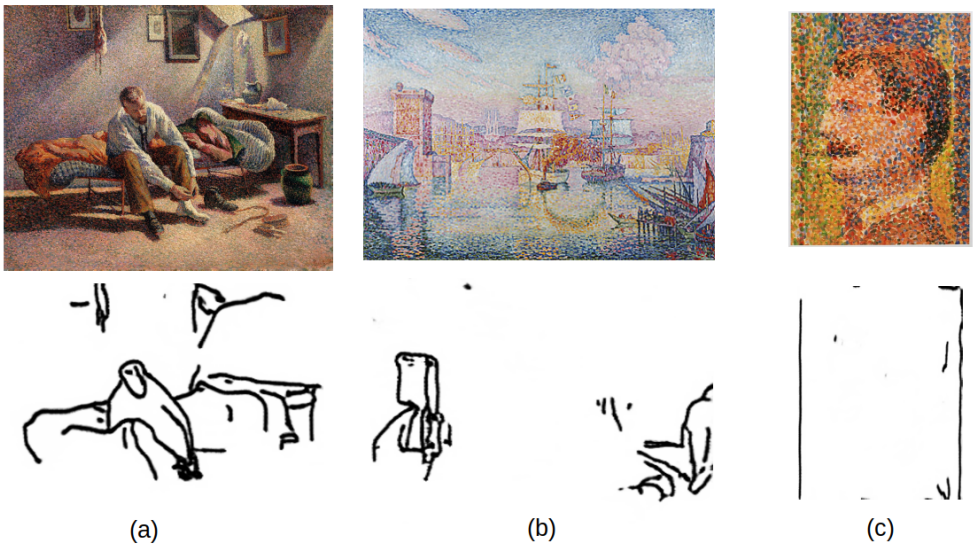

Figure 12: Pretrained PhotoSketch detection results on real Pointillist paintings

Figure 13 shows PhotoSketch's performance on our synthetic Pointillist dataset, which seems to yield better performance than the real Pointillist paintings.

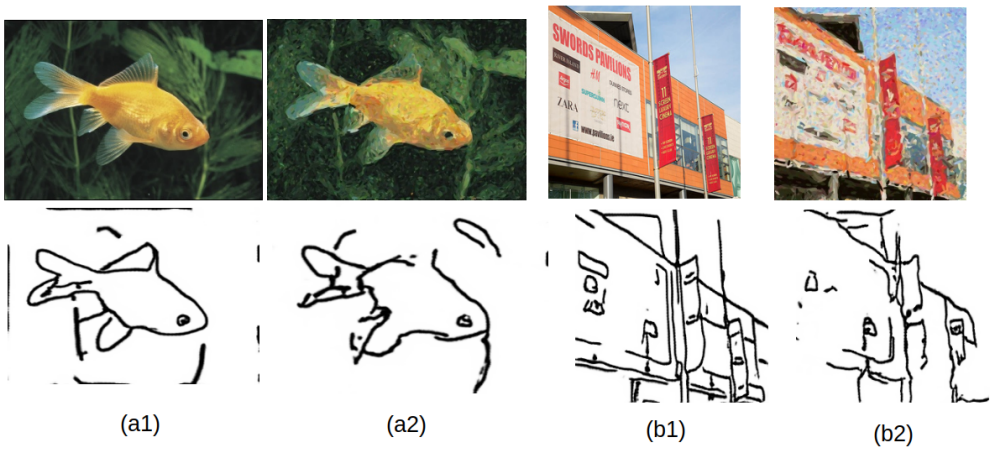

Figure 13: Pretrained PhotoSketch detection results on original vs corresponding synthetic Pointillist styles

Overall it is encouraging that PhotoSketch already performs quite well in contour and edge detection on Pointillist paintings, considering that it has not been trained on such data before, and also does not require any preprocessing on the Pointillist images. Nevertheless, the limitation is that it takes effort to build a good training dataset of ground-truth contour labels for the Pointillist image if we were to tune it to work well on Pointillist images. This is why we consider another deep learning based training strategy that does not require ground-truth contour labels.

