2020 2nd International Conference on Information Technology and Computer Application (ITCA) | 978-1-6654-0378-8/20/$31.00 ©2020 IEEE | DOI: 10.1109/ITCA52113.2020.00075

Sketch-to-Color Image with GANs

Wenbo Zhang
Arts, Sciences and Engineering University of Rochester
Rochester, United States
wzhang42@u.rochester.edu

***Abstract* - Unsupervised Learning is a trending research field of artificial intelligence, which aims to interpret and understand the hidden structure of the data. However, the development of deep learning with Generative Adversarial Networks (GANs) creates more possibilities for unsupervised learning. GAN is a category of Neural Networks, which are mostly applied to generating images. In this paper, how GAN was implemented to help with sketch-to-color translation is illustrated. In order to achieve this goal, data-preprocessing is implemented first. Then, the model is trained for 65 epochs, and the performance of the model is improved by virtue of loss functions and optimizers. In the end, a proper User Interface (GUI) is designed to have a full application, and people could turn any sketch picture they want into a colored image.**

Keywords-Colorization, Generative Adversarial Networks, Deep Learning

I. INTRODUCTION

Deep Learning is a subset of machine learning, enabling the machine to mimic human behaviors [1]. To be more specific, deep learning can achieve artificial intelligence without human intervention. Meanwhile, the independence of deep learning requires a much larger data size to support its function. The models of deep learning are trained like how we train humans. We applaud them when they get the correct answer and punish them when they make mistakes.

Generative Adversarial Networks, as known as GANs, is a generative model in machine learning [2]. There are two typical models in GANs, the generative model and the discriminative model. The generative model generates new samples and is adversarial to the discriminative model, which classifies the samples as either real or fake until the discriminative is unable to classify in most cases.

GANs has been applied in many fields. It can achieve the generation of realistic photographs or cartoon characters, color the old black-and-white pictures, generate any kinds of image-to-image translation, and even achieve the text-to-image translation, and so on [3]. Figure 1 shown below collects the faces of people generated by GANs who do not exist [4].

Figure 1. Images of People Who Don't Exist

This paper illustrates how the GANs can be used to achieve one type of image-to-image translation, which is the sketch-to-color cartoon image generation. A good sketch is the first step to the success of artistic work. Aesthetically pleasing color combinations, however, also play an extraordinary role in the production of artistic work and raise the artistic work to a much higher level. Indeed, colorization is always time-consuming and labor-intensive. However, if it is possible to color the sketch images automatically instead of using the manual way, there is no doubt that the efficiency will be improved. Much more artistic works will be produced in the near future. Therefore, how the Conditional Generative Adversarial Network is implemented to colorize the sketch image is discussed in the following sections.

II. MATERIALS AND METHODOLOGY

Sketch-to-image generation is achieved by an image-to-image translation model using the Generative Adversarial Networks. As mentioned before, GANs are composed of a discriminative model [5] and a generative model [5, 6].

Discriminative models are used to do the discrimination just like a classifier. To be specific, this model gets a single possible output from the input data. For example, it is able to make decisions from the possible classes, such as “fraud or not fraud”, or “spam or not spam”. More complexly, it is also able to do digit recognition and others. The underlying idea of the discriminative model, from the perspective of math and statistics, is that it will estimate the conditional probability whose input belongs to a class.

A generative model is used to learn any kind of data distribution by unsupervised learning. To be more specific, the goal of the generative model is to find out the density function from the distribution of a given dataset. However, generative models employed by GANs generate new samples, which fit the structure of the given dataset instead of finding out the density function accurately.

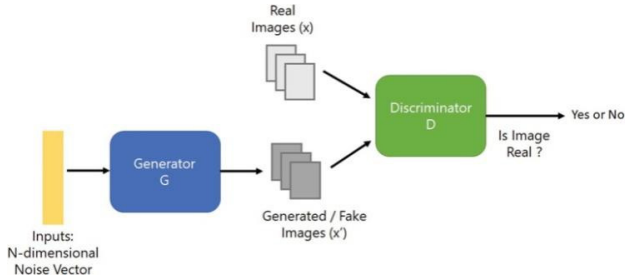

Figure 2. Basic Structure of GANs

Figure 2 shows the basic structures of GANs. As these two models are employed at the same time in GAN, the generative

978-1-6654-0378-8/20/$31.00 ©2020 IEEE
DOI 10.1109/ITCA52113.2020.00075

model generated candidates while the discriminative network evaluated them. In other words, a generator is just like a forger who produces forgery and trick people into believing that the products are real. And then, the discriminator model is the police who discriminate the fake products and catch the forgery.

