Image Colorization using Convolutional Neural Networks

Alex Aranburu[*] **Arghyadeep Giri**[†] **Rene Gutierrez**[‡] **Steven Reeves**[§]

Abstract

For the culmination of the course CMPS 242, Machine Learning, the authors present a method for image colorization using convolutional neural networks. Colorization, taking a black and white image and turning into a color (RGB) image, is inherently an underdetermined problem. Because of this we aim to generate plausible colorizations using the technology of convolutional neural networks.

1 Introduction

Image colorization is a difficult problem to solve, since one must predict a higher dimensional object from a lower dimensional one. Our approach to this underdetermined problem is to use a 7 layer convolutional neural network to generate the color channels.

1.1 Color spaces

The Red-Green-Blue(RGB) space is only one way to represent a color image. Going from a traditional RGB representation of an image to grayscale is a trivial matter, in which the grayscale image is just a linear combination of the red, green, and blue channels. However, going the opposite way has many challenges, that is we must predict a three dimensional object from a one dimensional input.

Before we construct our neural network, we change the problem. Since going from one channel to three is very difficult, we change our bases into something more managable - the LAB color space. Here L is the lightness, and is analogous to the grayscale image. A and B, is a two dimensional mapping of RGB. We can transition between RGB and LAB using the following method: First transition to the XYZ specific illuminants and observers and then construct the "Hunter-Lab" colorspace:

$$\begin{bmatrix} X \\ Y \\ Z \end{bmatrix} = [M] \begin{bmatrix} r \\ g \\ b \end{bmatrix}$$

where $[M]$ is calculated from the RGB reference white primaries (X_r, Y_r, Z_r) of the computer system. After obtaining the XYZ space from the original RGB image, the LAB space is calculated via the relationship

$$\begin{aligned} L &= 116 f_y - 16 \\ A &= 500(f_x - f_y) \\ B &= 200(f_y - f_z) \end{aligned}$$

[*] aaranbur@ucsc.edu
[†] argiri@ucsc.edu
[‡] rgutie17@ucsc.edu
[§] sireeves@ucsc.edu

Submitted to 31st Conference on Neural Information Processing Systems (NIPS 2017). Do not distribute.

where

$$f_d = \begin{cases} \sqrt[3]{d_r} & \text{if } d_r > \epsilon \\ \frac{\kappa d_r + 16}{116} & \text{else} \end{cases} \quad d_r = D/D_r$$
$$d = x, y, z; \; D = X, Y, Z$$
$$\epsilon = \frac{216}{24389}; \; \kappa = \frac{24389}{27}$$

as per the CIE standard [1].

We do this transformation to reduce the dimension of our problem. In the original situation, the grayscale image is used to predict the RGB channels. After the transormation, we use the L channel, analagous to grayscale, to predict only the A and B channels. This is a common strategy used in the literature[2,3,4]. The opencv library offers routines to translate RGB to LAB using the formalism above, we used this library in our preprocessing and post processing stages in our algorithm.

2 Neural network architecture

To create a model for color images given only a black and white one, we use a 7 layer convolutional neural network to extract the A and B channels in the LAB color space. Each layer contains a convolution, a rectified linear activation and a batch renormalization stage. We iclude the batch renormalization to reduce internal covariate shift while training, this helps accelerate the training stage of the model [5]. After each layer we downsample using a stride of two. At the output layer, we apply a deconvolution to upsample the data back to a useable dimension. Figure 1 is a schematic of our network architecture, adapted from Zhang et al's technique [2]. To implement this neural network, we use tensorflow and the opencv library.

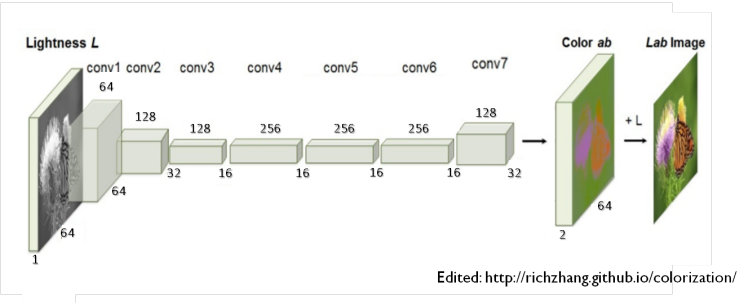

Figure 1: Schematic of the convolutional neural network.

