Machine Learning Colorization Engine using U-Net Architecture

Rygel Yance Daniel Pitzele Minsi Hu

University of Maryland, College Park
{ryance, dpitzele, mhu0315}@umd.edu
https://github.com/rygelyance/Machine-Learning-Colorization-Engine

Abstract

This project addresses the challenge of image colorization by using a U-Net neural network to predict color information from grayscale images. Working in the LAB color space, we created a model that learns to infer the A/B chrominance channels from the Luminance channel, using mean squared error loss. We trained separate models on datasets of landscapes, people, and general images to evaluate performance across non-specific, diverse content. Category-specific models produced more accurate and vibrant outputs, while the general model tended toward muted tones due to loss averaging. Future improvements include training with higher resolutions, larger datasets, and using GANs or other custom loss functions to enhance color richness and realism.

1. Introduction

The problem of image colorization presents a variety of challenges and issues. How is one supposed to figure out which color each pixel is supposed to be when you only have a grayscale image to start with? A grayscale image is described with a single brightness channel. Given the brightness of a single pixel, should the pixel be red, blue, or hot-pink? Additionally, the color of a pixel depends on many factors: the context of the image, the object the pixel is a part of, and ultimately the characteristics of all other pixels in the image. The problem of image colorization is inherently under-constrained, which makes a solution to it non-trivial to implement.

1.1. Motivation

Our motivation for this project was to learn at least one method for how the problem of image colorization could be solved. It's a very interesting problem and we wanted to tackle it not just to learn more about it, but also to see if we can apply modern machine learning techniques to solve it.

1.2. Related Work

Some research we have found related to this topic are papers using self-attention (Kumar et al. 2021), generative adversarial networks (GANs) (Isola et al. 2016) & (Mayali, 2022), and an analysis of different loss functions for use in image colorization (Ballester, 2022). Our implementation is different from these other approaches because we created the structure of our model using an approach not seen in the previous works mentioned, that being a U-Net structure. We also elected to train our models ourselves locally rather than using a pre-trained existing model. Our approach is also different from prior approaches in that we specialize our models on specific image types to help increase performance rather than trying to create a model that works across all image types.

2. Method & Implementation Approach

To implement a system capable of colorizing black and white images, we elected to take a machine learning approach using a U-Net neural network model architecture implemented using the PyTorch package in Python and training it on some existing image sets. The general approach is as follows:

1: Convert the original training images from the RGB color space into the LAB colorspace (Luminance and two color axes A and B). This allows our model to only need to predict two unknowns (A and B) rather than three channels (R, G, and B).

2: Create the U-Net model structure using a series of convolutional layers. We used residual blocks and batch normalization to add stability. Additionally, we used max pooling to improve efficiency and shortcut connections to improve gradient flow during training.

3: Train the model to minimize the loss (MSE) between what it predicts for the A/B color axes and the actual A/B color axes for each training image. During training, the model is only given access to the L channel of the image and is required to predict the A/B channels. Each image is resized to 256x256 to allow local training to work on the hardware available. We elected to use just 5 epochs for training as convergence occurred relatively quickly for all training sets used.

4: Once the model is trained, it can be used to predict the A/B channels of any grayscale image in order to perform colorization. The 256x256 A/B color channels are then upscaled back to the original image resolution and combined with the original L channel containing the image detail, then converted back to RGB to be displayed.

2.1. U-Net Architecture Implementation

Our implementation of the U-Net architecture essentially works in three stages, the encoder, bottleneck, and decoder. The encoder progressively reduces spatial resolution in order to increase feature density. The bottleneck acts as a bridge between the encoder and decoder and uses 128-channel deeper layers and the residual blocks in order to perform complex feature extraction on the output of the encoder. The decoder then progressively up-samples to restore the original spatial resolution and produce the activation values. The in-depth specifications of our model are as follows. A "residual block" feeds the input into a convolutional layer, batch normalization layer, ReLU layer, another convolutional layer, and then into a batch normalization layer. The input to the initial convolutional layer is added to the output of the last batch normalization layer. The residual block retains the same number of channels as the input. A "convolutional block" feeds the input into a convolutional layer going from *input_dim* to *output_dim* number of channels, then into a batch normalization layer, ReLU layer, then into a residual block. Our model uses both residual blocks and convolution blocks in the following sequence with output dimensions:

TABLE I

	Layer Description	*Output dim.*
1	Input Layer	256 x 256 x 1
2	Convolutional block	256 x 256 x 32
3	Max pooling block	128 x 128 x 32
4	Convolutional block	128 x 128 x 64
5	Max pooling block	64 x 64 x 64
6	Convolutional block	64 x 64 x 128
7	Residual block	64 x 64 x 128
8	Transpose convolution	128 x 128 x 64
9	Concatenate with output of **4**	128 x 128 x 128
10	Convolutional block	128 x 128 x 64
11	Transpose convolution	256 x 256 x 32
12	Concatenate with output of **2**	256 x 256 x 64
13	Convolutional block	256 x 256 x 32
14	Convolutional layer	256 x 256 x 2
15	Tanh layer	256 x 256 x 2

A visualization of the model architecture can be seen in Fig. 1 below.

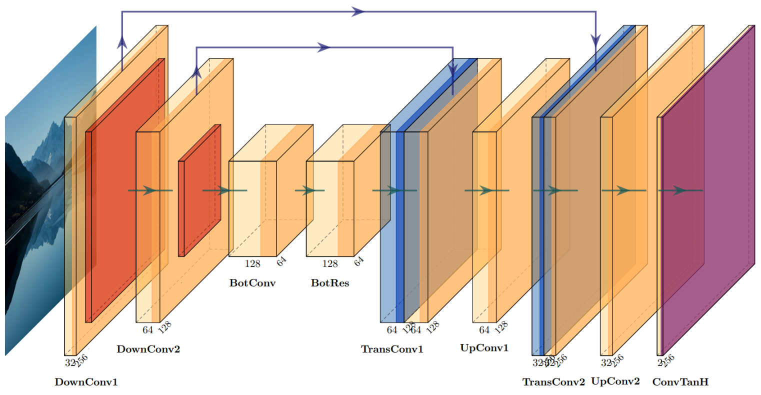


Figure 1. **Diagram of our U-Net architecture.** Each labeled block represents multiple layers, as detailed in **Table I**: *DownConv1* (2 & 3), *DownConv2* (3 & 4), *BotConv* (6), *BotRes* (7), *TransConv1* (8 & 9), *UpConv1* (10), *TransConv2* (11 & 12), *UpConv2* (13), *ConvTanH* (14 & 15). Illustrated with the PlotNeuralNet framework [5].

The model is then evaluated with mean squared error (MSE) with the expected A and B channels of the input image.

2.2. Image Post-Processing

After each image has the A and B color axes predicted by the model, we scale each of these by constant factors, those being 1.6x and 2x. This artificially boosts the saturation and produces more realistic colorizations. Methods to help eliminate this post-processing step are discussed in Section 9.

3. Image Categories & Training Sets

For this project, we elected to try and stay within the confines of two distinct "image categories" to give the neural network the best chance of being able to properly colorize new images. We chose to test the two distinct categories of landscapes and people, alongside tests with a general image set to see how it performs on non-specific training data.

3.1. Landscape Training Data

For the landscapes training data, we utilized an open-source dataset from Kaggle containing a total of 4,300 images of various landscapes. These images originated from the website Flickr, and were spread across the categories of landscape, mountain, desert, sea, beach, island, and Japan.