6.3 Removing Pointillism style from input images

Another idea we tried out was to reverse style transfer with image to image translation, i.e. we use the same architecture as GAN-based style transfer models, and instead of transferring the Pointillist style to the original input image, we convert Pointillist style images to their original forms. Then, we apply preprocessing and Canny edge detection on the restored image just like what we did for traditional edge detection methods in section 5. This method does not require extensive ground-truth labelling of contours, unlike algorithms like PhotoSketch [4] which learns to draw contours directly from input images. The implementations of Pix2Pix [6] and CycleGAN [7] for style removal were based on PaddlePaddle [12].

6.3.1 Pix2Pix [6]

Pix2Pix uses a conditional GAN to learn a mapping from an input image to an output image. In our case, our input image is a Pointillist image, and our output is the original image without Pointillist style. Using our dataset of 8000 pair of Pointillist-ground truth image pairs, and with default parameters as in the paper [6], we trained Pix2Pix for 300k epochs. The result of restoring a synthetic Pointillist image is shown in Figure 14. The texture of the restored image indeed becomes much smoother, although the edges are not very clean. It seems like our idea is working to some extent. Next we test out the traditional Canny edge detection with preprocessing to observe whether results improve compared to no style removal.

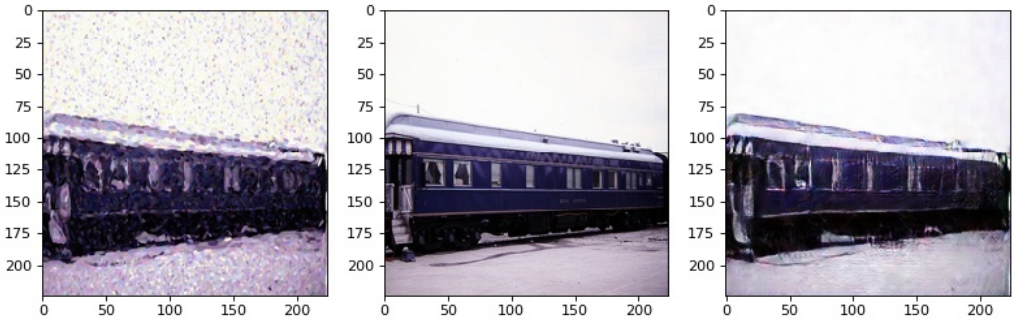

Figure 14: Pix2Pix training result after 300k epochs: Pointillist image, ground truth, restored Pointillist image

Figure 15 shows the Canny edge detection results on the Pix2Pix style-removed Pointillist images after resizing and Gaussian Blur. Indeed, the edges seem to be better defined and there is less noise from the Pointillist dots. A comprehensive visual comparison with the other traditional edge detection methods is conducted in section 7.

