Final report

Anime style camera filter

Ziyu SU

1. Introduction

The overall goal of my project is to create a camera filter which can change the real world landscape picture into a *Shinkai Makoto* Japanese anime style picture by image processing methods. The outcoming result will have this kind of effect:

Figure 1: Visual effect of the camera filter

In this social network era, camera filter is becoming more and more important for young people to decorate their Instagram, facebook, etc. Thus, lots of camera apps, like Prisma, VSCO and Meitu, thrive in these days. The Japanese anime movie *Your name*'s success spark a huge interest among those camera apps in Asia and many of them have launched their own anime style camera filters. The characteristic of *Shinkai Makoto*, who is the director of *Your name*, is that he combines the really existed landscape with fantastic color and fair tale like sky and cloud in his anime film. Therefore, in order to create a perfect fantastic effect, most of the camera apps choose to use machine learning or GAN technic to achieve their goal. This means the picture should be uploaded by user and processed within the deep learning model in the cloud. This is a time consuming strategy and really depends on the surrounding network signal strength, so that users cannot get their filtered picture when they are traveling in the places with poor signal. However, if we can process the picture in local, the whole process will only take a short time and be independent of the surrounding signal condition. That's why I want to create such a filter based on image processing method.

I think this method can make the filter more popular in the market. That's because, in addition to the perfect effect, ease of use is also a significant point for a product.

2. Related work

My project is highly related to the methods introduced in class. The whole procedure included image segmentation, thresholding, smoothing, edge detection, color mapping, image fusion etc.. Image stylization problem is always a hot topic in image processing and computer graph areas. As I introduced above, the method to realize this idea have been commonly researched by those popular camera apps' company. The most famous one is *Prisma* which can use deeplearning method to recreate the picture into different cartoon style.

Figure 2: Effect of Prisma's filter; (a) Original image; (b) filtered image

Besides, in 2018 CVPR conference, a team from Tsinghua University proposed their Cartoon GAN (Yang Chen et al. [1]) which uses GAN model to generate the Anime style image. Their result's effect is exactly what I want in my result. The details in their output image was fairly reduced and the overall image looks like a painting combined realism and fantastic style together.

Figure 3: Effect of Cartoon GAN

(collected from Yang Chen et al. [1])

In addition to those recently used deeplearning application, there were also lots of works which used image processing method to solve this problem. In 1998, Aaron Hertzmann et al.[2] proposed the *Image Analogies* to do oil painting stylization. They used a given picture

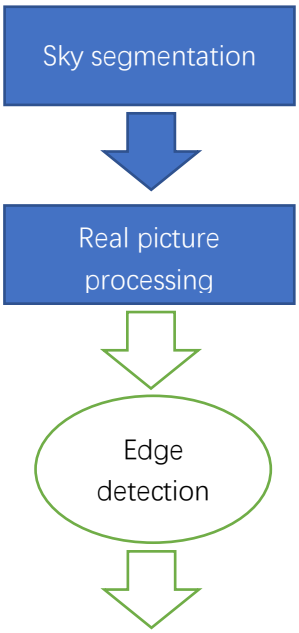
and its oil painting version, which drawn by painters, as template and use the relation between template's patches to map a new image into a stylization one. This is a straightforward approach, because they didn't do specific processing for specific parts, like outlines, color and sky. Thus, their result is quite "impressionistic". In 2002, Doug DeCarlo et al.[3] did a similar job as mine, which also used edge detection to create black outlines and tried to reduce details. Their approach mainly focused on how to use human's visual form in input image to simplify the image. Although their output image was simplified too much and worked only on portrait, their processing inspired me on how to create my own stylization method.

