# OrGANic Image Colorizations using Deep Learning

Alex Hamrick, Sunny Siu

## Introduction

Since the invention of cameras, much of history has been recorded through photos. However, before the invention of the Lumiere Autochrome process in 1907, the photos were only in black and white. Nowadays photographers have taken these photos and manually colorized them based on their own knowledge of the era, an example of which is shown below (Fig. 1). But this takes time and requires an expert for an *accurate* colorization. We believe that neural networks can be used to give a black a white photo a *plausible and realistic* coloration.

Fig. 1. An image of Albert Einstein in 1939 colorized by @Edvos on Reddit

We know that deep learning can be effective at colorizing images, as it has been used for this task before, as described in Anwar et al. Both feedforward CNNs (Zhang et al.) and conditional generative adversarial networks (CGAN) (Nazeri et al.) have been successful in colorizing historical images. Additionally, deep learning strategies are likely superior to other methods given the sheer volume of "plausible" colorations for any given image. If the discriminator is sufficiently trained, it should effectively serve to ensure that the generated images are reasonably realistic.

In this paper, we will describe how we replicated the Deep Convolutional GAN (DCGAN) from Nazeri et al. using PyTorch, then will examine our attempts at substituting out various pieces (generators and discriminators) of this DCGAN for other networks. We then compare these different combinations to highlight various networks' ability to colorize images in a realistic

manner. We also highlight some of the issues with our selected networks and propose next steps to improve GAN performance on this task.

## Dataset

For training and testing, we randomly sampled an 8000 image subset from The Microsoft COCO Image Dataset, then used 80% for training and 20% for evaluation. We thought this dataset would be suited for the task since it was made for image recognition, COCO stands for Common Objects in Context, so hopefully the classifier could recognize common objects and the colors they have. All these images went through a series of transformations, they were converted from RGB to LAB, resized to 256x256, and then given random horizontal flips. We converted to LAB so we could use the L channel, black and white, as input and our network would predict the AB channels. Due to various computing and memory limits, it was infeasible for us to train on the entire dataset, but prior solutions have used similarly sized datasets with promising results (See Fig 2), and this set of images provided enough data for us to see preliminary results and determine which networks were feasible in this task. Given more advanced resources, we would have liked to train on even more images.

Fig 2. Results from Medium author @Moein Shariatnia after training on 10000 images

After training our networks, we used them to recolor real historical images. These images were taken from various sources and we use them to analyze our networks' outputs more qualitatively.

## Network Architectures

The first, and most important, part of this project was replicating DCGAN from Nazeri et al in PyTorch. This GAN uses a UNET as the generator and a network similar to PatchGan as the discriminator. Fig. 3 is taken from their paper and describes the UNET generator, and the discriminator largely just consists of the first (contractive) half of the same network.

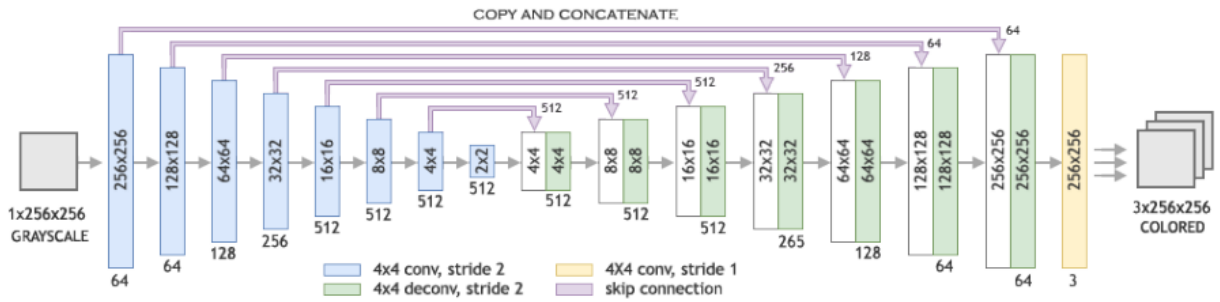



Fig. 3. The Generator for the DCGAN

We also had to write functions for the losses from the paper, which was essentially a binary cross entropy loss(BCE) plus the L1 between the fake and real image in the case of the generator, and a BCE loss for the fake and real image for the discriminator. These can be seen below in Fig. 4.

