CNN Assisted Colorization of Gray-scale Images

Ross Daly
Stanford University
ross.daly@stanford.edu

Tian Zhao
Stanford University
tianzhao@stanford.edu

Abstract

We propose a colorization tool which utilizes both minimal user input and traditional convolutional neural networks to color gray-scale images. Deep learning can only go so far in solving colorization. Our major contribution is to show how we can condense color information into a low dimensional color palette and be able to infer this color palette from minimal user input. This allows an interactive system in which a user can provide insightful color hints to gray-scale images in order to generate better colored image results.

1. Introduction

There has been a growing interest in recoloring gray-scale images. A preliminary search reveals many interested parties working on completing this task. The traditional approach for realistically colorizing gray-scale images is for an artist to directly color it using digital photo editing tools. Our approach is to semi-automate the process utilizing Convolutional Neural Networks. This problem is both technically challenging and the results are visually enticing. There exists a large amount of black and white images and videos in which there is personal interest in realistic and interactive coloring.

1.1. Motivation

The first cameras were produced in the early 1800s and could only produce images in black and white. Black and white photograph persisted well into the 1900s and is surprisingly making a comeback even in the modern day. Colored photography and film did not really exist until the early 1900s and did not become economically feasible for people until even later. Because of this, there are a large amount of black and white images and videos that people would like to see colored.

This problem has sparked dedicated subreddits (/r/colorization, /r/ColorizedHistory, /r/ColorizationRequests) for people who have interest in coloring black and white images. The people who use these subreddits typically put in a lot of time and effort using expensive high-end photo editing platforms in order to complete the colorization task. In addition to these subreddits, numerous online tools and guides exist for the purpose of assisting in the colorization of black and white images. We ask to what extent can this process be automated? What trade-off space exists relating quality of output and human effort? With the rise of machine learning and Convolutional Neural Networks, it seems plausible to be able to automate or semi-automate this task.

If there exists a system that can colorize gray-scale images well enough for certain applications, then it is possible to construct a scenario in which one could compress data transfer to only a single gray-scale image instead of the typical 3 channel RGB or YUV images. For example low-quality always-on cameras are typically black and white. It might be useful to be able to convert chunks of this video feed into colored images without needing the entire stream to be saved as colored images.

1.2. Inputs and Outputs

Our architecture, RTNet, takes a gray-scale image and a hint image as inputs. The hint images are prepared by the user. A hint image contains 4 channels, The first channel is the gray-scale image, the next two represent the color channels and the final channel is a binary mask which implies whether a pixel contains any user-provided color information. RTNet then uses the color information from the hint image in addition to the original gray-scale image to colorize produce a colored image.

2. Related Work

There have been many attempts on using machines to assist coloring images. Most of these attempts do not use deep learning methods, and would require users to provide reference images that are very similar to the ground truth for the architectures to work. For example, the color transfer algorithm proposed by Welsh, Ashikhmin and Mueller [1] requires a fully-colored reference image to be passed in along with the target gray-scale image. Similarly, Sousa, Kabizadeh [2] proposed a colorization algorithm that as-

signs color to an individual pixel based on its intensity as learned from a color image with similar content. These algorithms have several downsides. First, in order to generate a good colorized image, the reference image provided by users must be very similar to the desired ones. Second, since features are extracted pixel-by-pixel, complexity of these algorithms are usually pretty high. For example, the latter group reported that the entire process of colorizing a 800-by-600 image takes about 30-45 minutes. This is very slow especially if the user wants to iterate over possible outputs.

Some other approaches do not require users to provide very good reference images, and can still generate good colored images efficiently. Levin, Lischinski and Weiss [3] provide a method that color objects in an image using color scribes provided by users. The algorithm is designed based on the assumption that pixels with similar gray-scale intensity in space-time should also have similar color. This premise is formulated using a quadratic cost function, and the colorization problem is transformed into an optimization problem that can be solved efficiently. This approach works well in a lot of cases. However, this architecture does not consider the problem of unsuitable color diffusion in different blocks. To solve this problem, Nie, Ma and Xiao [4] proposed a similar approach, which uses the standard deviation of intensity values of sampling points as a threshold to determine if a color diffusion is unwanted.

