Image Colorization

Ede Komjati
komjatiede@gmail.com

***Abstract*. This paper contains description about three approach for the colorization problem. First, I will introduce already known researches. I started out from these, because I wanted to avoid already solved part of the main problem issues. I created two type of architecture. One which handles the problem as a classification one, and the other which handles as a regression one. I am comparing these models and show the results with sample pictures.**

Keywords—deep learning, image colorization, classification

I. INTRODUCTION

In these days there is an increasing demand on artificial intelligence-based solutions. I am member of a Facebook group where people collect old, legacy pictures where most of them are gray, colorless. They already tried to colorize those pictures with more or less success. I chose this problem to produce a solution what can help them out with the colorization. I was also curious to implement a deeper Convolutional Neural Network – CNN. First, I will describe similar solutions to this problem. After that I describe my own methods and solutions with Neural Network architectures. I started this homework with a fellow student of the Deep learning subject, but in the semester, he didn't has the time to work on this homework so I become alone for the project. He only helped me with the existing researching papers.

II. EXISTING RESEARCHES

A. Colorful Image Colorization

I started out of this article. Zhang et al. approached as a classification problem. They trained a CNN to map from a grayscale input to a distribution over quantized color value outputs. First let's see their objective function. [1]

1) Objective Function

Given an input lightness channel $X \in R^{HxWx1}$, their objective is to learn a mapping $\hat{Y} = F(X)$ to the two associated color channels $Y \in R^{HxWx2}$, where H, W are image dimensions.
(I denote predictions with a ˆ symbol and ground truth without.) They performed this task in CIE Lab color space. Because distances in this space model perceptual distance, a natural objective function is the Euclidean loss $L2(\cdot, \cdot)$ between predicted and ground truth colors:

$$L_2(\hat{Y}, Y) = \frac{1}{2}\sum_{h,w} \left\| Y_{h,w} - \hat{Y}_{h,w} \right\|^2 \qquad (1)$$

However, this loss is not robust to the inherent ambiguity and multimodal nature of the colorization problem. If an object can take on a set of distinct ab values, the optimal solution to the Euclidean loss will be the mean of the set. In color prediction, this averaging effect favors grayish, desaturated results.

Instead they use multinomial cross entropy loss defined as:

$$L_{cl}(\hat{Z}, Z) = -\sum_{h,w} v(Z_{h,w}) \sum_q Z_{h,w,q} log(\hat{Z}_{h,w,q}) \quad (2)$$

Where v(.) is a weighting term that can be used to rebalance, the loss based on color-rarity. [1]
They made it by quantizing the *ab* output space into bins with grid size 10 and keep the $Q = 313$ values which are in-gamut. For a given input X, they learn a mapping $\hat{Z} = G(X)$ to a probability distribution over possible colors $\hat{Z} \in [0,1]^{HxWxQ}$, where Q is the number of quantized ab values.

2) Class rebalancing

The distribution of ab values in natural images is strongly biased towards values with low ab values, due to the appearance of backgrounds such as clouds, pavement, dirt, and walls. They gathered from 1.3M training images in ImageNet [2] and they produce from this an empirical distribution, which can weight the loss function to dominate desaturated values.

$$v(Z_{h,w}) = w_{q*}, where\ q *= arg\ max Z_{h,w,q} \quad (3)$$

3) Class Probabilities to Point Estimates

They defined H, which maps the predicted distribution $\hat{Z}$ to point estimate $\hat{Y}$ in ab space. One choice is to take the mode of the predicted distribution for each pixel. This provides a vibrant but sometimes spatially inconsistent result. On the other hand, taking the mean of the predicted distribution produces spatially consistent but desaturated results, exhibiting an unnatural sepia tone. This is unsurprising, as taking the mean after performing classification suffers from some of the same issues as optimizing for a Euclidean loss in a regression framework. To try to get the best of both worlds, they interpolated by re-adjusting the temperature T of the softmax distribution and taking the mean of the result. They draw inspiration from the simulated annealing technique, and thus refer to the operation as taking the annealed-mean of the distribution:

$$H(Z_{h,w}) = E[f_T(Z_{h,w})] \qquad (4)$$

$$f_T(z) = \frac{exp(log(z)/T)}{\sum_q exp(log(z_q)/T)} \qquad (5)$$

B. Image Colorization using Generative Adversarial Networks

The other article [3] which I want to introduce is a solution for colorization with a GAN based network [4]. In their approach, they attempt to fully generalize the colorization procedure using a conditional Deep Convolutional Generative Adversarial Network (DCGAN)

[5] [6]. The network is trained over datasets that are publicly available such as CIFAR-10 and Places365. The result as compared against a based Convolutional network.