In this study, the generator was used to produce the images, and the discriminator was used to distinguish between real and fake images. In the beginning, the generator randomly produced pictures that could not resemble the real images at all. After being discriminated as the fake image by the discriminator several times, they learned from the mistakes and produced better images than before. Meanwhile, the discriminator was better at discriminating the fake images from the real images. Therefore, these two models were adversaries of each other, but in fact, they helped each other to improve themselves simultaneously, until they reached the balance between them in the end. In other words, the discriminator was no longer able to find out the fake sample created by the generator in most cases.

A. Initial Setup & Data Preprocessing

The packages, including ***Tensorflow***, ***Matplotlib***, ***os***, and ***time*** need to be imported. ***Tensorflow*** is the machine learning framework used in this study, and ***Matplotlib*** is used to plot and show images. The ***os*** is the module, which is used for accessing and modifying path variables to save the checkpoints during the training process, and ***time*** is for counting the time needed for each epoch of training. All the packages rest are used for GUI I made to display the colorful picture generated by this model.

A dataset is available on the Kaggle, and it contains more than 14200 pairs of Sketch-to-Color images, as shown in Figure 3.

Figure 3. Original Image in Dataset

First of all, the image was divided into two groups, the input image with a black and white sketch and the colorized real image. Therefore, one function, which takes the original image file as the input, is used. This function not only loads the original images but also splits the images into a sketch image and a colored image. The details of the function are presented below.

Function 1: Loading Images and Split
Input: image file 1: get original image 2: get half width of the image 3: get real image which is left half of the image 4: get input image which is right half of the image Output: real image, input image

Then, as for the data preprocessing, unifying the size of all the images is needed as there may exist images with different sizes. Additionally, the size decreased from 512x512 px to 256x256 px in order to make the model training process much faster. Then, the images are normalized, and the unwanted area is removed.

After that, the ***data augmentation*** [7] is used to increase the training size and alleviate the problem of overfitting. Data augmentation is a technique to artificially create new training data from the existing training data, which is done by applying different techniques to examples from the training data that create new training examples. Based on all the images that already exist, random rotation, shear, zoom, and shift were done on the dataset to create more images. Moreover, advanced approach, like the Gaussian Noise, were incorporated into the process of ***data augmentation***.

B. Model Setup & Training

A generator model is a U-net architecture. The U-net is quite "lazy" because if the lower-level layers are able to solve the problem, the higher-level layers will do nothing. Correspondingly, the Generator model skips the connections to other layers.

Then, a down-sample is needed to make the computer performs better. The down-sampling stack of layers employing Convolutional layers led to the decreased size of the input image. Indeed, the up-sampling is processed as well to restore the size of the image back to the original size. Therefore, the output of the generator model is still 256x256 px. The generator model is mainly composed of two layers, which are the sequential layer and the concatenate layer.

By virtue of the generator model, pictures could be produced. And then, the discriminator model was invoked to do the discrimination procedure. Generally, the discriminator model is not as complex as the generator model. However, except for sequential and concatenate layers, it might be composed of different layers, such as BatchNormalization, Conv2D, LeakyReLU. The structure and its parameters of the discriminator model are summarized below.

Table 1. Structure of Discriminator

Layer (type)	Param No	Output Shape	Connected to
input_image (InputLayer)	0	(None, 256, 256, 3)	
target_image (InputLayer)	0	(None, 256, 256, 3)	
concatenate_10 (Concatenate)	0	(None, 256, 256, 6)	input_image[0][0] input_image[0][0]
sequential_31 (Sequential)	6145	(None, 128, 128, 64)	concatenate_10[0][0]
sequential_32 (Sequential)	131641	(None, 64, 64, 128)	sequential_31[0][0]
sequential_33 (Sequential)	533423	(None, 32, 32, 256)	sequential_32[0][0]
zero_padding2d (ZeroPadding2D)	0	(None, 34, 34, 256)	sequential_33[0][0]
conv2d_20 (Conv2D)	2123123	(None, 31, 31, 512)	zero_padding2d[0][0]
batch_nonrmalization_32 (BatchNormalization)	2048	(None, 31, 31, 512)	conv2d_20[0][0]
leaky_re_lu_10 (LeakyReLU)	0	(None, 31, 31, 512)	batch_normalization_32[0][0]
zero_padding2d_1 (ZeroPadding2D)	0	(None, 33, 33, 512)	leaky_re_lu_10[0][0]
conv2d_21 (Conv2D)	8193	(None, 30, 30, 1)	zero_padding2d_1[0][0]

After building the models, model training is another crucial process, which was achieved by the ***train_step*** function, which requires three parameters, including input_image, target, and epoch. In this case, the model is trained for 65 epochs. Then, the model is fitted. For each epoch, the following code is used to do the validation and see how the model works. Basically, after each epoch is finished, the window, including input image, ground truth, and predicted image, will show up.

