Automated Colorization of Flowers with Generative Adversarial Networks

Yiding Chen [1] **Feng Jiang** [1] **Newton Wolfe** [1] **Yuqing Zhu** [1]

Abstract

We implement deep convolutional generative adversarial neural networks (DCGANs) to generate full-color photorealistic images of flowers from edge patterns. We input edge images to the generator to generate full color pictures,and use the discriminator to judge whether the generated images are real. Both generator and discriminator are implemented using convolutional neural network (CNN). We analyze the challenges that stand in the way of training generative networks, and suggest future work to refine our model.

1. Introduction

While neural networks have been incredibly effective at computer vision tasks like recognizing objects (Krizhevsky et al., 2012; Simonyan & Zisserman, 2014; Szegedy et al., 2015), classifying handwritten digits (LeCun et al., 1998), and image segmentation (Kuntimad & Ranganath, 1999), only recently has research begun on using neural networks in generative models. One of the most compelling problems for generative models focusing on images is to add detail to images. When a painter paints a picture, their process usually consists of a first sketch and then layering on detail. Along that vein, in this paper, we aim to generate full-color realistic images of roses from edge patterns. Our goal is to develop a system that can go from a human sketch of a rose to a full-color photorealistic image. This has parallels to work in image-to-image translation problems, which have become a popular research topic in recent years. (Isola et al., 2016)

Significant work has been applied to solving similar image translation problems via convolutional neural networks (CNNs). (Zhang et al., 2016; Deshpande et al., 2015) However, the CNN relies upon minimizing a given loss function, and applying naive loss functions such as minimize pixel-wise Euclidean distance does not produce a photorealistic output. (Isola et al., 2016) This is because Euclidean distance is minimized by averaging all plausible outputs. Designing a reasonable loss function for a CNN that preserves photorealism may require significant domain knowledge related to image processing.

To address this problem, we propose using the Generative Adversial Network (GAN) framework, as first introduced by Goodfellow et al. (2014). In a GAN, there are two neural networks with opposing objective functions, the generator and the discriminator. In our case, the generator network will take the edge boundaries of a rose (simulating a human sketch), and output an RGB image. The discriminator will take the output RGB image and then determine if the image is a real photo of a rose. These two networks will then simultaneously perform gradient descent, the generator towards the discriminator classifying its output as a real photo, and the discriminator towards correctly classifying real photos versus false outputs.

We performed further evaluation by comparing our model with the GAN framework against a convolutional neural network baseline using a similar structure to our GAN generator, and to the work in Isola et al. (2016) using conditional GANs (CGANs).

2. Methods

2.1. Dataset

Our dataset included about 500 full color (RGB) images of red rose with white background, scraped from various sources on the internet. These images were then resized to 64 x 64 pixels, a size chosen to trade off between recognizability and computational cost to process. For each image of a rose, we generated the edge pattern by Canny edge detection, which provided a detailed basis for reconstruction. (Canny, 1986) This edge detection was chosen because it seemed to provide

[1]University of Wisconsin-Madison, Madison, Wisconsin, USA. Correspondence to: Newton Wolfe <nwolfe@cs.wisc.edu>, Yiding Chen <ychen695@wisc.edu>, Feng Jiang <fjiang23@wisc.edu>.

CS 760 Class Project, Spring 2017.

the best balance between detail for reconstruction and human sketch plausibility. This batch processing was done using the scikit-image library in Python. (van der Walt et al., 2014) Sample images and their edges are displayed in Figure 1.

Each pair of image and associated edges was then used in the training of our models as a training instance. The edge images are used as inputs drawn from the random noise distribution, and the generator will take them to output fake full images, while the full images were used as the real images drawn from the dietribution that the generator intends to simulate. Both the fake and real images are then used to train the discriminator. Theoretically, a successfully trained generator in our GAN should later on be able to take an arbitrarily drawn edge image and output an colorful flower image that the discriminator finds it hard to distinguish.