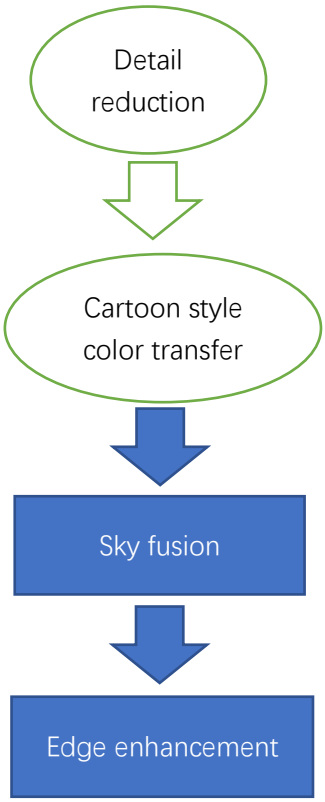
3. Data collection

The data I need to collect is the sky image from *Shinkai*'s anime and real world landscape pictures. I collected the sky images by searching on google image. In order to have a good effect, I specifically chose the movie stills with big sky and beautiful clouds on it. I collected the real-world picture by taking photos with my iPhone X which has dual rear cameras with 12M pixels and also from google Image which should be comparably clear as my own photos. So far, I have selected 45 scenery pictures from my album and website including different weather, different light strength and different sky conditions (cloudy, few clouds, pure blue, sunset).

4. Technical Approach

All of my processing was implemented by python 3.6, OpenCV toolbox and Anaconda's building packages, no extra packages need be downloaded.





4.1 Sky segmentation

Commonly used fast sky detection algorithm is color space thresholding method, so that I threshold the images under HSV color space. For here, I choose HSV instead of RGB, because HSV can help us to directly realize the hue and separate brightness information from color information. According to the analysis demonstrated in Frank Schmit et al.'s article [4], pixels with values satisfied at least one of these criterions could be detected as sky:

1. S<13 AND V>216
2. S<25 AND V>204 AND 190<H<250
3. S<128 AND V>153 AND 200<H<230
4. V>88 AND 210<H<220

According to my own experiment, the first two thresholds take charge of detecting the cloudy sky, the third one take charge of detecting light blue and the forth one take charge of detecting dark blue. Here are some binary results that images from left to right means original image and thresholding result of criterions 1,2,3,4 separately:

Figure 4: Thresholding result of using the four criterions separately. From left to right are the results of using criterion1, 2, 3 and 4

Before thresholding, I used histogram equalization on V channel of images to enhance the contrast of their brightness. Without doing this, some part of the ground in the images may have low difference with the sky and be mis-detected:

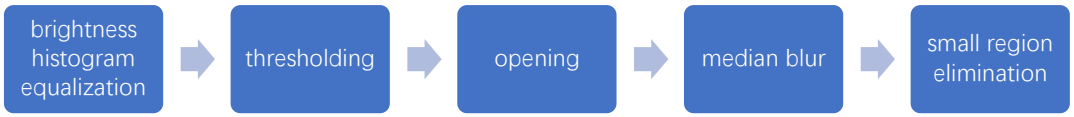
Figure 5: (a) Original image; (b) with V channel equalization; (c) without equalization

After the equalization and thresholding, I applied morphological opening on the thresholding image to erode the tiny sky connection. Otherwise, there may be some cracks near the boundary of sky and ground. I used OpenCV's MORPH_OPEN to apply the opening and my kernel is a 5x5 square kernel. Besides, I also used median blur to make the boundary of sky and ground to be smoother. The size of my median blur kernel is 9x9 and I used OpenCV's medianBlur to implement it.

There were still some bigger mis-detected regions in both ground and sky which cannot be eliminated by opening and blurring. Therefore, I sorted the segmented region's area to find small miss classified regions to get rid of them and only left the biggest regions in both sky and ground. I used OpenCV's regionprops function to achieve the result here.

Figure 6: (a) Sky segmentation before sorting; (b) segmentation result after eliminating small regions

In conclusion, the whole segmentation procedure is:



The binary segmentation sky mask generated here will be used for cartoon sky fusion later.

4.2 Real world picture processing

In addition to the fantastic sky, sharp edges and smooth details are also a characteristic of cartoon.

4.2.1 Edge detection

In order to create cartoon's sharp edge effect, I used canny algorithm to detect the edges and add them back with light black color after finishing the whole process.

In order to avoid the edges to be in the sky, which will destroy our fantastic cartoon sky, I used following procedure to eliminate them:

Pseudo code:
ground_img=img[ground_mask]
skyline_edge_mask=Canny(sky_mask)
edge_mask=Canny(ground_img)
final_edge_mask=edge_mask-skyline_edge_mask

I used the ground mask gotten from the sky segmentation part to create the ground image whose sky region is black. Then, the edge mask can be generated by applying Canny on the ground image. However, there will be a lone line on the skyline, since the sky region is black. Therefore, I detected that long line by applying Canny on the sky mask and subtract this long line from the former edge mask to get the final edge mask.

