Low-light Stereo Image Enhancement Using Convolutional Neural Network

LOW-LIGHT STEREO IMAGE ENHANCEMENT USING CONVOLUTIONAL NEURAL NETWORK

BY

HAMED HASSANISAADI, M.Sc.

A THESIS

SUBMITTED TO THE DEPARTMENT OF ELECTRICAL & COMPUTER ENGINEERING

AND THE SCHOOL OF GRADUATE STUDIES

OF MCMASTER UNIVERSITY

IN PARTIAL FULFILMENT OF THE REQUIREMENTS

FOR THE DEGREE OF

MASTER OF APPLIED SCIENCE

Master of Applied Science (2018) McMaster University

(Electrical & Computer Engineering) Hamilton, Ontario, Canada

TITLE:	Low-light Stereo Image Enhancement Using Convolutional Neural Network
AUTHOR:	Hamed Hassanisaadi M.Sc., (Artificial Intelligence) Shiraz University, Shiraz, Iran
SUPERVISOR:	Dr. Shahram Shirani, Dr. Xiaolin Wu
NUMBER OF PAGES:	xii, 69

To my wife Zahra

Abstract

We present a method that can increase the quality of a low-light stereo image. While traditional stereo imaging methods have focused on estimating depth from stereo images, our method utilizes stereo images to enhance the low-light condition. The critical challenge for enhancing the low-light condition of stereo images is the disparity between the left and the right images. We proposed an end-to-end convolutional neural network to enhance the low-light condition in stereo images without estimating the disparity. Our proposed network has two sub-networks: the first network learns how to enhance the low-light condition of stereo images in luminance, and the second network learns how to reconstruct a normal-light full-color image from enhanced luminance and chrominance of the input image. Our two-stage joint network enhances the low-light condition of stereo images significantly more than single-image low-light enhancement method.

Acknowledgements

I would like to thank my wife Zahra for supporting and accompanying me during these two years.

Hamilton, Ontario, Canada, September 2018

Hamed Hassani Saadi

Notation and abbreviations

CNN	Convolutional Neural Network
CPU	Central Processing Unit
GPU	Graphical Processing Unit
DL	Deep Learning
NN	Neural Network
DNN	Deep Neural Network
GC	Gamma Correction
HistEq	Histogram Equalization
AE	Auto Encoder
GAN	Generative Adversarial Network
MSE	Mean Squared Error
PSNR	Peak Signal-to-Noise Ratio
SSIM	Structural Similarity

Contents

Abstract **iv**

Acknowledgements **v**

Notation and abbreviations **vi**

1 Introduction **1**

1.1 Dual Camera Image Processing 2

1.1.1 Image enhancement 2

1.1.2 Depth Estimation 4

1.2 Problem Statement 5

1.3 Structure of the thesis 5

2 Background **6**

2.1 Neural Network 6

2.1.1 Multi-layer networks 7

2.1.2 Forward Propagation 9

2.1.3 Back Propagation 9

2.1.4 Activation Functions 13

2.2 Convolutional Neural Networks 13
2.2.1 Basic Structure 14
2.3 Overfitting and Regularization 16
2.4 Data Augmentation 17
2.5 Application of CNN on Dual Camera Systems 17
2.5.1 Depth Estimation 17
2.5.2 Super Resolution 26

3 Low-Light Image Enhancement 32
3.1 Introduction 32
3.2 Gamma Correction 33
3.3 Image Histogram Equalization 34
3.3.1 Histogram 34
3.3.2 Histogram Equalization 35
3.4 Low-Light Image Enhancement Using Deep Convolutional Network . 39
3.4.1 Algorithm and Network Architecture 39
3.4.2 Traning Dataset and Results 43

4 Dual Camera Low-Light Image Enhancement 44
4.1 Problem Definition and Motivation 44
4.2 Network Architecture 48
4.2.1 Network Construction 49
4.3 Loss Function 51
4.4 Training Data-set 52
4.5 Evaluation Criteria 55

4.6 Results 57

5 Conclusion and Future Work **62**

5.1 Conclusion 62

5.2 Future Work 62

List of Figures

1.1 The Bayer filter mechanism (Losson *et al.*, 2010). 3

1.2 The Bokeh effect captured by Apple Iphone using a dual camera system (https://support.apple.com/en-ca/HT208118). 4

2.1 An artificial neuron 7

2.2 A fully-connected multi-layer neural network. 8

2.3 ADAM Optimization algorithm (Kingma and Ba, 2014) 12

2.4 Input f, filter mask M, and output h for a blurring filter. 14

2.5 An example of a CNN (Stenroos, 2017) 16

2.6 The input pair (left and right camera) images and the output disparity map. Note that objects closer to the camera have larger disparities than objects farther away. Larger disparities are shown with warmer colors. (Zbontar and LeCun, 2016) 18

2.7 The fast architecture proposed by (Zbontar and LeCun, 2016). 21

2.8 The accurate architecture proposed by (Zbontar and LeCun, 2016). . 22

2.9 A scene example from Middlebury data-set (Zbontar and LeCun, 2016). 24

2.10 The fast and accurate CNN output for an example of Middlebury data-set (Zbontar and LeCun, 2016). 25

2.11 Registering stereo images with parallax (Jeon *et al.*, 2018). 28

2.12 Proposed network architecture by (Jeon *et al.*, 2018) 28
2.13 Results of super-resolution with the x2 factor (Jeon *et al.*, 2018). . . . 31
3.1 Gamma transformation function with different number of γ. 33
3.2 Input and output of gamma transformation for $\gamma = 0.5$ 34
3.3 Histogram equalization results implemented by MATLAB. 38
3.4 A low-light image and the results of f_1, f_2, f_3 (Shen *et al.*, 2017). . . 42
3.5 The architecture of MSR-net (Shen *et al.*, 2017). 42
3.6 ground truth, input, and output of the method proposed by (Shen *et al.*, 2017). 43
4.1 Input and output of the proposed model. 45
4.2 A superimposed photo of a stereo image pair. This image shows the disparity between the two left and right images. 46
4.3 The process of how we extract patches to train the CNN. 47
4.4 The proposed network architecture for low-light enhancement using stereo images. All the convolution layers has the same number of filters (64) with filter size 3×3, stride=1, and padding. The network has two parts: the luminance and color networks. 48
4.5 Residual block used in the proposed network. Both convolutional layers have 64 filters with filter size 3×3, stride=1, and padding. 49
4.6 Different scenes of Middlebury data-set used in training our CNN network. 53
4.7 Different ambient lights (every column) and exposure times (every row) for one of the Middlebury images. 54

4.8 The disparity mapping for image pair shown in Figure 4.7. Bright intensities are corresponding to the close objects to the dual camera setup while dark intensities represent the far objects. 55
4.9 Luminance loss function for the proposed network. 58
4.10 Color loss function for the proposed network. 58
4.11 Results for different methods compared to our method. Gamma correction (GC) with $\lambda = 0.5$, Histogram equalization (HistEq), Single left image (Single), Ours. 59
4.12 Results for different methods compare to our method. Gamma correction (GC) with $\lambda = 0.5$, Histogram equalization (HistEq), Single left image (Single), Ours. 60

