Image Colorization Using Deep Learning Methods

Tengda Han
Australian National University
tengda.han@anu.edu.au

Abstract

Image colorization is an interesting topic in image-to-image translation. Nowadays, many cameras are still capturing greyscale images, like surveillance cameras and satellite cameras. With colorization techniques, we can recover plausible colored images for safety and research purpose. Besides, image colorization techniques can help us to colorize legacy images and videos to better understand the history. In this paper, I implement 3 types of image colorization frameworks based on deep learning methods. I verify their colorization performance on 4 different datasets, and I find Generative Adversarial Network is a powerful model to produce plausible colored images. Additionally, I summarize some future research directions about image colorization.

1. Introduction

Image colorization is a fairly broad topic. In this project, I define image colorization to be the process of transferring greyscale images into colored images. In modern computer graphics, a colored image is represented as a 3-channel data array. Generally, greyscale image can be represented as a 1-channel data array. Thus, within the scope of this project, the objective of image colorization is to learn a mapping function $y = f(x)$, which maps 1-channel data array x into 3-channel data array y, and produces plausible and visually pleasant results.

In the following sections, I will briefly explain the basic knowledge about colorful image representation, including RGB color space and Lab color space. Then in Section 2, I will introduce my methods. My experimental results and discussions are shown in Section 3. Finally I summarize future research directions and conclude this project in Section 4 and 5.

1.1. RGB Color Space

RGB color space is a very common representation of colorful images in computer graphics. Specifically, a colorful image is represented by a 3-channel array. Each channel represents the chromaticity of Red, Green and Blue respectively. Chromaticity means the quality of the color regardless of its luminance. Red, green and blue are the additive primary colors. *RGB* color map is capable to any chromaticity defined by these three primary colors. In computer graphics, every data point in every channel can take the value from 0 to 255, normally as integers. If this value is closer to 0, the color of the corresponding channel looks darker. Several examples include $\text{Black} : [R, G, B] = [0, 0, 0]$, $\text{White} : [R, G, B] = [255, 255, 255]$, and $\text{Red} : [R, G, B] = [255, 255, 255]$.

1.2. Lab Color Space

Lab color space is another famous color space. There are several variants of *Lab* color space. The *Lab* in this paper refers to CIE *Lab* color space. In *Lab* color space, a colorful image is also represented as 3-channel data array. Comparing with *RGB*, the difference is the meaning of each channel. The first channel *L* in *Lab* color space represents lightness, which is a black/white color intensity. It represents black at $L = 0$, and white at $L = 100$, and takes value from $[0, 100]$. The second channel $a \in [-128, 127]$ is the red/green channel, with green at negative *a* values and red at positive *a* values. The third channel $b \in [-128, 127]$ is the yellow/blue channel, with blue at negative values and yellow at positive values. When $a = 0$ or $b = 0$, both channels represent neutral grey

color.

Lab and *RGB* color spaces are inter-changeable using some non-linear transformation, which is commonly available in image processing packages. A visualization for *L*, *a* and *b* channel is shown in figure 1.

Figure 1. A visualization of *Lab* channels for a colored image. Published by FloydHub.

2. Method

In this section, I will introduce three image colorization models: Simple ConvNet, ConvNet for classification, and Conditional Generative Adversarial Networks (GAN).

2.1. Simple ConvNet

As alluded above, the objective of image colorization is to learn a mapping function from 1-channel images into 3-channel images. *Lab* color space shows its advantages in this situation. In *Lab* color space, the lightness channel *L* can be regarded as a greyscale image, and can be directly used for neural network training. With this configuration, the mapping function only needs to learn to generate 2 channels, *a* and *b*. And the original lightness channel can be re-used to generate colorful images. By re-using lightness channel, most of the features from input images can be retained in the output, such that the output can be visually 'closer' with the input image, and will not change the input image too much. Additionally, generating only 2 channels simplify the tasks, and will make it easier to train the neural network. In contrast, if we colorize directly on *RGB* color space, first we have to obtain a reliable greyscale version of the raw image; second the model has to generate all three channels for each color respectively. With these reasons, using *Lab* color space appears to be more convenient. This convenient first-consideration of *Lab* color space is also mentioned by [9], [4], and [1].

