QATAR UNIVERSITY

COLLEGE OF ENGINEERING

ILLUMINANT ESTIMATION BY DEEP LEARNING

BY

HASSAN H. MIQDAD

A Thesis Submitted to

the Faculty of the College of

Engineering

in Partial Fulfillment

of the Requirements

for the Degree of

Master of Science /Electrical Engineering

June 2018

COMMITTEE PAGE

The members of the Committee approve the Thesis of Hassan H. Miqdad defended on 08/05/2018.

[illegible]

____

Prof. Mustafa Serkan Kiranyaz
Thesis/Dissertation Supervisor

____

Prof. Abdesselam Bouzerdoum
Committee Member

____

Dr. Noor Ali S A Al-Maadeed
Committee Member

Approved:

____

Khalifa Al-Khalifa, Dean, College of Engineering

ABSTRACT

MIQDAD, HASSAN, H., Masters : June : 2018:, Masters of Science in Electrical Engineering

Title: Illuminant Estimation by Deep Learning

Supervisor of Thesis: Mustafa, Serkan, Kiranyaz.

Computational color constancy refers to the problem of estimating the color of the scene illumination in a color image, followed by color correction of the image through a white balancing process so that the colors of the image will be viewed as if the image was captured under a neutral white light source, and hence producing a plausible natural looking image. The illuminant estimation part is still a challenging task due to the ill-posed nature of the problem, and many methods have been proposed in the literature while each follows a certain approach in an attempt to improve the performance of the Auto-white balancing system for accurately estimating the illumination color for better image correction. These methods can typically be categorized into static-based and learning-based methods. Most of the proposed methods follow the learning-based approach because of its higher estimation accuracy compared to the former which relies on simple assumptions. While many of those learning-based methods show a satisfactory performance in general, they are built upon extracting handcrafted features which require a deep knowledge of the color image processing. More recent learning-based methods have shown higher improvements in illuminant estimation through using Deep Learning (DL) systems presented by the Convolutional Neural Networks (CNNs) that automatically learned to extract useful features from the given image dataset. In this thesis, we present a highly effective Deep Learning approach which treats the illuminant estimation problem as an illuminant classification task by learning a Convolutional Neural Network to classify input images belonging to certain pre-defined illuminant classes. Then, the output of the CNN which is in the form of class probabilities is used for computing the illuminant color estimate. Since training a deep CNN requires large number of training examples to avoid the “overfitting” problem, most of the recent CNN-based illuminant estimation methods attempted to overcome the limited number of images in the benchmark illuminant estimation dataset by sampling input images to multiple smaller patches as a way of data augmentation, but this can adversely affect the CNN training performance because some of these patches may not contain any semantic information and therefore, can be considered as noisy examples for the CNN that can lead to estimation ambiguity. However, in this thesis, we propose a novel approach for dataset augmentation through synthesizing images with different illuminations using the ground-truth illuminant color of other training images, which enhanced the performance of the CNN training compared to similar previous methods. Experimental results on the standard illuminant estimation benchmark dataset show that the proposed solution outperforms most of the previous illuminant estimation methods and show a competitive performance to the state-of-the-art methods.

DEDICATION

It is my genuine gratefulness and warmest regard that I dedicate this work to my beloved family and colleagues, to express my gratitude for their endless support.

ACKNOWLEDGMENTS

I would first like to express my gratitude and thanks to my supervisor Prof. Serkan Kiranyaz at the Department of Electrical Engineering at Qatar University for his endless support, guidance, enthusiasm, and motivation throughout the thesis study while his door was opened for me all the time.

I would like also to thank my colleagues at my work place for their help and support in sharing their technical knowledge related to the image processing field. Moreover, they kindly offered me a proper annual leave for the full-time writing of the report which helped me a lot in finishing the thesis on time.

Finally, I cannot forget to thank my family for providing me the full support and the quiet environment to be able to think, write, and apply my thoughts creatively. This accomplishment would not happen without their support.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

CHAPTER 1: INTRODUCTION

The area of Artificial Intelligence (AI) and machine learning has recently gained much attention and popularity in the field of computer vision due to the remarkable success and breakthrough performance introduced by the Deep Learning systems and more specifically Convolutional Neural Networks in the task of object classification [1], and many other high-level computer vision tasks which have been inspired by that success including object detection and tracking [2], semantic segmentation [3], and face recognition [4]. Besides computer vision, Deep Learning and CNNs are getting more involved and became state-of-the-art techniques in other research areas such as speech recognition [5], and machine translation [6].

Today, the intensive number of research efforts on the implementation of CNNs in various applications have been initiated after the rapid development of high-end powerful computers and processors introduced by the latest Graphical Processing Units (GPUs). The need for such sophisticated processors has become a must due to the computationally intensive complicated structures of Deep Learning models including CNNs which are getting more complex in terms of increased depth and number of parameters [7]. According to NVIDIA Corporation, a pioneer trademark for the design and manufacturing of computer graphics, the high speed and energy efficiency features provided by GPUs compared to conventional CPU-based platforms made GPUs state-of-the-art in training deep neural networks [8].

Computer vision problems can typically be categorized into high-level and low-level problems. High-level vision problems include but not limited to object classification, object detection and tracking, semantic segmentation, and face recognition. Low-level vision problems are mostly related to the image processing domain include image de-noising and de-blurring [9], image super-resolution [10], image enhancement [11], and the well-known 3A's image processing functions in digital camera image processing pipeline which are: Auto-Focus, Auto-Exposure, and Auto-White Balance [12].

The problem of Auto-White Balance in digital photography is commonly known in the literature as "computational color constancy", which is still a longstanding problem and an active research topic due to its ill-posed nature [13]. In this thesis, we propose a highly effective approach to achieve an utmost accuracy for white balancing in digital images. In the next sub-section, we shall detail this problem in depth whilst discussing the state-of-the-art approach.

1.1 Background

Color constancy is a feature of the human vision system (HVS) which enables humans to account for the color of the illumination source, and hence it allows to perceive true colors of objects in a scene unchanged to different light sources and varying illumination conditions. Then, the objective is to emulate the ability of HVS in color images through the computational color constancy which refers to the process of estimating the color of the scene illumination in a color image, followed by color correction through some sort of transformation which uses the illuminant estimate information so that colors of the image will be viewed as if the image was taken under a reference illuminant (e.g. canonical white light source) [14], [15]. An illustration of correct and incorrect white balanced image is depicted in Figure 1. Despite its apparent simplicity, it is still a challenging task yet for computers due to the ill-posed nature of the illuminant estimation problem which arises from the fact that the observed color in an image is a function of the intrinsic properties of the object surface (surface reflectance), and the illuminant color (spectral distribution of the illuminant) where both quantities are unknown. Therefore, it is an under-constrained problem [13].

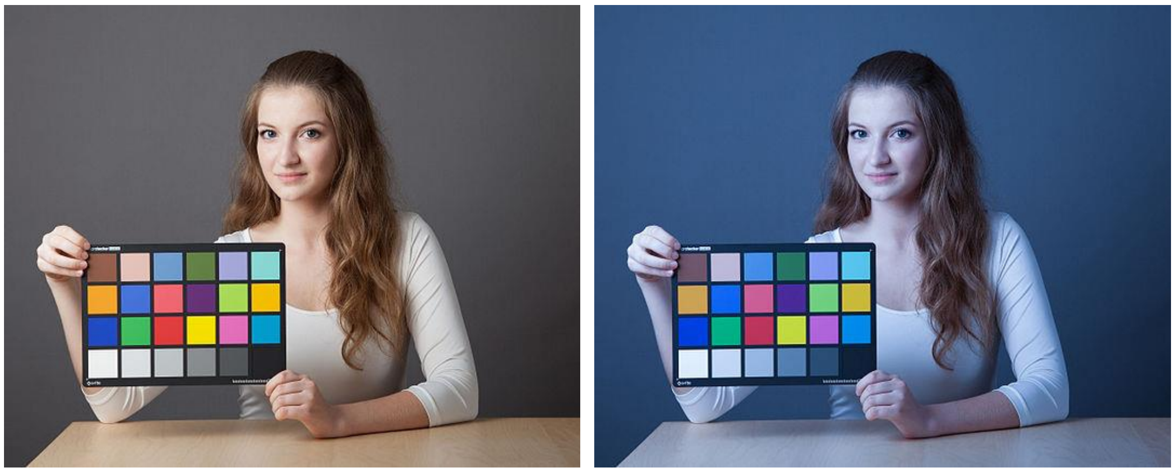


Figure 1. Correct white balance (neutral) vs. incorrect white balance (bluish) [16]

Computational color constancy is considered a fundamental concern in the camera industry (both still image and video) that requires the production of a plausible natural looking images/video frames without user intervention (i.e. automatically estimating and removing the illuminant color casts) [13], [19]. The so-called "Automatic White

Balance" process in digital cameras is crucial, especially in mobile phones cameras as the manual mode of selecting the right preset for white balancing in professional cameras requires a user experience in photography and color image processing. Besides, setting the camera to manual white balancing in long outside live events of TV broadcasting requires the attention of the cameraman or the end-user to varying illumination conditions which affect the aesthetic look of the picture. In addition, color constancy is a necessary pre-processing step for various computer vision applications such as semantic segmentation, texture classification, and visual recognition where color is an important feature.

The computational color constancy problem has yielded a great deal of research, especially for the illuminant estimation part, and many methods have been proposed in the literature. The vast majority of these methods assume a uniform illumination across the scene, and they mainly fall into two groups: static-based methods and learning-based methods [13]. The former group of methods do not need any training of data and can apply directly to images (i.e. fixed/static parameters setting), while the later estimate the illuminant based on a model that needs to be trained on some dataset. The learning-based technique has become more prevalent in literature and most of the recent works in the illuminant estimation follow this approach due to its higher accuracy in general compared to the static-based approaches [17]. Most learning-based methods are based on some handcrafted features (i.e. for a given dataset, the extracted features are manually designed), which require a domain expertise in the image processing and data representation [14], [17]. Hence, recent state-of-the-art works have been motivated toward using the power of CNNs in learning illuminant estimation models [14], [17]. The main advantage behind the remarkable success of CNNs in a variety of applications is the automatic learning capability to extract useful and complex features from the given dataset for a specific task. The majority of the recent CNN-based methods treat the illuminant estimation as a regression problem [17], [18], [19], except for one work which approached the problem as an illuminant classification [14]. Besides, most of these methods are patch-based input (i.e. each input image of the dataset is sampled to multiple patches) to overcome the limited number of images available in computational color constancy benchmark datasets. Such patch-based approach can be thought as a data augmentation, which is a well-known issue and usually a required process in training deep CNN models. However, this may offer many challenges to the task and deteriorate the overall performance due to the ambiguity presented in some local estimates of patches where semantic information is nearly absent.

1.2 Thesis Objective

The key objective set for this thesis is to achieve a high performance and accurate illuminant estimation for the computational color constancy problem by performing the following steps:

1) Study and review the different proposed approaches in the literature, and more specifically investigate the gaps and limitations presented in the CNN-based illuminant estimation methods.
2) Exploit the proven capabilities of CNNs in the task of object classification, and propose an optimized CNN classification-based illuminant estimation solution.

3) Propose a new approach for dataset augmentation which shall enhance the performance of CNN training. In fact, the size and representation of the dataset is very crucial to the performance of the CNN.

1.3 Thesis Scope

The scope of this thesis is to tackle the illuminant estimation part of the computational color constancy problem which is the most crucial part of the task. This thesis works on the common assumption of single illuminant within a scene and illumination variations occur only from scene to scene. Besides, the proposed CNN solution is camera-specific (i.e. different models will be trained separately for different camera-specific benchmark datasets) where sensor variations among different camera models are out of the scope of this thesis.

1.4 Thesis Outline

This thesis started firstly with an introductory chapter (Chapter 1) which aims to present the motivations of utilizing recent state-of-the-art DL systems in the field of computer vision. Besides, it states the problem of computational color constancy and recent approaches to solve it. The rest of the thesis is organized as follows. In Chapter 2, a formulation of the illuminant estimation problem of computational color constancy is described, together with a literature review of classical and state-of-the-art approaches. In Chapter 3, a background on the Artificial Neural Network (ANN) and CNN is explained. In Chapter 4, a detailed illustration of the proposed CNN solution, datasets, and the training strategy is presented. In Chapter 5, the experimental results on benchmark datasets based on some performance metrics are assessed and discussed. Finally, Chapter 6 concludes the thesis and draws some remarks on the limitations of this work and future directions.

CHAPTER 2: PROBLEM FORMULATION AND LITERATURE REVIEW

2.1 Problem Formulation

The image values $\boldsymbol{\rho}(x,y) = (\rho_R(x,y), \rho_G(x,y), \rho_B(x,y))^T$ under the common assumption of a Lambertian surface, which reflects light equally among all directions [13], can be expressed as a function of the light source (illuminant spectral power distribution) $E(x,y,\lambda)$, the surface spectral reflectance $S(x,y,\lambda)$, and the camera sensors spectral sensitivities $\boldsymbol{C}(\lambda) = (C_R(\lambda), C_G(\lambda), C_B(\lambda))^T$:

$$\boldsymbol{\rho}(x,y) = \int_{\omega} E(x,y,\lambda)\ S(x,y,\lambda)\boldsymbol{C}(\lambda)\, d\lambda \qquad (1)$$

where λ is the light wavelength, ω is the visible spectrum, and (x,y) corresponds to the pixel spatial location. Under the further assumption of a single uniform illumination across the scene, the observed color of the illuminant $\boldsymbol{\rho}^E = (\rho_R^E, \rho_G^E, \rho_B^E)^T$ in the image (i.e. the projection of the illuminant spectral power distribution on the camera sensors spectral sensitivities) can be expressed as:

$$\boldsymbol{\rho}^E = \int_{\omega} E(\lambda)\ \boldsymbol{C}(\lambda)\, d\lambda \qquad (2)$$

Then, the aim of the computational color constancy is to estimate this quantity (i.e. the scene illuminant chromaticity) to correct for the colors in an image. However, since the only known information is the camera sensors spectral sensitivities, $\boldsymbol{C}(\lambda)$ and the observed image values, $\boldsymbol{\rho}(x,y)$, illuminant estimation is an under-constrained problem, and hence it needs further assumptions to solve it. Therefore, various illuminant estimation methods have been proposed in the literature, each of which is based on different simplifying assumptions. In the next sub-section, a review of the different approaches and methods for the illuminant estimation will be presented.

2.2 Literature Review

As mentioned earlier, illuminant estimation methods can typically be categorized into static-based and learning-based methods. The former involves methods that do not need any supervision or tuning and can be applied directly to images (i.e. fixed/static parameters setting), while the latter involves methods that estimate the illuminant based on a model that is learned by a supervised manner.

2.2.1 Static-Based Methods

Static-based methods estimate the scene illuminant by making assumptions about the nature of the color images in the aim of exploiting statistical or physical properties of the scene, so they can be further distinguished to methods based on low-level statistics and methods based on the physics-based dichromatic reflection model [13].

The most common and well-known assumption of the low-level statistics based methods is made by the Gray-World method [20], which states that under a neutral illumination, the average reflectance in a scene is achromatic (i.e. gray), and hence the color of the illuminant can be estimated as the shift or deviation from gray of the averages in the color channels of the image. The main drawback of this method is that it may fail to estimate the illuminant color when the captured scene is dominated by large uniformly colored surfaces such as walls. White-Patch is another well-known method belonging to the first group of static-based methods [21], where it is assumed that the maximum values in the RGB color channels of an image are caused by a perfectly reflecting surface in the scene, hence the name white patch. Therefore, the illuminant color estimation in practice using this assumption is simplified by computing the maximum value in the separate color channels. Besides Gray-World and White-Patch methods which make use of the distribution of colors in an image (i.e. pixel values) to build their assumptions, Gray-Edge method [22] utilizes higher order statistics, namely image derivatives to build their assumption. Hence, instead of the average reflectance, Gray-Edge method assumes that the average color of edges or gradient of edges is gray, then the color of the illuminant is estimated as the offset of the averages of the edges (or gradient of edges) in the color channels from gray. Van de Weijer et al. [22] proposed a unified formulation which incorporates different low-level statistics-based methods into a single framework. Indeed, the aforementioned methods and other extended versions of them have been shown to be instantiations of this formulation:

$$\boldsymbol{\rho}^E(n,p,\sigma) = \frac{1}{k}\left(\iint |\nabla^n \boldsymbol{\rho}_\sigma(x,y)|^p dxdy\right)^{\frac{1}{p}} \tag{3}$$

where n denotes the derivative order, p denotes the Minkowski-norm, k is constant to make $\boldsymbol{\rho}^E$ (color of the illuminant) has unit length, and $\boldsymbol{\rho}_\sigma(x,y) = \boldsymbol{\rho}(x,y) * g_\sigma(x,y)$ is the image convolution with a Gaussian filter with scale parameter σ. Table 1 illustrates the combination of the parameters setting (n,p,σ) which correspond to different methods.