Chapter 1

Introduction

Computer vision is an academic field which is concerned with extracting high-level information from digital images. Digital image processing is another field which deals with processing such as image enhancement, image compression, etc. Both these fields have been active for decades and there has been a significant improvement in each of them. Cell phones and digital cameras use image processing scheme to generate high-quality images. Many computer vision-based detection algorithms such as face detection have been proposed, and their accuracy is close or even better than the human's performance. We believe there is still room for improvement. In fact, many of image processing and computer vision tasks can benefit from a deep learning approach. When Alex Krizhevsky and Geoffrey E. Hinton (Krizhevsky *et al.*, 2012) proposed AlexNet in ILSVRC competition in 2012 in object recognition task, computer vision and image processing researchers realized that there is a massive potential in the deep learning.

Neural network is not a new idea and there have been a large number of papers about it in the previous century (Kurenkov, 2015). There might be a question about

why deep learning is a hot topic today if it is not a new idea.

There are two reasons to answer this question. First, considerable amount of data (image, videos, text, wearable sensors, etc.) enabled deep learning to generate desirable results. These days millions of people are using modern electronic devices and a lot of data is created every single day. Therefore, the data-hungry deep learning algorithms are celebrating this data availability.

Another reason why deep learning algorithms work is due to computational power. Modern multi-core CPUs and GPUs gave high computational powers to researchers to help them in training their neural networks. Training a neural network with a larger training set used to take weeks or a months a few years ago. However, these days it takes only a few hours by running on GPUs.

1.1 Dual Camera Image Processing

Recently, dual cameras, which consist of two image sensors, have become a norm trend in smartphones due to the benefits they introduced in image enhancement, detection, and recognition tasks. In this section, we review a few examples of dual camera systems that have been presented by some leading companies.

1.1.1 Image enhancement

Some Huawei[1] smartphones have dual cameras which are used for image enhancement. The dual camera systems consist of a monochrome and a Bayer sensor which are utilized to enhance the captured images. A monochrome sensor captures all the light in the space in creating a digital picture. However, the output image is a gray-scale

[1] `www.huawei.com`

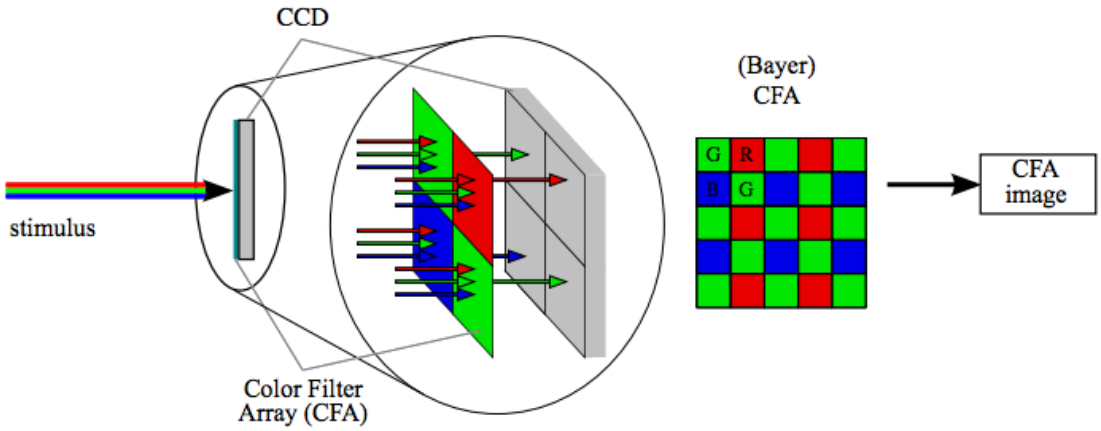


Figure 1.1: The Bayer filter mechanism (Losson *et al.*, 2010).

image. The intensities in the monochrome image show all the details and high-frequency information because the captured light is not being passed through a color filter array. In contrast, a Bayer sensor captures a gray-scale image (similar to the monochrome sensor) and creates a full-color image by passing the image through an image processing pipeline. The captured light passes through a filter called the Bayer filter located in front of the sensor, and a mosaiced image is produced. The Bayer filter assigns every pixel of the image sensor to a color component (e.g. R, G, B) based on a pre-determined pattern which is called Bayer pattern. Figure 1.1 shows Bayer filter mechanism.

The image processing pipeline of the camera applies a number of cascaded steps such as denoising, white balance, demosaicing, gamma correction on the mosaiced image to generate the full-color image. The demosaicing step generates the full-color image by interpolating the missing values in the main red, green, and blue components or more sophisticated methods (Li *et al.*, 2008).

The Huawei smartphones have a stereo image (monochrome and Bayer) to enhance the captured image quality. However, there is no publication or documentation about

Figure 1.2: The Bokeh effect captured by Apple Iphone using a dual camera system (https://support.apple.com/en-ca/HT208118).

how they perform the enhancement.

1.1.2 Depth Estimation

Apple[2] has presented another dual camera system to perform the Bokeh effect and optical zoom. Bokeh effect in photography is focussing on an interested region in an image and blurring other parts. Figure 1.2 shows the Bokeh effect produced by an Apple smartphone. The dual camera estimates the depth information of the scene and a segmentation algorithm uses the depth information to separate the foreground part from the background. Compact lenses used in the smartphones cannot distinguish the background from the foreground accurately due to lack of focal length. However, the dual camera system helps to simulate an optical lens with large focal length.

[2] `www.apple.com`

1.2 Problem Statement

In this thesis, we propose a convolutional neural network (CNN) approach for low-light stereo image enhancement. Therefore, the input to our model is a low-light stereo image pair (left and right), and the output is an enhanced-light image. Instead of using fusion or registration methods to align the left and right images, we exploit a CNN model for enhancing the under-exposed images. To our best knowledge, nobody has used a stereo image pair to improve the low-light condition using a CNN-based method. Our results show the efficiency of the proposed method.

1.3 Structure of the thesis

In chapter 2, we explained the fundamentals of the neural network, convolutional neural network, and applications of CNN in dual camera systems. Chapter 3 discusses the previous methods on low-light image enhancement using a single camera. In that chapter, we review state-of-the-art techniques and CNN-based methods. Chapter 4 discussed the proposed method for the low-light stereo image enhancement and chapter 5 states the conclusion and future work.

Chapter 2

Background

In this chapter, we provide the fundamental and essential knowledge of Deep Learning (DL) and Convolutional Neural Network (CNN) and review a few applications of CNN on dual camera systems.

