Image Priors for Image Colorization

Presented to

Brandeis University

Department of Computer Science

James Storer, Advisor

by

Talie Massachi

May, 2018

Contents

List of Figures

Acknowledgements

There are a few people without whom I would not have been able to complete this thesis.

First, Nick Moran, Aaditya Prakash, and Solomon Garber, thank you for teaching me so many different things, from Tensorflow to how to write a paper. Without your help and patient guidance I would have been lost during this whole process.

Professor Storer, thank you for giving me the opportunity to do this thesis. I have learned so much from my senior research this past year. I would not have had the chance if not for you.

My family and friends, especially my parents, Ryan Marcus, Liana Simpson, Kiana Khozein, and Seth Rait, thank you so much for providing the support and encouragement I needed to complete this thesis.

CHAPTER 1

Introduction

The idea of image colorization is nearly, if not as old as the existence of photography. Stretching all the way back to Isenring's hand-coloring of daguerreotypes [**10**]. Though today we have color photography, there are still artists that hand-color black and white photographs. These artists argue that bringing color to old black and white pictures allows modern people to associate more with the subject of a picture [**23**]. There is some controversy regarding this, as some people argue that black and white images are composed in the way they are partially because of the fact that they are colorless, and by colorizing them some of the artistic merit is lost [**18**]. Regardless of this, when we take into account the more recent context of images used in machine learning, we find a few more tangible uses.

There are numerous machine-learning-based image classifiers used today, however most of these classifiers work best on colored images. As stated in [**25**], when classifying the 10 thousand images in the ImageNet validation set, the standard VGG-16 network [**22**] finds the correct classification as its top classification 68.3% of the time. When those same images are changed to black and white and fed to the same network, accuracy drops to 52.7% [**25**]. Theoretically, by colorizing a black and white image, would would improve its chances of being classified correctly [**25**].

Consider also that space constraints continue to be an issue, especially when it comes to image and video data. If an image can be stored in black and white, and

a colorization function run on it to restore it to its original version, then the storage requirements for that image could be reduced.

In this thesis, I explore the constraints of an existing colorization system [**25**], made to produce plausible hallucinations. I explore how modifications to the training, and the network affect the performance of the system. I then explore different priors and the types of effects they produce in the system's output.

CHAPTER 2

Definitions

2.1. Lab Color Model

There are many different systems used to encode color information in digital images and videos. The most familiar color models are RGB (red, green, blue), the model used to display colors on most computers, and CMYK (cyan, magenta, yellow, black) which is used in most printers. In this case specifically, we are using the CIELAB color model, commonly referred to as Lab, (lightness, a-channel, b-channel). The Lab color model covers a broader range of colors than RGB [**6**], and, like the eye [**16**], it has a separate channel for brightness, completely separate from the channels used for colors [**6**]. This brightness channel is the L channel in Lab, and the a and b channels represent a sliding scale from red to green and blue to yellow respectively, as shown in figure 2.1 [**15**]. In Lab, L values have the range [0, 100], while the a and b channels have values in a range of approximately (-110, 100), where the actual range in the gamut changes depending on the L channel. This is demonstrated in figure 3.2, where we can see that the available a and b values when L is in the range [0, 25] is significantly different from the possible values when L is in the range [50, 75]. We use the Lab color model because it conveniently provides a single value per pixel in a black and white image, the L channel. We can then produce and overlay a and b values to get a colorized image.

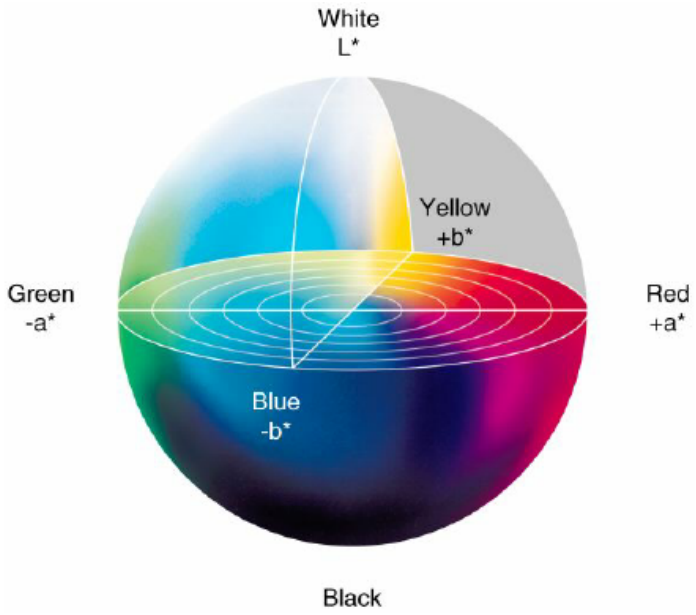

FIGURE 2.1. An approximation of the CIE L*a*b* color space. Credit [**19**]

2.2. Artificial Neural Networks

Artificial neural networks are, very generally, a series of transformation functions. They take in some input, x, put it through a series of mathematical functions, f, and get an output, y, which can represent any number of things. Rewritten this gives us y= f(x). In practice they are quite a bit more complicated, but this provides a simpler picture to work off of.

2.2.1. Neurons. To be more specific, we can start by considering just one neuron of the network. A neuron in the human brain very generally works by first taking in a large number of inputs from its dendrites. When the power of these combined input signals pushes above a specific threshold, the neuron fires its own signal down its axon and fires this signal towards the next neurons in the network. Again, the reality is much more complicated than described here, but this gives an idea of what a neuron in an artificial neural network is doing. An artificial neuron takes in a set of

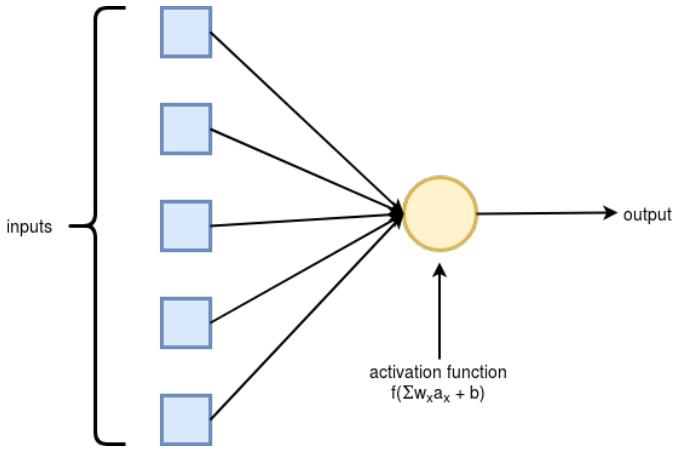

FIGURE 2.2. Each input has a value a_x and each corresponding arrow has a weight w_x where $x \in 1, 2, 3, 4, 5$

inputs, say a_1 and a_2. Each input value is multiplied with some weight, w_x, to give each input a final value of $w_x a_x$. These values are summed, and a bias term is then added. Finally, some function, f, is applied to this value, giving the final equation

$$y = f(\sum_x w_x a_x + b)$$

The result, y, is then treated as an input for the next neuron or neurons in the sequence.