1) Baseline Network

They followed a "fully convolutional network" [7] model guideline. They replaced the fully connected layers by convolutional layers which include up sampling instead of pooling operators. This idea is based on encoder-decoder networks where input is progressively down sampled using a series of contractive encoding layers, and then the process is reversed using a series of expansive decoding layers to reconstruct the input. Using this method, they could train the model end-to-end without consuming large amounts of memory.

Their baseline model needed to find a direct mapping from the grayscale image space to color image space. However, there was an information bottleneck that prevents flow of the low-level information in the network in the encoder-decoder architecture. To fix this problem, features from the contracting path were concatenated with the up sampled output in the expansive path within the network. This also makes the input and output share the locations of prominent edges in grayscale and colored images. This architecture is called U-Net [8], where skip connections were added between layer i and layer n-i.

The architecture of the model is symmetric, with n encoding units and n decoding units. The contracting path consists of 4 × 4 convolution layers with stride 2 for down sampling, each followed by batch normalization [9] and Leaky-ReLU [10] activation function with the slope of 0.2. The number of channels are doubled after each step. Each unit in the expansive path consists of a 4 × 4 transposed convolutional layer with stride 2 for upsampling, concatenation with the activation map of the mirroring layer in the contracting path, followed by batch normalization and ReLU activation function. The last layer of the network is a 1 × 1 convolution which is equivalent to cross-channel parametric pooling layer. They used tanh function for the last layer.. The number of channels in the output layer is 3 with L*a*b* color space.

2) Convolutional GAN

For the generator and discriminator models, they followed Deep Convolutional GANs (DCGAN) [5] guidelines and employed convolutional networks in both generator and discriminator architectures. The architecture was also modified as a conditional GAN instead of a traditional DCGAN. The architecture of generator G is the same as the baseline. For discriminator D, they used similar architecture as the baselines contractive path: a series of 4 × 4 convolutional layers with stride 2 with the number of channels being doubled after each down sampling. All convolution layers are followed by batch normalization, leaky ReLU activation with slope 0.2. After the last layer, a convolution was applied to map to a 1-dimensional output, followed by a sigmoid function to return a probability value of the input being real or fake. The input of the discriminator was a colored image either coming from the generator or true labels, concatenated with the grayscale image.

III. Architecture

For the neural network architecture, I was sure to use convolutional layers for the better understanding of the features. Because the RGB channeled representation didn't separate the lightness and the color information I switched the color space type into Lab [11]. In short this means each pixel built up from 3 parameter: L as Lightness, this adds the picture look; A, B these will be the color parameter. Because the lack of GPU memory, I didn't want to train the network for hundreds of hours just to understand what is in the picture. Because of that I am using a pretrained convolutional network for my first model called VGG16 in Keras [12].

A. Classification VGG16 model - CLVGG

Based on this, the preprocessing step was the same as in the VGG: I subtracted the mean pixel value of every pixel. This layer wants a 224x224x3 shaped input with RGB images, so I made it, with the trick: I added the original picture modified Lightness value, the modification moved L interval from 0-100 to 0-25, to all three channels. It is like I gave a grayscale picture as input. The VGG layers based on 5 convolutional blocks continued by fully connected layers. The blocks build up like this: 2 or 3 2-dimensional convolution layer, filters always 3x3 with padding to stay in the same shape, followed by a batch normalization and a max pooling filter 2x2. After every block the channel counts doubled, as the picture size halved. For example, input size 224 -> 112 -> 56 ->28 ->14->7. The layer with the picture size 7 didn't belong to blocks, it is just a single maxpooling layer. I connected the VGG block with the Keras Subsample layer. Because I tried out more model architecture, I have to change the subsampler often.