Figure 1. Random sample of 100 pictures and their edge patterns that we used in our dataset

2.2. Baseline based on CNN

We developed a baseline algorithm using a simple convolutional neural network structure, with an objective function minimizing the Euclidean distance between the average over the pixel of the prediction and the ground-truth figure. More formally, our mean squared error function is as follows:

Suppose our training dataset is $X = \{(x_i, y_i) \mid i = 1, 2, \cdots, M\}$, where M is the number of training images in our training dataset, x_i be the edge pattern input, and y_i the corresponding ground truth full color output image, θ represents the parameters in CNN.

$$L(x;\theta) = \frac{1}{N}\sum_{p=1}^{n} |F(x;\theta)^p - y^p|^2$$

We train our baseline model on the same dataset with minibatch stochastic gradient descent. We define our minibatch loss as the mean of the individual example losses,where X_B is the minibatch sampled from our input set X.

$$L_B = \frac{1}{|B|}\sum_{x \in X_B} L(x;\theta)$$

We used the same structure for our CNN as we used as the generator in our GAN; details are in subsubsection 2.3.1 and specifically Figure 2.

2.3. Method base on GAN

Cite directly from (Radford et al., 2015), Architecture guidelines for stable Deep Convolutional GANs:

- Replace any pooling layers with strided convolutions (discriminator) and fractional-strided convolutions (generator).
- Use batch normalization (Ioffe & Szegedy, 2015) in both the generator and the discriminator.
- Remove fully connected hidden layers for deeper architectures.
- Use ReLU activation in generator for all layers except for the output, which uses Tanh.
- Use Leaky ReLU (He et al., 2015) activation in the discriminator for all layers.

In our GAN, there are two networks with competing objectives. The discriminator attempts to differentiate between the real images of roses and images that the generator network generates based on the edges and random noise. We call the generator network $G(x : \theta_G)$ where x is some prior distribution, in our case the edges derived from an image, and θ_G are the network parameters. In our specific implementation, G is a deep neural network whose architecture we explain in subsubsection 2.3.1, and thus it is differentiable by backpropagation. The discriminator, then, is a function that takes an input y and parameters θ_D; again, in our case it is a deep neural network. Algorithm 1 explains the mathematical interaction between these two networks; while the objectives aren't completely in opposition, these objectives have been empirically shown to work. (Goodfellow, 2017)

2.3.1. GAN Architecture

Our generator is a five layer CNN. The first convolution layer has 16 kernels, each with 10×10 convolutions. The second convolution layer has 32 kernels,

Algorithm 1 Minibatch stochastic gradient descent of our adversarial colorization nets. We used a standard normal distribution for the noise prior $p(x)$. In our experiments, used Adam optimizer for our gradient updates.

1: **for** number of training iterations **do**
2: Sample minibatch of m edge image $\{z(1), \cdots, z(m)\}$ and corresponding RGB color examples $\{x(1), \cdots, x(m)\}$
3: Update the discriminator D by ascending its stochastic gradient:

$$\nabla_{\theta_d} \frac{1}{m} \sum_{i=1}^{m} \left[\log D\left(\mathbf{x}^{(i)}\right) + \log\left(1 - D\left(G\left(\mathbf{z}^{(i)}\right)\right)\right)\right]$$

4: Sample new minibatch of m edge image examples from our data set
5: Update generator G by descending its stochastic gradient:

$$\nabla_{\theta_g} \frac{1}{m} \sum_{i=1}^{m} \log\left(1 - D\left(G\left(\mathbf{z}^{(i)}\right)\right)\right)$$

6: **end for**

each with 5×5 convolutions. The third convolution layer has 16 kernels, each with 3×3 convolutions. The fourth convolution layer has 8 kernels, each with a 2×2 convolutions. The last convolution layer has 3 kernels, each has with 1×1 convolutions; this is also our output layer. The stride of each kernel in the convolution layers sampling the output of the previous layer is $(1, 1)$. All the layers, excepts for the last layers, use the leaky RELU as the activation function. The last layer uses tanh as the activation function. There are zero-paddings in each layer to keep the size of every convolution plate the same. The architecture is so desined as to try our best not lossing information by keeping the same convolution plate size, but at the same time to generate more information as we increase the kernel numbers. A visualization of this network can be found in Figure 2.