Table 1: *Parameters Setting for Various Low-Level Statistics Based Methods [19]*

Method	n	p	σ
Gray-World [20]	0	1	0
White-Patch [21]	0	∞	0
Shades-of-Gray [23]	0	4	0
General Gray-World [22]	0	9	9
First order Gray-Edge [22]	1	1	6
Second order Gray-Edge [22]	2	1	1

The second line of research on static-based methods estimate the illuminant color by analyzing the content of the scene in an image to exploit the information about the physical interaction between the objects and the illuminant, and hence such methods are referred to as physics-based methods. While the vast amount of static-based methods restricts their assumptions to the Lambertian reflectance assumption of image formation model (1), the physics-based methods make use also of the more generalized image formation model that incorporates the dichromatic reflection model which considers the specular reflection component as well [13]. In [24], the dichromatic reflection model has been exploited, so that pixels of one surface in a scene is projected into the CIE chromaticity space. Then, it was found that the color of the illuminant can be recovered when the surface dichromatic line is intersected with the Planckian locus (plot of the CIE chromaticity coordinates of a black-body radiator). In addition, other methods of the same category attempt to exploit bright areas in the captured scene represented by specularity or highlights, to obtain a good estimation of the illuminant color [25], [26],

[27], [28]. For example, Drew et al. [26] showed that the best illuminant estimation can be found by computing the geometric mean of bright pixels (usually, specular highlights).

Recent advancement in building more useful statistical assumptions for improved computational color constancy has been made through the understanding and the emulation of the human vision system [29], [30]. However, these methods are still believed to be not fully understanding the complex nature of the human vision system and the built-in mechanism behind the color constancy process [14]. Therefore, the recent motivation toward adopting Deep Learning/CNN in solving the illuminant estimation problem is because of its complex structure that is inspired by the concept of the biological visual systems.

Despite the advantages of the static-based methods of simple implementation and fast computation of the illuminant color, the accuracy, in general, is very low compared to the learning-based methods and the overall performance is limited to the pre-defined assumptions that may not be satisfied in some cases. Such an example is mentioned earlier where it was argued that the Gray-World assumption can be violated in the case that the captured scene is dominated by large uniformly colored surfaces.

2.2.2 Learning-Based Methods

One line of research of learning-based methods seeks to estimate the illuminant through a combination of different illuminant estimation methods and then, learning a model which decides the best performing method or combination of methods for each input image based on exploiting certain scene characteristics trained prior on a dataset of images. Various methods following this approach have been proposed in the literature.

Some methods like in [31], [32] use either the output of all the involved methods or some of them in producing the final illuminant estimate. In [31], a combination of a statistics-based method and a physics-based method is introduced where each of them produces likelihood outputs for a predefined set of illuminants so that the final illuminant estimate is a contribution of the weighted average of both likelihood outputs. In [32], various combining techniques have been explored for different illuminant estimation methods, where the output of all or some of the investigated methods can contribute to the final illuminant estimate. For example, one examined technique is to use the average value of all estimates. Another explored technique is to compute the Euclidean distances between the illuminant estimates of all methods projected into the *rg*-chromaticity color space, then the average of the two closest illuminant estimates is used as a final illuminant estimation. In comparison to these methods, Gijsenji and Gevers [33] propose a different combining strategy where the natural image statistics are exploited to train a model which selects the most appropriate or the best performing illuminant estimation among the different existing methods for each input image. In this method, a maximum likelihood classifier based on a mixture of Gaussians is trained on the statistical features of natural images extracted by the Weibull parameterization which provides some sort of a weighting relation between the image content and the selection of the most proper illuminant estimation method. Rather than features extracted by Weibull parameterization, several other features are investigated in different frameworks to select the best performing illuminant estimation method or combination of methods. For example, in [34] the selection of the most appropriate illuminant estimation method among different given methods for each input image is decided by a decision forest which is trained based on features expressed as low-level visual properties of the image content. In addition to the exploitation of natural image statistics, the incorporation of the scene semantic information is found to be also useful in selecting the appropriate illuminant estimation method. In [35], a selection technique of the illuminant estimation method is proposed based on analyzing and classifying the semantic content of the image belonging to a specific known scene category. Another similar approach which exploits the scene semantic information is proposed by Bianco et al. [36] where images at first are classified into indoor and outdoor, as well as an "unsure" class that corresponds to images that the indoor-outdoor classifier is uncertain about. Then, various strategies to decide and tune the most appropriate method or combination of methods among the different low-level statistics-based illuminant estimation methods [22], are explored for each class. In comparison to [35], [36] which use the scene semantic information to merely classify input images to a certain scene category/class and apply different illuminant estimation methods based on the scene category, Van de Weijer et al. [37] propose to exploit high-level visual information by firstly modelling the image as a composition of semantic classes like road, grass, and sky, where each class can be described based on its color information, texture, and position in the image. After that, a set of illuminant estimates is computed for each input image using different illuminant estimation methods, and then selecting the estimate that produces the most likely semantic content of the image (i.e. plausible image such that sky tends to be bluish and grass tends to be greenish).

Another line of research of learning-based methods attempts to estimate the illuminant color directly by learning their own model based on exploiting various scene characteristics and features extracted from a data set of images. An earlier well-known kind of methods following this approach is the Gamut-based method which has been firstly proposed by Forsyth [38], where it is assumed that under one illuminant, only a limited number of colors can be perceived in real-world images. Hence, any differences in the colors of an image occur due to a deviation in the illuminant color. Following this assumption, the method works as described in the below steps:

1) Compute the canonical gamut (i.e. the limited number of perceived colors under the known canonical illuminant) which is learned from the training images by observing as much as possible the colors of objects under the canonical illuminant.
2) Estimate the illuminant color of a test image by firstly computing its gamut which is assumed to be represented by all the observed colors in the image, then select the most proper mapping among a group of feasible mappings (i.e. mappings which can transform the test image gamut to be within the canonical gamut) and apply it to the canonical illuminant to estimate the test image illuminant color.

The selection procedure of the proper gamut mapping is rather a point of discussion in the literature where for example in [38], the appropriate mapping is the diagonal matrix with the largest trace which produces the most colorful scene. However, Bernard [39] proposed to take the average or a weighted average over the feasible mappings instead of selecting a certain mapping as in [38], which improved the results. Many extensions and variations of the Gamut-based method have been studied in the literature either to improve results or to solve the problems associated with the failure of the diagonal model which may result in a null solution [40], [41], [42]. However, all of these extensions deal only with image intensities wherein a more recent framework [43], it has been shown that the use of image derivatives results in a better performance and also overcomes the issues related to the null solution of the diagonal model. Another group of direct learning-based methods uses chromaticity histograms as the key features to learn their model for illuminant estimation such as in the color by correlation work [44], and the machine learning frameworks [45], [46]. For example, in [45], [46] neural network and support vector regression are trained on large binarized chromaticity histograms that represent the training images as features, then the output is represented by the *rg*-chromaticity of the estimated illuminant. However, extending the chromaticity histograms to the full three-dimensional RGB histograms as in the Bayesian frameworks [47], [48] has been found to be more useful in providing more accurate illuminant estimation. In [49], Chakrabarti et al. have shown that rather than building models based on statistics of per-pixel colors, exploiting statistics that can be deduced from the spatial color structure in a color image is much effective and more informative for the illuminant color estimation. They made a comparison between the empirical histograms or distributions (over a dataset of images after white-balancing) of the red channel values of individual pixels and the output of a band-pass filter, and they observed that the histogram for the filter output present more informative structure (unimodal and symmetric) that can be represented by simple parametric models. From that key observation, they constructed their method by firstly defining a parametric statistical model for a white-balanced image through a decomposition of the image using a series of spatially de-correlating filters followed by modeling color statistics independently of each sub-band. Then, the model parameters are learned to fit to a training dataset of white-balanced images and use this learned model in a maximum likelihood framework to estimate the color of the illuminant for a test image. In the exemplar-based framework [50], the authors propose to exploit surfaces in the image as the key feature and the problem has been addressed by unsupervised learning of a proper model for each segmented surface in the training images where both texture and color features are utilized for building the surface models. Then, the model of each segmented surface in a test image is compared to all training surface models and the set of the nearest neighbor models to the test surface model is selected for the illuminant estimation of the test surface through a histogram matching process. The final illuminant estimate is computed by taking the mean or the median of the illuminant estimates over the test surfaces. In a recent work, Cheng et al. [51] propose to extract four simple color features from the image and train an ensemble of *K* pairs (*rg*-chromaticity) of regression trees for each feature. The training images are firstly sorted based on the ground truth *r*-chromaticity and grouped equally into *K* local overlapping groups. Then, the regression trees are trained such that each tree pair is computed from the training samples that are biased to the local region of that pair. The final estimate of the illuminant chromaticity is computed by finding cross-feature consensus deduced from the estimations of all the regression trees.

The key observation from all the surveyed methods is that the extracted features from the image dataset for building different models are manually designed (i.e. handcrafted features), which require a domain expertise in the color image processing and data representation. Therefore, most of the recent works have been motivated toward utilizing CNN for solving the illuminant estimation problem due to its automatic learning ability in extracting the useful features for the given task.

The first attempt to use CNN for the illuminant estimation was proposed by Bianco et al. [19] where the problem has been addressed by incorporating the automatic feature learning with regression to estimate the illuminant color. The adopted CNN architecture operates on small, local patches of spatial size of 32×32 as inputs in attempt to overcome the limited number of training images in the available benchmark dataset which is a necessary requirement for training CNNs. Besides, the network architecture is simple consisting of only one convolutional layer followed by one max-pooling layer for feature extraction and ends with a single fully-connected layer that eventually outputs three values representing the RGB values of the locally estimated illuminant. These local estimates from an image are then pooled to obtain a one global illuminant estimate. Although the proposed method shows a satisfactory performance compared to many previous methods, but it is still lower than two recent learning-based methods that are built upon handcrafted features. This is likely because of the relatively small size of the sub-sampled patches compared to the original image size where many of these patches may not carry any semantic information which can lead to estimation ambiguity. In addition, the shallower network utilized in the method is not deep enough to extract sufficient features for the illuminant estimation problem. In [18], Shi et al. improved the performance of the patch-based CNNs for the illuminant estimation by introducing a novel architecture which consists of two interacting CNNs of moderate size called "Hypotheses network" and "Selection network". The former is designed to produce two hypotheses for an illuminant estimation of an input patch in a two-branch structure, while the latter adaptively makes a decision or selection of one of the generated estimations by the Hypotheses network to be the final illuminant estimate for the input local patch. The global illuminant estimate for the full input image is produced by performing a median pooling on the local estimates. A more effective solution than [18], [19] is proposed by Hu et al. [17] at Microsoft research where a fully convolutional network architecture is integrated with a novel pooling layer, namely Confidence-weighted pooling layer in an end-to-end learning process. Unlike the patch-based CNN methods [18], [19], the proposed method by Microsoft takes the full image as input and the task of the Confidence-weighted pooling layer is to mask out the local patches within the image that lead to estimation ambiguity (e.g. textureless walls) and merge only the estimates from patches which are more informative (e.g. human faces) for estimating the global illuminant color of the image. In the work proposed by Barron [52], the problem of the illuminant estimation is reformulated as a 2D spatial localization task in a log-chrominance space, so the input image is transformed into various chroma histograms, for which convolutional filters are trained to discriminatively evaluate possible illuminant color estimations in the chroma plane. Although the method shows a competitive performance, it highly ignores the semantic context in the image as the spatial information is weakly encoded in these chroma histograms. It can be noticed that the majority of these CNN-based methods [17], [18], [19] treat the illuminant estimation problem as a regression problem, except for one work which approached the problem as an illuminant classification [14]. In the work proposed by Oh et al. [14], the images firstly are clustered into *K*-clusters based on the ground-truth *rg*-chromaticity and then, the CNN is trained using the training images with the new cluster labels to classify images into *K* illuminant clusters where the output is in the form of class probabilities.

The illuminant color estimate of a test image is computed by taking a weighted sum of the cluster centroids using the class probabilities of the CNN output. This method also operates on patches as input to overcome the shortage of training images and the adopted CNN architecture is so deep in terms of the number of learnable parameters which may worsen overfitting since the available dataset for this task is small.

In this thesis, we will follow the classification approach using CNN for solving the illuminant estimation problem and we will address all the possible issues in [14], especially the choice of the CNN architecture and the data augmentation strategy.

CHAPTER 3: MACHINE LEARNING BACKGROUND

3.1 Artificial Neural Networks

An Artificial Neural Network (ANN) is a Machine Learning based algorithm that has a massively parallel distributed structure composed of simple processing units: “artificial neurons”, and it has the natural tendency to store experiential knowledge so that it has the characteristic of learning ability [57], [58]. The idea behind the ANNs was inspired by the brain and biological neural systems that consist of a very large number of neurons connected together by synapses and operating in parallel [57], [58], [59], [60]. For example, the human nervous system carries up to almost 86 billion neurons connected together by approximately 10^{14}-10^{15} synapses. The dendrites of each biological neuron are receiving inputs signals, then the neuron generates output signals carried along its axon that eventually branches out and connects to other neuron’s dendrites through synapses. The information travels among neurons and gets influenced by these connecting synapses with varying degrees of strength from one neuron to another [57], [60].

The simplest structure of ANN is the Perceptron, which is the basic computational unit in ANNs as depicted in Figure 2. Similar to a biological neuron, it takes a number of input signals x_j, where each input is multiplied by a synaptic weight (strength) w_j to mimic the synapses influnce on the neuron’s dendrites. Then, it outputs the signal y computed by applying a non-linear activation function $f(\cdot)$ on a weighted sum of its inputs [57], [59], [61]:

$$y = f\left(\sum_{j=1}^{d} w_j x_j + w_0\right) \tag{4}$$

where w_0 can be defined as the associated synaptic weight of a bias unit $x_0 = +1$. The sigmoid function $f(x) = 1/(1 + e^{-x})$, and hyperbolic tangent function $f(x) = \tanh(x)$ are examples of traditionally used non-linear activation functions [57], [59], [61], while more recenctly used functions especially with CNNs will be discussed later in this chapter.

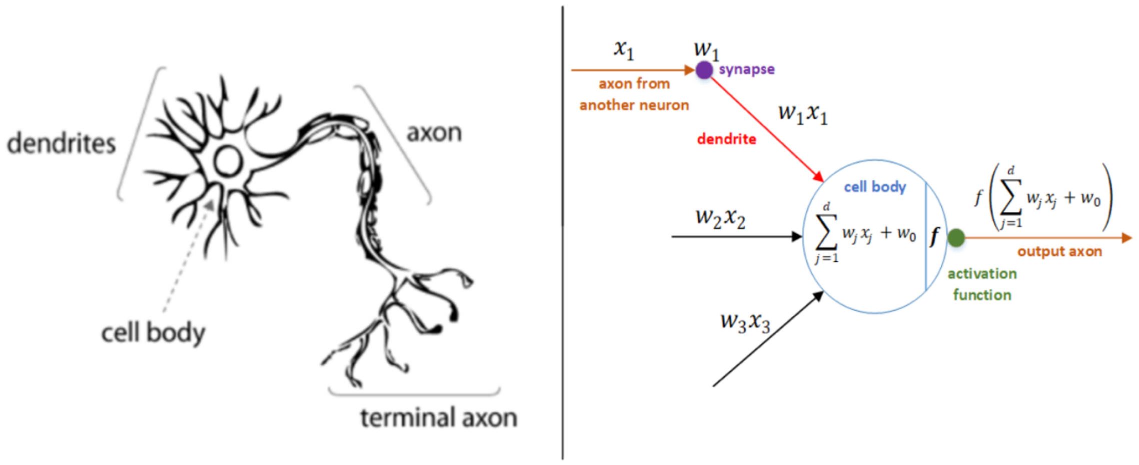


Figure 2. A biological neuron and a simulated model of an artificial neuron [57], [62]

3.1.1 Multi-Layer Perceptron

A Multi-Layer Perceptron (MLP) is the most common ANN architecture which is constructed by hooking together many of simple artificial neurons in a feed-forward and fully-connected layers format such that the output of each neuron in every layer is connected with certain weights w_{hj} to all neurons of the adjacent layer. Figure 3 illustrates an example of a three-hidden-layers MLP with four inputs in the input layer and six outputs in the output layer. In addition, each hidden layer has a certain number of hidden artificial neurons to form together a more complex fully-connected ANN structure useful for many tasks and applications [57], [59].

