Image Colorization

Akshay Ravikumar
MIT
akshayr@mit.edu

Bishesh Khadka
MIT
bkhadka@mit.edu

Brian Xu
MIT
bwxu@mit.edu

Jacqueline Liu
MIT
jacqliu@mit.edu

Abstract

In this paper, we attempt an implementation of the image colorization problem, which, given the grayscaled version of an image, predicts its colored version. We propose an automatic system inspired from related works by Zhang et al [2]. Our system is composed of a feed forward pass through a convolutional neural network with a squared loss function. This network is trained on a couple hundred color images.

1. Introduction

The *image colorization* problem attempts to predict a plausible coloring of an image based on grayscale values. The rise of deep learning has enabled surprisingly accurate solutions to a seemingly intractable problem.

By training with a dataset that contains a wide variety of features, a neural network can generate estimated feature-to-color mappings that can be used to predict the colorization for unseen images. For example, if the neural network has observed several images of green grass, it learns that grass must be green: it can apply this mapping to color any unseen images that contain grass. With a broad-ranging dataset that provides sufficient information about these feature mappings, we can train a convolutional neural network to predict empirically reasonable colorizations.

For this project, we attempted image colorization through a data-driven method. We created a convolutional neural network and trained it on the Places2 small images data set [3]. Success was evaluated by comparison with the ground truth values, as well as observationally determining whether the generated colorizations are feasible.

2. Related Work

There exists two main previously explored approaches to solving this image colorization problem: data driven and user guided.

There have been several instances of image colorization using the data driven approach through a convolutional neural network. An example of a relatively successful CNN is the work done by Zhang et al. [2], which inspired our CNN and will be explained more in depth in the next section. There have also been other implementations with neural networks, including work done by Iizuka et al. [1] with joint learning of global and local features to colorize and classify images.

Previous work also explores a user guided method where users iteratively mark small portions of the image with colors. These markings are used to derive the colorings of the rest of the image. While this approach requires user input in order for the images to be colored, it also allows for more vibrant coloring of objects that don't have one specific color. For example, because cars can come in a large variety of colors, a data driven CNN coloring can lead to dulled tones because it would average over all of the possible car

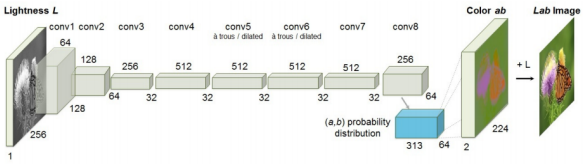

Figure 1: Zhang CNN Architecture [2].

colors. The user-driven coloring avoids this problem by basing ambiguous colors off user markings.

3. Approach

In this paper, we implement the approach taken in "Colorful Image Colorization" by Richard Zhang et al [2].

In particular, we consider images in the Lab color space. We train a CNN that, given the L values of an image (which correspond with intensity or grayscale values), and predict the corresponding ab values for that image (which correspond with color).

The architecture presented by Zhang consists of a succession of 8 layers, each consisting of a few convolution and ReLU layers followed by a BatchNorm layer. See Figure 1 for a detailed description of the architecture. The architectures we created are inspired from Zhang's.

Specifically, we work with $K \times K$ images (in this paper we use $K = 256$). Given the L values $\mathbf{X} \in \mathbb{R}^{K \times K \times 1}$, we attempt to find the prediction function $\mathcal{F} : \mathbb{R}^{K \times K \times 1} \rightarrow \mathbb{R}^{K \times K \times 2}$, which predicts ab values given L values. Defining $\hat{\mathbf{Y}} = \mathcal{F}(\mathbf{X})$ to be the predicted color values, and letting $\mathbf{Y}$ be the ground truth values, we attempt to minimize the following objective function:

$$J(\hat{\mathbf{Y}}, \mathbf{Y}) = \sum_{i,j} ||\hat{\mathbf{Y}}_{ij} - \mathbf{Y}_{ij}||^2$$

In other words, we consider the squared loss between the prediction and ground truth values. While Zhang et al. uses a multinomial cross-entropy loss function to increase robustness when compared to squared loss, we implement squared-loss for the sake of simplicity.

In implementing this model, we generated a training set of various images, downsized to 256×256 and transformed to the Lab color space. We then fed the input and ground truth values into the CNN model, which was implemented in TensorFlow. We used an AdamOptimizer to minimize the least-squared loss with respect to the input images. We then run the generated model on a validation set of 10 images and record validation accuracy.

4. Experimental Results

4.1. CNN Results

We implemented two CNNs, a small CNN in `cnn.py` and a large CNN in `big_cnn.py`. The code for our CNNs can be found at `https://github.mit.edu/bwxu/6.819`. The small CNN was a simple architecture of 4 convolutional layers, while the large CNN attempted to implement the architecture described in Zhang et al. (Figure 1).

To train and test our CNNs, we converted images from the Places2 data set to the Lab color space. After training the CNN, we ran test images through the CNN, converted the predicted Lab values to RGB, and saved the prediction image.