In addition, Chen, Wang, Schillings and Meinel [5] designed an approach to color gray-scale images by modelling the foreground and background color distributions using spatially varying sets of Gaussians. This approach generates good results, but is limited to the task of processing images with confusing luminance distributions.

Kekre and Thepade [6] proposed a method that prepares a color palette for coloring gray-scale images. The color palette is prepared using pixel windows taken from reference color image provided by user. Then the target gray-scale image is divided into pixel windows in the same manner. For every window of gray-scale image the palette is searched for equivalent color values to color the gray-scale window. This approach is very efficient as it does not need to optimize for the color palette for every single pixel. However, it may average the color within each window and reduce the color differences between different objects.

Horiuchi and Hirano [7] proposed an algorithm to color a gray-scale image by allowing users to plant 'color seeds' in the gray-scale sample. The color information of these pixels are then propagated through the picture to create a colored one. This approach requires minimal user input. However, colors of different objects within the picture would look very similar because they are generated by propagating the color information of a small set of pixels. Konushin, Vezhnevets [8] proposed a similar approach where users initialize some pixels with color. They were able to improve the accuracy of the algorithm by allowing users to do further tweaking after a rough color image is generated. The modification and refining of the image colors can be done very efficiently by utilizing the coupled map lattices (CML). Although we do not want users to spend too much effort on tweaking the final results, we are still inspired by this idea that we shall allow users to pass in more information than just a few pixels.

Luan [9] proposed a colorization algorithm based on color labeling and color mapping. At the first stage, pixels that should roughly have similar colors are grouped into coherent regions in the color labeling stage.The color mapping stage then fine-tunes colors in each region. Compared to the one proposed by Konushin and Vezhnevets, this approach may be more efficient in the sense that the fine-tuning stage needs to process no more than the total number of all the pixels.

Unfortunately, most of these aforementioned methods need users to provide sufficient reference images. In addition, the coloring process can take very long to complete. To overcome these problems, it is natural to use deep learning methods to assist the colorization process. Forward passing through a CNN can be very efficient and the models can have memory of various colorization techniques by training over sufficient amounts of data. One such architecture is designed by Cheng, Yang and Sheng [10]. They use a CNN with three hidden layers to extract high-level color palette, and use that to reconstruct color information for the targeted gray-scale image. One problem with this design is that they use mean square error to measure the difference between UV channels of the prediction and ground truth images as the loss function. Since there is no unique solution to coloring a gray-scale image, using mean square error as a loss measure may increase false positives.

Recently, a blog post entitled "Automatic Colorization" [11] by Ryan Dahl tries to solve this problem with a unique Convolutional Neural Network architecture inspired by ResNet ([12]). The results from this work were very promising and is what inspired us to take on this project. First, the model does not depend on the user to give a very good reference image. Second, although training takes a long time, forward-passing through the model can be done very efficiently. However, even though this approach can yield very good results, it is also limited by various factors. For example, the model is quite complex, and training it to a degree where colorization can be done properly requires significant amount of computation resources. In addition, this model attempts to reduce the loss by averaging the color assigned to various objects in the picture. This would cause the averaging color problem, and will be further discussed in the next section.

3. Methods

3.1. Problem Statement

The goal of this set of work is to find a universal function that transforms a gray-scale image to a full colored image. Specifically, given a single-channel gray-scale image, we want to output two-channel color information of the same size. A final colorized image can be constructed using the initial image and the function output. More precisely, The input will be assumed to be the Y channel of a YUV image and output will be the U and V channels for the YUV image. UV = F(Y). Note that the transformation between RGB and YUV space is a simple affine matrix multiply per pixel and therefore the two spaces can be considered equivalent. The Y "Luma" channel represents how bright the pixel is while the U and V "Chroma" channels represent a two dimensional color space.