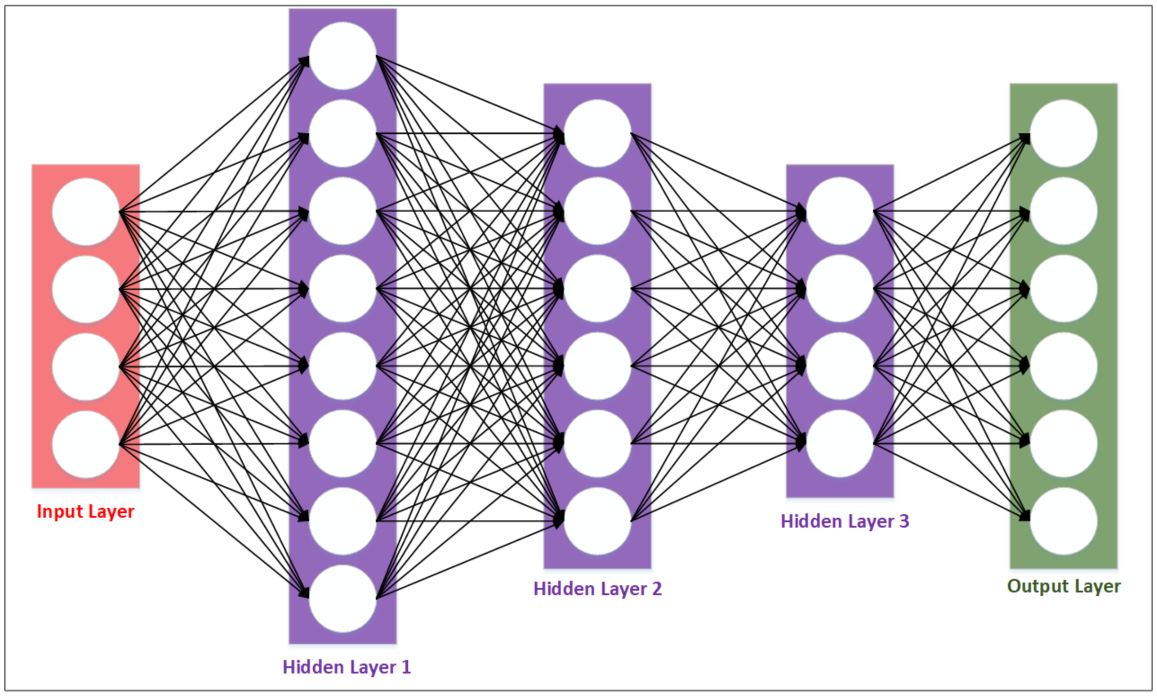


Figure 3. Three-hidden-layers MLP structure with 4 inputs and 6 outputs

3.1.2 Artificial Neural Network Training

The weights in an ANN are not set manually, but rather they are converged (learned) during a training process. The most common learning approach for training neural networks is the supervised learning scheme by providing a set of training examples, $\boldsymbol{x}$ with corresponding ground-truth labels, $\boldsymbol{y}$, and then the objective is to minimize some loss or error function, $\boldsymbol{E}$, which is used to measure how much the deviation of the predicted output $\hat{\boldsymbol{y}}$ by the ANN from the desired (ground-truth) output $\boldsymbol{y}$. The process of minimizing the loss function on the training data is carried by optimizing the network parameters or weights $\boldsymbol{W}$ during an optimization session. The most popular and widely adopted optimization method is the Gradient Descent algorithm. There are mainly two learning approaches for the Gradient Descent algorithm which are the online learning approach presented by the Stochastic Gradient Descent (SGD) method, and the batch learning approach using the Mini-Batch Gradient Descent (MGD) method [57], [59], [63]. For a set of training examples $\boldsymbol{x}$ with ground-truth labels $\boldsymbol{y}$, the SGD method works as follows:

SGD Optimization

Inputs: Normal distribution $N(\cdot)$ with mean (μ) and standard deviation (σ), Max number of epochs (n), Total number of training examples (m), Learning rate (η)

Output: Trained model

```
1:  Initialize all the network weights with small random values (e.g. from $N(\mu, \sigma)$)
2:  Repeat for epoch=1:n
3:    Repeat for $x$=1:m
4:      Compute the predicted output $\hat{y}$ through a forward pass to the neural network
5:      Compute the loss or error function $\boldsymbol{E}$ using $\hat{y}$ and $y$
6:      Compute the gradients of all weights by backpropagation of the output error to all
            the network parameters through a backward pass using the chain rule
7:      Update all the network weights by a specific amount in the negative direction of
            their corresponding gradients, according to the update rule $\Delta w_j^t = -\eta \frac{\partial E^t}{\partial w_j^t}$
8:    End
9:  End
```

Discussion: the amount of the weight update is determined by the so-called learning rate η which usually decreases when training progresses for convergence. Besides, it is considered to be in practice one of the most important hyper-parameters in training neural networks [57], [63]. For clarification, one complete pass over the whole training set is called an epoch, where the required number of epochs for a sufficient training depends on the complexity of the ANN structure and the dataset type. The MGD method works similar to the SGD method whereas the only difference is that updating the network parameters or weights is applied after computing and averaging the gradients over a mini-batch of training examples, which potentially results in a much faster training process. The main advantage of using the MGD method is its applicability to large-scale datasets (i.e. in order of hundreds of thousands or even higher). Then, it is quite efficient for much faster convergence to perform less network updates using averaged gradients computed over mini-batches, which in practice are considered to be good approximations of the gradients of the full loss function. In addition, this kind of mini-batch learning allows for much more efficient computations in practice which can be executed on platforms (e.g. GPUs) that support vectorized code optimizations [57]. Recently, the term “SGD” becomes more popular among machine learning practitioners when referring to MGD method.

One of the widely known issues about the SGD optimization method is that it converges slowly. Thus, a simple and popular technique which considerably improves the performance of SGD and helps accelerate gradients toward the right direction is to introduce a term called “momentum” to the SGD update rule [57], [59]:

$$\Delta w_j^t = -\eta \frac{\partial E^t}{\partial w_j} + \alpha \Delta w_j^{t-1} \tag{5}$$

where α (momentum parameter) is typically chosen in practice between 0.5 to 0.99 depending on the task. It can be noticed from (5) that the update rule is modified by incorporating the previous update in the current update which has the advantage of smoothing the error trajectory towards convergence by help damping the oscillations that may occur around some local optima as a result of using plain SGD updates [57], [59].

As discussed before, the learning rate η has the crucial impact on the whole training process and is considered the most important hyper-parameter to be carefully chosen in the SGD method and its momentum version. This hyper-parameter can be set to a fixed value or more commonly in practice is set into a schedule. This schedule usually attempts to decay the learning rate over time, and typical choices may include a Step decay, Exponential decay, and Linear decay schedules [57]. Recently, much research has been focused into proposing new optimized Gradient Descent techniques that aim to adaptively tune the learning rate during the training process. The most common algorithms in practice are the Adadelta [64], Adagrad [65], RMSprop [66], and ADAM [67].

3.2 Convolutional Neural Networks

A Convolutional Neural Network (CNN or ConvNet), is a special type of neural networks that has a special structure which is particularly well-suited for 2-D input data type such as color images [57], [63], [68]. CNNs are widely known for their remarkable achievements and the breakthrough performance in many computer vision tasks including object classification [1], object detection and tracking [2], and face recognition [4]. The motivation behind adopting CNNs for image-related tasks and the description of their special architecture will be discussed in the next sub-sections.

3.2.1 Motivation

CNNs are quite similar to regular ANNs on the basis that both are built from interconnected simple artificial neurons arranged in layers and have learnable parameters which are optimized during a training process to minimize some loss function placed after the last (fully-connected) layer of the network. However, they are mainly different in the way how their neurons are connected to other neurons between some certain layers, and this is necessary when utilizing neural networks for image-related tasks as regular

ANNs do not scale well to images of typical sizes. For example, suppose an RGB image with a typical resolution of 300×300 pixels is given as input to a regular MLP, then a single hidden neuron in the first fully-connected layer would have $300 \times 300 \times 3 = 270{,}000$ weights. This number of weights is only for a single neuron and for sure a typical MLP would contain many of such neurons in each layer, then the total number of learnable parameters would be infeasibly high which inevitably would lead to stagnation (i.e. inability to learn). In addition, the architecture of such MLPs does not take into consideration the spatial structure of the image, but rather it rearranges the 2-D structure into a 1-D vector and treats pixels that are spatially apart on the same footing as pixels that are spatially adjacent. Therefore, all these issues and drawbacks of using regular MLPs for image-related applications have led to the motivation for using a more efficient architecture design that takes advantage of the spatial structure of images, and hence introducing the Convolutional Neural Network architecture, which addresses all of the issues of regular MLPs through leveraging three main ideas: local receptive fields (limited connection), parameter sharing, and pooling [57], [63], [68].

3.2.2 Neurons Arrangement

Compared to a regular MLP, the hidden neurons in the convolutional layers of a CNN are arranged in 3-D volumetric patterns specified by width, height, and depth, which is more advantageous in dealing with input types like RGB images. For example, an input RGB image with a resolution of 300×300 can be expressed as an input volume with a width of 300, height of 300, and depth of 3 (i.e. the color channels). Thus, each layer in a CNN accepts a 3-D input volume and transforms it to a 3-D output volume [57]. A simple visualization of the concept of neurons arrangement in 3-D volumes is depicted in Figure 4.

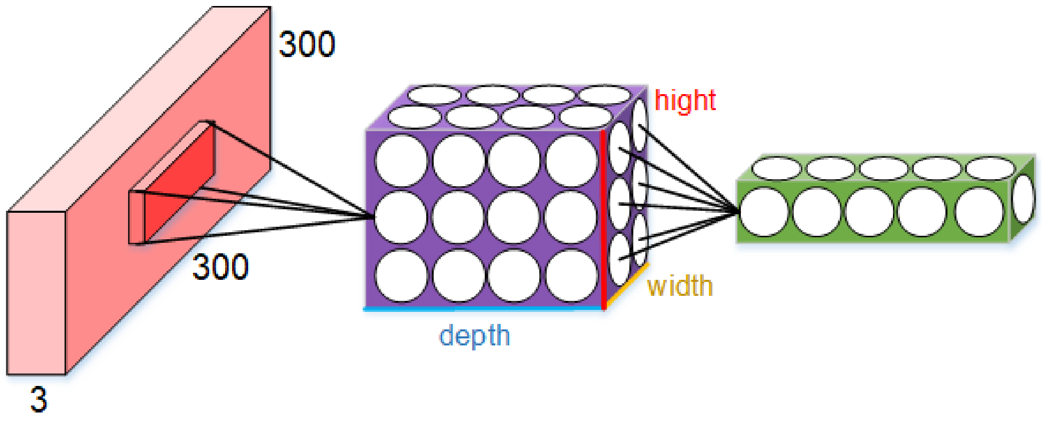


Figure 4. A simple visualization of the neurons arrangement in 3-D volumes in a CNN

3.2.3 CNN Layers

The architecture of a CNN typically consists of many layers of several types, which are stacked together in a certain order or format to transform the input volume (e.g. image pixel values) into an output volume which holds, for example, the class scores if the CNN would be used for an object classification task. There are several types of layers used for building different CNN architectures, but the most important layer types that almost exist in every CNN topology are the Convolutional layer (CONV), Pooling layer

(POOL), ReLU activation function layer, and Fully-Connected layer (FC) identical to the hidden layers of a regular MLP [57], [63].

3.2.3.1 Convolutional Layer

The convolutional layer is considered to be the core layer of a CNN which is responsible for extracting all the useful features from a given input data to accomplish a certain task, and it performs most of the computational heavy lifting in the CNN.

Limited connectivity. As discussed earlier, connecting every single artificial neuron in one layer to all other neurons in the previous layer is impractical when the input data is of large size like typical RGB images. Thus, each neuron in a convolutional layer is connected only to a local, small region of the input volume, where this region is called the "local receptive field" of the neuron in the CONV layer, but it should be noted that the connectivity of the neuron is always full across the depth dimension of the input volume [57], [63], [68]. Then for example, if an input RGB image has a resolution of $227 \times 227 \times 3$ and suppose that the receptive field $\boldsymbol{F}$ in the first CONV layer is defined with a spatial size of 3×3, then each hidden neuron will have a total number of only $3 \times 3 \times 3 = 27$ learnable weights (+1 bias), rather than $(227 \times 227 \times 3) + 1 = 154588$ parameters. The arrangement of connections between the CONV layer neurons and the input volume is identified through sliding the local receptive field by a certain amount across the whole input volume, where this amount is defined by a hyper-parameter called the stride, $\boldsymbol{S}$ [57], [63], [68]. Figure 5 illustrates a simple example using a small input image of size $27 \times 27 \times 3$, local receptive field of size 3×3, and stride of 2.

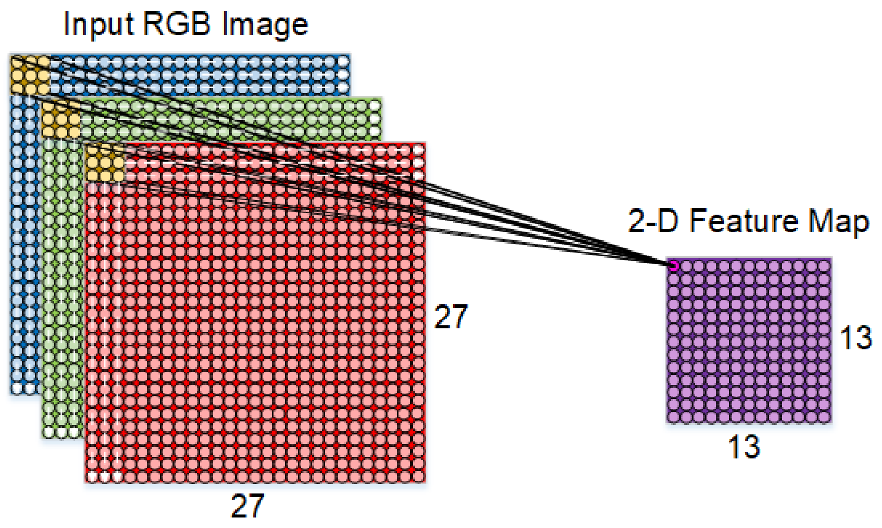


Figure 5. A visualization of the 2-D convolution between an input image and a 3× 3 filter with stride of 2

It can be noticed that as we slide the local receptive field across the entire input image, a 2-D activation map (also called feature map when the parameter sharing scheme is applied) is generated from computing the dot products between the connection weights and the input values under the regions defined by the local receptive fields [57], [63], [68]. As shown in Figure 5, the spatial size of the 2-D feature map is halved the input image because the local receptive field can only move 12 steps along the horizontal or vertical direction of the input image. The CONV layer typically generates a ***K*** number of 2-D activation maps (stacked together along the depth dimension of the output volume) for extracting different features from the given input data [57], [63], [68].

Parameter sharing. The number of parameters (i.e. learnable weights and biases) in the CONV layer can be significantly reduced by using the "parameter sharing" scheme which is built upon a reasonable assumption that if a certain feature is useful to compute or extract at some local receptive field, then it is likely to be also useful to extract at other local receptive fields in the input image. In other words, if the parameters associated with a certain neuron in the CONV layer are such that it can detect for example a certain feature such as a horizontal edge at the local receptive field of that neuron, then it should be also useful to detect the same feature at other local regions in the input image. This means that all neurons in a single 2-D activation map in the CONV layer are sharing exactly the same weights and bias, and that is why the activation map is called a feature map. Hence, it turns out that every 2-D feature map is generated by convolving the shared weights of that feature map with the input volume, biased by the shared bias value. So, this is the reason for the name Convolutional layer, and the shared weights are usually referred to as a filter (or alternatively a kernel) [57], [63], [68]. Parameter sharing scheme in practice is a big advantage in the CNNs which greatly reduces the memory requirements. For example, the input image in the AlexNet [1] has a resolution of $227\times227\times3$ and the output volume of the first CONV layer has a size of $55\times55\times96 = 290{,}400$ neurons results from using 96 filters each of size $11\times11\times3$ slide over the input image by a stride of 4. Then, if each neuron in the output volume would connect to the input image with unique parameters, this will lead to a total number of $290{,}400*(11\times11\times3+1) = 105{,}705{,}600$ parameters which is certainly a huge number. However, if the parameter sharing scheme is applied, then the required number will be $96*(11\times11\times3+1) = 34{,}944$ parameters, which is considered a significant reduction.

3.2.3.2 Non-linear Activation Function Layer

The non-linear activation function layer is placed directly after each convolutional layer to apply an element-wise non-linear activation function to each neuron in the convolutional layer. The most commonly used activation function for CNNS is the Rectified Linear Unit (ReLU) which has the mathematical form $f(x) = \max(0, x)$. Other types of activation functions are the conventional sigmoid function $f(x) = 1/(1 + e^{-x})$, and the hyperbolic tangent function $f(x) = \tanh(x)$, but these functions become rarely used for deep learning compared to ReLU function due to their computational complexity, slower convergence rate, and the gradient vanishing effect [57], [69]. Figure 6 illustrates a comparison between these functions.

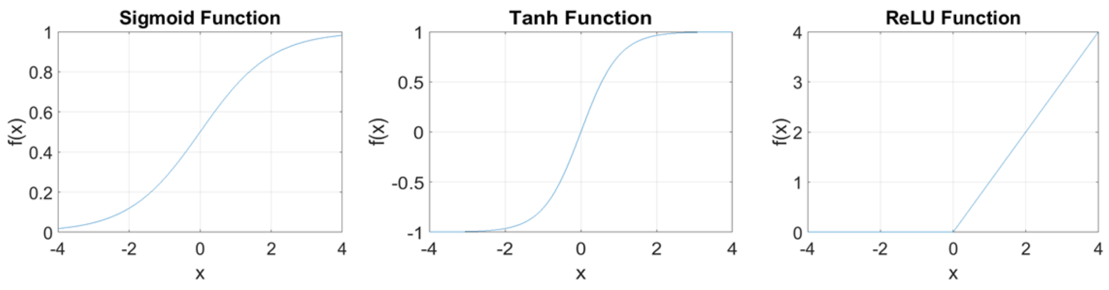


Figure 6. The non-linear activation functions: sigmoid, tanh, and ReLU

3.2.3.3 Pooling Layer

Pooling layers are usually inserted after some CONV layers in the CNN to reduce the spatial size of the 2-D feature maps output from the CONV layer by summarizing a spatial region of the feature map into one output value, which further helps to reduce the number of required parameters and computations in the subsequent layers in the network. The most common choices for pooling are *Max-pooling* and *Average-pooling*, both of which operate independently on each 2-D feature map to generate a downsampled version of them by computing Max or the Average operation over a local spatial region of a feature map [57], [63], [68]. In practice, pooling layers are commonly applied to small regions with the spatial extent of 2×2 or 3×3, and with a stride of 2 [57]. Besides, they can be also used to perform a *global pooling* which reduces the spatial size to 1×1 output, and it is typically used at later stages in some CNN topologies. Figure 7 illustrates a simple example of pooling operation on two feature maps of size 12×12, where pooling is applied on 2×2 regions with a stride of 2.

