Black and White Image Colorization with Convolutional Neural Networks

Jamie Santiago & Jane Sieving

Machine Learning, Fall 2019 @ Olin College

Project description

The main goal of our project was to create a Machine Learning Program (MLP) that used Convolutional Neural Networks (CNN) to add color to black and white images. In doing so, we hoped to increase our understanding of the ideas we encountered during class. Along with creating an operational Image-Colorizing MLP, our understanding of the concepts behind the work grew as we did research into other successful applications of CNNs to colorize grayscale images.

Model & Methods

In order to create a convolutional neural network capable of colorizing images, we implemented an architecture similar to the ones used by Emil Wallner in his blog post “Colorizing B&W Photos with Neural Networks” and Richard Zhang et al. in the paper “Colorful Image Colorization”. For our first iteration, we used the former resource as a conceptual guide while using the architecture of the latter. Our network consists of 8 “groups” of convolutional layers. In each group, 2 or 3 convolutional layers are applied, with batch normalization applied after every group but the last. As the information moves through the layers, kernel size and stride are varied to make the image smaller (that is, gathering finer details in the early layers and larger details later.) Before the final layer, the image is scaled up by a factor of 2. After the final layer, a convolution with a 1x1 kernel size and 2 output channels is applied, to turn the 128 channels of existing data into 2 channels representing the A and B channels in LAB color space. Finally, these color results are each scaled up by a factor of 4, to fit the original size of the image.

To calculate the loss from the estimated output, we use the mean squared error between each pixel in the 2 channels, A and B. Our method of determining output color and calculating error is a large simplification from Zhang’s method. In that model, the AB color gamut was split into 313 “buckets”, and for each pixel a probability distribution was computed over all of the buckets. The loss was computed using a cross-entropy loss function which compared the predicted bucket to the actual bucket. The loss function was also modified in that model to reward less common colors, in order to avoid muddled, desaturated results. We would have liked to try this or another method to see how it improved the results, but we did not sufficiently understand

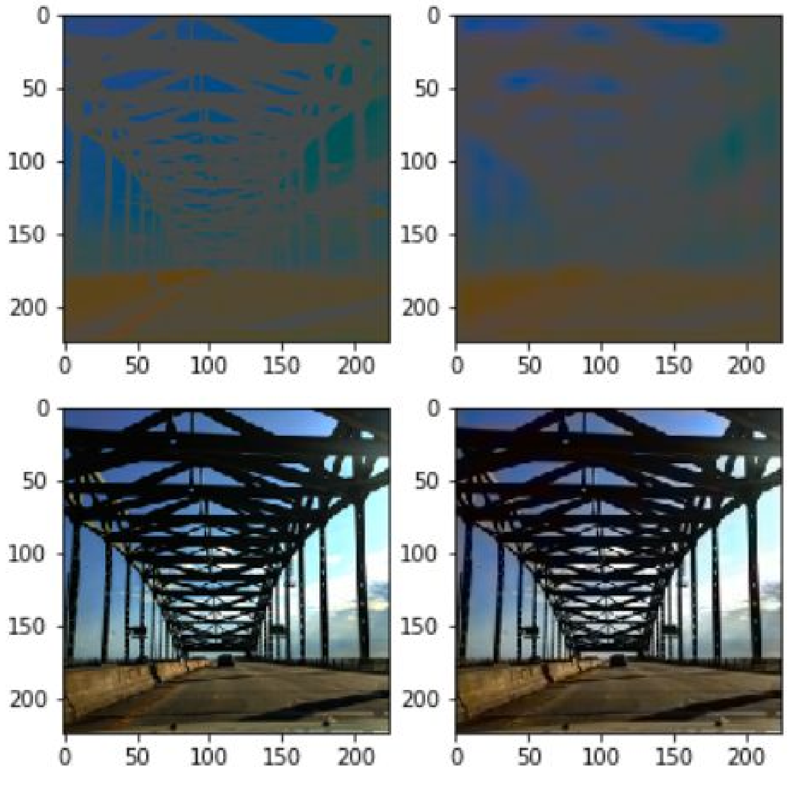
We found our dataset from an Image-Colorization Kaggle Project by Shravankumar Shetty. This dataset was appropriate given that the images were a convenient and consistent size for our use (224x224 pixels), and they were already split into separate files containing grayscale image data and color-only image data. This dataset came in the form of 4 folders, 3 of which were AB-color channels of images and the last being the grayscale channels of the same images. There were 25,000 pairs of grayscale (L) and color (AB) images total. The images, as far as we have seen, feature a large variety of scenes ranging from portraits to landscapes, with varying objects and people. The images are all NPY files so it was not as easy to view them as if they were JPEGs. This meant that we did not have an understanding of what subjects were more or less represented in the dataset. Because of this, we are under the impression that a more deliberate and diverse dataset would be necessary in the future if we were to continue to improve this model.