So, the A version of this model: the subsampler should output 56x56 sized picture. After the subsampler, I created and connected with each other 3 2-dimensional convolutional layers with filter 3x3, activation ReLU which output channels are 313. The output layer is a same 2-dimensional convolution layer, but with a Softmax activation and the filter size is 1x1. (CLVGG-A)

In the B version (CLVGG-B) I created a subsampler which increase the size for twice. After that two 3x3 convolution coming with 'same' padding, 256 channel, and activation ReLU. Here come the second subsampler which increase the size by 4 times, after that 3x3 convolution with 256 channel, padding same, and activation ReLU. It is continuing with two 313 channeled 3x3 convolution, after that the output layer is a 1x1 convolution with 313 channel and softmax activation. I am using a custom loss function which equation is here:

$$L_{cl}(\hat{Z}, Z) = -\sum_{h,w} v(Z_{h,w}) \sum_q Z_{h,w,q} log(\hat{Z}_{h,w,q}) \quad (6)$$

It is a multinomial cross entropy loss, with a helping term in v(.). V(.) is a weighting term that can be used to rebalance the loss based on color-class rarity.

$$v(Z_{h,w}) = ((1 - \lambda)\tilde{p} + \frac{\lambda}{2})^{-1} \quad (7)$$

$\tilde{p}$ is the empirical distribution each of the 313 color, λ is a hyperparameter what I can decide what value can be.

B. U-Net based model - Unet

My second attempt to solve this problem is similar to the above section. I use again a pretrained net, it is called U-Net [8].

It consists of the repeated application of two 3x3 convolutions (unpadded convolutions), each followed by a rectified linear unit (ReLU) and a 2x2 max pooling operation with stride 2 for downsampling. At each downsampling step I doubled the number of feature channels. Every step in the expansive path consists of an upsampling of the feature map followed by a 2x2 convolution ("up-convolution") that halves the number of feature channels, a concatenation with the correspondingly cropped feature map from the contracting path, and two 3x3 convolutions, each followed by a ReLU. At the final layer a 1x1 convolution is used to map each channel into 2 channel: a,b. For the output activation, I am using a Tanh loss function. (Unet)

IV. Methods

A. Data preparation

I collected the pictures from 2 different sources. The 1st is the Google Open Images V4 dataset. I have downloaded several classes from here. The labels with the unique names separated in another csv file than the picture urls. I created a class which joins these csv-s. It uses the Pandas Dataframe solve the problem. Because the csv which contains the urls is big (about 4.5 Gb) and there are 3 other csv file which needed to download the required labels, the data preparation method is a memory heavy method, my computer needed 12-13 Gigabyte memory to prepare the operations. After this, I have got a class which contain a joined table with the urls what I queried for. There is a method, that starts the download from the urls and collect into a desired folder. The last preprocessing step is to separate the train validation and test data, because I will load into separate data generator classes. I do the separation with the help of the Linux command line.

The second source is the Facebook group, where the community uploading several grayscale images. These images were hard to colorize for my network as I've tested.

B. Output classification description

In naïve way to solve this problem is to output the problem as a regression problem where the output layer is a convolutional layer with 2 channels or a fully connected layer 2 channel x height x width number nodes. In this situation I would use the mean squared error function for training. The thing is that in this situation the colors of the image look too desaturated for the viewer. The other approach what I used up is to create a classification problem where every pixel needs to classify. The unique classes are A,B tuple pairs (A,B from Lab colorspace). Based on this, I collected a list of a,b tuples on it. Its length is 313. I shaped the pixels into 313 length hot encoded vectors. This 1x313 vectors equivalent to a class from the A,B tuple list which equivalent to a color. In the next section I will also write about the problem and my solution against the memory heavy target data.

C. Training

In this section I will talk about more details about my training sessions, and other methods to speed up and be able to do at all.

1) Training with naïve data loading

My first attempt was loading all of my picture into memory, produce the target data and after this is in the memory I start the training. For this attempt I already prepared 5000 picture of city, landscape and other label name which reference into buildings, cities, landscapes. First, I preprocessed the images to meet the 224x224 input size requirement and concatenate the lightness value 3 times this produce a Numpy ndarray with the shape of 5000x224x224x3. I store each number in float which size is 8 byte. This means that just the collection cost the memory about 5.6 Gigabyte!! Alright, I also need the Y targets too, how much is it for the memory? Well the target shape is: 5000x56x56x313, the numbers are floats (8-byte for memory, one variable), which means: 36.56 Gigabyte!! Just my training data costs the memory about 42 Gigabyte. After some training, I decided to create something that boosts the performance because only a single thread did the loading, I needed distributed data loading.

