Colorization Model Replication with Loss Function Analysis

Emilia Psacharopoulos
University of Michigan Computer Science and Engineering
1301 Beal Ave Ann Arbor, MI 48109
emiliap@umich.edu

Abstract

In this paper, I replicate a novel Colorization Model that learns to convert gray scaled images into colored images. Furthermore, I investigate my results using Mean Squared Error against known results using Cross Entropy. This Colorization Model uses a network of up sampling, down sampling, convolutional, batch normalizations, Leaky ReLU, and classification layers before feeding into a loss function to classify gray pixels into colored pixels. Its learning is accomplished through iterative gradient computations and subsequent parameter tuning. My results prove that a mean squared error loss function is a poor decision for this Colorization Model.

1. Introduction

The primary challenge associated with Colorization research initiatives is that the majority of objects do not have an associated color. Even if there are certain items that tend towards a certain color, such as green grass, think about the effects of painting a surface, natural changes in the environment, or passing an image through a tinted filter beforehand. Colorization techniques that rely on simply classifying image regions as known objects to then assign the most probable color would fail dramatically. The problem of colorization is therefore undefined, as there is no perfect a priori color assignment. Modern colorization techniques are divided into two main categories: User-Guided Methods and Deep Learning Methods.

User-Guided Methods are simple and quick algorithms that depend on human input to manually adjust model parameters to better represent ground truth images. The models themselves use K-Nearest-Neighbors with advanced machine learning methods in conjunction with these tunable parameters. This technique has major limitations including the quality of ground truth images chosen and the slow training times due to manual input [4].

Deep Learning Methods automatically update its parameters by continuously classifying images over training periods spanning several days to weeks. These architectures are typically single path or multi-path Generative Adversarial Networks (GANs) or Convolutional Neural Networks (CNNs) with probabilistic distributions to map gray scaled pixels to colored pixels. Typically, these larger networks with small kernels tend to outperform smaller networks with large kernels. The best advantage of the Deep Learning Method is it can train on a practically unlimited number of images, where the researcher is only limited by their available processing resources. To the point of practically unlimited training time, researchers can optimize for any given criteria they choose such as image context (e.g. sketches or film) or a circumstantial color re-balance. However, the Deep Learning Methods are computationally expensive, and prototyping takes a long time as any architectural update to the model would take several days to be realized.

My chosen Colorization Model is a fully automatic deep learning model that uses a network of up sampling, down sampling, convolutional, batch normalization, Leaky ReLU, and classification layers before feeding into a loss function to classify gray pixels into colored pixels layers. After feeding each image through its network, this model also re-balances the classes of colors which effectively produces more vibrant and accurate colored images. The researches trained this model on over 1,000,000 images, and it proved to be highly accurate when testing for raw accuracy (AuC) and against human judgements. The model's strengths definitely include this high level of accuracy and fully-autonomous design (without the slow training times associated with user-guided methods). Its main shortcomings include computational expenses, and its dependency on class-re-balancing image processing before feeding the gray scaled image into the network (leading to even more expensive computations) [5].

My hypothesis tests if a Mean Squared Error loss function would perform as well as the researcher's choice of a Cross Entropy Loss function. One of my major design choices in this experiment is neglecting to include the proposed class re-balancing method. Please see Section 3 Results for a thorough analysis of this decision.

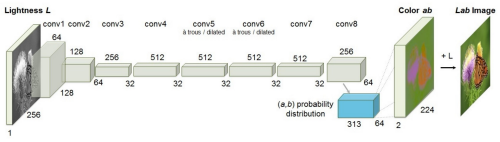

Figure 1. The CNN Architecture from the Colorization Model.

The remainder of this paper discusses the approach I took to replicate this model, and the results I achieved in doing so.

2. Approach

This Section discusses my approach to implement the Colorization Model. The main features discussed include the Data Set, CNN Architecture, Optimizer, Classifier, and Loss Function

2.1. Data Set

The researchers of this Colorization Model trained their model on over 1,000,000 images. I do not have the computational resources to achieve even a fraction of the processing power nor memory required to train on 1,000,000 images. Therefore, I opted to use a subset of the MIT Mini Places data set that contains 90,000 unique training samples. This data set has more than enough images for my given computational capabilities. I used the raw colored images from the data loader output as ground truths, and passed each image through Torch vision's Gray scale transform function as inputs for my replicated model. As discussed in Section 3.5 Limitations, I only trained my version on 30 Epochs and a batch size of 256 on 64x64 pixel images.