The discriminator is a four layer CNN. The first convolution layer has 16 kernels, each with 4×4 convolutions. The second convolution layer has 32 kernels, each with 4×4 convolutions. The third convolution layer has 64 kernels, each with 4×4 convolutions. The stride of each kernel in the convolution layers sampling the output of the previous layer is $(2, 2)$. The last layer is a fully-connected layer with only one output node; this is also our output layer. All the layers, excepts for the last fully-connected layer, use the leaky RELU

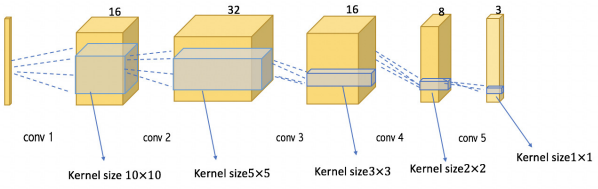

Figure 2. Architecture of the generator CNN

as the activation function. The fully-connected layer uses sigmoid as its activation function. The architecture is so designed as to summerize information from the RGB pixels and determine the input image as fake or real. A visualization of this network can be found in Figure 3.

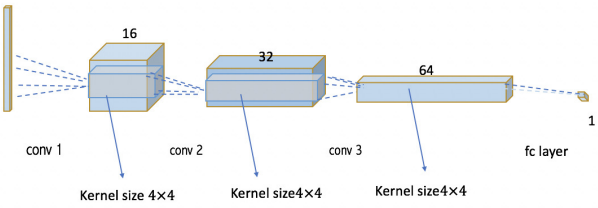

Figure 3. Architecture of the generator CNN

In both networks, we use batch normalization at each layer to improve training speed, as recommended by Ioffe & Szegedy (2015).

2.4. Method based on cGAN

Here, we also use conditional GAN as a contrast (Isola et al., 2016). Instead of learning a mapping from random noise vector z to output image, the conditional GANs learn a mapping from observed image s and random noise vector z to output image: $G : \{x, z\} \rightarrow y$. The objective of a conditional GAN can be expressed as

$$L_{cGAN}(G, D) = \mathbb{E}_{x,y \sim p_{data}(x,y)}[\log D(x, y)] + \mathbb{E}_{x \sim p_{data}(x), z \sim p_z(z)}[\log(1 - D(x, G(x, z)))]$$

3. Results

3.1. Baseline based on CNN

Using our CNN structure and the objective mentioned in subsection 2.2, we trained a network, and got output in Figure 4. This output is not desirable, which may be caused by several factors. First, the loss function based on Euclidean distance between the average

Figure 4. Output from a CNN-based colorization network

over the predicted and real pixels may not be fit for our purpose. This loss function may tend to average the output color, which would not lead to a photo-realistic reconstruction. Designing a reasonable loss function for a CNN that preserves photorealism may require significant domain knowledge related to image processing. Secondly, the size of our training data may limit the performance of our convolutional neural network; there is a large hypothesis class available in our CNN and very limited data to optimize the weights. The adversarial structure of GANs sidesteps this issue, but a direct colorization training on a CNN cannot do the same. Finally, the convolution-only structure of our CNN may cause difficulty when training in a colorization case from edges, because the structure of edges only rather than a more traditional colorization problem may restrict the ability of the CNN to learn colorization of features.

3.2. Method based on GAN

We initially attempted to use a simpler discriminator structure and minimal pretraining; while the results were far from photorealistic, they showed that the structure of the GAN was working. They also exhibited interesting artifacts at the edges and especially corners of the images, which signalled that the backgrounds might have been causing issues.

From upper left to lower right in Figure 5 are the rose

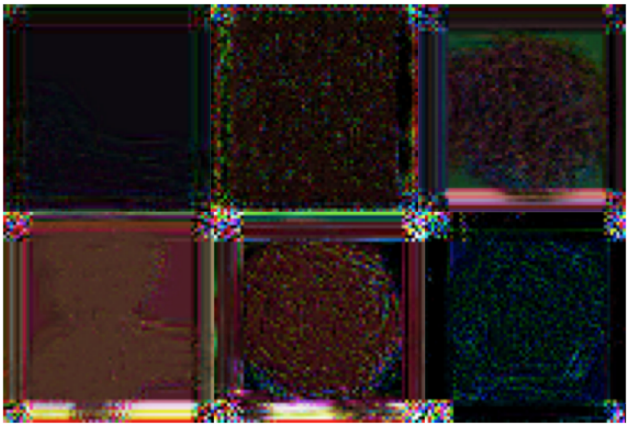

Figure 5. Outputs from an early generator, across many epochs

colorization outputs from the generator as we input to it an edge image at epoch 510, 1230, 1450, 1520, 2100, and 2550 respectively. (One epoch is defined as one forward-back propagation for the discriminator together with two forward-back propagations for the generator) Patterns of roses are emerging as the training going on from epoch 0 to 2100, but after that the output images become darker and chaotic. The experiment was repeated three times and similar patterns were gotten, where the pattern of rose was showing up but disappearing later on. A plot of the generator and discriminator losses is shown in Figure 6; notably, while the discriminator loss starts relatively high and continues to fall throughout the experiment, the generator loss never lowers in the same way.

