Image Colorization with CNN and comparison of network models

Andree Hultgren Koray M. Kaya Tobias Ekman

Abstract

Self supervised colorization of grayscale images using convolutional neural networks have become a very popular approach due to its good results in the field recent years. We trained and compared CNN models to colorize images to explore best practice within a specific narrow area, namely cats. The baseline model is a trimmed pretrained VGG-16 [reference here] as the encoder and a generative CNN as the decoder. We measure and present the models loss and accuracy, as well as some visual results from the baseline network. We do this using the same training and validation dataset for comparison.

1 Introduction

Automatic image colorization is a problem of generating new data. A common way of colorizing an image is to use the three dimensional LAB color space, where L is the lightness of a picture and the other two channels are dual color combination channels. Making predictions on these two channels given the L channel allows us to perform automatic colorization.

Deep learning models used when facing this problem range in complexity. An example of a complex solution to this problem is provided by (1) who chose to treat this as a classification problem. (1) worked under the assumption that objects that are similar will most likely have a similar color. This gave great results on their evaluation through using a "colorization Turing test", where they successfully fooled 32% of the participants. On the other hand, we attempted to solve this problem through a CNN model, VGG-16, with different implementations. **The implementations had varying results which we will be comparing in this study to find an optimal implementation of VGG-16 for colorization.**

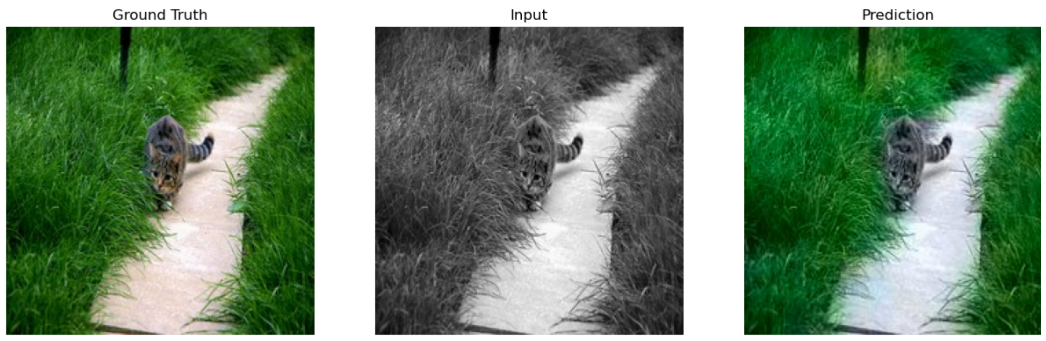

Figure 1: The best results from our model

2 Related Work

2.1 VGG-16

The famous VGG-16 model (2) is an effective method for image classification. While great contemporary achievements have been made in increasing the accuracy of CNN models they have also increased the use of computational resources. Zhang et al. (2) focuses on accelerating test-time performance of CNNs. Their model succeeds in achieving a 4x acceleration with minimal increase in error.

2.2 Automatic Colorization

Using a CNN network with class rebalancing, Zhang et al. (1) achieves great results. Putting notice into the fact that nature is dominated by desaturated colors, which in this context means low AB values, they try to fight this imbalance with through reweighing the loss of each pixel at training based on the pixel color rarity. This practically translates to a form of resampling the training data. In comparison to our predictions, we see that Zhang et al.'s class rebalancing effectively saturizes the pictures, giving more realistic colors, while our pictures can look a bit dull.

3 Method

Restoring color to an image is close to removing noise from an image. Thus, a network looking like an autoencoder. We would not want to do this with normal dense layers. This process should be done with a convolutional autoencoder.

3.1 Network architecture

Learning the entire process would take a long time. There are plenty of image recognition models available. We chose to pick VGG-16 and cut it off before it flattens an image into a dense network. Using this trimmed model, we will perform transfer learning by appending a generative model at the end of the trimmed model. We also clamp the weights of the trimmed VGG-16 in order to not change the model when training our model.

