LCAV
- AUDIOVISUAL COMMUNICATIONS LABORATORY

ÉCOLE POLYTECHNIQUE
FÉDÉRALE DE LAUSANNE

Semester Project Report (2020–2021)

Automatic Flat Coloring

Shuo Wen

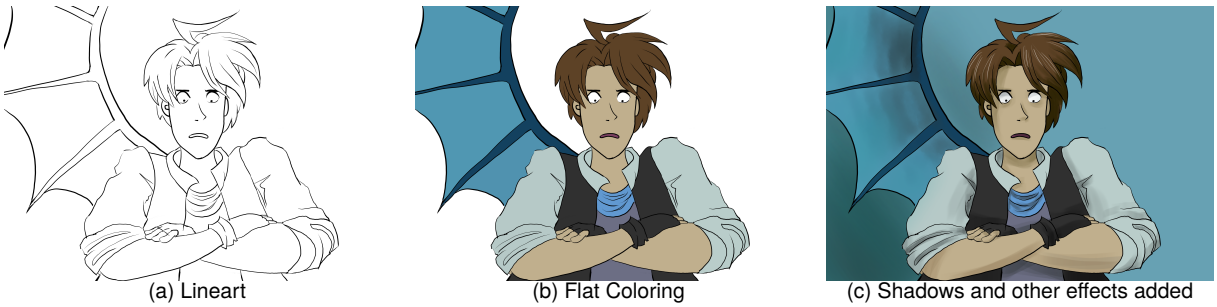
January 15, 2021

Student:	Shuo Wen
School:	I&C
Program:	Computer Science
Cycle:	M.Sc
Semester:	3rd
Supervisors:	Michalina Pacholska, Krzysztof Lis, Matthieu Simeoni

Contents

1 Introduction

Fig.1 shows a typical comic making process: the artists create line-arts for every panel of the comic, then create so-called flat colors, and finally add shading and other effects. There are specialists, called flatters whose job is exactly to prepare high-quality color images. However, small projects and independent artists cannot afford to hire a flatter. This creates a need for automatic coloring of lineart.



(a) Lineart (b) Flat Coloring (c) Shadows and other effects added

Figure 1: Comic making process

From Fig.1(b) shows that flat coloring is to assign each region of the lineart a color. Thus, there are two main steps of flat coloring:

(1) Divide the lineart into regions.

(2) Assign each region a color.

For dividing the lineart into regions, we only need to close the lines in the lineart. Fortunately, there is already an algorithm[1] related to this task and the algorithm is implemented in GMIC[2] (function *fx_colorize_lineart_smart*).

For assigning colors for the regions, we need to first assign each pixel a color, and then determine the color of each region according to the color of the pixels within it. There are many simple strategies to determine the color of each region, so the heart of this step is to assign each pixel a color, that is, to get a color map.

We prefer to use deep learning to get the color map, and there are mainly two network structures. The first one is Image-to-Image network, which is shown in Fig.2. This method is the most direct way of generation. However, the results obtained by the generator completely depend on the information obtained during training, which means the generator can only color the linearts which are similar to the ones in training set.

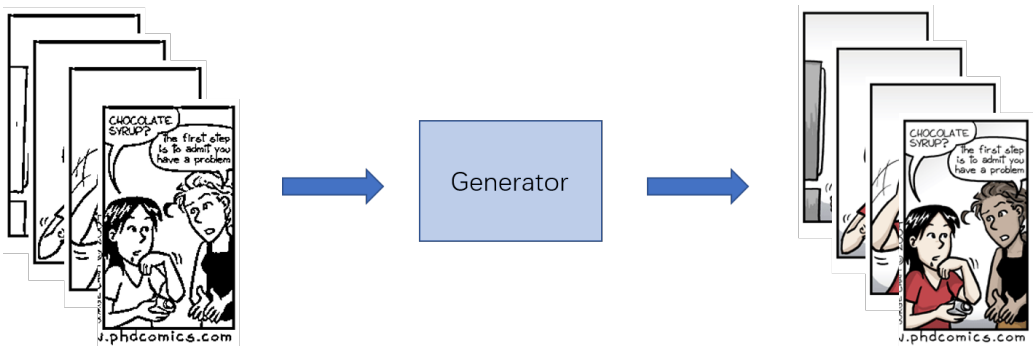


Figure 2: Image-to-image network

The second method is style transfer, whose architecture is shown in Fig.3. The advantage of this structure is the generator obtain the results not only depending on the information from the training

set, but also on the information from the reference image. This makes it possible that the generator can color all kinds of linearts, even if the ones that are totally different from the linearts in training set, as long as the related reference images are provided.