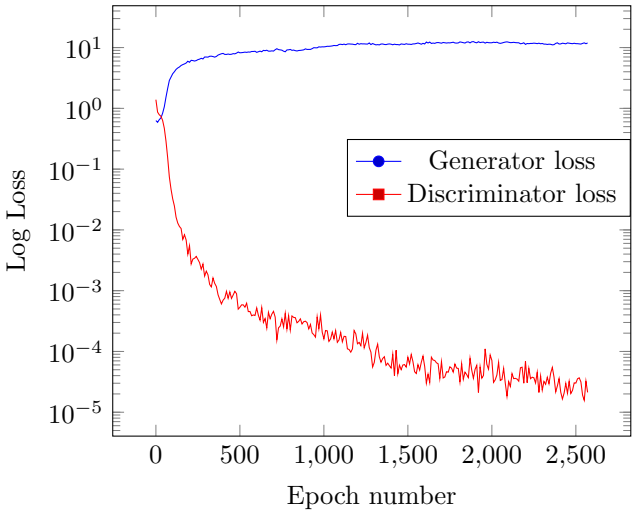

Figure 6. Loss curve for an experiment without pretraining

Our next attempt focused upon adding pretraining, as well as cleaning up the data from the above trial that we suspected was causing the edges and corners to have their distinctive artifacts. We added 100 minibatches of discriminator pretraining; instead of training with negative examples being created by the generator, we instead used noise on all 3 channels; our theory was that this would cause the classic CNN structure for object detection to emerge, so instead of the nearly uniform images we saw in Figure 5, the generator would be encouraged to produce images whose output was defined by the edges. This change was successful, as seen in Figure 7, as the generator network learned to follow the boundaries of the lines much more cleanly and had a strong distinction between the foreground and background.

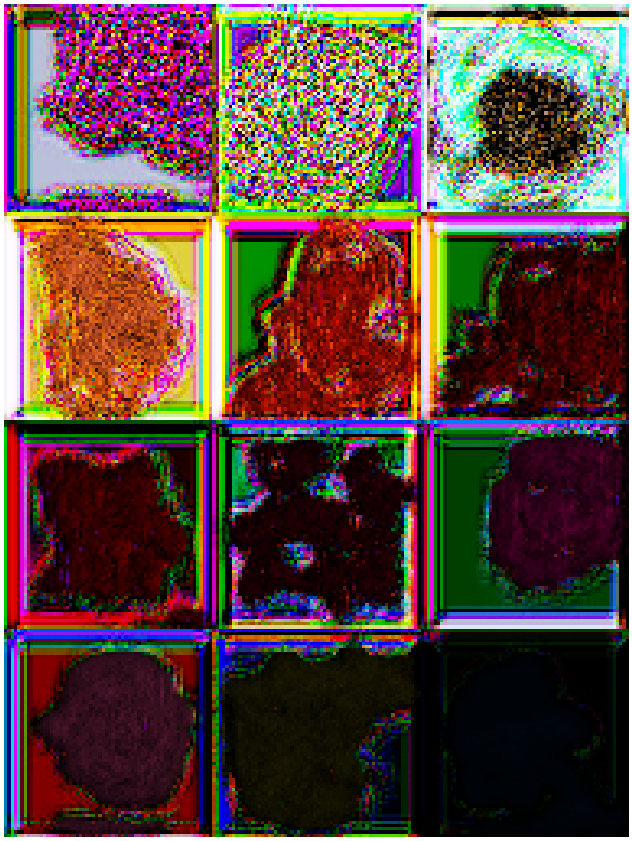

Figure 7. Outputs from a generator whose discriminator had been pretrained, across many epochs

Figure 7 shows the evolution of our pretrained network. We examine the images in chronological order, left to right, top to bottom. In the upper left at epoch 20, the structure of the rose is visible but the colorations have not yet converged to generate a textured image. In the next image, at epoch 120, there is some internal structure to the rose but the coloration has been lost to random noise; one theory as to the cause of this behavior is that the pretraining of the discriminator on random noise is backpropagated to the generator. In the third image, generated at epoch 220, the structure has further converged, and the background has been further contrasted with the rose, and the coloration within the rose has begun to converge away from the random noise into a dark red. These patterns continue into the fourth image, from epoch 270, where the rose is mostly red and the background is at a high contrast with the rose; however, the internal detail of the rose is not yet clear.