3.2 Data

The dataset chosen for the network at hand is the CAT dataset (3). VGG-16 only takes inputs of size $(224, 224, 3)$, so all images were reshaped to that particular size.

In order to feed the images into the network, a custom pipeline/data generator had to be constructed. This enables the solution to take in essentially an infinite amount of data without running out of RAM. This solution enables the model to have millions of images if so desired. It also enables us to alter the dimensions or color space of the input and output.

3.3 Color space

There are two ways to predict a colored image, either we can predict the entire picture, or the colors. It is more convenient to map a gray image to an RGB. Simply map one image to another. However, the clarity of the image suffers. The network has more data to generate, and thus yield worse results.

An alternative method is to convert the images to a LAB color space. The first color channel L indicates the brightness intensity of the image with a range $[0, 100]$. The A and B includes the other color channels and indicate the color of the image. These layers has the range $[-128, 127]$.

We want the network to take a $(224, 224, 3)$ gray image which is the L channel copied to all three channels. The desired output is the AB color channels of size $(224, 224, 2)$. This allows us to preserve the image clarity and only predict the colors.

Once the AB color channels have been predicted, we convert the LAB channels to one RGB image.

Table 1: Trimmed pre-trained VGG-16 (transfer learning)

Name	Type	Dimensions
Layer 1	Input	$224, 224, 3$
Layer 2	Conv2D	$224, 224, 64$
Layer 3	Conv2D	$224, 224, 64$
Layer 4	MaxPooling2D	$112, 112, 64$
Layer 5	Conv2D	$112, 112, 128$
Layer 6	Conv2D	$112, 112, 128$
Layer 7	MaxPooling2D	$56, 56, 128$
Layer 8	Conv2D	$56, 56, 256$
Layer 9	Conv2D	$56, 56, 256$
Layer 10	MaxPooling2D	$28, 28, 256$
Layer 11	Conv2D	$28, 28, 512$
Layer 12	Conv2D	$28, 28, 512$
Layer 13	Conv2D	$28, 28, 512$
Layer 14	MaxPooling2D	$14, 14, 512$
Layer 15	Conv2D	$14, 14, 512$
Layer 16	Conv2D	$14, 14, 512$
Layer 17	Conv2D	$14, 14, 512$
Layer 18	MaxPooling2D	$7, 7, 512$

Table 2: Decoder that is to be learned

Name	Type	Dimensions
Layer 19	Conv2D	$7, 7, 256$
Layer 20	Conv2D	$7, 7, 128$
Layer 21	UpSampling2D	$14, 14, 128$
Layer 22	Conv2D	$14, 14, 64$
Layer 23	UpSampling2D	$28, 28, 64$
Layer 24	Conv2D	$28, 28, 32$
Layer 25	UpSampling2D	$56, 56, 32$
Layer 26	Conv2D	$56, 56, 16$
Layer 27	UpSampling2D	$112, 112, 16$
Layer 28	Conv2D	$112, 112, 2$
Layer 27	UpSampling2D	$224, 224, 2$

3.4 Activation functions

The decoder consists of relu activation functions all throughout the network, except for the last layer, where a tanh activation function exists. The network is mapping towards the AB color space with the range $[-128, 127]$, but which is re-scaled with a factor of $\frac{1}{128}$. This gives us an output space of approximately $[-1, 1]$, which is the range of the tanh activation function.

We cannot alter the activation functions in the VGG-16 model since we have set it to a static function.

3.5 Optimizer and loss function

Most networks, especially this type of network, is heavily reliant on a good loss function. We lack the mathematical ingenuity to develop our own loss function. We decided to pick a simple but effective loss function Mean Squared Error. The learning method is a variant of Stochastic Gradient Descent called Adaptive Moment Estimation (Adam). This is generally a stable optimizer for Convolutional Neural Networks.