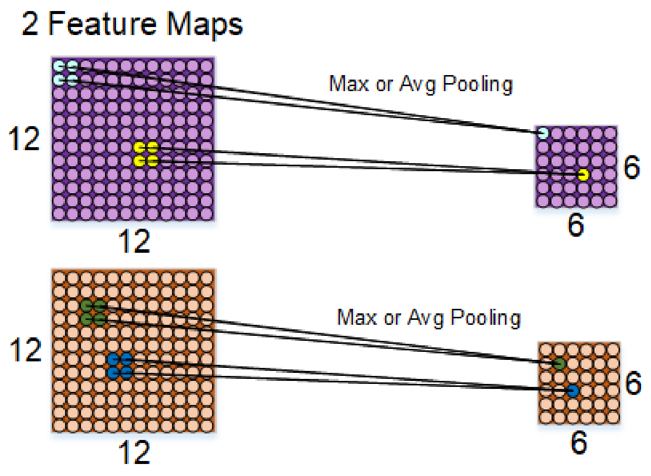


Figure 7. A simple visualization of the pooling operation applied on two feature maps

3.2.3.4 Fully-Connected Layers

The Fully-Connected (FC) layer used in CNNs is the same layer type used in regular MLPs. Therefore, all neurons in the FC layer have full connectivity to every neuron in the previous layer, and their activations are computed in the same manner as seen before in regular MLPs. The output volume of the FC layer has a spatial size of 1×1 and a depth identified by the number of the FC hidden neurons [57], [68]. FC layers are placed at later stages in the CNN for the computation of the class scores in classification tasks, but their use became less in more recent CNN topologies such as in [76], because they often account for most of the network parameters.

CHAPTER 4: THE PROPOSED SOLUTION

4.1 System Overview

The proposed solution for the illuminant estimation problem is similar in concept to that proposed in [14], but the main differences between them lie in the CNN architecture, the training strategy, and the data augmentation which significantly improved the results. Figure 8 illustrates the overview of the proposed system which consists of three main design steps. Firstly, images in a given dataset are clustered into *K*-clusters based on their associated ground-truth illuminant color (Figure 8-1). Then, the CNN is trained using the images in the train dataset with the new labels (i.e. the associated cluster labels) to classify images into their *K* illuminant clusters where the output of the CNN is in the form of *K* probabilities (Figure 8-2). Finally, the trained CNN is used to estimate the illuminant color of a test image by taking a weighted sum of the cluster centroids using the *K* probabilities of the CNN output (Figure 8-3). A detailed discussion of the implementation of each of these design steps will be presented in the next sub-sections.

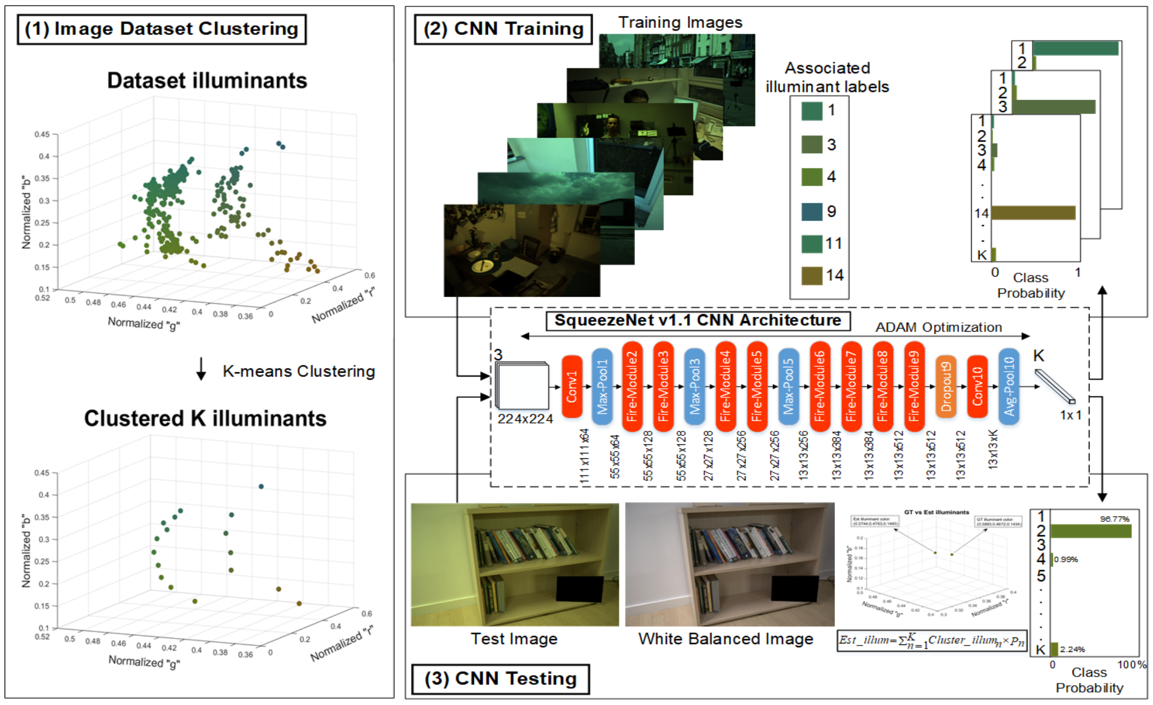


Figure 8. The system overview of the proposed CNN-based illuminant estimation

4.2 Dataset Illuminants Clustering

Clustering the illuminants is considered to be an important factor to use CNN in a proper way for the illuminant estimation problem. This is because of the similarity between many existing illuminants which would make the CNN discrimination of these illuminants quite a difficult task. For example, the benchmark Gehler-Shi dataset [47], [70] that consists of 568 images, contains several images captured under similar illumination conditions as shown in Figure 9. Consequently, clustering the image dataset of similar illuminations into well-separated classes would make the discrimination task of the CNN more efficient and easier.

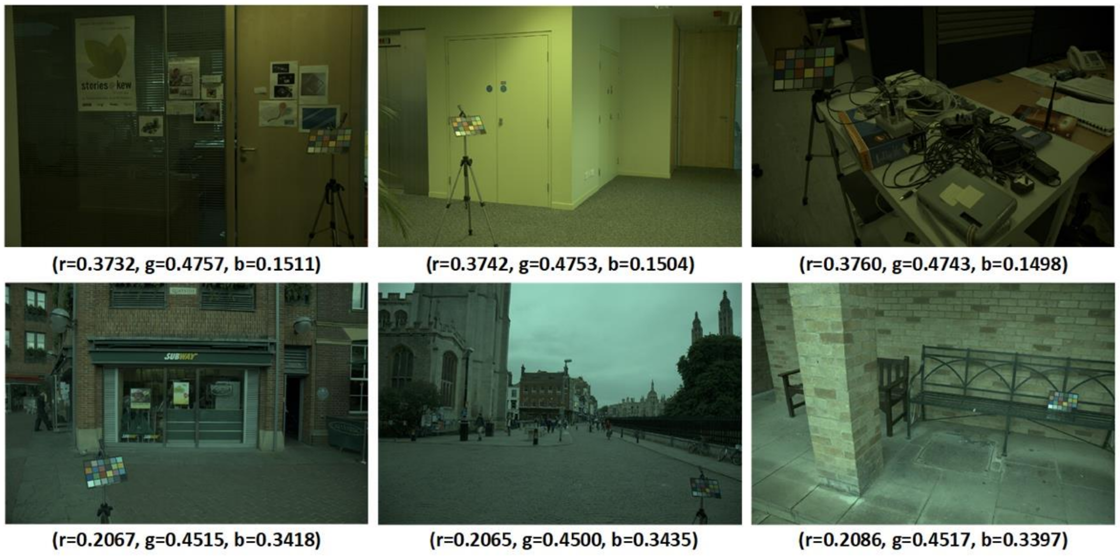

(r=0.3732, g=0.4757, b=0.1511)
(r=0.3742, g=0.4753, b=0.1504)
(r=0.3760, g=0.4743, b=0.1498)
(r=0.2067, g=0.4515, b=0.3418)
(r=0.2065, g=0.4500, b=0.3435)
(r=0.2086, g=0.4517, b=0.3397)

Figure 9. Sample images in the Gehler-Shi dataset [47], [70] that are captured under similar illumination conditions. The ground-truth illuminant color in normalized rgb is shown below each image. (Note: images are gamma-corrected for display purpose)

The utilized method for clustering the illuminants is the *K*-means clustering [71] with the squared Euclidean used as the distance measure, and the *K*-means++ algorithm [72] used for cluster center initialization. The determination of the optimal number of clusters *K* for a given dataset is often not an obvious task, and hence several techniques have been proposed such as the elbow method [73] and the Bayesian Inference Criterion (BIC) method [74] in attempt to provide the best choice of *K*. However, we followed the recommendations in a recently published work [75] which carried very comperhensive experiments and evaluations over different datasets using different fitness functions and clustering validity indices (CVIs) to provide conclusions about the best CVIs that both assess the quality of the clustering solution provided by the clustering method, and the optimal number of K. Hence, three of the best CVIs according to [75] are selected in this thesis which are the Xu, WB, and Calinski-Harabasz (CH) indices. The mathematical formulas of these CVIs are given below along with Table 2 which explains the notations used in these formulas [75].

$$Xu = Nlog\left(\sqrt{\frac{ssw}{Nn^2}}\right) + log(k) \tag{6}$$

$$WB = K\frac{ssw}{ssb} \tag{7}$$

$$CH = \frac{ssb}{ssw} \times \frac{n-K}{K-1} \tag{8}$$

Table 2: *Notations Used in CVI Formulas [75]*

Description	Notation and Formula
Data item	$\mathbf{x}$
Set of all items	C
Number of items	n
Total mean vector	$\mathbf{m} = \frac{1}{n}\sum_{\mathbf{x}\in C}\mathbf{x}$
Number of clusters	K
Data dimension	N
i^{th} cluster	C_i
Number of items in cluster C_i	n_i
Centroid of cluster C_i	$m_i = \frac{1}{n_i}\sum_{\mathbf{x}\in C_i}\mathbf{x}$
Sum-of-squares for cluster C_i	$ssw_i = \sum_{\mathbf{x}\in C_i}\lVert\mathbf{x} - \boldsymbol{m_i}\rVert^2$
Within-cluster sum-of-squares	$ssw = \sum_{i=1}^{K} ssw_i$
Between-cluster sum-of-squares	$ssb = \sum_{i=1}^{K} n_i \lVert\boldsymbol{m_i} - \boldsymbol{m}\rVert^2$

According to [75], Xu and WB indices should be minimized, while CH index should be maximized. Then, the conclusion is that the best data clustering solution among different possible solutions for a certain *K* (i.e. different cluster centroids for every *K*-means experiment of the same *K*) is the one that would maximize CH index or minimize Xu and WB indices. Besides, the optimal number of clusters *K* can be decided by the maximum value of CH index or the minimum values of Xu and WB indices. Since every run of a *K*-

means clustering results in different cluster centroids due to different initial cluster centroid positions each time, then we decided to conduct an intensive experiment for the generation of the *K*-means cluster centroids to avoid possibilities of having poorly cluster center initialization and/or less accurate clustering solution for a certain *K*. The flow of the experiment is as follows:

```
K-means Clustering Experiment
Inputs   : Max Number of Clusters K =100, Max Number of experiments E =1000,
           For each K-means run: Max Iterations =100, Number of replicates = 100
Outputs: 100 K-means solutions (cluster centroids, indices), Optimal number of K
1:   Repeat for K=2:100
2:     Repeat for E=1:1000
3:         Define K-means arguments Dataset, K, Max Iter = 100, Replicates = 100
4:         Compute Cluster centroids with the associated index of each data point, CVIs
5:         Save Results
6:     End
7:     Find A(K) = Min (Xu (1:1000)), B(K) = Min (WB (1:1000)), C(K) = Max (CH (1:1000))
8:     Return Values of the K-means clustering trial that satisfies the conditions in step (7)
9:   End
10:  Find Min (A (2:100)), Min (B(2:100)), Max (C(2:100))
11:  Return the Optimal number of clusters (K) based on the results of step (10)
```

Discussion: for clarification, the Max Iterations argument mentioned in step (3) of the *K*-means clustering experiment denotes the number of times that the *K*-means algorithm iterates to find the clustering solution, while the Replicates argument denotes the number of times that the *K*-means algorithm repeats clustering using new initial cluster centroid positions and returns the clustering solution with the lowest within-cluster sums of point-to-centroid distances. This is useful in practice to help to find a better solution with lower local minima or possibly a global minimum of the objective function. The *K*-means experiment has been conducted on the benchmark Gehler-Shi dataset [47], [70], where the full RGB information of each of the 568 illuminants (Figure 10) is used in the clustering process compared to [14], where the clustering of the illuminants has been only applied on the *rg*-chromaticity space.

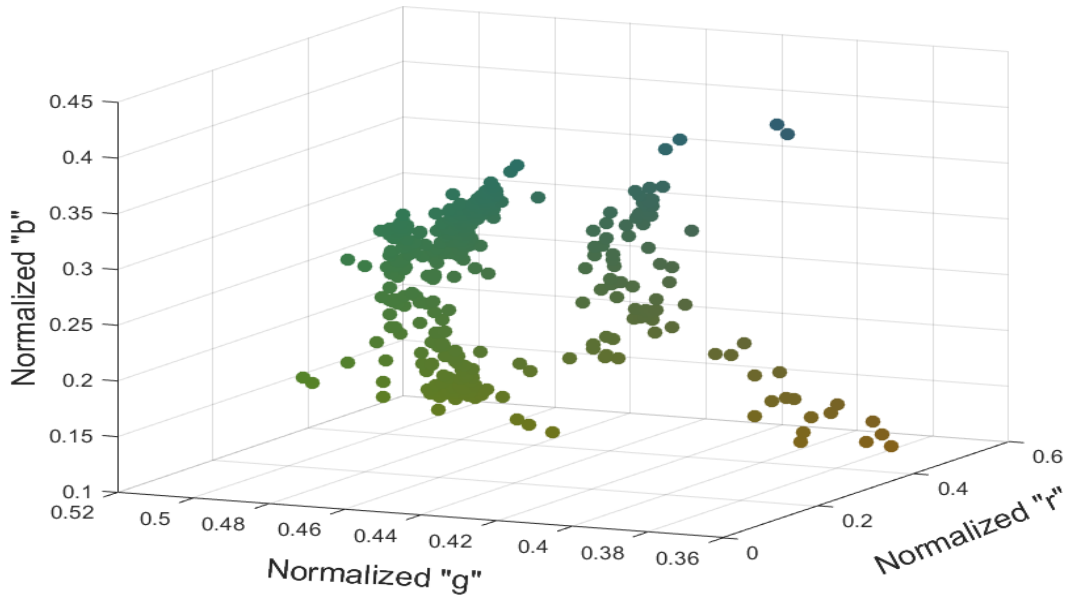


Figure 10. Normalized rgb values of the illuminants in the Gehler-Shi dataset [47], [70]

Then, the main observations of this experiment can be stated as follows:

1) All the three CVIs agreed on the same solution among the 1000 possible solutions for every value of *K*, which gives more confidence about the provided solution.
2) Regarding the determination of the optimal number of clusters *K*, none of the CVIs could give a clear or an explicit answer about the best choice of *K* as shown in Figure 11, Figure 12, and Figure 13, probably due to the biased distribution of the dataset which almost contains no clear or direct informative pattern for clustering. As evident from Figure 10, a huge majority of the images in the dataset have been collected in similar illumination conditions, while few numbers of other images were collected under different illumination conditions. Thus, some images are easy to group them in nearly unique clusters, while many others are left with hard decisions about the proper number of clusters that should belong to. However, one can deduce some useful hints by looking at the behavior of some of these CVIs, and more specifically the WB and Xu indices. It can be noticed from their plots against *K* that the significant amount of reduction in their values occurs almost in the interval between *K*=2 and *K*=50, while beyond that the reduction becomes insignificant as evident in the WB index in Figure 11. In addition, as discussed earlier that the more the clusters are further apart, the more efficient the CNN is to discriminate classes. Therefore, the number of clusters *K* has been empirically chosen to be 17 (Figure 14), so as to make a trade-off between the ease of CNN classification and accuracy in the computation of the estimated illuminant color from the cluster centroids.

