Colorizing Grayscale Images

Ishan Gupta
San Jose State University
ishangupta3@gmail.com

Rakesh Konda
San Jose State University
rakesh.konda07@gmail.com

Abstract Image colorization is known to be a difficult and painstaking process when done by hand. It may take hours or days to accurately colorize black and white photos. In this paper we propose a new solution of colorizing black and white photos using techniques from Machine Learning. Our approach uses Convolutional Neural Networks to predict accurate color channels when given a grayscale image as the input. Previous research has used Convolutional Neural Network Regressor, but we explore other regression techniques to understand strengths and weaknesses. Our primary use of technique revolves around CNN's which is able to extract low-level features from images.

I. Introduction

COLORIZING a black and white image manually can take up to months for a professionally trained colorist. This time-consuming process would be very useful to automate, saving colorists and artists significant time. There have been many services that allow automatic colorization but require some use input to help differentiating different colors in the same image. Since different colors may appear the same in grayscale images, some user input is needed.

In our approach we use machine learning techniques such as MLP-Perceptron and Convolutional Neural Networks to automatically colorize black and white photos without user input. In order for the colorization to work properly we convert RBG images to the LAB color space where L represents the Lightness and a* and b* represents the green-red and blue-yellow color components. This allows for CNN's to learn significantly from the images and predict the ab* color channels given only the L* channel.

Currently, there is a huge set of true black and white images and video available for colorizing. Providing automatic grayscale image colorization will enable historians to gain an insight into the society before their time.

II. Background

A. Related Works

Manually colorizing an image includes using software such as Adobe Photoshop and having previous experience in the domain. It can take anywhere from 20 minutes to months to successfully color a black and white image.

From a technical perspective, the go – to method is using Convolution Neural Networks with no pooling layers. This paper revolves around this specific method, understanding and optimizing it to a deeper level. Topics of high interest within CNNs are number of layers used and creating a generalized model.

GANs is another method which has become increasingly popular in solving many computer vision problems. On a high level, a GAN is a minimax game which has 2 CNNs (generator and discriminator) trying to reach Nash Equilibrium.

B. Deep Learning

Dealing with images calls for many potential features which can be extracted. Using a deep learning approach allows us to understand the image from a much more magnified sense. Learning all these features allows the model to generalize better – encapsulating the goal of the project to colorize authentic black and white content. Using methods such as Convolution Neural Networks we are able to detect features such as edges and particular angles of various objects. The goal is to be able to create a generalized model, allowing for the model to accurately colorize images that the model has never seen before.

III. Dataset

A. Cifar-10

Initially we had planned to use the Cifar-10 dataset which is publicly available dataset with 60,000 images from 10 classes. The images in Cifar-10 are 32 by 32 pixels in size, which are very low-quality images and seem difficult to understand what the images are actually of. We did use this cifar-10 dataset to train our CNN and the results weren't as great. We decided that since the images are of such low quality that our model isn't performing well because of the lack of features. So, we decided to pursue and find another image dataset with higher resolutions.

B. *Tiny ImageNet-200*

The next dataset we were able to find after doing a lot of digging was the Tiny ImageNet-200 dataset. This dataset consists 200 classes of 500 images (64 x 64 pixels) each for a total of 100,000 images. The dataset also included a test dataset which contained 10,000 images from various classes. These images were twice the resolution of Cifar-10 dataset and seemed to be a viable option in training our model. We used this dataset extensively throughout our research in training and testing.

C. *Portrait Dataset*

We explored datasets that contain human faces – in particular named Helen which contained 2000 full resolution portraits. Due to a portion of the dataset containing black and white images, we manually had to delete respective images to keep only colored ones. The full resolution images included further preprocessing such as down sampling the images into a smaller dimension (64 x 64). This allowed for the model to train better and quicker with an increased training set. The shape of the training input is (len(input), 64, 64, 1) and the output resulted in the following shape (len(input), 64, 64, 2).

IV. Approach

Our first approach was using a multi-layer perceptron as the baseline method before moving forward to CNNs. The multi - layer perceptron would be used as a regressor outputting the respective coloring pixels used in the LAB color space. Since, the multi-layer perceptron is not tailored toward images, it creates a joint distribution between the input and the ground truth. Doing this, creates a layer of abstraction between the pixel values and the image itself. Building this multi - layer perceptron resulted in tuning parameters such as the activation functions and number of hidden layers. A clear pitfall of using a multi-layer perceptron is its ignorance toward extracting features from images.

