Big Data Analytics Final Report -- Image Translator

Zixiao Zhang
Electrical Engineering
Columbia University
e-mail: zz2500@columbia.edu

Xuelun Li
Electrical Engineering
Columbia University
e-mail: xl2678@columbia.edu

Abstract—**Before we stepped into the digital colorful world, the photos we took and the movies we watched were all in black and white. This only allowed us to take the contour of history while eliminating color. When we wanted to restore some historical scenes, the first thing came up was to restore color. With big enough datasets of black-and-white and corresponding colorful images, we implemented an algorithm of Conditional Generative Adversarial Nets (cGAN) and built our model. The model is of such strong adaptability and stability that it can process images of different sizes. In the model, there is a generator and a discriminator, in which a generator serves to colorize images and the discriminator functions to coordinate the training process with the generator. After times of training, the generator thus promotes its performance and generating from black-and-white images to more lifelike images.**

Key words: colorization, big data, GAN, image processing, machine learning

I. Introduction

Image colorization is always an interesting topic in both areas of art and computer vision, by adding colors to the black and white images, people can get different sense of them: colorizing historic images enables us to restore the historical scenes and to some degree, experience the past lives of people; colorizing the black and white photos of our beloved ones can make up for our colorful memory with them, etc.

Unlike its inverse process: transforming colorful images into BW ones, which is easy to realize because of no information increasing in the original images, colorization requires not only keeping the original structure of BW images but also generating more information about different channels. For a long time, colorization is always done inefficiently by hand, and now there are several ways that can help finish this task automatically. Conditional Generative Adversarial Nets (cGAN) is one of the most advanced methods which can perform colorization, based on big data and deep learning, the quality of images generated by cGAN exceeds most of other methods. Our project, named "Image Translator", aims at realizing a cGAN to "translate" BW images to lifelike colorful ones.

II. Related Works

With thousands of available training images, there are several different methods that can be used in colorization. [4] proposed an implementation based on feature descriptors and fully-connected neural networks. They combined different features (low, mid and high level) of pixels and trained a three-hidden-layer model with these features. Differing from [4], [2] implemented a fully convolutional architecture which only utilizes convolution operations to downsample or upsample the input images. The advantage of using convolutional layers is that there is no need to elaborately design features as input for the neural network, thus makes the whole model less complicated. Besides, [3] used cGAN to successfully generate diverse colorization results for single grayscale input (mainly for bedroom decoration); students from Stanford also used cGAN to add colors to grayscale manga, meanwhile compared the results with fully-convolutional nets and verified the superior performance of cGAN. [1] put forward a more generic structure of cGAN, which is not only applicable for colorization but also capable of other image translation tasks such as synthesizing photos from label maps, reconstructing objects from edge maps and so on.

Given the exciting performance of cGAN, we implemented our own one and finally obtained a model that can generate lifelike colorful images corresponding to grayscale input. particularly, our model can be applied to process images with various sizes, not only those which have the same width and height as training data.

III. System Overview

We use plenty of tools to construct our system, including python, tensorflow, google cloud platform, flask, html, javascript, and css. First we use python and tensorflow to preprocess our data, and then build a cGAN model in tensorflow, after which we train our model on google cloud platform. As soon as we finish training our model, we freeze it and serialize it in order to make it plantable. Finally, we build a web application based on our model with flask in the backend, and design a website by using html, javascript and css, in order to demonstrate our result in an interactive way.

