Image Colorization with Convolutional Neural Networks

Hanna Lee, Jason Liang, Narindra Peaks

Abstract

We attempt to replicate the results of the 2016 paper "Let there be Color!: Joint End-to-end Learning of Global and Local Image Priors for Automatic Image Colorization with Simultaneous Classifcation" by Iizuka, Simo-Serra, and Ishikaw. Their work uses a novel convolutional neural network, which extracts low-level, mid-level, and global features in order to perform colorization. Our work differs in that we train on 128x128 pixel images (as opposed to 224x224 in the original paper) and we use a smaller dataset in order to reduce training time. As a result our neural network architecture is slightly different, as described in the Approach section. Even so, we find that our neural net is capable of generating reasonable colorizations.

I. Introduction

Image colorization has a long history as a computer vision problem. We attempt to replicate the results of the 2016 paper "Let there be Color!: Joint End-to-end Learning of Global and Local Image Priors for Automatic Image Colorization with Simultaneous Classification" by Iizuka, Simo-Serra, and Ishikaw.

Image colorization is the problem of coloring the input pixels of a grayscale image in some RBG color space. Standard techniques in this domain include segmenting the image and coloring each segment with a different color, and annotating an image with colors and propagating each color to similar pixels. This paper introduces a novel deep neural net architecture which needs no human intervention to color an image (Figure 1).

This neural net extracts low-level features using a few convolutional layers. These low-level features are then fed as inputs to a series of layers which extract mid-level features and a series of layers which extract global features. The mid-level and global features are then combined and fed through a colorization network, which outputs final predicted chroma values for each pixel.

II. Related Work

As mentioned previously, standard techniques in this domain include segmenting the image and coloring each segment with a different color and annotating an image with colors and propagating each color to similar pixels. Image colorization using segmentation is several decades old, and uses thresholding and clustering techniques to partition images into related components. For each component, all the pixels in that component are assigned the same color.

An extension of this approach was pioneered by Levin, Lischinski, and Weiss in their 2004 paper "Colorization Using Optimization". They used manual annotation to color subsets of pixels in an image, then used affinity functions to propagate those colors to neighboring pixels. These affinity functions imposed the constraints that neighboring pixels should have similar colors and that pixels with similar grayscale intensity values should have the same colors.

With the widespread use of deep learning in the past few years, approaches using convolutional neural networks have gained traction. A recent example of such a technique can be found in the paper "Colorful Image Colorization" by Zhang, Isola, and Efros. In addition to using convolutional networks, they structure the colorization problem as a classification problem in order to produce pixel colors which are not dull.

"Let there be Color!" differs from previous approaches in that the network explicitly attempts to learn both mid-level and global features, then combines them in order to produce a colorized image which is closer to the ground truth.

III. Approach

We use the TensorFlow library, which is a popular machine learning framework for deep learning written in Python. TensorFlow provides abstractions for dealing with the architecture of convolutional neural networks. Beyond that, we did not use any preexisting code. Our code can be found at https://github.com/hlee95/image-colorization.

In order to leverage the power of GPUs for deep learning, we use AWS instances to run our code. This decreased the amount of real-world training time for our network by several orders of magnitude.

The dataset that we used was a subset of the Places 2 dataset from MIT downloaded from the 6.819 final project website, which consists of 100,000 grayscale and color images. These images have dimensions of 128x128 pixels. We read the images in Lab format, where the L channel encodes luminosity and the a and b channels encode color coordinates. The L channel is used as the input grayscale image, and the a and b channels are used as the target values for the output color image (with the values normalized to be between 0 and 1).

The downloaded dataset contains a number of directories, where each directory contains images of the same object or concept. In order to train the classification network of our neural net, we use one-hot encoding to encode the object/concept class that each picture belongs to.

We now discuss some of the properties of our network architecture. As the activation function for each layer besides