2.2. CNN Architecture

I implemented the network architecture from this Colorization Model almost exactly by following their explicit layer-by-layer steps on GitHub [6, 7]

The data set in my implementation has a slight difference to their implementation. The MIT Mini Places data set has 3 channels (R,G,B channels) while theirs have 4 channels (R,G,B,Y). As a result, I modified a few layers to change the stride or kernel size by +/- 1 such that each outputted layer would have the same effective shape as the original model.

The other difference in my implementation was neglecting to implement their class-re-balancing method, as discussed in Section 3.5 Limitations.

2.3. Optimizer

The researchers opted for the ADAM optimizer with parameters B1 = .9, B2 = .99, weight decay = 10^-3, and learning rate = 3 * 10^-5. The ADAM optimizer iteratively updates network weights to efficiently anneal each step. Their learning rate is incredibly small because they had the time and resources to train for an extended duration [2].

I had to adjust the ADAM parameters in my implementation because their parameters assume the intense computational power and enormous number of inputs that I cannot supply. I made multiple attempts to fine-tune the learning rate, betas, and epsilon parameters until I achieved the best results after 30 Epochs and a batch size of 256 on 64x64 pixel images. I ultimately decided that learning rate=0.001, betas=(0.9, 0.999), and epsilon=1e-08 visually produced the best results.

2.4. Classifier

The researchers passed the model output through another convolutional layer before implementing their class-re-balancing algorithm. That algorithm consists of yet another convolutional layer, a hyperbolic tangent layer, up scaling, a Softmax layer, and lastly an annealed-mean layer. The class-re-balancing algorithm effectively maps the predicted color outputs to actual colors that look good to the human observer.

I did not implement their class-re-balancing algorithm, as discussed in Section 3.5 Limitations.

2.5. Loss Function

The researchers used the multinomial Cross Entropy Loss function. The function maps predicted probabilities per class label (M total), where the probabilities (p) range from [0,1], to a sum of the total log losses for that class label (y). Stereo-typically, Cross Entropy Loss is the best option in machine learning classification problems such as Colorization because it tends to lessen the impacts caused by small gradients and weight decay which therefore allows more modifications to the model as the testing samples progress [1]. The Cross Entropy Loss equations are defined below:

$$-(y \log(p) + (1-y) \log(1-p)) \tag{1}$$

$$-\sum_{c=1}^{M} y_{o,c} \log(p_{o,c}) \tag{2}$$

My chosen area of investigation for this project is how the choice of Loss function qualitatively impacts the system results. I decided to pick the Mean Squared Error Loss function to compare against Cross Entropy. Mean Squared Loss is not stereo-typically used in classification models because it assumes the underlying data follows a normal distribution, and that its inputs (x) can range from +/-∞ instead of [0,1] when mapping to an output (y) [3]. The MSE equation is defined below:

$$\sum_{i=1}^{D}(x_i - y_i)^2 \tag{3}$$

Despite these known shortcomings, I wanted to investigate exactly what differences appear during testing in the specific use case of image colorization.

3. Results

This section compares results between my model version using a Mean Squared Error (MSE) Loss function and the researcher's version using a Cross Entropy Loss function. Ultimately, I could not outperform or match the quality of results produced by the original Colorization Model. The reasons could either be entirely dependent on my choice of Loss function, or somewhat dependent on the Loss function and also the subtle other differences listed in Section 3.5 Limitations. For the sake of this paper, I will make the assumption that my implementation of their model was correct, which is not a drastic assumption to make considering it performed well in a series of very specific pictures. Instead, I will assume and attempt to reason that every difference to the Colorization Model's output trends without class-re-balancing is caused by my choice of Loss function.

