Automatic Colorization with Deep Convolutional Generative Adversarial Networks

Stephen Koo
Stanford University
Stanford, CA
sckoo@cs.stanford.edu

Abstract

We attempt to use DCGANs (deep convolutional generative adversarial nets) to tackle the automatic colorization of black and white photos to combat the tendency for vanilla neural nets to "average out" the results. We construct a small feed-forward convolutional neural network as a baseline colorization system. We train the baseline model on the CIFAR-10 dataset with a per-pixel Euclidean loss function on the chrominance values and achieve sensible but mediocre results. We propose using the adversarial framework as proposed by Goodfellow et al. [5] as an alternative to the loss function—we reformulate the baseline model as a generator model that maps grayscale images and random noise input to the color image space, and construct a discriminator model that is trained to predict the probability that a given colorization was sampled from data distribution rather than generated by the generator model, conditioned on the grayscale image. We analyze the challenges that stand in the way of training adversarial networks, and suggest future steps to test the viability of the model.

1. Introduction

In this project, we tackle the problem of automatically colorizing grayscale images using deep convolutional generative adversarial networks (DCGANs)[14]. The input to our system is a grayscale image. We then use conditional convolutional generative adversarial networks to output a prediction of a realistic colorization of the image.

Colorization of old black and white photos provides an eye-opening window to visualizing and understanding the past. People, at least from my generation, tend to imagine any scene from the years between the invention of the photograph and the invention of the color photograph as black and white, when of course humans have always perceived color from the dawn of civilization. It stretches the imagination to mentally add color to those old timey photographs, but being able to picture 19th century London as anything other than a distant black and white caricature can enhance one's connection with, and appreciation for, history.

Figure 1: Left: the original black and white image. Center: image colorized by non-adversarial convolutional neural network. Right: image colorized by a human (Reddit). Images courtesy of Ryan Dahl [4].

Despite the lack of color information in black and white photos, humans are able to take contextual clues from the contents of the image to fill it in with realistic colors. This indicates that black and white images still contain latent information that may be sufficient for full colorizations, though a human may take time on the order of hours in order to fill in colors for a single photo. Convolutional neural networks (ConvNets), with their incredible results in extracting features from images, constitute a promising method for fast automatic colorization, which could be applied toward efficient colorization of black and white videos.

However, previous systems that use ConvNets toward automatic colorization tend to produce sepia-tone colors for objects for which the color might be ambiguous[4], even to human eyes. For example, a dark-looking car in a black and white photo could easily be dark green or dark red in reality, and a plain feed-forward ConvNet archiecture that uses a Euclidean distance loss function will tend to take a middle of the road solution, averaging the possible colors and thus outputting a brownish color (Figure 1).

To address this problem, we propose using the generative adversarial network framework, as first introduced by

Goodfellow et. al. last year[1]: in our case, the generative net will take as input the grayscale image in additional to some random noise, and generate color channels for the image. Meanwhile, the discriminator net will randomly be given either a generated colorization or a true color image, predicting whether or not the input was a true color image. While the discriminator net will try to maximize the accuracy of its predictions, the generative net will try to minimize the accuracy of the discriminator, leading to natural loss functions for backpropagation that do not depend on Euclidean distance measures while working to match the generated distribution of colors to the true distribution in the dataset. By modifying the objective to producing more realistic colorizations, rather than colorizations that are "close" to the training set, we hope that our system will produce brighter and more life-like colorizations.

2. Related Work

2.1. Hint-based colorization

Levin et al. [10] proposed a simple but effective method that incorporates colorization hints from the user in a quadratic cost function, imposing that neighboring pixels in space-time with similar intensities should have similar colors. The hints are provided in the form of imprecise colored "scribbles" on the grayscale input image, and with no additional information about the image, the method is able to efficiently generate high quality colorizations. Extensions of this approach have further improved its performance: Huang et al. [7] addressed color-bleeding issues using adaptive edge detection, [19] used luminance-based weighting of the user-supplied hints to boost efficiency for video applications, and Qu et al. [13] extended the cost function to enforce color continuity over similar textures in addition to similar intensities.

