Colorizing Grey Images Through a CNN

Christoffer Lundell Johansson **Felix Broberg** **Rodrigo Retuerto**

Abstract

The problem of converting black and white images to colored images has many real-life applications. Zhang et al. [2016] presents a deep neural network architecture that attempts to solve this problem. It is a VGG-style 8 block architecture. In this project, we have implemented said network in the machine learning framework Keras. Once the implementation was complete and tested we changed the various parameters and observed the results. Our final trained model is able to produce realistic looking images but is not as stable as the model presented in the paper.

1 Introduction

There is no lack of black and white media created over the years. Everything from old movies and pictures to high-end security footage has the possibility to be lacking in color. Artists both with brush and digital tools have spent many hours over the years trying to adapt these gray scale pictures into colored ones. Deep Learning can in the future become an competitive alternative to this.

By training a convolutional neural network (CNN) on colored images and minimizing the loss between the output and the original image, a deep-learning solution can be created. This is done by taking grey-scale images as input and producing smaller sized color predictions. The output is then upscaled and combined with the original image to present the final coloured image from the process. In depth information of training, the network and discussions will follow in later sections.

We produce two models that were trained on different sized inputs[1]. The results are varying, but we achieve realistic results for many images. The models are however far from the level that can be seen in the authors web-application[2]. We assume that a major factor in the difference of results is the amount of input data and the amount of time they have spent on training the models.

2 Related Work

The paper "Colorization using Optimization", by Levin et al. [2004], serves as a model example regarding image colorization. Previous work required both extensive human interaction utilizing expensive segmentation techniques. Their method is instead based on the premise that neighbouring pixels in space-time that have similar luminance (lightness) should have similar colors. This allows for efficient solutions to the problem to be computed with minimal human input, showing that computer colorization is a feasible area.

The paper "Colorful Image Colorization", by Zhang et al. [2016], improves upon this feasibility by presenting a CNN that has been trained on 1.3 million images from ImageNet. It improves upon previous work by producing vibrant and plausible results that require no human input. As color images can be used for both the ground truth as well as the training input, by extracting the grey-scale image, training data is essentially free. By utilizing the probabilities of each color, the authors were able to improve upon previous work and output images that are both plausible and vibrant. When

[1]Source code at https://github.com/RoddeTheR/deep-learning-image-colorization

[2]https://demos.algorithmia.com/colorize-photos/

comparing their output images from the full model with the ground truth, humans are shown to believe that their output is the original image in 32% of the trials.

The paper "Real-Time User-Guided Image Colorization with Learned Deep Priors", by Zhang et al. [2017], builds upon the CNN solution of the "Colorful Image Colorization" paper (Zhang et al. [2016]). Like the paper "Colorization using Optimization" (Levin et al. [2004]), it utilizes a small amount of human interaction to guide the search towards a good solution and is fast enough for real-time use. As opposed to the solution in "Colorization using Optimization" where human input is a necessity, the color suggestions in this paper are used as a tool for better end results. The CNN is also able to present colorization suggestions to the user to choose from.

3 Data

In this project, the required data for training are ordinary coloured images. The images were obtained through a Python script that takes a file containing URLs and downloads the images. The URLs were obtained from the Fall 2011 ImageNet database (Deng et al. [2009]). Due to time limitations, we opted to use the first 71k images from the dataset instead of 1.3 million images like Zhang et al. [2016] did. Our selection of images might be a point of failure due to how the image URLs are arranged. We may get many images from a few classes instead of a few images from many classes. We also used the entire Tiny ImageNet dataset to avoid this problem when training on small resolution images.

The LAB color space (Luo [2014]) is a three layer color space, L A B, where L decides the amount of lightness, A the green-red intensity and B the blue-yellow intensity. In our pre-processing step, we do two major operations. The first is resizing the images into the appropriate size. The second step converts the RGB channels into LAB channels, where the L channel will be used as input and AB channels will be used as the output in the network. The aforementioned conversion was done using the OpenCV (Bradski [2000]) library. A similar post-processing pipeline for the network output was also implemented.