2.1 Neural Network

Although biology inspires neural networks, it can be misleading to think that our brain (as a complex neural network) works exactly like NN models. The human brain contains approximately 100 billion neurons operating in parallel (Long and Gupta, 2008). However, artificial neural networks are designed by humans, and they are much simpler than the human brain's neurons.

A simple artificial neuron is shown in Figure 2.1. It takes an input vector $x \in \Re^n$ and calculates the dot product of input and its weights vector $w \in \Re^n$ and add the result to a scalar bias value b. The output is then fed to an activation function a which generates the neuron output. Therefore, we can formulate a neuron to a linear

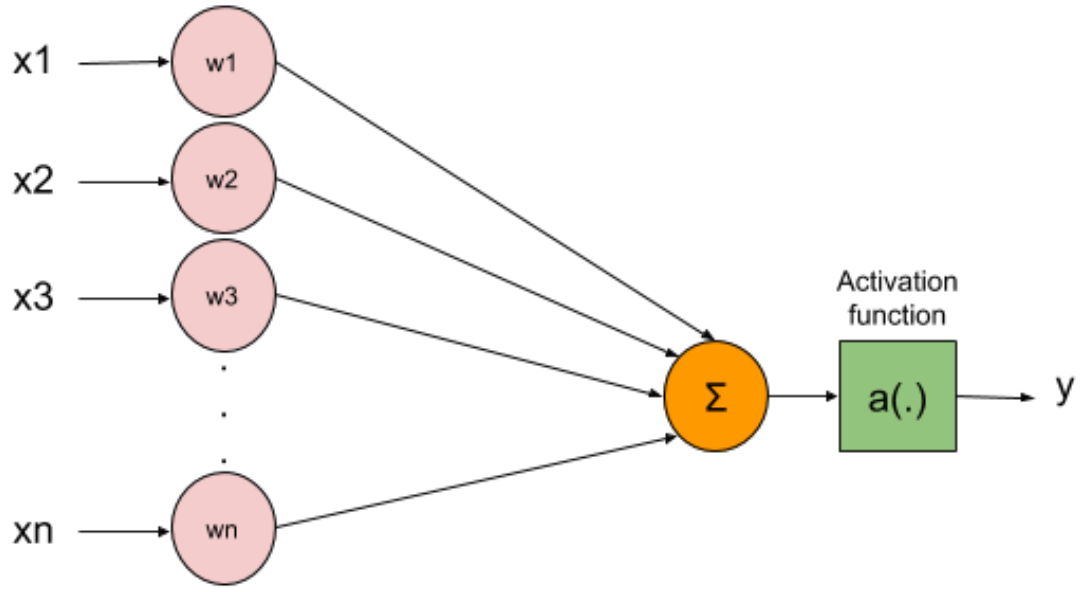


Figure 2.1: An artificial neuron

equation as the following:

$$y = a(w^T x + b) \tag{2.1}$$

where y is a scalar value and output of the neuron.

The primary keys in designing neural network are 1) selecting network architecture and 2) choosing the weight and bias values. The first one depends on the application and data-set. However, the second one is related to training the network which is a process to tune the weight values.

2.1.1 Multi-layer networks

Each neural network contains three types of layers: an input layer, some hidden layers, and an output layer. Figure 2.2 shows a typical fully-connected neural network.

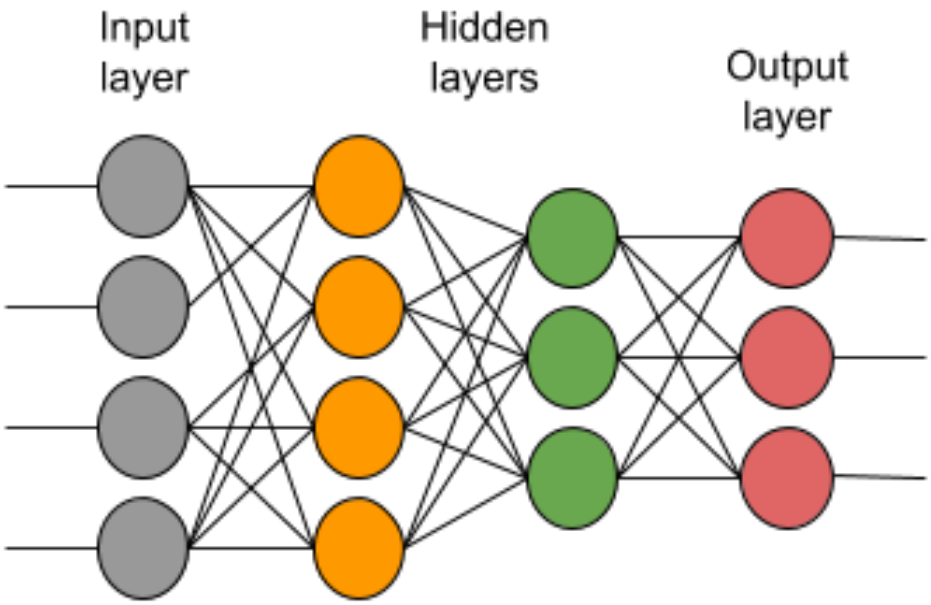


Figure 2.2: A fully-connected multi-layer neural network.

NN models estimate a complex (or simple) functions by using a training data-set. We can estimate any functions accurately if we have enough data and a well-designed network with a sufficient number of layers. During the training phase, we use an optimization algorithm to calculate the network weights by minimizing a loss function.

2.1.2 Forward Propagation

In general, feeding the network with the input and generating output is called forward propagation. Each neuron gets its input and passes its output to the next unit. In the testing phase, only forward propagation is performed. However, in the training phase, we need back propagation as well to tune the network parameters.

2.1.3 Back Propagation

Back-propagation is a process to calculate the weights in order to generate sound output. Basically, in this process, the network uses an optimization algorithm to minimize a cost function. The cost function depends on the application.

Gradient Descent is one of the famous algorithms used in the neural networks. It is an iterative and in some cases a time-consuming algorithm but works well enough in practice.

First, all the weights of the network are initialized with random values. Then, the algorithm takes an input and calculates the output by forward-propagation. Now, the output is compared to the desired or ground truth output using a loss function. The difference between the ground truth output and estimated output is called error value. The optimizer uses this error value to update the weights in a reverse (back)

order. In other words, it uses the error value and back propagates this error through the network. In this section, we discuss the optimization algorithm and how we can calculate the weights in more detail.

Similar to all optimization algoritms, we need a cost function $J(w)$ where w is the network's parameters. In order to minimize this cost function, we need an optimization algorithm. Usually NN models use different variations of Gradient Descent algorithm in the training phase. In this section we want to discuss the simple gradient descent algorithm and then explain the ADAM optimizer which we used and is used in many NN models these days too.

We assume the our cost function is defined as

$$J(w) = \sum_{i=1}^{N} (y_i - \hat{y}_i)^2 \tag{2.2}$$

where N is the number of training samples, y_i is the ground truth output, and $\hat{y}_i$ is the NN's output which depends on network's weights w. Thus, J is a function of our input and network weights. Since we cannot change our input data, we need to minimize our cost by changing the weights.