Welsh et al. [18] proposed an alternative approach that further reduces the burden on the user by only requiring a full-color example image of similar composition. By matching luminance and texture information between the example image and the target grayscale image, the algorithm achieves realistic results as long as a sufficiently similar image can be found to use as the example image.

Regardless of the degree of automation, however, both these "scribble"-based and example-based techniques require significant human assistance, in the form of hand-drawn color hints or suitable examples.

2.2. Deep colorization

In our proposed method, we aim to leverage the the large amount of image data available on the internet to fully automate the colorization process without any human intervention. Neural networks have shown great promise in learning a hierarchical model necessary for understanding images, and so we turn to neural networks. Cheng et al. [3] proposed training neural networks on per-pixel patch, DAISY [17], and semantic [2] features to predict the chrominance values for each pixel, with joint bilateral filtering to smooth out accidental image artifacts. When trained on a large-scale image database with a simple Euclidean loss function against the ground-truth chrominance values, this method resulted in equivalent or superior performance to example-based methods that used hand-selected examples.

Ryan Dahl [4] went a step further, forgoing the potentially limited image segmentation features and utilizing a *convolutional* neural network pretrained for image classification as a feature extractor in a novel residual-style architecture that directly outputs full color channels for the input image. Trained on the ImageNet database with a Euclidean loss function on the chrominance values, the approach achieved mixed results: the predicted colors were almost always reasonable, but also in general tended toward desaturated and even brownish colors. The Euclidean loss function in this case likely led to "averaging" of colors across similar objects.

2.3. Generative adversarial networks

First proposed by Goodfellow et al. [5], the adversarial modeling framework provides an approach to training a neural network model that estimates the generative distribution $p_g(x)$ over the input data x. Using neural networks networks as universal function approximators [1], we use neural network $G(z;\theta_g)$ with parameters θ_g to represent a mapping from input noise variable with distribution $p_z(z)$ to a point x in the data space, and use neural network $D(x;\theta_d)$ as mapping from point x in data space to probability that x came from the data rather than $G(z;\theta_g)$.

Radford et al. [14] applied the adversarial framework to training convolutional neural networks as generative models for images, demonstrating the viability of *deep convolutional generative adversarial networks* (DCGANs) with experiments on class-constrained datasets such as the LSUN bedrooms dataset and human faces scraped from the web.

In this proposal, we reformulate the DCGAN framework for a conditional generative modeling context, training $G(x_Y, z;\theta_g)$ to jointly map input noise z and grayscale image x_Y to the color image x, and training $D(x;\theta_d)$ to assign the correct label to generated colorizations and true colorizations.

3. Methods

We use the YUV colorspace instead of the RGB colorspace, since the YUV colorspace minimizes the per-pixel

[1] http://arxiv.org/abs/1406.2661

[2] Per-pixel category labels from a state-of-the-art scene parsing algorithm [11].

correlation between the color channels. The Y channel encodes the luminance component of the image, and can be intepreted as the grayscale version of the image, while the U and V color, or *chrominance*, channels encode the colors of the corresponding pixels. Our colorization systems thus take as input the Y component and outputs a prediction for the UV components. More formally: given a set of images X, where the full color image $x \in X$ is composed of grayscale and color components x_Y and x_{UV} respectively, we aim to build a model to predict x_{UV} given x_Y. Note that this may or may not match a user's actual objectives for colorization: given a grayscale image x_Y, there may be many *realistic* colorizations of the image that do not necessarily match the original colors x_{UV} of the image exactly. In fact, there may not be enough information in x_{UV} inherently to infer the corresponding colors unambiguously, regardless of the method. Thus we relax this objective to proposing realistic colors $\hat{x}_{UV}$ given x_Y in our adversarial model.

To determine the benefit of introducing the adversarial net framework to deep automatic colorization approaches, we first construct a simple baseline implementation, then incorporate the baseline implementation into an adversarial framework.