2) Training with custom data generators

After my naïve implementation I created a data streamer class. This generates the target data from the images in real time on CPU when Keras access to the batches. With this method, I can parallelize the target data creation. I can set the number of workers that produce target batches and put it into a sequence. This sequence contains neural network train ready data. I am using the help of Keras.utils.Sequence class, I inherit from it. With this, I can use more picture without the fear of load up too much data into the memory.

D. Evaluation

For CLVGG models,I am using the categorical cross entropy loss function and the weighted categorical cross entropy loss function what I implemented with the help of the keras backend. My metrics are simple accuracy and categorical accuracy. I chose those metrics, because they have got better representation against the loss function. I thought it is modeling the problem better and the state how accurate my network is. In practice every epoch they had got the same value, so I will continue only with the one of those, accuracy. After the trainings I concluded, accuracy not always the best to represent the performance. I write about this in the test section.

For Unet model I am using mean squared error loss, with the metrics of accuracy. Because of the poor quality of colorization I spent more time with the CLVGG models.

After training my models accuracy move into a 30-55% interval. I save every epoch information into a log csv file. This csv file contains the loss, validation loss, accuracy, val. accuracy, categorical accuracy, val. categorical accuracy for each epoch. I used the logs to visualize the training curves and to make predictions after the training. Below you can see the better training accuracy from each models.

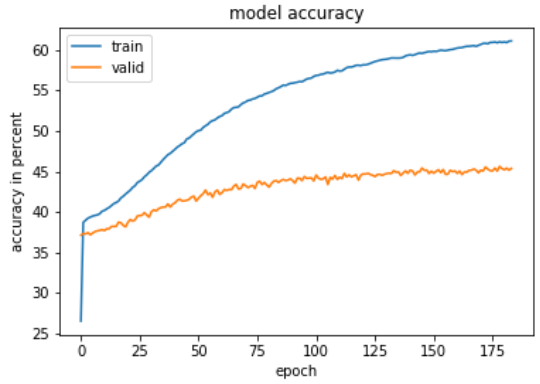

Figure 1

Accuracy with the CLVGG-A model.

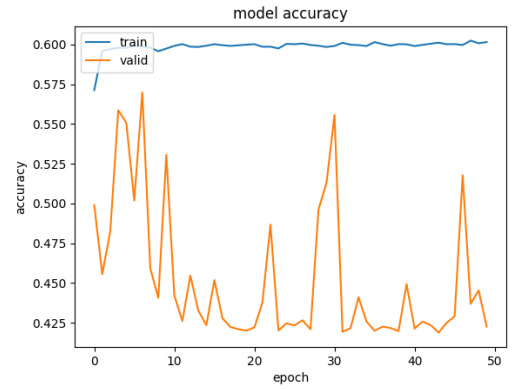

Figure 2

Accuracy with the Unet model.

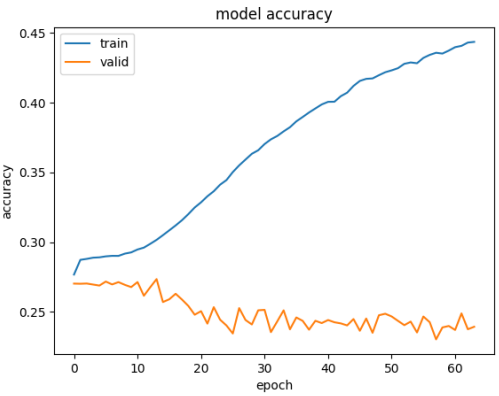

Figure 3

Accuracy with the CLVGG-B model.

As the results show the best model is Unet, if I base on only the accuracy. I visualize the performance in the next section. I came into a conclusion that not always the accuracy is the best metric for the trainings.

E. Tests

1) Test database

I created a separated test sample case, which useful to give me a feedback how is my model doing the colorizing against with those pictures. The tests always run after the training and give me a loss and accuracy number. For the first test I am using just four labels from Google Open Image v4, these are: city, landscape, Cityscape, boathouse.

Figure 4

Image samples from my dataset.

After the first good results I increased the count of the images. I put there some legacy picture to test from Facebook. In the next section I wrote about it.

2) Tests with the downloaded pictures from the facebook group

I used up images from a Facebook group, they have a lot of legacy grayscale picture from the old Budapest. My goal is to create a model what can colorize automatically.

