Can We Color It? Yes We GAN!

Jackson Sargent
Computer Science Engineering
University of Michigan
jacsarge@umich.edu

Naitian Zhou
Computer Science Engineering
University of Michigan
naitian@umich.edu

Nathan Martin
Computer Science Engineering
University of Michigan
nathmart@umich.edu

1. Introduction

Image colourization is an open problem in computer vision: given a grayscale image, we'd like to generate a realistic, colored version of the image. This task has applications in photo restoration, image processing, computer-aided animation, and showing off to your computer vision pals. This seems like a Lofty goal [1]. Clearly, colors cannot be determined exactly from a greyscale image; instead, the task is to generate a realistically coloured image.

Advances in deep learning have allowed not only for the classification of, but also the generation of images. Other image generation tasks include constructing semantic segmentation maps from raw images and image style transfer [3, 7]. Previous work has involved using convolutional neural networks to generate images. Early work resulted in desaturated images, because of the inappropriate loss functions which favor conservative color prediction. Later attempts tried to address this by hand-crafting loss functions specifically for the colorization task [8]. In these approaches, they adopted a "It's better to do one job well than two jobs... not so well." mentality [1].

Some of these prior works included colorization via classification, which used a CNN to classify each black and white pixel over a distribution of values each representing a color in a pre-selected palette of colors [2].

As an alternative approach, GANs have proven effective at many image generation tasks, including image colorization. GANs, or generative adversarial networks, include a pair of networks: a generator and a discriminator. The generator learns to generate outputs while the discriminator learns to tell the difference between real images and generated images.

They tend to produce more realistic images compared to previous computer vision methods, without requiring a custom, task-specific loss function. Instead, the discriminator learns an effective loss function by comparing the generated versus ground truth images. We are implementing a general image-to-image architecture from the pix2pix paper, using a modification for the colorization task [4].

We used a generative adversarial network (GAN) to produce images because previous methods with a set objective function tend to result in muddier images. In contrast (pun intended), GANs tend to produce more natural-looking images with vivid colors. We trained on a subset of the CoCo dataset [5] and perform both quantitative (peak signal-to-noise ratio) and qualitative evaluations.

2. Approach

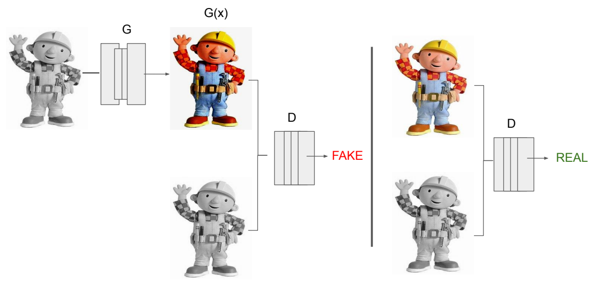

Figure 1. How a GAN learns to colorize an image (modified from [4])

Generally, GANS are models that generate a mapping from random noise to an output image. For our work, we implemented a conditional GAN, which maps from a generated image and random noise to an output image. The GAN is made up of a generator, which learns to produce these