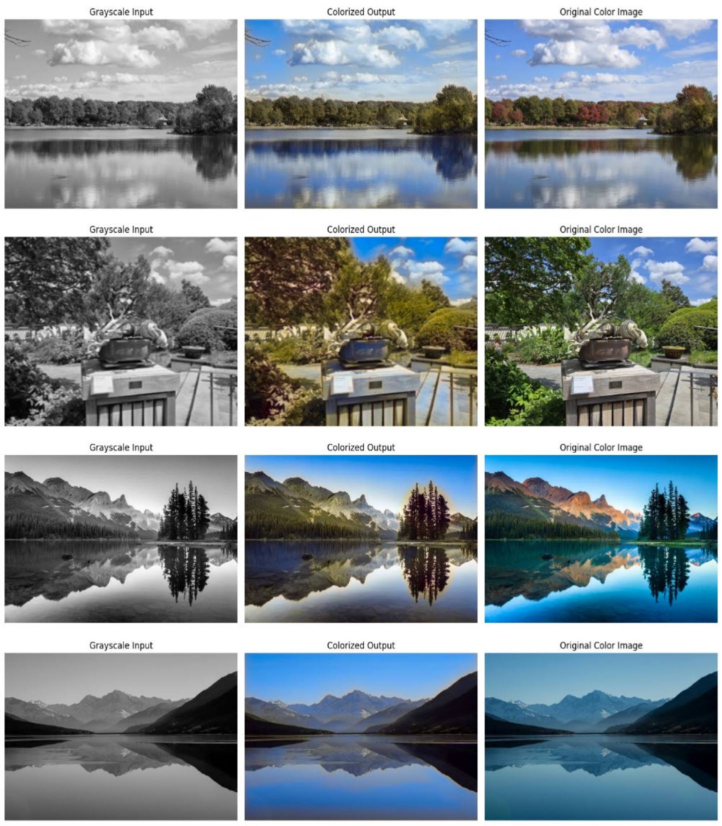
3.2. People Training Data

For the training data on people, we utilized another open-source dataset from Kaggle containing 1,668 images with people as the primary subject of each photo. This dataset was originally intended to be used for gender detection and was stratified by gender but was mixed to be used to train our colorization engine.

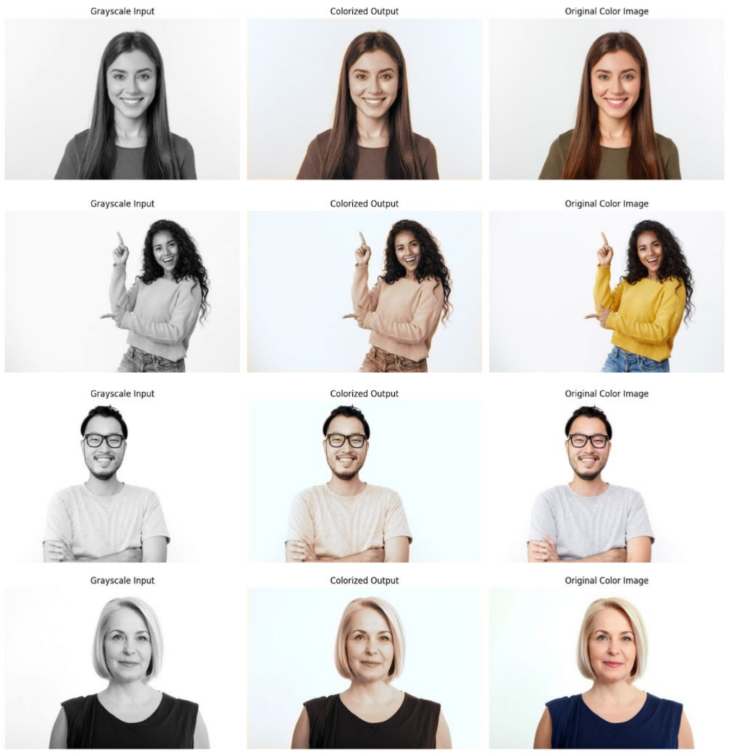
3.3. General Training Data

For the general training data, we utilized a subset of the Common Objects in Context (COCO) dataset, specifically the 2017 validation dataset, consisting of 5000 images in total. While this dataset contained labels for each image to be used in context, in this case we did not make use of them to have a general overall dataset. This set contains not only images of people and landscapes, but also images falling into other categories. For example, there's images of animals, vehicles, rooms, and objects. This is our "stress test" for our colorization engine, as the model would have to minimize the error across different types of images at once.