3.6 Regularization

To fight the overfitting we faced we resorted to using methods for regularization. Initially we used batch normalization which is supposed to increase learning speed and generalization in the network. (4) mentions in section 3.4 that batch normalization can either reduce or remove overfitting, compared to dropout which simply reduces it. Batch normalization normalizes the output of a previous activation layer through shifting and scaling with the use of the batch mean and the batch standard deviation.

Parallel to our batch normalization network we also trained a dropout regularized network. We decided to give the dropout method a try since it usually is the "go-to" regularization method in CNN networks. The method omits certain activations in the network with a given probability to decrease the complexity of the network and therefore also the overfitting.

4 Results

4.1 Baseline

Our first attempts at automatic colorization proved to overfit to the training data very quickly. The model would overfit to the training data after a few epochs. While the model gave great results on the training data, it did not do so on the validation or test set. Example of this is below.

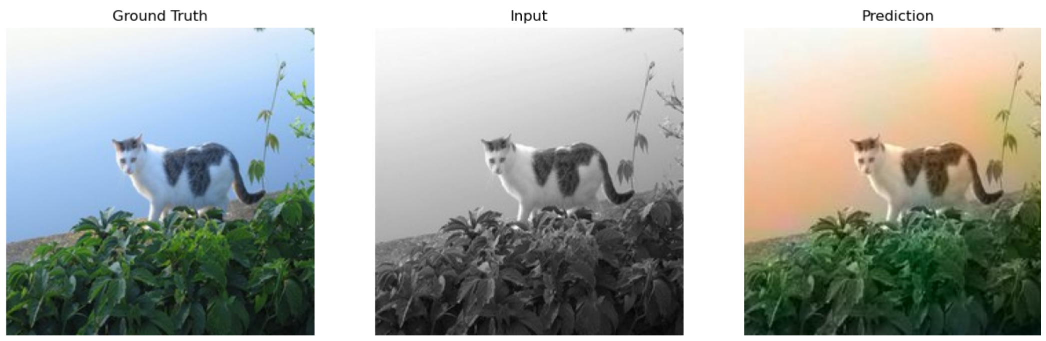

Figure 2: Example of training prediction.

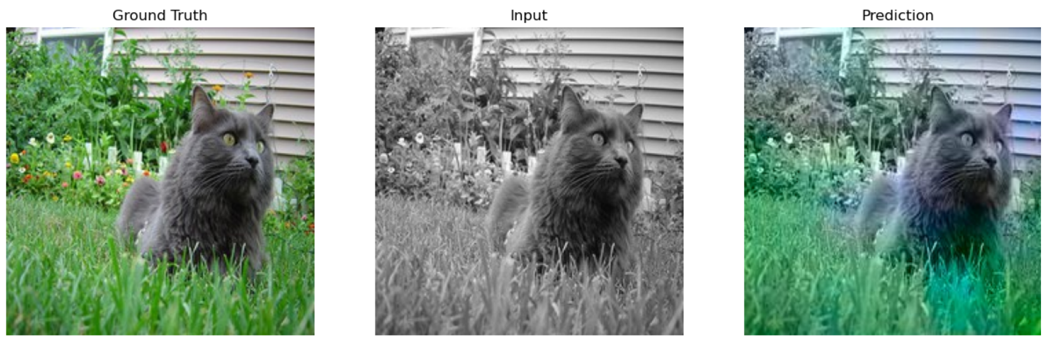

Figure 3: Example of test prediction

Overfitting can be caused by a number of reasons, our assumptions are mainly three: low amount of data, complex model and lack of regularization. Simply put, if the model lacks sufficient reference points the pattern it learns will be very limited. This may be our main concern since our dataset consisted of merely 10,000 images in comparison to Zhang et al.'s (1) model which is based on approximately 1.3M training images. A complex model with too many moving parts can in a similar way adapt the model too well to the training data, leading to poor generalizations when it comes to test data. Our CNN model consisted of approximately 1.5M parameters which is extremely large when compared to the size of our small dataset. And finally, we did not implement a regularization method to our baseline model but we will be exploring this addition in the following results.