I used OpenCV's Canny function to implement this procedure and set the threshold as 50 and 210 to get appropriate amount of edges to avoid to make the result be messy.

4.2.2 Detail reduction

Then, to reduce the details in the picture, I will use bilateral filter to smooth the solid regions without smoothing the edge. Bilateral filter uses weighted neighbor values to compute the output value.

$$O(i,j) = \frac{\sum_{k,l} f(k,l)w(i,j,k,l)}{\sum_{k,l} w(i,j,k,l)}$$

The value of weights depends on the spatial closeness:

$$d(i,j,k,l) = exp - \frac{(i-k)^2 + (j-l)^2}{2{\sigma_d}^2}$$

(i,j are location of pixel, k,l are location of neighbors)

And intensity difference:

$$r(i,j,k,l) = exp - \frac{||I(i,j) - I(k,l)||^2}{2\sigma_r^2}$$

$$w(i,j,k,l) = d(i,j,k,l)r(i,j,k,l)$$

As the color difference between computed pixel and neighbor pixels are big in the edge, the weight will be small and the edge information will be reserved.

I used OpenCV's bilateralFilter function to implement this procedure and set the kernel size as 10, and σ_d, σ_r as 50

4.2.3 Cartoon style color transfer

Transferring our image's color to Japanese style bright color is also important for making our result looks like a Japanese cartoon image. I want my camera filter to do everything automatically, so that I take Erik Reinhard et al.'s [5] work as reference which can change images' color by calculating the statistic relation between our images' color space and the template image's color space.

Dr. Reinhard chose lαβ color space to do color transfer, but it is difficult to do numerical computing to get the sum and mean value of α and β spaces, since their value are too small. Therefore, I chose to transfer our source images and template image from RGB space into HSV space.

After I got the HSV value, I subtracted the mean value of data points:

$$S' = S - mean(S)$$

$$V' = V - mean(V)$$

Then, I scaled the data points by factors determined by the respective standard deviations:

$$S_{new} = \frac{\sigma_t^S}{\sigma_s^S} S' + mean(S_t)$$

$$V_{new} = \frac{\sigma_t^V}{\sigma_s^V} V' + mean(V_t)$$

Here, S and V are the source image's value, S_t and V_t are the template image's value, σ_t^S, σ_t^V are the standard deviation of template image and σ_s^S, σ_s^V are the standard deviation of source image.

I only chose S and V value to do transferring, because Hue value's changing may lead to a totally different color in our output.

The template image I used here is a cartoon image which has a typically color style. It will be shown in the intermediate result part.

After finishing the whole process above, we can get a cartoon like real world picture except the sky and black outlines.

4.3 Cartoon sky fusion

In this step, I used Poisson seamless clone to fuse the cartoon sky collected from animation films onto the picture acquired above.

Poisson seamless clone can perfectly fuse one image to another, since it is not trying to directly joint the image, but using the gradient of the second image to generate an image on the first image.

This algorithm formulates the problem into following equation:

$$\min_{f(x,y)\in\Omega} \iint_{\Omega} \| \nabla f(x,y) - \nabla S(x,y) \|^2 \, dxdy$$

$$s.t. \;\; F(x,y) = S(x,y) \; in \; connection \; region$$

F means first image, S means second image, f means the composite part in the result image and Ω means the composite region. In order to achieve the minimum, we set that:

$$\nabla^2 f(x,y) = \nabla^2 S(x,y) \; in \; \Omega$$

$$f(x,y) = \mathrm{F}(x,y) \; in \; connection \; region$$

Therefore, we only need to calculate the f's value under those settings and the second derivative can be calculate by Laplacian kernel.