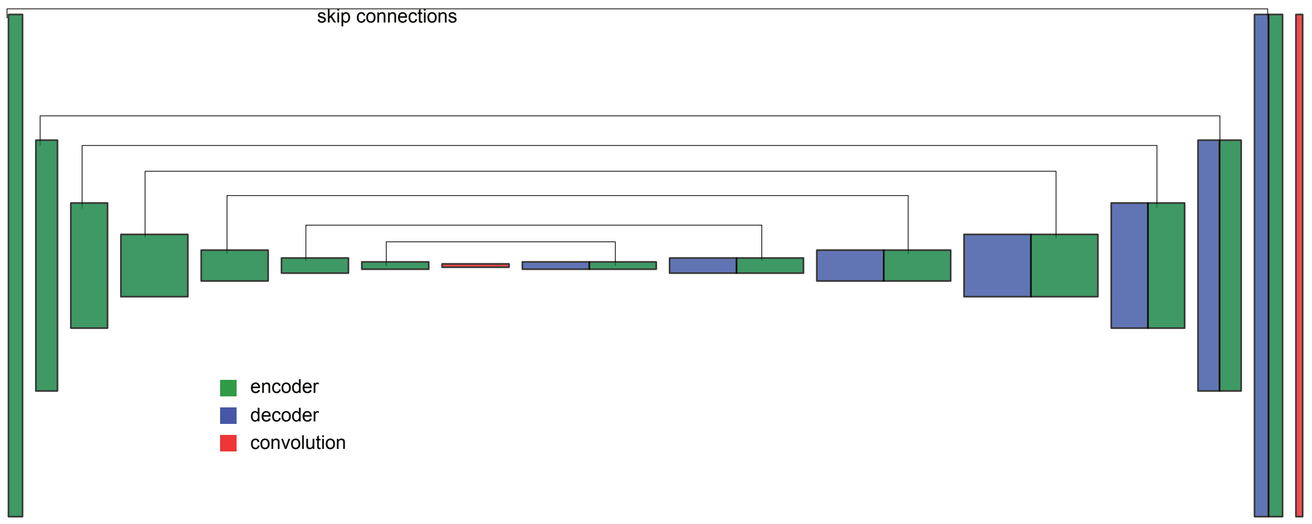

Figure 2. U-Net architecture used for the generator in this project

outputs, and a discriminator which is trained to detect the fake outputs generated by the generator.

We implemented the pix2pix conditional GAN used for image colorization as initially proposed by Isola et al. [4]. Their work proposes a generally applicable GAN for image to image translation, used for tasks like converting landscape images from night to day, converting labels to facades, and converting edges to real images. Our goal was to reproduce their work for the purpose of image colorization.

For this work, we used PyTorch [6] to implement the generative adversarial network that utilizes a discriminator and a U-Net architecture based generator [7].

2.1. Generator

One interesting feature of the model is the U-Net generator, which utilizes 'skips' to preserve the structure of the input image that is learned in the encoding stage of the network, and concatenates these encodings to the stages of the decoder that generates the image. Our model has 8 blocks for encoding, and 8 for decoding, following the structure given in Figure 2.

Each encoder was comprised of a 2D convolution layer, with kernel size 4×4, padding of 1 and stride of 2. This means the image size is effectively halved with every layer. This was followed by a Leaky ReLU activation layer with leaky slope of 0.2. For all encoders besides the first one, the convolutional output underwent batch normalization before the non-linear activation.

The decoder is comprised of a transpose convolution layer with kernel size 4×4, stride 2 and padding 1, followed by a batch normalization layer. The output of the decoder is concatenated with its corresponding skip layer before being passed into the succeeding decoder layer.

2.2. Discriminator

The discriminator is similar to the generator in it's use of Convolution-BatchNorm-Relu blocks. Our discriminator makes use of 4 of these blocks, where the first doesn't use BatchNorm and every one of them has a Leaky ReLU activation layer with a leaky slope of 0.2. Our discriminator's first layer has 64 filters, and then doubles each block. The last block is followed by a single convolution which takes the 512 channels of the last block and turns into 1 output channel.

2.3. Training

We trained our model on the a subset of the CoCo dataset. We shuffled the sampled images and divided into a training and validation set with 10,000 images used for training and 2,000 for validation. We down-sampled the images to 256×256 pixels and performed random horizontal flips as data augmentation. For learning, we used a ADAM optimizer with a learning rate of 2×10^{-4}, and two different loss functions.

We used BCE with Logits Loss for training the discriminator, by first running the discriminator on the input paired with the generated output, and then running with the input paired with it's ground truth output. For the generator, we use the same BCE with Logits Loss by running the generated image again through the discriminator and combining that loss with the L1 Loss in the difference between the ground truth and the generated image.