Gradient descent algorithm is a greedy algorithm and on each step, it goes in opposite direction of the cost function's gradient. In general, it convereges to a local minimum after a sufficient number of iterations. We are not going through every detail here, however, the reader can refer to (Ruder, 2016) for more details.

We need to calculate the gradient of the cost function with respect to every weight in the network. We start from the last layer (output layer) and calculate the gradient with respect to the weights of this particular layer. Then using the chain rule we can calculate the derivative of the cost function with respect to all the other weights and

parameters (Ruder, 2016).

The next step is about updating the weights. The general update rule is as follows

$$w := w - \alpha \frac{\partial J}{\partial w}, \tag{2.3}$$

where w is the weight vector we want to update, α is called *learning rate*, and $\frac{\partial J}{\partial w}$ is the derivate of the cost function with respect to the w.

Learning rate is a parameter that determines the step size of going in negative gradient direction. It is a hyper-parameter of our model. Large value of learning rate goes faster toward the local minimum while it might miss the minimum and causes some unstability in convergence process. In contrast, small values of learning rate make the convergence slow (Boyd *et al.*, 2004).

Although the gradient descent algorithm works in many problems, these days many researchers use an upgraded version of gradient descent which leads to a better result. In the next section, we discuss the ADAM optimizer.

ADAM Optimizer

ADAM Optimizer (Kingma and Ba, 2014) uses momentum technique in optimization of stochastic objective functions. It is computationally efficient, needs a small amount of memory, and is very suitable for problems with larget number of data or parameters.

ADAM algorithm uses a history of previous gradients and parameters. Roughly speaking, it uses a moving average of gradients and parameter values in previous steps. Figure 2.3 illustrates the ADAM optimizer.

The ADAM algorithm needs a few parameters and authors in (Kingma and Ba, 2014) suggest these values for machine learning algorithms: $\alpha = 0.001$, $\beta_1 = 0.9$,

Algorithm 1: *Adam*, our proposed algorithm for stochastic optimization. See section 2 for details, and for a slightly more efficient (but less clear) order of computation. g_t^2 indicates the elementwise square $g_t \odot g_t$. Good default settings for the tested machine learning problems are $\alpha = 0.001$, $\beta_1 = 0.9$, $\beta_2 = 0.999$ and $\epsilon = 10^{-8}$. All operations on vectors are element-wise. With β_1^t and β_2^t we denote β_1 and β_2 to the power t.

Require: α: Stepsize
Require: $\beta_1, \beta_2 \in [0, 1)$: Exponential decay rates for the moment estimates
Require: $f(\theta)$: Stochastic objective function with parameters θ
Require: θ_0: Initial parameter vector
$m_0 \leftarrow 0$ (Initialize 1st moment vector)
$v_0 \leftarrow 0$ (Initialize 2nd moment vector)
$t \leftarrow 0$ (Initialize timestep)
while θ_t not converged **do**
$t \leftarrow t + 1$
$g_t \leftarrow \nabla_\theta f_t(\theta_{t-1})$ (Get gradients w.r.t. stochastic objective at timestep t)
$m_t \leftarrow \beta_1 \cdot m_{t-1} + (1 - \beta_1) \cdot g_t$ (Update biased first moment estimate)
$v_t \leftarrow \beta_2 \cdot v_{t-1} + (1 - \beta_2) \cdot g_t^2$ (Update biased second raw moment estimate)
$\widehat{m}_t \leftarrow m_t / (1 - \beta_1^t)$ (Compute bias-corrected first moment estimate)
$\widehat{v}_t \leftarrow v_t / (1 - \beta_2^t)$ (Compute bias-corrected second raw moment estimate)
$\theta_t \leftarrow \theta_{t-1} - \alpha \cdot \widehat{m}_t / (\sqrt{\widehat{v}_t} + \epsilon)$ (Update parameters)
end while
return θ_t (Resulting parameters)

Figure 2.3: ADAM Optimization algorithm (Kingma and Ba, 2014)

$\beta_2 = 0.999$, $\epsilon = 10^{-8}$.

As you can see in Figure 2.3, vector m_t and v_t contain both the current gradient of cost function and the previous values. Both these values are used in the update rule of ADAM algorithm.

In summary, ADAM optimizer is used in problems with a significant amount of data, and high dimensional parameter spaces. It also combines the strength of two previous optimization algorithms:

- AdaGrad (Duchi *et al.*, 2011): The ability to deal with sparse gradients, and
- RMSProp (Hinton *et al.*, 2014): the ability to deal with non-stationary objectives.

Also, the ADAM uses a small amount of memory. Authors found ADAM algorithm robust and well-suited to a wide range of non-convex optimization problems in the field of machine learning (Kingma and Ba, 2014).

2.1.4 Activation Functions

The activation function can have different types. The following is a list of some activation functions:

- Identity $a(s) = s$
- Rectified linear units (ReLU) $a(s) = max(0, s)$
- Sigmoid $a(s) = \frac{1}{1+e^{-s}}$
- Hyperbolic tangent $a(s) = tanh(s)$

These functions are easy to compute the derivative for back-propagation process.

2.2 Convolutional Neural Networks

A neural network can have many hidden layers. In this case, it is often called deep neural network (DNN). Although multi-layer networks have been introduced since the 1980s (Goodfellow *et al.*, 2016), many reasons prevented the training of networks with many hidden layers.

The first problem is the curse of dimensionality (Goodfellow *et al.*, 2016). As the number of input's feature grows, the number of weights should increase. It means we need more data to train the network effectively which in many cases, it is hard to collect a significant amount of data; and more computational power and time.

To avoid this problem, *Convolutional Neural Networks* has been introduced. They are very useful for image processing or computer vision problems because images have a lot of information and researchers want to use all these information efficiently.

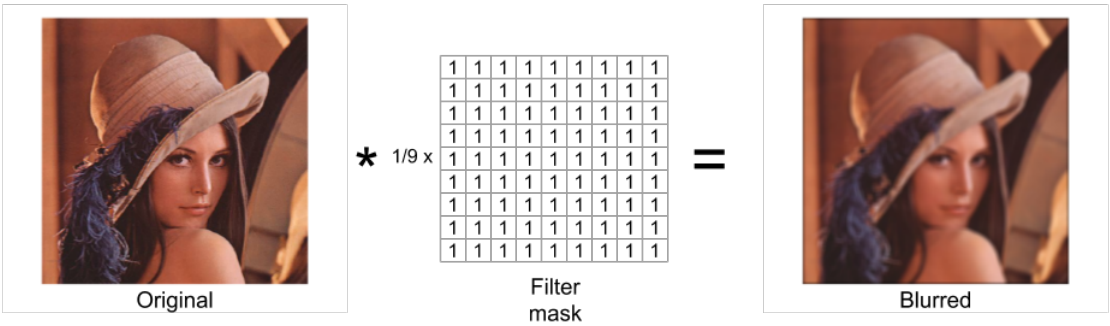


Figure 2.4: Input f, filter mask M, and output h for a blurring filter.