Figure 2. A simple pipeline to colorize *Lab* images.

With the analysis above, a basic pipeline is shown in Figure 2. For each *RGB* image in the training set, first I convert it into *Lab* color space. Second, the *L* channel is fed into the model as a greyscale image input. Then the model is trained to produce a 2-channel image with same size, which represent the prediction for *a* and *b* channel. Finally, I compare the prediction with ground-truth *ab* channels by a loss function, and the loss is back-propagated throughout the model.

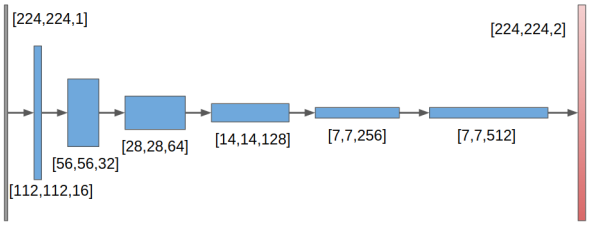

Figure 3. The architecture of the simple ConvNet model.

I implement a fairly simple convolutional neural network to learn the mapping function $f()$, which has an encoder-decoder scheme. The encoder part shrinks the height and width, and enlarge the depth of the feature map; in contrast, the decoder part enlarges the height and width, and shrinks the depth of the feature map. In this simple model, there are 6 encoder blocks and only 1 decoder block. Specifically, in the encoder, max-pooling layers with stride 2 are used to shrink the dimension by 2 at each step. In the decoder, a transpose-convolutional layers with stride 2 are used to expand the dimension by 2 at each step. Each encoder block contains a convolutional layer, a ReLU activation function, and a max-pooling layer; The decoder block contains a transpose-convolutional layer and a ReLU activation function. The height and width of the final output are the same as the input image's, such that they can be directly compared by a loss function.

For loss function, I choose *L2* distance, specifically,

$$L_2(\hat{\mathbf{Y}}, \mathbf{Y}) = \frac{1}{2} \sum_{h,w} \|\hat{\mathbf{Y}}_{h,w} - \mathbf{Y}_{h,w}\|_2^2 \qquad (1)$$

where the ground truth *ab* channel $\mathbf{Y}_{h,w}$ and the predicted *ab* channel $\hat{\mathbf{Y}}_{h,w}$ has the same dimension, $\mathbf{Y}_{h,w}, \hat{\mathbf{Y}}_{h,w} \in \mathbb{R}^{h \times w \times 2}$.

However, the Euclidean loss like *L1* and *L2* tend to average the color when the pixels have more than one possible plausible colors. Therefore, it will lead to brownish or greyish color in the output [9] [4] [1]. For instance, apple may be red or green. If the dataset contains both red and green apples, the trained model will tend to generate brown apple, which is roughly the average of red and green color. And I do find similar situation in my experiments in Section 3.

2.2. ConvNet for classification

The simple ConvNet model shown above treats image colorization problem as a regression problem, which is proven to generate brownish outputs (See Section 3). In order to avoid averaged colors, one possible solution to encourage high variety is to replace regression by classification. Here comes the first question: what is the ground truth for classification problem?

For *Lab* color space, channel *a* and *b* take continuous values, which makes direct classification impossible. Zhang *et al*. [9] proposed a method to classify color by quantizing gamut into small squares. Gamut is the collection of all visible colors in a color space. For example, the gamut of *Lab* color space looks like the left figure in Figure 4. Normally, gamut is an irregular solid region. For *Lab* color space, gamut lies in a 3-D space defined by *L*, *a* and *b* respectively. All the colors within this solid region can be represented by *Lab* color space. Specifically, Zhang *et al*. [9] quantize *ab* channels by small squares. Each square spans over 10×10 values on *ab* axis, and the color within each square keeps the same. In this way, they quantize 313 distinct colors on *ab* channels. And then a 313-class classifier can assign plausible color to *ab* channels, and thus gives colorization result. In my project, I try this classification configuration to verify the model performance.

