UKRAINIAN CATHOLIC UNIVERSITY

BACHELOR THESIS

Template-based color correction in alternating lighting conditions

Author:
Yur TEPLIUKH

Supervisor:
Oles DOBOSEVYCH

A thesis submitted in fulfillment of the requirements for the degree of Bachelor of Science

in the

Department of Computer Sciences
Faculty of Applied Sciences

Lviv 2019

Declaration of Authorship

I, Yur TEPLIUKH, declare that this thesis titled, "Template-based color correction in alternating lighting conditions" and the work presented in it are my own. I confirm that:

- This work was done wholly or mainly while in candidature for a research degree at this University.
- Where any part of this thesis has previously been submitted for a degree or any other qualification at this University or any other institution, this has been clearly stated.
- Where I have consulted the published work of others, this is always clearly attributed.
- Where I have quoted from the work of others, the source is always given. With the exception of such quotations, this thesis is entirely my own work.
- I have acknowledged all main sources of help.
- Where the thesis is based on work done by myself jointly with others, I have made clear exactly what was done by others and what I have contributed myself.

Signed:

Date:

UKRAINIAN CATHOLIC UNIVERSITY

Faculty of Applied Sciences

Bachelor of Science

Template-based color correction in alternating lighting conditions

by Yur TEPLIUKH

Abstract

Having the right colors is one of the best ways for picture to look good. But not only humans' eye will appreciate correctly colored image, but also a computer, especially when the task is to track object by its color. While this might not seem like a difficult one, but when you have really long-term tracking, in an environment where lightning conditions change over time, you want to be sure that your algorithm is still able to find desired object. Or maybe you cannot or don't want to manually adjust camera color settings, but still need to track same object during the day, sunset and in the afternoon, when everything changes it colors to red and blue afterwards.

The main idea behind this work is to create an automatic pipeline which will output either corrected image, or the mask, which should be applied to produce corrected image, depending on selected method. This is achieved by using a template with reference color values, which is then compared with the colors on the input image and proceeded to color estimation algorithms. Different types of templates and color estimation algorithms were compared to select the most effective to provide color stability. We show that you don't need a special templates to achieve good results in color calibration.

Here is the link to the dataset images, as well as calibrated ones.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Motivation

Everyone who has ever taken a photo, which is probably just everyone, might experienced situations when the photo had terrible colors. There can be a lot of reasons for that, especially now, when before you get a photo taken, it comes a long way through different algorithms that are supposed to make it look better. After that, when an image is saved, the compression algorithm does his job to decrease the photo's size, making things even worse. There are different ways to post-process an image and countless amount of software, where you can do whatever you want with what you have filmed. But usually, to get the result you would be satisfied with, you must do some tweaks be yourself. Adjust brightness, add some saturation, increase temperature, maybe even play with channel curves. But how would you know whether you are doing it correctly? What is the measurement of the properly corrected image? We think that this can be how accurate the colors on the image are. And while for an average user, it might not always be the case, because natural colors are not always the prettiest one, and there are many fields where having the right colors is extremely important. For example, food sciences, where depending on the color of foodstuff, one can tell whether they are fresh or not. Or in computer vision, especially color detection. Imagine that your task is to track red objects during the whole day, from early morning to late evening. During the day, light conditions are changing: sunrise, afternoon, evening, sunset, night. Light is changing all the time, and you should either implement some technique to compensate these changes or manually change the color, which is tracked by your algorithm. Or you can use the approach proposed in this thesis.

1.2 Goals

- Create a framework for automatic color detection, which (in general) does not require manual adjustments.
- Describe and compare different color correction algorithms.

1.3 Structure

- Chapter 2. Background information

  This chapter contains some main concepts in computer vision: object and color detection, convolutional neural networks as well as color correction theory.

- Chapter 3. Related works

Here we analyze various research topics and available methods for object detection and color correction, describe ones which are used in this thesis.

- Chapter 4. Test datasets description

  In this this chapter, an overview of gathered datasets is being presented, as well as comments about data selection with visual references.

- Chapter 5. The proposed approach

  This chapter is a description of proposed pipeline, architecture details and description of result evaluation methods.

- Chapter 6. Achieved results

  In this part achieved results are being presented, as well as comparison between different color correction approaches with visual examples.

Chapter 2

Background information

Beyond doubt, our eyes are incredibly important part of our life. According to [18], we receive near 875 KB of information from our eyes every second. In a research [15] by Fabian Hutmacher, where respondents were asked to select losing of which sense scares them the most, the answer of the majority (73.63%) was vision.

Humans' eye is a complex system which is the result of constant evolution for nearly 550 million years (3) since the first image-forming eyes evolved with the appearance of the stem mollusk *Clementechiton sonorensis* [21]. Despite the fact that studying computer vision began in the late 1960s, just about 50 years ago, due to advancements of computer technologies and growing interest to this field during the last decades we already have impressive results in detecting, classifying and localising certain things on photos and videos in real time, which opens endless possibilities to such things as image stitching, autonomous driving, facial authentication and many others.

2.1 Object detection

Computer vision is a field of study focused on learning computer to extract meaningful information form images or videos. From the 2015 ILSVRC paper [29], it can be split into three tasks:

- Image classification: prediction of class of an object in an image. Results in class labels.
- Object localization: locate on object in an image and draw a bounding box indication its position. Results in coordinates of objects bounding box.
- Object detection: Basically, combination of two previous tasks. Results in bounding boxes with labels for each recognized object.