Generally, we see more than one neuron, arranged into a network, such as the one shown figure 2.5. In a network such as this, it is the weights, represented here by solid arrows, that are changed as the network searches for a solution.

2.2.2. Activation Functions. As mentioned previously, when calculating the value of a neuron, a function is applied to the sum of a bias value and each weight multiplied by its corresponding input value. This function is generally called an activation function.

There are any number of activation functions, most trying to mimic the way that neurons fire. That is to say, having a steep slope once the input value hits a certain number. One popular function is the rectified linear unit, or ReLU, which is defined

$$f(x) = max(0, x)$$

In this case, the "threshold" after which the neuron "fires" is when $f(x)$ becomes positive.

Other popular functions include the scaled exponential linear unit, or SELU, defined as

$$f(x) = \lambda \begin{cases} x & \text{if } x > 0 \\ \alpha e^x - \alpha & \text{if } x \leq 0 \end{cases}$$

sigmoid, defined

$$f(x) = \frac{1}{1 + e^{-x}}$$

and tanh, defined

$$f(x) = \frac{2}{1 + e^{-2x}} - 1$$

2.3. Gradient Descent

Changes to the weights of a neural network are made using a number of different processes, the most common being gradient descent. Put simply, once a neural network makes a guess at a solution given an input, we find the difference, called loss, between this guess and the true solution. Gradient descent takes this loss value and changes the weights of the network slightly so that, should the same inputs be given to the network again, the loss would decrease. That is to say, the proposed solution would be slightly closer to the true solution.

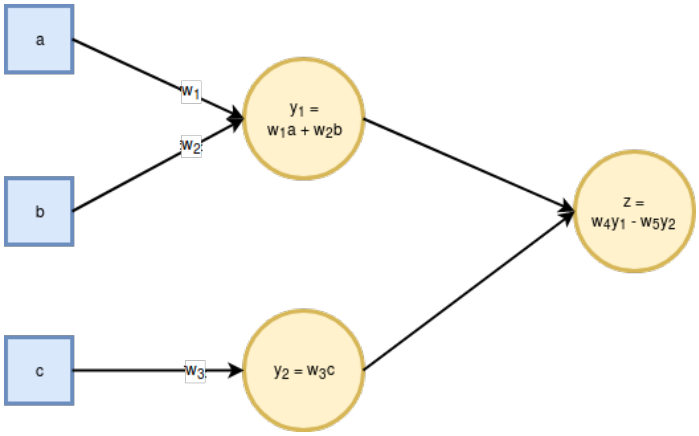

FIGURE 2.3. A simple neural network

Figure 2.3 is an example of a very simple neural network. In it, the weights being trained are w_1, w_2, and w_3. Suppose that we randomly initialize the weights of the network to be $w_1 = 1$, $w_2 = 2$, and $w_3 = 1$. Let us then say that we give the network input values of $a = 5$, $b = 2$, and $c = 3$, and that the correct output is 10. When we give our inputs to the network, it will propose a solution based on its current weights.

$$y_1 = w_1 a + w_2 b = 1 \times 5 + 2 \times 2 = 9$$

$$y_2 = w_3 c = 1 \times 3 = 3$$

$$z = y_1 - y_2 = 9 - 3 = 6$$

The difference, L, between the correct value and the guess our neural network made is $L = |10 - 6| = 4$. We want this loss to be as low as possible, so, in this case, we need to increase our output of 6. Remember that a derivative $\frac{ds}{dt}$ finds the change in ds given a positive change in dt. Therefore, if we find each derivative $\frac{dL}{dw_x}$ $x \in 1, 2, 3$ then we can find what change in each w_x will cause a positive change in the loss.

$$\frac{dL}{dw_1} = \frac{d}{dw_1}(10 - (w_1 a + w_2 b - w_3 c)) = -a$$

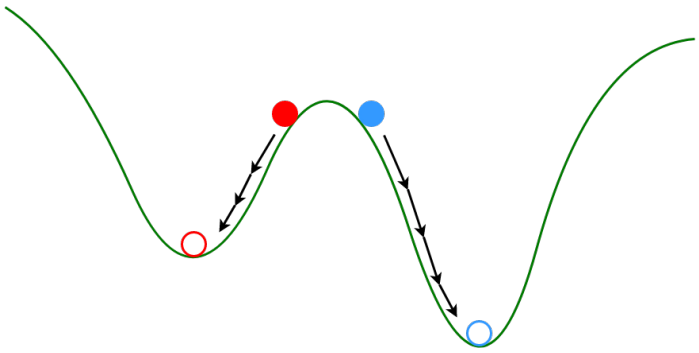

FIGURE 2.4. Filled circles represent starting points, and unfilled circles represent ending points for each ball.

$$\frac{dL}{dw_2} = \frac{d}{dw_2}(10 - (w_1a + w_2b - w_3c)) = -b$$
$$\frac{dL}{dw_3} = \frac{d}{dw_3}(10 - (w_1a + w_2b - w_3c)) = c$$

For w_1 and w_2 we get a negative slope. Since we want to decrease the loss, we want to increase the weights slightly. On the other hand, w_3 has a negative slope so we want to decrease it slightly. Usually, we would handpick a value before-hand, called the learning rate, and multiply that value by $-1 \times \frac{dL}{dw_x}$ to get the amount by which w_x is changed. This can be represented by the following equation, where η is the learning rate,

$$w_x = w_x - \eta \times \frac{dL}{dw_x}$$

In this case, let's just change each weight by .25. This leaves us with $w_1 = 1.25$, $w_2 = 2.25$, and $w_3 = .75$. If we recalculate the output from our neural network with our updated weights, we get a much closer number to the desired output of 10.