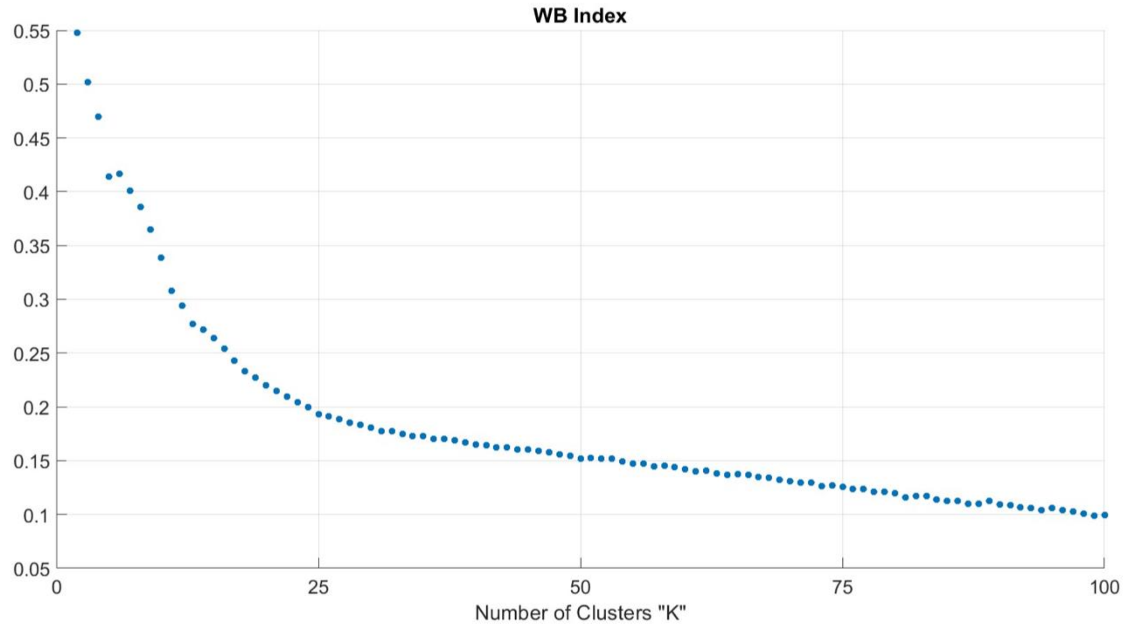


Figure 11. WB-index vs. number of clusters *K* for Gehler-Shi dataset [47], [70] clustering

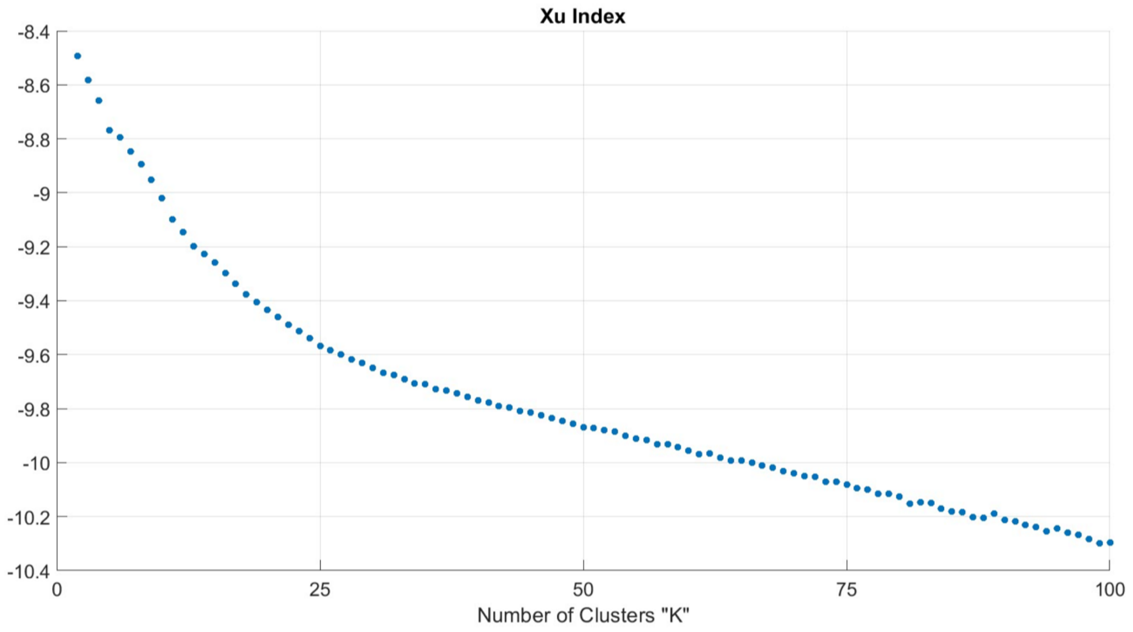


Figure 12. Xu-index vs. number of clusters *K* for Gehler-Shi dataset [47], [70] clustering

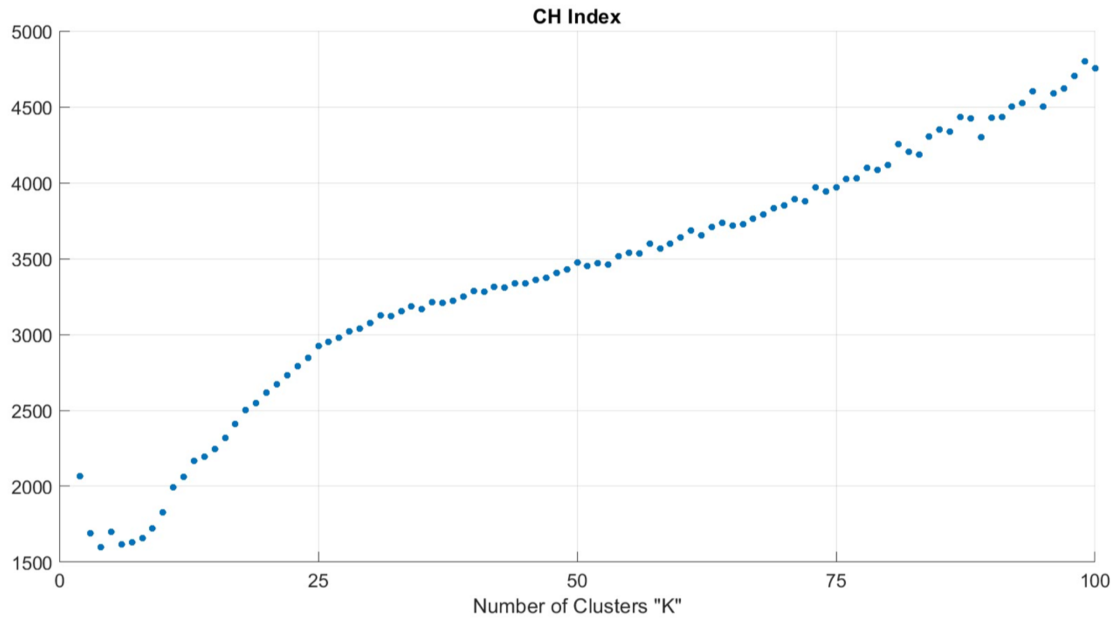


Figure 13. CH-index vs. number of clusters *K* for Gehler-Shi dataset [47], [70] clustering

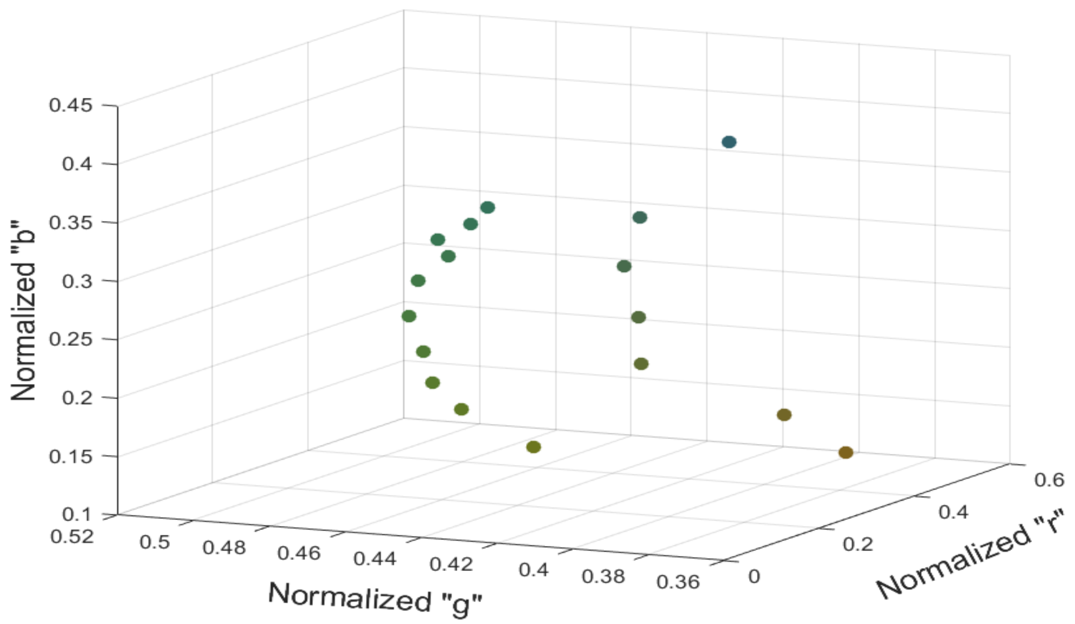


Figure 14. Normalized rgb values of the Gehler-Shi illuminants clustered into *K*=17

4.3 Convolutional Neural Network Architecture

The CNN architecture that has been adopted in this work is the SqueezeNet v1.1 proposed by Iandola et al. [76]. The main contribution of this architecture is that it could achieve the AlexNet [1] classification accuracy on the ImageNet dataset [77] with 50× less parameters (1,235,496 parameters), which offers many advantages of reducing the overfitting problem in the tasks with small datasets, fast training and inference suitable for real-time applications, and the feasibility of deploying small size models on FPGAs and limited memory devices. The authors could successfully achieve this objective by leveraging three main strategies for designing the CNN topology which are:

1) Reducing the overall network parameters by using many 1×1 filters and less 3×3 filters, as the former has 9× fewer parameters.
2) Reducing the overall network parameters by also decreasing the number of input channels or feature maps to the CONV layers that have the 3×3 filters by using "Squeeze layers".
3) Preserving or maintaining higher classification accuracy with the available fewer parameters by downsampling the feature maps at later stages in the network (i.e. setting a stride of 1 in most of the network layers, and stride > 1 toward the end of the network) so that most of the CONV layers in the network have larger feature maps, which is found to provide higher classification accuracy.

Hence, they introduced what is called the "Fire module" which is the basic building block of the SqueezeNet architecture, and it is composed of:

- Squeeze layer: a convolutional layer of $(s_{1\times1})$ number of 1×1 filters with a stride of 1.
- Expand layer: a concatenation of two convolutional layers (across the depth dimension) where the first one has $(e_{1\times1})$ number of 1×1 filters with a stride of 1, and the second has $(e_{3\times3})$ number of 3×3 filters with a stride of 1. The input of both sub-layers is taken from the output of the previous Squeeze layer. Besides, the number of filters used in the Squeeze layer is always less than the number of filters used in the sub-layers of the Expand layer, which helps to limit the number of input feature maps to the 3×3 filters. It should be noted that the input feature maps to the Expand sub-layer of 3×3 filters are 1-pixel zero-padded at the borders to have the same spatial size of the output feature maps of the Expand sub-layers.

An illustration of the high-level design of the SqueezeNet v1.1 architecture is depicted in Figure 15. The architecture starts with a convolutional layer (Conv1), followed by 8 Fire modules (Fire2-9), and ends with a convolutional layer (Conv10). Three Max-pooling layers with a stride of 2 are used after Conv1 layer, Fire module 3, and Fire module 5. The output of all the convolutional layers including the Fire modules is activated with the ReLU activation function. The number of filters in the Fire modules is gradually increasing across the network. A Dropout layer has been used after Fire9 module for regularization, and an Average global-pooling layer is applied at the end of the network which reduces the spatial size of the output feature maps from the last convolutional layer to 1×1 for holding the class scores. The number of filters in the Conv10 layer is adjusted to K=17 to match the number of the illuminant clusters in our own task. Finally, a

Softmax layer is used after the Average global-pooling layer which squashes the class scores into class probabilities that sum up to 1. The full architecture design and the specifications of the network layers are presented in Appendix A.1 and A.2.

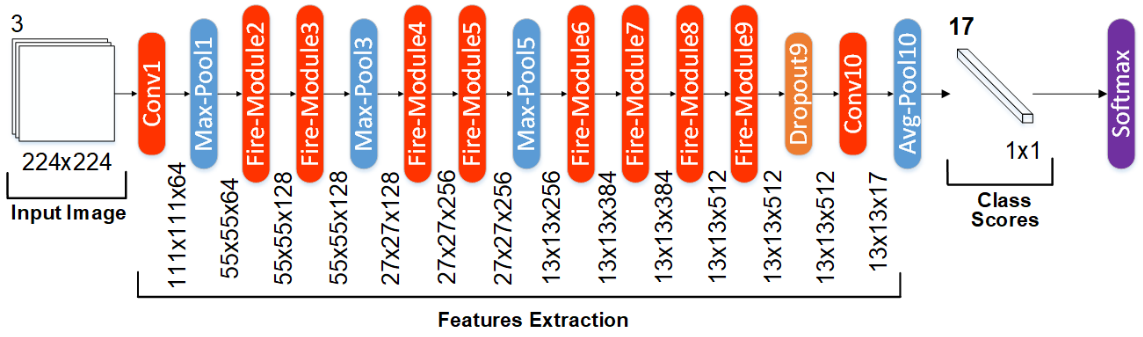


Figure 15. A high-level design of the SqueezeNet v1.1 architecture

4.4 Training Strategy

4.4.1 Image Dataset

The benchmark Gehler-Shi image dataset [47], [70] is used for the training and evaluation. This dataset consists of 568 images and comprises a variety of indoor and outdoor scenes, which was originally captured by Gehler et al. [47] using two high quality DSLR cameras of type Canon 1D and Canon 5D, and saved in both Camera RAW format free of any manipulation or correction, and TIFF image format. Shi and Funt [70] proposed to reprocess the RAW images again because the provided TIFF images were generated automatically and hence, are non-linear (i.e. gamma corrected), demosaiced, contain clipped pixels, and have the camera white balance effect. Therefore, the reprocessed version of the RAW images by Shi and Funt [70] are created in almost-raw 12-bit (preserved dynamic range) PNG image format, and they are linear images (i.e gamma=1) in camera RGB color space. In each image, a Macbeth Color Checker (MCC) is included to allow for an accurate measure of the ground-truth illuminant color. Figure 16 shows some examples of images with different illuminations in the Gehler-Shi dataset.

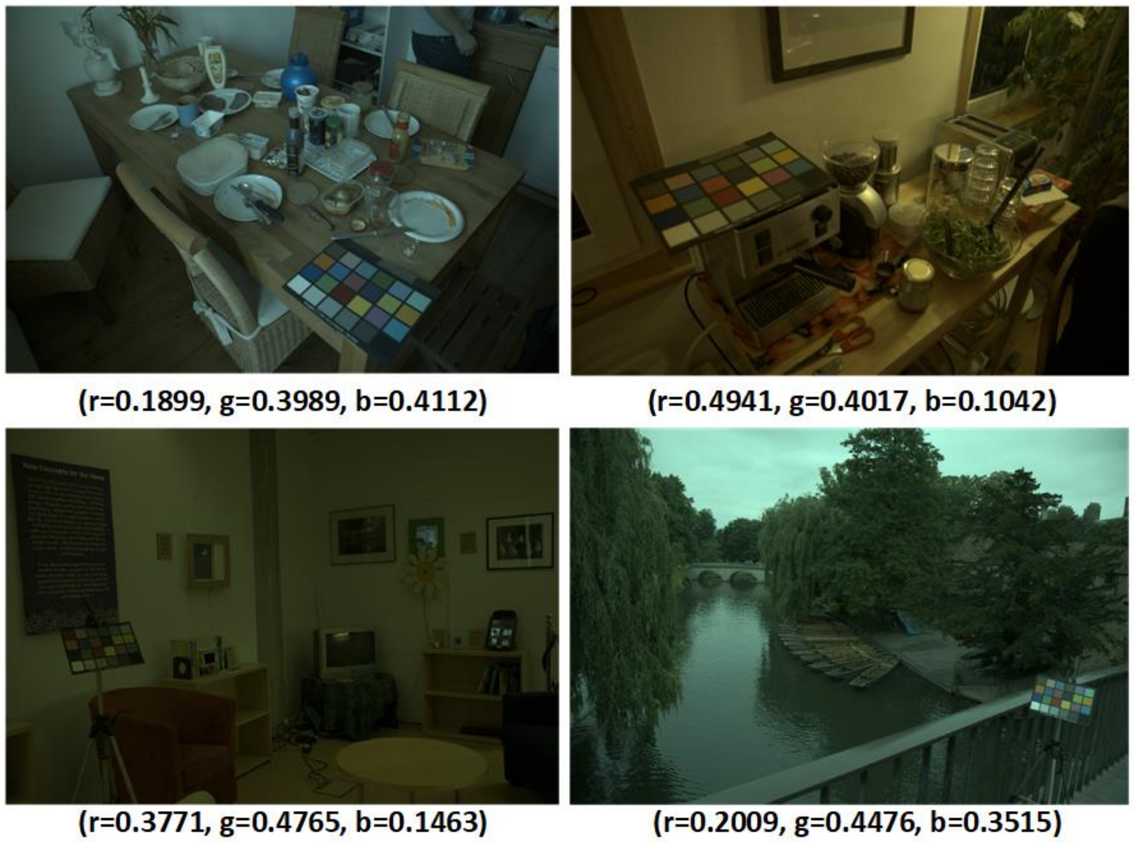

(r=0.1899, g=0.3989, b=0.4112)
(r=0.4941, g=0.4017, b=0.1042)
(r=0.3771, g=0.4765, b=0.1463)
(r=0.2009, g=0.4476, b=0.3515)

Figure 16. Examples of images with different illuminations in Gehler-Shi dataset [47], [70]

4.4.2 Data Preprocessing and Augmentation

Providing a well-prepared dataset is indeed a crucial step for training deep learning systems. Thus, data preprocessing and augmentation should be handled carefully before submitting to the CNN. As advised by [70], the camera's black level offset (0 for Canon 1D and 129 for Canon 5D) is subtracted from each image. In addition, the Macbeth Color Checker (MCC) in each image is masked out before training and testing, which is the common practice in the illuminant estimation literature. The spatial coordinates of the MCCs corners are measured and provided by the dataset [47]. A further preprocessing step is to mask out also the saturated pixels where the saturation is considered to happen at a threshold of 0.95 from the given saturation value (3692 for Canon 1D and 3563 for Canon 5D), so as to have a more conservative saturation masking. A good

practice in training deep neural networks is to normalize the input data by firstly computing the per-channel mean and standard deviation over the training set, then each image used for training and testing is subtracted from it the pre-computed per-channel mean followed by a division by the pre-computed per-channel standard deviation [78].