3. Experiments

In order to train our GAN, we used a subset of the CoCo dataset, provided by fastai. We were not concerned with

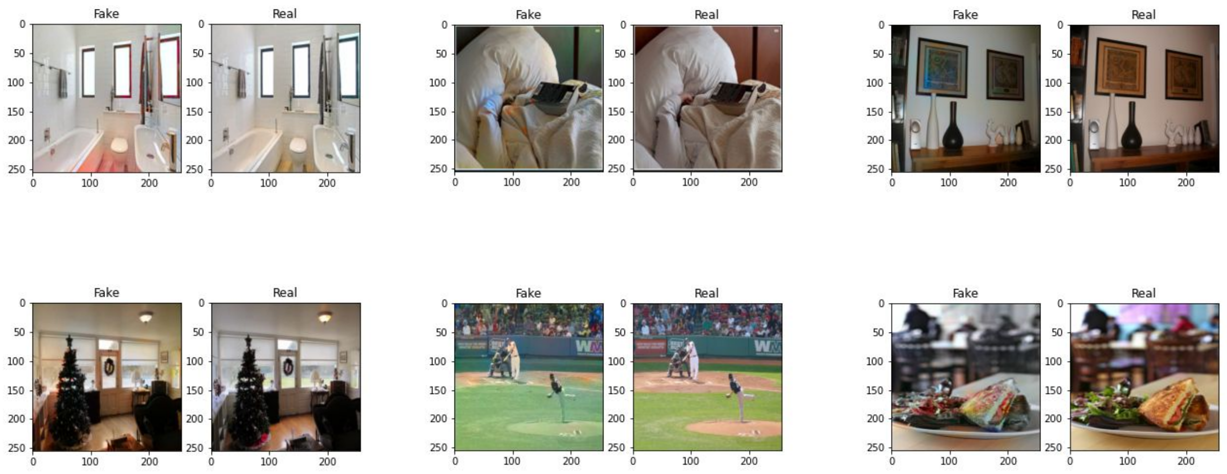

Figure 3. GAN results

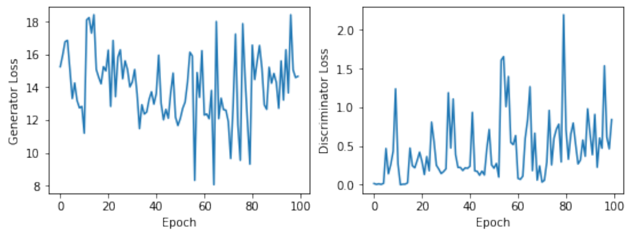

Figure 4. Discriminator and Generator loss per epoch

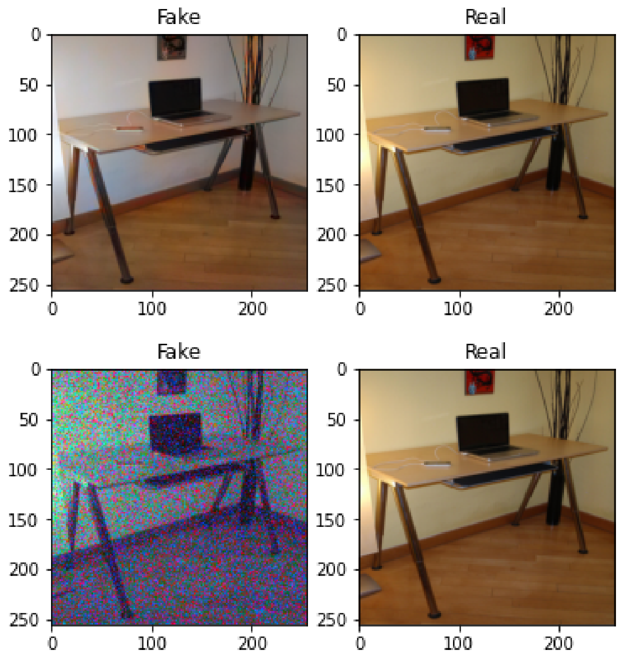

Figure 5. An example of our GAN's colorzation (top) vs random ab channel creation (bottom)

any one type of image colorization, so we wanted to train on the broadest possible image set. Our CoCo sample allowed us access to a number of images of varying lighting levels, color combinations, and objects. We trained the model on Google Colab using the cloud GPU runtime for 100 epochs over the training dataset. While this was cost-effective, it did lead to difficulties when we tried to train for longer stretches of time, and when we tried to save model files.

The discriminator and generator training loss curves are provided in Figure 4. GANs tend to have noisy training curves because of the adversarial nature of the training process: an improvement in the generator would lead to a decline in the performance of the discriminator. When comparing our results to loss curves in other work, we found the loss curves tend to converge and stabilize after a couple thousand epochs. This suggests we may benefit from training for much longer; this was not possible given our resources both in terms of time and computation.