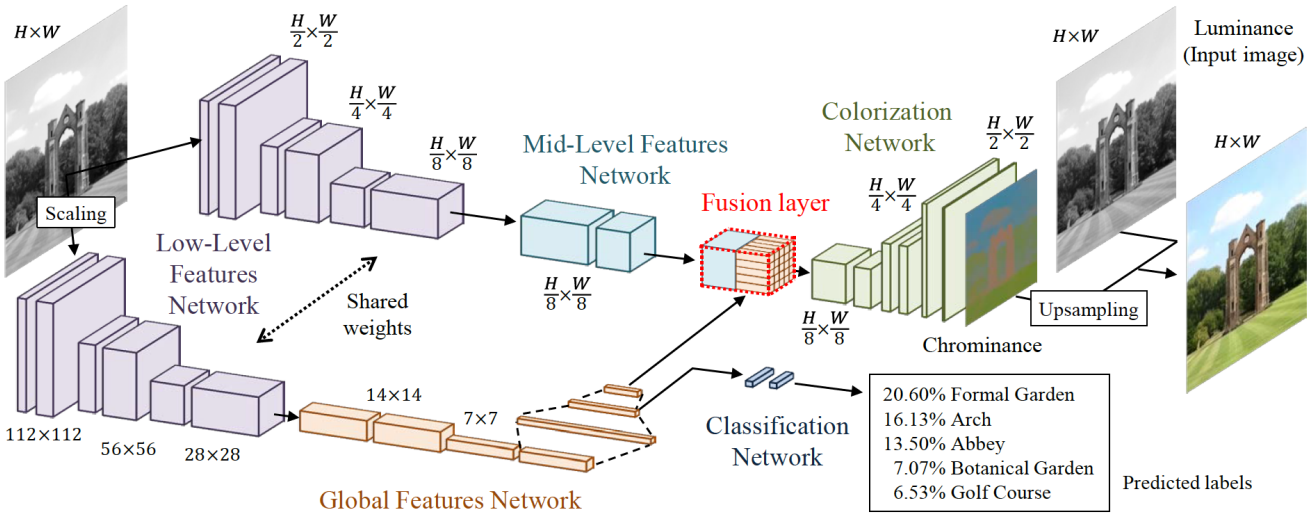

Fig. 1. Architecture of the network of the original paper. Taken from "Let there be Color!"

the color output layer, we use rectified linear units. For the color output layer, we use the sigmoid transfer function. In addition, similarly to the original paper, we use the Adadelta optimizer as opposed to gradient descent so that we do not need to optimize a global learning rate.

As is the standard for neural net architectures, we initialize the bias terms to be identically zero. In addition, for each node, we initialize the weights so that they have a standard deviation which is proportional to the square root of the number of input nodes to that node, which is also standard. When initializing the weights to have constant standard deviations, the resulting a and b values were either 0 or 1, since the inputs to the sigmoid activation function were strongly positive/negative.

We also apply batch normalization, which normalizes the mean and standard deviation of the input of each layer. The motivation behind batch normalization, which was introduced in the 2015 paper "Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift" by Ioffe and Szegedy, is that noisy training data may perturbe the weights of hidden layers, leading to long convergence times. Batch normalization normalizes the input of each layer to have a mean of zero and a standard deviation of one in order to speed up training of the network. We implement batch normalization using the same strategy which is presented in the demonstration on this website: http://r2rt.com/implementing-batch-normalization-in-tensorflow.html.

Our low-level network consists of four convolutional layers (Table I), which produces 256 filters which are 32x32 in size.

Type	Input dimension	Kernel	Stride	Outputs
conv	128x128x1	3x3	2x2	64x64x64
conv	64x64x128	3x3	1x1	64x64x128
conv	64x64x128	3x3	2x2	32x32x128
conv	32x32x128	3x3	1x1	32x32x256

TABLE I
HYPERPARAMETERS OF LOW-LEVEL FEATURES NETWORK

The outputs of the low-level network are then fed into a mid-level network which consists of two convolutional layers (Table II), which produces 128 filters which are 32x32 in size.

Type	Input dimension	Kernel	Stride	Outputs
conv	32x32x256	3x3	1x1	32x32x256
conv	32x32x256	3x3	1x1	32x32x128

TABLE II
HYPERPARAMETERS OF MID-LEVEL FEATURES NETWORK

The outputs of the low-level network are also input into a global feature network which consists of four convolutional layers and three fully-connected layers (Table III), which produces an output layer with 128 neurons.

Type	Input dimension	Kernel	Stride	Outputs
conv	32x32x256	3x3	2x2	16x16x256
conv	16x16x256	3x3	1x1	16x16x256
conv	16x16x256	3x3	2x2	8x8x256
conv	8x8x256	3x3	1x1	8x8x256
FC	16384			512
FC	512			256
FC	128			128

TABLE III
HYPERPARAMETERS OF GLOBAL FEATURES NETWORK

The mid-level and global features are then combined into a fusion layer with dimensions of 32x32x256 (Table IV). The global feature vector of size 128 is concatenated onto each 1x1x128 volume of the mid-level network output in order to form the fusion layer. In other words, for each mid-level coordinate (u, v), the corresponding coordinate of the fusion layer is

$$y^f_{u,v} = \sigma\left(W \begin{bmatrix} y^g \\ y^{mid}_{u,v} \end{bmatrix} + b\right)$$

where y^g and $y^{mid}_{u,v}$ are 1-dimensional vectors of length 128.