We produced the image in Figure 6 using our small CNN architecture by using 300 training samples and 100 epochs. The predicted image seems to convert the image into a shade of beige. We expected something similar to this output because of the relatively ambiguous nature of the black and white image along with the fact that our training data set was very small. Further analysis of the CNN output is explored in the Desired Results section.

We produced the image in Figure 7 using our large CNN architecture by using 10 training samples and 20 epochs. The predicted image ends up having better per pixel square loss than the small CNN (see Figure 2 and Figure 3). Again, because of the very small amount of training data

Epoch	Avg. per pixel squared loss
1	369.87527712
2	368.92566705
3	367.98631603
4	367.06158086
5	366.15178964
6	365.25707438
7	364.37742311
8	363.5124665
9	362.6617131
10	361.82546603

Figure 2: Per pixel square loss for 100 images after the corresponding epoch of training for Small CNN

Epoch	Avg. per pixel squared loss
1	15.3476163388
2	15.3476047928
3	15.3475989311
4	15.3475924162
5	15.3475843999
6	15.3475801709
7	15.3475730507
8	15.3475664638
9	15.3475602968
10	15.3475532768

Figure 3: Per pixel square loss for 100 images after the corresponding epoch of training for Large CNN

we used, the image produced was not very close to the actual image. However, with significantly more training data and more epochs, we can expect that the large CNN would perform much better.

Figure 4: Original Color Input Image

4.2. Implementation Challenges

Unfortunately, we weren't able to get the exact Zhang et al. architecture working properly in our big CNN. In implementation, we encountered several challenges; following is a description of these challenges, along with our efforts to overcome them.

Figure 5: Original Black and White Input Image

Figure 6: Output Image from Our Smaller CNN

Figure 7: Output Image form Our Large CNN

1. **Lack of resources:** A challenge we faced was the lack of resources we had at our disposal. We relied primarily on Athena cluster machines and so training a large data set as Places2 [3] was not feasible. Thus we trained on a much smaller subset of the data, using a maximum 300 images, for 100 epochs. This underfitting contributed to our model's less

than realistic output.

2. **Discrepancies between Caffe and TensorFlow**: Zhang et al. have hosted a repository on Github outlining their architecture as a Caffe model. As the Caffe model protocol contains more detail than present in the paper, we decided to focus our efforts on converting this Caffe model to TensorFlow. In doing so, we encountered several difficulties, some of which are outlined below:

   - **Lack of Dilation:** Adding dilation to convolutional neural networks is a relatively new approach, which increases the efficiency of a CNN. While Caffe supports layers with dilation, we had some problems converting this to TensorFlow. For example the `atrous_conv2d` and `conv2d` functions don't support dilation. For this reason, we weren't sure how to best convert this model into TensorFlow while maintaining the integrity of the existing architecture.
   - **Upsampling:** In addition, the Caffe architecture contains an upsampling layer, i.e. a deconvolutional layer with a fractional stride of 0.5. While we were able to implement a deconvolutional layer in TensorFlow (via `conv2d_transpose`) we're not confident this was implemented correctly.
   - **Scale Layers:** During the final layer of the Zhang et al. architecture is a "Scale" layer, which, in short, performs pointwise multiplication between a layer and a tiled transformation of that layer. While we looked into how to convert this into TensorFlow, our research didn't return conclusive results. Instead of applying the Scale layer, we attempted to compromise for this by adding a deconvolutional layer that reshaped the output to the desired shape. This might have contributed to the high loss values in our attempted reproduction.

   In total, due to elements in the Caffe architecture that are difficult to reproduce in TensorFlow, we weren't able to construct a model that converged to desirable loss values. Given more time, we might have decided to use a different deep learning framework, or further investigated how to implement these functions. As an aside, we investigated a module that converts between Caffe and TensorFlow models, but this module didn't support features such as Scale layers and batch normalization. As this module is rather popular (700 Github stars), this corroborates our discovery that converting between the frameworks was disappointingly nontrivial.

5. Conclusion

Taking a deep-learning approach to the image colorization problem allows a feature-based model, which can efficiently predict the colors in an unseen image based on matching features in previously seen images.

We attempted to implement the Zhang et al. CNN architecture, and accompanied this with a smaller, simpler architecture. Because of some difficulties and reproducing elements of the Caffe model in TensorFlow, unfortunately we were unable to reproduce the accuracy values generated in Zhang et al. Despite our constraints and relatively simple implementation, our CNN managed to output results which constitute real progress towards colorizing the image.

References

[1] H. I. Satoshi Iizuka, Edgar Simo-Serra. Let there be color!: Joint end-to-end learning of global and local image priors for automatic image colorization with simultaneous classification. *ACM Trans-*

actions on Graphics (Proc. of SIGGRAPH 2016), 35(4).

[2] R. Zhang, P. Isola, and A. A. Efros. Colorful image colorization. *European Conference on Computer Vision*, 2016.

[3] B. Zhou, A. Khosla, A. Lapedriza, A. Torralba, and A. Oliva. Places2 dataset, `http://places2.csail.mit.edu/download.html`.