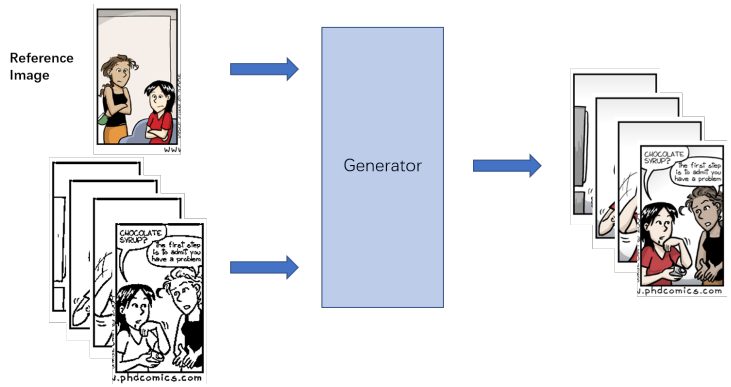


Figure 3: Style transfer network

In this project, we have tried both of the two methods, and finally choose the Image-to-Image network. More details will be provided in Sec.3.

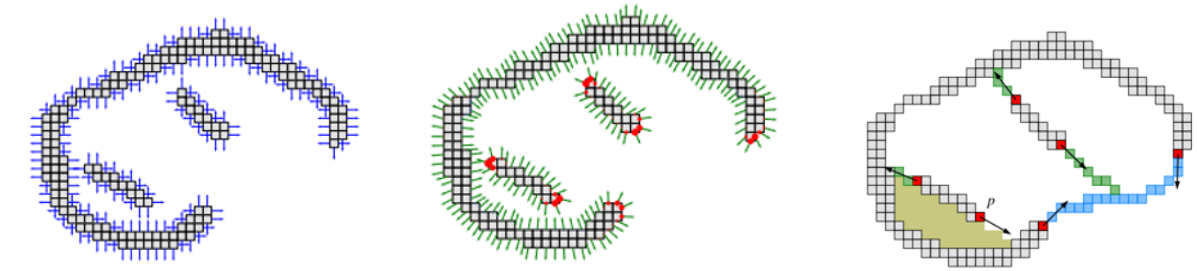
The contributions of this work are threefold. First, we create three datasets for training the flat coloring model. Second, based on attention U-net, we introduce a generator which uses mixing colors to generate colored images. Third, combining the coloring model and the algorithm of line closing, we propose a pipeline which can colorize linearts with flat colors.

In the following sections we will first explain two existing methods: one is the algorithm which gets the regions from the linearts, and another is Attentioned Deep Paint, a model of color transfer. Then we will show the steps of obtaining our coloring model, which include dataset creation, and three main stages of getting the final model. After that, we will show how we obtain the flat color image from the color image generated by our generator.

2 Existing methods

2.1 Closure of the strokes

As we discuss before, the first step of flat coloring is to divide the linearts into regions, and for the linearts we only need to close the lines. The stroke closing algorithm in [1] performs this task. Fig.4 shows the three main steps of this line closing algorithm.



(a) Step 1 (b) Step 2 (c) Step 3

Figure 4: Closure of the lines processing[1]

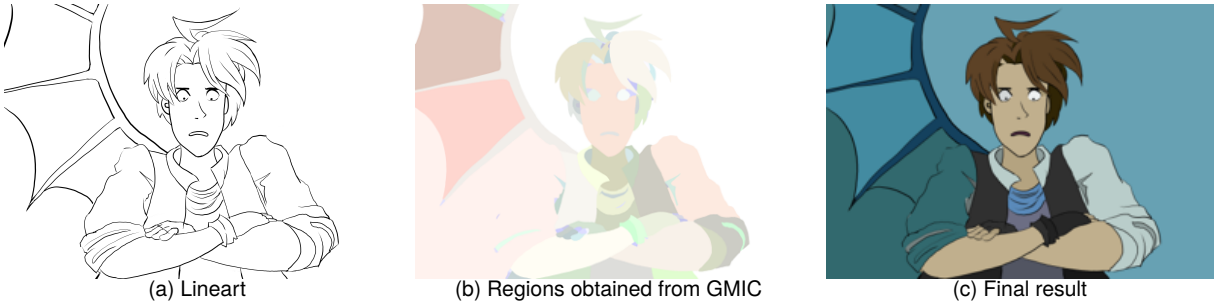
The three main steps of the line closing algorithm[1] are:

(1) Calculate the normals of the line-art.

(2) Calculate the curvature from the normals and set the points with large curvature as the key points.

(3) Line closing using splines (in blue) and straight segments (in green).

Using GMIC[2], an example of the above-mentioned algorithm is obtained and shown in Fig.5.



(a) Lineart (b) Regions obtained from GMIC (c) Final result

Figure 5: Result of line closing using GMIC

From Fig.5 we can find two main problems:

(1) As shown in the red part of Fig.6, the algorithm did not close same lines that artist would. As a result, the several connected regions are assigned to be the same color.

(2) As shown in the blue part of Fig.6, the algorithm sometimes over-divides the regions. This can be corrected at the color assignment step.