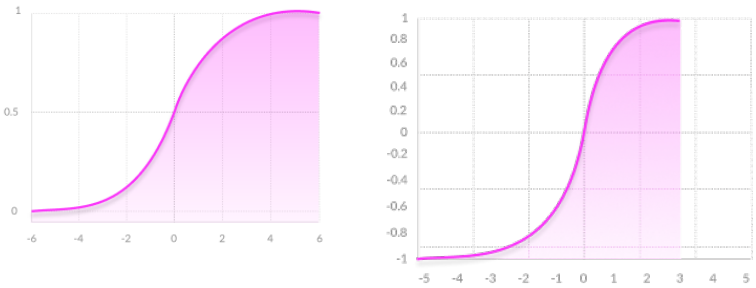
While object detection is an effortless task for a human, to teach computer not only to see things, but also be able to specify them was proven to be quite a challenging task. Back in 2000s the most popular technique was bag-of-visual-words, which used hand crafted descriptors, such as David Lowe's scale-invariant feature transform (SIFT) [20] and histogram of oriented gradients (HOG) [11] by Dalal et. al. The first deep convolutional neural network (CNN) has been introduced at ILSVRC 2012 by Krizhevsky et.al. [19], significantly outperforming all the other methods: classification error was reduced from 25.2% to 15.3%, and localisation error was decreased from 50% to 34.3%. Since then, error rate of convolutional networks decreased to just 2.25% classification error [14] and 6.23% localisation error in 2017, while mean average detection precision increases from 23% in 2013 to 73% [8] in the same 2017.

2.2 Neural Networks

In computer science, artificial neural network (ANN) are algorithms which are inspired by humans brain. They consist of neurons, which are connected together into several layers, where neurons have connection to some (or, in case of fully connected layer, to all) neurons from previous layer, with a weight coefficient associated to each connection (which works like brain synapses) to show how important its input data is. Each neuron can have several input and out connections. Equation below shows how single neuron works:

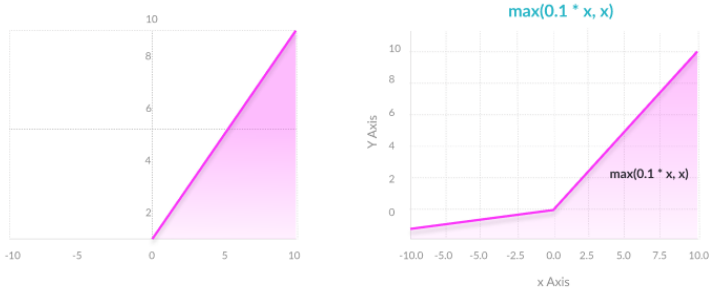
$$o_k = \phi\left(\sum_{i=0}^{m} w_{kj} x_j\right) \tag{2.1}$$

For a given neuron, x_0 to x_m are input signals with associated weights w_j and ϕ is called *activation function*. The purpose of activation function is to map previously calculated sum between 0 and 1, some examples of such functions are sigmoid (2.1a), tanh (2.1b), ReLU (2.1c) [26], Leaky ReLU (2.1d).



(A) $f_\theta(x) = \frac{1}{1+e^{-\theta^T x}}$ [source image]

(B) $f(x) = tanh(x)$ [source image]



(C) $f(x) = max(0, x)$ [source image]

(D) $f(x, \alpha) = max(\alpha * x, x)$ [source image]

FIGURE 2.1: Example of activation functions (from left to right): sigmoid, tanh, ReLU, Leaky ReLU

On Fig. 2.2 is shown what a simple neural network might look like. It has 2 inputs (*x1* and *x2*), single hidden layer with 2 neurons (*h1* and *h2*) and a single output neuron *o1*. The layers between input and output are called *hidden layers*.

Neural network are capable of finding dependencies in complex data, and to achieve this training process is being used. It is performed in two steps: first the data with known ground-truth values is being forwarded to network, and the error between received and ground-truth values is calculated, and then this error is used to update initial weights. Training is the form of loss function minimization, and common methods for this are stochastic gradient descent-based [4] optimization

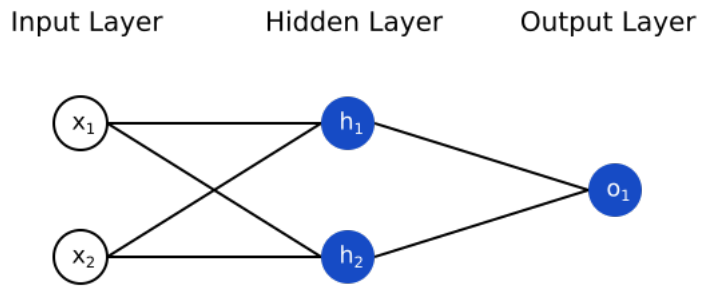


FIGURE 2.2: Schematic image of simple one-layer neural network [source image]

methods, which take only part of data from each iteration and use them to optimize learning. Despite the fact that part of the data is not always fully representative, making a lot of smaller optimizations eventually converges to minimum of the loss function [5].

To improve the learning process, input data is into several subsets according to [28]:

- Training set: the biggest subset, used for training itself.
- Validation set: this part is used to evaluate model performance and calculate error to tune hyperparameters and test model generalization capabilities. Hyperparametets are certain parameters which are used to control learning process and they are set before learning begins.
- Test set: subset, which is not involved in training process at all, used for final evaluation. It is used to check how accurate network performs on the new data and to test whether it was not overfitted by weights calculated on validation set.

2.3 Convolutional Neural Networks

Convolutional networks are special types of neural networks which are suited for object detection and classification on images and videos. The problem of using traditional networks is that for a black&white image of size 1280x720 pixels each neuron inside a fully-connected layer requires $1280 * 720 = 921600$ weights, and for colored RGB image number of weights will increase by three times. Such high number of weights raises the problem of potential overfitting and reduced performance. To decrease number of weights, convolutional networks search for relations in local neighbourhood of points: there's no need to consider the value of bottom-right pixel if the object we are searching for is in the top-left corner. Searching for simple structures such as edges and lines helps to derive complex objects from them.

The name convolutinal network cames form mathematical operation convolution: it is an operation between discrete functions f and g, which is defined as

$$(f * g)(t) = \sum_{x=-\infty}^{\infty} f(x) \cdot (t - x) \tag{2.2}$$

and applied to each image pixel.