Function 2: Visualization of Images for Validation
Input: model, input image, ground truth 1: set the variable named prediction as the image model colorizes 2: construct a list contains input image, ground truth and prediction 3: constructs the list contains the title of each image 3: **for** i = 0 to 2 4: plot the i_{th} image in the list 5: set the title of the i_{th} image 6: display the image with title 7: **end for**

Figure 4 shows the result displayed by the code above for each epoch.

Figure 4. Visualization of Prediction Outcome

As there were two models, two different loss functions were required to measure their loss independently. The loss for the generator was calculated by finding the sigmoid cross-entropy loss of the output images generated by the generator. Also, the L1 loss was used for the loss function of the generator, which is used to minimize the error and make the output image more similar to the target image. For the loss of the discriminator, the sum of the sigmoid cross-entropy loss of the target image and the cross-entropy loss of the output images generated by the generator were taken.

In order to minimize the loss, the ***weights*** need to be adjusted. Also, there is one more parameter named ***learning rate*** [8] to be concerned with, and that is how fast this model is expected to "learn". ***Learning rate*** is a constant, which controls the changing rate of weight since we want to avoid big steps during the adjustment of weight. For example, if the weight is adjusted by a large amount, it may overpass the minimum point, and the loss may not be lowered, even higher. Therefore, an optimizer, which is used to change these two parameters, is helpful to be employed, and ***Adam optimizer*** [9] was employed in this case.

III. Results and Discussion

Finally, the sketch-image was successfully colorized, and the GUI was developed, using the Tkinter and CV2 [10], in order to display the images in a more aesthetically pleasing way, as shown in Figure 5.

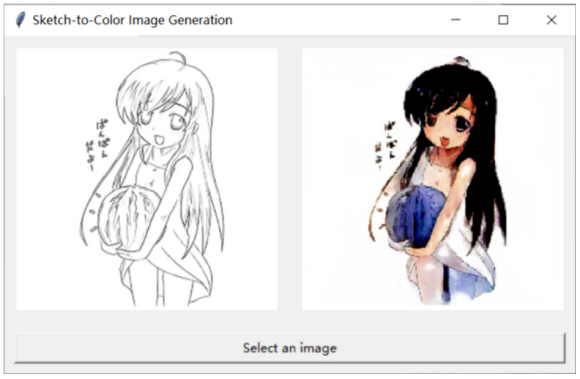

Figure 5. GUI for Colorization and Display

The overall effect is not bad, and it is not time-consuming at all for the model to finish the colorization comparing with hand-coloring by artists. However, when the sketch is complex and distorted, this model may fail to identify the features and get confused while colorizing, just like Figure 4. It is obvious that the model is doing well in most regions except the grids on the cloth since that region is relatively more complex and denser, and it is harder to precisely colorize that dense region composed of plenty of small grids. The main reason why the colorization is not perfect is that the model was not trained with enough epochs due to the limitation of time and lag of equipment.

IV. Conclusion

In this paper, the Generative Adversarial Networks is implemented to achieve one kind of image-to-image translation, which is sketch-to-color image generation. Also, extensive trials show that the model is doing well after comparing with the manual work, and it is much more time-saving to colorize in this way. In the age of advanced technology, tech makes life easier.

References

[1] Jason Brownlee. What is Deep Learning? August 16, 2019.

[2] Jason Brownlee. A Gentle Introduction to Generative Adversarial Networks (GANs). June 17, 2019.

[3] Jason Brownlee. 18 Impressive Applications of Generative Adversarial Networks (GANs). June 14, 2019.

[4] David Nield. New AI Generates Freakishly Realistic People Who Don't Actually Exist. February 19, 2019.

[5] Tanya Dembelova. Introduction to generative and discriminative models. August 7, 2020.

[6] Prakash Pandey. Deep Generative Models. February 1, 2018.

[7] Arun Gandhi. Data Augmentation | How to use Deep Learning when you have Limited Data – Part 2. August 20, 2018.

[8] Jason Brownlee. Understand the Impact of Learning Rate on Neural Network Performance. January 25, 2019.

[9] Vitaly Bushaev. Adam - latest trends in deep learning optimization. October 22, 2018.

[10] (Pandey 2018)] Adrian Rosebrock. OpenCV with Tkinter. May 23, 2016.