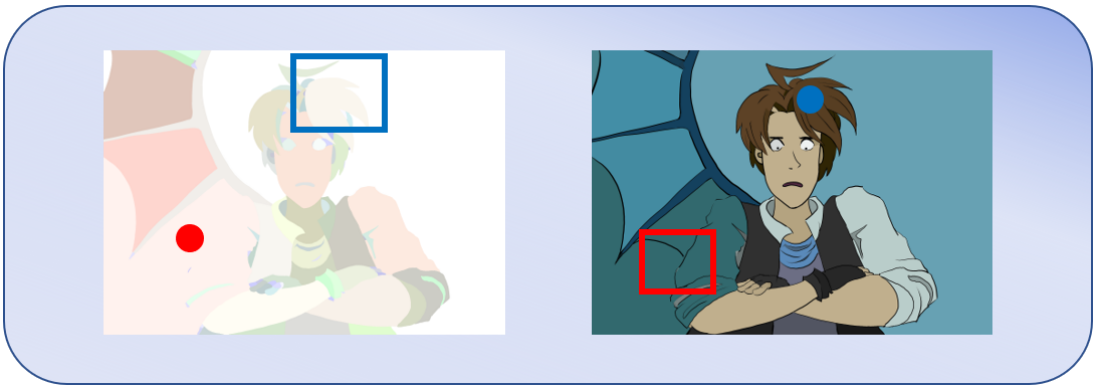


Figure 6: The problem of the line closing algorithm

Despite the above two problems, the results of the algorithm are generally good and can be improved by tuning GMIC parameters such that minimize region size.

2.2 Attentioned Deep Paint

An existing model of color transfer is Attentioned Deep Paint (ADP)[3]. This model is a GAN structure and uses Attentioned U-net as the generator, and the whole pipeline of getting the color image is shown in Fig.7: first extracting colorgram from the reference image, and then inputting both the colorgram and the lineart into the network and getting the result.

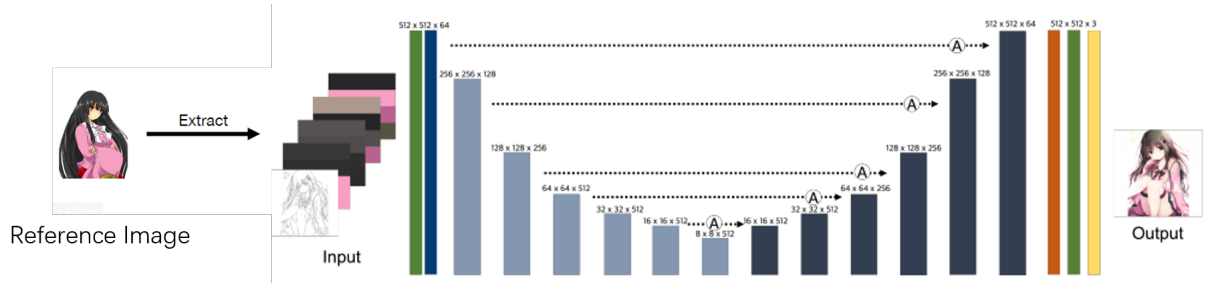


Figure 7: The generator structure of Attentioned Deep Paint[3]

The loss function this model uses is

$$L = \arg\min_G \max_D L_{GAN}(G,D) + \lambda L_1(G) \quad ,$$

where $L_{GAN}(G,D)$ is the standard GAN loss and $L_1(G)$ is the standard l_1 reconstruction loss.

The results generated by ADP is shown in Fig.8. We can see that given a sketch (left one of each group), the model can generate a colored image (right one of each group) according to the reference (middle one of each group). Also we can see that the dataset ADP uses is characters only, using sketches as lines, and using colored images with shadow and other effects.

Figure 8: Result of training on Attentioned Deep Paint dataset

3 Coloring Model

3.1 Dataset Creation

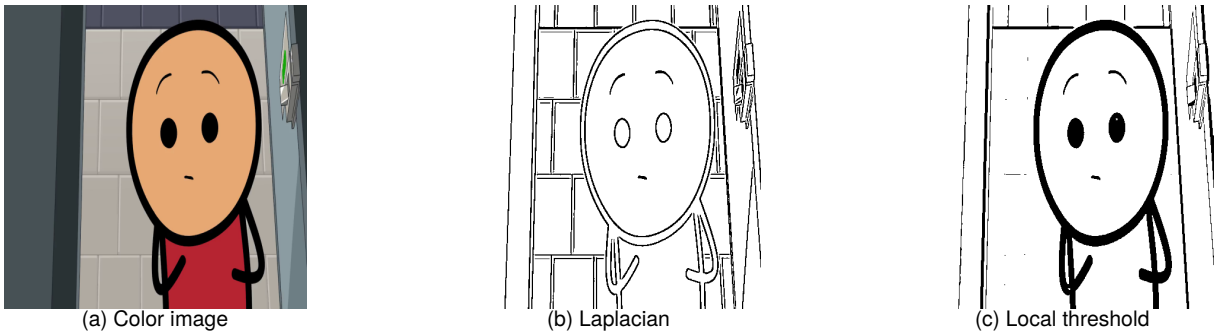
We want to train a network to transform linearts into colored images. This requires a training set of matched pairs of lineart and colored images. Currently existing datasets are all like the Dataset of Attentioned Deep Paint (ADP). They cannot be used for our task for two reasons, shown also in Tab.9:

(1) The line style of the current datasets is sketch, while we need lineart. From Tab.9 we see that sketches use the density of lines to express the image, which makes them more complicated in the use of lines. Compared to sketches, linearts are more clear with thick lines.