One of the well-known requirements for training deep CNN models is the size of the dataset, which is required to be as large as possible to be able to train the complex structures for extracting the generalized features, and preventing the model from overfitting the data which considerably deteriorates the generalization capacity of the trained model. In fact, this is the main concern for utilizing deep learning systems for the illuminant estimation task due to the relatively small size of the available benchmark illuminant estimation dataset. Therefore, this issue has been dealt with through data augmentation. In comparison to other methods [14], [18], [19], which augment the training data using only the conventional approaches such as sampling input images to smaller crops, rotating images randomly, and flipping images horizontally, we suggested another approach by synthesizing new training images having the same scene content of the original images, but with different illuminations using the ground-truth illuminant color of other training images. Besides, we additionally increased the number of the training images by creating a horizontally flipped copy from each synthesized training image.

4.4.3 Implementation and Training Settings

The implementation of the CNN was done by using the MatConvNet framework [79], on a computer with specifications: Intel i7-7700HQ CPU at 2.80 GHz, 32 GB RAM, and an NVIDIA GTX 1070 GPU. To further overcome the issue of having a small dataset, we followed a useful strategy commonly known as Transfer Learning [80] by training the CNN firstly with a very large dataset such as the ImageNet [77] which consists of more than 1.2 M images used for object classification task. Then, the pre-trained network is fine-tuned for our own illuminant classification task instead of training the network from scratch to avoid the overfitting problem. When fine-tuning the SqueezeNet, the number of outputs in the Conv10 layer is changed from 1000 (number of ImageNet classes) to K=17 (number of illuminant classes), and its weights are initialized again from the Gaussian distribution similar to [76]. The network parameters are optimized by back-propagation using ADAM [67] with a base learning rate (LR) of 0.0003, a batch size (BS) of 20, and a weight decay (WD) of 0.00005 for regularization.

The LR for all the pre-trained layers is set the same as the Conv10 layer, instead of smaller values, because the illuminant estimation task is quite different from the object classification task and hence, their weights should be more tuned to be adapted for the new purpose.

4.5 Illuminant Estimation

Following the training phase of the CNN which is learned to predict the probability of an image $(\boldsymbol{I})$ belonging to one of the (K) illuminant clusters similar to [14], the illuminant color estimate $(\boldsymbol{\rho}^E)$ of a test image is computed by taking a weighted sum of the cluster centroids $(\boldsymbol{C})$ using the K probabilities from the CNN output $\left(\widehat{\boldsymbol{Y}}\right)$ of the input test image:

$$\boldsymbol{\rho}^E = \sum_{n=1}^{K} C_n \, \hat{Y}_n \tag{9}$$

$$\widehat{\boldsymbol{Y}} = \begin{pmatrix} P\left(\hat{Y} = 1 \middle| \boldsymbol{I}\right) \\ P\left(\hat{Y} = 2 \middle| \boldsymbol{I}\right) \\ P\left(\hat{Y} = 3 \middle| \boldsymbol{I}\right) \\ \vdots \\ P\left(\hat{Y} = K \middle| \boldsymbol{I}\right) \end{pmatrix} \tag{10}$$

Indeed, incorporating all the output probabilities in the computation of the final illuminant color estimate is found to be in general more useful than using only the cluster centroid with the highest probability, as this would help to reduce the likelihood of making high errors in the illuminant computation for the cases when the CNN may fail to predict the correct cluster.

CHAPTER 5: EXPERIMENTAL RESULTS AND DISCUSSION

5.1 Experimental Results

In this section, the performance of the proposed method is compared against various existing illuminant estimation methods on the benchmark Gehler-Shi dataset [47], [70]. The most widely used error metric for evaluating the illuminant estimation methods is the angular error (in degrees) between the ground-truth and the estimated illuminant colors which is expressed as follows [13]:

$$err_{Angular}\left(\boldsymbol{\rho}^{E}{}_{GT}\,,\boldsymbol{\rho}^{E}{}_{EST}\right) = \frac{180^{\circ}}{\pi}\cos^{-1}\left(\frac{\boldsymbol{\rho}^{E}{}_{GT}\ \cdot \boldsymbol{\rho}^{E}{}_{EST}}{\left\|\boldsymbol{\rho}^{E}{}_{GT}\right\|\ \cdot \left\|\boldsymbol{\rho}^{E}{}_{EST}\right\|}\right) \tag{11}$$

where ($\boldsymbol{\rho}^{E}{}_{GT}\ \cdot \boldsymbol{\rho}^{E}{}_{EST}$) is the dot product of the ground-truth illuminant color $\boldsymbol{\rho}^{E}{}_{GT}$ and the estimated illuminant color $\boldsymbol{\rho}^{E}{}_{EST}$, and $\|\cdot\|$ indicates the Euclidean norm. We follow the standard experimental settings as done in all methods for dataset evaluation by using the 3-fold cross validation, and report several statistical metrics of the computed errors of the dataset including the mean, the median, the tri-mean, the mean of the lowest 25% errors, the mean of the highest 25% errors, and the 95th percentile error as shown in Table 3. Besides the quantitative analysis, some examples of color-corrected indoor and outdoor images are presented in Figure 17 to Figure 26 using the ground-truth and the estimated illuminant colors by the proposed method as a qualitative performance measure. For every presented image, the angular error is reported under the image, but it should be noted that the Macbeth Color Checkers are masked out during training and testing. The selected images are chosen by sorting images by increasing error, so Figure 17 represents the image with the lowest error and Figure 26 represents the image with the highest error, which indicate that the proposed method can produce plausible images, even for some extreme cases. The computational time to estimate the illuminant color for one image on a GPU using our unoptimized Matlab code is around 43 msec.

Table 3: *Performance Comparison between Various Methods on the Gehler-Shi Dataset*

Method	**Mean**	**Median**	**Tri-mean**	**Best-25%**	**Worst-25%**	**95th Pct.**
Support Vector Regression [46]	8.08	6.73	7.19	3.35	14.89	-
White-Patch [21]	7.55	5.68	6.35	1.45	16.12	-
Grey-World [20]	6.36	6.28	6.28	2.33	10.58	11.30
Edge-based Gamut [43]	6.52	5.04	5.43	1.90	13.58	-
1st-order Gray-Edge [22]	5.33	4.52	4.73	1.86	10.03	11.00
2nd-order Gray-Edge [22]	5.13	4.44	4.62	2.11	9.26	-
Shades-of-Gray [23]	4.93	4.01	4.23	1.14	10.20	11.90
Bayesian [47]	4.82	3.46	3.88	1.26	10.49	-
General Gray-World [22]	4.66	3.48	3.81	1.00	10.09	-
Grey Pixels [56]	4.60	3.10	-	-	-	-
Natural Image Statistics [35]	4.19	3.13	3.45	1.00	9.22	11.7
Intersection-based Gamut [43]	4.20	2.39	2.93	0.51	10.70	-
Pixel-based Gamut [43]	4.20	2.33	2.91	0.50	10.72	14.1
Bilayer Sparse-Coding [55]	4.00	2.50	2.80	1.00	10.80	-
Double-Opponency [29]	3.98	2.43	-	-	9.08	-
CART-based Combination [34]	3.90	2.91	3.21	1.02	8.27	-
Spatio-Spectral [49]	3.59	2.96	3.10	0.95	7.61	-
Bright-and-Dark Colors PCA [28]	3.52	2.14	2.47	0.50	8.74	-
High-level Visual Information [37]	3.48	2.47	2.61	0.84	8.01	-
Local Surface Reflectance [30]	3.31	2.80	2.87	1.14	6.39	-
Exemplar-based [50]	2.89	2.27	2.42	0.82	5.97	-
Corrected-Moment [53]	2.86	2.04	2.22	0.70	6.34	6.90
CNN Regression [19]	2.63	1.98	-	-	-	-
Luminance-to-Chromaticity [54]	2.56	1.67	1.89	0.52	6.07	-
Regression-Tree [51]	2.42	1.65	1.75	0.38	5.87	-
CNN Classification [14]	2.16	1.47	1.61	0.37	5.12	-
CCC [52]	1.95	1.22	1.38	0.35	4.76	5.85
DS-Net / Shi et al. 2016 [18]	1.90	1.12	1.33	0.31	4.84	5.99
FC^4 [17]	1.65	1.18	1.27	0.38	3.78	4.73
Proposed	**1.85**	**1.25**	**1.36**	**0.40**	**4.35**	**5.33**

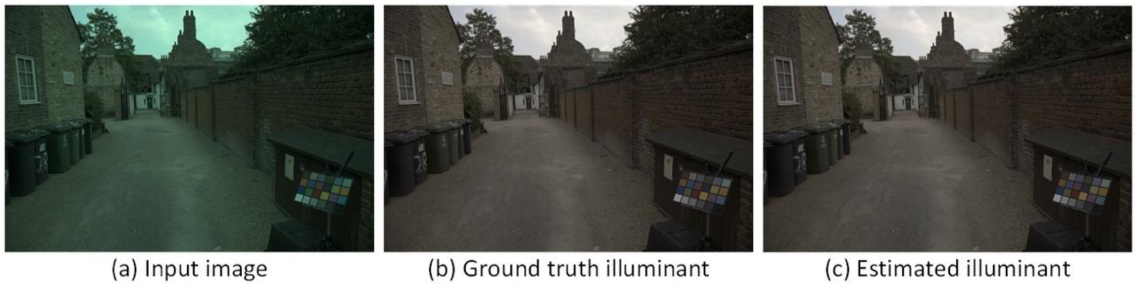


Figure 17. A corrected image from the Gehler-Shi dataset, Angular Error = 0.0361°

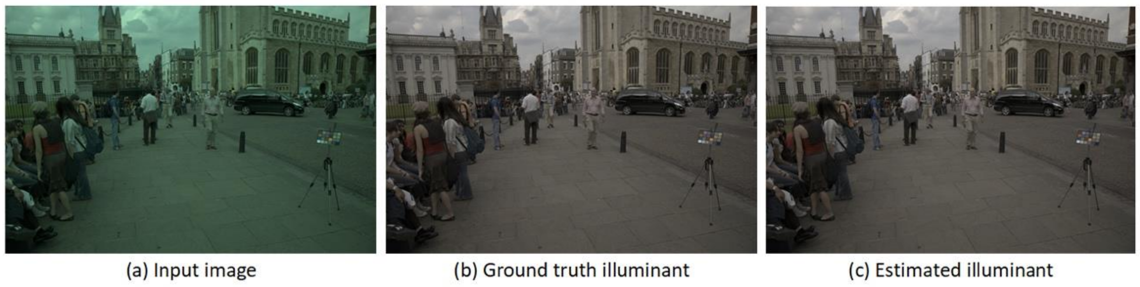


Figure 18. A corrected image from the Gehler-Shi dataset, Angular Error = 0.2469°

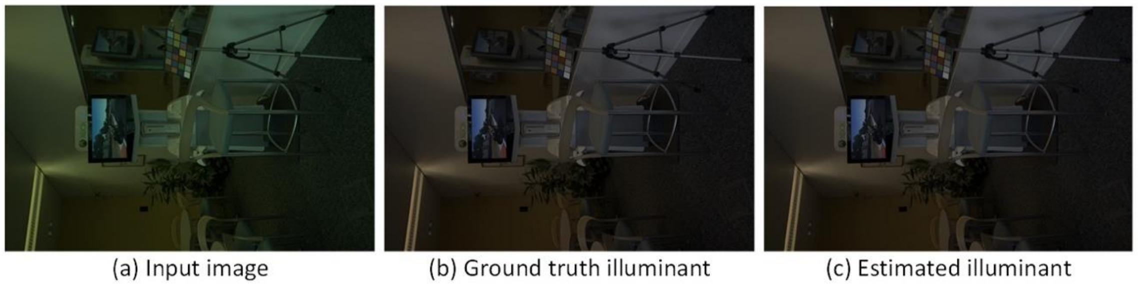


Figure 19. A corrected image from the Gehler-Shi dataset, Angular Error = 0.4575°

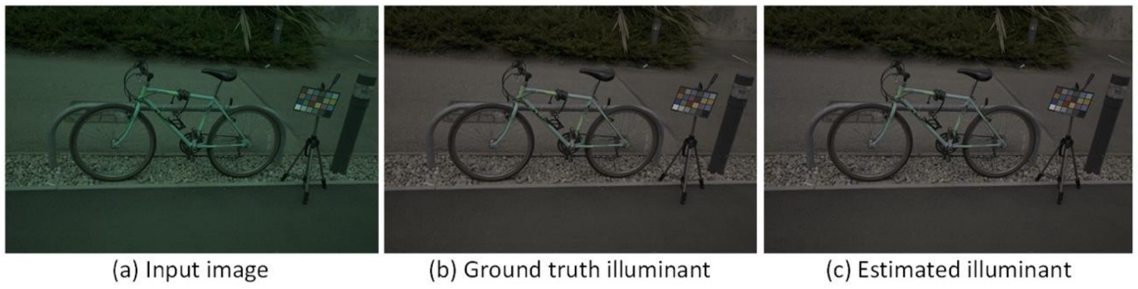


Figure 20. A corrected image from the Gehler-Shi dataset, Angular Error = 0.6395°

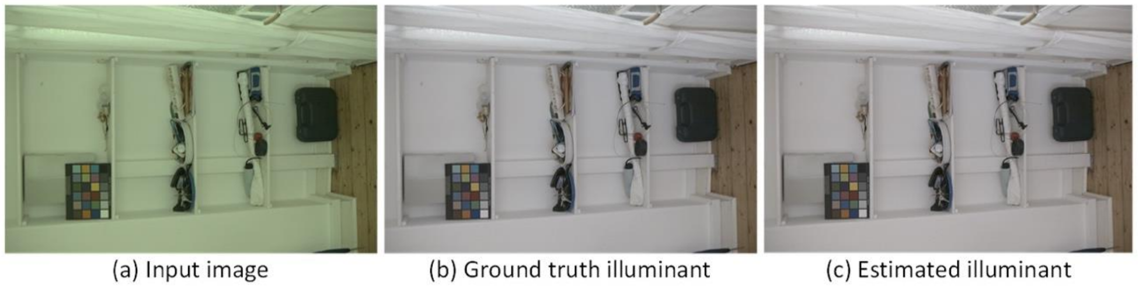


Figure 21. A corrected image from the Gehler-Shi dataset, Angular Error = 0.8391°

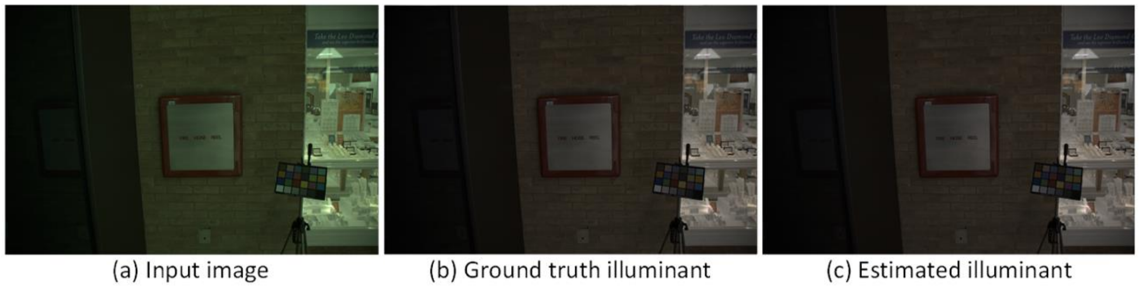


Figure 22. A corrected image from the Gehler-Shi dataset, Angular Error = 1.1416°

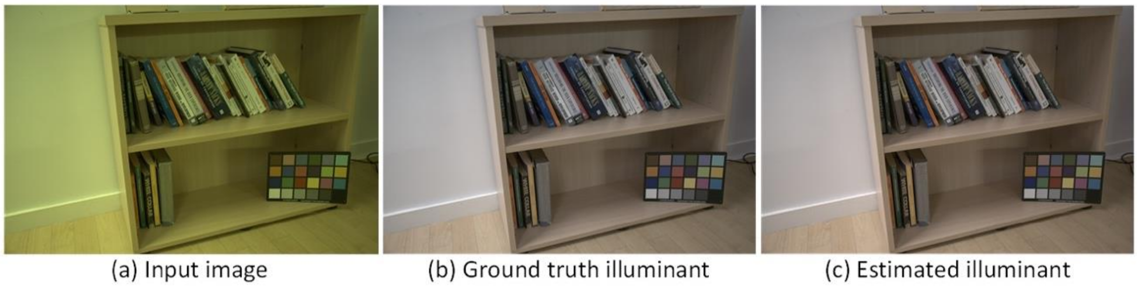


Figure 23. A corrected image from the Gehler-Shi dataset, Angular Error = 1.6891°

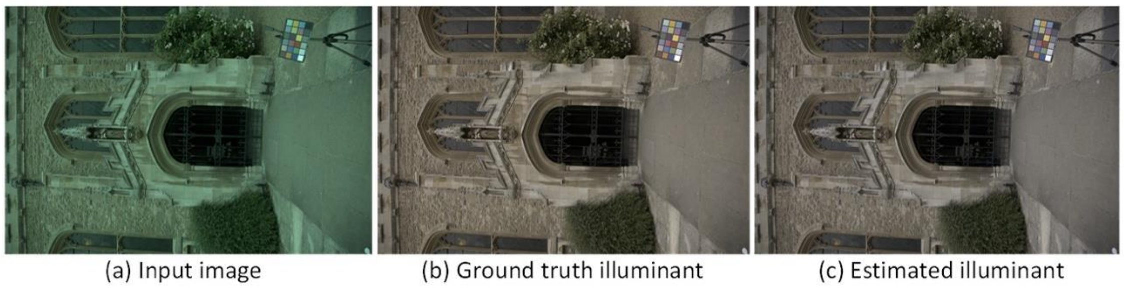


Figure 24. A corrected image from the Gehler-Shi dataset, Angular Error = 2.4024°