In the subsections that follow, I explore results relating to Classifying Reds and Yellows, Classifying Neutral Colors, General Loss of Vibrancy, and Edge Cases that Excel. As noted below, the most glaring difference between the two versions of the model is the general loss of vibrancy associated with the MSE loss function.

3.1. Classifying Reds and Yellows

The Colorization Model and my version of this model both fail at distinguishing the majority of reds and yellows from grays or blues. Clearly, this is a problem caused by a common misclassification across both models, and therefore is solved directly by implementing their class-re-balancing algorithm. Since I did not implement that algorithm, I will make the assumption that this error is not caused by my choice of Loss function.

3.2. Classifying Neutral Colors

Both versions of the model performed well at classifying images with predominant neutral toned regions. There is some noticeable loss of the tan tone across both classifications, but otherwise both outputs are decent replications of the ground truth images.

3.3. General Loss of Vibrancy

The general loss of vibrancy in my model is the main differentiating qualitative factor that I uncovered. Only blues ever tend to have vibrancy across all my samples. The

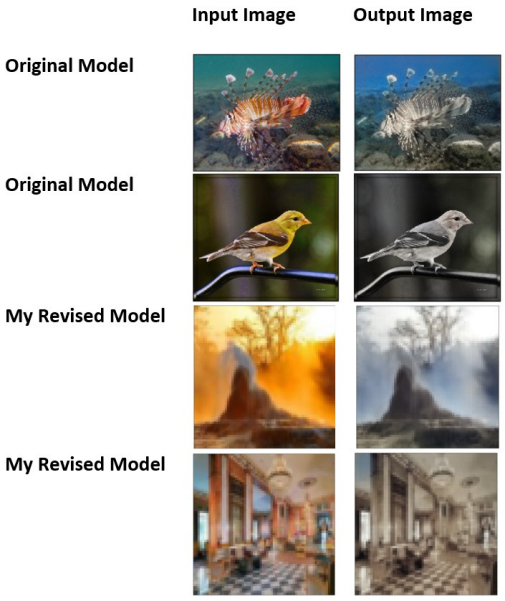

Figure 2. Both my revised model and the original model without the color re-balancing technique consistently classified reds and yellows as blues or grays

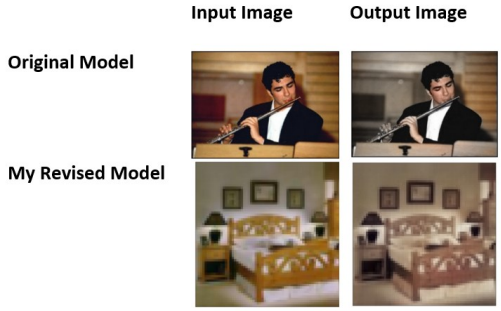

Figure 3. Both my revised model and the original model without the color re-balancing technique performed relatively well with neutral toned images.

original model does not tend to suffer the same dull consequence. My theory as to why this is occurring is since the MSE is specialized for regressions instead of classifications, I think my output is converging much slower than the original converges. So, when you also consider the hundreds of thousands of less training images, this inaccuracy should in fact be expected to follow, therefore leading to those incorrect color predictions and more monochromatic results.

3.4. Edge Cases that Excel

My revised model tended to excel in a very specific subset of test cases: dark pixel values, white pixel values, or predominantly monochromatic images with no presence of red nor yellow,

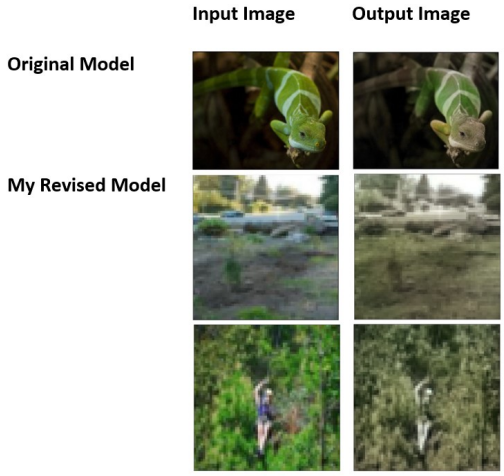

Figure 4. My model failed to keep the same amount of vibrancy as displayed from the original model without the color re-balancing technique.