3.1. Baseline

Our baseline approach directly learns a mapping from the grayscale image space to the color image space. We train a convolutional neural network F as a mapping $\hat{x}_{UV} = F(x_Y; \theta)$, where $\hat{x}_{UV}$ is our estimate of x_{UV}.

We train our baseline model to minimize the Euclidean distance between the prediction $\hat{x}_{UV}$ and the ground-truth x_{UV}, averaged over the pixels. More formally, we aim to minimize this least-squares objective:

$$L(x; \theta) = \frac{1}{n} \sum_{p=1}^{n} \| F(x_Y; \theta)^{(p)} - x_{UV}^{(p)} \|_2^2 \qquad (1)$$

where the superscript $*^{(p)}$ represents the vector of components of the pth pixel of the image, and n is the total number of pixels in an image. Note that this corresponds with the objective of matching the original ground-truth colors exactly, and does not necessarily reward different but realistic colorizations.

3.1.1 Architecture

We use an entirely convolutional model architecture without any pooling or upscaling layers, such that the output images have the same spatial dimensions (i.e. width and height) as the input images. Specifically, we use a series of 3×3 convolutions, followed by a series of 1×1 convolutions that successively collapse the activation volume to a depth of 2, representing the predicted U and V chrominance channels of the image (see Figure 2). These 1×1 convolutions essentially constitute a series of fully-connected layers that map the activation depths of each pixel to a color prediction for that pixel. Our architecture employs batch normalization [8], which stabilizes the learning process by normalizing the input to each of the following activation units to have zero mean and unit variance. This has been shown to make training more robust against poor weight initialization as well as improve gradient flow when backpropping through deeper networks. We also use the ReLU activation function [12] to ensure proper gradient flow through our model.

3.1.2 Training

We train our baseline model on the dataset $X = \{(x_Y, x_{UV})\}$ using minibatch stochastic gradient descent with momentum, as described in Algorithm 1 with momentum μ and learning rate λ. We defined our minibatch loss as the mean of the individual example losses:

$$L_B = \frac{1}{|X_B|} \sum_{x \in X_B} L(x; \theta) \qquad (2)$$

where X_B is a minibatch sampled from X.

Since we are using ReLU activation units, we use the following weight initialization for all components $w_l \in \mathbb{R}^{d_l \times n_l}$ of the filter weights of every convolutional layer l as proposed by He et al. [6]:

$$w_l = \frac{z}{\sqrt{n_l / 2}} \qquad (3)$$

where d_l is the number of filters in l, n_l is the number of connections of a response to l, and $z \sim N(0, 1)$. This ensures proper gradient flow through the ReLU neurons.

3.2. Adversarial nets

We build on top of this baseline model by modifying it to take a sample of random noise z as a second additional input, effectively transforming it into a generative model $G(z|x_Y; \theta_g)$, which generates a colorization conditioned on the grayscale input image x_Y. We also construct a discriminator model $D(\hat{x}_{UV}|x_Y; \theta_d)$, which takes as input both the grayscale image x_Y and a colorization $\hat{x}_{UV}$, outputting a prediction of the probability that $\hat{x}_{UV}$ was the true colorization x_{UV} rather than the generated colorization $G(z|x_Y; \theta_g)$. As will be described in greater detail in the following sections, while D is trained to assign the correct labels to its input colorizations, G is trained to generate colorizations that "fool" D into assigning incorrect labels to them (see Figure 3).

3.2.1 Architecture

The architecture of the generator G is identical to that of the baseline model, with the sole addition of a fully-connected

Algorithm 1 Baseline Model Training

Initialize weights w_l in θ according to Equation 3.
Initialize update velocity $v := \vec{0}$.
repeat
  for each minibatch X_B sampled from X **do**
    Forward grayscale components x_Y of $x \in X_B$ through network to compute $F(x_Y; \theta)$.
    Compute minibatch loss L_B according to Equation 2.
    Compute gradients $\frac{\partial L_B}{\partial \theta}$ of L_B w.r.t. θ.
    $v \leftarrow \mu v - \lambda \frac{\partial L_B}{\partial \theta}$ (momentum update)
    $\theta \leftarrow \theta + v$ (momentum update)
  end for
until convergence