4 Methods

Our main approach to solving the problem is using the procedure outlined in "Colorful Image Colorization" by Zhang et al. [2016]. The paper presents an 8 block convolutional neural network as shown in figure 1. The general structure of the network, a VGG-style network without pooling or fully connected layers, seems to be agreed upon but the specific details less so. Even the table specifying the network architecture, the figure specifying the architecture and the provided source code for the report differ in aspects such as kernel sizes and the number of channels. For this project, we chose to follow the table from the report, figure 1, as that is the improved version 2 of the network. Additionally, we used a kernel size of 3x3 for all layers except from the final one where a 1x1 kernel is used. Our implementation of the network was done using the Keras (Chollet et al. [2015]) framework. A limitation due to this choice is our initialization. The paper suggests using K-mean initialization that was presented in "Data-dependent initializations of convolutional neural networks" by Krähenbühl et al. [2015] for the network. Due to the algorithmic complexity of the initializer and unavailability in Keras, we decided to try other initializers that we encountered during the course. We will explore initialization in more detail in chapter 5.

As a pre-processing step, the images need to be converted from the RGB to the LAB colour space. The LAB colour space is preferred due to the fact that lightness is an independent channel in the LAB colour space, unlike in the RGB one. We also save the images in HDF5 (The HDF Group [2000-2010]) files as saving them all in memory was unfeasible due to hardware limitations and reading them from disk wasn't fast enough.

In this paper, the A and B channels are treated as a single variable. The 2-D space is then quantised into $q = 313$ bins as shown in figure 2. This transforms the network problem into a multiclass classification problem.

The input of the network is an array of the lightness value of each pixel of a given input image. The output of the network is an array that contains the probability for each AB bin for every pixel. This output will have 1/4 of the original width w and height h, which is taken care of in the post-processing

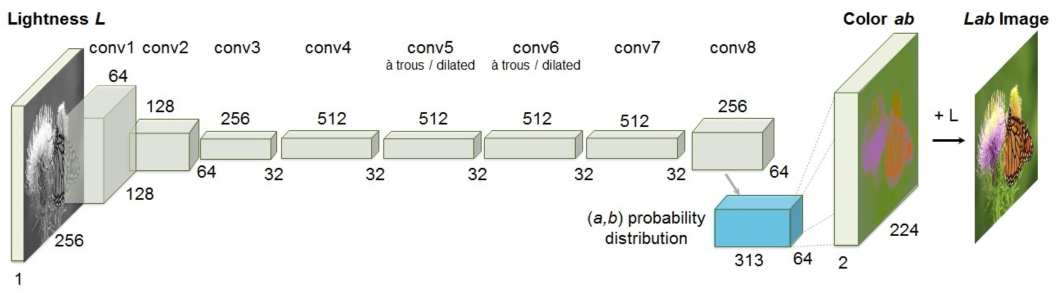


Figure 1: The original architecture. Version 2 presents slight differences where conv7 has 256 channels of output and conv8 has 128 channels of output. The architecture is explained in detail in table 1 (Zhang et al. [2016])

Layer	**X**	**C**	**S**	**D**	**BN**	**K**
data	256	3	-	-	-	3
conv1_1	256	64	1	1	-	3
conv1_2	128	64	2	1	✓	3
conv2_1	128	128	1	1	-	3
conv2_2	64	128	2	1	✓	3
conv3_1	64	256	1	1	-	3
conv3_2	64	256	1	1	-	3
conv3_3	32	256	2	1	✓	3
conv4_1	32	512	1	1	-	3
conv4_2	32	512	1	1	-	3
conv4_3	32	512	1	1	✓	3
conv5_1	32	512	1	2	-	3
conv5_2	32	512	1	2	-	3
conv5_3	32	512	1	2	✓	3
conv6_1	32	512	1	2	-	3
conv6_2	32	512	1	2	-	3
conv6_3	32	512	1	2	✓	3
conv7_1	32	256	1	1	-	3
conv7_2	32	256	1	1	-	3
conv7_3	32	256	1	1	✓	3
conv8_1	64	128	0.5	1	-	3
conv8_2	64	128	1	1	-	3
conv8_3	64	128	1	1	✓	3
1x1_layer	64	313	1	1	-	1