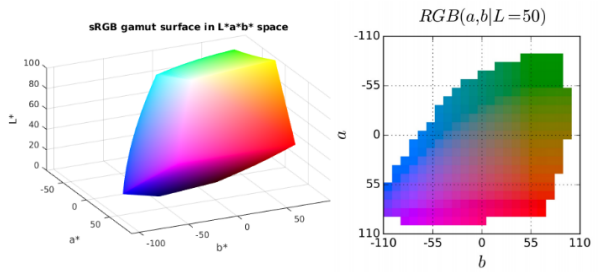

Figure 4. Left: 3D view of *Lab* color space gamut; Right: Visualization of quantized gamut when $L = 100$ from Zhang *et al*. [9]

Cross-Entropy loss is the common loss function for classification problems. In this situation,

$$L_{cl}(\hat{\mathbf{Z}}, \mathbf{Z}) = -\sum_{q} \mathbf{Z}_{h,w,q} \log\left(\hat{\mathbf{Z}}_{h,w,q}\right) \qquad (2)$$

where the ground truth classes $\mathbf{Z}_{h,w,q}$ and predicted classes $\hat{\mathbf{Z}}_{h,w,q}$ have the same dimension, and $\mathbf{Z}_{h,w,q}, \hat{\mathbf{Z}}_{h,w,q} \in [0,1]^{h,w,q}$, where the number of quantized color $q = 313$.

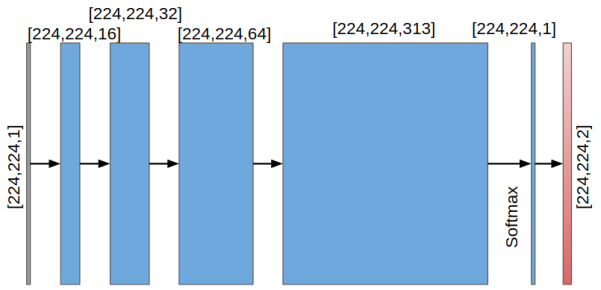

Figure 5. The architecture of the ConvNet model for classification

The architecture of this ConvNet model is different with previous one. According to [9], the height and width of the feature maps from hidden layers are not changed. This model can be regarded as a pure decoder. I use convolutional layer and ReLU activation function in each block. The final output of the model is a 313-channel image containing the probability distribution of possible colors for each pixel. Then the cross-entropy loss is computed and back-propagated throughout the network. Importantly, due to the large depth (313) and the unchanged height and width (224×224), this model requires a high amount of memory usage. Thus, during the training process, the batch size is strictly limited.

2.3. Conditional GAN

Generative Adversarial Network(GAN) proposed by Goodfellow*et al*. [2] is a famous generative model. The traditional GAN has two parts: a generator (G) and a discriminator (D). For image generation, the generator is trained to generate images from noise to fool the discriminator. And the discriminator is trained to discriminate whether a image is real (from dataset) or fake (generated by G). To handle image colorization problem, we modify the traditional GAN into a Conditional GAN [5] to generate images from input images, instead of from noise.

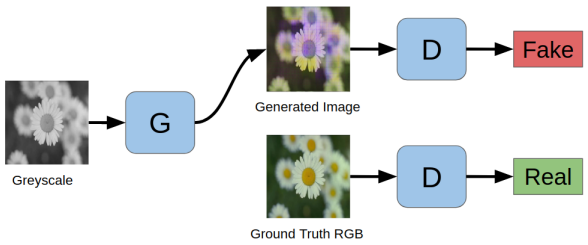

Figure 6. The concept of Conditional GAN for image colorization, where 'G' stands for Generator and 'D' stands for Discriminator.

Specifically, the generator takes greyscale images (lightness channel from *Lab* color space) as inputs, and generates colorized 3-channel *RGB* images. The discriminator is trained both on ground-truth *RGB* images and the generated images, to determine whether the given image is generated. The overall objective for this Conditional GAN is a min-max game:

$$\min_{\theta_g}\max_{\theta_d}\Big[\mathbb{E}_{y\sim p_{data}}\log D_{\theta_d}(y) + \mathbb{E}_{x\sim p_{data}}\log\left(1-D_{\theta_d}(G_{\theta_g}(x))\right)\Big] \tag{3}$$

where x represents the greyscale image and y represents the colored image. θ_g and θ_d are the parameters for generator G and discriminator D respectively. The objective of generator is to minimize the function; and the objective of discriminator is to maximize the same function. That is the adversarial identity of GAN.