$$\min_{\theta_G} J^{(G)}(\theta_D, \theta_G) = \min_{\theta_G} -\mathbb{E}_z \left[\log(D(G(\mathbf{0}_z|x)))\right] + \lambda \|G(\mathbf{0}_z|x) - y\|_1 \quad (7)$$

$$\max_{\theta_D} J^{(D)}(\theta_D, \theta_G) = \max_{\theta_D} \left(\mathbb{E}_y \left[\log(D(y|x))\right] + \mathbb{E}_z \left[\log(1 - D(G(\mathbf{0}_z|x)|x))\right]\right) \quad (8)$$

Fig 4. Conditional GAN Loss for generator and discriminator from Nazeri et al.

Getting the network to effectively train proved more difficult than expected, as GANs are very finicky and require copious fine tuning of hyperparameters. Ultimately, however, we successfully implemented the DCGAN from Nazeri et al, which used a UNET generator and PatchGAN discriminator.

Next, we tried plugging in an out-of-the-box Resnet into a UNET, as the encoder, and utilizing the PatchGAN discriminator. This network used the same loss functions as the paper. We tried training this network with various lambda factors to change the emphasis on the L1 loss for the generator.

Finally, we tried writing our own generator and discriminator networks. Our custom generator was a series of convolutional blocks with residual connections (but no expanding or contracting). This was because we wanted to avoid the contractions of a UNET while still retaining the residual connections to previous layers. Our discriminator looked like a typical sequential classifier, and consisted of contracting convolutional layers, followed by batch normalizations and max pooling layers, with intermittent dropout and a linear layer at the end. It's important that

discriminators aren't too strong when used in a GAN or else the generator will never improve, so we thought this simple network would be effective and inexpensive.
Additionally, we used the L*A*B* color space for training our networks. We fed the generator the L (brightness) channel, and then had the networks generate both the A and B channels (representing the colors) before concatenating them with the L layer again to get the full image. This allowed our networks to only have to go from one to two channels rather than one to three. We initially assumed going from one to three channels would be easy since one channel would simply require the identity function applied to the input, but in our testing using only two output channels was far more effective and resulted in quicker and more consistent convergence. All 3 LAB channels were used as input for the discriminators with the output shape varying for each network.

## Results

For our experiments we ran 4 combinations of generators and discriminators. First was our simple generator and discriminator, which we used as a baseline for our other networks, that was trained for 20 epochs. Although the table shows that the MSE is low and the SSIM is among the highest, an examination of the actual images shows they are just black and white versions of the image. Even when excluding the L1 distance from our loss, the generator still outputted black and white images, so we concluded the network was simply not complex enough to learn the problem. These results can be seen in Appendix a and b.
Second, we used the UNET and PatchGAN, as suggested by Nazeri et al, and trained it for 20 epochs. This GAN gave the best results despite having the worst scores. Though the generated images may not be as similar to the original image, they still provided a realistic coloring as seen in Figure 4 as well as Appendix c and d.


Figure 4. Results from the UNET+PatchGAN combo on 5 of the evaluation images.

Third, we used a pretrained Resnet18 as the encoder for the UNET, as suggested by Shariatnia's blog post, and PatchGAN that was trained for 20 epochs. We had to use lambda=1 as the L1 portion of the loss was too dominating otherwise and led to black and white images. These results can be seen in Appendix e and f.

Finally, we tried training the same GAN as the second, but with 20 epochs of pretraining, with just an L1 loss, on the generator. This was only trained for 15 epochs, as results began to look overfit after this. These results can be seen in Appendix g and h.