Here, f is the intensity value of a given pixel and g is a 2-dimensional non-zero square array called *kernel* of size k (usually 3×3, 5×5 or 7×7). Depending on kernel values it can be used for different image manipulations: detecting of horizontal edges (2.3a), vertical edges (2.3b), applying Gaussian blur to the image(2.3c).

$$\begin{pmatrix} 1 & 2 & 1 \\ 0 & 0 & 0 \\ -1 & -2 & -1 \end{pmatrix} \quad \begin{pmatrix} 1 & 0 & -1 \\ 2 & 0 & -2 \\ 1 & 0 & -1 \end{pmatrix} \quad \begin{pmatrix} 1 & 2 & 1 \\ 2 & 4 & 2 \\ 1 & 2 & 1 \end{pmatrix}$$

(A) (B) (C)

FIGURE 2.3: Example of convolution kernels

CNN consists of several convolutional layers, where each layer is a set of neurons where each neuron has its own region (also called *receptive field*) where convolutions are applied. First layers are responsible for detecting such features as corners, edges, gradient orientations, etc. The next ones are combining them into more complex shape detectors.

2.3.1 Padding

The size of CNN layer output is calculated as $inputSize - (kernelSize - 1)$, which means that if kernel size is bigger that 1, output would be smaller than input. To deal with this, zero-padding of size $\frac{kernelSize-1}{2}$ is introduced.

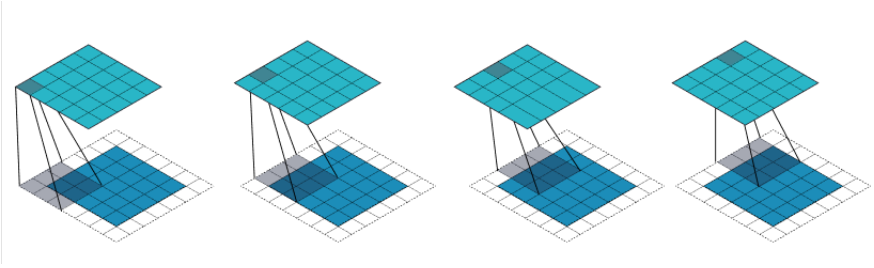


FIGURE 2.4: Example of convolution steps with $kernelSize = 3$ and $padding = 1$ [source image]

2.3.2 Batch normalization

Batch normalization [16] is a method to improve stability and performance of ANN. To increase the stability of a neural network, batch normalization normalizes the output of a previous layer by subtracting the batch mean and dividing by the batch standard deviation resulting in re-scaled data with zero mean value and standard deviation of one, which results in decreased dependencies on previous layers. It also helps to reduce sensitivity of initial weights and allows to use higher learning rates (it is a hyperparameter which controls how much the model will be changed in response to estimated error on each weights update iteration) - and get higher overall learning speed. Too small learning rate means that learning process will take longer, while choosing it too large may result in non-optimal weight updates.

2.4 sRGB

RGB color model is the one which represents color gamut as a combination of tree main chromaticities of red, green, and blue, which are added together in different amounts to produce any other chromaticity.

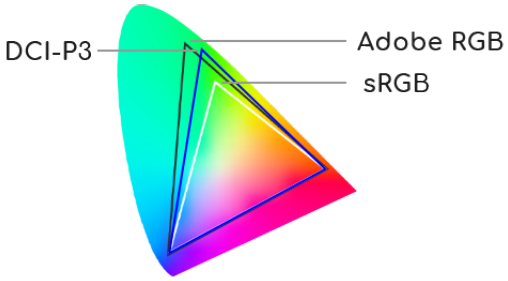


FIGURE 2.5: sRGB, AdobeRGB and DCI-P3 color spectre representation

sRGB stands for standard Red Green Blue – it is an standardized RGB color space created by HP and Microsoft in 1996 to use on consumer CRT monitors, printers, and over the internet [1] and it still the most popular color standard. Having common standard for all devices is to be convenient that the colors of the same image would look the always same, however common measure for modern display is not only sRGB coverage percentage (which represents how many of possible colors in sRGB color space the screen is capable of showing), but also AdobeRGB and DCI-P3, which are newer and both have wider color gamut. DCI-P3 standard was created by Digital Cinema Initiatives and it is meant to replace sRGB for smartphones, cameras and monitors.

2.5 Partial least squares regression

PLS regression is a method of finding fundamental relations between two matrices (A and B), that are also known as explanatory and response variables and predicting B from A. It is commonly used in cases where number of variables to be estimated is larger than number of known data points. Unlike regular least squares regression, PLS can deal with situations when sample size is small or the data has noise or missing values. It is considered to provide more stable model that does not change in case of added data than multiple linear regression and principle component analysis regression methods. PLS regression searches for components (also called latent vectors) that perform simultaneous decomposition of A and B, which maximize covariance between A and B, thats why alternative and more correct according to S. Wold term for PLS is projection to latent structures.

2.6 Color detection

As simple as it is, the process of color detection is the process of detecting the name of any color. But while humans brain learns to map color names and color values since childhood under different lighting conditions, computer can only classify colors by the values, set by human. But if the colors are set under certain light, and the test image was taken in other conditions, there is a possibility that color values have shifted towards for example red if it is sunset time or blue if it is evening. And while in most cases we can still classify them correctly, because we can make parallels between colors of object that we see, and color in which we are used to see this object, and adjust out color perception, computer usually does not know, which values are

"correct" and how to compare them to the other ones to get the right ones. Here, the colors which were set as a reference ones are being classified as the right ones.

It means, that the easiest way for computer to be able to correctly classify colors is to have constant lighting conditions, but that's not always how it happens in real life. In real life, especially outside, colors are altering during the day, and some preprocessing should be done to an input to provide the best possible similarity under any condition. This can be achieved by either color calibration of the input image itself, or, if it is an option, by calibration the camera itself.

Chapter 3

Related works

3.1 HardNet descriptor

Recently there were a lot of efforts to replace hand crafted descriptors such as SIFT [20], SURF [3] and detectors with learned ones: LIFT [31], DeepCompare [25], DeepMatching [27], however it was proven that different SIFT-based implementations (RootSIFT-PCA[7], DSP-SIFT [12]) still outperform learned descriptors in image matching, and the reason for this is lack of diversity in datasets which are used for training which reduces applicability of such descriptors.