Results/Analysis

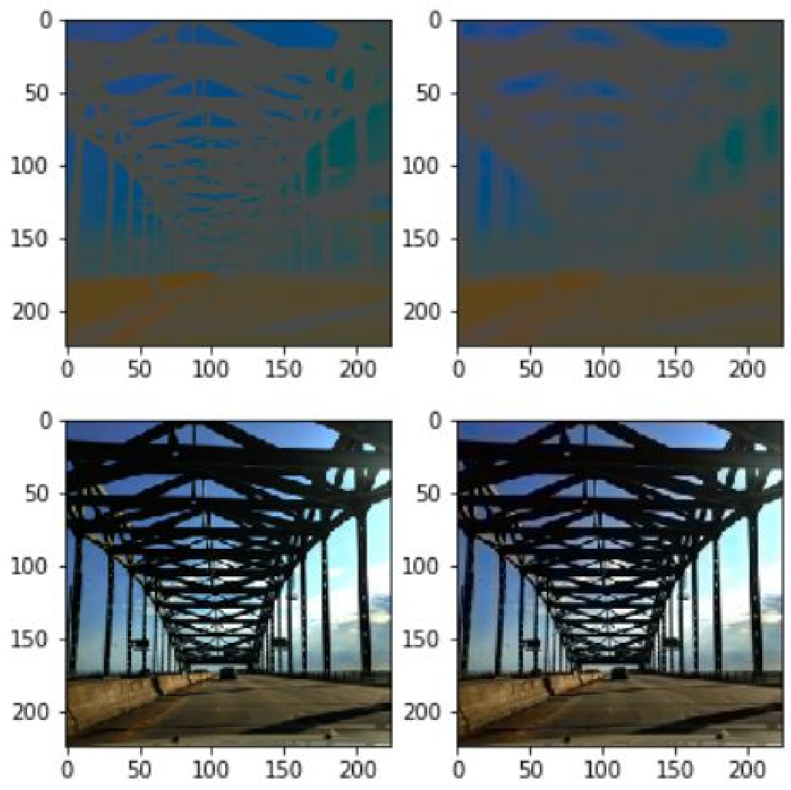
In the end, our program did not succeed in learning the colorization of images and transferring that learning to new images. There were instances in which our model produced fairly appropriately colored results, but this did not work for large sets of images, and it seemed to be memorization more than "learning" that occurred.

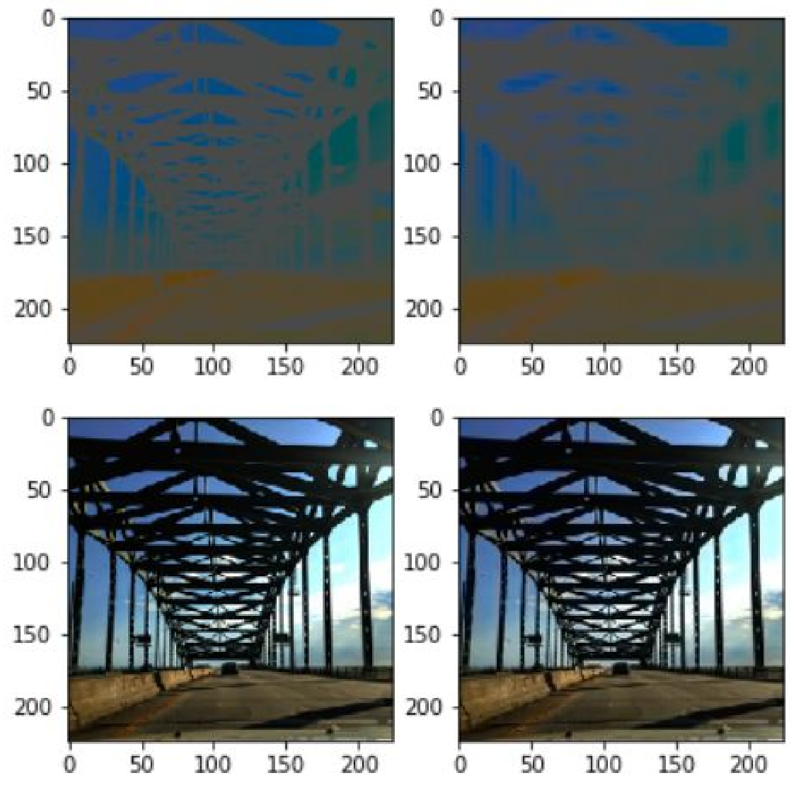
We managed to get our model to overfit to a single image quite well. In fact, the predicted image could be a real image of a bridge. Below, the result of training the network for 100, 200 and 400 epochs are shown. As you can see, the model learns the coloring rapidly and produces an accurate result early on. We achieved a quickly-adapting model by lowering the learning rate to 10^{-4}, which helped the loss descend more consistently. We also reduced the number of kernel filters applied at each layer from multiples of 64 (as in the Zhang paper) to multiples of 16. For small numbers of images, this simplification created superior results in less time.

Originals are on the left; predictions are on the right. The top images average the original lightness values, for easier inspection of color accuracy.