Our evaluation stage had 2 portions: quantitative and qualitative. As our ultimate goal was a *plausible and realistic* coloration it was pretty hard to measure quantitatively, but we still felt these measures would help. The quantitative measures we used were Mean Squared Error (MSE) and the Structural Similarity Index (SSIM), but as we will see later these do not tell the whole story. These measures were picked because they were common in papers we read for this task. Results can be seen in Table 1. The qualitative measures came from us manually examining how realistic the images looked after coloring. These images can be seen in the Appendix and the reader can decide how well the networks did.

| | Custom Gen + Custom Disc | UNET Gen + Patch Disc | UNET w/ ResNet18 Gen + Patch Disc | Pretrained UNET Gen + Patch Disc |
|---|---|---|---|---|
| MSE | 0.0071 | 0.0123 | 0.0089 | 0.0085 |
| SSIM | 0.91 | 0.8503 | 0.8414 | 0.8899 |

Table 1. Scores for each GAN

# Conclusion

We found a large degree of success when replicating the network from the Nazeri et al paper, but most of the modifications we made to this GAN resulted in a degradation of performance. In fact, most of the modifications we made (some of which aren't even shown here) simply resulted in black and white images. Our best guess is that this is caused by an imbalance between the generator and discriminator. Either the discriminator gets too strong and the generator can never make convincing images or the generator is too focused on a local minimum loss value due to the L1 loss term and gets stuck producing black and white images. This is also evidenced by the fact that, after reducing the factor of L1 loss with the Resnet, we saw the network learn to display at least muted colors. However, it is clear that training a generator to accurately color images is quite challenging.

Something interesting to note about our results is that the networks that generated the best colorizations had the worst metric scores (MSE and SSIM). The UNET generator created the most realistic images, despite having the lowest MSE and SSIM scores. This may indicate that utilizing a more forgiving loss function (or a worse discriminator) could help improve our colorizations even more.

Given more time, the next steps for improving our GAN would be to experiment with new loss functions and try alternative training methods (like alternating training between generator and discriminator rather than training end to end). Additionally, we would like to experiment with training these networks in RGB space rather than L*A*B* space in order to see what effects that would have on our results. Finally, once a network is accurate enough, it would be interesting to

try and use real humans for evaluation. As we saw, MSE and SSIM were not the best metrics for assessing realistic colorations, so for a more complete qualitative analysis it would be ideal to have real people guess whether an image was generated or not and average their accuracy.