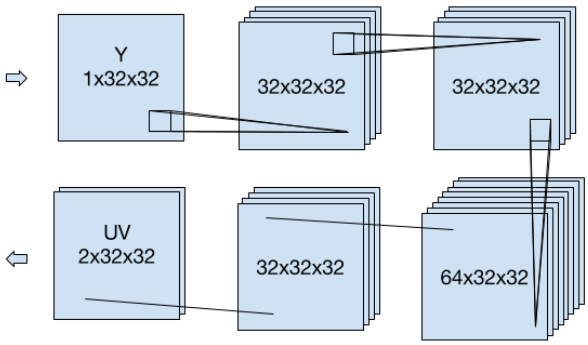

Figure 2: The convolutional architecture of the baseline model. The model is first composed of three convolutional layers with 3×3 filters, with 32, 32, and 64 filters respectively. These layers are then followed by two convolutional layers with 1×1 filters, with 32 and 2 filters respectively. Each convolutional layer except the last is followed by a spatial batch normalization layer and a ReLU activation layer.

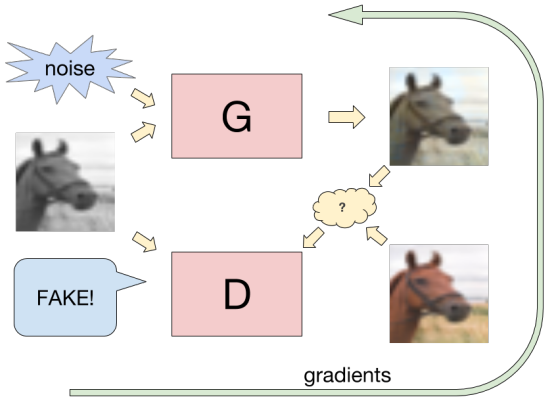

Figure 3: Diagram of the adversarial nets system. The generator G takes a grayscale image along with some random noise and proposes a colorization for the image. The discriminator D takes a colorization along with the original grayscale image and predicts whether or not the colorization came from the data distribution or the generator.

layer from the input noise z to a activation vector of size 1024, which is then reshaped into a single-channel 32×32 activation layer that is added to the grayscale input image before the first convolutional layer. This essentially adds random perturbations to the input, with the fully-connected layer theoretically allowing G to learn what this added noise should look like. We keep all other elements of the architecture the same; batch normalization and ReLU activation units have been shown to be critical to ensuring that the learned generative model is able to cover the colorspace of the data distribution it is trying to model [14].

We construct the discriminator D as a conventional convolutional neural network classifier: a series of three 3×3 convolutional layers with max-pooling followed by two fully-connected layers and a sigmoid activation to output a single probability prediction. We use standard ReLU activations and likewise initialize weights according to Equation 3. We use dropout [15] to regularize the network, but do not use batch normalization, to prevent the discriminator from learning too quickly and overwhelming the generator early in training [9].

3.2.2 Training

We adapt the training procedure proposed by Goodfellow et al. [5] (see Algorithm 2). In essence, we train D to minimize the cross-entropy loss of assigning the correct labels to colorizations given the original grayscale image, while we train G to maximize the loss, where we define the binary cross-entropy loss on a pair of examples $x, \tilde{x} \in X$ as:

$$-\log D(\tilde{x}_{UV}|\tilde{x}_Y) - \log(1 - D(G(z|x_Y)|x_Y)) \quad (4)$$

The most important consideration when training adversarial nets is carefully balancing this minimax game: if the discriminator performs too well and labels all of the images correctly with high confidence, the final sigmoid activation will become saturated and the gradient signal to the generator will vanish. On the other hand, training the generator too much without training the discriminator can allow the generator to exploit meaningless weaknesses in the discriminator (e.g. always outputting a blue pixels because the discriminator happens to correlate blue with a positive label in the current iteration). In our experiments, we only focus on addressing the former: we assume that because both the generator and discriminator take the original grayscale image as a conditional input, the discriminator has moreinformation to reject naive attempts to exploit its weaknesses. We also assume that our generator model is unlikely to be powerful enough to exploit such weaknesses due to its size and thus limited expressiveness.