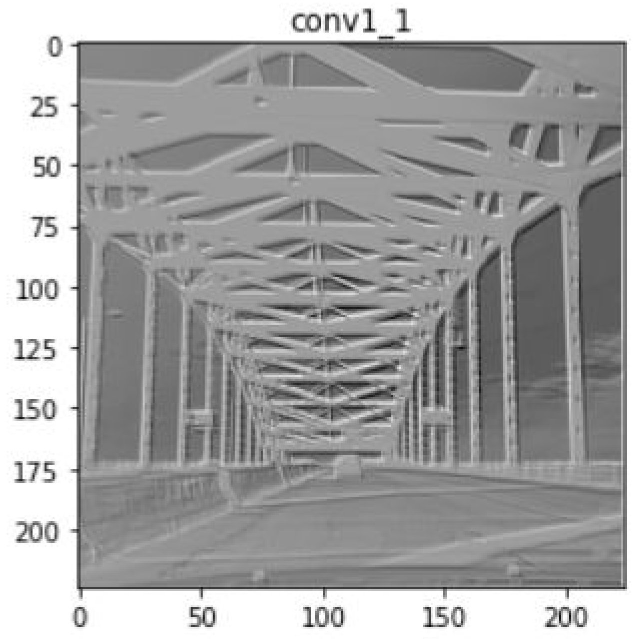


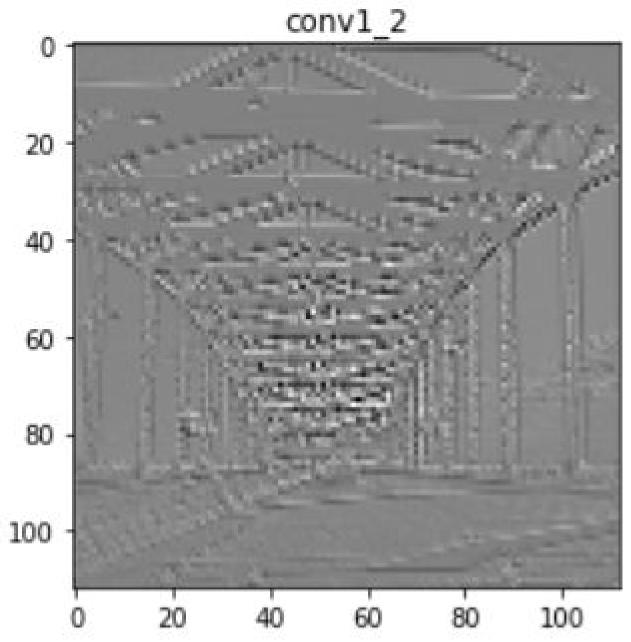
100 epochs: loss ~= 40

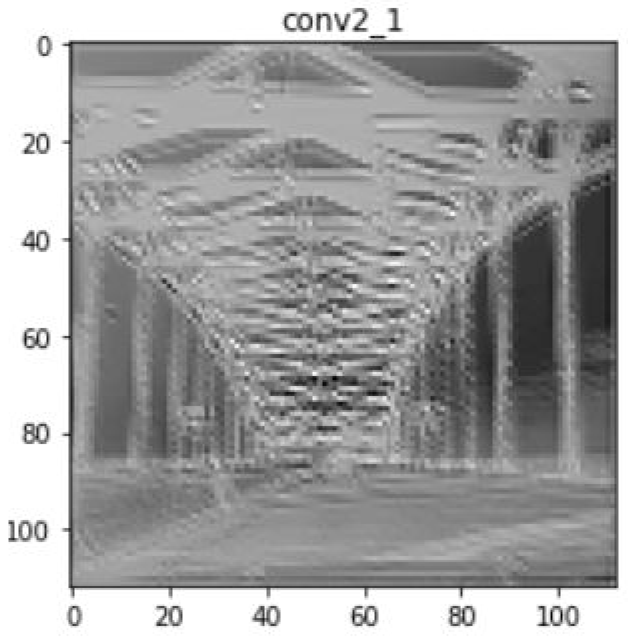

200 epochs: loss ~= 31

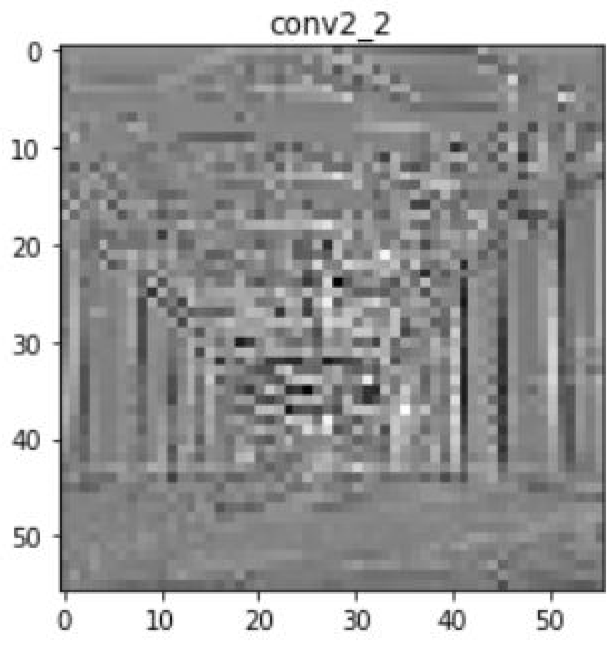

400 epochs: loss ~= 24

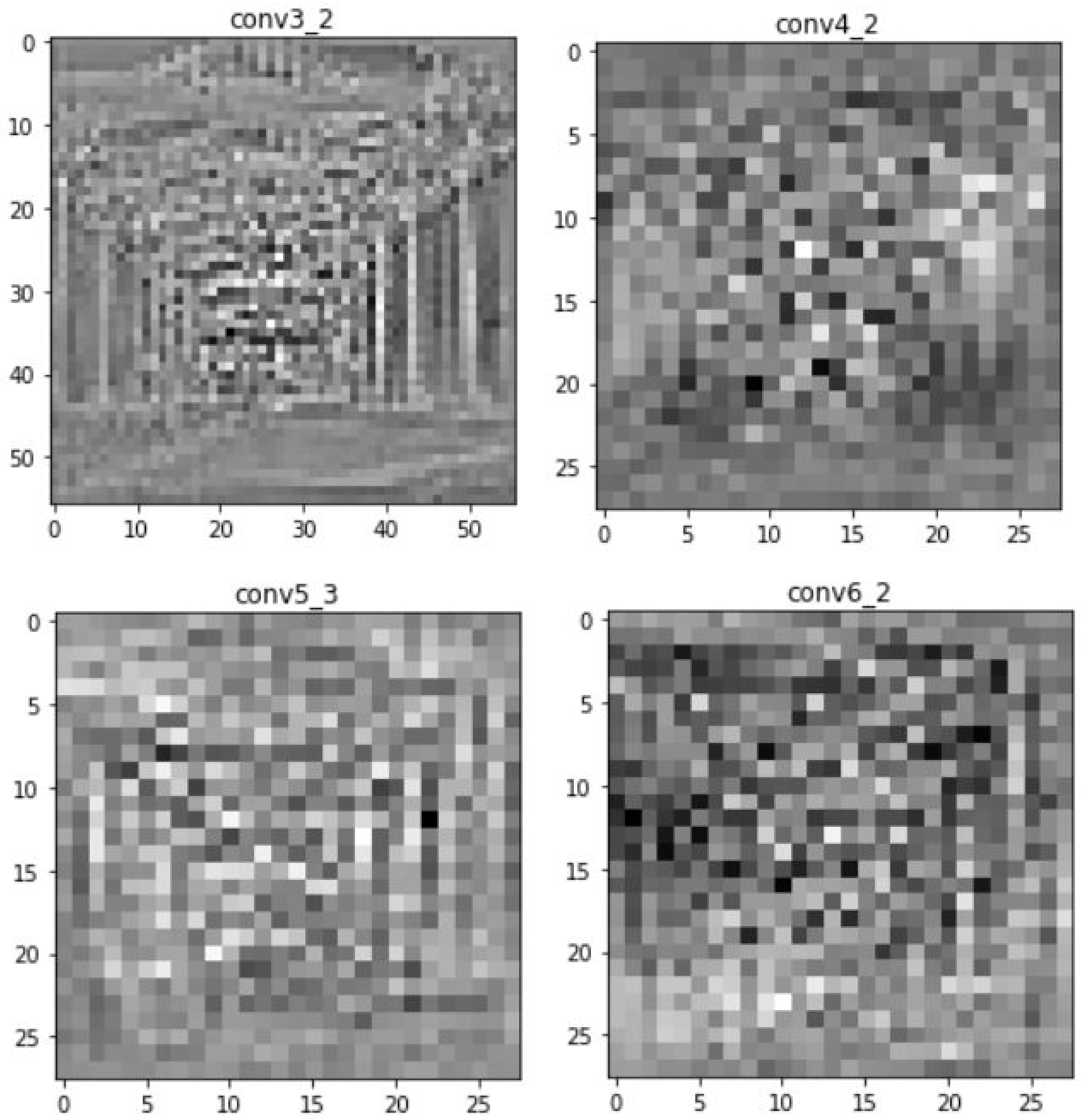
This test gave us confidence that our model was, at least, recognizing where an image should have certain colors, and reproducing this with improving accuracy over training. To further inspect our model, we visualized the activations of the image going through the model. Of many convolutional layers, we randomly chose 1 kernel to show from each of them. Some are shown here, but the rest can be viewed (and more can be produced) in our notebook.