Instead of flattening images to use them in fully-connected neural networks, CNNs try to use the spatial information by convolving a filter with the input. In this section, we discuss more details about CNNs and how they work.

2.2.1 Basic Structure

In image processing, images can be filtered using *convolution* operation to produce different visible effect. Figure 2.4 shows the input and output to a simple blurring filter. The equation for the convolution is

$$h[x, y] = M[x, y] * f[x, y] = \sum_{n} \sum_{m} M[n, m] f[x - n, y - m] \tag{2.4}$$

where $h[x, y]$ represents the output value for row x and column y, f is the input image, and M is the filter mask. In fact, convolution looks like moving a sliding window (mask) on an input image and for each step it calculates the dot product of that overlap region and generates the output. For further information about convolution please refer to (Gonzalez and Woods, 2006).

A convolutional layer in a CNN contains a set of convolutional filters. All the

filters are considered as the layer's parameters and similar to the neurons in fully-connected layers. Therefore, each convolutional layer takes an input matrix (tensor) with size $h \times w \times c$ where h is the height, w is the width, and c is the number of input channels, and convolves the input with all its filters. The filter parameter for each convolutional layer is a matrix with size $f_h \times f_w \times f_c$ where f_h and f_w are the height and width of filters and f_c is the number of filters which are going to get convolved with the input and generates the output. The output size depends on two other parameters: stride and padding. Stride is a parameter which determines the jump size to the next pixel in the convolution operation, i.e., stride 2 means the convolution skip every other pixel in both horizontal and vertical directions. Thus the output size is halved in both directions. Another factor is how the input is padded while treating the border pixels. For border pixels, we can append some zero rows and columns to the input image and start the convolution operation from the first actual pixel, or we can start the convolution from the $\frac{f_h}{2}, \frac{f_w}{2}$ rows and columns of the input image in which the output image size shrinks. In general, the output size is

$$o = \frac{w - k + 2p}{s} + 1, \tag{2.5}$$

where o is the output height/width, w is the input height/width, k is the filter size, p is the padding, and s is the stride.

Since the same filters are used for all parts of the input image, the number of filter parameters is much smaller than the fully-connected network parameters. Moreover, CNNs take into account all the spatial information in images.

Cascading several convolutional layers forms a convolutional neural network. Figure 2.5 shows a CNN. The back-propagation process is used to tune the parameters

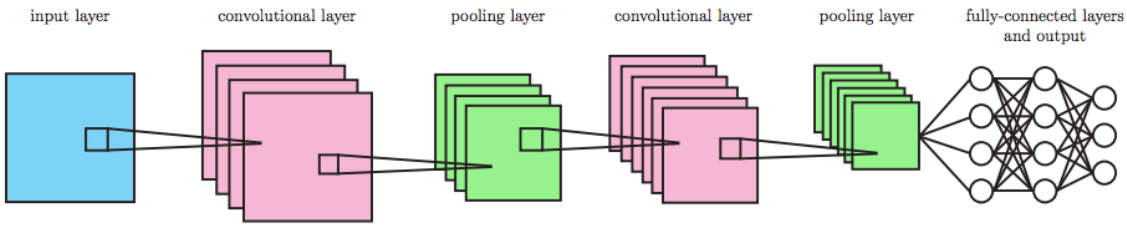


Figure 2.5: An example of a CNN (Stenroos, 2017)

like fully-connected NN models. In theory, the first layers of a CNN represent the low-level features of the input image such as edges, corners, and the last layers represent more complex features (Fukushima, 1988).

2.3 Overfitting and Regularization

Usually, there is a relation between the model complexity, i.e., number of neurons or number of layers, and the amount of training data-set and also performance of the model. We call a model overfitted if the training error is small but the testing/evaluation error is high. It means our model is biased to the training data-set and cannot generalize well and has a poor performance on test samples.

In this case, regularization can help us. Regularization means using a method to prevent the model to be overfitted. The simple version of regularization is adding a term to the cost function to penalize certain types of weights (Goodfellow *et al.*, 2016). Different applications and algorithms use different regularization terms.

Another technique for regularization is *dropout* (Srivastava *et al.*, 2014). Dropout tries to reduce the co-adaptation of neurons (Stenroos, 2017). This is achieved by randomly dropping out neurons during training. In other words, during training, we disable a neuron in one mini-batch run randomly and train the model without this

neuron. In each iteration, we disable a set of neurons, and in the next iteration, another set of neurons are disabled. This causes the network to not depend so much on a particular neuron. In convolutional networks, dropout is used in the last fully-connected layers (Simonyan and Zisserman, 2014).

2.4 Data Augmentation

Overfitting can be diminished by augmenting the training data-set. In many applications, acquiring data is expensive or time-consuming. However, using data augmentation techniques can help in having more training data. Data augmentation attempts to generate new samples with the current samples of training data. For example, while dealing with images as training samples, flipping, rotation, affine transformation, modifying contrast, brightness are some techniques to apply on the images of a data-set to generate new samples.

2.5 Application of CNN on Dual Camera Systems

In this section, we discuss two applications of CNN on dual camera systems. First, we review the depth estimation problem using CNN. Then, one of the recent work on how to perform super-resolution on stereo images is discussed.

2.5.1 Depth Estimation

In order to calculate the depth using a pair of stereo images, we need to compute the disparity between the two images first. The disparity for each pixel refers to the difference in horizontal location of an object in the left and right image (Zbontar and

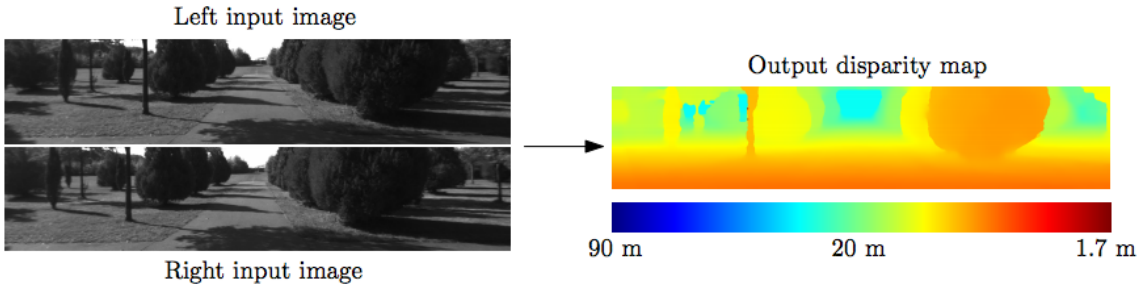


Figure 2.6: The input pair (left and right camera) images and the output disparity map. Note that objects closer to the camera have larger disparities than objects farther away. Larger disparities are shown with warmer colors. (Zbontar and LeCun, 2016)

LeCun, 2016). In other words, for a pixel at (x, y) in the left image, the corresponding pixel in the eight image is $(x - d, y)$ where d indicates the disparity. In this problem, the stereo images are aligned vertically, and we assume there is no disparity in the y dimension. These two images are called rectified.