We then decided to change our model and use Convolutional Neural Networks instead. CNN's have the ability to detect patterns, features and objects from grayscale images which enabled us to colorize grayscale images. The results from our CNN were much better than the results obtained from our ANN.

After analyzing our initial results from our CNN, we decided to train our model even further with different activation and loss functions as well as larger training dataset to improve our results.

A. *Data Processing*

First step before we went on to train our ANN was to pre-process our dataset. We had images of various sizes and in order to properly input our images into our model, we first had to resize our images into 64 by 64 pixels and convert our images from RGB to Lab color space.

B. *Cloud Computing*

Training took an extremely long time when training our model on Macbook Pro's, this showed that our computers needed more computing power to help speed the training process. So, we looked into cloud computing and using AWS to help train our model. After doing extensive research we decided to ditch AWS services and Google cloud computing services to use Floydhub's cloud computing power which is built to deploy machine learning models. Using Floydhub's services we trained our model on Nvidia Tesla K80 GPU's which were significantly faster than using our computers. Training models will large datasets and multiple layers become relatively easy after we became familiar with their UI.

FloydHub's computing services aren't free either and we invested around $20 to use their services and to train our models.

C. *Activation Function*

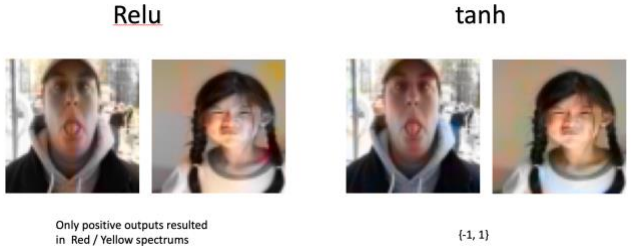
Choosing the appropriate activation function was an integral part of our CNN. We came across many activation functions and we decided to use the rectified linear unit activation function, ReLu, which is defined as follows.

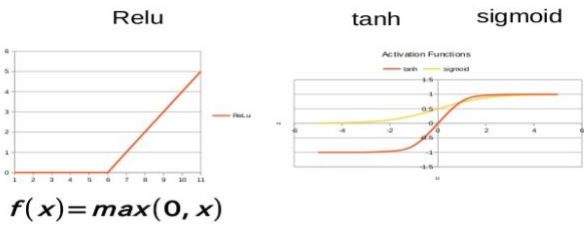
$$(f, x) = \max(0, x)$$

We used ReLu on each of our convolutional layers since we it's a very simple function to compute and it helps accelerate our model to converge during training.

D. *Training and Image Input*

The input for our model was the L* channel and the predictions were the a* and b* channels. After we converted our training images to the Lab color space, we extracted the L* channel into an array of integers, the integers in the array represent the Lightness values and were in the range from 0-100. We used the "relu" activation function for our Convolutional layers and with our final layer having a "tanh" activation function. The reason we had a "tanh" activation function for our final layer was because since "relu" is only able to produce positive numbers, our model wasn't able to predict the respective blue and green color spectrums, and to fix this we used the "tanh" activation function.





Furthermore, we also decided to use reasonable epochs to train our models. We used initial epoch values from 100 – 1000 to train our model and test out how well our model performs with the training data. But as we trained our model with more data, we decided to increase the epochs to 1000+ and specify small but reasonable batch sizes from 25 – 100.

In addition to epochs and batch sizes, one of the most important features was choosing our loss function. After much research we found the Mean Squared Error loss

$$MSE := \frac{1}{n}\sum_{t=1}^{n} e_t^2$$

function was the best choice when training our CNN. MSE is a common loss function that is widely used in CNN's and it can accurately show the loss in training, since loss is calculated by taking the difference of the predicated value and labeled value, then squaring and finally taking the average of all these squared values. This allows for MSE to give us a proper insight into how our model is predicting the a* b* color spaces. By taking the mean squared error our model was more sensitive to outliers was able to learn more and extract greater features.

In order to make our training even more effective, we specified a validation data sample of .1 from our total training sample. This allows for our model to even further fine tune the weights.

The final function we chose was the optimization function and after doing some research we found that using "rmsprop" was the most effective function to use when training a CNN on images.

E. Dropout

To prevent overfitting the model, dropout was integrated between the hidden layers. This allows for some neurons to become muted during the training process - forcing the model to utilize other neurons which might be 'dead' from the RELU effect. We tried multiple combinations of dropout consisting of 25 - 50%, but results did not improve accuracy.