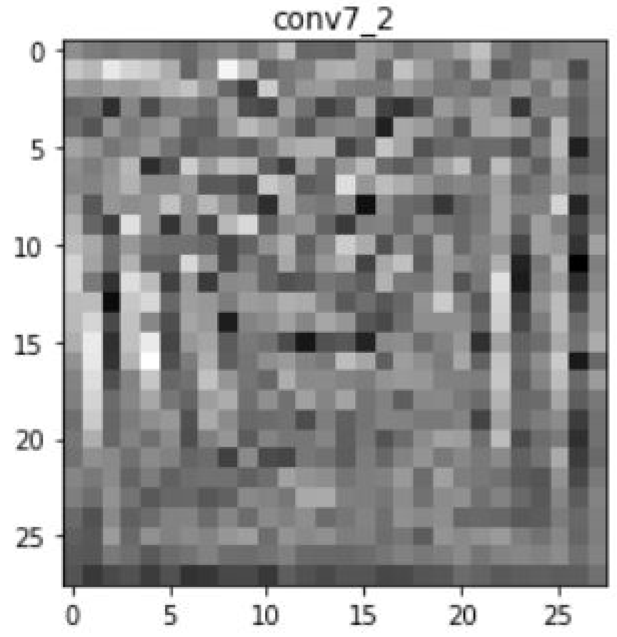


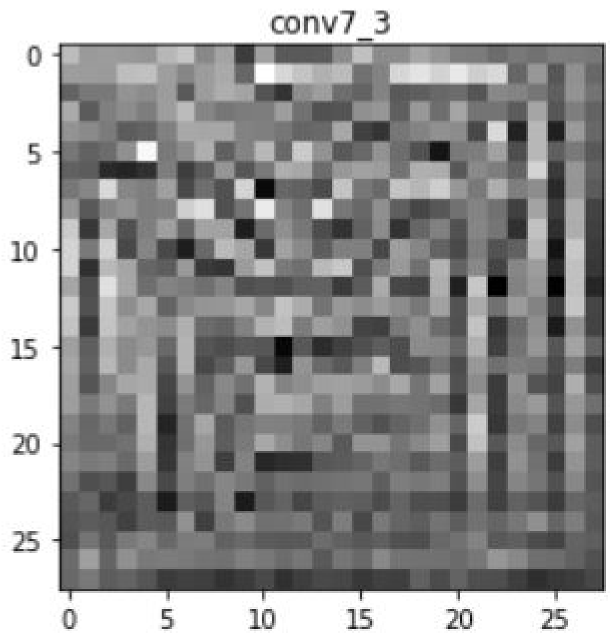


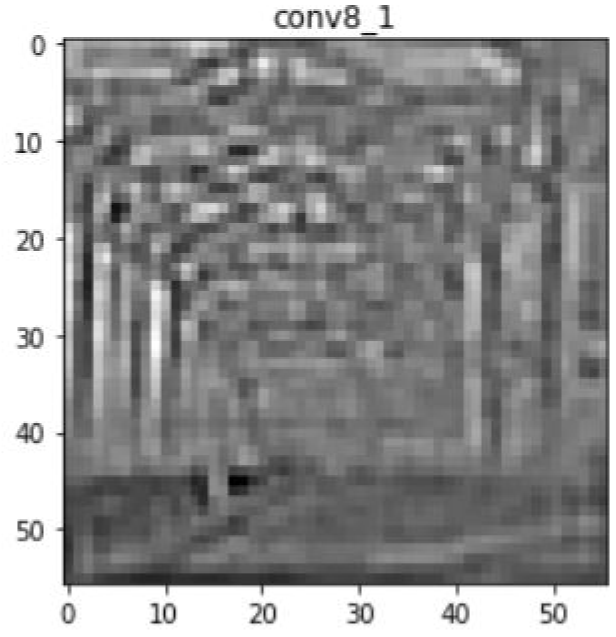


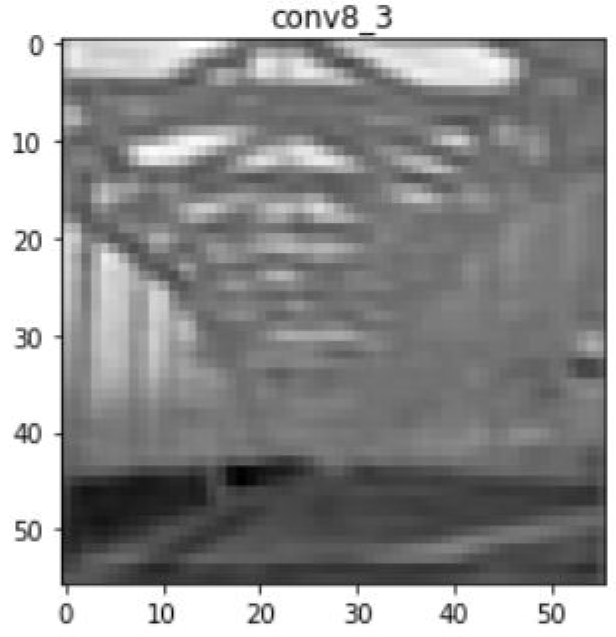



conv3_2
conv4_2
conv5_3
conv6_2



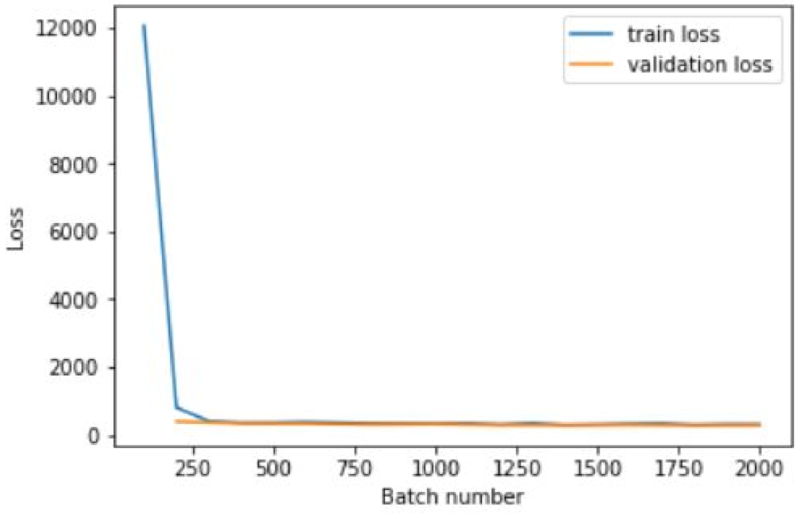




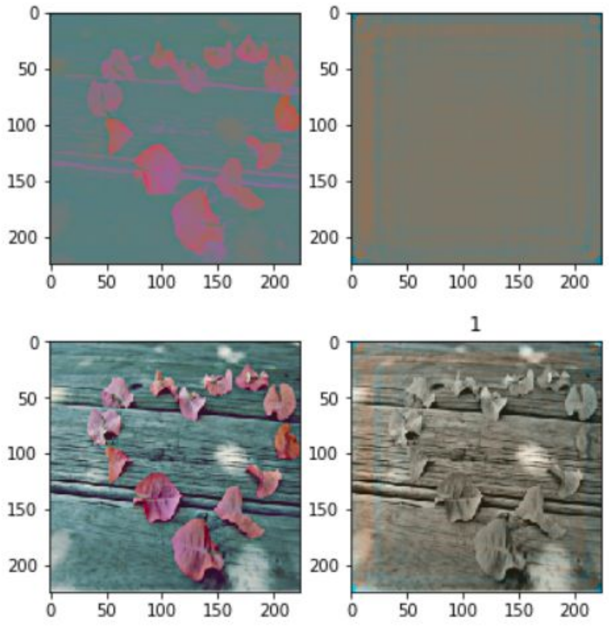


This is a lot of information, so we'll try to justify it with some general observations from seeing more of the filters: layer groups 1 and 2 seem to detect lightness/darkness and simple features like edges. 3 and 4 are probably detecting more advanced textures and patterns as a composite of those. In layers 5 and 6, it is hard to see any pattern in the activation, making it hard to interpret. However, this apparent randomness must have useful meaning for the model, because in layer 7, it is again possible to make out the structure of the original image. In layer 8, the activations are clearly related to the original features of the model. It is interesting that these representations go from logic to chaos to logic again; this sense-making is of course necessary for a meaningful prediction.