If we know the disparity for every pixel, we can calculate the depth with the following equation

$$z = \frac{fB}{d} \tag{2.6}$$

Where f is the focal length of the camera and B is the distance between the camera centers (Zbontar and LeCun, 2016). Figure 2.6 illustrates the input and output of a depth/disparity estimation algorithm.

The described problem of stereo matching is important in many fields such as autonomous driving, robotics, intermediate view generation, and 3D scene reconstruction (Zbontar and LeCun, 2016). The author of (Scharstein and Szeliski, 2002) determined the structure of a stereo matching algorithm. Every stereo matching algorithm has four steps: matching cost computation, cost aggregation, optimization,

and disparity refinement. The first two steps referred to matching cost that is the primary focus of this section.

The work done by (Zbontar and LeCun, 2016) is the first work that applied CNNs to disparity matching (stereo matching) problem. Here we explain this work in more details.

Matching Cost

A simple method for calculating the matching cost is the sum of absolute differences at each position p for all disparities d under consideration:

$$C_{SAD}(p,d) = \sum_{q \in N_p} |I^L(q) - I^R(q-d)| \tag{2.7}$$

where $I^L(p)$ and $I^R(p)$ are image intensities at position p in the left and right image and N_p is the set of locations withing a fixed rectangular window centered at p.

Equation (2.7) can be considered as measuring the cost corresponding to matching a patch from the left image, centered at position p, with a patch from the right image, centered at position $p-d$. The goal is to minimize this equation for two patches that are centered around the image of the same 3D point. In contrast, we want to maximize this cost for two patches that are not associated with the same 3D point.

We can attempt to solve this problem by extracting good (matched) and bad (not matched) patches from publicly available data-sets (for example KITIT[1] and Middlebury[2] data-sets) and training a CNN on these extracted patches. In fact, the training phase is a two-class classification problem: determining a pair of patch is

[1] `http://www.cvlibs.net/datasets/kitti/`
[2] `http://vision.middlebury.edu/stereo/data/`

matched (class 0) or not matched (class 1).

Constructing the Data Set

The authors of (Zbontar and LeCun, 2016) used the Middlebury (Scharstein *et al.*, 2014) and KITTI (Geiger *et al.*, 2012) data-sets in this experiment. These data-sets provide the input left and right images and the corresponding disparities for both left and right images. KITTI data-set used a LIDAR and Middlebury used a structured light to acquire the ground truth disparities.

To train the CNN, authors in (Zbontar and LeCun, 2016) created binary classification data-sets: matching patches (class 0) and mis-matching patches (class 1). At each position where the actual disparity is known, they extract one negative and one positive training sample. This ensures that two classes have the same number of samples. A positive sample consists of two image patches from an image pair that the centers of the two patches correspond to the same 3D point, while a negative example is a pair of patches where this is not the case.

Let $< P^L_{n\times n}(p), P^R_{n\times n}(q) >$ denote a pair of patches, where $P^L_{n\times n}(p)$ is an $n\times n$ patch from the left image centered at position $p = (x, y)$, $P^R_{n\times n}(q)$ is an $n \times n$ patch from the right image centered at position q, and d denoted the true disparity at position p. A negative example is obtained by setting the center of the right patch to

$$q = (x - d + o_{neg}, y), \tag{2.8}$$

where o_{neg} is chosen from a pre-determined range which makes sure that the two patches are not matched. A positive example is similar to the negative ones derived

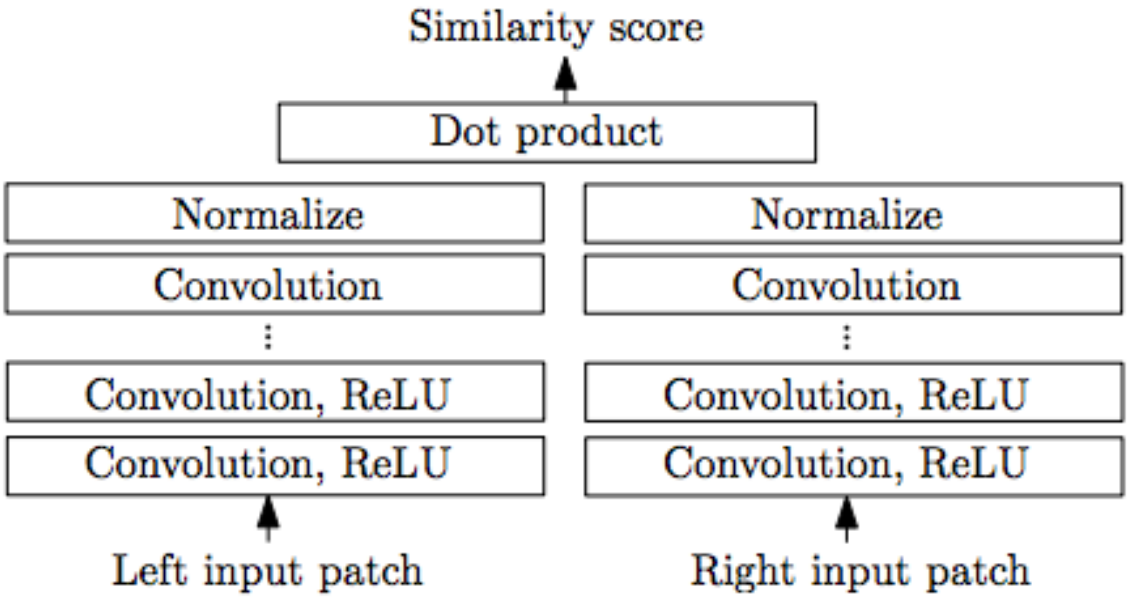


Figure 2.7: The fast architecture proposed by (Zbontar and LeCun, 2016).

by setting

$$q = (x - d, y). \tag{2.9}$$

Network Architecture

Authors in (Zbontar and LeCun, 2016) proposed two network architecture to estimate the disparity mapping. The first architecture is faster but generates less accurate results while the next one is slower and generates more accurate results. The fast architecture uses a fixed similarity measure to compare the two feature maps that are extracted from the first part of CNN, while the accurate one attempts to learn a desired similarity measure on the extracted feature maps. Figure 2.7 and 2.8 indicate the fast and accurate architectures respectively.