Figure 1. Averaging Problem

It is important to note that it is impossible to create a function to exactly reconstruct color. Imagine two images of identical cars, one being red, and one being blue. It is possible to convert them both to gray-scale, and have the gray-scale images be identical. Because unique inputs can map to identical outputs, this implies that the function transforming a colored image to a gray-scale image is not injective. Therefore, an exact inverse function mapping gray-scale images to colored images cannot exist. Any function created will necessarily be approximate. Naively creating and training a large neural-net will cause what we term the "averaging problem". Objects that have a variety of colors for the same texture like cars, or fields will be predicted to have an average color. Ryan Dahl's results have corroborated this averaging problem as shown in Figure 1. The center image shows the car being an average color of cars in the training dataset. Our key insight is that more information might need to be given in addition to the gray-sale image in order to reconstruct the colored image. In this project we propose colorization to be supplemented with a user-created hint image. The function now becomes UV = F(Y, hint).

An example hint image can be seen as the input in Figure 3. The goal of this hint image is to provide minimal color information to objects thus hinting to the system as to what the color should be. A properly working function should be able to infer that if a small portion of an object is colored a certain color, the object itself should be colored similarly and realistically.

3.2. Architecture

3.2.1 RTNet

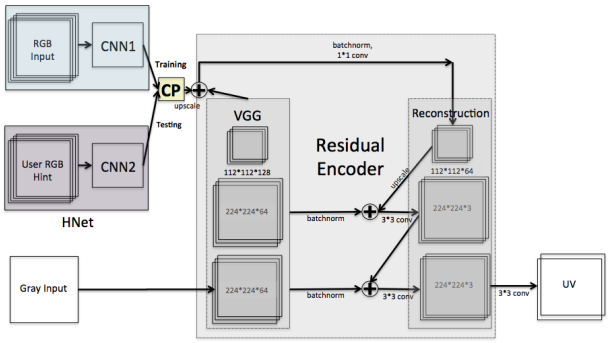

Figure 2. Architecture of RTNet

Our architecture contains two parts. The first part is the initial training that builds a model with the color information. The architecture of this part is shown in 2 , and is referred to as RTNet. The layers contained in the gray box can be recognized as a residual network similar to the one propose by Dahl. We use the residual network to solve two problems. First, using a residual network helps us prevent gradient from vanishing at deep layer of the network. Second, by using the residual network, we force the reconstruction layers to learn color information in addition to the encoded semantic information.

The VGG box represents a portion of the VGG16Net ([13]. We use this architecture based on the intuition that the VGG16 part will be able to extract semantic structure from the image. On top of the semantic information, we can use the color information from the Color Palette to colorize the image. The output is constructed by concatenating the $(N-i)^{th}$ layer with the i^{th} layer, where N is the number of all the layers in the residual encoder part. This way we are able to combine the semantic information from the VGG part with the color information at different granularities.

The gray input of the architecture is simply a 224x224x1 tensor representing the Y channel (gray-scale) portion of the image. The RGB input is a 224x224x3 tensor representing the entire YUV image. Note that this architecture trains a layer called the Color Palette which is a low dimensional space that should represent the colors in the image. The output of the entire model is a 224x224x2 tensor representing the U and V color channels. Note, that a sigmoid layer is used as the final layer causing all the pixels to be squashed between 0 and 1. Combining the original input Y channel with the predicted U and V channels and transforming to RGB will produce the final output image.

Here is a detailed explanation of how the whole training procedure for RTNet is done:

- The Y channel is forwarded into Gray Input.
- The YUV channels are forwarded into RGB Input.
- At the output of VGG, the Y channel is condensed into a $128 \times 112 \times 112$ tensor.
- At CP, the YUV information is first condensed into an R^{196} vector representing the color palette. It is then up-sampled with smoothing convolutions to a $32 \times 112 \times 112$ tensor.
- The two tensors obtained from the last two stages are concatenated. Then the result is fed into a reconstruction pipeline.
- At each new stage of reconstruction, we batch-normalize ([14]) the output at corresponding stage of VGG and concatenate this information with the reconstruction result from the last stage.
- At the last stage of reconstruction, we recover a $2 \times 224 \times 224$ tensor representing the U and V channels
- We then compute the loss of the UV channels and the expected UV channels. We back-propagate through the entire model using Adam optimizer.