F. Training Samples

We trained our models on various dataset samples to see how effective our model was.

We first performed training on Fish Image samples, then Lizard Image Samples, Portraits samples, and finally on the entire Tiny ImageNet 200 dataset.

V. Testing and Results

A. Multi-Layer Perceptron

Training on a multi-layer perceptron didn't perform too well when we tested various images. The multi-layer perception couldn't detect features from the training images and performed very poorly when tested. Below is an sample

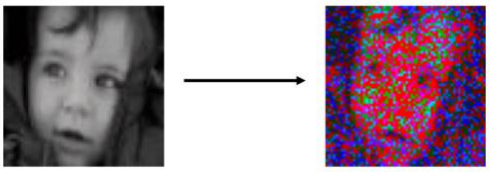


image that resulted from the MLP. A very interesting output which came from the testing set, was the accuracy for edge detection within the respective dataset.

B. Testing on Fish Images

We trained our model on 100, 500 and 1,000 epochs on the sample fish dataset we obtained from TinyImageNet200. There was a total of 500 images where we split the dataset into 450 from training, 50 for testing and 45 images of the training dataset as a validation set.

The results from our 100 and 500 epoch models weren't too impressive and there was some color appearing on our test images, but not a great amount. After testing the images on a higher epoch, 1000, there was a considerable amount of color in many of the testing images. Some of the resulting images were almost near perfect when comparing them to the ground truth.

The results motivated signaled that our model was functioning efficiently and the layers we've provided was sufficient to reproduce colorized images.

Training metric data was also collected from training our model on Floydhub and we found that as we increased the epochs the accuracy of our model also increased. Our 1,000-epoch model had an accuracy of .9283 compared to .8011 from our 100-epoch model. Overall, our validation accuracy on the other seemed to slightly decrease with

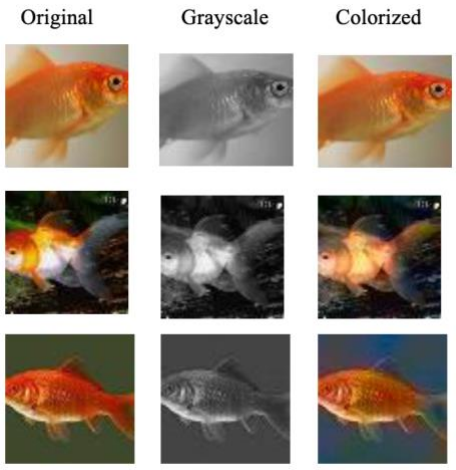


Figure 1: Test result from 1,000 epoch CNN model

more epochs, which signaled that we were overfitting our model.

Though there was some slight overfitting even with dropout set in place in our model, but we decided to keep our 1000 epoch model since it was able to produce accurate color predictions. Being amazed by the results we decided to keep training our model on different datasets and see how it will perform.

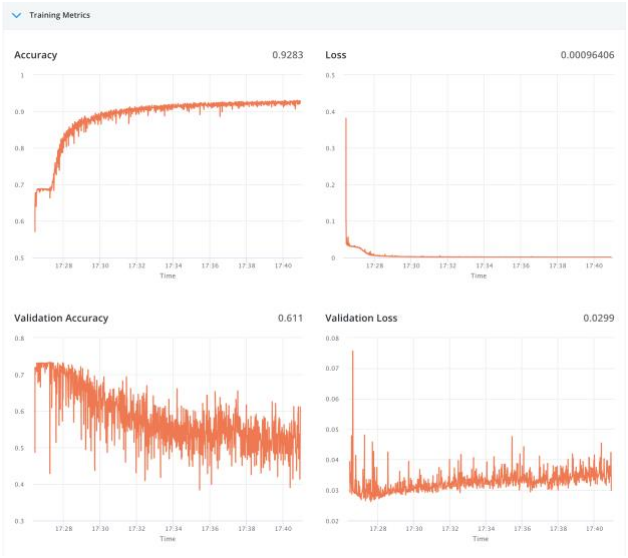
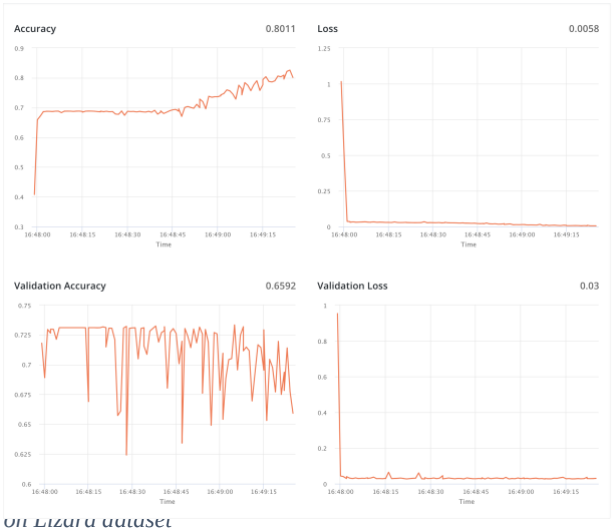