We use heuristics adapted from Larsen and Sønderby [9]: for each iteration of training, if the discriminator's cross-entropy loss is less than a certain margin, then we skip the gradient update for D. This gives the generator a chance to catch up and harness the gradients from the discriminator before the discriminator starts to perform too well.

We also transfer weights from the pre-trained baseline model to initialize the generator model: from this point, the generator need only learn how to employ the stochasticity of the noise input and continue optimize its outputs to fool the discriminator. This should give the generator an additional handicap to stay at pace with the discriminator.

4. Dataset

We trained and evaluated our models on the CIFAR-10 dataset of 60,000 32×32 RGB color images in 10 classes. We chose the CIFAR-10 datasets because smaller images are much faster to train on, and thus more suitable for the first prototypes of our proposal. Limiting our dataset to 10 classes also constrains the space of potential objects that the model would need to learn how to color, giving our smaller models a chance to fit the data. Moreover, since the images in CIFAR-10 are taken "in the wild," there are often auxiliary unclassified objects in the images that can still test the generalization capabilities of the model. However, 1024 pixels is still very little information to go off of, there is less detail in the image to infer color from, so ultimately this is still just a exploration.

We withhold 10,000 images for the validation set, leaving 50,000 images for the training set.

4.1. Preprocessing

To each image in the dataset, we apply each of the following preprocessing steps:

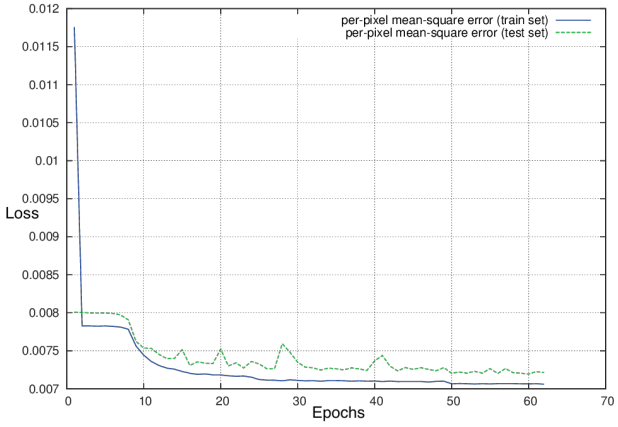

Figure 4: The loss history over 60 epochs of training the baseline model on the naive least-squares objective. The training loss of each epoch in this chart was computed as a running average of the minibatch losses, while the validation loss was computed at the end of each epoch.

1. Convert the image from the RGB colorspace to the YUV colorspace.

2. Apply spatial contrastive normalization to the Y component with a 7-pixel Gaussian kernel. This brings out the local details in the grayscale images, making it easier for the models to discern patterns and textures in the image.

3. Normalize the U and V components globally across the dataset to have zero mean and unit variance (e.g. normalize all U values in the dataset by a single global mean and variance). This smooths out variations in color profiles across the images in the dataset due to different illumination contexts.

We did not perform data augmentation in our experiments, but future work should incorporate data augmentation in the preprocessing pipeline.

5. Experiments

5.1. Baseline

We trained our baseline model with minibatch size of 128 images, an initial learning rate of $\lambda = 1$, momentum of $\mu = 0.9$, a learning rate decay of 1×10^{-7} for every iteration, and a step decay of $\frac{1}{2}$ for every 25 epochs. We define the loss on an entire dataset to be the average per-pixel loss averaged over the examples:

$$L(X;\theta) = \frac{1}{|X|} \sum_{x \in X} L(x;\theta) \tag{5}$$

We trained the baseline model for 60 epochs and plot the loss at each epoch for both the training and validation sets

Algorithm 2 Minibatch stochastic gradient descent of our adversarial colorization nets. We used a standard normal distribution for the noise prior $p(z)$. In our experiments, used RMSProp [16] for our gradient updates, and a margin of $\gamma = 0.5$ for the error heuristic.