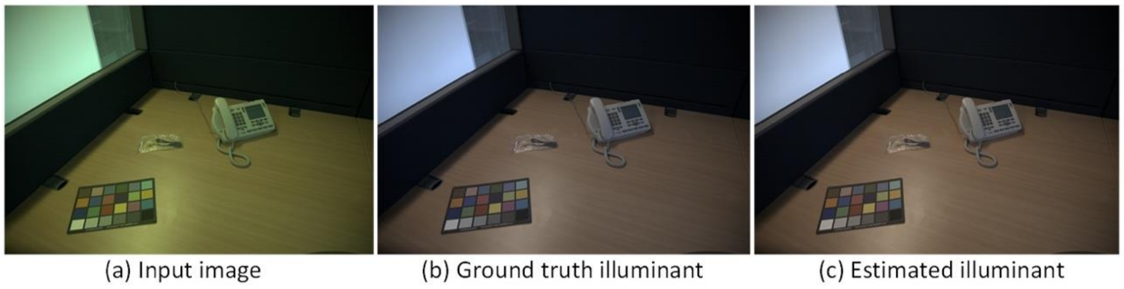


Figure 25. A corrected image from the Gehler-Shi dataset, Angular Error = 3.8635°

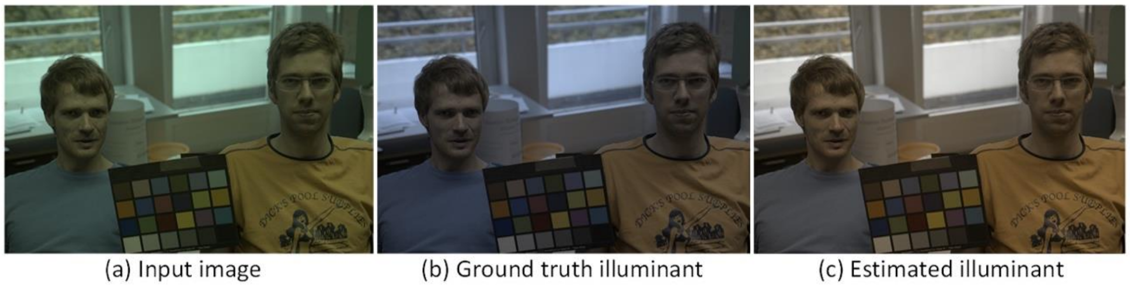


Figure 26. A corrected image from the Gehler-Shi dataset, Angular Error = 14.3812°

5.2 Discussions

As evident from Table 3, the proposed method outperforms most of the methods on all metrics including the CNN-based classification method [14] that we share the same approach of utilizing CNN for illuminant estimation through classification. The major improvement in performance compared to [14] is likely because our adopted network which is the SqueezeNet v1.1 has much fewer learnable parameters than the AlexNet used by [14] which helps to combat the overfitting problem on small datasets such as the Gehler-Shi dataset. Besides, the data augmentation used in our method by synthesizing images provides the advantage of training the CNN with nearly full image content, rather than augmenting the data by sub-sampling the images to multiple patches, which is the case also for the CNN-based regression method proposed by Bianco et al. [19], where some of these patches may not carry any semantic information and hence, can be considered as noisy labels for the CNN. For some metrics, the proposed method shows a competitive performance to the recent state-of-the-art methods presented by the CCC method [52] and the DS-Net method [18], but our method performs better on the worse-25% and the 95th percentile metrics, which indicates a better robustness of the method in the worst-case examples compared to [18], [52]. The proposed method still shows a slightly lower performance than the state-of-the-art CNN-based method (FC^4) [17] proposed by Microsoft research, likely because of their presented novel Confidence-weighted pooling layer which helps improving the CNN training performance by automatically learning to eliminate or mask out the noisy local regions of the input image that do not contain useful informative data for the illuminant estimation which may cause ambiguity to the CNN.

CHAPTER 6: CONCLUSIONS AND FUTURE WORK

In this thesis, we have introduced an effective Deep Learning based solution for the illuminant estimation problem. Specifically, we trained a CNN to perform classification on input images belonging to one of the possible pre-defined K illuminants, and the classification results are combined to produce the final illuminant color estimation. The performance of the proposed method has been validated by comparing it against many other illuminant estimation methods including the most recent state-of-the-art methods on the available benchmark illuminant estimation dataset. The results show that our method outperforms all the previous methods including most of the CNN-based methods, and it could show a competitive performance to the most recent state-of-the-art methods, especially on the worst-case metrics. The main contribution of this work is that we were able to boost the idea of solving the illuminant estimation problem through the CNN-based classification approach by addressing all possible issues that limit the performance of the previous related work [14], especially the choice of the CNN architecture and the strategy of the data augmentation, which greatly enhances the performance of the CNN-based classification approach and makes it among the top performing state-of-the-art methods. In our work, we have utilized the SqueezeNet v1.1 that has the advantage of having the same classification accuracy of the AlexNet used by [14] with 50$\times$ less learnable parameters, which helps reducing the overfitting problem of deep CNNs with relatively small datasets. In addition, we proposed a novel strategy for data augmentation that allows for training the CNN with nearly full image content, instead of sampling the input images to smaller patches, where some of these patches may contain no semantic information and hence, deteriorating the performance of the CNN training as these patches can be considered noisy labels.

As the future work, we plan to examine the effect of different choices for the number of illuminant clusters and we will also evaluate the performance of the proposed method on other benchmark datasets. We also plan to investigate other CNN architectures such as the recently published MobileNet v2 [81] by Google research which has the big advantage of having less number of learnable parameters, only slightly higher than the SqueezeNet v1.1, but with much higher classification accuracy on the ImageNet dataset. Besides, we seek to combine the Confidence-weighted pooling layer proposed by the current state-of-the-art method to our CNN architecture, which will greatly benefit from masking out the local noisy regions in the image, and hence improve more the training and inference performance.

REFERENCES

[1] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "ImageNet Classification with Deep Convolutional Neural Networks", *Advances in Neural Information Processing Systems 25*, pp. 1097-1105, 2012.

[2] S. Ren, K. He, R. Girshick and J. Sun, "Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks", *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 39, no. 6, pp. 1137-1149, 2017.

[3] E. Shelhamer, J. Long and T. Darrell, "Fully Convolutional Networks for Semantic Segmentation", *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 39, no. 4, pp. 640-651, 2017.

[4] M. Arsenovic, S. Sladojevic, A. Anderla and D. Stefanovic, "FaceTime — Deep learning based face recognition attendance system", *2017 IEEE 15th International Symposium on Intelligent Systems and Informatics (SISY)*, 2017.

[5] T. Sainath, B. Kingsbury, G. Saon, H. Soltau, A. Mohamed, G. Dahl and B. Ramabhadran, "Deep Convolutional Neural Networks for Large-scale Speech Tasks", *Neural Networks*, vol. 64, pp. 39-48, 2015.

[6] J. Zhang and C. Zong, "Deep Neural Networks in Machine Translation: An Overview", *IEEE Intelligent Systems*, vol. 30, no. 5, pp. 16-25, 2015.

[7] X. Li, G. Zhang, H. Huang, Z. Wang and W. Zheng, "Performance Analysis of GPU-Based Convolutional Neural Networks", *2016 45th International Conference on Parallel Processing (ICPP)*, 2016.

[8] "GPU-Based Deep Learning Inference: A Performance and Power Analysis", *Nvidia.com*, 2015. [Online]. Available: https://www.nvidia.com/content/tegra/embedded-systems/pdf/jetson_tx1_whitepaper.pdf. [Accessed: 15- Nov- 2017].

[9] K. Zhang, W. Zuo, S. Gu and L. Zhang, "Learning Deep CNN Denoiser Prior for Image Restoration", *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2017.

[10] W. Lai, J. Huang, N. Ahuja and M. Yang, "Deep Laplacian Pyramid Networks for Fast and Accurate Super-Resolution", *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2017.

[11] M. Gharbi, J. Chen, J. Barron, S. Hasinoff and F. Durand, "Deep bilateral learning for real-time image enhancement", *ACM Transactions on Graphics*, vol. 36, no. 4, pp. 1-12, 2017.

[12] R. Ramanath, W. Snyder, Y. Yoo and M. Drew, "Color image processing pipeline", *IEEE Signal Processing Magazine*, vol. 22, no. 1, pp. 34-43, 2005.

[13] A. Gijsenij, T. Gevers and J. van de Weijer, "Computational Color Constancy: Survey and Experiments", *IEEE Transactions on Image Processing*, vol. 20, no. 9, pp. 2475-2489, 2011.

[14] S. Oh and S. Kim, "Approaching the computational color constancy as a classification problem through deep learning", *Pattern Recognition*, vol. 61, pp. 405-416, 2017.

[15] C. Aytekin, J. Nikkanen and M. Gabbouj, "A Data Set for Camera-Independent Color Constancy", *IEEE Transactions on Image Processing*, vol. 27, no. 2, pp. 530-544, 2017.

[16] "Photoskop visual learning: new way to learn photography", *Photoskop.com*, 2017. [Online]. Available: http://www.photoskop.com. [Accessed: 15- Dec- 2017].

[17] Y. Hu, B. Wang and S. Lin, "FC^4: Fully Convolutional Color Constancy with Confidence-Weighted Pooling", *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2017.

[18] W. Shi, C. Loy and X. Tang, "Deep Specialized Network for Illuminant Estimation", *Computer Vision – ECCV 2016*, pp. 371-387, 2016.

[19] S. Bianco, C. Cusano and R. Schettini, "Color constancy using CNNs", *2015 IEEE Conference on Computer Vision and Pattern Recognition Workshops (CVPRW)*, 2015.

[20] G. Buchsbaum, "A spatial processor model for object colour perception", *Journal of the Franklin Institute*, vol. 310, no. 1, pp. 1-26, 1980.

[21] D. Brainard and B. Wandell, "Analysis of the retinex theory of color

vision", *Journal of the Optical Society of America A*, vol. 3, no. 10, p. 1651, 1986.

[22] J. van de Weijer, T. Gevers and A. Gijsenij, "Edge-Based Color Constancy", *IEEE Transactions on Image Processing*, vol. 16, no. 9, pp. 2207-2214, 2007.

[23] G. Finlayson and E. Trezzi, "Shades of Gray and Colour Constancy", *Color Imaging Conference*, 2004.

[24] G. Finlayson and G. Schaefer, "Solving for colour constancy using a constrained dichromatic reflection model", *International Journal of Computer Vision*, vol. 42, no. 3, pp. 127-144, 2001.

[25] H. Lee, "Method for computing the scene-illuminant chromaticity from specular highlights", *Journal of the Optical Society of America A*, vol. 3, no. 10, p. 1694, 1986.

[26] M. Drew, H. Joze and G. Finlayson, "Specularity, the Zeta-image, and Information-Theoretic Illuminant Estimation", *Computer Vision – ECCV 2012. Workshops and Demonstrations*, pp. 411-420, 2012.

[27] H. Joze, M. Drew, G. Finlayson and P. Rey, "The Role of Bright Pixels in Illumination Estimation", *Color Imaging Conference*, 2012.

[28] D. Cheng, D. Prasad and M. Brown, "Illuminant estimation for color constancy: why spatial-domain methods work and the role of the color distribution", *Journal of the Optical Society of America A*, vol. 31, no. 5, p. 1049, 2014.

[29] S. Gao, K. Yang, C. Li and Y. Li, "Color Constancy Using Double-Opponency", *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 37, no. 10, pp. 1973-1985, 2015.

[30] S. Gao, W. Han, K. Yang, C. Li and Y. Li, "Efficient Color Constancy with Local Surface Reflectance Statistics", *Computer Vision – ECCV 2014*, pp. 158-173, 2014.

[31] G. Schaefer, S. Hordley and G. Finlayson, "A Combined Physical and Statistical Approach to Colour Constancy", *2005 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'05)*, pp. 148-153, 2005.

[32] S. Bianco, F. Gasparini and R. Schettini, "Consensus-based framework for illuminant chromaticity estimation", *Journal of Electronic Imaging*, vol. 17, no. 2, pp. 023013-1–023013-9, 2008.

[33] A. Gijsenij and T. Gevers, "Color Constancy using Natural Image Statistics", *2007 IEEE Conference on Computer Vision and Pattern Recognition*, pp. 1-8, 2007.

[34] S. Bianco, G. Ciocca, C. Cusano and R. Schettini, "Automatic color constancy algorithm selection and combination", *Pattern Recognition*, vol. 43, no. 3, pp. 695-705, 2010.

[35] A. Gijsenij and T. Gevers, "Color Constancy Using Natural Image Statistics and Scene Semantics", *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 33, no. 4, pp. 687-698, 2011.

[36] S. Bianco, G. Ciocca, C. Cusano and R. Schettini, "Improving Color Constancy Using Indoor–Outdoor Image Classification", *IEEE Transactions on Image Processing*, vol. 17, no. 12, pp. 2381-2392, 2008.

[37] J. van de Weijer, C. Schmid and J. Verbeek, "Using High-Level Visual Information for Color Constancy", *2007 IEEE 11th International Conference on Computer Vision*, pp. 1-8, 2007.

[38] D. Forsyth, "A novel algorithm for color constancy", *International Journal of Computer Vision*, vol. 5, no. 1, pp. 5-35, 1990.

[39] K. Barnard, "Improvements to Gamut Mapping Colour Constancy Algorithms", *Computer Vision - ECCV 2000*, pp. 390-403, 2000.

[40] G. Finlayson and R. Xu, "Convex programming colour constancy", *IEEE workshop on color and photometric methods in computer vision*, pp. 1-8, 2003.

[41] M. Mosny and B. Funt, "Cubical gamut mapping colour constancy", *Proc. CGIV2010 IS&T Fifth European Conf. on Colour in Graphics, Imaging and Vision*, 2010.

[42] G. Finlayson, S. Hordley and R. Xu, "Convex programming colour constancy with a diagonal-offset model", *IEEE International Conference on Image Processing 2005*, pp. 948-951, 2005.

[43] A. Gijsenij, T. Gevers and J. van de Weijer, "Generalized Gamut Mapping using Image Derivative Structures for Color Constancy", *International Journal of Computer Vision*, vol. 86, no. 2-3, pp. 127-139, 2010.

[44] G. Finlayson, S. Hordley and P. HubeL, "Color by correlation: a simple, unifying

framework for color constancy", *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 23, no. 11, pp. 1209-1221, 2001.

[45] V. Cardei, B. Funt and K. Barnard, "Estimating the scene illumination chromaticity by using a neural network", *Journal of the Optical Society of America A*, vol. 19, no. 12, pp. 2374-2386, 2002.

[46] W. Xiong and B. Funt, "Estimating Illumination Chromaticity Via Support Vector Regression", *Journal of Imaging Science and Technology*, vol. 50, no. 4, pp. 341-348, 2006.

[47] P. Gehler, C. Rother, A. Blake, T. Minka and T. Sharp, "Bayesian color constancy revisited", *2008 IEEE Conference on Computer Vision and Pattern Recognition*, 2008.

[48] C. Rosenberg, A. Ladsariya and T. Minka, "Bayesian color constancy with non-gaussian models", *Advances in Neural Information Processing Systems 16 (NIPS 2003)*, 2003.

[49] A. Chakrabarti, K. Hirakawa and T. Zickler, "Color Constancy with Spatio-Spectral Statistics", *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 34, no. 8, pp. 1509-1519, 2012.

[50] H. Joze and M. Drew, "Exemplar-Based Color Constancy and Multiple Illumination", *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 36, no. 5, pp. 860-873, 2014.

[51] D. Cheng, B. Price, S. Cohen and M. Brown, "Effective learning-based illuminant estimation using simple features", *2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2015.

[52] J. Barron, "Convolutional Color Constancy", *2015 IEEE International Conference on Computer Vision (ICCV)*, 2015.

[53] G. Finlayson, "Corrected-Moment Illuminant Estimation", *2013 IEEE International Conference on Computer Vision*, 2013.

[54] A. Chakrabarti, "Color Constancy by Learning to Predict Chromaticity from Luminance", *Advances in Neural Information Processing Systems 28 (NIPS 2015)*, 2015.

[55] B. Li, W. Xiong and W. Hu, "Illumination Estimation Based on Bilayer Sparse Coding", *2013 IEEE Conference on Computer Vision and Pattern Recognition*, 2013.

[56] K. Yang, S. Gao and Y. Li, "Efficient illuminant estimation for color constancy using grey pixels", 2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR), 2015.

[57] A. Karpathy, "Stanford University CS231n: Convolutional Neural Networks for Visual Recognition", 2018. [Online]. Available: http://cs231n.github.io/.

[58] I. Aleksander and H. Morton, An introduction to neural computing. Chapman and Hall, 1990.

[59] E. Alpaydin, Introduction to machine learning, 3rd ed. Cambridge, MA: MIT Press, 2014.

[60] S. Houzel, "The human brain in numbers: a linearly scaled-up primate brain", *Frontiers in Human Neuroscience*, vol. 3, no. 31, 2009.

[61] C. M. Bishop, Pattern recognition and machine learning (information science and statistics). New York: Springer-Verlag New York, 2006.