The final colorization network is formed from alternating layers of convolutional layers and upsampling layers, where each upsampling layer is generated using the nearest neighbor technique.

The last network that we use is a classification network, which consists of two fully connected layers. The input to

Type	Input dimension	Kernel	Stride	Outputs
fusion				32x32x256
conv	32x32x256	3x3	1x1	32x32x128
upsample	32x32x64			64x64x64
conv	64x64x64	3x3	1x1	64x64x32
output	64x64x32	3x3	1x1	64x64x2

TABLE IV
HYPERPARAMETERS OF COLORIZATION NETWORK

this network is the second-to-last global features layer, and the output is a 1D vector with c elements, where element i represents the probability that an element is in class i (the softmax function is used in order to acheive normalization). The classification layers are used to augment the network with additional global feature information. For instance, the network can be trained to not color indoor images with colors which are associated with grass or the sky.

Type	Input dimension	Kernel	Stride	Outputs
FC	256			256
FC	256			205

TABLE V
HYPERPARAMETERS OF CLASSIFICATION NETWORK

The dimensions of each layer in our network attempt to mimic the dimensions of each layer in the network of the original paper as much as possible. There are deviations due to the size of our input images being 128x128, while the sizes of the input images in the original paper are 224x224.

Our final loss function, which is optimized using Adadelta, is a combination of the colorization loss and the classification loss. Colorization loss is measured using mean squared error, while classification loss is measured using cross-entropy loss. Specifically, our loss function is

$$L(y^{\text{color}}, y^{\text{class}}) = ||y^{\text{color}} - y^{\text{colora}}||^2_{\text{FRO}} + \alpha L_{\text{cross-entropy}}(y^{\text{class}}, y^{\text{classa}})$$

where α is a parameter which weights the importance of the cross-entropy loss, y^{color} are the predicted colors, y^{colora} are the actual colors, y^{class} is the predicted class, y^{classa} is the actual class, and $||\cdot||_{\text{FRO}}$ denotes the Frobenius norm.

The α parameter is used to keep the magnitude of the two losses equivalent. This is to ensure that the classification network has an equal amount of influence over the training of the network. We originally used an α value of $\frac{1}{300}$ as suggested by Iizuka et al., but after further inspection of the magnitude of the colorization and classification losses, we discovered that $\alpha = 5$ is needed to ensure magnitudes of the same scale.

IV. Results and Discussion

We trained our network for 25 epochs on the training set. We used a learning rate of .00001 and a decay of 0.95. We then produced color predictions for 1,000 test images in the dataset. Some results are shown in figures 2 and 3 on the next page. The classification loss after scaling by α was between 1700-2100, and the colorization loss was in a similar range. In the first few training iterations, the colorization loss began at approximately 34,000, so a drop to 2000 is improvement. It seems that the loss converged very quickly to 2000, but then didn't reduce any further. One possible issue could be that the network is converging to a local instead of global minimum.

While our network mostly succeeds in detecting boundaries and coloring objects all one color, all of our results appear to contain only green, red, or blue, with no intermediate colors. We have a couple hypotheses for why this might be the case. The first is that the loss is converging to a local minimum instead of a global minimum, and that somehow these colors are safe choices that make the loss relatively small regardless of what the actual colors are. Another hypothesis is that our data representation or conversion between color spaces is wrong.

Given more time, our next steps would be to isolate the problem. One possible approach is to design the network to train in the RGB instead of LAB color space. This would mean using the luminance as input, and using all three RGB color channels as the desired output. This eliminates any potential color space conversion error, so if we implement this and get similar results to what we currently have, then we know that the problem is likely independent of the color space. Another possibility is that we simply need to further fine tune the parameters. We could search through the space of parameters to find ones that minimize loss.

V. Contributions

I designed and programmed the first iteration of the neural network architecture, implemented batch normalization, determined the correct standard deviations for the weights in every layer of the network, and debugged scaling errors. In addition, I wrote the majority of parts I, II, and III of the paper.

VI. References

S. Iizuka, E. Simo-Serra, and H. 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), vol. 35, no. 4, 2016.

A. Levin, D. Lischinski, and Y. Weiss, "Colorization using optimization," in ACM Transactions on Graphics (TOG), ACM, vol. 23, 2004, pp. 689-694.