Our loss function is defined as:
Loss $= \frac{1}{3}(\text{MSE}(Y_{\text{predict}} * \mathcal{N}_{3\times3}(0, 2.25) - Y_{\text{true}} * \mathcal{N}_{3\times3}(0, 2.25)) + \text{MSE}(Y_{\text{predict}} * \mathcal{N}_{5\times5}(0, 6.25) - Y_{\text{true}} * \mathcal{N}_{5\times5}(0, 6.25)) + \text{MSE}(Y_{\text{predict}} - Y_{\text{true}}))$.

We find that the differences between prediction and ground-truth images is better described by averaging the mean square errors of these two images and their blurred versions. We choose to use this loss function based on the observation that there is no unique way of coloring a gray-scale image. However, we can take the assumption that a pixel in the prediction image is colored correctly if it is close to the color of neighboring pixels around the ground-truth pixel. Using this loss function has several benefits. First, compared to using the euclidean distances, this loss function is more tolerant to pixels that are not exactly the same as their ground truth values but are very close. Second, using the blurred images for calculating errors increase the fault-tolerance threshold. As a result, it allows the model to converge faster. The second advantage is particularly important to us because our model contains very deep layers which would normally require significant amount of training. By having the model converging faster, we can train the model over larger data set and build better insights into how to improve this architecture.

3.2.2 HNet

The second part of our training is designed for images with hints. In a separate smaller CNN referred to as HNet, we have the input being a $224 \times 224 \times 4$ tensor and the output being the same Color Palette from the RTnet. The input's first three dimensions are the YUV channels and the last dimension will store the binary hint mask. If a pixel in the hint mask is marked as 1, then our architecture should use the color information at this location as a hint. Otherwise this pixel is considered as gray-scale. We use this part of the training to find the best hyperparameters that lead to good Color Palette. The loss of this architecture can be calculated by comparing the outputs from HNet and from RTnet.

The training data set for this part is obtained by augmenting the data set of color images from the first training step. For each (RGB Input, R^{196}) pair, we generate M hint images where each hint image is a $224 \times 224 \times 4$ tensor.

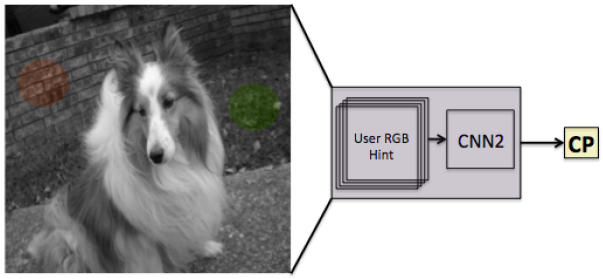

Figure 3. Architecture of HNet

For forward passing, we first replace the CNN1 network in RTNet with HNet. During prediction, we feed the hint image to the input of HNet and the gray scale image to the other input of the RTNet. The color palette produced by HNet will be forwarded into the corresponding spot in RTnet. Ideally, if HNet is sufficiently trained, it should be able to convert the User RGB Hint image into a Color Palette that is a close approximation to the one generated by the ideal RGB reconstruction of this hint image. By combining this information and forward-passing it into RTNet, we shall be able to obtain a colored image that is very close to its ground truth image without causing the averaging color problem.

4. Data Set and Features

As difficult of a problem colorization is, it is fortunately easy to obtain large quantities of training data for the task. Almost any RGB photograph will suffice. We used a large subset of the imagenet ILSVRC2012 datatset [15]. Images were preprocessed as follows.

- Any non-RGB image was thrown out
- The image was converted to YUV using an affine matrix multiplication.

- The YUV was scaled such that each channel's range was 0-256
- The scaled YUV was normalized by mean subtraction

First, we loaded the VGG portion of RTNet with pre-trained VGG weights. The final normalized Y channel was used as the gray-scale input. The normalized YUV was used as the training-time RGB input. Since the last layer of RTNet is a sigmoid which squashes values to the 0-1 range, our correct UV output also needed to be rescaled to be between 0 and 1 in order to correctly backpropagate.