Figure 2: Training metric from 1,000 epoch model

Figure 3: Training metric from our 100-epoch model

C. Testing on Lizard Images



on Lizard dataset

After seeing impressive results from training on our model on fish dataset we decided to train the model on lizard dataset and we've found very similar results. The test results from our lizard dataset was also very impressive as most of the pictures came out accurately colorized. There were some pictures that lacked color in several areas and some pictures that had the most randomized color scheme, but overall our model preformed really well.

D. Testing on Food Dataset

After training our model on the food dataset we found that the results weren't as impressive as our previous test results.

Our model seemed to fall short when coloring pictures with different colors. But our model was able to perform really well with brownish and beige colors.

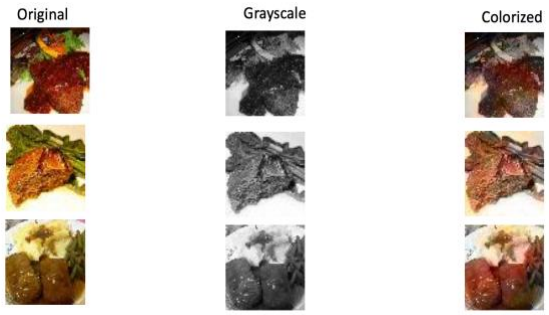


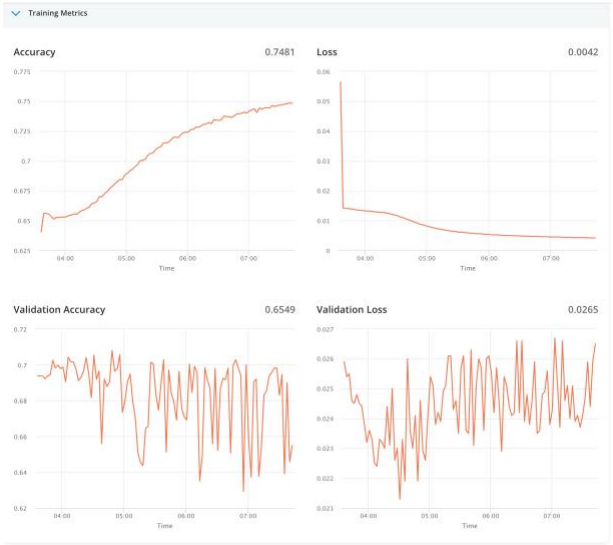
Figure 5: Test results from training on food dataset

E. Testing on Entire TinyImageNet-200

Testing on the entire TinyImageNet-200 dataset took 4 hours and 18 minutes. We trained our model on FloydHub's cloud computing services on a Tesla K80. With 100,000 images from 200 different classes we set epochs at 100 and batch size at 1,000. We initially set our batchsize to 5,000 but system we were training our model ran out of memory, so because of memory constraints we experimented with smaller batch size values until we found that 1,000 batch

size was enough to stop throwing the "not enough memory" errors.

Figure 6: Training metrics from training on entire TinyImageNet-200 dataset



After training our model's accuracy was at a .7481 with a loss of .0042. Validation accuracy was .6549 and .0265. This shows that our model's accuracy could've gone higher if we had let our model train longer with a higher number of epochs. But since even 100 epochs took 4 plus hours, we didn't have enough credits on Floydhub to run a longer training session. Our validation accuracy had barely changed throughout the training session which is another indication that we should've increased the number of epochs, since 100,000 images of various classes is a lot of data with a huge variety of features in each image.