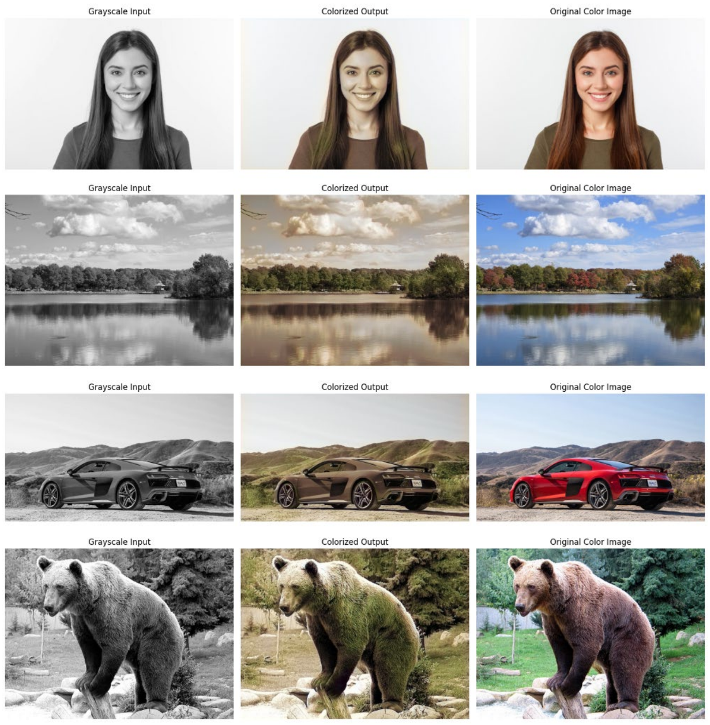
4. Results on Images of Landscapes



5. Results on Images of People



6. Results on General Images



7. Discussion of Results

Overall, we're quite pleased with the results that our implementation was able to achieve given the limitations we encountered during this project (discussed in section 9). When trained on specific image categories, the colorization engine performed well and produced results that were often remarkably similar to the original images. In cases where they differed, the generated colorings were plausible given the context of the photo. The images chosen for the results section were images not in the training set that gave the best subjective results when colored.

7.1. Landscape Images Discussion

The landscape images generally performed quite well, but there are still a few minor imperfections to note. The main imperfections when using this model were the overuse of blue in some parts of the image and a "color halo" around certain edges. The bias towards blue makes sense with the training data containing many images of blue skies. It sometimes spills into areas where it shouldn't be, such as the light wooden planks in the image of the bonsai tree. This might've been helped somewhat by a few more training epochs. The color haloing around some edges and objects is present too, such as around the trees in the third image and around the branch at the top left corner of the first image. This is probably due to the low spatial resolution used during the training and colorization process (See Sections 8.1 & 8.2). There simply isn't enough resolution to color in small features and details properly.

7.2. People Images Discussion

The images of people also seemed to perform very well overall, producing plausible colorizations for skin and hair. Again, the spatial resolution seems to be an issue, as the smaller details such as eye color are entirely lost. Interestingly, the white backgrounds gain a slight blue tint that is not present in the original color image. We theorize that this has something to do with some images of people being taken outdoors (i.e. sky in the background) being present in the training set, leading to a slight bias towards a blue background.

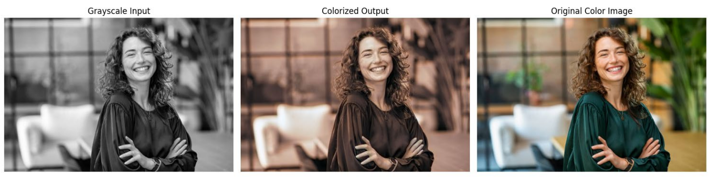
7.3. General Images Discussion

As can be seen, the general image model doesn't perform very well. This has to do with the loss function and is further explained in the discussion section. In essence, using either an L1 or L2 (MSE) loss leads to the model minimizing the loss across all images in the general training set by consistently guessing average colors. This means the A color channel would often be the average of red-greens, which are browns, and the B color channel is often the average of blue-yellows, which are greens. This explains why we're seeing muted shades of browns and greens being placed across most of the images. There might also be a slight bias towards greens and yellows from the presence of greenery and nature within the training set. This helps explain why trees and greenery, like in the second image, are frequently colored more correctly than other objects.

Another reason why the general image model doesn't perform as well as models trained on specific datasets may be due to the lack of model complexity. A deeper, more complex model given more training time may lead to the general image model performing just as well as the models trained on specific datasets.

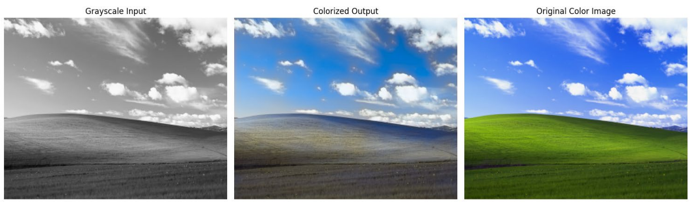
8. Edge Cases & Interesting Behaviors

8.1. People Edge Case



As can be seen in this case, we found that for any image of a person with a complicated background (i.e. any background that is not a solid color), the model would always just tint the entire image a skin-tone color. This is likely due to the training data, which consists primarily of stock images using solid-color backgrounds. The model simply doesn't have enough data on how to color in the background in these cases. A different, larger training set than the one we used would likely produce better results on images such as these.

8.2. Landscape Edge Case



As can be seen with this particular edge case of the famous Windows XP "Bliss" wallpaper, this model doesn't seem to recognize the grass as grass. The model is likely recognizing this as a mountain image and coloring it as rock or stone instead of grass. The grass in this image might be too short that it appears smooth enough to trick the network into thinking it's stone.