For HNet, we automatically generated M Hint images. The assumption is that the blotches provided by the user will look roughly like a juxtaposition of colored circles on top of the gray-scale image. Because of this, the hint images consist of a random number of randomly sized circles. Specifically we used a uniform distribution of between 1 and 25 circles each having a circle radius sampled from a uniform distribution between 25 and 35 pixels. The circles can be overlapping. This was derived empirically based on the resulting distribution of image coverage was relatively uniform between 1% and 80%. Once this combined circle mask is generated, the hint image is generated by just multiplying the circle mask by the UV channels to extract color from only the circle regions.

RTNet was trained on roughly 65,000 unique images. Hnet was trained on roughly 65,000 hint images where the number of unique base images was roughly 2,000. For both RTNet and HNet, the inputs and outputs were preprocessed beforehand and written in a contiguous manner for ease of loading.

5. Experiments and Results

To actually train our architecture, we implemented both RTnet and HNet in Keras [16]. We chose this platform mostly for the modularity, readability, and ease of utilizing GPU resources. All of our training and testing procedures are run on an Amazon AWS instance with 1 GPU.

We started experimenting the full residual encoder architecture listed in Ryan Dahl's blog post. However we later realized that due to the complexity of the full model, it would take about a week to finish training 1 epoch of imagenet's training data. In order to reduce the training time, we redesigned the residual encoder such that it contains fewer layers. This likely decreased the quality of our results and it would be future work to use a larger architecture.

5.1. Choice of Hyperparameters

For RTNet, we set the learning rate at 10^{-3} using Adam optimizer. We did grid search to find the best combination that can drive down the loss function in the first few epochs. Also during our training process, it is shown that Adam is the most efficient optimizer.

We observed that larger batch size would lead to quicker convergence. However, the GPU memory limitation on our training machines only allows for no more than 16 samples per batch during training. Therefore, we limit batch size of 16.

We chose a validation split of 95:5. The curve of training loss and validation loss of RTNet is included in Figure 4. The curve does seem to imply convergence but is also noisy both in training and validation. The reason is that the validation loss was calculated after every 973 samples. We choose 973 samples because it is the maximum number of samples that we are allowed to load into our architecture without triggering memory error on the training machine.

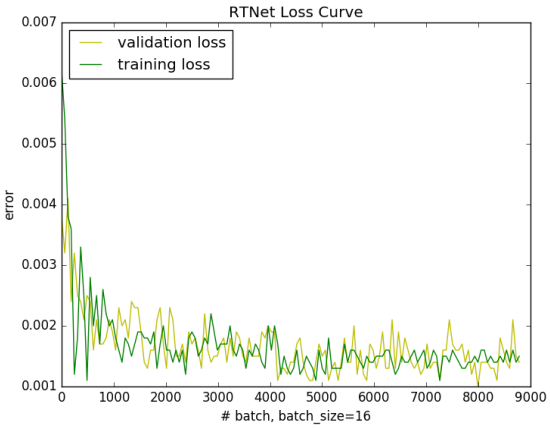

Figure 4. Training Loss and Validation Loss Curve of RTNet

After obtaining the best weights for RTNet, we collect the Color Palette corresponding to each training image. An example of a Color Palette is shown in 5. We observe that a lot of entries within the Color Palette are empty. This indicates that we should be able to further condense the color information from our training images. Specifically we used a ReLU layer for this Color Palette, while a sigmoid or tanh probably would have been more appropriate. Also, having a condensed Color Palette indicates that the VGG part of RTNet may be able to provide extra color information. Ideally we would want VGG to provide purely semantic information. We will discuss in the next section on how we can improve this.

We then train the HNet using hint images and Color Palettes. Our learning rate is set to 10^{-4}, and we are using Adam optimizer. Since HNet is a relatively light-weight CNN, it converges very quickly without too much work on tweaking the hyperparameters. The curve of training loss and validation loss is shown in 6.