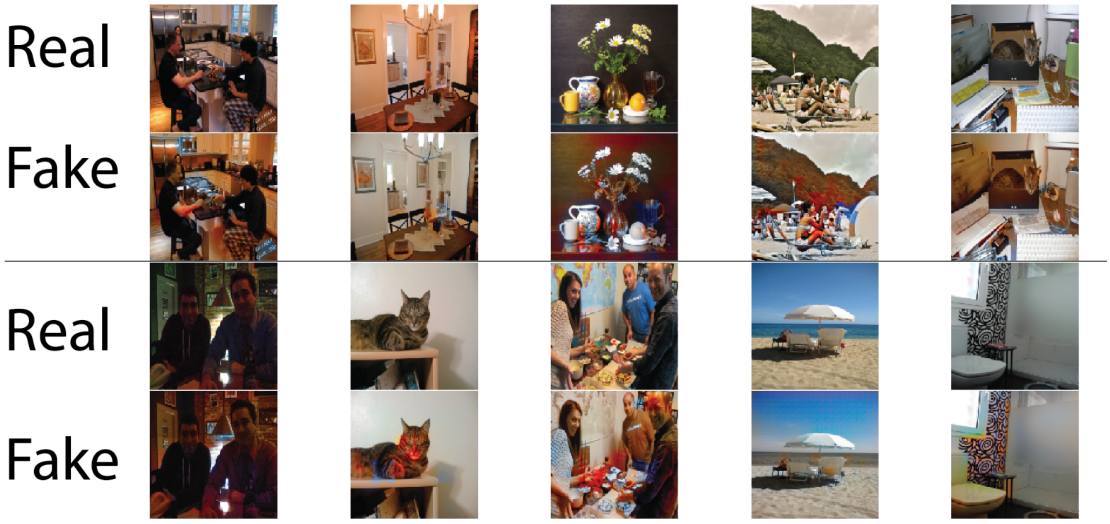
## References


- Dataset: https://cocodataset.org/#home
- Anwar, S., Tahir, M., Li, C., Mian, A., Shahbaz Khan, F., and Wahab Muzaffar, A., "Image Colorization: A Survey and Dataset", *arXiv e-prints*, 2020.
- Nazeri, K., Ng, E., and Ebrahimi, M., "Image Colorization with Generative Adversarial Networks", *arXiv e-prints*, 2018.
- Zhang, R., Isola, P., and Efros, A. A., "Colorful Image Colorization", *arXiv e-prints*, 2016.


## Appendix


a. Custom GAN Test Results. Unable to add additional color info


b. Custom GAN Historic photo results. Unable to add color info.



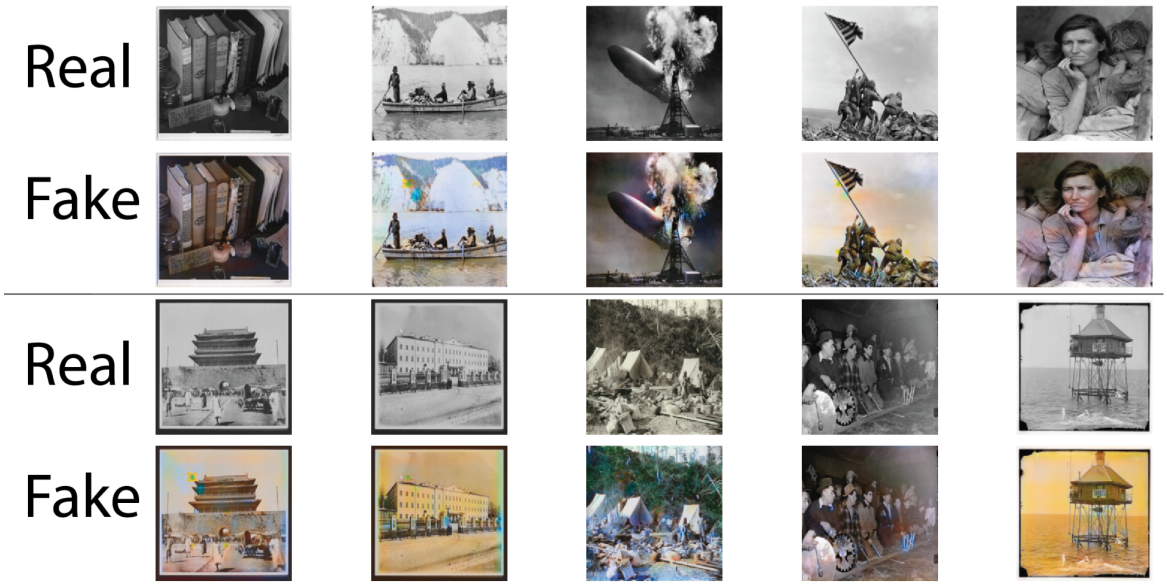

c. UNET + PatchGAN Test results. Good at coloring inanimate things, terrible at skin and fur.


d. UNET + PatchGAN Historic Photo results. Photos look more like a painting than a restoration. Nature and low lighting (books) look decent.


e. UNET w/ Resnet + PatchGAN Test Results. Unable to add much color, better than sequential because as it attempts to add some color.


f. UNET w/ Resnet + PatchGAN Historic Photo results. Same as e.


g. Pretrained UNET + PatchGAN Test Results. Similar results to non-pretrained. No clear advantage in coloration quality. Took less training epochs.




h. Pretrained UNET + PatchGAN Historic Photo results. Fails to add color in many cases. Possibly overfitting on training data.