$$z = w_1a + w_2b - w_3c = 1.25 \times 5 + 2.25 \times 2 - .75 \times 3 = 8.5$$

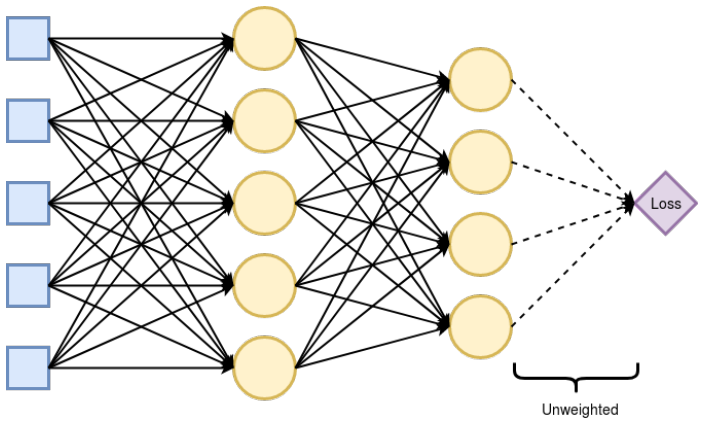

FIGURE 2.5. An example Artificial Neural Network. Blue boxes represent inputs, yellow circles represent neurons, and solid lines represent weights.

Consider for a moment that the way gradient descent works is similar to a ball rolling down a hill. The ball will roll in the direction of the steepest downward slope, only stopping when it no longer has a downward slope to roll down. The elevation of the land is equivalent to the loss, and each update of the weights that decreases the loss is like the ball rolling a little bit down in the hill. The end goal is to have the smallest possible loss, like finding the lowest elevation point of the hill. However, consider figure 2.4. You will notice that, though both balls follow the greatest downward slope from their starting points, the blue ball on the right ends up at a significantly lower elevation than the red ball on the left. Thought they are out of the scope of this thesis, there are modifications to the gradient descent algorithm that attempt to avoid this issue, including stochastic gradient descent and mini-batch gradient descent.

2.4. Backpropogation

Consider the neural network in figure 2.5. Given that each arrow has a weight (other than those labeled unweighted), if we were to try to train this network by hand

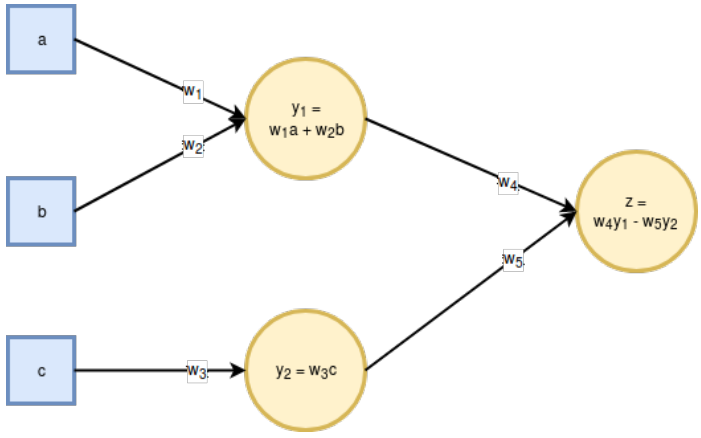

FIGURE 2.6. A basic neural network with inputs a, b, and c. Neuron values are notated inside of the neurons. Note that this network has no bias values.

using gradient descent then we would have to calculate the value $\frac{dL}{dw_x}$ for each arrow in the network. Given the number of weights in this network, it would take us a very long time to do so by hand. Once networks get to a certain size (potentially millions of weights) calculating each derivative individually can take too long to calculate, even for a computer. For this reason, we use a system called backpropogation to speed up the process. Simply put, backpropogation uses a few mathematical tricks to cut down on the number of computations necessary for gradient descent.

Consider figure 2.6, a slightly modified version of the network in figure 2.3. Say that for this network, loss is $L = (\widehat{Z} - z)^2$, where $\widehat{Z}$ is the true solution given the inputs. To update one weight, say w_1, we need to find $\frac{dL}{dw_1}$ where

$$\begin{aligned}\frac{dL}{dw_1} &= \frac{d}{dw_1}\left(\left(\widehat{Z} - (w_4y_1 - w_5y_2)\right)^2\right) \\ &= \frac{d}{dw_1}\left(\left(\widehat{Z} - (w_4(w_1a + w_2b) - w_5(w_3c))\right)^2\right)\end{aligned}$$

Alternatively, we can use the chain rule, which states that $\frac{ds}{dt} = \frac{da}{dt} \times \frac{ds}{da}$. To describe this in words: the change in s caused by a change in t can also be described as the change in a caused by a change in t times the change in s caused by a change in a.

If we use the chain rule to solve for $\frac{dL}{dw_1}$ we get the following

$$\begin{aligned}\frac{dL}{dw_1} &= \frac{dy_1}{dw_1} \times \frac{dL}{dy_1} \\ &= \frac{dy_1}{dw_1} \times \frac{dz}{dy_1} \times \frac{dL}{dz}\end{aligned}$$

Similarly,

$$\begin{aligned}\frac{dL}{dw_2} &= \frac{dy_1}{dw_2} \times \frac{dL}{dy_1} \\ &= \frac{dy_1}{dw_2} \times \frac{dz}{dy_1} \times \frac{dL}{dz}\end{aligned}$$

We can use the same process to find $\frac{dL}{dw_3}$,

$$\begin{aligned}\frac{dL}{dw_3} &= \frac{dy_2}{dw_3} \times \frac{dL}{dy_2} \\ &= \frac{dy_2}{dw_3} \times \frac{dz}{dy_2} \times \frac{dL}{dz}\end{aligned}$$

Following this pattern on the next layer, we can apply the same logic to w_4 and w_5.

$$\frac{dL}{dw_4} = \frac{dz}{dw_4} \times \frac{dL}{dz}$$

$$\frac{dL}{dw_5} = \frac{dz}{dw_5} \times \frac{dL}{dz}$$

Notice that all of these equations contain the term $\frac{dL}{dz}$, and $\frac{dL}{dw_1}$ and $\frac{dL}{dw_2}$ both contain $\frac{dz}{dy_1}$.