After training our model on such a huge dataset our test results were very impressive and most of the pictures were almost accurately colorized while the same problem of

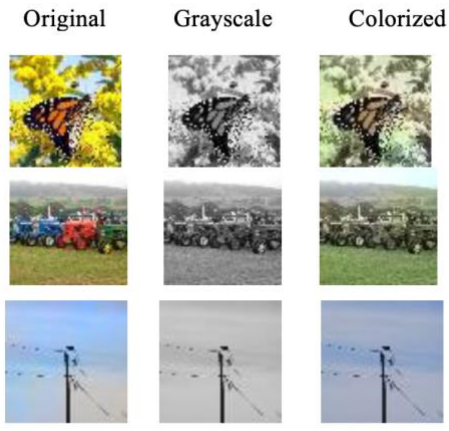


Figure 7: Test results after training our model on entire TinyImageNet-200 dataset

failing to colorize images with multiple colors persisted.

Our model was able to perform well on different images from different classes and that was our final goal. Our colorized images just fall short of the original images and some lack color where there are multiple colors present..

We tested our model on sole grayscale images without having an original colorized image. The colorized images that our model generated seemed were very believable.

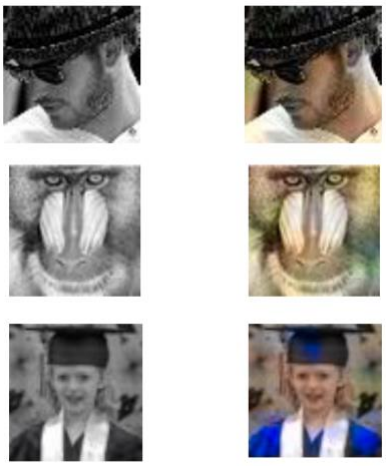


Figure 8: Test results from testing on grayscale images with no original images

F. Testing on Portrait Images

The testing image dataset produced varied results, where some images captured edge detection extremely well and others were blob effects. It was apparent that the model was not able to reproduce the exact colors for objects that could be multi - colored such as shirts and other objects. It did perform well on faces and was able to reproduce the appropriate face complexion. The CNNs, seemed to be able to detect faces well and combining that knowledge with the luminosity layer allowed for the proper coloring of faces. Whereas, other objects were not colored to the fullest capacity - resulting in discoloring and fading.

Below are some sample images from the resulting test dataset.

VI. Conclusion

After all the training and testing we performed, we came to realize that CNN's are incredible powerful and promising for colorizing black and white photos. Since CNN's are able to extract features from images using convolutional layers, colorizing black and white images works really well.

We found that CNN's are able to perform really well on colorizing images when the dataset is small and related.

Training on a larger dataset needed to have more epochs so the CNN could learn more about the vast differences from the images in TinyImagesNet. Though we trained for 100 epochs, it would've been better if we were able to train on 1000 epochs with a different batch size.

Choosing an appropriate activation function was an integral part of the CNN. Choosing ReLu as our activation function for all the convolutional layers except the final layer proved to be very efficient in producing the results we were aiming for. Using the tanh activation function in our final layer allowed for blue and green color to show up in our results. We spent a lot of time digging why our model was failing when we used ReLu activation functions throughout our model.

Our loss function "mse" also gave us a proper insight into how our model was performing during training and allowed for our model to converge properly.

We initially found that we were overfitting our model during training since during our first initial run we used dropout to help the model prevent from overfitting. But during our later runs with larger datasets and more layers, using dropout didn't really improve our results.

Tuning the hyperparameters was an essential part in training our CNN's and with the amount of time it takes to train CNN's on large datasets, we were limited to the amount of tuning we could perform.

Overall our CNN performed really well, and the results were as good as we had hoped for.

Future work could include training our CNN with more layers, a larger dataset, higher number of epochs and working with GAN's or Generative Adversarial Networks.

REFERENCES

[1] Zhang, R., Isola, P., and Efros, A. Colorful image colorization. In ECCV, 2016

[2] Kr¨ahenb¨uhl, P., Doersch, C., Donahue, J., Darrell, T.: Data-dependent initializations of convolutional neural networks. International Conference on Learning Representations (2016)

[3] E. Shelhamer, J. Long, and T. Darrell. Fully convolutional networks for semantic segmentation. TPAMI, 2017.

[4] Mairal, J., Koniusz, P., Harchaoui, Z., and Schmid, C. Convolutional kernel networks. In NIPS, 2014.

[5] Simonyan, K., Zisserman, A.: Very deep convolutional networks for large-scale image recognition. arXiv preprint (2014)

[6] Ian Goodfellow. Nips 2016 tutorial: Generative adversarial networks. arXiv preprint arXiv:1701.00160, 2016