In order to constrain the modification made by GAN, an extra regularization term is added to the generator. Without this regularization term, GAN can still be trained but the generator will learn to fool the discriminator using some weird patterns which does not look plausible. Specifically, apart from maximizing the existing term, a *L1* distance between generated image and ground truth is added to generator's loss function with a weight λ.

$$\min_{\theta_g}\Big[\mathbb{E}_{x\sim p_{data}}\log\left(1-D_{\theta_d}(G_{\theta_g}(x))\right) + \lambda\|G_{\theta_g}(x)-y\|_1\Big] \tag{4}$$

There are many small tricks to train a stable GAN. For the architecture of generator, we still use a encoder-decoder scheme. But in the decoder layers, we add skip-connections between its mirror encoder layers by simply adding the feature map before moving on to the next layer. This technique is called U-Net [7], and is able to retain the details from the original images much easier. In encoder layers, I use convolutional layers with stride 2 to shrink the height and width. In decoder layers, similarly I use transpose-convolutional layers with stride 2 to expand the height and width. Each block in encoder contains a convolutional layer, a batch normalization layer and a Leaky ReLU activation function. Each block in decoder contains a transpose-convolutional layer, a batch normalization layer and a Leaky ReLU activation except the last one. The last decoder layer has a transpose-convolutional layer and a tanh activation function to gives $[-1,1]$ values in the prediction. The ground truth *RGB* images and input images are also normalized to $[-1,1]$.

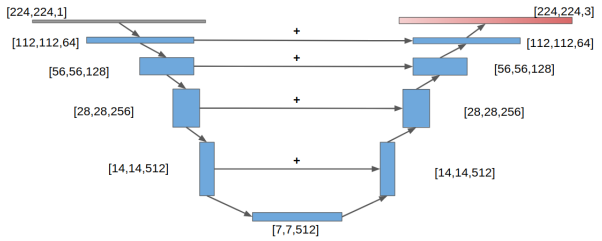

Figure 7. The architecture of generator model in conditional GAN. The left path is encoder, and the right path is decoder. The arrows with plus sign indicates the skip connection between mirror features.

The architecture of discriminator only has the encoder path. And in the final layer discriminator gives a 1-channel label indicating whether the given input is real or fake. Thus in the final layer a sigmoid activation function is used to bound the output to $[0,1]$

range. For other blocks in the encoder path, similarly I use a convolutional layer with stride 2, a batch normalization layer and a Leaky ReLU activation function.

The value for the regularization weight λ is determined empirically.

3. Experimental Results

In this section, I will introduce my experimental results and have some discussions. First, I introduce the datasets used in this project. Then, I show the results for three methods respectively.

3.1. Datasets

Overall, I use four datasets in this project: LFW, OxFlower, SpongeBob and SC2. Two of them are published datasets, and the other two are assembled by myself. For image colorization problem, the choices of dataset are very broad. Because theoretically all colored images can be used as datasets as we can extract the greyscale images from colored images themselves.

Labeled Faces in the Wild [3] (LFW) is a dataset containing human faces for 5749 distinct people spanning over 13000 images. Human faces are aligned in the center by deep funneling algorithm [3]. One characteristic of this dataset is that the color of human faces in this dataset have small variance, which varies from light skin color to brown color. I use their official training-testing split, which contains about 8000 images in the training set.

The second published dataset is the Flower Dataset (OxFlower) from VGG group of Oxford University [6]. The dataset contains flower images over 17 categories like Daisy and Sunflower, and has 80 images for every class. The reason to choose OxFlower dataset is because flowers has a variety of plausible colors like yellow, white, purple, etc. This dataset can report the model performance on datasets with large color variance. I use their official training-testing split, which contains about 1000 images in the training set.

The first dataset (SpongeBob) I collected by myself is from the cartoon television series called SpongeBob SquarePants. The motivation to choose this cartoon TV series is because most of the characters has a constant color throughout the videos, like SpongeBob is always yellow. I download one MP4 video on-line, extract its frames with strides, and do some resizing and random cropping to control the size of images and ensure the images contain some meaningful feature to learn, like a cartoon character. Finally I obtained a small dataset contains about 2000 images. I split $1/5$ of them as a testing set, and keep the rest as a training set.