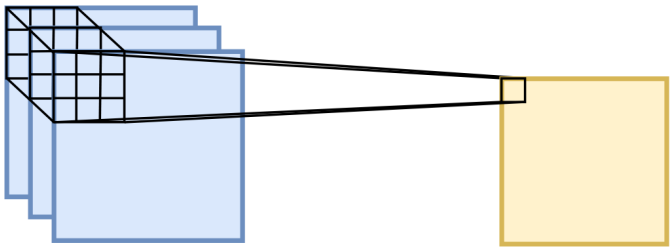

FIGURE 2.7. A single step in a convolution. The three blue squares on the left represent the input, while the yellow square on the right represents the output. The grid on the left squares is the area of the convolution, where each small square has a weight. The sum of the dot products of each of these weights and the values they cover is placed in the marked location of the output.

Therefore, consider that if we first solve for the value containing the last layer of the network, that is, $\frac{dL}{dz}$, then we can store that value and use it later while calculating the other derivatives we calculate for gradient descent. Furthermore, if we go in the exact opposite direction of the feed-forward network, starting from the last layer, then the second to last, etc. then we can avoid a large percentage of the total calculations we would have otherwise had to do.

Backpropogation is this process of first calculating the values for the last layers of the network and moving in opposite order while computing derivatives.

2.5. Convolution

Convolutional neural networks are a modification on the standard structure of neural networks that has already been discussed. Consider that in the previous discussion, neurons take some numeric input, run some function on that input, and then output a numeric value. This process lends itself well to finding functions or distributions. Convolution is formed such that it lends itself well for image processing.

Consider figure 2.7. The black grid on the stack of thee blue squares on the left represents a convolutional filter on a starting image. There are three squares representing the image because most images have three channels (for example RGB or Lab). Each small square in the grid contains a weight that is multiplied by the pixel value that it covers. Figure 2.7 therefore shows a filter with 27 weights. Similarly to before, each pixel of the image covered by the filter is multiplied by the weight in that spot of the filter. All of these values are then summed. Like in a normal neural network, a function is then applied to this sum and the resulting value is placed in the corresponding spot in a new matrix. This can also be described by the following equation.

$$Output = \sum_{l=A}^{C} \sum_{x=1}^{9} W_{l_x} l_x$$

Where l is some layer of the input and x is some pixel currently within range of the filter. x has a maximum of nine because we are assuming a 3 by 3 pixel filter in this case, but that size is variable.

You will notice that the output of the convolution in figure 2.7 is smaller than the input. This is because each output pixel is the result of the filter covering a larger area of the input (in this case 3 by 3). The filter shifts over one pixel to produce the next pixel in the output, but because it cannot go beyond the edges of the input, the output becomes smaller. To avoid this issue, inputs are often padded, that is, surrounded, with either zeros or repeats of the values at the edge of the input. This allows the filter to go beyond the edge of the input, producing an output the same size as the input.

2.6. Image Quality Metrics

There are a number of metrics used to test image quality. While most are outside of the scope of this thesis, there are three relevant metrics: mean squared error (MSE), peak-signal-to-noise ratio (PSNR), and PSNR-HVS-M.

All of these metrics produce a score giving the similarity between an original image and a modified version of the image. They are often used to see how well a compression technique works.

MSE, as its name implies, simply finds the average squared difference between corresponding pixels over an image. It is defined

$$MSE = \frac{1}{n}\sum_{i=1}^{n}(Y_i - \hat{Y}_i)^2$$

Where there are n pixels in an image, $\hat{Y}$ is the modified image, and Y is the original image. Thus, the intent is to minimize MSE.

Though MSE is simple and makes intuitive sense, it is not optimized for images. Thus, PSNR is more often used as a metric of image quality, as it was created to be one. For this reason as well, we will not be reporting MSE in this thesis. PSNR is, however, defined using MSE:

$$PSNR = 20 \times log_10(MAX_I) - 10 \times log_10(MSE)$$

Where MAX_I is the maximum possible pixel value of the image. Notice that we are subtracting the MSE, so we want to maximize PSNR.

After the creation of PSNR, a few improvements were made. The first, PSNR-HVS, modified PSNR to take into account human visual system (HVS) properties [**5**]. This was taken a step further with PSNR-HVS-M, which also takes into account visual masking, and was found to be significantly more accurate than PSNR [**20**].

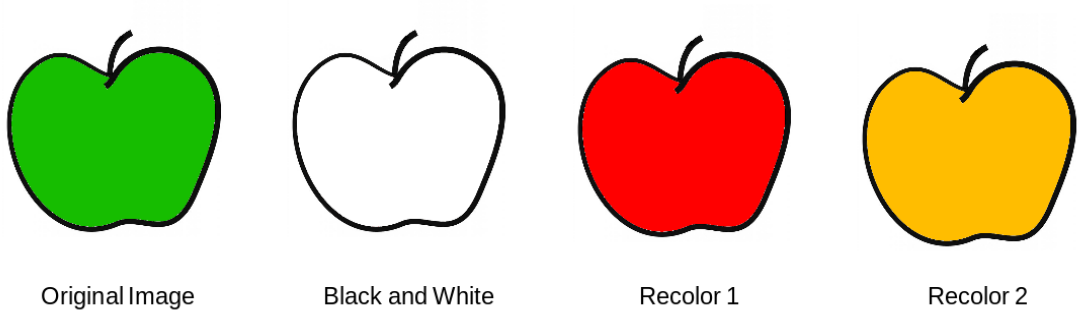

FIGURE 2.8. An original image of an apple, the same image with color removed, and two recolors.

TABLE 1. PSNR and PSNR-HVSM of figure 2.8

	PSNR	PSNR-HVSM
Original Image	inf	100,000
Color Removed	9.4	5.1
Recolor 1	19.9	15.7
Recolor 2	15.4	11.1

2.7. Color Hallucination

Consider that, even to humans, coloring a black and white image doesn't have only one correct answer. An apple in a gray-scale image, for example, could have originally been red, yellow, or green. Any of these options would have made sense given that we know it is an apple, but only one is technically true according to the reality of what color the apple was.

When color is hallucinated, we are using this exact idea. The true color of the object we are trying to colorize doesn't matter so much as whether the colorization makes sense.

It is important to note that when color is hallucinated, metrics such as PSNR, and PSNR-HVS-M don't quite make sense. These metrics take color into account when finding the difference between two images. Consider figure 2.8. In this case,

both of the recolors as well as the original colorization of the apple are all reasonable and acceptable colorizations of the black and white image. However, when we find the PSNR and PSNR-HVS-M of all versions of the image, as compared to the original image, only the original colorization gets a good score, as shown in table 1.