for number of training iterations **do**
  Sample minibatch of m noise samples $\{z^{(1)}, \ldots, z^{(m)}\}$ from noise prior $p(z)$.
  Sample minibatch of m grayscale examples $\{x_Y^{(1)}, \ldots, x_Y^{(m)}\}$ from X.
  Sample minibatch of m full color examples $\{(\tilde{x}_Y^{(1)}, \tilde{x}_{UV}^{(1)}), \ldots, (\tilde{x}_{UV}^{(m)}, \tilde{x}_{UV}^{(m)})\}$.
  $\texttt{errorFake} := -\frac{1}{m}\sum_{i=1}^{m} \log\left(1 + D\left(G\left(z^{(i)}\middle|x_Y^{(i)}\right)\middle|x_Y^{(i)}\right)\right)$
  $\texttt{errorReal} := -\frac{1}{m}\sum_{i=1}^{m} \log D\left(\tilde{x}_{UV}^{(i)}\middle|\tilde{x}_Y^{(i)}\right)$
  if $\texttt{errorFake} > \gamma$ and $\texttt{errorReal} > \gamma$ **then**
    Update D by ascending its stochastic gradient:

$$\nabla_{\theta_d} \frac{1}{m}\sum_{i=1}^{m}\left[\log D\left(\tilde{x}_{UV}^{(i)}\middle|\tilde{x}_Y^{(i)}\right) + \log\left(1 + D\left(G\left(z^{(i)}\middle|x_Y^{(i)}\right)\middle|x_Y^{(i)}\right)\right)\right]$$

  end if
  Sample new minibatch of m noise samples $\{z^{(1)}, \ldots, z^{(m)}\}$ from noise prior $p(z)$.
  Sample new minibatch of m grayscale examples $\{x_Y^{(1)}, \ldots, x_Y^{(m)}\}$ from X.
  Update G by descending its stochastic gradient:

$$\nabla_{\theta_g} \frac{1}{m}\sum_{i=1}^{m}\log\left(1 + D\left(G\left(z^{(i)}\middle|x_Y^{(i)}\right)\middle|x_Y^{(i)}\right)\right)$$

end for

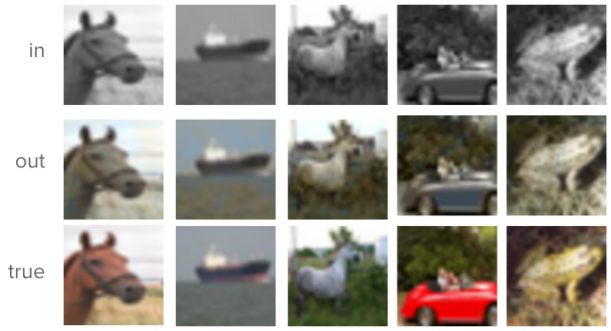

Figure 5: Example colorizations from the baseline model after training on 50,000 images for 60 epochs. (in) The non-normalized grayscale images. (out) The images colorized by the baseline model, converted back into the original data space. (true) The original "ground-truth" color images.

(Figure 4). Starting from a training loss of 0.0118, the system achieved a final training loss of 0.0071 and a validation loss of 0.0073. The loss history indicates that the system was able to learn a fit on the data, though further improvements in the baseline model could potentially bring the loss even lower.

To subjectively evaluate the results, we feed some images from the validation set through the trained model, and combine the predicted colorizations with the original unnormalized grayscale channels, and perform the inverse of the global normalization on the U and V channels to convert the prediction the original data space. The results are shown in Figure 5.

We can see that the baseline model is able to assign some sensible colors to the images. Patches of sky are colored with varying shades of blue—if a little patchy—while foliage and grass receive greenish or brownish tints. Other surfaces, such as the horse and the frog, are often given a brownish color. The car in the fourth column is even given a blue color, which may indicate the frequent occurrence of blue vehicles in the training data. In general, the model is able to learn a sensible if naive mapping from textures to colors.