The best result from the network comes at epoch 320 in the fifth image, where the rose is red and some internal structure can be seen; however, the background is green, for reasons that are not clear to us. In images 6 through 10, there is a distinct red shape that fills where the rose should be (the edges of image 8 were extracted from a bouquet), but the overall image darkens and the contrast with the background worsens. These images represent epochs 370 through 570. Finally, in the last two images, taken at epochs 620 and 820, we see the end state of the generator network, where the output is completely darkened. We are not sure what causes this; one of our theories is that because the discriminator's performance is so high with almost 0 loss, it causes the generator's weights to be pushed to zero. This theory is based on the log-loss plot of the generator and discriminator losses in Figure 8.

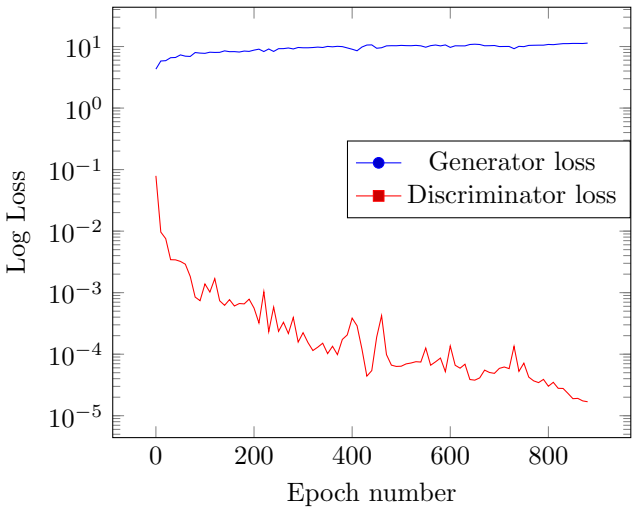

Figure 8. Loss curve for an experiment with pretraining

3.3. Method based on cGAN

We use the code on https://github.com/affinelayer/pix2pix-tensorflow to train the cGAN on our dataset. Here is the loss curve include the $L2$ distance loss and $L1$ distance loss of discriminator and generator, **??**.

And the following is the outputs of cGAN, Figure 9

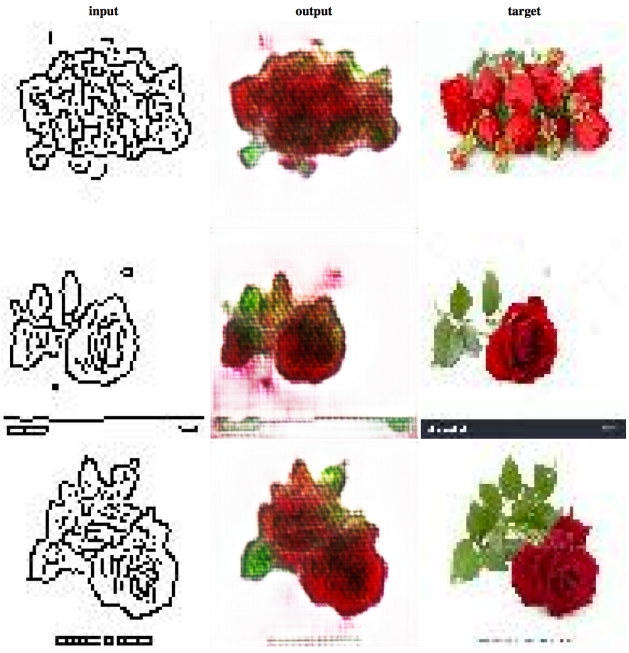

Figure 9. Outputs from cGAN

4. Discussion

Unfortunately, we were unable to achieve higher performance with our GAN architecture at colorization than we showed in Figure 7, despite extensive modification of hyperparameters. It is possible that the GAN architecture we used was not appropriate for the image colorization task that we chose; however, a similar structure was used in (Isola et al., 2016), which suggests otherwise. It is not clear, then, why the network consistently darkened after many iterations; the trend until darkening was generally quite promising.