However, when we moved beyond a single image, we found that our model really struggled to make the connections to output correctly colored images. One big reason for this is the lack of training time the model had for the large number of images. We found early on that in order to effectively debug and improve the model, it was necessary to run it (who would have thought?), and because of the long runtime of the model, while debugging we could not give it the time it needed to reach very high accuracy. In fact, it seems that to give the model adequate training time for the thousands of images in our dataset, it would have taken days. We can hypothesize that it takes the model at least 100 epochs to learn to accurately color an image, and this number would probably be much higher when it had to reconcile different inputs (as opposed to overfitting). A single epoch with our full dataset (25,000 images) took around 40 minutes. 10 epochs with a subset of 1000 images took around 15 minutes. Interestingly, we initially thought we had been successful, as we saw the loss appear to "bottom out" around 300-400:



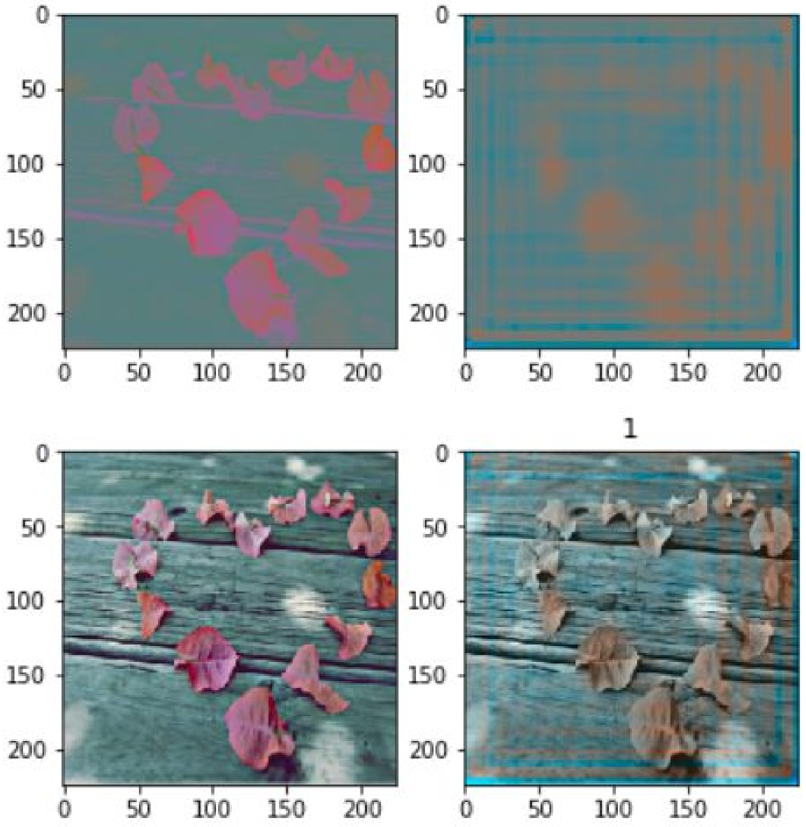
When in fact, the coloring on images was quite meaningless, as seen here:



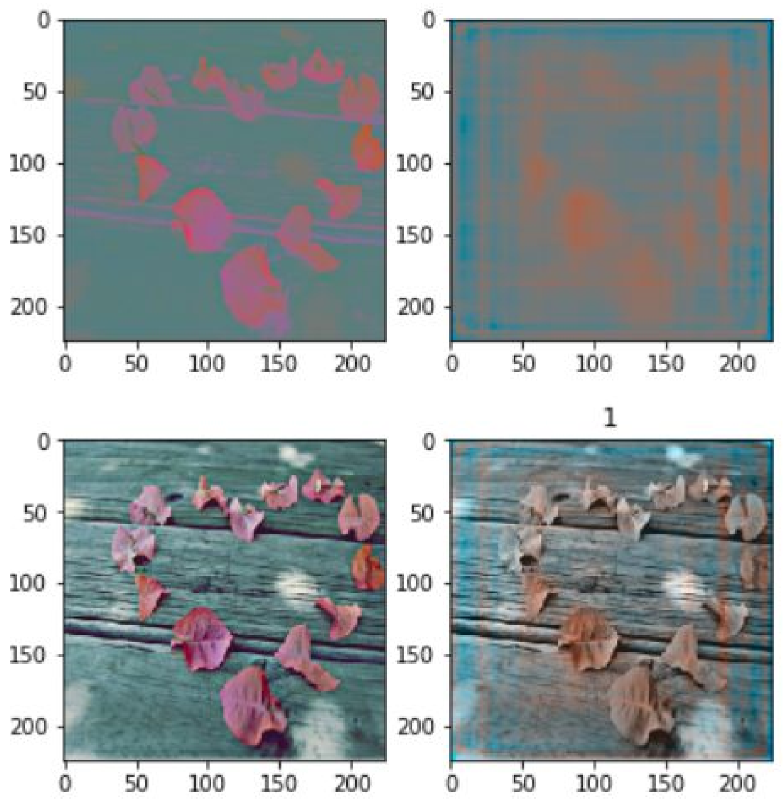
At the time that we saw this type of result, we did not know what loss to expect. That led us to run the experiment with the bridge image shown previously, investigating qualitative and quantitative accuracy over training time. We learned that around loss=200, there was visible evidence of correlation between the image and the predicted coloring; around 100, the model did a decent job of coloring (well, about what you'd get if you gave a 4-year-old 2 crayons) and at 50 the coloring might be said to be "good".

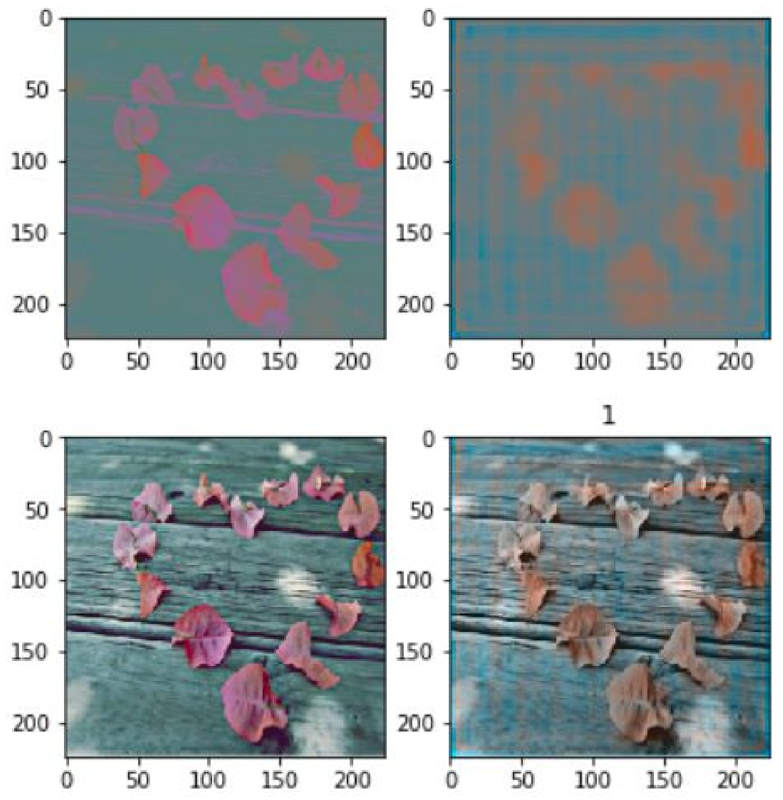
We began investigating our model's behavior on much, much smaller datasets, to see under what conditions it could "learn" something about the coloring. This gave us a chance to investigate the effects of training parameters, which we later used to optimize the bridge image as shown above. We quickly realized that small batch sizes were much better and did not increase the time by much, so we settled on a batch size of 5-10. We found a learning rate of 10^{-4} to be fast enough but also accurate, and learned that reducing the kernel count in our model helped it learn more quickly and much faster for small sets of data. However, more kernels would likely be necessary to account for all the information in a full dataset for a working colorizer.