[62] V. Maltarollo, K. Honório and A. Da Silva, "Applications of Artificial Neural Networks in Chemical Problems", *Artificial Neural Networks - Architectures and Applications*, 2013.

[63] I. Goodfellow, Y. Bengio and A. Courville, Deep Learning. MIT Press, 2016.

[64] M. Zeiler, "ADADELTA: An Adaptive Learning Rate Method", *arXiv:1212.5701*, 2012.

[65] J. Duchi, E. Hazan and Y. Singer, "Adaptive Subgradient Methods for Online Learning and Stochastic Optimization", *Journal of Machine Learning Research*, pp. 2121-2159, 2011.

[66] G. Hinton, N. Srivastava and K. Swersky, *Neural Networks for Machine Learning, Lecture 6a: Overview of mini-batch gradient descent*. 2016, p. 26.

[67] D. Kingma and J. Ba, "Adam: A method for stochastic optimization", *International Conference on Learning Representations*, 2015.

[68] M. Nielsen, "Online Book: Neural Networks and Deep Learning", 2017. [Online].

Available: http://neuralnetworksanddeeplearning.com/.

[69] Y. LeCun, Y. Bengio and G. Hinton, "Deep learning", *Nature*, vol. 521, no. 7553, pp. 436-444, 2015.

[70] L. Shi and B. Funt, "Re-processed version of the gehler color constancy dataset of 568 images". [Online]. Available: http://www.cs.sfu.ca/~colour/data/.

[71] S. Lloyd, "Least squares quantization in PCM", *IEEE Transactions on Information Theory*, vol. 28, no. 2, pp. 129-137, 1982.

[72] D. Arthur and S. Vassilvitskii, "K-Means++: The Advantages of Careful Seeding", *Proceedings of the Eighteenth Annual ACM-SIAM Symposium on Discrete Algorithms*, 2007.

[73] R. Thorndike, "Who belongs in the family?", Psychometrika, vol. 18, no. 4, pp. 267-276, 1953.

[74] G. Schwarz, "Estimating the Dimension of a Model", The Annals of Statistics, vol. 6, no. 2, pp. 461-464, 1978.

[75] J. Raitoharju, K. Samiee, S. Kiranyaz and M. Gabbouj, "Particle swarm clustering fitness evaluation with computational centroids", *Swarm and Evolutionary Computation*, vol. 34, pp. 103-118, 2017.

[76] F. Iandola, S. Han, M. Moskewicz, K. Ashraf, W. Dally and K. Keutzer, "Squeezenet: Alexnet-level accuracy with 50x fewer parameters and < 0.5 mb model size", *ArXiv:1602.07360*, 2016.

[77] J. Deng, W. Dong, R. Socher, L. Li, K. Li and L. Fei-Fei, "ImageNet: A large-scale hierarchical image database", *2009 IEEE Conference on Computer Vision and Pattern Recognition*, 2009.

[78] "Normalizing inputs - Practical aspects of Deep Learning | Coursera", Coursera, 2017. [Online]. Available: https://www.coursera.org/learn/deep-neural-network/lecture/lXv6U/normalizing-inputs.

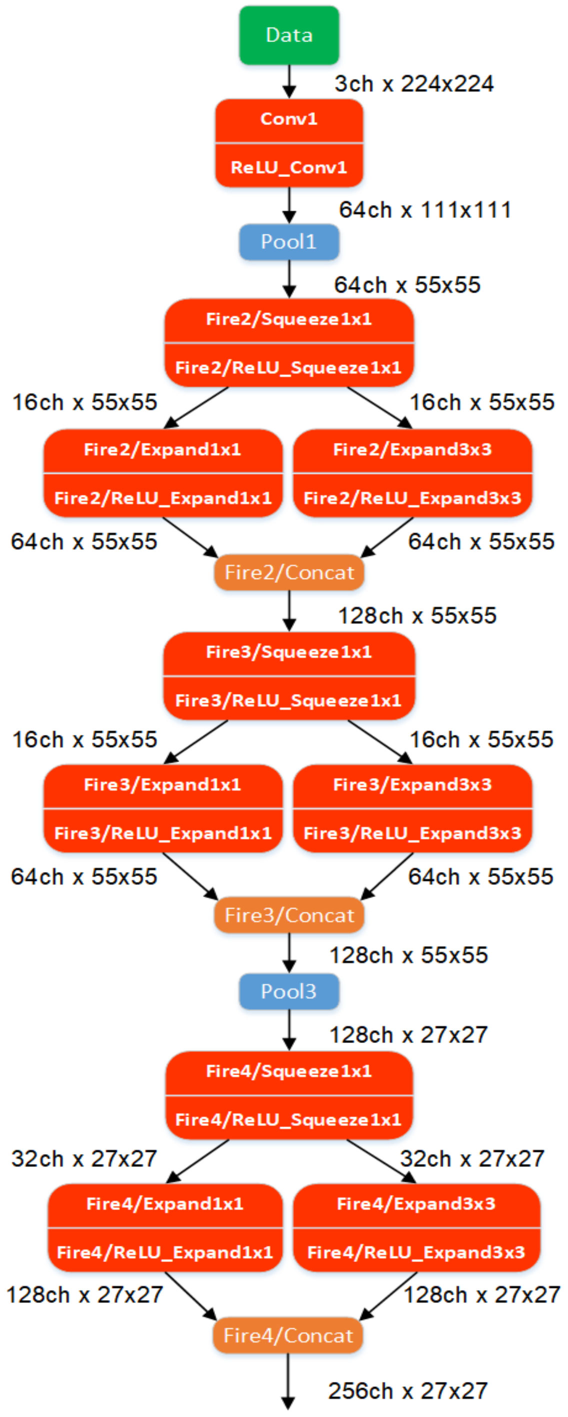
[79] A. Vedaldi and K. Lenc, "MatConvNet: Convolutional Neural Networks for MATLAB", *Proceedings of the 23rd ACM international conference on Multimedia - MM '15*, 2015.

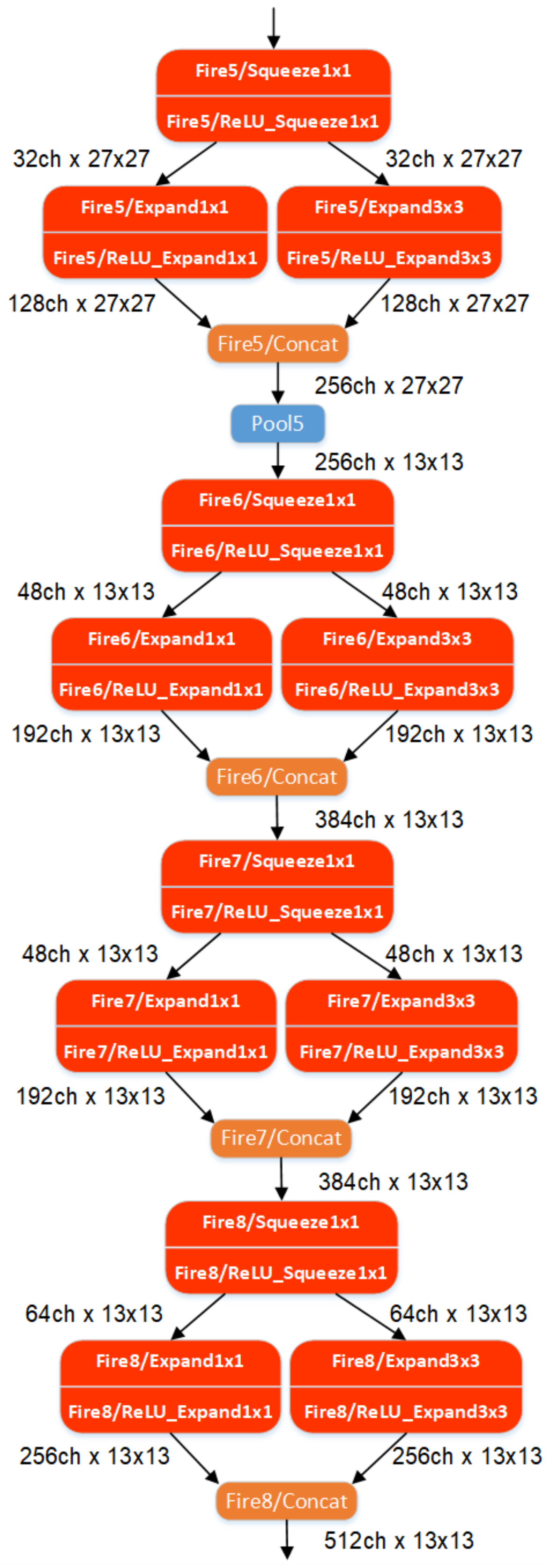
[80] M. Oquab, L. Bottou, I. Laptev and J. Sivic, "Learning and Transferring Mid-level Image Representations Using Convolutional Neural Networks", *2014 IEEE Conference on Computer Vision and Pattern Recognition*, 2014.

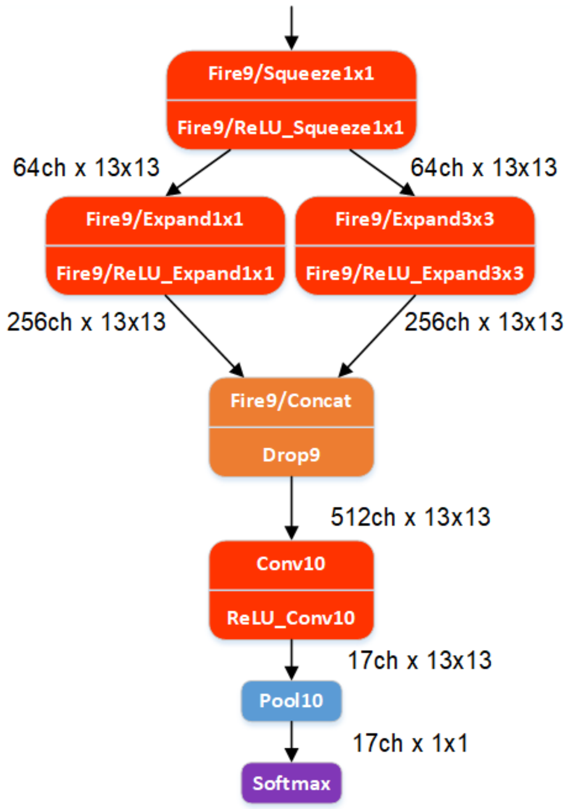
[81] M. Sandler, A. Howard, M. Zhu, A. Zhmoginov and L. Chen, "MobileNetV2: Inverted Residuals and Linear Bottlenecks", *arXiv:1801.04381*, 2018.

APPENDIX

A.1 Architecture of the SqueezeNet v1.1




Fire5/Squeeze1x1
Fire5/ReLU_Squeeze1x1
32ch x 27x27
32ch x 27x27
Fire5/Expand1x1
Fire5/ReLU_Expand1x1
Fire5/Expand3x3
Fire5/ReLU_Expand3x3
128ch x 27x27
128ch x 27x27
Fire5/Concat
256ch x 27x27
Pool5
256ch x 13x13
Fire6/Squeeze1x1
Fire6/ReLU_Squeeze1x1
48ch x 13x13
48ch x 13x13
Fire6/Expand1x1
Fire6/ReLU_Expand1x1
Fire6/Expand3x3
Fire6/ReLU_Expand3x3
192ch x 13x13
192ch x 13x13
Fire6/Concat
384ch x 13x13
Fire7/Squeeze1x1
Fire7/ReLU_Squeeze1x1
48ch x 13x13
48ch x 13x13
Fire7/Expand1x1
Fire7/ReLU_Expand1x1
Fire7/Expand3x3
Fire7/ReLU_Expand3x3
192ch x 13x13
192ch x 13x13
Fire7/Concat
384ch x 13x13
Fire8/Squeeze1x1
Fire8/ReLU_Squeeze1x1
64ch x 13x13
64ch x 13x13
Fire8/Expand1x1
Fire8/ReLU_Expand1x1
Fire8/Expand3x3
Fire8/ReLU_Expand3x3
256ch x 13x13
256ch x 13x13
Fire8/Concat
512ch x 13x13


Fire9/Squeeze1x1
Fire9/ReLU_Squeeze1x1
64ch x 13x13
64ch x 13x13
Fire9/Expand1x1
Fire9/ReLU_Expand1x1
Fire9/Expand3x3
Fire9/ReLU_Expand3x3
256ch x 13x13
256ch x 13x13
Fire9/Concat
Drop9
512ch x 13x13
Conv10
ReLU_Conv10
17ch x 13x13
Pool10
17ch x 1x1
Softmax

A.2 Specifications of the SqueezeNet v1.1 layers

Table 4: *Specifications of the SqueezeNet v1.1 Layers*

ID	Layer Name	Layer Type	Kernel	Stride	Pad	Channel_in	Channel_out	Notes
1	Data	Data					3	
2	Conv1	Convolutional	3×3	2	0	3	64	
3	ReLU_Conv1	ReLU				64	64	
4	Pool1	Max-pooling	3×3	2	0	64	64	
5	Fire2/Squeeze1x1	Convolutional	1×1	1	0	64	16	
6	Fire2/ReLU_Squeeze1x1	ReLU				16	16	
7	Fire2/Expand1x1	Convolutional	1×1	1	0	16	64	
8	Fire2/ReLU_Expand1x1	ReLU				64	64	
9	Fire2/Expand3x3	Convolutional	3×3	1	1	16	64	
10	Fire2/ReLU_Expand3x3	ReLU				64	64	
11	Fire2/Concat	Concatenation				128	128	
12	Fire3/Squeeze1x1	Convolutional	1×1	1	0	128	16	
13	Fire3/ReLU_Squeeze1x1	ReLU				16	16	
14	Fire3/Expand1x1	Convolutional	1×1	1	0	16	64	
15	Fire3/ReLU_Expand1x1	ReLU				64	64	
16	Fire3/Expand3x3	Convolutional	3×3	1	1	16	64	
17	Fire3/ReLU_Expand3x3	ReLU				64	64	
18	Fire3/Concat	Concatenation				128	128	
19	Pool3	Max-pooling	3×3	2	0	128	128	
20	Fire4/Squeeze1x1	Convolutional	1×1	1	0	128	32	
21	Fire4/ReLU_Squeeze1x1	ReLU				32	32	
22	Fire4/Expand1x1	Convolutional	1×1	1	0	32	128	
23	Fire4/ReLU_Expand1x1	ReLU				128	128	
24	Fire4/Expand3x3	Convolutional	3×3	1	1	32	128	
25	Fire4/ReLU_Expand3x3	ReLU				128	128	
26	Fire4/Concat	Concatenation				256	256	
27	Fire5/Squeeze1x1	Convolutional	1×1	1	0	256	32	
28	Fire5/ReLU_Squeeze1x1	ReLU				32	32	
29	Fire5/Expand1x1	Convolutional	1×1	1	0	32	128	
30	Fire5/ReLU_Expand1x1	ReLU				128	128	
31	Fire5/Expand3x3	Convolutional	3×3	1	1	32	128	
32	Fire5/ReLU_Expand3x3	ReLU				128	128	
33	Fire5/Concat	Concatenation				256	256	
34	Pool5	Max-pooling	3×3	2	0	256	256	
35	Fire6/Squeeze1x1	Convolutional	1×1	1	0	256	48	
36	Fire6/ReLU_Squeeze1x1	ReLU				48	48	
37	Fire6/Expand1x1	Convolutional	1×1	1	0	48	192	
38	Fire6/ReLU_Expand1x1	ReLU				192	192	
39	Fire6/Expand3x3	Convolutional	3×3	1	1	48	192	
40	Fire6/ReLU_Expand3x3	ReLU				192	192	
41	Fire6/Concat	Concatenation				384	384	
42	Fire7/Squeeze1x1	Convolutional	1×1	1	0	384	48	
43	Fire7/ReLU_Squeeze1x1	ReLU				48	48	
44	Fire7/Expand1x1	Convolutional	1×1	1	0	48	192	
45	Fire7/ReLU_Expand1x1	ReLU				192	192	
46	Fire7/Expand3x3	Convolutional	3×3	1	1	48	192	
47	Fire7/ReLU_Expand3x3	ReLU				192	192	
48	Fire7/Concat	Concatenation				384	384	
49	Fire8/Squeeze1x1	Convolutional	1×1	1	0	384	64	
50	Fire8/ReLU_Squeeze1x1	ReLU				64	64	

51	Fire8/Expand1x1	Convolutional	1×1	1	0	64	256	
52	Fire8/ReLU_Expand1x1	ReLU				256	256	
53	Fire8/Expand3x3	Convolutional	3×3	1	1	64	256	
54	Fire8/ReLU_Expand3x3	ReLU				256	256	
55	Fire8/Concat	Concatenation				512	512	
56	Fire9/Squeeze1x1	Convolutional	1×1	1	0	512	64	
57	Fire9/ReLU_Squeeze1x1	ReLU				64	64	
58	Fire9/Expand1x1	Convolutional	1×1	1	0	64	256	
59	Fire9/ReLU_Expand1x1	ReLU				256	256	
60	Fire9/Expand3x3	Convolutional	3×3	1	1	64	256	
61	Fire9/ReLU_Expand3x3	ReLU				256	256	
62	Fire9/Concat	Concatenation				512	512	
63	Drop9	Dropout				512	512	P=0.5
64	Conv10	Convolutional	1×1	1	0	512	17	
65	ReLU_Conv10	ReLU				17	17	
66	Pool10	Avg-pooling	13×13	1	0	17	17	Global
67	Softmax	Softmax				17	17	