9. Limitations & Future Recommendations

While we're quite happy with the results that we were able to produce, we encountered many interesting problems and limitations that warrant a discussion here. The main limitation of our particular implementation and approach is the fact that each type of image needs a specific set of training data and a specific model which we need to switch manually. In theory we could train several sets of models for different situations and combine it with feature detection to automatically detect and switch to the correct model, but there are better ways to accomplish a "general" image colorization which we'll discuss later.

9.1. Spatial Resolution Limitations

In addition, we found that the addition of extra convolutional layers (4th and 5th) to the model did not help with the performance of the engine in any significant way. This likely stems from the low input resolution. As stated before, for our training architecture to work, all images must be compressed to the same, low resolution of 256x256 to be able to be trained locally on the computers at our disposal. At that low a resolution, some of the finer features that would benefit from more convolutional layers are simply lost.

9.2. Hardware Limitations

The limiting factor preventing us from using higher resolution inputs for training is the amount of video memory (VRAM) available to our team. The GPU with the most VRAM our team had access to was the NVIDIA RTX 3060 with 12GB of VRAM. We attempted training using 1024x1024 and 512x512 resolution images, but they either had memory errors or ran too slowly to be feasible. To accommodate this, we landed on 256x256 as our final image resolution for the models. During the training process we observed that this model uses around 10GB of VRAM.

9.3. Loss Function Limitations

Another interesting thing to note is the lack of saturation produced by the colorization engine in its default state. The raw images produced by the engine are usually lacking in saturation, even if the colors are there and are plausible colorizations given the context. To adjust for this, we boosted the A and B channels in post, essentially just increasing the saturation for better looking results. As mentioned before, this likely has something to do with the loss function used (the model tries to minimize the loss by just using the "average" or muted colors). We found a model during our research which also implemented image colorization using a U-Net architecture but coupled with a Generative Adversarial Network (GAN), to solve the problem of the model just taking the average colors across the training set to minimize the loss (Mayali, 2022). Using a GAN for this task is a very interesting concept and has many other applications, many of which are explored in the pix2pix paper (Isola et al., 2016).

9.4. Future Recommendations

In the future, using some kind of custom loss function for the model would likely be very beneficial and serve to produce more vibrant colors overall. In addition, finding optimizations to train the model more efficiently and with higher resolution base images to improve spatial resolution would be very beneficial. Finally, larger training sets of images are recommended. The aforementioned Isola paper used the entire ImageNet dataset of 1.3 million images for use in various tasks including colorization. Ours only used relatively small datasets by comparison, and larger ones would likely produce better results.

10. Division of Labor

Overall, most sections of our project were contributed to by all three of our group members. The project's code implementation had all of us working simultaneously using GitHub for version control. The sections were as follows:

- Model design: Daniel, Minsi, Rygel
- Model training: Daniel, Minsi, Rygel
- Model testing: Daniel, Minsi, Rygel
- Report writing: Daniel, Minsi, Rygel
- Video voiceover: Daniel, Minsi, Rygel
- Video editing: Rygel

References

[1] M. Kumar, D. Weissenborn, and N. Kalchbrenner, "Colorization Transformer," *arXiv.org*, 2021. https://arxiv.org/abs/2102.04432

[2] C. Ballester *et al.*, "Analysis of Different Losses for Deep Learning Image Colorization," *arXiv.org*, 2022. https://arxiv.org/abs/2204.02980

[3] P. Isola, J.-Y. Zhu, T. Zhou, and Efros, Alexei A, "Image-to-Image Translation with Conditional Adversarial Networks," *arXiv.org*, 2016. https://arxiv.org/abs/1611.07004

[4] Bercay Mayali, "GitHub - mberkay0/image-colorization: Image colorization using GANs (Generative Adversarial Nets).," *GitHub*, 2022. https://github.com/mberkay0/image-colorization (accessed May 17, 2025).

[5] H. Iqbal, "PlotNeuralNet: Latex code for drawing neural networks," GitHub repository, 2018. [Online]. Available: https://github.com/HarisIqbal88/PlotNeuralNet

Training Data Sources

[6] General Images Training Set (COCO2017 Validation): https://cocodataset.org/#home

[7] Human Images Training Set: https://www.kaggle.com/datasets/snmahsa/human-images-dataset-men-and-women

[8] Landscape Images Training Set: https://www.kaggle.com/datasets/arnaud58/landscape-pictures