Results from the model are shown in Figure 3. We also provide a comparison to a random generation baseline in Figure 5. Compared to the random baseline, our model produces realistic colors and a much cleaner image; the baseline result (understandably) does not pick realistic colors for the objects depicted, and is significantly noisier (since it hasn't learned the relationships between pixels to form objects).

In order to evaluate our model quantitatively, we calculated the peak signal-to-noise ratio between the generated images and ground-truth images in our test set. We achieved an average PSNR, using cv2's PSNR function, across a test set of 2,000 images of 68.52. We compared this to random generation of ab channels for our test set, which had a PSNR average of 56.912. In PSNR, a higher value generally indicates better quality, with values over 60 generally considered high quality, indicating our colorization performed well quantitatively.

Quantitative evaluation is beneficial, but for colorization a qualitative approach is needed due to the subjective nature of potential coloring. A certain object could have multiple colors and so even if our generation deviates from the original image, it may still be entirely plausible, and that's what we're looking for. We showed 10 images to friends and family and asked them to rate each picture color on a scale of 1-5. Five images were generated by our model and five were the ground-truth images.

	Real	**Fake**
Mean	4.4706	3.1667
Std. Dev.	0.7001	1.2837

Table 1. User evaluation ratings on real and fake images.

Qualitatively, the GAN can generate some realistic images, though the quality varies. We surveyed 17 of our closest friends, family, and instructors with 5 real and 5 fake images and found that overall they were able to discern fake images from real ones, giving fake images a generally lower score. For real images, the average score was 4.47 out of 5 with a standard deviation of 0.70. For the generated images, the mean was 3.17 out of 5 with a standard deviation of 1.28. These are also listed in Table 1. Just for kicks, we performed a single tailed t-test to see if we can conclude that the ratings of fake images come from a statistically significantly different distribution compared to the ratings of real images. We found a p-value of 0.19 that we cannot conclude that the scores for the fake images come from a statistically significantly different distribution at the 95% confidence level.

The generated images still have visual artifacts (see top middle image of Figure 3 for one example). There are also some semantically miscolored portions of images, where the color of an object does not align with reality such as the bottom image of the same figure.

4. Implementation

We implemented the data loader, discriminator and generator in PyTorch, as well as the training loop. The dataset we wrote included functions for randomly flipping an image horizontally. For our discriminator and generator, we implemented individual functions to create the blocks of layers described in the Approach section, and used these functions to create a separate network for each. We used the `fastai` library to download a subset of the CoCo dataset [5] and `opencv` to quantitatively evaluate our results using the PSNR metric. For qualitative evaluation, we used Google Forms to present our survey, and Google Sheets to perform the analysis.

References

[1] Bob the builder. 1

[2] V. Billaut, M. de Rochemonteix, and M. Thibault. Colorunet: A convolutional classification approach to colorization. *arXiv preprint arXiv:1811.03120*, 2018. 1

[3] L. A. Gatys, A. S. Ecker, and M. Bethge. Image style transfer using convolutional neural networks. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 2414–2423, 2016. 1

[4] P. Isola, J.-Y. Zhu, T. Zhou, and A. A. Efros. Image-to-image translation with conditional adversarial networks. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 1125–1134, 2017. 1, 2

[5] T.-Y. Lin, M. Maire, S. Belongie, J. Hays, P. Perona, D. Ramanan, P. Dollár, and C. L. Zitnick. Microsoft coco: Common objects in context. In *European conference on computer vision*, pages 740–755. Springer, 2014. 1, 4

[6] A. Paszke, S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan, T. Killeen, Z. Lin, N. Gimelshein, L. Antiga, et al. Pytorch: An imperative style, high-performance deep learning library. *arXiv preprint arXiv:1912.01703*, 2019. 2

[7] O. Ronneberger, P. Fischer, and T. Brox. U-net: Convolutional networks for biomedical image segmentation. In *International Conference on Medical image computing and computer-assisted intervention*, pages 234–241. Springer, 2015. 1, 2

[8] R. Zhang, P. Isola, and A. A. Efros. Colorful image colorization. In *European conference on computer vision*, pages 649–666. Springer, 2016. 1