We learned that for 10-20 images, 200-400 epochs were necessary to get images that were showing some sort of correlation (keep in mind the image outputs were not good, but showed some intentional color matching). Training with 40 images, there was occasionally some hint of correlation with 400 epochs of training. Here is one image result from a few different training setups:



10 images - kernel multiplier = 32 - batch size = 5 - learning rate = 10^{-4} - 200 epochs


20 images - kernel multiplier = 32 - batch size = 5 - learning rate = 10^{-4} - 200 epochs


20 images - kernel multiplier = 16 - batch size = 5 - learning rate = 10^{-4} - 400 epochs

After that we hit a ceiling: if we tried to train with more images, with a reasonable training time, all the model would output was randomness. We concluded in all cases that memorization was occurring rather than informed guesses based on features as we had hoped for. When the model was trained and tested on different sets of images (rather than recoloring the images it was trained on), the results were random. So while we know the model is capable of colorizing an image pretty well, as seen with the overfitting and signs of good memory with slightly more images, we did not have the time or resources to train the model enough for it to meet the expectations we set at the beginning of this project.

Reflection

In the end, we created a model that was pretty good at remembering the colors of a small set of images. The model was successful in terms of being able to add color to images, and getting better at that task the more we trained it. Through this we developed a more concrete understanding of the development of Machine Learning Programs, and the applications of ML in the real world.

In terms of how we could have made this work better, there's a lot that we could do with more time and knowledge. For example, we could have utilized the Transfer Learning technique that Paul shared with us last week, but by the time we realized how much we had under-estimated the challenges, we felt that we should use the time we had left to finish debugging, and did not have enough time left over to incorporate the technique, considering we would have to integrate then inevitably debug. In addition to the time constraint that dissuaded us taking advantage of Transfer Learning, we also didn't currently have a great model to transfer from. The colorization model we were hoping to create is fairly uncommon, and while others have tackled and accomplished this project, our model was structurally different and used a different library, and with the little time we had to complete the project, we decided that is was better to assess what we had learned rather than put in time to adapt someone else's work to make up for our challenges.

While we believe that our model could produce decent results with much more time and training, we know there are ways of processing the images that could improve predictions more by working "smart, not hard". For example, Wallner's model utilized object classification, so that beyond just textures and lightness, the model could also infer colors based on what object it was coloring. This would take time to integrate, due to it involving a whole other machine learning task to write, debug, and tune to balance accuracy boost with speed cost. Nonetheless, if we ever work on a similar project it would definitely be something to try.

In terms of real-world application, image colorization provides a way to add details and discover more about moments in time that are currently lost to us. It can be used to colorize historic black and white images that were taken over a century ago, or old family photos. This provides all interested parties with a more detailed glimpse into the past.

There are also ethical implications which must be considered for these applications. If bias exists within the program, then the colorized outputs would not be reliable. Color signifies a lot in photos, from someone's allegiance being shown in the colors they choose to wear to the pigment of their skin. If the program isn't trained on a diverse, then it might be biased towards specific racial characteristics in terms of skin color. If this tool were used to color historical photos (which is often what people use this technique for), it could under-represent racial groups in a certain historical setting. This might diminish our understanding of the history of people of that race, or prevent people from thinking about the influence of race on historical events. In order to prevent this, one would have to make sure that the training data is as diverse as possible, by throwing in images of people of all shapes, sizes, and physical characteristics.

This links to a question we still have about image colorizing: how accurate can image colorizing can be? Since our model determines based on textures, how close can it get to being 100% accurate? Especially considering that there are so many instances of color and texture not being related, such as something simple like a sweater, where one sweater could come in several different colors, or in skin color. However on the other hand, the model also accounts for brightness, so maybe that is enough when added to texture?

A final little takeaway would be to be more picky about what dataset we choose to use, because when looking at the outputs we realized how unnecessarily artsy the images were, and because of the level of artsiness some of the input images were unexpected colors, with odd lighting. We should have chosen a more natural bunch of images that would be less likely to trip up our model.

References

Emil Wallner, "Colorizing B&W Photos with Neural Networks":
https://blog.floydhub.com/colorizing-b-w-photos-with-neural-networks/

Richard Zhang et al.,"Colorful Image Colorization": https://richzhang.github.io/colorization/

Shravankumar Shetty, Image Colorization data: https://www.kaggle.com/shravankumar9892/image-colorization