CHAPTER 3

Background

3.1. General Background

There are many papers investigating automatic image colorization. Early attempts required user input to gain a general direction for the colorization [**13**] [**7**]. There are also more recent methods that similarly take user input of color patches or pixels [**26**], but the general trend has moved away from this concept.

Later techniques used other, user-provided related images as a basis for colorization of a new image [**4**] [**17**] [**9**] [**24**] [**21**], avoiding the need for users to provide color suggestions directly on the image to be colorized. User input was further reduced when techniques were suggested to find these reference images either through a web search or by providing their own set of general reference images [**14**] [**2**] [**3**].

More recently, there has been a push into using neural networks to colorize images without any user input [**25**] [**12**] [**8**] [**1**] [**11**].

This includes systems that use more than one network [**12**] [**8**], as well as those that are a singular system across the board [**25**] [**1**] [**11**].

Of those systems that use multiple networks, some extract information from the weights of one network and use that information to colorize. For example [**12**], where images are classified using a standard VGG-16 network [**22**] and hypercolumns from the VGG network are used as the input data for a colorization network. A hypercolumn here being a tensor containing the values found for a single pixel across all layers of a convolutional neural net. There are also systems that use multiple networks in

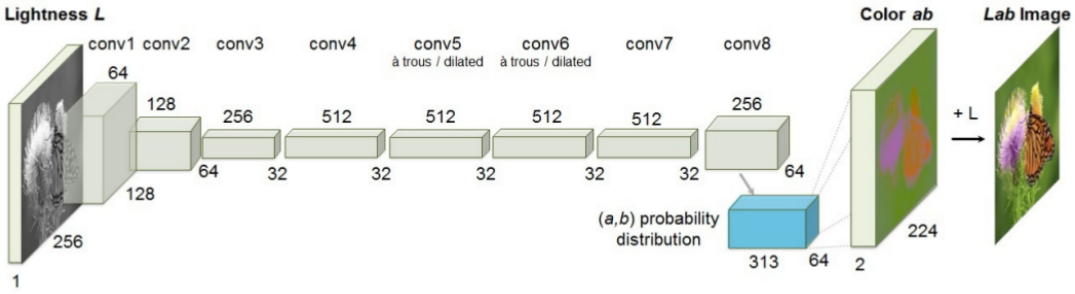

FIGURE 3.1. Network architecture as depicted in [**25**]

order to extract different data, such as [**8**], where there are separate networks for extracting low, mid, and high level data.

Of those systems that are a single network, there are many that use CNNs, such as [**25**] [**1**], however there are systems which use other structures, such as [**11**], which uses cGANs.

3.2. Colorful Image Colorization

One specific approach for image colorization [**25**] is used as the basis for this thesis. This specific system takes images using the Lab model for their color space as input and output. The greyscale input is simply the one (L) channel of the image, with the neural network trying to hallucinate the other two (a and b) channels based on that input.

The overall structure of this network is fully convolutional. That is, it contains only convolutional and batch normalization layers. Figure 3.1 shows the general structure of the network as provided in [**25**].

While this seems to imply that there are eight convolutional layers in the structure, with the blue box representing some kind of different layer type, the true architecture is a bit more complicated. Each "conv" layer in the depicted architecture is, in reality,

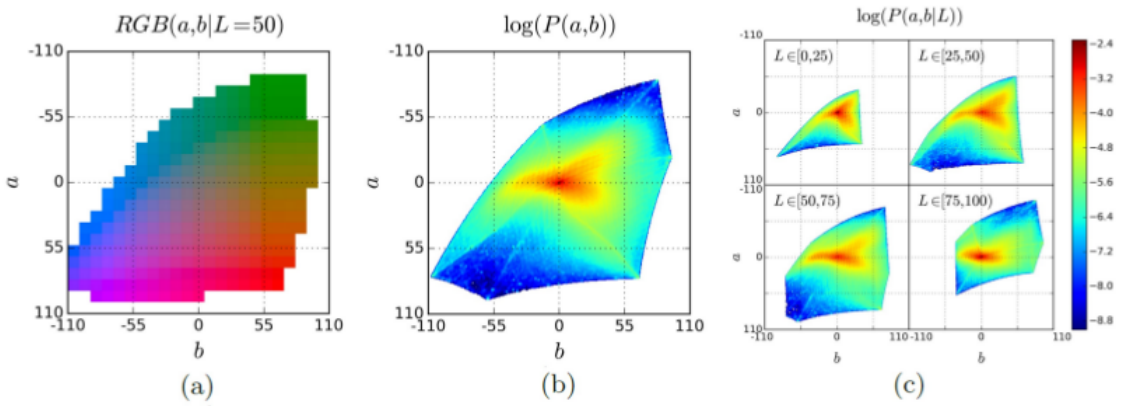

FIGURE 3.2. (a) Quantized ab color space with a grid size of 10. A total of 313 ab pairs are in gamut. (b) Empirical probability distribution of ab values, shown in log scale. (c) Empirical probability distribution of ab values, conditioned on L, shown in log scale. Credit [**25**]

either two or three convolutional layers followed by a batch normalization layer. The final layer, shown in blue in figure 3.1 is a 1x1 convolutional layer.

In this system, colorization is being treated as a classification task. Where every pixel of the image is being classified to a color [**25**]. Therefore, there must be a set of classes that the pixels can get classified to.

If we take the entire space of possible colors in the Lab model and flatten it such that we can see every possible (a, b) pair across all values of L, then we will get approximately the shape shown in figure 3.2. In order to get a set of discrete classes, this space is tiled with 10x10 squares, and each is treated as a potential (a,b) pair. By using this quantization we end up with a set, Q, of 313 possible (a,b) pairs.

Note that calculating a loss value in classification relies on being able to compare the probability distribution the neural network gives for a specific input to the true distribution. However, in this case, we are quantizing the output possibilities yet the true color value for any given pixel is not necessarily one of these 313 chosen value pairs in Q.

We therefore need to translate the true color value into a "ground truth" that uses the quantized color values. To do this, when producing the ground truth used in the loss, the 5 closest pairs in Q are found. The values for each of these pairs is determined by a Gaussian distribution, with standard deviation of 5.

