Colorizing Images: Convolutional Neural Network

Jacob Baker   
CPSC 4430  
UTCChattanooga, TN  
[hpv212@mocs.utc.edu](mailto:hpv212@mocs.utc.edu)

Yoseph Yemin  
CPSC 4430  
UTCChattanooga, TN  
[hxc513@mocs.utc.edu](mailto:hxc513@mocs.utc.edu)

*Abstract*—In this paper we will be discussing the process used to train a convolutional neural network (CNN) on a set of images, so that it can be used to colorize black and white images. The ability to use machine learning techniques to automate image colorization can be used to streamline the process of turning old movies, or historical images into colored representations, without the typically large amount of manual labor involved. The project was somewhat limited by the computational limitations of the computers available to us, and therefore the images were scaled down to meet those limitations. The results that were produced demonstrate that there is merit to this approach, which could be improved upon, given a larger set of training images and better computational hardware.

Keywords—CNN, convolutional neural network, colorize, image, machine learning, black and white

# Introduction

Advancements in machine learning have opened a world of possibilities for the future of computer science. When utilized correctly, machine learning enables us to train a computer to perform tasks that would otherwise require untold hours of human labor, so that it can be automated, and potentially done in a fraction of the time. Some tasks once considered infeasible due to the amount of data involved, the time required, or the complexity of the task, can be streamlined to such an extent that we are able to make new advancements that were previously unimaginable. One such process, that historically required an extreme amount of manual work, was colorizing images from black and white. Whether it be a series of unrelated still photos, or an entire black and white film. The process involved people hand coloring the individual images meticulously, which could take hours or even days per image, depending on the size of the image and the quality of the colorization.

Using a CNN, we’re able to train a neural network on a large set of images, having it colorize black and white versions, and then compare its results to the original-colored images to measure its own performance. Once properly trained, the model can colorize black and white images in just seconds. This process, however, requires a high degree of computational power, limiting the size and quality of the images used to train the model, to correspond with the available time and hardware. Training can take hours, or even days, depending on the training data used, but once trained, the model does not need to be retrained, and can be used quickly and efficiently.

# Methodology

## Data: Format

For this project we utilized a dataset that contained roughly 25,000 unique images of both cats and dogs. These images were in color and varied in size and shape. However, all of the images were saved with the .jpg extension. The dataset, which can be found [here](https://drive.google.com/file/d/1Ul4DIgCpj8fGm5BKt8mOxsa_GKTIDsB9/view) for download is synonymous with classifying machine learning models, yet for this project we wanted to utilize it for our purpose of colorizing images. Within the dataset, there were a few pictures (< 100) of cats and dogs being held by a person. The different color clothing worn by the people, improved our training data, adding a larger variety of color samples to the model.

A picture containing cat, domestic cat

Description automatically generated

Figure 1 A image within the dataset before we transform it for our model. Color 500X386px

## Data: Pre-Processing

To be able to use these images, we need to first establish a method of preprocessing them and getting all images to a common format to be prepared for passing through the CNN. We used python’s PIL library to reshape the images and convert them to grayscale. We then used the function called image\_to\_array to convert and append them to a numpy array. This enables the neural network to work on data that is easier to process and improves performance. The function takes four arguments, ‘files’, ‘destination’, ‘size’, and then ‘BW’. By using a for loop, we can iterate over every file in the dataset, resize it 256x256px, recolor it to greyscale using PIL, and then save it to a new directory that we called train\_files. We then can load that new folder of preprocessed images into a numpy array, we can save that array to a folder, so that we do not have to preprocess the images for every test run. Afterward we split the data into a training and testing set of 95/5.

A picture containing cat, sitting, mammal, domestic cat

Description automatically generated

Figure 2 The same image as figure1 now in the size of 256x256px

A cat looking at the camera

Description automatically generated

Figure 3 An image that has been reshaped and converted to greyscale.

## Data: Post-Processing

Once the images were processed and converted to a numpy array, we were ready to train our model. After being trained, we used the predict function to colorize the black and white test data. That data then needed to be converted back into an image format and saved to the device. We used the skimage ‘imsave’ and ‘lab2rgb’ functions to convert the numpy array data back into an image format, saving the files as .png.