On a side note, our model did a great job recognizing greenery such as grass and trees. This might be due to the simple geometry of grass and plants.

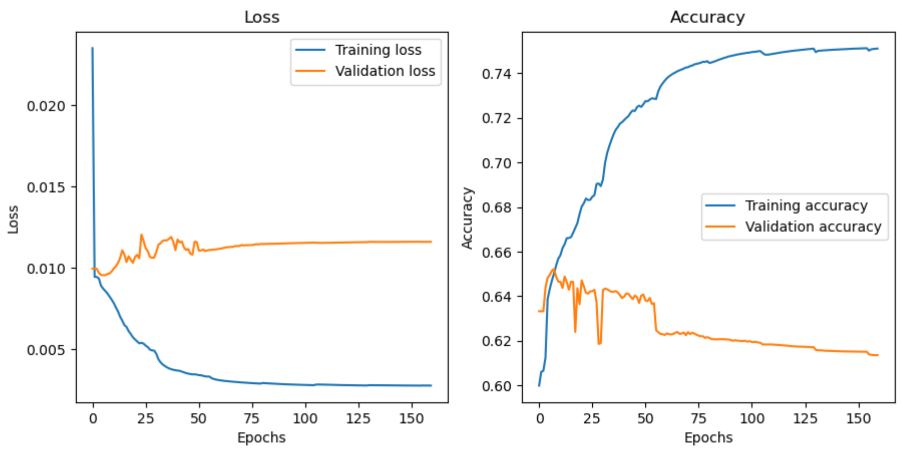

Figure 4: Loss and accuracy plot of the baseline model

4.2 Batch Norm

In an attempt to improve the speed and performance of our model we decided to incorporate batch normalization. Prior to each convolutional layer the batch normalization would normalize the data, giving the model an easier time recognizing patterns using lower weights and thus making better generalizations. This worked for an epoch or two, after which the network started drastically overfitting leaving us to the conclusion that the batch normalization only made it worse, see figure 5.

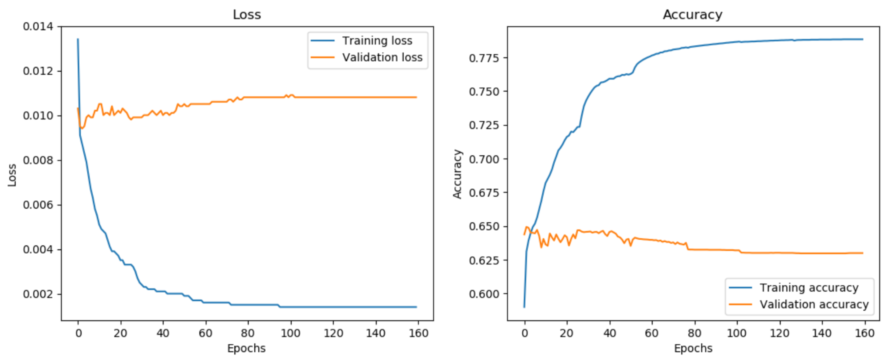

Figure 5: Loss and accuracy plot with batch normalization

4.3 Dropout

We can see on figure 6 that while the accuracy of the model does not substantially increase, the dropout method is regulating the overfitting compared to the loss plot on figure 5.

In contrast to batch normalization, the dropout method seemed to slow down the learning. Although the dropout method omits signals in the network, it also adds additional calculations to be made which slows does the learning rate.