Table 1: $\mathbf{X}$: Resolution of output, $\mathbf{C}$: Channels of output, $\mathbf{S}$: Stride, $\mathbf{D}$: Dilation, $\mathbf{BN}$: Indicates if Batch Normalizaton was used, $\mathbf{K}$: Kernel size. (Zhang et al. [2016])

step. We also considered having an output of two values, A and B, but decided against it due to the unnecessary complexity added to the last layer of the network and the fact that the presented loss methodology wouldn't work.

The network requires the ground truth Z for the given input in order to calculate the loss. To calculate the ground truth in the same format as the output of the network, the five nearest neighbours were selected through the ball tree algorithm. The neighbours were weighted based on their distances from the truth using a Gaussian kernel with standard deviation of 5, as described in the paper. To compare with the network output $\hat{Z}$ we use the categorical cross entropy loss $\mathrm{L}_{cl}(*, *)$ (Equation 1) as our loss function. Unlike the paper, our loss function doesn't include the class re-balancing factor that should

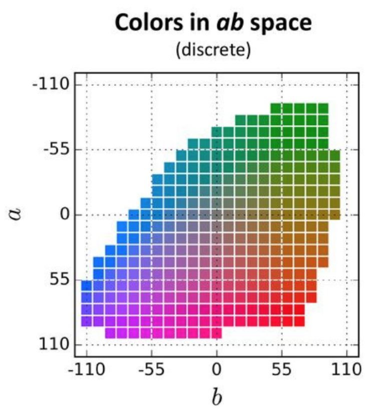


Figure 2: The quantised bins. (Zhang et al. [2016])

make the colors appear more vivid. It can be noted that the class re-balancing looks worse in some cases, such as in two colorized videos where rebalancing is used in one[3] but not the other[4].

$$\mathrm{L}_{cl}(\hat{\mathbf{Z}}, \mathbf{Z}) = -\sum_{h,w}\sum_{q} \mathbf{Z}_{h,w,q} \log(\hat{\mathbf{Z}}_{h,w,q}) \tag{1}$$

The paper solves the size difference of the input and the output image by upsampling the data using spline interpolation. Due to the natural multimodal distribution of colours in nature (green banana, yellow banana), the mean of the output data would lead to dull colours and therefore this method cannot be used. The mode cannot be used either as it leads to spatial inconsistencies. The authors present the concept of the annealed-mean $\mathcal{H}$ (see formula 2) which is a mix of the mean and the mode with a balancing parameter T. In our case T was set to 0.38 as recommended by the paper. The expected value for f_T was based on the color distribution of all images in ImageNet and was obtained from the source code of the paper. It is to be noted that the paper implements this as the last step of the network whilst we implemented it as a post-processing step to be executed afterwards.

$$\mathcal{H}(\mathbf{Z}_{h,w}) = \mathbb{E}[f_T(\mathbf{Z}_{,h,w})], \quad f_T(\mathbf{z}) = \frac{\exp(\log(\mathbf{z})/T)}{\sum_q \exp(\log(\mathbf{z}_q)/T)} \tag{2}$$

5 Experiments

Although we have mostly followed the paper by Zhang et al. [2016] for network architecture, we have tried various hyperparameter settings to try to obtain good results. The authors of the paper had more time and resources than us and initially trained a slightly different model compared to the one we are training. We thus expected that our parameter settings would have to differ slightly. Additionally, regression values were not specified in the paper. As such we have had to experiment with this as well as learning rate, optimizer settings and weight initialization. The following parts will describe our experiments for determining correctness as well as some of the results we achieved.