(2) The color style of the current datasets is color with shadow and other effects, while we need only flat color.

Figure 9: Comparison of the Dataset from ADP and what we need

To create the datasets we use the online comic "Phd Comic"[4]. And we also extract frames from video "Snoopy"[5] and "Explosm"[6]. We chose those works of art because they have relatively little shading and other effects. We extract the linearts form the images by applying local thresholding or Laplacian (gradient detection). An example of lineart extraction is shown in Fig.10.



(a) Color image (b) Laplacian (c) Local threshold

Figure 10: Lineart Extraction

As we can see in Fig.10, there is a problem for each method: the problem of Laplacian is it will extract two lines for a thick line; the problem of local threshold is it will keep all pixels whose colors are nearly black, for example, the eyes of the person in Fig.10. Since the linearts are closer to the results obtained from local thresholding, we use the local thresholding to get the lineart. We assume that, for example, the eyes of the person are originally black in the linearts.

The examples and the properties of the three datasets we have created are shown in Fig.11.

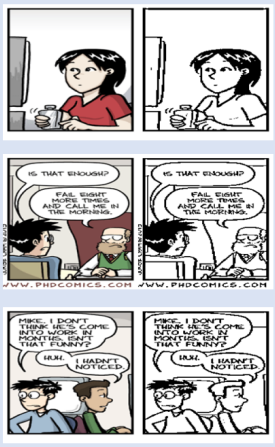
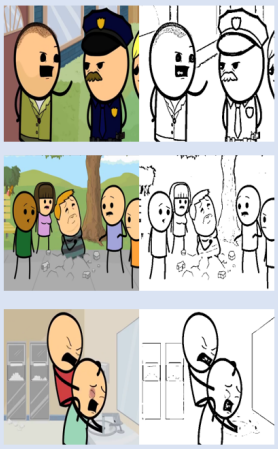
	Phd Comic	Snoopy	Explosm
Examples			
Properties	1. Few colors 2. Enough images 3. Highly repetitive roles	1. Many colors 2. Enough images	1. Many flat colors 2. Limited number of images

Figure 11: Datasets Created

3.2 Newly created dataset on ADP

After creating the datasets, we can train the model. We use the existing model Attentioned Deep Paint as our basic model, and train it on our datasets to see what will happen.

We train this model on Phd comic for 50 epochs, and the result after training 30 epochs are shown in Fig.12. The top two rows of Fig.12 shows that we get a great result of coloring. The most exciting finding is it seems from the last row that the model really transfers the color instead of ignoring the reference image!

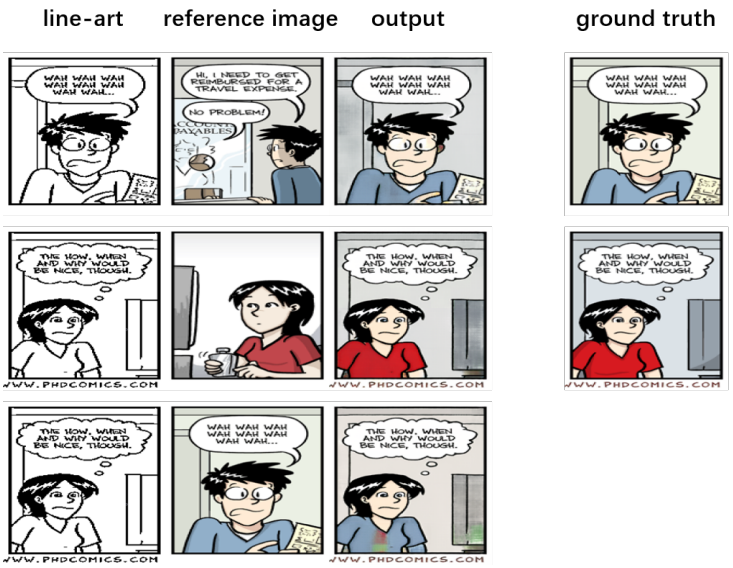


Figure 12: Result of training 30 epochs on Phd comic dataset

However, it is not the case.

After training 50 epochs, we get the the result shown in Fig.13. The result shows that the network does not transfer the color anymore.

Figure 13: Example result of training 50 epochs on Phd comic dataset

A possible explanation of the above result is the network is overfitting to the training set and finds that it can colorize the lineart without the help of reference images. To overcome this problem, we expand the dataset by exchange the RGB channels. An example result is shown in Fig.14. We expect that the network will stop ignoring the reference images since the same object can be colorized in different colors.