## CNN Structure

The structure of our convolutional neural network (CNN) ended up being rather dense with over 3.8 million trainable parameters. We implemented the CNN using the Keras API. We started by defining our model which was sequential. This means that the model is a linear stack of layers. Our first layer is the input layer. This tells out model what to expect in terms of the size of the image we are passing though it. For this model, as previously stated, we are using 256x256px images. Thus, out input layer has the shape of (256,256,1). The ‘1’ refers to the number of color channels. Since we are using greyscale, it will be looking for 1. If it were color images, it would be ‘3’. One for R, G, and B. The first convolutional fully connected layer has a size kernel size of (3x3), 64 channels, uses the ‘relu’ activation function, and the padding of ‘same’. We continue with 9 more conv2d layers that get the image down to a size of (32,32,128). We then add out first up sampling layer which is responsible for taking the picture and resizing it back up to our original dimensions. This first up sampling layer gets it to a size of (64,64,64). Next, we have another fully connected layer. Followed by another upsampling layer that takes the size up to (128,128,64). We add two more fully connected layers, and then a final upsampleing layer that gets up to our original size of (256,256).

Table

Description automatically generated

Figure 4 Summary of neural network

The design of the neural network, along with much of the other features, was an adaptation of Emil Wallner’s work using CNNs to colorize images [1]. Above is a summary of the neural network, showing the layer dimensions and number of parameters created by each layer (figure 4). Below is a diagram of the neural network that shows a physical representation of those layers in relation to one another (figure 5).

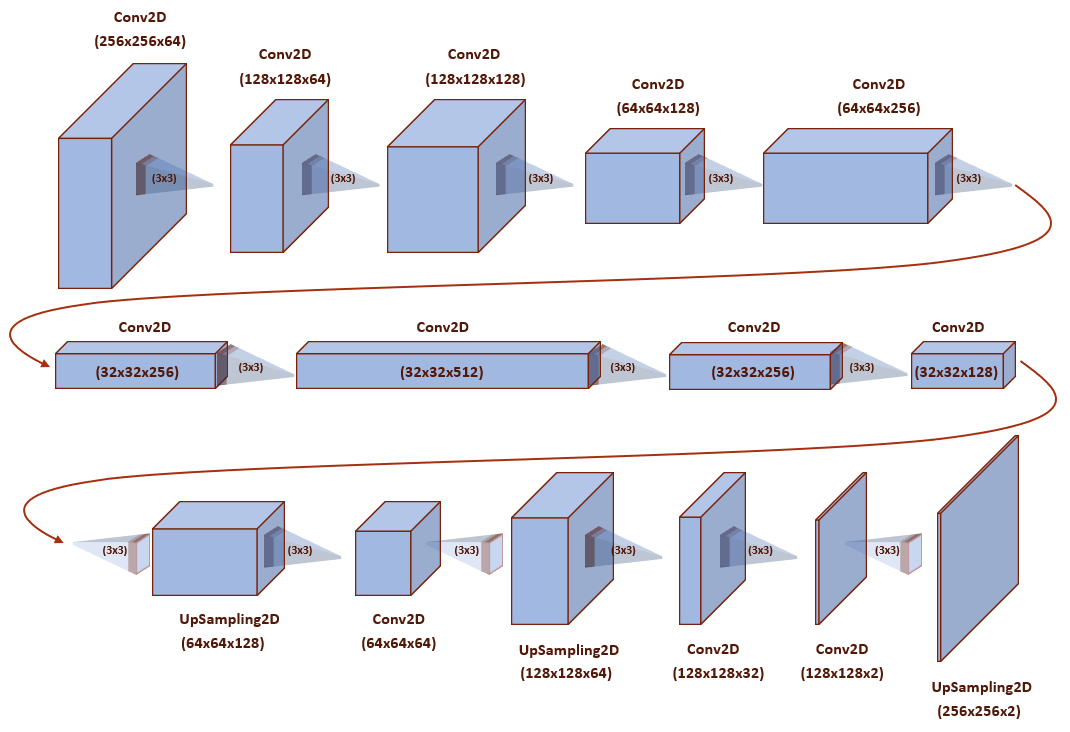


Figure 5 Computational diagram of neural network

# Results

Due to computational limitations, we were not able to train the model on all 25,000 images. Since we chose the size of 256x256px, we were only able to fit roughly 1,500 images into memory for the training of our model. Before we attempted to produce good results, we wanted to ensure that our code was able to process the given data and then output an image. We did this using a single epoch and ran several iterations until we were able to produce a satisfactory output. We trained the model a few different ways and compared results to see which model hyper-parameters produced the best outcomes. In the first configuration we trained the model for only 10 epochs with a batch size of 10 and a step of 10. This gave us the advantage of being able to train the model very quickly, as it only took approximately 2 minutes with these parameters. However, the drawback was that the resulting image wasn’t very well colored, with the model producing a somewhat uniform brown hue over the entire image.