The fast architecture is a siamese network. Siamese network is a two shared-weight sub-networks joined at the head proposed by Bromley *et al.* (1994). The sub-networks

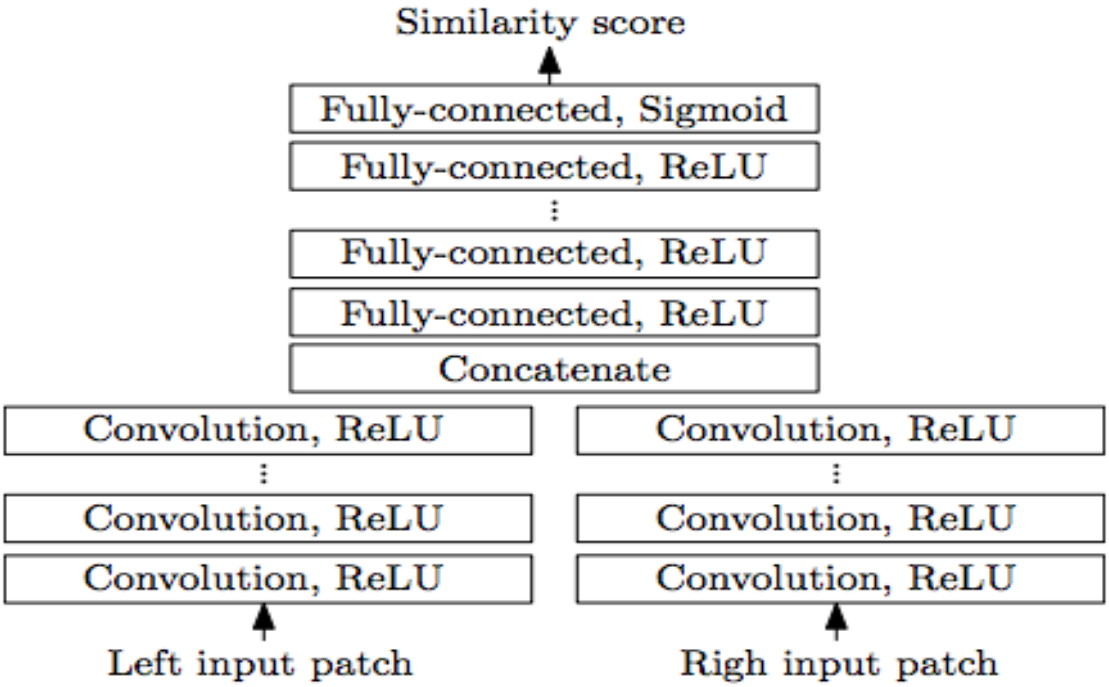


Figure 2.8: The accurate architecture proposed by (Zbontar and LeCun, 2016).

include some convolutional layers following ReLU layer. Both sub-networks produce a feature map for the two input patches. The two feature maps are compared using a dot product, and a similarity score is generated.

The loss function used in the fast network is a hinge loss:

$$J = max(0, m + s_- - s_+) \tag{2.10}$$

where s_+ is the output for a positive example, s_- is the network output for a negative example, and m is a constant margin parameter which is set to 0.2. The loss is zero when the similarity of positive example is greater than the similarity of negative example by at least the margin m.

The accurate architecture is similar to the fast architecture replacing the dot product with another sub-network. The input for this sub-network is the concatenation of the feature maps of the left and right sub-networks. Basically, this new sub-network learns the similarity score between input patches.

Authors used the binary cross-entropy loss for training:

$$J = tlog(s) + (1-t)log(1-s) \tag{2.11}$$

where t is the true label for the input patch pair (1 for positive and 0 for negative examples) and s is the output of the network.

Computing the Matching Cost

They used the output of CNN (fast or accurate) network as an initialization for the matching cost:

$$C_{CNN}(p,d) = -s(< P^L(p), P^R(p-d) >), \tag{2.12}$$

where $s(< P^L(p), P^R(p-d) >)$ is the output of the network when run on input patches $P^L(p)$ and $P^R(p-d)$ for a set of possible d values. The minus sign is for converting the similarity score to the matching cost.

To calculate the matching cost for a pair of images, the authors suggest to run the sub-networks once for each image and run the fully-connected layers (accurate architecture) d times, where d is the maximum disparity under consideration. For the fast architecture, the dot product operation is run d times likewise.

Authors in (Zbontar and LeCun, 2016) proposed a few post-processing steps to create the final disparity map which we skip here due to being irrelevant to our

Table 2.1: The top ten methods on the Middlebury data-set as of October 2015. The "Error" column is the weighted average error after upsampling to full resolution and "Runtime" is the time, in seconds, required to process one pair of images (Zbontar and LeCun, 2016).

Rank	Method		Resolution	Error	Runtime
1	**MC-CNN-acrt**	Accurate architecture	Half	8.29	150
2	MeshStereo	Zhang et al. (2015)	Half	13.4	65.3
3	LCU	Unpublished work	Quarter	17.0	6567
4	TMAP	Psota et al. (2015)	Half	17.1	2435
5	IDR	Kowalczuk et al. (2013)	Half	18.4	0.49
6	SGM	Hirschmuller (2008)	Half	18.7	9.90
7	LPS	Sinha et al. (2014)	Half	19.4	9.52
8	LPS	Sinha et al. (2014)	Full	20.3	25.8
9	SGM	Hirschmuller (2008)	Quarter	21.2	1.48
10	SNCC	Einecke and Eggert (2010)	Half	22.2	1.38

Figure 2.9: A scene example from Middlebury data-set (Zbontar and LeCun, 2016).

application.

Table 2.1 indicates the performance of the proposed method by (Zbontar and LeCun, 2016) on Middlebury data-set. We did not cite other methods on this table as this is just the review of a CNN application on stereo images. However, readers can find more details in this paper (Zbontar and LeCun, 2016). Figure 2.9 shows the input and ground truth disparity map for an image scene of Middlebury data-set and Figure 2.10 shows the output of the CNN for both fast and accurate architecture.