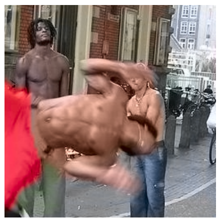
5.1 Parameter evaluation

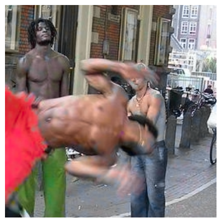
The pre- and post-processing steps are important and must be implemented correctly for our solution to work. We implemented the required parts and tested them on several images by feeding the pre-processing ground truths to the post-processing function and observed the end result. Through these checks, we discovered a problem with HDF5 type conversions that we were able to fix. Given that the results were now equal to the original images, we continued with our implementation of the CNN with the assumption that the implementations were correct.

[3]https://www.youtube.com/watch?v=Hf5fedWk0Ac
[4]https://www.youtube.com/watch?v=HvaOiUTKbl0

Following the implementation of the CNN, we tried to overfit the model on a few images. Figure 3 shows the results after 300 iterations.

(a) Original image.


(b) learning_rate = 1e-3, β_1 = 0.9, β_2 = 0.99, decay = 1e-3.


(c) learning_rate = 1e-4, β_1 = 0.9, β_2 = 0.99, decay = 1e-3.

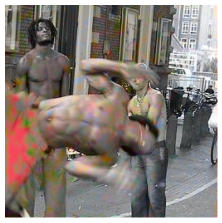
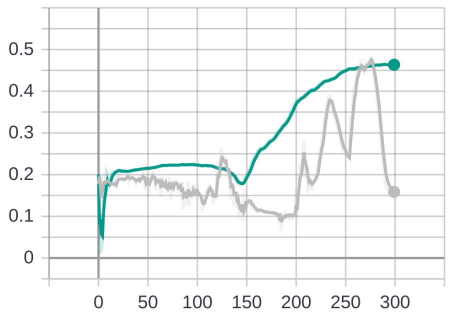

(d) learning_rate = 3e-5, β_1 = 0.9, β_2 = 0.99, decay = 1e-3.

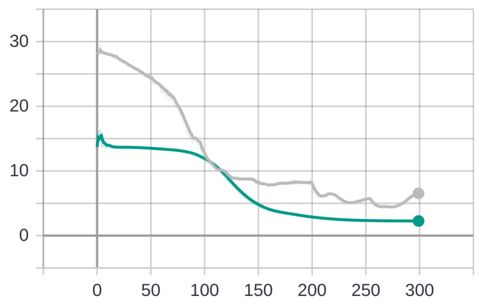
Figure 3: Comparing the results of overfitting the network on a single image with different learning rates. The learning rate is reduced by a factor 0.2 after reaching a loss plateau.

We experimented with different intializations, such as Xavier normal and He normal initializations, and chose to initialize our network with He normalization. This is also theoretically sound as all of our activation layers are ReLU (He et al. [2015]). The paper uses Adam (Kingma and Ba [2014]) as optimizer with learning_rate = 3e-5, β_1 = 0.9, β_2 = 0.99, decay = 1e-3. These parameters produced overfitted results with spatial inconsistencies, see figure 3 (d), that could be due to our choice to not use K-means initialization. Tuning the learning rate to 1e-4 produced better results with our initialization, see figure 3 (c) and is the option we chose.

As the paper by Zhang et al. [2016] did not mention regularization, we performed a coarse-to-fine random search to set the L2-regularization value on a relatively small set of data. After 200 epochs, we found that a regularization constant of $\approx 1e-3$ works well. Since our dataset that we use for training is much larger than what we used in the coarse-to-fine random search, we chose to use the constant $1e-4$ for regularization. To improve the variety of images during training, we augment both training data and the validation data by randomly cropping images and randomizing the batch order.



(a) Training accuracy for 300 epochs.