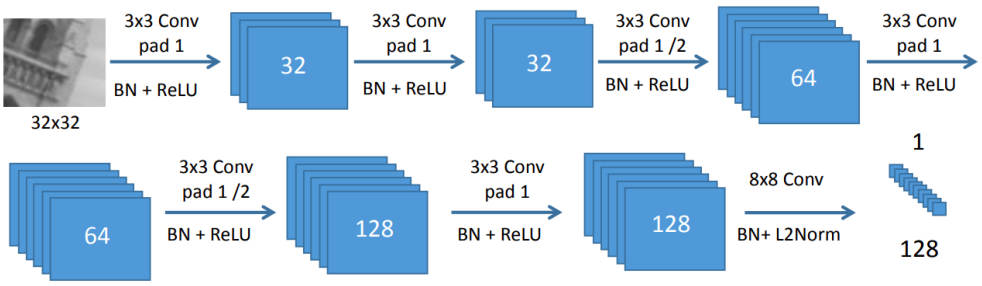


FIGURE 3.1: HardNet model architecture, /2 stands for stride 2

Hardnet [23] is a convolutional network for local image description, based on modified L2NET [30] architecture: zero-padding with batch normalization and ReLU non-lirearity are applied to all convolutinal layeys except the last one. Before the last layer there is dropout regularization with 0.1 rate. Compared to traditional SIFT, it uses triplets to calculate margin loss and distance loss, instead of pairs. The output is 128-D descriptor. For model training, UBC Phototour Brown [6] dataset of 400000 64×64 patches with test test of 100000 patches was used with DoG keypoint detector.

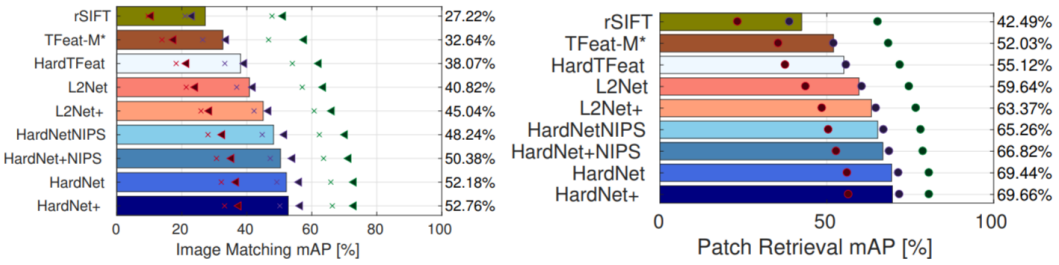


FIGURE 3.2: Results of benchmark on HPatches, mAP

Performance evaluation was made on HPatches dataset [2], which consists of 59 sequences of 6 images with viewpoint change and 57 sequences of 6 images with illumination changes, and shows that HardNet outperforms both hand-crafted and learned descriptors.

3.2 AffNet

The main idea behind AffNet [24] local feature affine shape estimator is that maximizing geometric repeatability does not lead to reliability in feature matching, and this method should improve the quality of the affine shape estimation. In their research they propose new a loss function and a method for learning affine shape.

Architecture for AffNet is similar to one in HardNet, but with reduced by 2 times number of channels in layer, and the output is a 3-dimensional predictor of ellipse shape, and dropout layer before last layer was increased to 0.25. Zero-padding, and batch normalization followed by ReLU are also present.

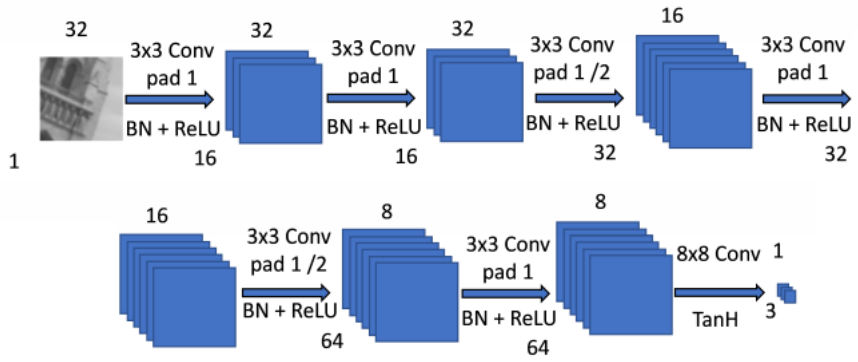


FIGURE 3.3: AffdNet model architecture, /2 stands for stride 2

For training a batch of matching patches pairs was generated from UBC Phototour dataset and then each patch was warped with randomly generated skew and rotation matrix and cropped to the size of 32×32 and fed into descriptor network (HardNet, SIFT).

3.3 DEGENSAC

DEGENSAC[10] is a RANSAC-based algorithm for estimation transformation between corresponding points. It is based on theorem, proven in the following paper, which states that if e.g. in seven point sample there are at least five inliers lying on a dominant plane that are related by homography, there always exists an epipolar geometry consistent with all homography-related points. Recent study by Yin et.al. [17] shows that this variation outperforms regular RANSAC and is capable of showing state-of-the-art results. Performance comparison is shown on Figure 3.4

3.4 TPS-3D

TPS-3D is an approach proposed by Menesatti et.al. [22] for reconstruction of the colors in sRGB colorspace. Main idea behind this method was to improve available