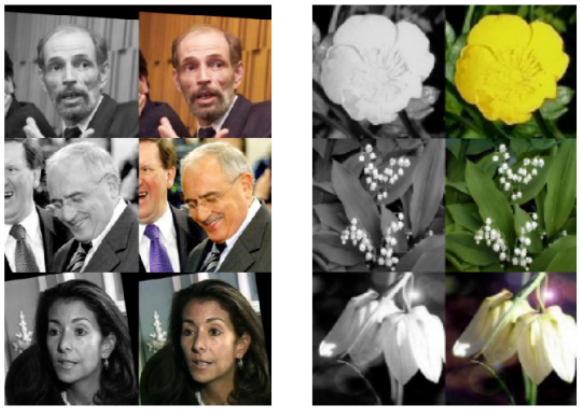

Figure 8. Left: Image samples from LFW dataset; Right: Image samples from OxFlower dataset. In both figure, the left column is greyscale images, the right column is colored images.

The last dataset (SC2) I assembled is from the video game StarCraft2 by Blizzard Company. StarCraft2 is a real-time strategy game recently popular because it is targeted as a research platform for Reinforcement Learning [8]. I download a 1-vs-1 replay video from professional players on-line, extract frames, resize and random crop to obtain a small dataset containing about 1800 images. Similarly, I split $1/5$ of the images as a testing set, and use the rest $4/5$ for training. Similar with SpongeBob, the motivation to choose StarCraft2 replay video is because the colors in StarCraft2 elements (units, buildings and backgrounds) are mostly pre-designed by people. Thus the color distribution in this video game has much less variant comparing with real-world videos. For instance, I choose a game between two different races. One race has golden buildings with green patterns, and some blue crystal-like structures. The other race has darker buildings with red patterns, and the units look more biologically.

3.2. Simple ConvNet

First, I try the simple ConvNet model on LFW dataset. The input images are resized to 224×224 pixels. I set the batch size to be 32, and use Adam optimizer with 10^{-3} learning rate. The training and

Figure 9. Left: Image samples from SpongeBob dataset; Right: Image samples from SC2 dataset. In both figure, the left column is greyscale images, the right column is colored images.

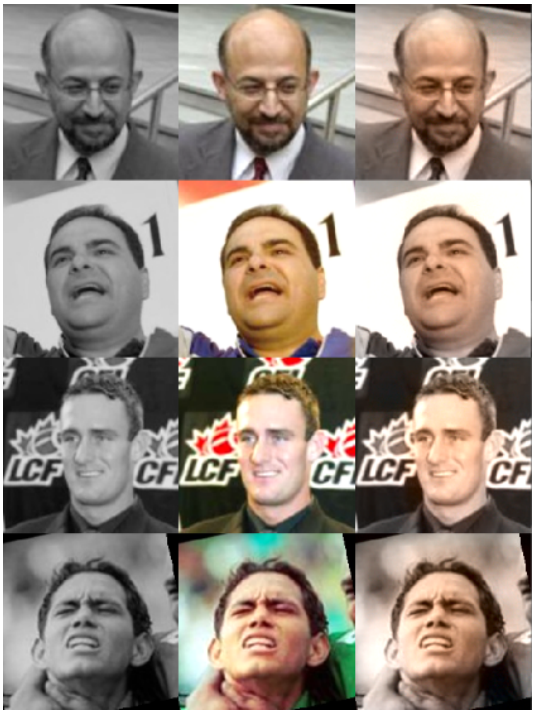

Figure 10. Testing results from Simple ConvNet on LFW dataset. Left column: greyscale images; Middle column: ground-truth images; Right column: colorization results.

validation loss decrease quickly and over-fitting appears quickly. Finally I visualize some results after 10 epochs to avoid severe over-fitting in Figure 10.

From Figure 10, first I observe that the output image looks very close to the input image in terms of the shape and patterns. That is because of the utilization of *L* channel in the output image. Second, I observe that the colorized images are given with averaged brownish color. There should have been some variant among the skin color of these four people, but the model assigns similar brown color to all of them. Third, the model is not capable to learn the color distribution for the background. For example the third row in Figure 10 contains some red icons in the background, however the model generates neutral grey on these regions. That is because the red color in the background is a rare situation in the dataset, thus with *L2* loss, it is much likely to be ignored during evaluation. These phenomena are expected because of the characteristic of *L2* loss function. Generating averaged color can minimize the loss well, but during this process, the desired color variety is lost.