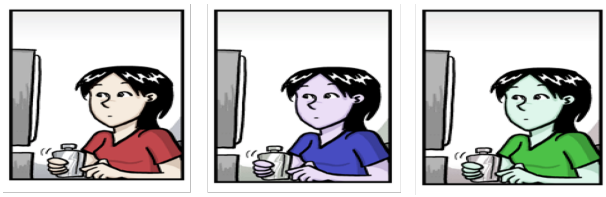


Figure 14: Expanding of the dataset

Then, something interesting happens. After training the model on the expanded dataset, we get the results as shown in Fig.15.

Figure 15: Result of training on the expanded dataset

From Fig.15 we can see that it seems there are three ways for the network to colorize the lineart according to the RGB, BGR, and GBR images from the training set, and the reference images only decide which way the network will choose rather than transferring the color. From this finding, a possible explanation of the result in Fig.12 and Fig.13 is: when training 30 epochs, the network finds two ways to colorize the girl's T-shirt (blue and red), and the reference image makes the network colorize the girl's T-shirt to blue. However, the network never transfers the color; when training 50 epochs, there is only one way to colorize the girl's T-shirt (only red), then whatever that reference image is, the network will colorize the girl's T-shirt to red.

3.3 Mixing Colors

From the previous result we find that the structure of ADP does not transfer colors when trained on Phd comic dataset. One explanation is: even if we input both the linearts and reference images, the generator can decide how to use them, which means ignoring the reference images is also a possible choice. Thus, to force the network using the color information of the reference image, we design the mixing colors network structure as shown in Fig.16.

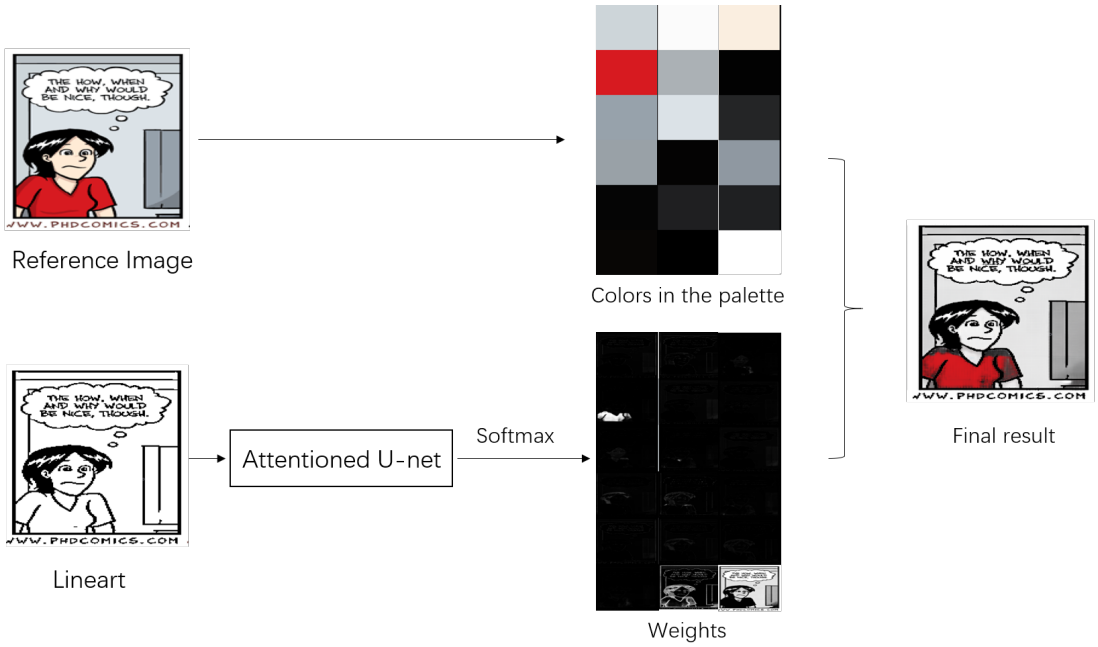


Figure 16: Newly created network structure

From Fig.16 we can see that in this structure, we first extract a color palette from the reference image. Note that we add black and white to every palette to adjust the brightness. For the part of Attentiond U-net, we only input the linearts, and then using Softmax instead of Relu as the activation function for the last layer we get weights of the colors extracted from the reference image. Finally, we use the weights to mix the colors in the palette and get the final result. However, the results are not what we expected. Most of the results are are grey level images, as shown in Fig.17.