While training, this network uses the following loss function:.

$$L_{cl}(\widehat{Z}, Z) = -\sum_{h,w} v(Z_{h,w}) \sum_{q} Z_{h,w,q} \log(\widehat{Z}_{h,w,q})$$

Where $\widehat{Z}$ is the prediction of the network, and Z is the calculated "ground truth" mentioned earlier. A pixel is defined by its height h and width w, and $q \in Q$. Also note that:

$$v(Z_{h,w}) = w_{q*}$$

$$q^* = argmax(q(Z_{h,w}))$$

$$w = \left(.5p^* + \frac{.5}{313}\right)^{-1}$$

$v(Z_{h,w})$ finds the weighting factor for the most highly predicted (a,b) pair for that pixel, where the weighting factor is related to a distribution p*. p* is, in turn, a smoothed distribution of the probabilities of each (a,b) pair appearing over a set of images. That is to say, the frequency with which each (a,b) pair appears over a set of images, also known as the prior probabilities for that set of images.

An interesting method is additionally used when producing the outputs that are compared to the ground truth values and used in this loss function. Rather than simply selecting the mean or mode of the distribution to find the (a,b) value for a pixel, as is common in networks such as this one, the *annealed-mean* is used.

FIGURE 3.3. The effect of temperature parameter T on the annealed-mean output. The left-most images show the means of the predicted color distributions and the right-most show the modes. T = 0.38 is used in this system. Credit [**25**]

The annealed-mean is defined as

$$H(Z_{h,w}) = \mathbb{E}[f_T(Z_{h,w})]$$

$$f_T(z) = \frac{exp(log(z)/T)}{\sum_q exp(log(z_q)/T)}$$

Essentially, distribution is modified before the average is taken. The distribution is left unchanged when $T = 1$, and the distribution simply chooses the mode when $T \to 0$. In other words, the peaks of the distribution become more pronounced as T decreases from 1 to 0. In this model, $T = 0.38$.

As shown in figure 3.3, when using the mean ($T = 1$) of the distribution to produce the output colors for an image, the colors tend to be a bit dull and washed out. If we consider that the final distribution for a pixel likely has a small probability for every value in Q, a simple average of all of these values is likely to shift towards brown. At the other extreme, when using the mode ($T = 0$), there is potential for a gradual shift

between one color and other. This has the potential to output colorizations with odd spots of sharp color changes, as shown in the rightmost column of figure 3.3.

CHAPTER 4

Methods and Results

4.1. Using a Smaller Dataset

In [**25**], one argument given for why the system works so well is the dataset they use for training; millions of images from ImageNet. However, the ImageNet training set includes hundreds of different objects and images of many different sizes. It seems reasonable that by restricting the range of objects and sizes of the images we use in training, we can train an equally effective network with fewer images. To this end, we used a set of 7985 images of flowers for training, all of the same size. A similar but distinct set of 204 flower images was used as a test set.

To gain a baseline, we can see how well the network trained on the ImageNet images colorizes a few random images, as well as the images from the flower test set. As we can see in figure 4.1, while the original network does best on landscape images, it does fairly well on all the images shown. The third row of images in figure 4.2 shows a selection of test images from the flowers test set of size 64x64 colored using only information from the ImageNet dataset. These images come out fairly uniformly green. It is likely that, though there are some images of flowers in the ImageNet dataset, there are not enough to make a significant change in the network weights. It is also likely that, since the network was trained on mostly images of a larger size, any flowers of a similar size were simply blended into the background of the image, which in most cases would be green grass.

FIGURE 4.1. Row 1: original gray-scale images. Row 2: recolored images using ImageNet training. Row 3: recolored images using Flowers training.

FIGURE 4.2. Row 1: original full-color images. Row 2: original images shown in gray-scale, as they were passed as input to the network. Row 3: images recolored images using ImageNet training. Row 4: images recolored using Flowers training.

We can compare both of these tests to images colorized by a network trained only on 64x64 images of flowers, as shown in figures 4.1 and 4.2. In the larger images in figure 4.1, it is clear that the flower-trained network had some trouble. Overall, the images are red tinted, with splashes of yellow and green, and small areas of blue. This is possibly also due to the size differences. The network trained on flower images wouldn't be used to large areas blue or green as the sky or grass in the background would take up very few pixels comparatively, given the image size. On the flower images, shown in the last row of figure 4.2, we can see that this new network uses brighter colors than the ImageNet-trained network. Furthermore, the variety of colors within each image is much higher in the flowers-trained network, and the network is able to detect the edges between the flower petals and the background of the image, unlike the original network.

4.2. Fine Tuning vs. Training from Scratch

We have established that the ImageNet network is unable to effectively color the flower images we are focusing on. However, consider that there could be some underlying shared information between the sets of images. On one hand this could include low-level information, such as edge detection, but it could also simply be that the final weights of the ImageNet network are closer to the ideal weights that will solve our problem than randomly selected starting weights. Therefore if we train using the ImageNet weights as our starting weights, we may get output as accurate as training from scratch, but with fewer steps of training. This process is called fine tuning.

The fourth row of figure 4.3 shows the output of a network using this technique. You may notice that the consistency of colors in the fine-tuned network is poor. In the first column, for example, the petals of the flower fluctuate between pink and green.

FIGURE 4.3. Row 1: original images. Row 2: original images in gray-scale. Row 3: images recolored having trained from scratch. Row 4: images recolored using fine-tuning.

It is interesting to note that when the network trained only on the ImageNet images was used to colorize these images of flowers, the output was more or less uniformly green, as shown in figure 4.2. This output shows the opposite problem.

A possibility for why this happened is that in the ImageNet images the colors that are shown in the same images here are often found near each other. For example, perhaps pink is often seen near green. Another possibility is that, given the different image sizes used in the ImageNet training and this training, the potentially shared information is made irrelevant, meaning that there may be a larger numerical difference between where the weights start and the weights that minimize loss. In other words, the network may simply need to train longer to get a comparable result to when we trained from scratch.

FIGURE 4.4. Row 1: original images. Row 2: images in gray-scale. Row 3: images recolored using a network trained on images of size 128x128 pixels.

TABLE 1. Average* PSNR and PSNR-HVSM by Training Image Size

	PSNR	PSNR-HVSM
64x64 pixel images	31.1	27.4
128x128 pixel images	28.7	26.7

*Average values are found across 204 test images

4.3. Changing Image Size

As mentioned previously, the initial use of this network was for images in ImageNet, which are generally larger images than those that we have been using to train thus far. It is perhaps possible that the network will work better on larger images for training and testing. Figure 4.4 shows 128 by 128 pixel images of flowers recolored by the network trained on images of the same size. Though the images are clearly of a higher quality, there are clear points at which the network fails to maintain a consistent color across one object in the image. Most noticeably, some flowers have petals that start one color, yet change to a different color at another point in the image. However, there are also some cases where the larger images are colorized

more consistently than their smaller counterparts. Given this evidence, it seems to be unclear if it is better to use one image size over the other.