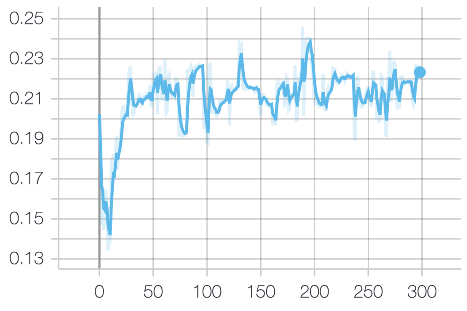
(b) Training loss for 300 epochs.

Figure 4: Comparison of the effects of reducing learning rate when overfitting on 1 image for 300 epochs. The smooth green curve shows the effect of reducing learning rate upon reaching a plateau while the grey erratic curve shows the effects of not reducing the learning rate.

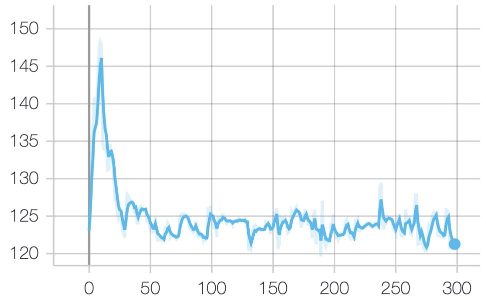
Our model initially struggled to improve on our validation loss and tuning of hyperparameters did little to fix this. Figure 4 shows an erratic behaviour when not reducing the learning rate, as can be seen by the grey curve. This was improved upon by reducing the optimizers learning rate when reaching a plateau, resulting in the green curve seen in figure 4. Once a more stable training was achieved, we trained two different models: one using the Tiny ImageNet dataset with 64x64 pixel images and one with a subset of ImageNet, consisting of 256x256 pixel images. This was done in order to compare the advantages of using larger images for training compared to smaller ones but for a longer time. Figure 5 and figure 7 shows the validation losses for both models while figure 6 figure 8 shows some example outputs.

5.2 Results

Model-64 was trained on the Tiny ImageNet dataset consisting of 200 classes and a total of 100000 training images, 10000 validation images and 10000 test images. All images were of the size 64x64 pixels and were randomly cropped to 56x56 pixels for each epoch. The training was performed for 300 epochs before stopping due to not improving on the validation loss. Validation loss and accuracy are shown in figure 5.



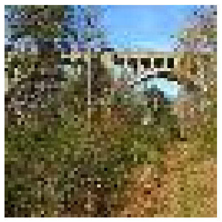
(a) Validation accuracy for 300 epochs.



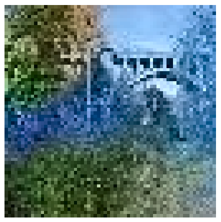
(b) Validation loss for 300 epochs.

Figure 5: Validation accuracy and loss when training on 100000 64x64 images from the Tiny ImageNet dataset. The hyperparameters used were: learning rate = 1e-4, β_1 = 0.9, β_2 = 0.99, decay = 1e-3, regularization constant = 1e-4. The learning rate was decreased by a factor 0.2 after a period of 10 epochs with no improvement in validation loss.

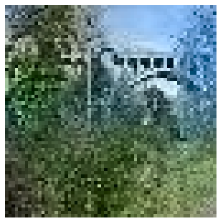
Although the validation loss can be seen to decrease and the accuracy increase, the movement is erratic. The learning rate was decreased several times but no apparent decrease in the erratic movement can be seen. Despite an apparent plateau, figure 6 shows improvement in actual coloring quality.



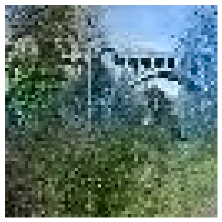
(a) Original image.



(b) Model-64 prediction after 10 epochs.



(c) Model-64 prediction after 100 epochs.

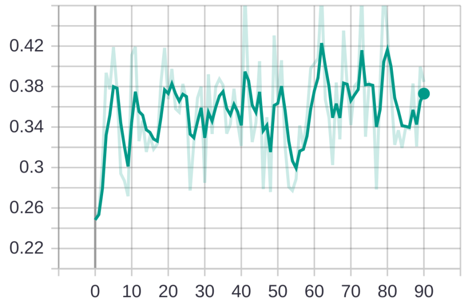


(d) Model-64 prediction after 300 epochs.