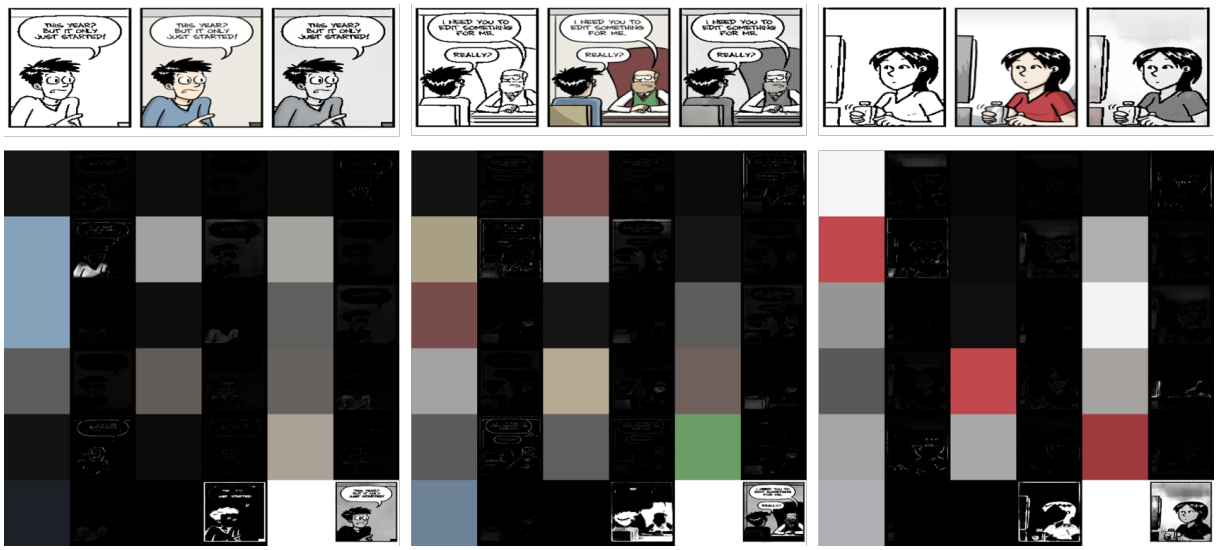


Figure 17: Example results of the newly created network

After observing the weights of the colors, we find that most of the weights are concentrated on black and white in the palette, which leads to the result that the outputs are grey level images. A possible explanation is that it is too hard for the network to learn how to use the palette whose colors change with the reference images, so the network tends to assign more weights on the reliable colors. (A reliable color means it always exist in the palette.) In our model, the reliable colors are black and white, so as a result, the generator always mix black and white to colorize the image and make the output grey level.

3.4 Global Color Palette

To make the generator use more colors other than black and white, we want to get more reliable colors for the reference images. Thus, we change the model to the one shown in Fig.18. In this structure, we extract a global color palette, which contains eight reliable colors, from each dataset instead of extracting color palette from each reference image.

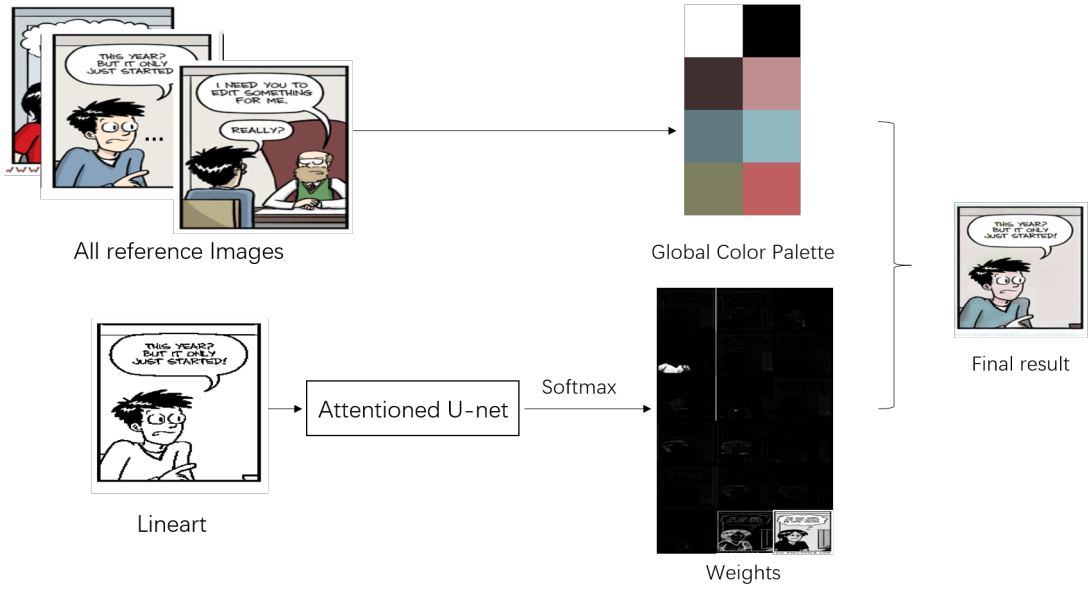


Figure 18: Newly created network structure with Global Color Palette

Fig.18 shows that instead of extracting a palette from each reference, we extract a global color palette from the set of all the reference images. The general steps are first divide the HSV space into n bins, and then for each pixel in the reference images put it into a bin, and finally take the top-8 bins which contain the largest number of pixels as the colors in the palette. Note that for the colorful datasets, it might be hard to extract the main colors, and under that condition we will use the 8 saturate colors to be the global color palette.