Because OxFlower dataset has less variety on colors. Next, I train the same model on OxFlower dataset with image size 224×224. Some results are visualized in Appendix Figure 15. We can see the model generates yellow color on white petals (like first row and last row). But for some yellow flowers, the model generates similar amount of yellow on it, which is not adequate (like the second row). The reason might be there are a large amount of yellow flowers in the dataset, and thus the final results are biased by yellow. For the third row, we can see the model assign purple on a red flower, and the purple region exceeds the region of petal. This is an indication that model is not able to clearly separate flower with its background.

Similarly for SC2 dataset (Figure 16), the results are assigned with averaged color, thus the blue minerals and green trajectory of weapon are indistinguishable in the results.

3.3. ConvNet for Classification

To train the classification model, given a ground-truth colored image, first I obtain its ground-truth color labels based on the quantized gamut. Then I feed the model with *L* channel, and compute the loss between the predicted labels with the ground-truth color labels. In the evaluation phase, I convert the predicted labels back to *ab* channel values and visualize the output images. Because of the large memory usage by this model, I have to set batch size to be 8. And the model is trained with Adam optimizer using 10^{-3} learning rate. Similarly, the input image size is still 224×224.

The results on LFW dataset do not have much vis-

Figure 11. Testing results from Simple ConvNet on OxFlower dataset. Left column: greyscale images; Middle column: ground-truth images; Right column: colorization results.

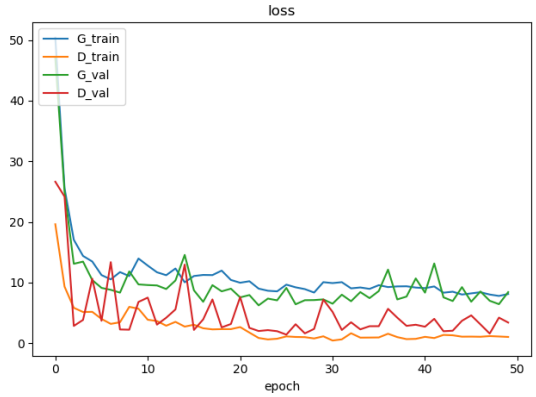

Figure 12. An example training and validation curve for Conditional GAN

ible changes (Figure 17). The most interesting result appears on the OxFlower dataset, which is shown in Figure 11. We can see the classification model increases the color variety in the output. For instance, we can see there are yellow spots appearing around the petals, although the location of the color is wrong or the flower is not supposed to be yellow (second row). In this example, the model is not able to clearly segment flowers with each other and the yellow color spans across the edge of each flower. Additionally, the model seems to be capable to assign green color to the background. Overall, the variety of color is improved, but the results are far from plausible.

3.4. Conditional GAN

For Conditional GAN, I train both generator and discriminator together in each batch, with same Adam optimizers and same 10^{-3} learning rates for both of them. The batch size is 32. After several experiments, I choose the regularization weight $\lambda = 100$, which means I put a large constrain on generating similar images. Without a large λ, the model will generate very fake images.

An example training and validation curve of Conditional GAN is shown in Figure 12. We can see after several epochs, the loss for both generator and discriminator get decreased. Note the generator loss keeps being larger than discriminator loss, which is mainly because of the regularization term with weight 100.

This Conditional GAN obtains very plausible output on artificial dataset like SpongeBob and SC2. The reason might be that the color of units in cartoon series and video games are pre-determined by human. Such that the color distribution complies some pre-determined rules, which is easier to be captured by neural networks.

Figure 13. Testing results from Conditional GAN on SpongeBob dataset. Left column: greyscale images; Middle column: ground-truth images; Right column: colorization results.

Figure 13 shows the output on SpongeBob dataset. We can see overall, the results are very plausible. For instance, SpongeBob is clearly yellow with sharp edge, and the shrimp shows red (second row), and the tank is generally green (first row). However, for

Figure 14. Testing results from Conditional GAN on SC2 dataset. Left column: greyscale images; Middle column: ground-truth images; Right column: colorization results.

a large region of mono-color, GAN generates some checkerboard patters coming from its convolutional kernels. This problem can be solved by applying further smoothing.