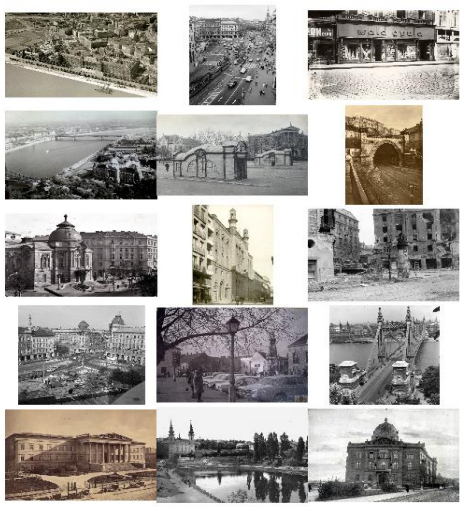

Figure 5

Image samples from the Facebook group.

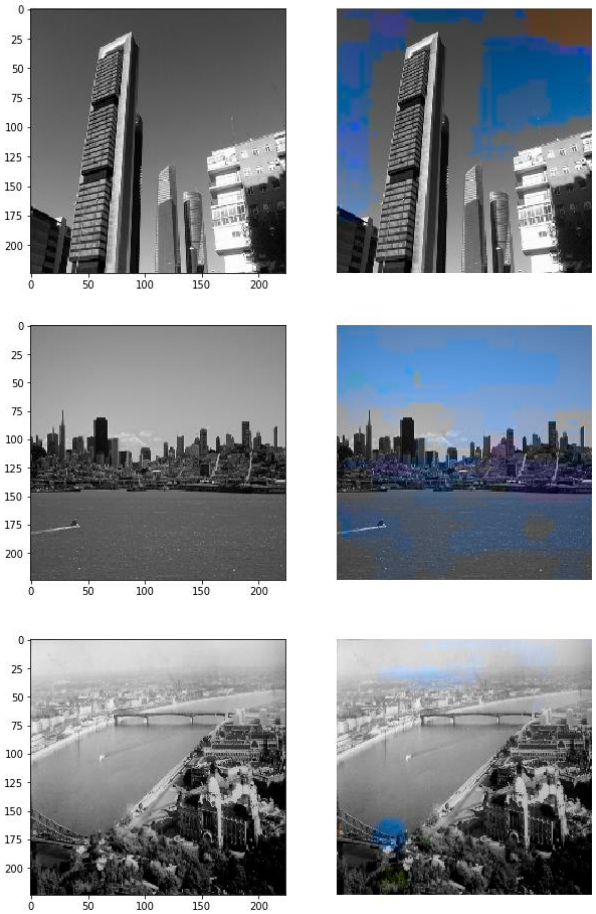

Figure 6

Better results with CLVGG-A

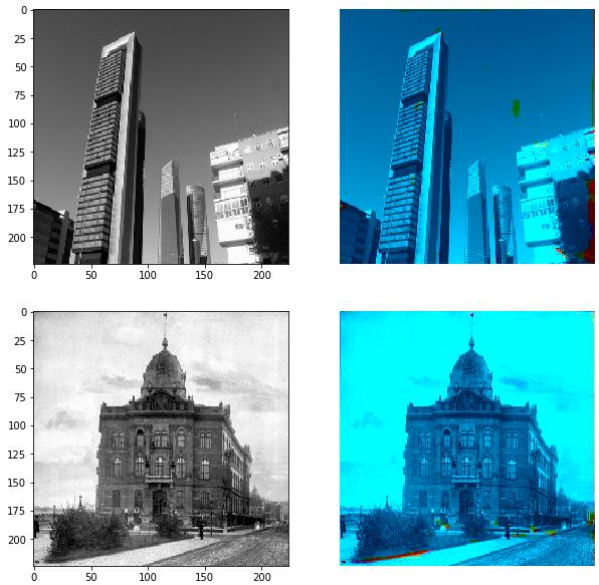

Figure 7

Unet results.

As you can see above at Figure 7. my Unet model tried to colorize everything into blue. I think this is because the high amount of blue pixels in the pictures. When I realized that, I created the weighting term for the loss function.