The newly created network structure with global color palette works very well and the results will be shown in Sec.4.

Another thing we should notice is: the newly created network structure with global color palette is no longer a style transfer model, and instead, it becomes a Image-to-Image model. However, it still has advantages compared to the original Attetioned Deep Paint model (since it ignores the reference image, it can be also seemed as an Image-to-Image model). The biggest advantage is we can change the color of the colorized image by manually change the colors in the global color palette, which will also be shown in Sec.4.

3.5 Flat coloring

Here we briefly introduce how we get a flat color image from a color image.

From Sec.2.1 we know that given a lineart we can obtain an image which is divided into regions. To assign each region a color, we need to aggregate the colors from the pixels belong to this region. There are three basic ways to aggregate the color: taking the mean, taking the median, and taking the mode.

Mean is easily influenced by the outliers, for example, black pixels. Mode is actually what we want since we aim to choose the dominant color for each region. However, in practice the dominant color are multiple colors which have very close RGB value, which means it is hard to take the mode in practice. Taking the median is more practical, and when the pixel number of dominant color is more than half, taking the median is equal to taking the mode. Thus, taking the median is what we use.

In short, to assign each region a color, we will first find the pixels belong to this region, and then take the median of RGB value of the pixels.

4 Results

4.1 Color images generated by our model

The color image generated by our model is shown in Fig.19 and Fig.20. Note that we train a generator for each of the datasets since our model is now an Image-to-Image structure.

Fig.19 is the result of Phd Comic dataset. From the global palette we can see the eight colors we extracted and their corresponding weights.

Figure 19: Color images generated by our model trained on Phd Comic

Fig.20 is the result of Snoopy dataset. Since this dataset is colorful and hard to extract the main colors, we use the eight saturated colors to be the global palette.

Figure 20: Color images generated by our model trained on Snoopy

4.2 Manually color changing

An big advantage of our color mixing model is that we can change the colors in the palette to colorize the lineart in different way.

One way is to change the colors in the global palette in our model (change in the python program). Since the model has been already trained, it knows how to mix the eight colors in the palette and change the colors will not influence the model parameters. An example is shown in Fig.21. From the example we can see that after swapping the two colors in the bottom right of the palette, the color of the boy's shirt changes accordingly.

Figure 21: Example of changing the colors in the global palette of our model

Also, we made a GUI to change the colors, which is more user friendly. The GUI is shown in Fig.22. We can use it as the following:

(1) Select a lineart from computer.

(2) Run the model on the lineart and get the result (On the right side of the lineart). If we just want to colorize the lineart in it origin style, we are done.

(3) If we want to change the colors, we should get the weights for the colors.

(4) From the weights shown next to each color, we can know which colors we want to change. Then we can select colors as we want, and the newly colored image will be automatically shown in the top left.

(5) Save it when we get a satisfying result.

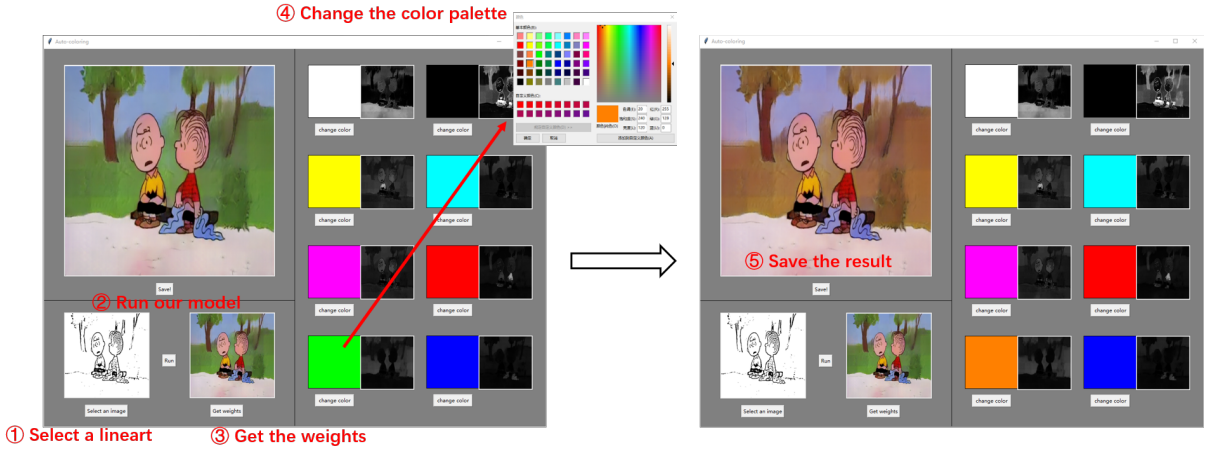


Figure 22: Explanation of GUI used to change the colors

4.3 Flat coloring

Here we show the result of the whole pipeline, that is: generate color images from the linearts, and then divide the images into regions based on the linearts, and finally assign each region a color and get the flat color image. The result examples are shown in Fig.23.