R. Zhang, P. Isola, and A. A. Efros, "Colorful image colorization," European Conference on Computer Vision, 2016.

S. Ioffe and C. Szegedy, "Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift".

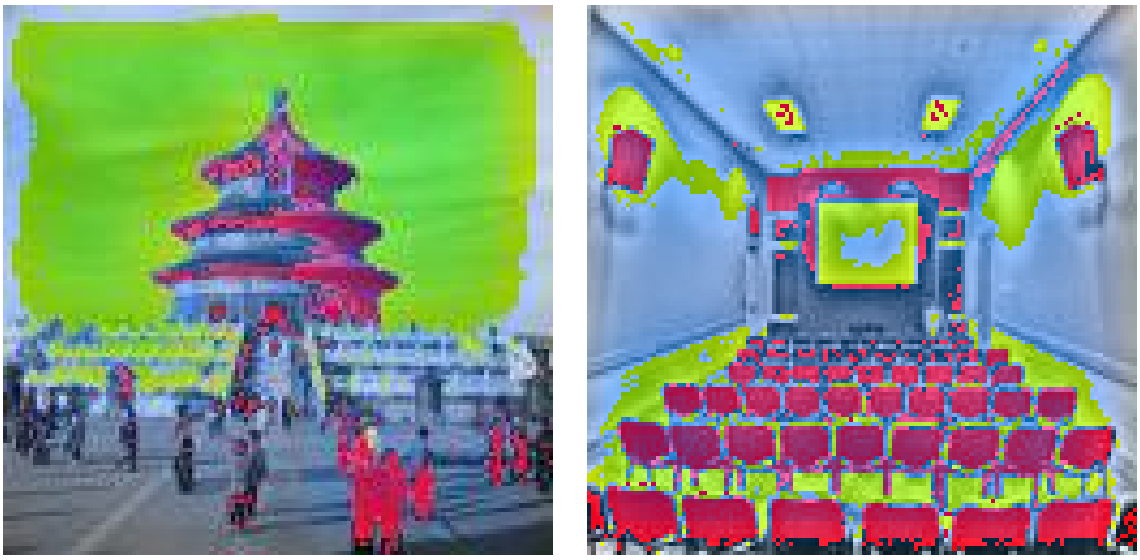

Fig. 2. Forbidden City; Lecture Hall

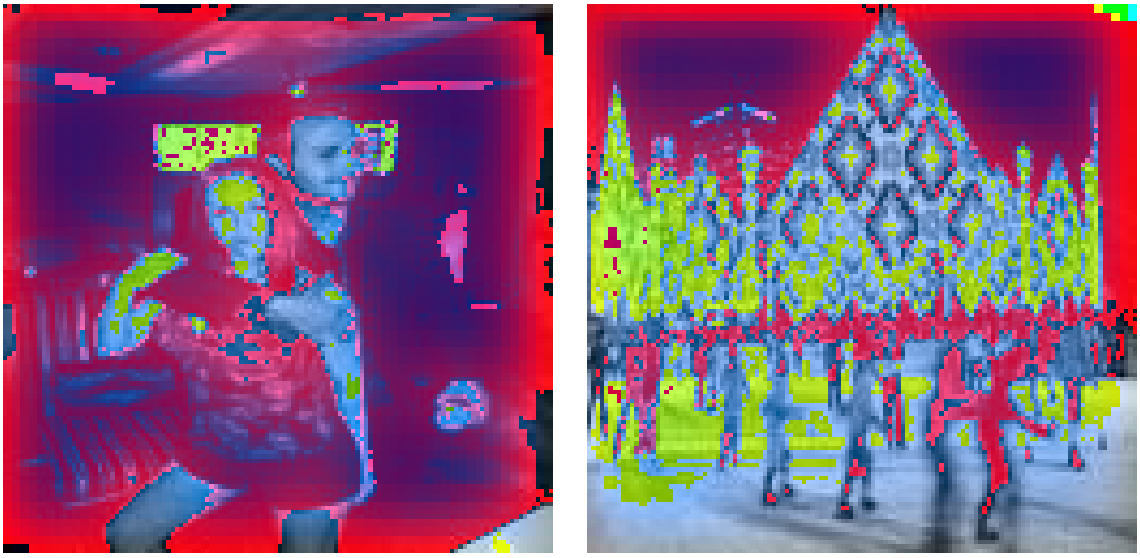

Fig. 3. Dancing Couple; Ice Skating