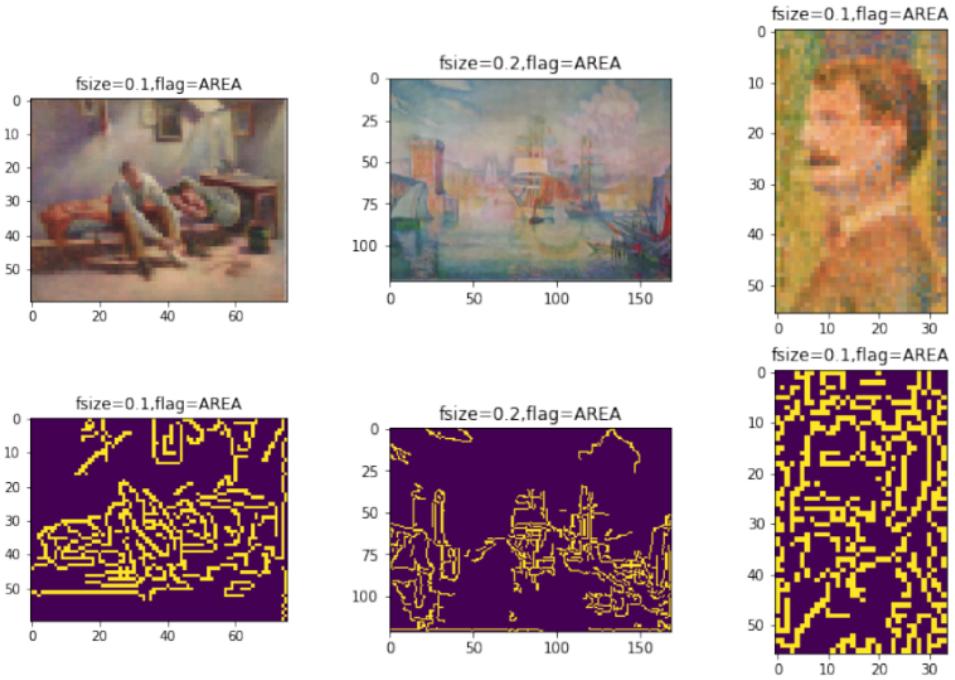

Figure 15: Pix2Pix Canny edge detection after resizing and Gaussian Blur

6.3.2 CycleGAN [7]

CycleGAN is another image-to-image translation network, but is superior to Pix2Pix because it does not require one-to-one image pairings, and instead only requires two discrete unpaired sets of images, to translate one style to another. In any case, we use the same training set as the one for Pix2Pix, and Figure 16 shows the style removal results after 100k epochs on three real Pointillist paintings. The results are rather poor, as the Pointillist dots clearly still have a strong presence, and it seems like only the image colours have changed slightly. Consequently, the edge detection results on CycleGAN restored images show no visible improvement (see section 7).

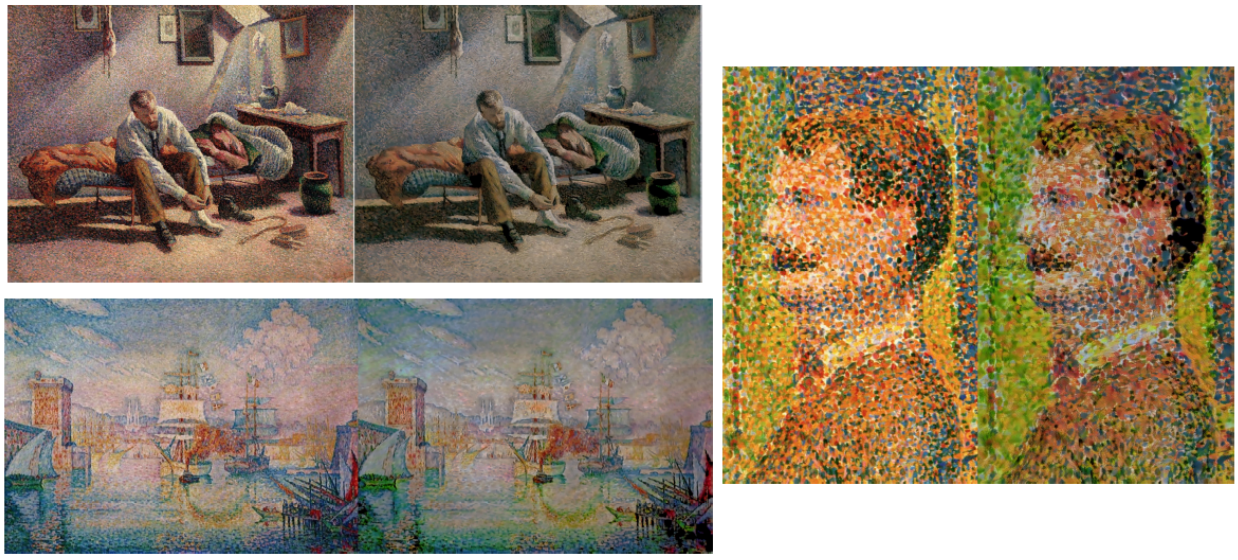

Figure 16: CycleGAN training result after 100k epochs for Pointillist painting and restored version