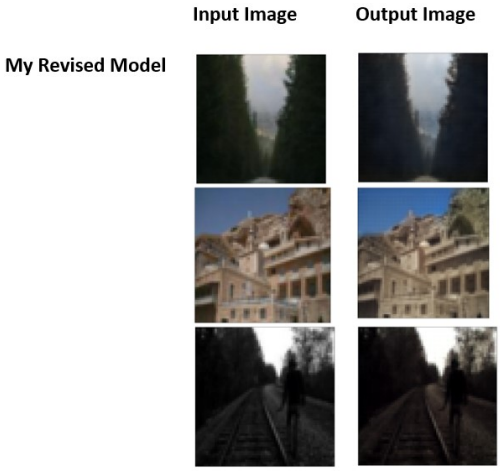

Figure 5. My revised model excelled on edge cases that were either dark or predominantly tan and blue.

3.5. Limitations

I did not have access to the same computational resources nor allotted time that the researchers of Berkeley had when developing this paper. To achieve my goal of testing a different loss function with this model in the allotted time, I had to make several adjustments to my approach. These adjustments include the number of training images, my neglecting to implement their class-re-balancing method and annealed-mean classification method, and my optimizer tuning.

The researchers of this Colorization Model trained their model on over 1,000,000 images. I do not have the computational resources to achieve even a fraction of the processing power nor memory required to train on 1,000,000 images. I only trained my version on 30 Epochs of 256 batches of 64x64 pixel images. Obviously, this is considerably smaller, but I am not too concerned about this smaller number of images because my losses tended to oscillate by only about 1% during training. The more pressing concern would be the initial mismatch shapes of the data-sets. I had to slightly change the kernel sizes and strides of certain layers to achieve the convolutional layer outputs. However, I do not think these slight changes (+/- 1) would make a large difference over these 30 Epochs.

The most obvious shortcoming is my neglecting to implement their class-re-balancing method. The researchers provided data with and without the re-balancing, and while the re-balancing performed significantly better, I was more interested in just changing the loss function, so this re-balancing method felt out of scope for my project. The researchers also implemented an annealed-mean classification step in their network architecture, but even they admit in the research paper that the impact is largely unnoticeable.

Lastly, I had limited time and resources to finely tune the ADAM optimizer parameters in Google Colaboratory. I ended up attempting 3 combinations of parameters on different sets of input images, and ultimately deciding learning rate=0.001, betas=(0.9, 0.999), and epsilon=1e-08 was visually the best. In hind sight, I would have benefited by keeping the input images the exact same while testing these parameters, as well as lowering the Epoch to around 10, when it typically starts stabilizing, to achieve the most efficient optimization parameters. This decision would have saved me a lot of time in the long run to then spend more time to test more thoroughly.

References

[1] bfortuner. Machine learning glossary. 2021.

[2] Jason Brownlee. Gentle introduction to the adam optimization algorithm for deep learning. July 2017. https://tex.stackexchange.com/questions/131471/how-to-cite-articles-from-a-well-known-website.

[3] Rafay Khan. Why using mean squared error(mse) cost function for binary classification is a bad idea? November 2019. https://towardsdatascience.com/why-using-mean-squared-error-mse-cost-function-for-binary-classification-is-a-bad-idea-933089e90df7.

[4] Grgic Sonja Vukovic Josip Sisul G Zeger, Ivana. Grayscale image colorization methods: Overview and evaluation. ieee access, 2021. PP. 1-1. 10.1109/ACCESS.2021.3104515.

[5] Isola P. Efros A.A Zhang, R. Colorful image colorization, 2016. In: Leibe, B., Matas, J., Sebe, N., Welling, M. (eds) Computer Vision – ECCV 2016. ECCV 2016. Lecture Notes in Computer Science(), vol 9907. Springer, Cham. https://doi.org/10.1007/978-3-319-46487-9$_4$0.

[6] Richard Zhang, Phillip Isola, and Alexei A Efros. Colorful image colorization. In *ECCV*, 2016.

[7] 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. *ACM Transactions on Graphics (TOG)*, 9(4), 2017.