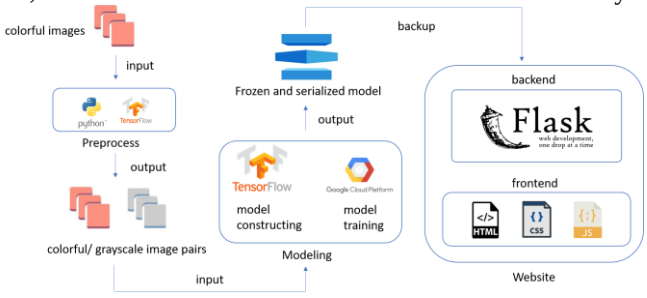


Figure1 block diagram of our project

IV. ALGORITHM

i. GAN

As we mentioned above, we use conditional Generative Adversarial Nets (cGAN) to perform colorization. Typically, GAN consists of two major parts: a generator and a discriminator. A generator aims at generating "fake" colorful images which then are delivered to the discriminator, a discriminator judges whether its inputs are "real" colorful images or "fake" ones. Generator can get feedback from the discriminator and thus keeps improving the quality of its output in order to deceive the discriminator, and at the same time discriminator continues trying to make right judgement. The best analogy should be the story of counterfeiters and polices. "The generative model can be thought of as analogous to a team of counterfeiters, trying to produce fake currency and use it without detection, while the discriminative model is analogous to the police, trying to detect the counterfeit currency. Competition in this game drives both teams to improve their methods until the counterfeits are indistinguishable from the genuine articles."[6] The true thing behind is that generator will finally learn to generate some distribution similar to our target distribution (true colorful images) from pure random noises.

Extended from GAN, cGAN has its power in generating something with designated purpose. The input for cGAN are some conditions we set as restrictions, in our case, they are original grayscale images, and we will concatenate them with both "fake" and "real" colorful images before they are passed to the discriminator. After lots of times' fighting against each other, our generator and discriminator will both become better and better in its own task, and eventually, generator will be capable of assigning natural colors to its input grayscale image based on the "knowledge" it learns from training data.

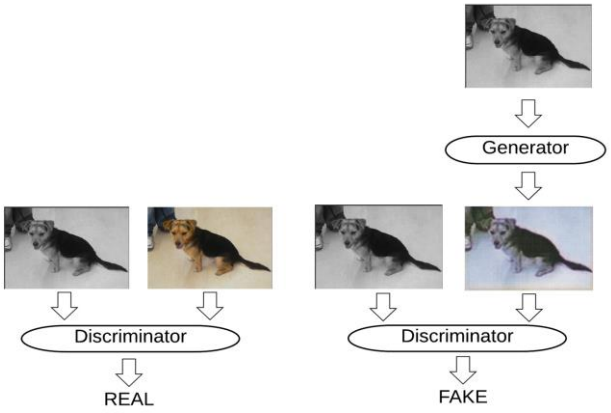


Figure2 conditional GAN

ii. Architectures:

Our generator has total 12 layers, the first 6 layers form an encoder, the other 6 layers form an decoder. Each layer contains a 2d convolution (2d deconvolution in decoder), a batch normalization and a ReLU (or leaky ReLU) activation (except the last layer of decoder, it's tanh activation). These 12 layers are organized as the structure of U-Net (shown in figure 3)

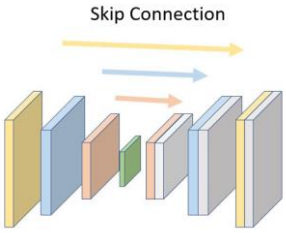


Figure3 U-Net

which means the mirroring layers in encoder and decoder will be concatenate to form skip connections. This structure can help model keep more structural information from the original images in order to make up for resolution reduction resulting from convolution operations, and thus make deep convolution architecture to become possible.

The discriminator shares the same structure as the encoder part of generator, the only difference is that we use sigmoid activation at the last layer.

iii. Objective function

The loss function for cGAN can be expressed as following:

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

where x is the condition, θ_d and θ_g are the parameters for discriminator and generator respectively.

We also introduce the L1 difference between input x and label y in generator as an extra loss term, that's say, the generator will minimize θ_g in following way:

$$\min_{\theta_g}\Big[-\log(D_{\theta_d}(x,G_{\theta_g}(x)))+\lambda\big\|G_{\theta_g}(x)-y\big\|_1\Big]$$

iv. Dataset

The original dataset we use are colorful images downloaded from IMAGENET, we search for different categories and collect 20,000 images covering animal, plant, landscape, building, people and so on. We first resize each picture to 250 * 175 and transform them into grayscale images, these colorful-grayscale pairs are then tiled and transformed to tfrecords files, which provide us with convenience (such as we can shuffle the images, and easily create batches) when fed into our model. We split them into training set and validation set, which contain 3/4 and 1/4 of the total images respectively.