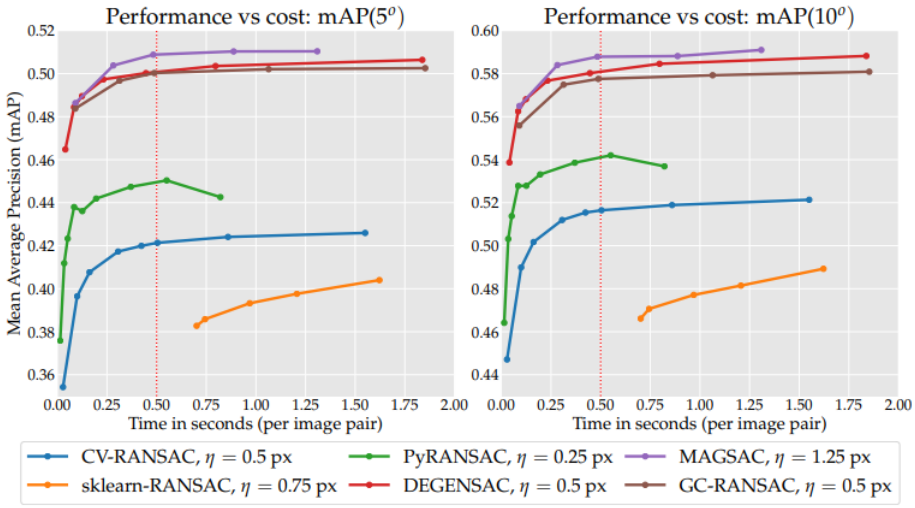


FIGURE 3.4: Performance comparison between implementations of RANSAC and DEGENSAC written in Python [source code]

methods of color correction, which did not have enough precision to applicable in fields with high demand on color accuracy, such as food sciences and biologial disciplines. It uses common data-fitment method, thin-plate spline, which was modified to work in a 3-dimensional space of RGB. The TPS algorithm estimates the random data from two paring sets of data to construct the spline map for the linear distortion and weighting factor for the non-linear distortion.

	Color Checker	Distances from reference mean ± SE	Inter-distances
NONE	7	29.66 ± 2.55	-
	24	29.86 ± 2.58	-
	140	30.64 ± 3.66	-
PROM	7	-	-
	24	30.81 ± 3.94	46.28 ± 6.99
	140	30.61 ± 4.00	42.92 ± 6.88
PLS	7	16.89 ± 0.72	11.76 ± 0.34
	24	15.42 ± 0.63	11.34 ± 0.33
	140	15.73 ± 0.58	8.9 ± 0.24
TPS-3D	7	14.3 ± 0.37	9.52 ± 0.28
	24	8.71 ± 0,44	9.78 ± 0.28
	140	8.16 ± 0.45	9.36 ± 0.25

FIGURE 3.5: Performance comparison between TPS-3D, PROM and PLS methods

On Figure 3.5 are shown test results and comparison between several popular color estimation techniques, where NONE stands for non-corrected images, PROM repsresents ProfileMaker Pro 5.0 commercial calibration system, and PLS stands for partial least squares method of estimation. Color checker column represents which color checker was used for estimation. Test pictures were taken using 2 different cameras with different sensors and lenses under four different light conditions. Three different color checkers were used: the GretagMacbeth ColorChecker SG with 140 color-patches, the GretagMacbeth ColorChecker with 24 color-patches and the IFRAO Standard ColorChecker with 7 color-patches.

Chapter 4

Datasets

For this work a custom dataset was created containing two sets of images with a specific object on each photo. All the photos were taken with Apple iPhone 7 Plus camera and saved as JPG images of size 956×1276 in RGB colorspace.

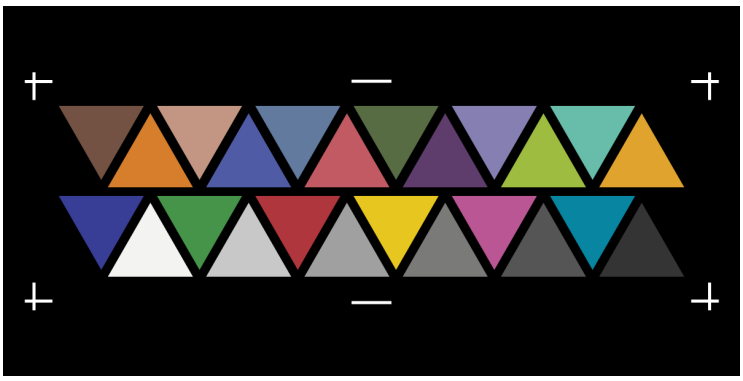


FIGURE 4.1: Reference image for the first set of data

Figure 4.1 represents reference image used for first test set, which contains 132 photos taken under different perspective, scale and light conditions, shown on fig. 4.2. Figure 4.1 was made based on the GretagMacbeth ColorChecker 24, and uses the same patch color values. It was observed that such pattern shows better results during detection process and increased percentage of correct recognition.

FIGURE 4.2: Sample images from dataset containing image from fig. 4.1.

FIGURE 4.3: Reference image for the second set of data

The second test set was created using an object containing different bright colors (Figure 4.3) to show that for the proposed method to work there is not need to have special color checker. 31 test images were taken under different light conditions, including extreme ones, as showed on Figure 4.4, where the exposure setting are set to very high or very low and image taken in black&white mode.