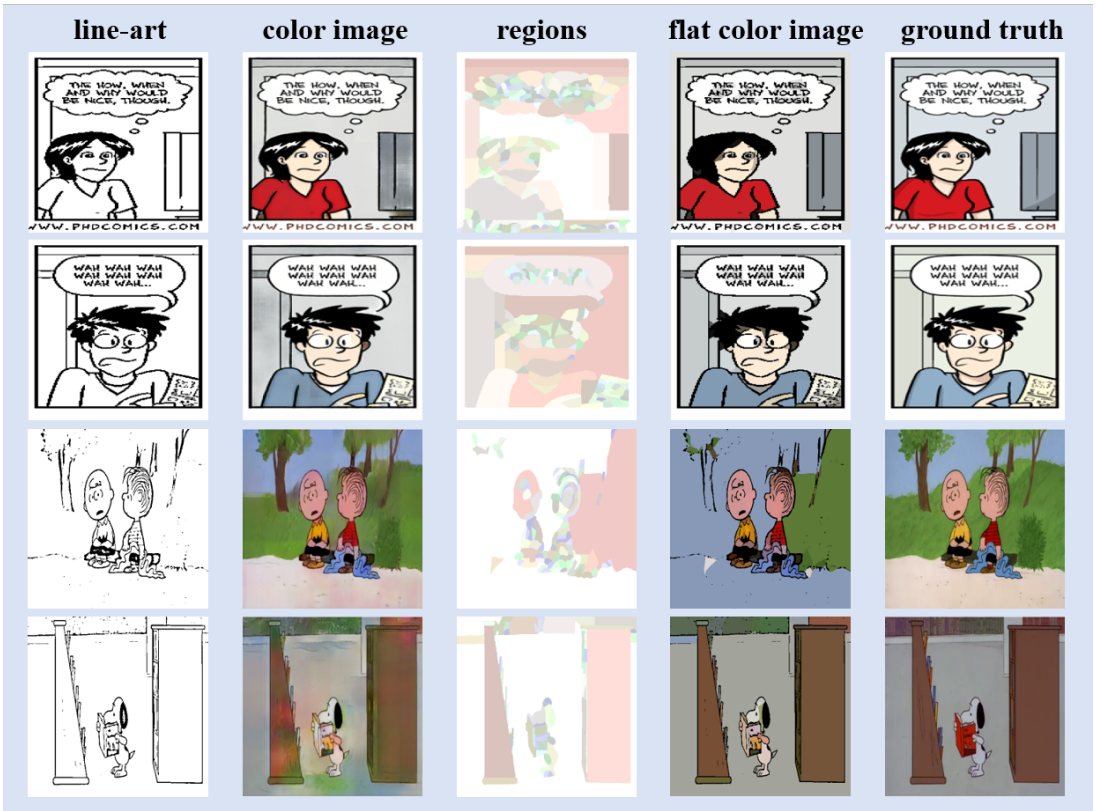


Figure 23: Result of flat coloring

From the result, we can see that the flat coloring step sometimes ruins the image when the regions divided by GMIC are far from what we expected like the third example. This might be improved by making a better lineart and adapting the parameter of the line closing algorithm. Also, the flat coloring step brings great improvement sometimes like the last example. When the generated color image contains a lot of noise, flat coloring can reduce the noise and make the result more smooth and clear.

5 Conclusion and Future Work

In this work we created three datasets for training the flat coloring model. And we introduced a generator which uses mixing colors and global color palette to generate colored images. A big advantage of this model is we can change the colors in the palette to colorize the lineart in different way, and we made a GUI for this. Also we proposed a flat coloring pipeline based the coloring model and the algorithm of line closing. In the future, we will try other style transfer model, for example, embedding model, to make a generator which is able to color all kinds of linearts.

References

[1] S. Fourey, D. Tschumperlé, and D. Revoy, "A Fast and Efficient Semi-guided Algorithm for Flat Coloring Line-arts," Oct. 2018. [Online]. Available: https://hal.archives-ouvertes.fr/hal-01891876 1, 2, 3

[2] D. Tschumperlé and S. Fourey, "G'mic (greyc's magic for image computing): A full-featured open-source framework for image processing." [Online]. Available: https://gmic.eu 1, 3

[3] T. Kim, H. Oh, D. Lee, and H. Kang, "Attentioned deep paint." [Online]. Available: https://github.com/ktaebum/AttentionedDeepPaint 3, 4

[4] J. Cham, "Piled higher and deeper," http://phdcomics.com, accessed: 2021-01-15. 5

[5] Peanuts, "The charlie brown and snoopy show," https://www.youtube.com/user/Snoopy. 5

[6] Cyanide and Happiness, "Explosm," https://www.youtube.com/user/ExplosmEntertainment. 5