5. Future Work

5.1. Better Architecture for GAN

It is possible that a different GAN architecture would be more powerful and work better for our image reconstruction problem; one particularly promising idea is to attempt to use a fully pretrained network like GoogLeNet (Szegedy et al., 2015) as the discriminator network; this would in theory allow the backpropagated gradients from the discriminator to be more strongly predictive of what a rose should look like. This sort of transfer learning is very promising in other

5.2. Simpler Concept

In retrospect, it may be instructive to attempt to recreate this project with a simpler concept to learn. This would bring many benefits, chief among them a lowering in computational time required to perform each test and thus ability to attempt more different network structures. The work in (Isola et al., 2016) uses handbags, which may be somewhat simpler than roses to represent due to a smaller amount of textural variations.

Acknowledgments

We would like to thank Professor David Page, whose slides on GANs helped us better understand the theoretical aspects and whose advice on how to write a conference-style paper helped immesurably; Professor Jude Shavlik, for insight into possible causes of the eventual darkening of our GANs, and the University of Wisconsin-Madison CS Department for allowing us to run experiments on instructional systems.

References

Canny, John. A computational approach to edge detection. *IEEE Transactions on pattern analysis and machine intelligence*, (6):679–698, 1986.

Deshpande, Aditya, Rock, Jason, and Forsyth, David. Learning large-scale automatic image colorization. In *Proceedings of the IEEE International Conference on Computer Vision*, pp. 567–575, 2015.

Goodfellow, Ian, Pouget-Abadie, Jean, Mirza, Mehdi, Xu, Bing, Warde-Farley, David, Ozair, Sherjil, Courville, Aaron, and Bengio, Yoshua. Generative adversarial nets. In *Advances in neural information processing systems*, pp. 2672–2680, 2014.

Goodfellow, Ian J. NIPS 2016 tutorial: Generative adversarial networks. *CoRR*, abs/1701.00160, 2017. URL http://arxiv.org/abs/1701.00160.

He, Kaiming, Zhang, Xiangyu, Ren, Shaoqing, and Sun, Jian. Delving deep into rectifiers: Surpassing human-level performance on imagenet classification. In *Proceedings of the IEEE international conference on computer vision*, pp. 1026–1034, 2015.

Ioffe, Sergey and Szegedy, Christian. Batch normalization: Accelerating deep network training by reducing internal covariate shift. In *Proceedings of The 32nd International Conference on Machine Learning*, pp. 448–456, 2015.

Isola, Phillip, Zhu, Jun-Yan, Zhou, Tinghui, and Efros, Alexei A. Image-to-image translation with conditional adversarial networks. *arXiv preprint arXiv:1611.07004*, 2016.

Krizhevsky, Alex, Sutskever, Ilya, and Hinton, Geoffrey E. Imagenet classification with deep convolutional neural networks. In *Advances in neural information processing systems*, pp. 1097–1105, 2012.

Kuntimad, G and Ranganath, Heggere S. Perfect image segmentation using pulse coupled neural networks. *IEEE Transactions on Neural Networks*, 10 (3):591–598, 1999.

LeCun, Yann, Bottou, Léon, Bengio, Yoshua, and Haffner, Patrick. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, 1998.

Radford, Alec, Metz, Luke, and Chintala, Soumith. Unsupervised representation learning with deep convolutional generative adversarial networks. *arXiv preprint arXiv:1511.06434*, 2015.

Simonyan, Karen and Zisserman, Andrew. Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556*, 2014.

Szegedy, Christian, Liu, Wei, Jia, Yangqing, Sermanet, Pierre, Reed, Scott, Anguelov, Dragomir, Erhan, Dumitru, Vanhoucke, Vincent, and Rabinovich, Andrew. Going deeper with convolutions. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 1–9, 2015.

van der Walt, Stéfan, Schönberger, Johannes L., Nunez-Iglesias, Juan, Boulogne, François, Warner, Joshua D., Yager, Neil, Gouillart, Emmanuelle, Yu, Tony, and the scikit-image contributors. scikit-image: image processing in Python. *PeerJ*, 2:e453, 6 2014. ISSN 2167-8359. doi: 10.7717/peerj.453. URL http://dx.doi.org/10.7717/peerj.453.

Zhang, Richard, Isola, Phillip, and Efros, Alexei A. Colorful image colorization. In *European Conference on Computer Vision*, pp. 649–666. Springer, 2016.