6.4 Conclusion for CNN-based methods of edge detection

We tried two ways of CNN-based edge detection, the first of which learns to draw edges and contours directly from input images, and the second which tries to learn to remove the Pointillist style from the Pointillist images, before we post-process the image with Canny edge detection. Due to time limitations, we were unable to fully try out the first method of contour learning, but results from the pretrained PhotoSketch model on Pointillist images that it has not trained before are promising, and perhaps might be the way to go for future work on edge detection on Pointillist images. The second idea of style removal before edge detection seem to yield decent result after post-processing with Canny edge detection for the Pix2Pix model, while the CycleGAN model fails to satisfactorily remove the Pointillist style.

7 Overall results comparison

To obtain the best result, it involves massive parameter tuning for each algorithm, such as threshold value for Canny detection, kernel size and sigma value for Gaussian Blur, the size and flag for image resizing, and the step for Gaussian pyramid downsampling. Without the groundtruth

of the actual edge result, we select the best visual result by eyeballing. The best edge result of each approach is shown as 17. Next, we preprocessed the image using pix2pix and CycleGan then used the generated image as input and followed by the above-mentioned parameter tuning. The results for GAN processed image are shown as 18. Our attempt to remove Pointillist styles from images using GAN helps to enhance the Pointillism image's feature and result in a better edge detection result.