Figure 14 shows the output on SC2 dataset. The results are much better than previous two models, but the network is still not able to distinguish detailed objects. I suppose the reason is that the view of my images are too large, and thus some units are represented by very limited amount of pixels. Considering this, I further assembled dataset with 480×480 pixel images, and train the GAN on them. Figure 21 shows the output on large SC2 dataset. It is hard for me to distinguish the difference between the ground-truth and generated ones. However when training on SpongeBob, the result does not look that good. The reason might be in the cartoon series, there are many frames for background or transition. Thus the model does not get enough data to learn. Potentially using larger dataset can further improve the performance.

4. Future Work

There are several future research directions on image colorization. First is to increase the variety of colors. I verified that Euclidean distance will decreases the variety of colors. Zhang *et al*. [9] provide colorization methods, which can improve the variety. Besides, they actually proposed more methods like re-balancing the color distribution, in order to encourage rare colors to appear. Second, most of the published dataset has labels. For example, the OxFlower dataset contains labels indication the category of flower. Utilizing these models can be helpful for colorization. Intuitively, if the model can learn what the flower is before colorization, the results are expected to be better. For example, a multi-task learning model might be very helpful in this case. Thirdly, we can explore the model performance on larger dataset to check the generalizablity. Currently for efficiency purpose, my experiments are run on datasets with several thousands of images, which is quite small. All the colorful images and videos can be used as datasets for colorization tasks. So theoretically, it is possible to verify colorization methods on huge datasets. Ideally, creating a generalized colorization model will be very attractive.

5. Conclusion

In this project, I explore three methods for image colorization. The simple ConvNet is fairly easy to implement, but it produces averaged color in the output. The ConvNet for classification increases the variety of the color, but the model has large memory usage and has limited learning ability for segmenting object edges. Conditional GAN is a powerful model producing plausible and visual pleasant outputs. These three models are evaluated on four different datasets. Finally I discuss some future research directions in this area. My code for this project is published online: `https://github.com/TengdaHan/Image-Colorization`.

References

[1] Z. Cheng, Q. Yang, and B. Sheng. Deep colorization. *CoRR*, abs/1605.00075, 2016.

[2] I. J. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville, and Y. Bengio. Generative adversarial networks. *arXiv*, 2014.

[3] G. B. Huang, V. Jain, and E. Learned-Miller. Unsupervised joint alignment of complex images. In *ICCV*, 2007.

[4] G. Larsson, M. Maire, and G. Shakhnarovich. Learning representations for automatic colorization. *CoRR*, abs/1603.06668, 2016.

[5] M. Mirza and S. Osindero. Conditional generative adversarial nets. *CoRR*, abs/1411.1784, 2014.

[6] M.-E. Nilsback and A. Zisserman. A visual vocabulary for flower classification. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, volume 2, pages 1447–1454, 2006.

[7] O. Ronneberger, P. Fischer, and T. Brox. U-net: Convolutional networks for biomedical image segmentation. *CoRR*, abs/1505.04597, 2015.

[8] O. Vinyals, T. Ewalds, S. Bartunov, P. Georgiev, A. S. Vezhnevets, M. Yeo, A. Makhzani, H. Küttler, J. Agapiou, J. Schrittwieser, J. Quan, S. Gaffney, S. Petersen, K. Simonyan, T. Schaul, H. van Hasselt, D. Silver, T. P. Lillicrap, K. Calderone, P. Keet, A. Brunasso, D. Lawrence, A. Ekermo, J. Repp, and R. Tsing. Starcraft II: A new challenge for reinforcement learning. *CoRR*, abs/1708.04782, 2017.

[9] R. Zhang, P. Isola, and A. A. Efros. Colorful image colorization. *CoRR*, abs/1603.08511, 2016.

A. More results for Simple ConvNet

Figure 15. Testing results from Simple ConvNet on OxFlower dataset.