First we use Laplacian kernel to calculate the second image's gradient:

$$\begin{vmatrix} 0 & 1 & 0 \\ 1 & -4 & 1 \\ 0 & 1 & 0 \end{vmatrix}$$

$$div(5) = [v2 + v4 + v6 + v8] - 4 \times v5$$

For a matrix like that:

$$\begin{vmatrix} 1 & 2 & 3 & 4 \\ 5 & 6 & 7 & 8 \\ 9 & 10 & 11 & 12 \\ 13 & 14 & 15 & 16 \end{vmatrix}$$

If we only have the gradients div(6),div(7),div(10),div(11), we can have:

$$[v2 + v5 + v7 + v10] - 4 \times v6 = div(6)$$

$$[v3 + v6 + v8 + v11] - 4 \times v7 = div(7)$$

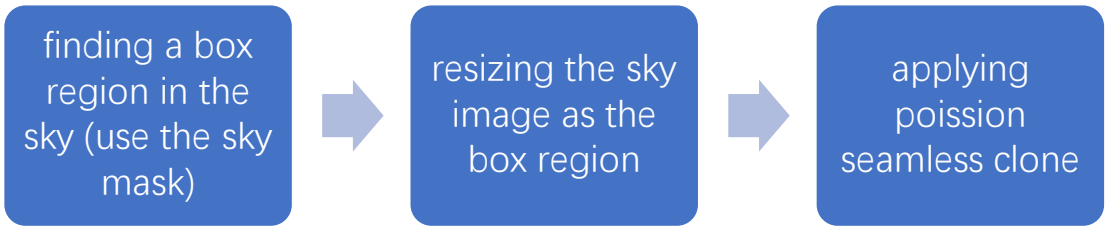
$$[v6 + v9 + v11 + v14] - 4 \times v10 = div(10)$$

$$[v7 + v10 + v12 + v15] - 4 \times v11 = div(11)$$

As the left side of the equations is a singular matrix, we won't have a unique root of this

question. However, in the image fusion task, we have the pixel values of the boundary of the first image. Therefore, it could be seem as we have the value of 1,2,3,4,5,8,9,12,13,14,15,16 and div(6),div(7),div(10),div(11), then the value of 6,7,10,11, which are the f's value, will be solvable.

The procedure of the fusion is:



By doing this algorithm, the cartoon sky image will be naturally fused with the real-world picture. In practice, I implemented this processing by using OpenCV's findingContours, resize and seamlessClone functions with the binary sky mask created formerly.

4.4 Edge enhancement

Here comes to the last processing. I put the edge mask onto every channels of the image, and set the edge regions as 100. Then, the outlines of the objects in the image will have a light black color.

By doing the whole processing in order, the real-world landscape picture will be stylized into a Japanese cartoon image.

5. Intermediate Results

5.1 sky segementation

These are the results of each steps in sky segmentation:

Figure 7: (a) Original image; (b) after thresholding; (c) after opening; (d) after median blur; (e) after small region elimination

We can find that there are a lot of cracks and small points in the binary mask after thresholding. Then the opening can eliminate all of the cracks and small point and median blur makes the boundary line to be smooth. However, opening and median blur cannot figure out bigger misdetection regions, such as those white blocks in the bottom, so that I did small region elimination.

Color thresholding method has its own limitation, since sky has so many kinds of conditions and colors. There will not be a universal criterion that can detect all kinds of sky. For example:

Figure 8: Failed segmentation

This picture has an ideal blue sky, at least in human visual, but our criterion failed to detect the sky. In addition, my filter cannot detect the orange and black sky. I think color thresholding will not be the perfect sky segmentation method and there should be more efficient methods based on more complex theorem.

5.2 Edge detection

In edge detection part, the most important improvement of my filter is that I subtracted the edge of the boundary between sky and ground from the original edge mask:

Figure 9: Images in the left are the results with original edge; Images in the right are the results with boundary subtracted edges

According to Figure 8, we can find that the long line in the sky has been significantly eliminated. However, my method still need some improvement, since there are still some short lines lying in where the original long line existed. I think it is because some of the zero value pixels were subtracted, and they increased to a big value under uint32 number accuracy in computer. As the Canny detector was applied on the image whose sky was totally black,

the gradient of the boundary between sky and ground must be 765 (add up the gradients in three channels,3x255). I have tried to eliminate the edges whose gradient is 765, but this method will eliminate some significant edges in the ground. Therefore, I need to explore furtherly in the future to eliminate the long line perfectly.

5.3 Detail reduction and edge enhancement

Bilateral smoothing and edge enhancement are significant steps which can make the image looks like a hand drawn image:

Figure 10: Intermediate results of bilateral smoothing and edge enhancement

According to Figure 9, we can find that the images have already looked like a hand drawn image with only smoothing and edge enhancement.