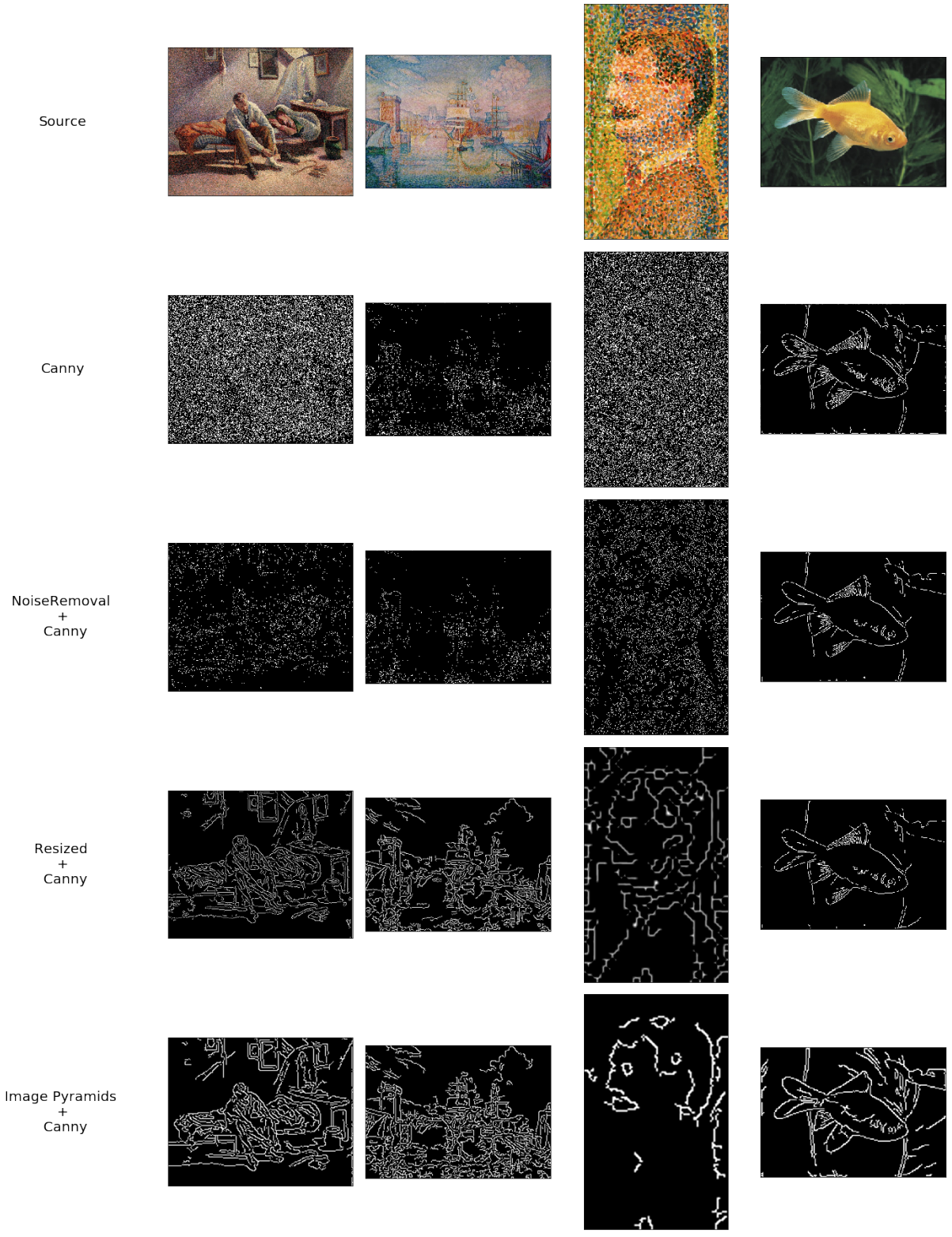

Figure 17: Results of edge detection methods on source images

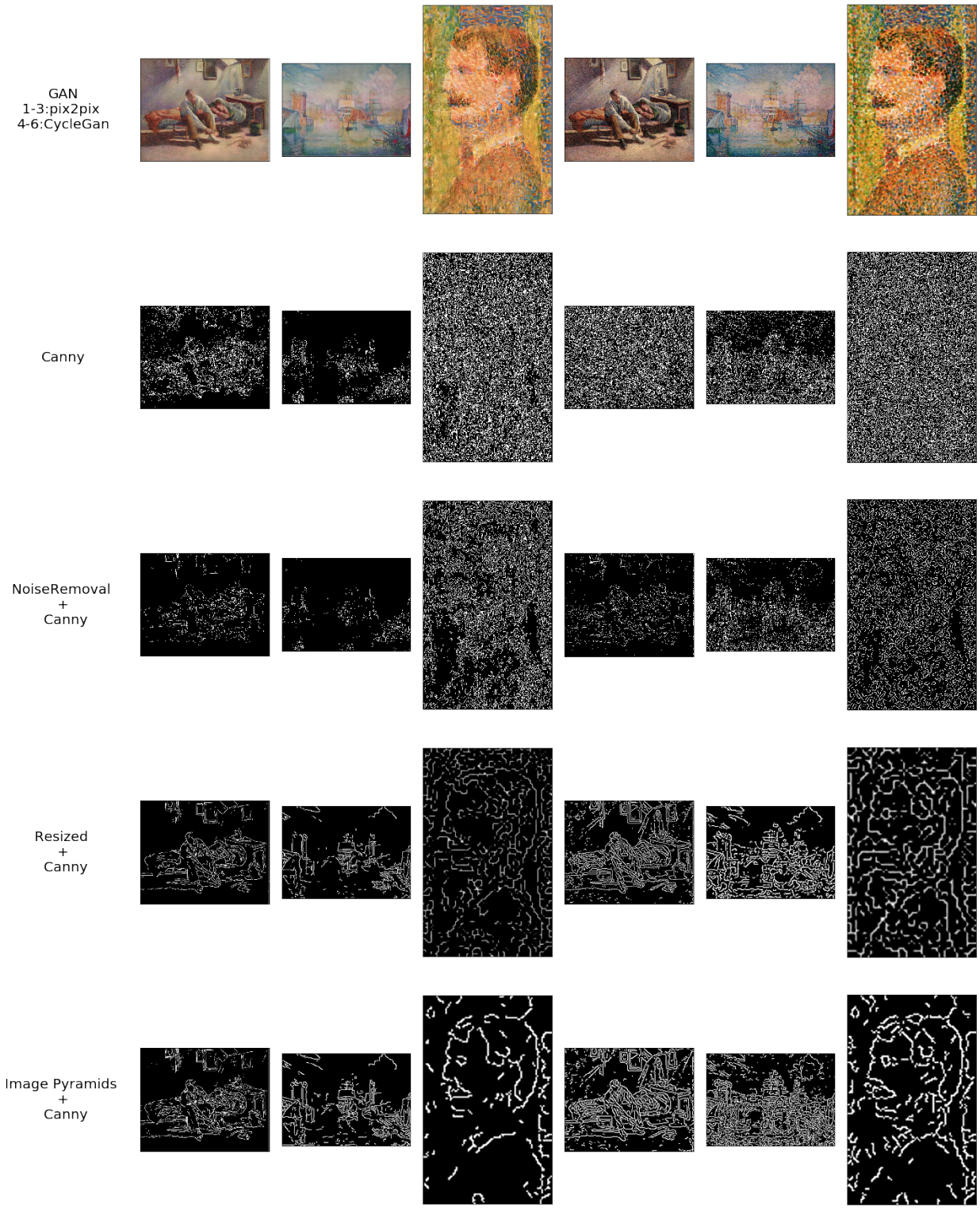

Figure 18: Results of edge detection methods on GAN processed images

8 Conclusion

In this project, we tried out various methods of edge detection on Pointillist paintings, and we found the best method to be removing Pointillist style using the Pix2Pix model, followed by downsizing, Canny edge detection, and then upsizing and edge line erosion. Nevertheless, there is still a whole area of methods we have yet to try, such as global saliency based edge detection algorithms that suppress texture information [2]. Furthermore, we did not manage to retrain GAN models like PhotoSketch [4] to directly learn contours from Pointillist paintings, which could yield even better results. The overall result comparision is shown in 19. Another issue is that we did not use quantiative metrics to evaluate our detection accuracy, which could be further worked on in future work. Finally, this project can be extended to other painting styles, with the aim of extracting contours and edges of the main subject regardless of artistic style and texture.