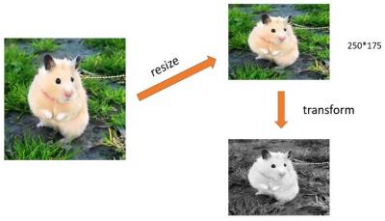


Figure4 image preprocessing

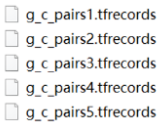


Figure5 tfrecords format

V. Software Package Description

Code is at link: https://github.com/Sapphirine/Image-Translator
Demo video is at link: https://www.youtube.com/watch?v=PsK3jaIu0iU&feature=youtu.be

VI. Experiment Results

i. Result

We train our model on google cloud platform, which provides GPU so that can accelerate training process. Training hyperparameters are set as following: batch size for input is 25, since larger batch size such as 128 can't fit in the memory of GPU; total epochs is 20; the learning rate for generator and discriminator are the same, 2e-4, and we use Adam optimizer for both of them, but the beta1 is set to be 0.5 rather than the default value 0.9; the weight of L1 distance in the generator loss function is 100 [1]; leaky slope for leaky ReLU is 0.2[1]; detailed convolution structure can be found in appendix.

Figure6 and figure7 show the training loss of discriminator and generator respectively. From these graphs we can observe the adversarial action between discriminator and generator: whenever the loss of discriminator decreases, the loss of generator will increase. Though they look oscillating all the time, actually in the long run we can see the loss of generator is decreasing slowly. And during training we keep sampling the validation results generated by generator, which also get more and more lifelike

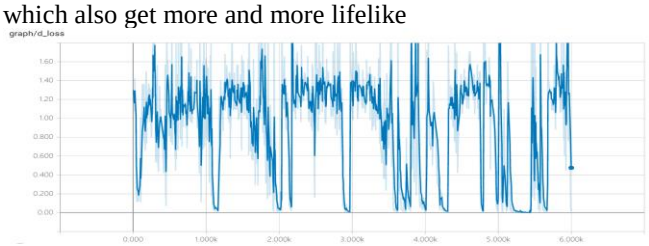


Figure6 loss of discriminator

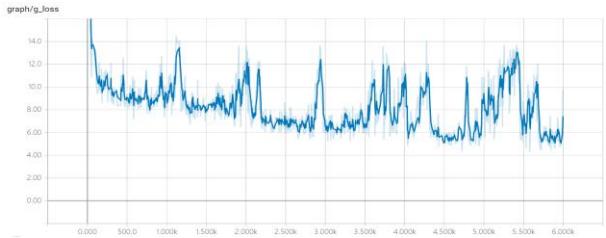
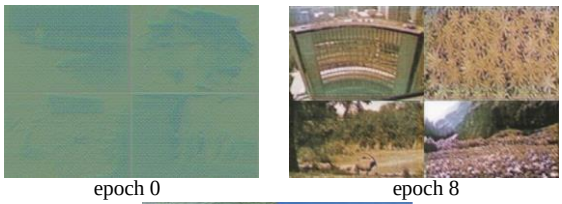


Figure7 loss of generator

We use per-pixel mean square error as evaluation metrics for our generator, and finally it goes down to around 0.1344. However, this evaluation metric has its drawback: it omits the structural information (joint distribution of all pixels) thus can't measure the spatial performance of synthesized images. Typically, evaluating the quality of synthesized images is difficult, there are other ways proposed in [1], and in our case, we mainly do it by observing the validation samples to judge whether our results are satisfying.



epoch 0 epoch 8

epoch 20

Figure8 samples from different epochs

We trained our model for 8 hours and then extract the generator part from the whole training graph (tensorflow build every operation in a graph). After freezing and serializing it, we can easily deploy it in other applications. Finally, we use flask as well as other tools to build a web app based on our model, which can easily transfer BW images into colorful ones.