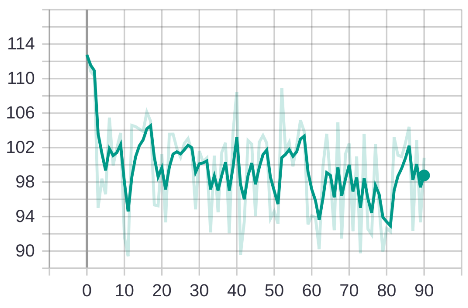
Figure 6: Comparison of the model predictions for Model-64 on a 64x64 image from the Tiny ImageNet test dataset.

Model-256 was trained on a subset of the ImageNet dataset, with 71000 training images, 1000 validation images and 1000 test images. All images were of the size 256x256 pixels and were randomly cropped to 176x176 pixels for each epoch. The training was performed for 90 epochs before stopping due to not improving on the validation loss. Validation loss and accuracy are shown in figure 7.

As with the case of the validation loss and accuracy of the Model-64, the learning displays erratic behaviour in the case of Model-256 as well even though the learning rate was decreased when plateauing. Figure 8 shows the image quality progressing, indicating that the model is learning despite this.



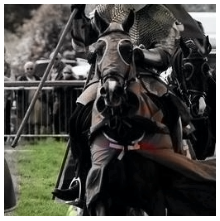
(a) Validation accuracy for 90 epochs.



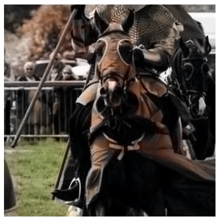
(b) Validation loss for 90 epochs.

Figure 7: Validation accuracy and loss when training on 71000 256x256 images from the ImageNet dataset. The hyperparameters used were: learning rate = 1e-4, β_1 = 0.9, β_2 = 0.99, decay = 1e-3, regularization constant = 1e-4. The learning rate was decreased by a factor 0.2 after a period of 10 epochs with no improvement in validation loss.

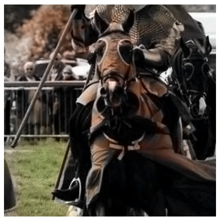
(a) Original image.



(b) Model-256 prediction after 10 epochs.



(c) Model-256 prediction after 50 epochs.

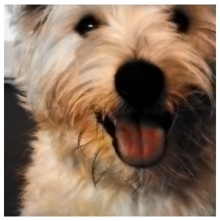


(d) Model-256 prediction after 90 epochs.

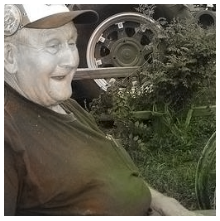
Figure 8: Comparison of the model predictions for Model-256 on a 256x256 image from the ImageNet dataset that was used for testing. It's important to note that the model is trained to produce plausible results, not the actual ground truth, and as such the blue colors in the original image is unlikely to appear in a our version of the image.

6 Conclusion

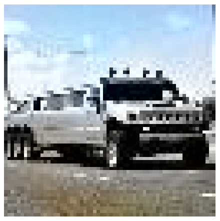
The two models do well at colorizing different images. Model-64 seem to do well with landscape pictures, but struggle with images on subjects such as animals. Model-256, on the other hand, do well when colorizing animals but struggle on images with many different items in them. In such cases, it often times produce dull images. Overall, Model-256 seems to produce colorizations that are both more vibrant and plausible than Model-64, despite a shorter training time. This could indicate that the training data for Model-256 was better versed for this task or that using larger images for training is to be preferred in the case of training this CNN for image colorization. Some examples of both successful and unsuccessful colorizations can be seen in figure 9.



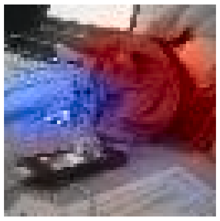
(a) Model-256: Successful colorization.



(b) Model-256: Failed colorization.