FIGURE 4.4: Sample images from dataset containing image from fig. 4.3

Chapter 5

Proposed approach

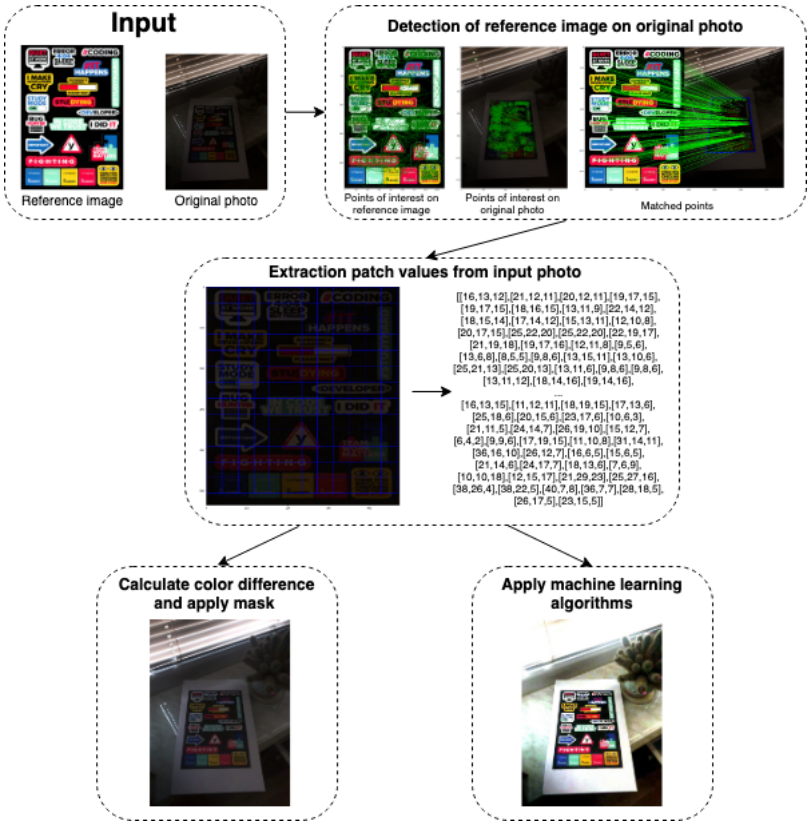


FIGURE 5.1

The high-level architecture of our approach is shown in Figure 5.1. The process of correcting colors on image contains of three steps:

- Detection: both reference and uploaded photo, that should be corrected, are fed into detection algorithm which consists of HardNet descriptor (section 3.1) and AffNet (section 3.2) which is used as a detector. After finding points of interest, tentative correspondences are found and the homography is calculated using DEGENSAC (section 3.3) algorithm.

- Patch color extraction: after the reference image was found on input photo, it is cropped out, warped using homography matrix and split into square patches. The mean value of each patches color is extracted. To get the values of the same patches from reference image, it should be resized to the size of cropped image, and split into patches of the same size.

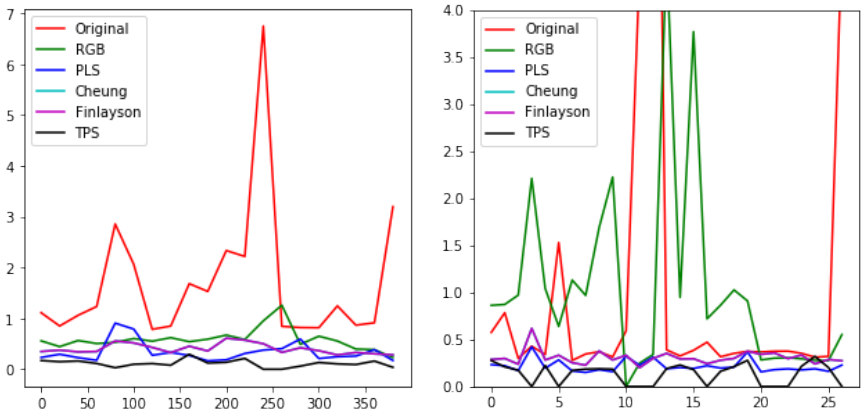
- Color correction. There are several ways how it can be accomplished:

    1. RGB mask: mean difference between output image and reference patches are calculated, and form "mask" values, which are then extracted from original image. This may be useful while the calibration of camera should be done right now and high precision is not important. Mask values are passed to camera and the output stream will already have the corrected colors, without any need of post-processing.
    2. PLS: uses partial squares regression (2.5) to calculate new values for each pixel by calculating transformation from output image patches to reference patches.
    3. TPS: uses 3D thin-plate spline approach (3.4). Overall the slowest one, but the most precise compared to other tested methods, however sometimes can fail completely or show some artifacts such as distorted colors in shadows and reflections.
    4. Root-Polynomial Regression by Finlayson et. al. [13]
    5. Polynomial techniques for color correction proposed by Cheung et. al. [9]

Chapter 6

Results

After testing different approaches to color correction, following results from the "stickers" dataset were achieved:



(A) Pairwise difference between images (B) Difference with reference image

FIGURE 6.1: Test results on stickers dataset

$$X = \frac{A - B}{B} \tag{6.1}$$

Difference was calculated using formula 6.1 between corresponding patches on test images. Figure 6.1a represents pairwise difference between images. After the calibration, deviation between images colors dropped significantly, and is not only much closer to zero than initial values, but also flatter. After the calibration, the deviation between images colors dropped significantly, and is not only much closer to zero than initial values but also flatter. Figure 6.1b represent deviation between the reference colors and the ones from calibrated images. Zero values on TPS result represent images that failed calibration. In both tests Cheung and Finlayson showed almost the same results.

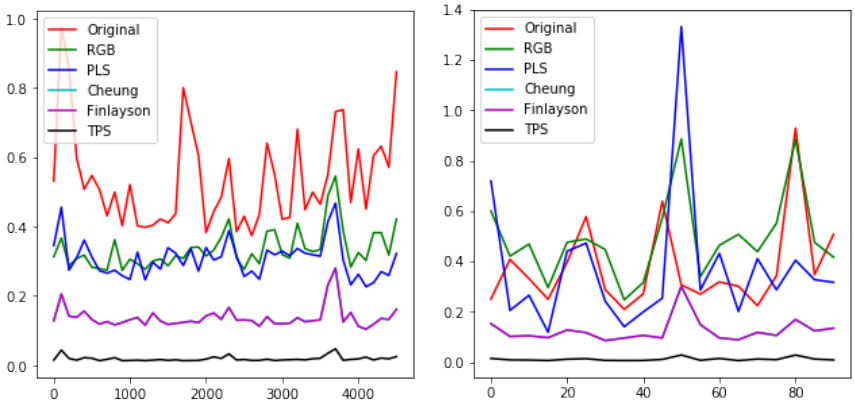
	Original	RGB	PLS	Cheung	Finlayson	TPS
Pairwise	1.69	0.58	0.34	0.4	0.4	0.11
Reference	1.14	1.05	0.22	0.31	0.31	0.13

TABLE 6.1: Calibration results table, showing mean difference between images

Table 6.1 shows mean difference values from experiments with different calibration methods on stickers dataset. It shows that TPS is, in fact, the most accurate

method to estimate colors. In terms of speed and accuracy combined, PLS produces better results, however, loses to Cheung/Finlayson in stability: there are several spikes on PLS line in Figure 6.1, while Cheung/Finlayson line remains flat most of the time. Mean TPS calibration value was calculated excluding cases, in which it fails to estimate values correctly, the other methods successfully finished with all test photos.