However, the overall lack of deeper or brighter colors and the abundance of brownish hues could indicate that the least-squares loss function results in overly conservative predictions. For example, given many objects with similar textures but very different colors in the training set will result in an averaging of those colors in the model output, which in the YUV colorspace often leads to brownish colors. Doing so leads to a lower least-squares loss, but also creates less colorful and thus potentially less realistic im-

	Fake	Real
Fake	64	0
Real	14	50

Figure 6: An example confusion matrix for the predictions by model D on a single minibatch of 128 images from the first epoch of training. The rows represent the true labels, and the columns represent the predicted labels.

(a) (b)

Figure 7: Example colorization from the adversarial models. (a) input grayscale image. (b) output color image.

ages. This is exactly the problem we seek to address with the adversarial nets framework.

5.2. Adversarial Results

We trained our adversarial models using Algorithm 2 with minibatch size of 128 images, a learning rate decay of 1×10^{-7} for every iteration, and a step decay of $\frac{1}{2}$ for every 25 epochs. The discriminator is heavily regularized with a dropout probability of 0.8 in the convolutional layers and 0.5 in the penultimate fully-connected layer.

In our experiments, we were unable to effectively train the adversarial models, regardless of the learning rates we chose. Invariably, after a few minibatch updates, D had already become too good at classifying the colorizations, and even after disabling the training of D by the heuristic defined in Algorithm 2, G was never able to catch up. The confusion matrix for the predictions on a minibatch by D after 3-5 iterations can be typified by Figure 6.

Meanwhile, the gradient to G vanished to near zero, with the norm of the total gradient vector $\|\frac{\partial L}{\partial \theta_g}\|_2$ reduced to less than 1×10^{-4}. Without convenient ways to improve the current models, we chose the best possible learning rates (i.e. 1×10^{-3} for D and 1×10^{-2} for G) and trained the models overnight for 125 epochs, and tested the resulting models on the horse image from Figure 5, generating the color image in Figure 7. As is easily surmised, the result is a meaningless image with no semblance of the original object.

5.2.1 Error Analysis

The generator was not able to "catch up" and learn how to fool the discriminator, regardless of the number of iterations that the discriminator was deactivated. Once the discriminator's predictions become too accurate, the resulting saturation of the sigmoid activation leads the gradient to G to vanish. And even when the learning parameters are set just right to freeze D at an imperfect state with a more varied confusion matrix, G was not able to maximize the cross-entropy loss significantly in our experiments. This seems to indicate that our generator model is not expressive enough to fit the data distribution—indeed, the current model simply maps textures in a 9×9 patch around each pixel to its corresponding color prediction. With this architecture, there is no way for the model to learn or leverage higher-level (e.g. semantic) features. Employing a much deeper and sophisticated model, potentially pre-trained on large image dataset, may yield better performance from teh generator. Meanwhile, concurrently limiting the capacity of the discriminator model may also yield better training progression.

The fully blue-green image in Figure 7 was generated after overnight training, at which point the discriminator was still accurately labeling almost all the colorizations. This seems to corroborate potential problems with the generator model architecture, and possibly with how gradients are propagated from the discriminator to the generator.

Be as it may, we only tried one model architecture and one training procedure, and there is much to be improved in tuning the adversarial training.

6. Future Work

We were able to demonstrate reasonable results with the baseline model, however there is much more work to be done to get the adversarial models to work.

To improve the expressiveness of the generator, we could use the deep VGG-based colorization model presented by Ryan Dahl [4] as the baseline generator model. Better ways to incorporate the noise inputs should also be investigated: perturbing the input image may not be the most effective way of introducing stochasticity to the generator—instead, noise could be added or concatenated to a higher-level layer. Goodfellow et al. [5] also suggested maximizing $\log D(G(z|x_Y)|x_Y)$ instead of minimizing $\log(1 - D(G(z|x_Y)|x_Y))$ when training the generator to improve the gradient.