(c) Model-64: Successful colorization.



(d) Model-64: Failed colorization.

Figure 9: Example results for both models on test data.

Despite getting the network to produce plausible colorizations, the validation losses seen in figure 5 and figure 7 indicate that further tuning of hyperparameters is necessary. Future work could focus more on more careful experimentation with these in order to obtain better results. Extensions of this implementation would include both class rebalancing to produce more vibrant colors and k-means initialization in order to improve training.

It has become apparent during the project that the values of the various hyperparameters plays a bigger role than we initially thought. This became obvious during our attempts to overfit the, as seen in figure 3, and it's likely that we have not reached optimal settings. The processing of images was a limiting factor for the efficiency of training the models. Our initial attempt of opening and reading each image individually proved too limiting due to low hard drive read speed. A second attempt of storing all processed training and validation data in a HDF5 file again proved slow as the amount of data to read each batch became unreasonable. We finally settled on storing all unprocessed images in a single HDF5 file and allowing several Keras workers to generate data from a thread-safe generator.

Even though our models successfully produce plausible results in many cases, the consistency would have to be improved in order to be useful. Zhang et al. [2016] struggle with this as well, as is described in their paper. However, their final model is still vastly superior to our final result.

References

G. Bradski. The OpenCV Library. *Dr. Dobb's Journal of Software Tools*, 2000.

François Chollet et al. Keras. `https://keras.io`, 2015.

J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei. ImageNet: A Large-Scale Hierarchical Image Database. In *CVPR09*, 2009.

Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Delving deep into rectifiers: Surpassing human-level performance on imagenet classification. *CoRR*, abs/1502.01852, 2015. URL `http://arxiv.org/abs/1502.01852`.

Diederik Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *International Conference on Learning Representations*, 12 2014.

Philipp Krähenbühl, Carl Doersch, Jeff Donahue, and Trevor Darrell. Data-dependent initializations of convolutional neural networks. *arXiv preprint arXiv:1511.06856*, 2015.

Anat Levin, Dani Lischinski, and Yair Weiss. Colorization using optimization. In *ACM SIGGRAPH 2004 Papers*, SIGGRAPH '04, pages 689–694, New York, NY, USA, 2004. ACM. doi: 10.1145/1186562.1015780. URL `http://doi.acm.org/10.1145/1186562.1015780`.

Ming Ronnier Luo. *CIELAB*, pages 1–7. Springer Berlin Heidelberg, Berlin, Heidelberg, 2014. ISBN 978-3-642-27851-8. doi: 10.1007/978-3-642-27851-8_11-1. URL `https://doi.org/10.1007/978-3-642-27851-8_11-1`.

The HDF Group. Hierarchical data format version 5, 2000-2010. URL `http://www.hdfgroup.org/HDF5`.

Richard Zhang, Phillip Isola, and Alexei A. Efros. Colorful image colorization. *CoRR*, abs/1603.08511, 2016. URL `http://arxiv.org/abs/1603.08511`.

Richard Zhang, Jun-Yan Zhu, Phillip Isola, Xinyang Geng, Angela S. Lin, Tianhe Yu, and Alexei A. Efros. Real-time user-guided image colorization with learned deep priors. *CoRR*, abs/1705.02999, 2017. URL `http://arxiv.org/abs/1705.02999`.

Appendices

A Felix Broberg

Overall, I've learned what it's like to implement a deep neural network from the ground up using frameworks that are used in the industry. I've gotten some hands on experience with a bit of everything: debugging incorrect layers in the network itself, data manipulation, hyperparameter tuning etc. The three main deep learning skills/knowledge I have aquired are/is:

Keras
Prior to this project, I hadn't implemented anything regarding neural nets. Using Keras was the most approchable alternative, and as we did not need any custom layers (apart from the post-processing step that we performed outside of the CNN), implementing the network in Keras was straightforward. As the network was descibed in detail in the paper, we could simply implement one layer after another and focus on hyperparameter tuning and similar things. As the Keras documentation was well written and up to date, small differences between the authors' approach and ours was easy to see most of the time. An example is that the authors use caffe where you need to explicitly state the padding size, while we could set padding='same' (in this case).