Figure 6.2 shows same plots for "triangle" dataset. Again, we can see how the variation between color values decreases after color correction.



(A) Pairwise difference between images (B) Difference with reference image

FIGURE 6.2: Test results on triangle dataset

Again, table 6.2 shows similar results as in the previous case. However, due to the fact that the reference calibration image had a lot less details, there were a lot more patches with one dominant colors, and the more such colors are, the easier for machine learning algorithms is to estimate colors correctly. But still, the final values are much closer to the reference ones, than they were before and the dominance of TPS method is obvious. However, this time Finlayson/Cheung showed better results compared to PLS.

	Original	RGB	PLS	Cheung	Finlayson	TPS
Pairwise	0.54	0.33	0.30	0.14	0.14	0.02
Reference	0.38	0.49	0.37	0.12	0.12	0.01

TABLE 6.2: Calibration results table, showing mean difference between images

To sum up, step-by-step algorithm would look like this:

- Select object with bright colors on it (ideally, with a flat non-reflective surface)
- Take a photo of selected object and save it as a reference one.
- Select whether you want to calibrate particular image or camera output stream.
- In case of single image calibration, TPS provides the best result of all tested methods.
- In case of real-time video calibration, in most cases the most stable and effective correction can be achieved by using either Finlayson or Cheung techniques.

6.1 Visual representation

In Figure 6.3 some examples of what the output image might look like are presented. The first row represents the original non-corrected images. The following are representing RGB mask corrected photo, PLS, TPS, and Finlayson/Cheung processed images, respectively. From our observations, TPS methods fail to correct the image if the overall colors on the image are dark. If the reference image has light reflections on it (Figure 6.3, fourth row, third photo), TPS results in distorted colors or being unable to process the image at all (Figure 6.3, fourth row, first photo from left). It also tends to overexpose the image (Figure 6.3, last column, almost none details in bottom left corner). Also the "cleaner" the patches are, the more chances it has to be able to calculate values correctly. Here, by "cleaner," it is meant that ideally, there would be only one color per patch. This might be the reason why the performance on triangle dataset is much higher. Nonetheless, despite being the most unstable and slow (average processing time of one 960 $\times$ 1280px image using TPS is 25 seconds on i5-4260u with 8GB RAM, while all the other took about 0.5s to execute). The speed is obviously dependent on implementation, in our case, it was Python NumPy implementation of Matlab code, provided in the original paper. There were no such limitations observed for other methods to work properly.

FIGURE 6.3: Calibration results

6.2 Additional observation

Unlike the other methods described in this thesis, TPS technique has one more ace up his sleeve, which both its advantage and disadvantage. Because it is performing pixel-wise correction, increasing variation of input values we also increase estimation precision. It is a disadvantage because it causes this method to run slower than the others, but if we can build any set of value pairs between reference and input image, what if we try to do this with black&white image?

FIGURE 6.4: Black&white photo restoration. Top left - original image, top middle - RGB calibrated, top right - PLS, bottom left - Finlayson, bottom middle - Cheung, bottom right - TPS

Figure 6.4 shows that, obviously, while RGB and PLS completely failed this test, TPS was able to partially restore colors from reference image, as well as from the hand: blue colors in the shadow, hand itself has rosy shades, and the rest is yellow (background is in fact wooden floor). Of course, the results are far from ideal, but still, it was not meant to deal with such tasks, and main colors and even shades are easily recognizable.

Chapter 7

Conclusions

In this study, different object detection techniques and color correction were studied and compared to each other. Based on numerous tests, the best ones were selected and described and defined which method best fits for which purposes. The custom dataset was created, proving that there is no need for specialized color checkers for the proposed approach to work. Basically, any object with bright colors can be used and produce good results in color calibration.

Bibliography

[1] *A Standard Default Color Space for the Internet - sRGB*. https://www.w3.org/Graphics/Color/sRGB.html.

[2] Vassileios Balntas et al. "HPatches: A benchmark and evaluation of handcrafted and learned local descriptors". In: *CoRR* abs/1704.05939 (2017). arXiv: 1704.05939. URL: http://arxiv.org/abs/1704.05939.

[3] Herbert Bay, Tinne Tuytelaars, and Luc Van Gool. "SURF: Speeded Up Robust Features". In: *Computer Vision – ECCV 2006*. Ed. by Aleš Leonardis, Horst Bischof, and Axel Pinz. Berlin, Heidelberg: Springer Berlin Heidelberg, 2006, pp. 404–417. ISBN: 978-3-540-33833-8.

[4] F.M. Bianchi et al. *Recurrent Neural Networks for Short-Term Load Forecasting: An Overview and Comparative Analysis*. SpringerBriefs in Computer Science. Springer International Publishing, 2017. ISBN: 9783319703381. URL: https://books.google.com.ua/books?id=wu09DwAAQBAJ.

[5] Léon Bottou. "Stochastic Gradient Descent Tricks". In: *Neural Networks: Tricks of the Trade: Second Edition*. Ed. by Grégoire Montavon, Geneviève B. Orr, and Klaus-Robert Müller. Berlin, Heidelberg: Springer Berlin Heidelberg, 2012, pp. 421–436. ISBN: 978-3-642-35289-8. DOI: 10.1007/978-3-642-35289-8_25. URL: https://doi.org/10.1007/978-3-642-35289-8_25.