It is observed that the loss decreases quickly within the first 100 batches. Most of the time the validation loss is

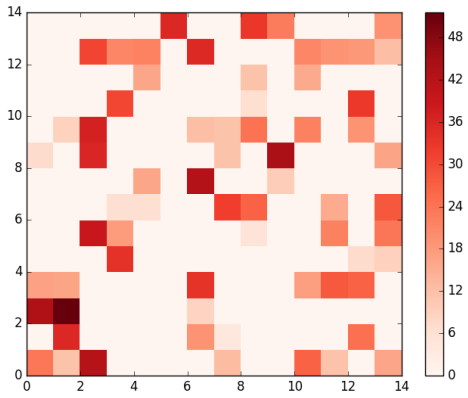

Figure 5. Visualization of Color Palette

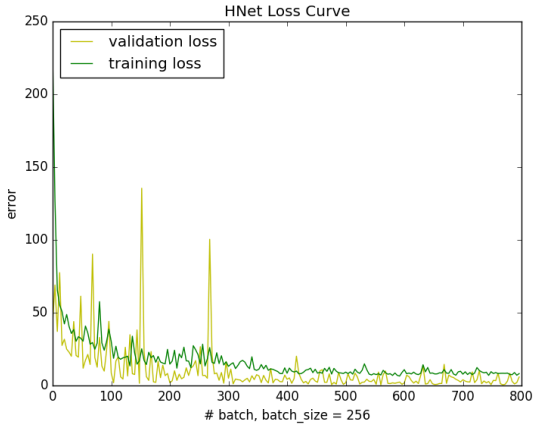

Figure 6. Training Loss and Validation Loss Curve of HNet

below the training loss, which indicates that the model is not over-fitting on the training set. However, we also observed that there exist spikes on the validation curve, which indicates that some of the validation pictures may not work well with the model we trained. To overcome this problem, we will propose several possible improvements in the next section.

Quality of hints are measured by the percentage of the hint filled in by color. To be clear, the worst possible hint is one that contains no (0%) color information while the best possible hint (resulting in the best possible output) is one which contains 100% of the color information. An experiment to show that increasing the amount of color in the hint corresponds to lower loss can be seen in The quantitative results are shown in Figure 7. A few examples of hints and their resulting outputs and diffs are shown in Figure 8. Hints are on the top row, resulting outputs in the middle row, and image diff from the best possible output is shown on the bottom row. As you can see, color information can somewhat be implied by small amount of hint color information.

Another thing to note is that our results show that the hint colors the sky to a similar color as the hint, not as an average result of the training set. Although coloring the sky can be considered an easier example, our model seemed to do well with both blue, red and yellow colors. As seen in the sky it was able to somewhat infer that a majority of the sky should be colored based off of a smaller amount of color information. Unfortunately the model had a tougher time with objects that are green and magenta. This is discussed in the next section and is likely a result of our architecture and image processing.

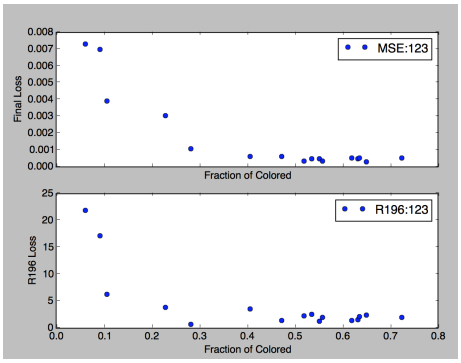

Figure 7.

6. Future Improvements

During the processing of implementing and validating our designs, we noticed several issues and would like to improve on them in the future.

First, we noticed that the green and magenta colors are not expressed enough in our final UV images. Ideally, all the colors should be expressed equally as we do not differentiate between them in our training and testing data set. On lead of the issue is that an empty V channel produces red while an empty U channel produces blue. Green and magenta require more complex combinations of the two channels and perhaps this is not captured by our model.

Second of all, the validation loss curve of our RTNet training is not very smooth. Though it may be a result of the fact that we only calculate validation loss at every 973 samples, it might as well be caused by our setting of the L2 regularizer. We are not able to retrain the model by using a more aggressive L2 weights since it takes very long for the model to complete back propagation; however we are interested to see if the training can be improved by more aggressive regularization.