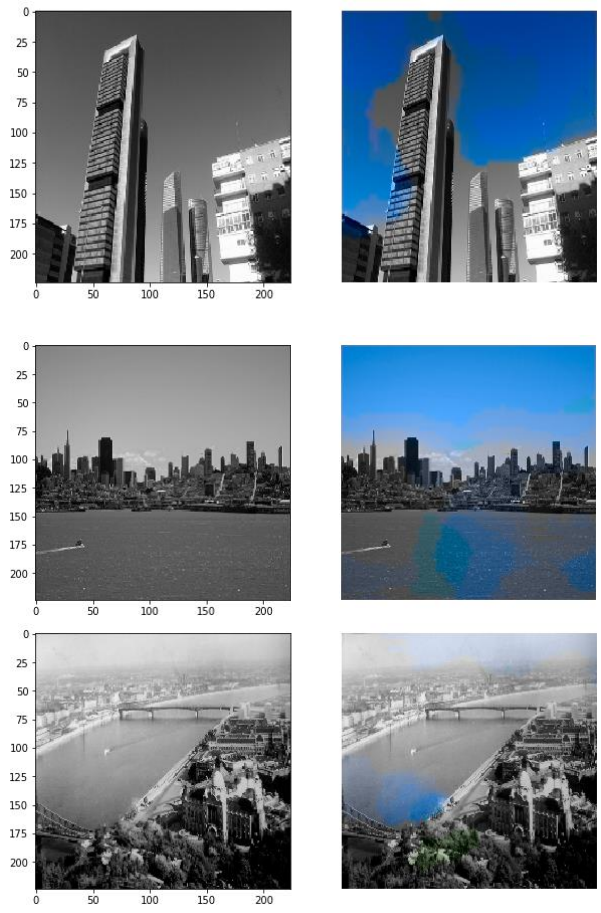

Figure 8

Better results with CLVGG-B

3) Grayscale video which can be colorized.

I had an idea, to colorize a simple video. I implemented the video colorizer. It is loading up my trained model, loads up the video and colorize every frame, displaying onto the screen.

V. Plans for the future, summing up my research

Because my teammate left our team, I did this whole project alone, although I could implement much more than I thought. Basically, I created a model which consumes a lot of hardware capacity, because of that the training was really slow. My model couldn't generalize from the pictures I have added to it, but I think I learned a lot of concept how wide perspectively can approach a problem (like building a GAN network for it, or just a simply convolutional classification), how to optimize data loading.

This problem is hard I think, if I had more time for it I would do more optimization. There are methods which is in my head but I couldn't implemented it: regression methods, train the whole network not just 3-7 layers (these layers have 4-5 million parameter).

A. Custom data generator

The only problem that I didn't solve for my custom data generator: the x input data, which loading into one list the memory. In the section Methods, Training, it means that if I had 10000 pictures then it would cost 10 Gigabyte of my memory.

VI. References

[1] Z. Richard, I. Phillip and A. E. Alexei, "Colorful Image Colorization," pp. 4-6, 2016.

[2] O. Russakovsky, J. Deng, H. Su, J. Krause, S. Satheesh, S. Ma and Z. Huang, "Imagenet large scale visual recognition challenge. International Journal of Computer Vision," pp. 211-252, 2015.

[3] K. Nazeri, E. Ng and M. Ebrahimi, "Image Colorization using Generative Adversarial Networks," 2018.

[4] I. J. Goodfellow, J. Pouget-Abadie and a. Et, "Generative Adversarial Nets".

[5] A. Radford, L. Metz and S. Chintala, "Unsupervised representation learning with Deep Convolutional Generative Adversial Networks," 2016.

[6] M. Mirza and S. Osindero, "Conditional Generative Adversarial Nets," 2014.

[7] J. Long, E. Shelhamer and T. Darrell, "Fully Convolutional Networks for Semantic Segmentation".

[8] O. Ronneberger, P. Fischer and T. Brox, "U-Net: Convolutional Networks for Biomedical".

[9] S. Ioffe and C. Szegedy, "Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift".

[10] A. L. Maas, A. Y. Hannun and A. Y. Ng, "Rectifier Nonlinearities Improve Neural Network Acoustic Models".

[11] "CIELAB color space," [Online]. Available: https://en.wikipedia.org/wiki/CIELAB_color_space.

[12] K. Simonyan and A. Zisserman, "Very Deep Convolutional Networks for Large-Scale Image Recognition," 2014.