[6] Matthew Brown and David G. Lowe. "Automatic Panoramic Image Stitching using Invariant Features". In: *International Journal of Computer Vision* 74.1 (2007), pp. 59–73. ISSN: 1573-1405. DOI: 10.1007/s11263-006-0002-3. URL: https://doi.org/10.1007/s11263-006-0002-3.

[7] Andrei Bursuc, Giorgos Tolias, and Hervé Jégou. "Kernel Local Descriptors with Implicit Rotation Matching". In: *Proceedings of the 5th ACM on International Conference on Multimedia Retrieval*. ICMR '15. Shanghai, China: Association for Computing Machinery, 2015, 595–598. ISBN: 9781450332743. DOI: 10.1145/2671188.2749379. URL: https://doi.org/10.1145/2671188.2749379.

[8] Yunpeng Chen et al. "Dual Path Networks". In: *CoRR* abs/1707.01629 (2017). arXiv: 1707.01629. URL: http://arxiv.org/abs/1707.01629.

[9] Vien Cheung et al. "A comparative study of the characterisation of colour cameras by means of neural networks and polynomial transforms". In: *Coloration Technology* 120 (2004), pp. 19–25.

[10] O. Chum, T. Werner, and J. Matas. "Two-view geometry estimation unaffected by a dominant plane". In: *2005 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'05)*. Vol. 1. 2005, 772–779 vol. 1.

[11] N. Dalal and B. Triggs. "Histograms of oriented gradients for human detection". In: *2005 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'05)*. Vol. 1. 2005, 886–893 vol. 1.

[12] Jingming Dong and Stefano Soatto. "Domain-Size Pooling in Local Descriptors: DSP-SIFT". In: *CoRR* abs/1412.8556 (2014). arXiv: 1412.8556. URL: http://arxiv.org/abs/1412.8556.

[13] G. D. Finlayson, M. Mackiewicz, and A. Hurlbert. "Color Correction Using Root-Polynomial Regression". In: *IEEE Transactions on Image Processing* 24.5 (2015), pp. 1460–1470.

[14] Jie Hu, Li Shen, and Gang Sun. "Squeeze-and-Excitation Networks". In: *CoRR* abs/1709.01507 (2017). arXiv: 1709.01507. URL: http://arxiv.org/abs/1709.01507.

[15] Fabian Hutmacher. "Why Is There So Much More Research on Vision Than on Any Other Sensory Modality?" In: *Frontiers in Psychology* 10 (2019), p. 2246. ISSN: 1664-1078. DOI: 10.3389/fpsyg.2019.02246. URL: https://www.frontiersin.org/article/10.3389/fpsyg.2019.02246.

[16] Sergey Ioffe and Christian Szegedy. "Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift". In: *CoRR* abs/1502.03167 (2015). arXiv: 1502.03167. URL: http://arxiv.org/abs/1502.03167.

[17] Yuhe Jin et al. *Image Matching across Wide Baselines: From Paper to Practice*. 2020. arXiv: 2003.01587 [cs.CV].

[18] Kristin Koch et al. "Report How Much the Eye Tells the Brain". In: (2006).

[19] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E. Hinton. "ImageNet Classification with Deep Convolutional Neural Networks". In: *Commun. ACM* 60.6 (May 2017), 84–90. ISSN: 0001-0782. DOI: 10.1145/3065386. URL: https://doi.org/10.1145/3065386.

[20] D. G. Lowe. "Object recognition from local scale-invariant features". In: *Proceedings of the Seventh IEEE International Conference on Computer Vision*. Vol. 2. 1999, 1150–1157 vol.2.

[21] M.A.S. McMenamin. *Dynamic Paleontology: Using Quantification and Other Tools to Decipher the History of Life*. Springer Geology. Springer International Publishing, 2016. ISBN: 9783319227771. URL: https://books.google.com.ua/books?id=7wBkDAAAQBAJ.

[22] Paolo Menesatti et al. *RGB Color Calibration for Quantitative Image Analysis: The "3D Thin-Plate Spline " Warping Approach*. 2012.

[23] Anastasiya Mishchuk et al. "Working hard to know your neighbor's margins: Local descriptor learning loss". In: *CoRR* abs/1705.10872 (2017). arXiv: 1705.10872. URL: http://arxiv.org/abs/1705.10872.

[24] Dmytro Mishkin, Filip Radenovic, and Jiri Matas. "Learning Discriminative Affine Regions via Discriminability". In: *CoRR* abs/1711.06704 (2017). arXiv: 1711.06704. URL: http://arxiv.org/abs/1711.06704.

[25] S. Murugesan et al. "DeepCompare: Visual and Interactive Comparison of Deep Learning Model Performance". In: *IEEE Computer Graphics and Applications* 39.5 (2019), pp. 47–59.

[26] Vinod Nair and Geoffrey E. Hinton. "Rectified Linear Units Improve Restricted Boltzmann Machines". In: *Proceedings of the 27th International Conference on International Conference on Machine Learning*. ICML'10. Haifa, Israel: Omnipress, 2010, 807–814. ISBN: 9781605589077.

[27] Jérôme Revaud et al. "Deep Convolutional Matching". In: *CoRR* abs/1506.07656 (2015). arXiv: 1506.07656. URL: http://arxiv.org/abs/1506.07656.

[28] B.D. Ripley. *Pattern Recognition and Neural Networks*. Cambridge University Press, 2007. ISBN: 9780521717700. URL: https://books.google.com.ua/books?id=m12UR8QmLqoC.

[29] Olga Russakovsky et al. *ImageNet Large Scale Visual Recognition Challenge*. 2014. arXiv: 1409.0575 [cs.CV].

[30] Y. Tian, B. Fan, and F. Wu. "L2-Net: Deep Learning of Discriminative Patch Descriptor in Euclidean Space". In: *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 2017, pp. 6128–6136.

[31] Kwang Moo Yi et al. "LIFT: Learned Invariant Feature Transform". In: *CoRR* abs/1603.09114 (2016). arXiv: 1603.09114. URL: http://arxiv.org/abs/1603.09114.