To improve the discriminator, Radford et al. [14] asserted that batch normalization in the discriminator is important to improving the gradient flow to the generator. Further suggestions also include using LeakyReLU activation in the discriminator, and replacing pooling layers in the discriminator with strided convolutions. Larsen and Sønderby

[9] also suggested increasing the regularization strength on D as its classification accuracy improves, which potentially improves upon the naive all-or-nothing heuristic in Algorithm 2.

Other things to try include employing proper data augmentation, such as rotating the images and programmatically perturbing their color profiles, as well as trying the L-a-b colorspace, which was used by Charpiat et al. [2] due to its basis on approximating human vision. Try L-a-b colorspace, designed to approximate human vision (used by Charpiat et al).

References

[1] Approximation capabilities of multilayer feedforward networks. *Neural Networks*, 4(2):251 – 257, 1991.

[2] G. Charpiat, M. Hofmann, and B. Schölkopf. Automatic image colorization via multimodal predictions. In *Computer Vision–ECCV 2008*, pages 126–139. Springer, 2008.

[3] Z. Cheng, Q. Yang, and B. Sheng. Deep colorization. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 415–423, 2015.

[4] R. Dahl. Automatic colorization, Jan 2016. `http://tinyclouds.org/colorize/`.

[5] I. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville, and Y. Bengio. Generative adversarial nets. In *Advances in Neural Information Processing Systems*, pages 2672–2680, 2014.

[6] K. He, X. Zhang, S. Ren, and J. Sun. Delving deep into rectifiers: Surpassing human-level performance on imagenet classification. *CoRR*, abs/1502.01852, 2015.

[7] Y.-C. Huang, Y.-S. Tung, J.-C. Chen, S.-W. Wang, and J.-L. Wu. An adaptive edge detection based colorization algorithm and its applications. In *Proceedings of the 13th Annual ACM International Conference on Multimedia*, MULTIMEDIA '05, pages 351–354, New York, NY, USA, 2005. ACM.

[8] S. Ioffe and C. Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. *CoRR*, abs/1502.03167, 2015.

[9] A. B. L. L. Larsen and S. K. Sønderby. Generating faces with Torch, Nov 2015. `http://torch.ch/blog/2015/11/13/gan.html`.

[10] A. Levin, D. Lischinski, and Y. Weiss. Colorization using optimization. In *ACM Transactions on Graphics (TOG)*, volume 23, pages 689–694. ACM, 2004.

[11] J. Long, E. Shelhamer, and T. Darrell. Fully convolutional networks for semantic segmentation. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 3431–3440, 2015.

[12] V. Nair and G. E. Hinton. Rectified linear units improve restricted boltzmann machines. In *Proceedings of the 27th International Conference on Machine Learning (ICML-10)*, pages 807–814, 2010.

[13] Y. Qu, T.-T. Wong, and P.-A. Heng. Manga colorization. In *ACM SIGGRAPH 2006 Papers*, SIGGRAPH '06, pages 1214–1220, New York, NY, USA, 2006. ACM.

[14] A. Radford, L. Metz, and S. Chintala. Unsupervised representation learning with deep convolutional generative adversarial networks. *CoRR*, abs/1511.06434, 2015.

[15] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov. Dropout: A simple way to prevent neural networks from overfitting. *The Journal of Machine Learning Research*, 15(1):1929–1958, 2014.

[16] T. Tieleman and G. Hinton. Lecture 6.5—RmsProp: Divide the gradient by a running average of its recent magnitude. COURSERA: Neural Networks for Machine Learning, 2012.

[17] E. Tola, V. Lepetit, and P. Fua. A fast local descriptor for dense matching. In *Computer Vision and Pattern Recognition, 2008. CVPR 2008. IEEE Conference on*, pages 1–8. IEEE, 2008.

[18] T. Welsh, M. Ashikhmin, and K. Mueller. Transferring color to greyscale images. *ACM Trans. Graph.*, 21(3):277–280, July 2002.

[19] L. Yatziv and G. Sapiro. Fast image and video colorization using chrominance blending. *Image Processing, IEEE Transactions on*, 15(5):1120–1129, 2006.