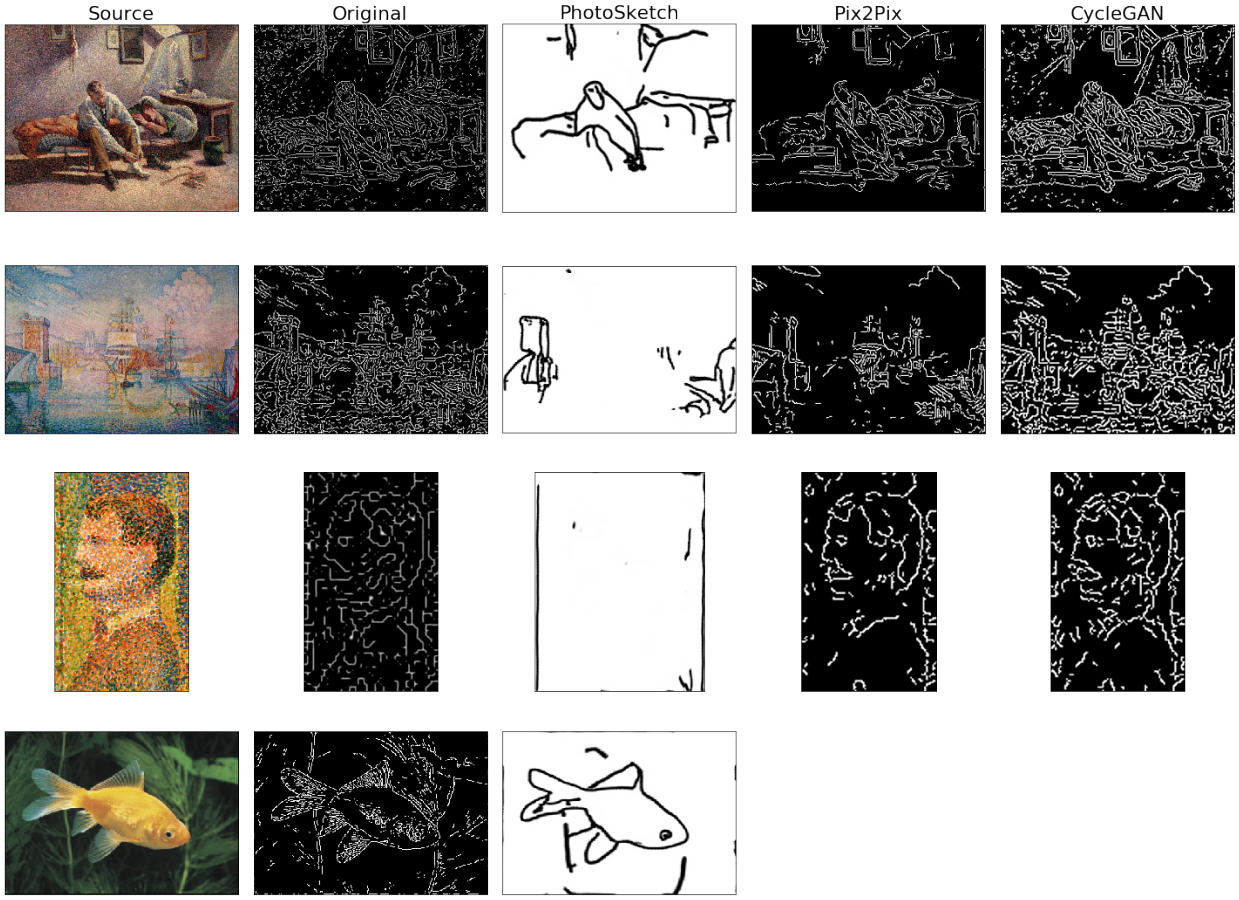

Figure 19: Best edge detection result of various approaches.

References

[1] A. Sindel, A. Maier, and V. Christlein, "Art2contour: Salient contour detection in artworks using generative adversarial networks," in *2020 IEEE International Conference on Image Processing (ICIP)*, 2020, pp. 788–792.

[2] G. Papari and N. Petkov, "Review article: Edge and line oriented contour detection: State of the art," *Image Vision Comput.*, vol. 29, no. 2–3, p. 79–103, Feb. 2011. [Online]. Available: https://doi.org/10.1016/j.imavis.2010.08.009

[3] P. Arbelaez, M. Maire, C. Fowlkes, and J. Malik, "Contour detection and hierarchical image segmentation," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 33, no. 5, pp. 898–916, May 2011. [Online]. Available: http://dx.doi.org/10.1109/TPAMI.2010.161

[4] M. Li, Z. Lin, R. Mech, E. Yumer, and D. Ramanan, "Photo-sketching: Inferring contour drawings from images," 2019.

[5] J. Canny, "A computational approach to edge detection," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. PAMI-8, no. 6, pp. 679–698, 1986.

[6] P. Isola, J.-Y. Zhu, T. Zhou, and A. A. Efros, "Image-to-image translation with conditional adversarial networks," 2018.

[7] J.-Y. Zhu, T. Park, P. Isola, and A. A. Efros, "Unpaired image-to-image translation using cycle-consistent adversarial networks," 2020.

[8] EliSchwartz, "Elischwartz/imagenet-sample-images." [Online]. Available: https://github.com/EliSchwartz/imagenet-sample-images

[9] [Online]. Available: https://people.eecs.berkeley.edu/~taesung_park/CycleGAN/datasets/

[10] J. E. Kyprianidis, J. Collomosse, T. Wang, and T. Isenberg, "State of the "art": A taxonomy of artistic stylization techniques for images and video," *IEEE Transactions on Visualization and Computer Graphics*, vol. 19, no. 5, pp. 866–885, 2013.

[11] M. Ronchetti, "Creating pointillist paintings with python and opencv," May 2019. [Online]. Available: https://medium.com/hackernoon/https-medium-com-matteoronchetti-pointillism-with-python-and-opencv-f4274e6bbb7b

[12] PaddlePaddle, "Paddlepaddle/paddlegan." [Online]. Available: https://github.com/PaddlePaddle/PaddleGAN/tree/master/docs/zh_CN/tutorials