2.1 Loss and tuning parameters

To train our network defined in figure 1, we use the classic L_2 loss:

$$Loss = ||X_{pred} - X_{labels}||^2$$

where X_{pred} are the predicted AB channels from the L input and X_{label} are the true AB channels.

2.2 Tune-in

There are several hyperparameters that are tunable in our tensorflow implementation. The batch size of training data, dropout rate of weights, kernel size, the dilation rate or stride and the final activation function are all things to tune within the model. In table 1, we evaluated different configurations of this hyperparameters individually, choosing the best performers. Clearly, this approach is very limited, as ideally each hyperparameter configuration should be considered by itself, and is very unlikely that the combination of the best performance on each parameter will result on the best performance overall. Furthermore, the number of configurations tried is very small, however we were very limited by time and resources, and the tune-in should be improved in the future. For the results presented we use the following values: a batch size of 10, dropout rate of 0.4, kernel are 3 by 3, and the dilation rate was 1. For the final activation we found that the hyperbolic tangent to be best suited for our loss function.

Table 1: Performance Statistics for Different Parameter Configurations.

Batch Size		*Dropout Rate*		*Kernel Size*		*Stride*	
10	*20*	*0.4*	*0.6*	*3x3*	*5x5*	*1*	*2*
195.55	217.08	215.69	221.48	200.09	209.24	206.2	214.87

3 Results

This section illustrates the results of our model using the above hyperparameters and loss function.

3.1 Data

We trained our model on 32 by 32 and 64 by 64 pixel images downloaded from Imagenet. The data set for each resolution contained 256232 images. We used 90% of the data for training, and 10% to test our model.

3.2 Results for the 32×32 pixel model

We find that the model performs well on certain images and not so well on others. Figure 2 contains images that were the model was more successful.

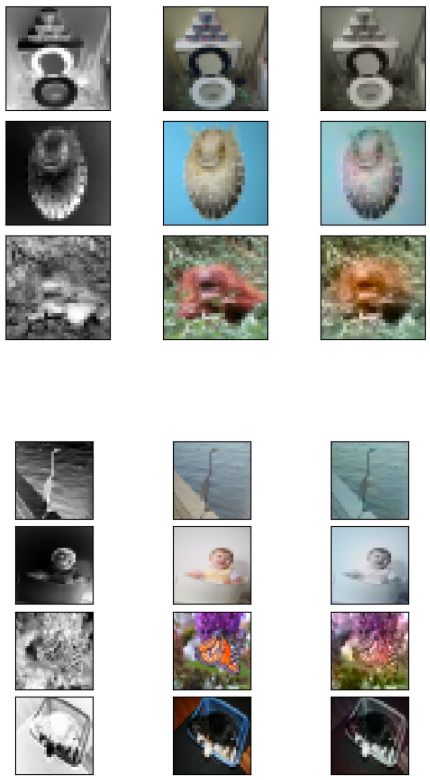

Figure 2: Relative successes on 32×32 pixel images. Left: Grayscale, middle: ground truth, left: model prediction.

In contrast, there were images where the model did not predict the color very well. Figure 3 shows two images where the model was not successful.

Figure 3: Failures of the model on 32×32 pixel images. Left: Grayscale, middle: ground truth, left: model prediction.

The first image in figure 3 of the African Grey Parrot was not well resolved. They grey in the prediction does not match the type of grey on the bird and the red hues of the branch and tail are not resolved. In the second image the model blurs all colors into different shades of green, losing the brown color of the turkey.

3.3 Test on 64×64 images

Perhaps some of the colors are not predicted because of the small resolution of each image in the previous section. The next figure illustrate the capabilities of the model trained on the 64 by 64 pixel images.