5.4 color modulation

This is my color template image:

Figure 11: movie still from ***Your name***

This is my color transfer result:

Figure 12: color transfer result;(a) Original image1; (b) color transfer result of 1; (c) original image2; (d)color transfer result of 2

Compare to the original images, the color transferred images have brighter and lighter color. This is the cartoon style that I want, but there are also some drawbacks. As I didn't want to change the key color information, the hue value of image had not been transferred. However, cartoon usually have their own paint rules and we may cannot achieve that style without changing the hue information. I should work on using lαβ color space to do transfer and figuring out its computer numerical computing problem in the future.

5.6 Sky fusion

This is an example sky image that I used for testing:

Figure 13: Cartoon sky

This is some zoomed in images:

Figure 14: Fusion results in connection part

We can find that the passion seamless can achieve a very natural transition between original image and sky. However, the drawback here is that some of the ground objects could be fused with the sky, just like image 2 in figure 13. That's because we resize the sky into a box region and there may be some overlapping area between sky and ground. If I only use sky mask and directly paste the sky on the original image, the problem could be figured out, but the transition part in the result will be very unnatural. Thus, we have a tradeoff on this problem.

6. Final results

Figure 15: Good results of my camera filter

Figure 14 shows some good results of my camera filter. Those results look good because the original landscape pictures have regular blue sky and regular building shape.

Figure 16: Bad results of my camera filter

Figure 15 shows some bad results whose sky were not well detected. Some of the buildings were even superimposed by the fake sky. I think for images in the first row, the failures were because my threshold cannot detect the dark blue sky and pure black sky. This is the limitation of color thresholding method.

Figure 17: Bad results of my camera filter

Figure 16 shows the result which were successfully processed, but the result didn't show a cartoon effect. I think the ocean's dark color cannot be changed by my color transfer method. Generally, the ocean in Japanese anime is in light blue, but this ocean is in dark green.

I took a survey to evaluate my results, there are 12 different kinds of results and 7 evaluation questions for each of them. So far, there are 77 people finished my survey and the overall response of my result is good. I will put the electronic format of my survey and my analysis of the survey in the Dropbox folder. I will also divide all of my result into 3 folders: Good, Median, Bad, and put them in the Dropbox folder.

7. Future work

If I have more time, I will do following works:

First of all, I want to find a method that can perfectly eliminate the long line on the boundary between sky and ground without affecting other edges.

Secondly, I want to find a good color mapping method that can not only transfer the brightness and saturation of image, but also change the color without creating some weird color block in the output. Actually, if I can do this project again, I may spend more time on figure out the lαβ space computing problem, instead of using HSV space to transfer. Color is a significant part to make the result looks like a true cartoon. As Erik Reinhard et al.'s [5] work showing, their method can perfectly transfer the color. Therefore, I need to modify some RGB to lαβ transfer procedure to make the α, β value higher. After that, it will be easily numerically computed.

Thirdly, there are many sky detection methods without using color thresholding. These methods could be very complex in mathematics, but universal in dealing with different sky condition.

Besides, I suppose my camera filter could be more intelligent, which means it can detect the condition of the picture (e.g. day or night, cloudy or sunny) and matching different kinds of sky onto the picture. The detection could be realized by analyzing the color and brightness of the picture, instead of using machine learning. In addition, I can use feature detection based on SIFT to detect some significant objects (like the landmarks) in the picture and remain more details on those objects.

References

[1] Chen, Yang, Yu-Kun Lai, and Yong-Jin Liu. "Cartoongan: Generative adversarial networks for photo cartoonization." *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 2018.

[2] Hertzmann, Aaron, et al. "Image analogies." *Proceedings of the 28th annual conference on Computer graphics and interactive techniques*. ACM, 2001.

[3] DeCarlo, Doug, and Anthony Santella. "Stylization and abstraction of photographs." *ACM transactions on graphics (TOG)*. Vol. 21. No. 3. ACM, 2002.

[4] Schmitt, F., & Priese, L. "Sky Detection in CSC-segmented Color Images". In *VISAPP (2)* (pp. 101-106).2009.

[5] Reinhard, Erik, et al. "Color transfer between images." *IEEE Computer graphics and applications* 21.5 (2001): 34-41.