A picture containing cat, indoor, standing, edge

Description automatically generated

Figure 6 after 10 epochs

After several adjustments, our next iteration was to train the model with the same parameters but increasing the epochs to 100. This had little to no effect on the images and made the model take a lot longer to train, roughly 2 hours. Eventually we got better results for our final iteration, where we chose was to train with an increased step size of 20, the epochs the same at 100, and the batch size the same at 10 as we could not increase that due to computational limits. This seemed to produce the best results, but also took the longest to train at close to 4 hours in total.

A picture containing cat, indoor, standing, edge

Description automatically generated

Figure 7 after 100 epochs

# Conclusion

This was in interesting and productive experiment in which we successfully colorized black and white images. We learned about the potential of our CNN model, along with some of its limitations. The resulting model can be trained on any collection of input images and is not restricted to the cat dataset we used for this test.

## Summary of Results

The overall results of our project were promising. As we added more training data and adjusted our hyper-parameters, we saw improved results. The black and white images were being colored in a non-random manner, with mostly appropriate coloring being placed in the correct areas. The results were, however, not flawless, and there was much room for improvement. There were large areas in some pictures that remained uncolored or that were colored incorrectly, and the model had difficulty replicating some of the less common colors appropriately (e.g., blues and greens).

One factor we believe influenced the outcome of our results was the dataset we chose to use. Our dataset consisted of several thousands of pictures of cats in the various environments. Many of the pictures has little color variation, consisting primarily of black, white, and gray. The most prominent color in the dataset otherwise, was orange/brown. This can be seen in the results, with many of the images being filled mostly with shades of orange/brown. It was not however exclusively shades of orange/brown, so there was evidence that we were on the right track, with blues, reds, and yellows showing up occasionally in the generated images.

Even some of our better results were somewhat flat, however, with the colored areas lacking a depth and variety of hues and saturation levels. It seems that the model takes a safe approach to coloring, not risking bolder coloring options, when in doubt.

## Difficulties

A major difficulty we encountered was the computational limitations of the devices available to us. Given the number of parameters our model was processing, we maxed out the capabilities of our devices. This meant that training our model took a prohibitive amount of time per test. As a result, we had to reduce the amount of training data from over 30,000 images, to about 5-10% of that. Even so, training our models took many hours every time we decided to test a new hyper-parameter configuration. AT the best, we were able to run our model one or two times a day, with long periods of waiting between tests.

We initially were optimistic about training the model for many thousands of epochs, but even a hundred epochs took many hours, severely limiting the potential of our model. Many of the test were run using just ten epochs, and we were reluctant to run more than that unless we were confident in our model.

Collecting and processing images to use for training and testing, was a lengthy manual process. Pre-processing the images was another lengthy procedure, which luckily only had to be performed once. The initial dataset took hours to process, and we would have ideally like to use a much larger set of images. This would have taken up a significant amount of hard drive space, as well as memory during runtime.

## Future Ambitions

Given the time and additional computing resources, we are optimistic about improving our model to the point that it could produce results that are comparable to the results of a skilled person hand colorizing black and white images by hand. To accomplish this, we would likely need to rent additional processing capabilities from an online service (e.g., AWS). Then we would be able to increase the amount of training data we provided the model, to include a wider variety of images and image subjects (e.g., people, vehicles, landscapes, etc.). The additional training data should help the model recognize a wider variety of colors, allowing it to produce better results and more accurately colorize most black and white images, regardless of their content.

Another thing that we would like to experiment with is using a generative adversarial network (GAN) and an Auto Encoder to colorize images and compare their results to our CNN model. Those networks configuration seems like they would be well suited for the task of image colorization and may be able to produce better results than the simple CNN model we were working with.

##### References

1. Wallner, Emil. “Coloring-Greyscale-Images/Full-Version at Master · Emilwallner/Coloring-Greyscale-Images.” GitHub, February 23, 2019. <https://github.com/emilwallner/Coloring-greyscale-images/tree/master/Full-version>.