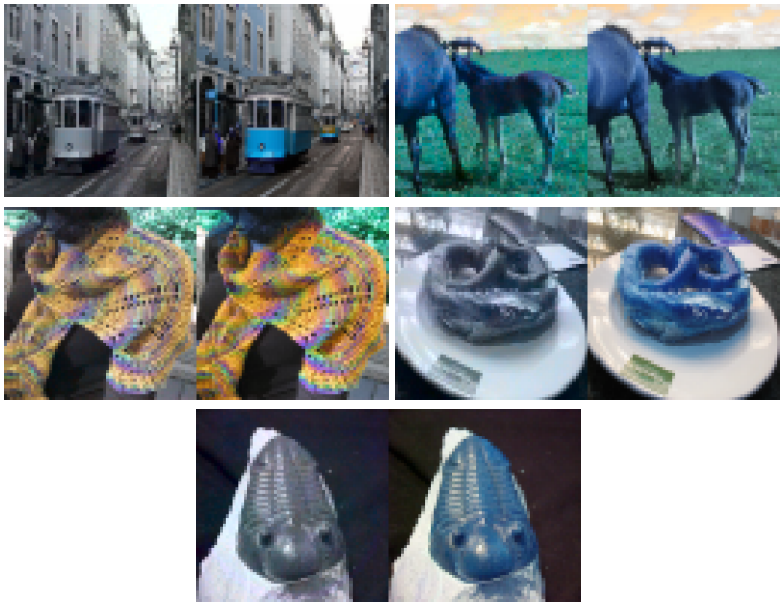

Figure 4: Accuracy of the model on 64×64 pixel images. Left: model prediction, right: ground truth.

The images in figure 4 gives more insight into the model than the lower resolution images. Here we see the effects of the Euclidean loss function. In images with sparse or very localized vibrancy, the model tends to blur the color, for example images 1,4 and 5 in figure 4. However, in images with

consistent color the model performs well, see images 2 and 3. Notice that in image 3, the vibrancy is still dimenished, giving the predicted photo a sepia like tone.

4 Discussion

The above results posit some insight into our methodology. The minimization of the Euclidean norm results in averaging out the vibrancy and color the images, resulting in a sepia-like tone. Furthermore, this type of model seems to preform better on larger resolution images.

To move forward, we plan to move to higher resolution images, 256×256, and change to a more exotic loss function. In [2], they created a loss function that allows for any "believable" configuration of colors in their colorization, essentially allowing for a multimodal loss. Another strategy would be to subtract out the mean luminosity from each training image use that to try and fix the vibrancy issue. As we increase the resolution, we aim to make the CNN into a deep neural network increasing the color feature extraction for better prediction.

Additionally, we investigated the use of the cross entropy loss. This would use the softmax function as the ending activation and could be weighted easily to favor vibrancy. However, we found it to be too computationally cost prohibitive for our study.

Other techniques could be used to solve this problem. Support vector machines would use 256 possible AB vectors using a K-means discretization, where K is chosen via validation. This would extract the SURF features from each pixel, along with a local min and variance. Another solution to remove the effects of a researcher chosen loss function would be to use an advisarial network.

References

[1] http://www.brucelindbloom.com/index.html?Eqn_RGB_XYZ_Matrix.html

[2] Richard Zhang, Phillip Isola, Alexei A. Efros. Colorful Image Colorization. University of California, Berkeley. https://arxiv.org/abs/1603.08511

[3] Gustav Larsson, Michael Maire, Gregory Shkhnarovich. Learning Representations for Automatic Colorization. University of Chicago, Toyota Technological Institute at Chicago. https://arxiv.org/abs/1603.06668

[4] Jeff Hwang, You Zhou. Image Colorization with Deep Convolutional Neural Networks. Stanford University. cs231n.stanford.edu/reports/2016/pdfs/219_Report.pdf

[5] Sergey Ioffe, Christian Szegedy. Batch Normalization: Accelerating Deep Network Training By Reducing Internal Covariate Shift. https://arxiv.org/abs/1502.03167