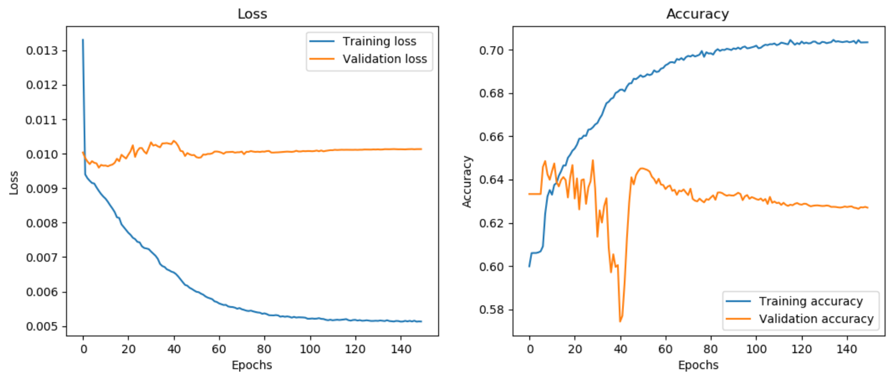

Figure 6: Loss and accuracy plot with dropout regularization

5 Conclusion

5.1 Dataset

We faced a severe case of overfitting. Our main assumptions are that the overfitting is caused by a small dataset and a high dimensional model. Fixing this problem would mean acquiring more data to satisfy the model or decreasing the complexity in the model - the optimal solution being acquiring more data since this would improve the results rather than simply decreasing the complexity of the model. Our dataset of 10,000 images is incredibly small compared to related studies such as Zhang et al. (1) who used 1.3M images for training. Although we aimed to keep the dataset smaller to reduce training time it may have been worth it to use a bigger dataset for the potential results. This is subject for further research: doing the same experiment with a dataset of minimum 500,000 images and up to 5M images. Since the complexity of the network is already very high we would likely not need to make any adjustments.

Given the small dataset, we are happy with our results. While using a dataset 130 times smaller than Zhang et al. (1), our results are still comparable. Zhang et al. (1) trained their model to color any sorts of photos while our model specialized in cats, this certainly is a factor in the results. Further interesting research can be to compare the need for data in specialized networks and general networks for similar results.

5.2 Network

While in theory batch normalization is supposed to increase our accuracy, we found the opposite. Batch normalization increased the speed of our training but also unexpectedly increased the overfitting very quickly.

On the other hand, dropout seemed to be an effective way of regularizing the overfitting in our model while not improving the accuracy much. The regularization worked but since it did not yield accuracy gains we were not satisfied with it.

5.3 Automatic colorization

Objects with ambiguous colors can be hard to learn for a CNN. Zhang et al. (1) acknowledges this problem and circumvents it through defining the problem as a "plausible colorization problem" rather than simply a "colorization problem". While we find that vegetation is generally correctly colored - green - in our model, coloring cats are much more difficult. We find that the model generally resorts to coloring the cats either black and white or a mean "cat-color".

Another coloring problem is the desaturization of the images. Even if some images are colored in the correct positions, we find that the colors are generally not satisfying. This might be due to a overwhelming amount of green color in the pictures compared to other colors such as "cat-colors". We could implement training data resampling to rebalance the classes and get more cat-color reference points based on what Zhang et al. (1) has done and succeeded with.

6 Note

We were not able to access any cloud computational resources during the span of this project. This setback limited our ability to scale up the data of the model, and to do many different experiments.

References

[1] R. Zhang, P. Isola, and A. A. Efros, “Colorful image colorization.” Springer, 2016, pp. 649–666.

[2] X. Zhang, J. Zou, K. He, and J. Sun, “Accelerating very deep convolutional networks for classification and detection,” *IEEE transactions on pattern analysis and machine intelligence*, vol. 38, no. 10, pp. 1943–1955, 2015.

[3] https://archive.org/details/CAT_DATASET.

[4] S. Ioffe and C. Szegedy, “Batch normalization: Accelerating deep network training by reducing internal covariate shift,” *arXiv preprint arXiv:1502.03167*, 2015.