We know that the measures of PSNR and PSNR-HVSM are largely irrelevant in this case, as we are trying to hallucinate color rather than reproduce the original images. However, when we compare these measures for the two sets, as shown in table 1 we find that they are fairly close. Though the larger images do have marginally better measures, they are close enough that it doesn't seem worth the extra training time to use larger images in our experiments.

4.4. Changing Priors

As mentioned in section 3.2, this network uses prior probabilities while calculating loss, giving more weight to less common colors. As reported in [**25**], this is to avoid sepia-toned images.

Up until this point we have used the priors as they were calculated from the ImageNet set, but the ratio of each colors appearance may be different in our smaller dataset. The output may therefore be affected by changing the prior probabilities given to the network to probabilities corresponding to the training set of flower images. As shown in figure 4.5, when we use the new prior values the output is oddly washed out and green. There are two potential reasons for this.

Given that the output is so drastically different, it is clear that the probability distribution for the priors has changed. Intuitively, consider that the prior distribution for the ImageNet images covers a huge range of images and subjects, thus it is likely that the distribution produced from this image set is somewhat close to that of the true distribution. Rather, the distribution is close to what a human eye might see in day to day life. It may be that this system, roughly based on the human eye and mind, works best given information close to that of what the human eye might see.

FIGURE 4.5. Row 1: original images. Row 2: original images in gray-scale. Row 3: images recolored using a network trained with priors based on ImageNet. Row 4: images recolored using a network trained using priors based on Flowers dataset.

On a more mathematical note, it is likely that the prior probabilities based off of the flower images has a higher probability for common colors we see in flowers (e.g. red, yellow), and a lower probability for background colors (i.e. green), given that most of these images are taken very close to the flowers themselves. Consider that, given how prior probabilities affect the loss, this would boost the amount of green we see in the output.

Consider again the equation used for loss, as shown in section 3.2. In this function, $v(Z_{h,w})$ is large when the ground truth of the current pixel is an uncommon color for that pixel, and small when it guesses a common color. Therefore, pixels with common colors will cause a relatively small change in the loss while pixels with uncommon colors will cause a large change in the loss. In this case, however, green has become the uncommon color, while red and yellow are very common. This would cause pixels containing the "common" reds and yellows in the petals of the flowers to have only a small effect on the loss, while the background green pixels will have a larger effect on

FIGURE 4.6. Row 1: original images. Row 2: full quality L channel with downsampled color channel - used as a baseline. Row 3: image recolored using network trained with L channel and 16 color value inputs on a scale of -110 to 110. Row 4: images recolored using network trained with L channel and 16 color value inputs on a scale of -50 to 50.

the loss. In order to minimize loss, then, green becomes a color guessed more often by the network.

4.5. Adding Additional Color Information

Everything we've done so far has been producing plausible colorizations given a grayscale image. However, if this system is able to successfully colorize an image, given some specific guidance on correct vs. plausible colors, it may be able to reproduce a colorization closer to that of the original image. Much like what was done in [**1**]. With this intention, we modified the network to take input including both a full quality L channel and sixteen color values from severely down sampled a and b channels.

We retrained the network using this new input. First, the a and b values were given to the network on a scale of -110 to 110, the common scale used when rgb values

are translated to Lab. Then, the a and b values were given on a scale of -50 to 50, the same scale as the one used for the L channel.

As we can see in figure 4.6, both of these cases came out fairly off-colored from the original image. This is clearly reflected in the PSNR for each of these tests. If we simply downsample the a and b channels of the image to a four by four pixel square, and then overlay these values over the image, we get the result shown in the second row of figure 4.6, with an average PSNR of 38.3 and PSNR-HVSM of 35.3. When we use the trained network with input values between -110 and 110, we get an average PSNR of 33.3 and PSNR-HVSM of 29.6. When we again retrain, using input values between -50 and 50, the average PSNR is 31 and PSNR-HVSM is 27.26. As would be expected given the outputs we see, the PSNR is lower for our recolors than for the simple downsample baseline.

It is unclear why using a different range of values when giving the inputs causes such a drastic change in color. It may simply be that, given the different starting points when these networks were trained, they found different local minimums through training.

The reason for them being off-colored is, however, much more straightforward. Consider that, by adding the down sampled color channels to the input of the network, we are essentially giving the network an additional set of priors for each image. It is likely that the discoloration we see in our outputs is due to the different prior values working against each other, particularly because of their aforementioned use of the prior probabilities in the loss function.

4.6. Removing Dual Priors

Given that the previous tests resulted in an odd color-shift, likely due to competition between the multiple priors given to the network (i.e. the prior probabilities

FIGURE 4.7. Row 1: original images. Row 2: full quality L channel with downsampled color channel - used as a baseline. Row 3: images recolored using network trained with L channel and 16 color value inputs and uniform prior probabilities. Row 4: images recolored using network trained with L channel and 16 color value inputs and prior probabilities based on the Flower dataset.

TABLE 2. Average PSNR and PSNR-HVSM For Networks Trained on Different Priors

	PSNR	PSNR-HVSM
Trained Using ImageNet Priors*	31.0	27.3
Trained Using Uniform Priors	31.7	30.0
Trained Using Flower Priors	31.7	27.9
No Color Values in Input	31.1	27.4

*input color values have range -50 to 50

and the 16 color values in the input), it would make sense to then remove the prior probabilities causing changes to the loss function.

To this end, we set the prior probabilities to a uniform distribution. This removes the effect of the weighting in the loss, leaving it as simply cross entropy loss. As we can see in figure 4.7, the output images seem to be much closer in color to the original images, though color seems to bleed beyond edges quite a lot.

The prior probabilities were also set to the reflect the set of flower images, producing the output on the fourth line of figure 4.7. The color does not seem to be overly affected by this change.

It is likely that the reason for both the bleeding of color and the seemingly unchanged color between the training with uniform priors and flower priors is simply because the color values given as part of the input are such a strong prior that the network has trouble training away from them. We also notice that, before the priors were changed (in figure 4.6), the color bleeding is still present, though a different color.

This is also evident in that the PSNR does not change significantly when we change the priors (see table 2).

CHAPTER 5

Discussion

In this study, the effects of different priors on a well known colorization network were investigated. First, we established a baseline for the network on a different set of images. We then established the best strategy for training, and investigated the effects of several common training strategies on our results. Additionally, we investigated the effect of different priors on our output, changing both the priors and the form in which they are given to the network.