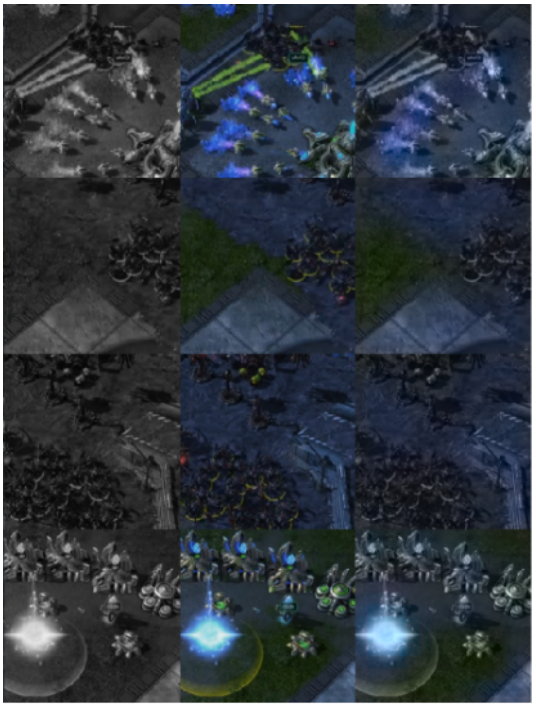

Figure 16. Testing results from Simple ConvNet on SC2 dataset.

Figure 17. Testing results from Simple ConvNet on LFW dataset. Left column: greyscale images; Middle column: ground-truth images; Right column: colorization results.

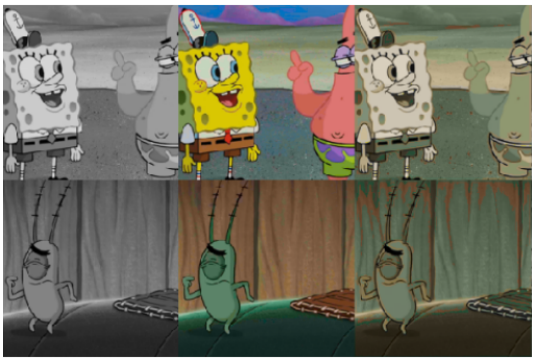

Figure 18. Testing results from Simple ConvNet on SpongeBob dataset. Left column: greyscale images; Middle column: ground-truth images; Right column: colorization results.

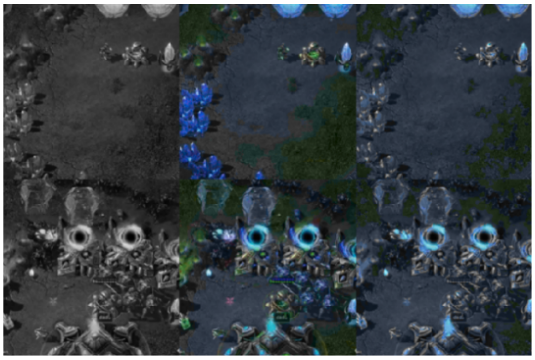

Figure 19. Testing results from Simple ConvNet on SC2 dataset. Left column: greyscale images; Middle column: ground-truth images; Right column: colorization results.

B. More results for ConvNet on classification

C. More results for Conditional GAN

Figure 20. Testing results from Simple ConvNet on OxFlower dataset. Left column: greyscale images; Middle column: ground-truth images; Right column: colorization results.

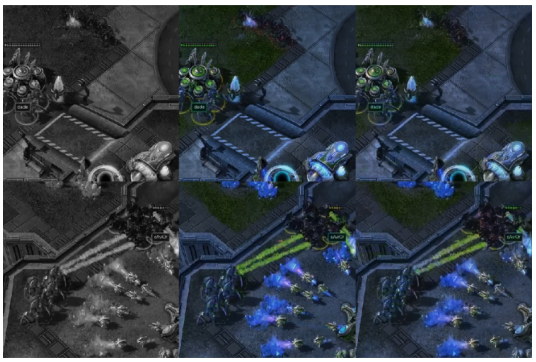

Figure 21. Testing results from Simple ConvNet on SC2 dataset with 480×480 pixels. Left column: greyscale images; Middle column: ground-truth images; Right column: colorization results.

Figure 22. Testing results from Simple ConvNet on SpongeBob dataset with 480×480 pixels. Left column: greyscale images; Middle column: ground-truth images; Right column: colorization results.