Third, from the visualization of the Color Palette, it seems that a lot of the space is not used. This implies that the color information may be further compressed. Using a small Color Palette has the advantage of reducing storage space and training time. If we want to train a more fine-grained model, we would want to move to a smaller Color

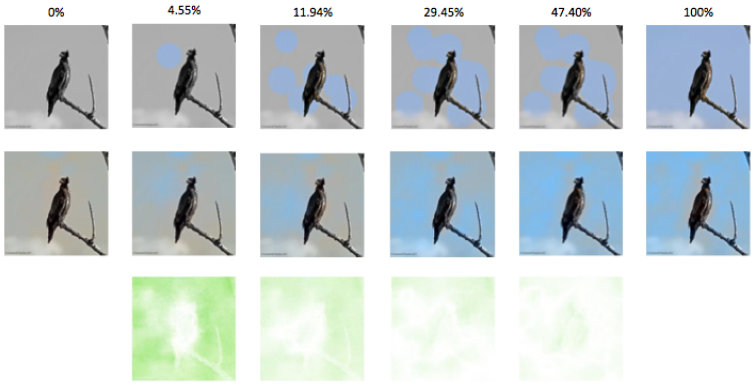

Figure 8. Colored results from hint images with different percentages of cover

Palette.

References

[1] Tomihisa Welsh, Michael Ashikhmin, and Klaus Mueller. Transferring color to greyscale images. *ACM Transactions on Graphics (TOG)*, 21(3):277–280, 2002.

[2] Austin Sousa, Rasoul Kabirzadeh, and Patrick Blaes. Automatic colorization of grayscale images.

[3] Anat Levin, Dani Lischinski, and Yair Weiss. Colorization using optimization. In *ACM Transactions on Graphics (TOG)*, volume 23, pages 689–694. ACM, 2004.

[4] Dongdong Nie, Qinyong Ma, Lizhuang Ma, and Shuangjiu Xiao. Optimization based grayscale image colorization. *Pattern recognition letters*, 28(12):1445–1451, 2007.

[5] Tongbo Chen, Yan Wang, Volker Schillings, and Christoph Meinel. Grayscale image matting and colorization. In *Proceedings of Asian Conference on Computer Vision*, pages 1164–1169. Citeseer, 2004.

[6] Hemant B Kekre and Sudeep D Thepade. Color traits transfer to grayscale images. In *Emerging Trends in Engineering and Technology, 2008. ICETET'08. First International Conference on*, pages 82–85. IEEE, 2008.

[7] Takahiko Horiuchi and Sayaka Hirano. Colorization algorithm for grayscale image by propagating seed pixels. In *Image Processing, 2003. ICIP 2003. Proceedings. 2003 International Conference on*, volume 1, pages I–457. IEEE, 2003.

[8] Vadim Konushin and Vladimir Vezhnevets. Interactive image colorization and recoloring based on coupled map lattices. In *Graphicon2006 conference proceedings, Novosibirsk Akademgorodok, Russia*, pages 231–234, 2006.

[9] Qing Luan, Fang Wen, Daniel Cohen-Or, Lin Liang, Ying-Qing Xu, and Heung-Yeung Shum. Natural image colorization. In *Proceedings of the 18th Eurographics conference on Rendering Techniques*, pages 309–320. Eurographics Association, 2007.

[10] Zezhou Cheng, Qingxiong Yang, and Bin Sheng. Deep colorization. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 415–423, 2015.

[11] Ryan Dahl. Automatic colorization.

[12] Kaiming He, Xiangy Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *arXiv:1512.03385*.

[13] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. In *arXiv:1409.1556*.

[14] Sergey Ioffe and Christian Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. In *arXiv:1502.03167*.

[15] Dong Wei Socher Richard Li Li-Jia Li Kai Deng, Jia and Li Fei-Fei. Imagenet: A large-scale hierarchical image database. In *Computer Vision and Pattern Recog- nition, 2009. CVPR 2009. IEEE Conference on, pp. 248 255. IEEE*.

[16] Franois Chollet. keras. `https://github.com/fchollet/keras`, 2015.