We found that the values of the prior probabilities had a great effect on the output, as expected of the network. When using priors such that one set of priors is used to affect all images, as opposed to priors only relevant to a single image, it appears best to use priors that mimic those of the human eye, regardless of our training set. Additionally, it is best not to use multiple types of priors in a single instance of training. It also seems that providing image-specific priors to the network creates too strong of a bias towards the given information, so the network is unable to improve upon the given input.

It would be interesting to investigate the output given a heavier downsampling of the color channels when the color data is given in the input. Additionally, it would be best to repeat each of these experiments, particularly the two using both sets of priors, to see how strongly the results are due to a bad starting point for the weights.

Bibliography

1. Mohammad Haris Baig and Lorenzo Torresani, *Colorization for image compression*, CoRR **abs/1606.06314** (2016).
2. Zezhou Cheng, Qingxiong Yang, and Bin Sheng, *Deep colorization*, CoRR **abs/1605.00075** (2016).
3. Alex Yong-Sang Chia, Shaojie Zhuo, Raj Kumar Gupta, Yu-Wing Tai, Siu-Yeung Cho, Ping Tan, and Stephen Lin, *Semantic colorization with internet images*, ACM Trans. Graph. **30** (2011), no. 6, 156:1–156:8.
4. Aditya Deshpande, Jason Rock, and David Forsyth, *Learning large-scale automatic image colorization*, ICCV, 2015.
5. Karen Egiazarian, Jaakko Astola, Nikolay N. Ponomarenko, Vladimir V. Lukin, Federica Battisti, and Marco Carli, *A new full-reference quality metrics based on hvs*, 2006.
6. Gernot Hoffmann, *Cielab color space*, `http://docs-hoffmann.de/cielab03022003.pdf`, Accessed 04-06-2018.
7. Yi-Chin Huang, Yi-Shin Tung, Jun-Cheng Chen, Sung-Wen Wang, and Ja-Ling Wu, *An adaptive edge detection based colorization algorithm and its applications*, Proceedings of the 13th Annual ACM International Conference on Multimedia (New York, NY, USA), MULTIMEDIA '05, ACM, 2005, pp. 351–354.
8. Satoshi Iizuka, Edgar Simo-Serra, and Hiroshi Ishikawa, *Let there be Color!: Joint End-to-end Learning of Global and Local Image Priors for Automatic Image Colorization with Simultaneous Classification*, ACM Transactions on Graphics (Proc. of SIGGRAPH 2016) **35** (2016), no. 4, 110:1–110:11.
9. Revital Irony, Daniel Cohen-Or, and Dani Lischinski, *Colorization by example*, Proceedings of the Sixteenth Eurographics Conference on Rendering Techniques (Aire-la-Ville, Switzerland, Switzerland), EGSR '05, Eurographics Association, 2005, pp. 201–210.
10. Johann Baptist Isenring, *Improvement in coloring daguerreotype-plates*, `https://patents.google.com/patent/US4369A/en`, January 1846, US Patent No. 4,369.
11. Phillip Isola, Jun-Yan Zhu, Tinghui Zhou, and Alexei A. Efros, *Image-to-image translation with conditional adversarial networks*, CoRR **abs/1611.07004** (2016).
12. Gustav Larsson, Michael Maire, and Gregory Shakhnarovich, *Learning representations for automatic colorization*, CoRR **abs/1603.06668** (2016).
13. Anat Levin, Dani Lischinski, and Yair Weiss, *Colorization using optimization*, ACM Trans. Graph. **23** (2004), no. 3, 689–694.
14. Xiaopei Liu, Liang Wan, Yingge Qu, Tien-Tsin Wong, Stephen Lin, Chi-Sing Leung, and Pheng-Ann Heng, *Intrinsic colorization*, ACM Transactions on Graphics (SIGGRAPH Asia 2008 issue) **27** (2008), no. 5, 152:1–152:9.
15. Bruce MacEvoy, *Modern color models*, `https://www.handprint.com/HP/WCL/color7.html`, August 2005, Accessed 04-05-2018.
16. Ethan Montag, *Rods and cones*, `https://www.cis.rit.edu/people/faculty/montag/vandplite/pages/chap_9/ch9p1.html`, Accessed 04-05-2018.

17. Yuji Morimoto, Yuichi Taguchi, and Takeshi Naemura, *Automatic colorization of grayscale images using multiple images on the web*, SIGGRAPH 2009: Talks (New York, NY, USA), SIGGRAPH '09, ACM, 2009, pp. 59:1–59:1.
18. Matt Novak, *Are colorized photos rewriting history?*, `https://paleofuture.gizmodo.com/are-colorized-photos-rewriting-history-1579276696`, Accessed 05-03-2018.
19. Pedro Pardo, J. Enrique Agudo, Hctor Snchez, A.L.Perez , and M.I.Suero , *A low-cost real color picker based on arduino*, **14** (2014), 11943–11956.
20. Nikolay Ponomarenko, Flavia Silvestri, Karen Egiazarian, Marco Carli, J Astola, and Vladimir Lukin, *On between-coefficient contrast masking of dct basis functions*, (2007).
21. Erik Reinhard, Michael Ashikhmin, Bruce Gooch, and Peter Shirley, *Color transfer between images*, **21** (2001), 34–41.
22. Karen Simonyan and Andrew Zisserman, *Very deep convolutional networks for large-scale image recognition*, CoRR **abs/1409.1556** (2014).
23. Meilan Solly, *Colorized footage is a vivid reminder that history didnt happen in black and white*, `https://www.smithsonianmag.com/history/what-did-1920s-look-color-180963882/`, Accessed 05-03-2018.
24. Tomihisa Welsh, Michael Ashikhmin, and Klaus Mueller, *Transferring color to greyscale images*, ACM Trans. Graph. **21** (2002), no. 3, 277–280.
25. Richard Zhang, Phillip Isola, and Alexei A. Efros, *Colorful image colorization*, CoRR **abs/1603.08511** (2016).
26. Richard Zhang, Jun-Yan Zhu, Phillip Isola, Xinyang Geng, Angela S. Lin, Tianhe Yu, and Alexei A. Efros, *Real-time user-guided image colorization with learned deep priors*, CoRR **abs/1705.02999** (2017).

Appendix

Input and Output for Network Trained on 64x64 Flower Images