Figure9 demo of our website

ii. Discussion

Training a well-behaved generative adversarial net is sometimes a hard task, and we also failed several times in our experiments. The first problem we met was out of memory error, this is because the images were too large to fit in the memory, that's why we resized them in the preprocessing part. We also met the very typical problem in training GANs, the loss of discriminator decrease to 0 in just several turns at the beginning, which will prevent generator's learning from data, we solved this by training generator twice in one iteration and just once for discriminator. Also there are other ways such as using a

weak optimizer for training discriminator whereas using a strong optimizer for generator.

Besides, normal deep learning model can only be applied to the data that has the same structure (such as width and length for image) as the training data, however, our model possesses the ability to deal with images with various size (as long as it's not too big for the memory), this essentially is because when we train convolutional layers, what we really get are some filters that can capture local features of the image, the size of processed image is not determinative of the size of filters, and fortunately due to the dynamic shape mechanism in tensorflow, we finally make our model more flexible (actually we can even train our model with images that have different size).

As for the performance of our model, it works very well in most of situations, but when there are regions filled with absolutely dark, it will generate some color similar to noises. And in terms of image with various size, it achieves best results in those whose size is close to training data. We believe, if we train our model with more images, it will definitely improve its performance in all the situations.

VII. Conclusion

In our project, we realized image colorization through a deep learning model, conditional generative nets. To finish it, we spent a lot of time and did lots of work. And finally we implemented the method and built a web app based on our trained model. Next, we will improve the performance of our model by training it with more data and also try to make our web app perfect.

Acknowledgment

Sincere thanks to professor Ching-Yung Lin for giving us vivid lectures on big data analytics and guiding us to think into real world problems, which inspired us doing this project.

Thanks to TA Gongqian Li and Mohneesh Patel for their patience, encouragement and wisdom in advising and assisting. They also pushed us further than we thought we could go.

Thanks to TA Aman Shankar for offering us feedback after the proposal and suggestion on timeline.

Thanks to TAs Sun Mao, Sheallika Singh, Vibhuti Mahajan, Vishal Anand, Congying Qiu, Lingqing Xu, and Ludwig Zhao. They listened to our presentations and gave us confidence and support on each stage of the project.

Thanks to Columbia University for providing us with such a nurturing environment and convenient facilities, especially computer lab at Mudd Building, where we conducted our research and project for nearly 2 months.

Special thanks to our beloved parents and friends for their love and care. They not only contributed the evolution of the initial idea of the project and also helped us through the stress and difficulties. We are truly appreciated!

Appendix

Convolution structure

The size of kernels for all convolution is 5*5, and convolution stride is 2, we also use zero paddings for shape adjusting.

The depths for generator convolution kernels are:

64-128-256-512-512-512-512-512-256-128-64

the output layer has depth of 3 (for R,G,B channels of colorful image.)

The depths for discriminator convolution kernels are:

64-128-256-512

the output layer is just a two dimensional matrix, which represents the probability of input being colorful image is true.

Contribution

Our team member Xuelun Li (xl2678) and Zixiao Zhang(zz2500) both finished 50% of this project, including data collection, data processing, modeling, web app developing; Senqi Cao (sc4084) participated in the discussion on the preliminary stage.

References

[1] Isola P, JunYan Zhu, Zhou T, et al. Image-to-Image Translation with Conditional Adversarial Networks[J]. 2016.

[2] Zhang R, Isola P, Efros A A. Colorful Image Colorization[J]. 2016:649-666.

[3] Cao Y, Zhou Z, Zhang W, et al. Unsupervised Diverse Colorization via Generative Adversarial Networks[J]. 2017.

[4] Cheng Z, Yang Q, Sheng B. Deep Colorization[J]. 2016.

[5] Radford A, Metz L, Chintala S. Unsupervised Representation Learning with Deep Convolutional Generative Adversarial Networks[J]. Computer Science, 2015.

[6] Goodfellow I J, Pouget-Abadie J, Mirza M, et al. Generative adversarial nets[C]. International Conference on Neural Information Processing Systems. MIT Press, 2014:2672-2680.