Parameter tuning
As stated in the report, I think I underestimated the importance of proper initialization, learning rate, regularization etc. I had understood it to some extent when doing the labs, but seeing the actual difference between, for instance, setting the learning rate in the Adam optimizer to 1e-4 and 3e-5 and getting vastly different results (see figure 4) was an eye opener. Because of this, I learned to spend more time on this and to be more careful to get good values from the start. As our final loss plots are still quite erratic, I believe we could have spent some more time on this to get it right. After all, it's worth spending a few more hours tuning these parameters to avoid unnecessary training which takes a lot more time given a problem such as this. The importance of this is probably one of the biggest takeaways for me.

Importance of utilizing traditional methods
Although deep-learning is a great way of approaching difficult and underconstrained problems such as this, I have gotten a better appreciation for the utilization of more traditional approaches to achieve even greater results. Some of the advances the main paper did can be attributed to their novel approach of quantising the 2-d output space into 313 bins and utilizing the probabilities associated with the possible colors. This not only uses deep-learning through a CNN, but is also a combination of nearest neighbour algorithms etc. Altough one can argue that the deep-learning part of it is the most important one, it is the combination that yielded the best results. My takeaway from this is that it's important to remember to utilize the "simpler" things as well.

B Rodrigo Retuerto

B.1 Keras

Learned about the widely used framework Keras. This knowledge was gathered through the implementation of the model and throughout the experimentation of parameters. Learned about many aspects such as model creation, training and saving.

B.2 Data processing

Learned how to handle big amounts of data through the HDF5 format and how to convert image to fit our purpose by using the CIELAB color space. Also how to data can be pre and post processed to match the required dimensions of the network.

B.3 Architecture

Learned on how a deep neural network can be structured for it to produce adequate results in a real application. First hand experience with how the different layers of the network interact.

Evidence for the first two mentioned skills can be found in the source code.

C Christoffer Lundell Johansson

When I returned to KTH after a long study break there was a lot of things I could focus on and push towards as goals towards finishing a masters. Deep Learning sounded interesting and fun. It came both recommended and not recommended by my peers that work at the school, so I wanted to make my own opinion. That being said, I have had issue with sickness and having to go to hospital a number of times, family issues and things of that nature. This has limited my exposure to some of the finer details of the project. But I found a lot of interest in K-means, Convolutional Layers and Tensorflow.

I always looked at Neural networks and found them interesting. So looking more into conventional layers, how they fit together with fully connected layers, filters and the like to form a network fascinated me. The fact that the paper only used convolutional layers was a learning experience. Most pictures and explanations talked about pooling, how it worked in tandem with the fully connected layers. So seeing the use of padding and extensive simplification to try to mine the most information possible from the pictures where an learning experience. It ties into things I worked with earlier. To split problems up and make them simpler.

I had heard about K-Means from earlier courses in Machine Learning. But I will admit that I had not payed a lot of attention to it as it was not an integral part of what I was working with at the time with projects and homeworks. But it came up again here. It is a great tool to use when you do not know exactly what traits and classes are present, but we find characteristics that tie everything together. In this case using the mean between K nearest neighbors to find what bucket of color a pixel fits into. It was interesting to revisit things I have not worked with since I did hyperplanes lab in earlier course. That said, I put a lot of hard study time into reading into a thing we did not focus on. Going for He normal instead.

Lastly, Tensorflow. A name I only heard in passing when discussing things like Bitcoin or from friends lamenting it and why people love it so much. I had only glancing knowledge before the course and I am now in love with it! I find the fact that you make a directed graph novel. But the efficiency of computation you get when you spread the workload out on different CPUs and the like is not to be looked down on. That being said, one should probably look into how much time and computation is done. Then put nodes with low workloads on same CPU and put the heavy hitters on their own! This is far from brief. But I hope it helps and count!.