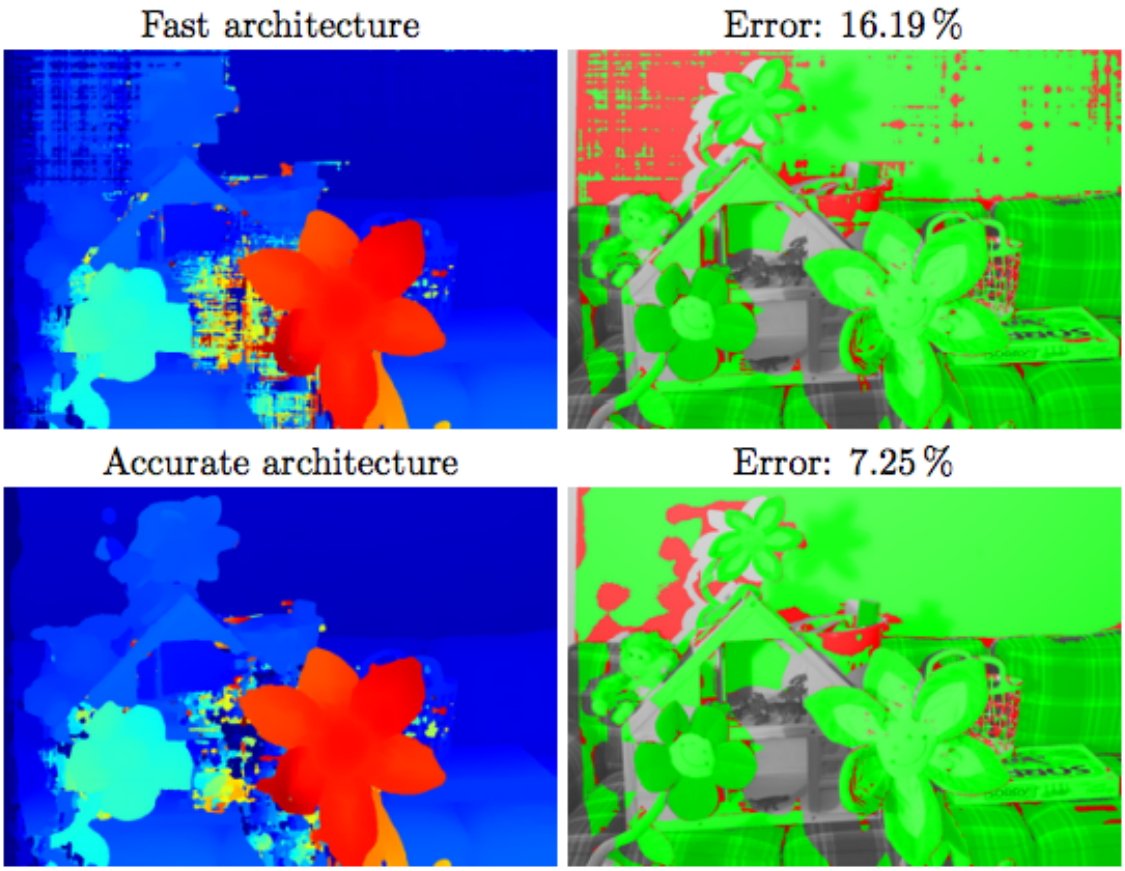


Figure 2.10: The fast and accurate CNN output for an example of Middlebury dataset (Zbontar and LeCun, 2016).

2.5.2 Super Resolution

Another application of CNN on image enhancement is super-resolution. In general, super-resolution is improving the quality of an image regarding resolution. This section discusses a proposed algorithm by (Jeon *et al.*, 2018) for super-resolution of stereo images using convolutional neural networks.

The disparity between two stereo images is much larger than one pixel due to having two images in a stereo pair. While the disparity in stereo images enables researchers to calculate the depth, using the disparity to register the two images pixel-wise is not sufficient due to the low-resolution of disparity estimation. Therefore, authors in (Jeon *et al.*, 2018) proposed a method that learns a subpixel parallax prior to enhance the spatial resolution of stereo images.

Super-resolution algorithms have been studied over decades (Park *et al.*, 2003). Here we briefly review some of the previous super-resolution algorithms.

Multi-Frame Super-Resolution: The traditional super-resolution algorithms used to enhance the resolution of an image using multiple images from the same camera with a little jitter between the input images (Park *et al.*, 2003). However, since all the images are taken with one camera, these methods are restricted to only static scenes.

Single-Image Super-Resolution: These methods attempt to improve the spatial resolution by using a single input image. They are divided into two categories: 1) example-based methods, 2) deep learning-based methods.

Example-based methods use similarities of different small patches in a single image or learn a dictionary from a different high-resolution low-resolution related image database. For instance, (Yang *et al.*, 2010) proposed a sparse representation-based

method that learns a joint dictionary from pairs of low- and high-resolution training data-sets.

Recently, example-based methods have been replaced by deep learning-based methods. For instance, (Dong *et al.*, 2016) proposed a convolutional neural network for super-resolution (SRCNN). There have been a few other works too which we refer the reader to (Jeon *et al.*, 2018).

Stereo Image Enhancement: The most challenge part of stereo image enhancement is the disparity between the two images. (Bhavsar and Rajagopalan, 2010) and (Park *et al.*, 2012) exploit stereo block matching to search pixel correspondences which are used to register two stereo input images.

Authors in (Jeon *et al.*, 2018) proposed a direct end-to-end mapping from a stereo image of low-resolution to a high-resolution image. The main idea behind their algorithm is the disparity between the two stereo images provides a sub-pixel parallax which enables them to have a sub-pixel resolution (Figure 2.11). This deep network approach enhances image resolution by using this knowledge to generate a high-resolution image without estimating disparity.

Network Architecture

Authors in (Jeon *et al.*, 2018) proposed a super-resolution neural network architecture to learn the parallax between the two left and right images. They divide the images into two luminance and chrominance parts with converting the RGB image into a YC_rC_b image. Thus, we have a luminance (Y) and a chrominance (C_rC_b) network to process each component. Figure 2.12 shows the two network architectures.

The luminance network takes the Y component of the left patch and Y component

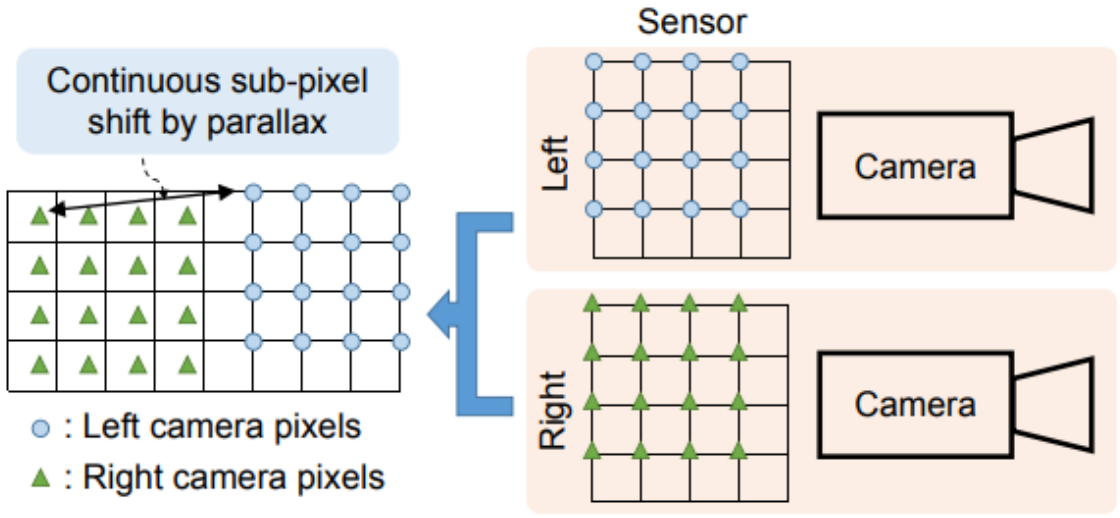


Figure 2.11: Registering stereo images with parallax (Jeon *et al.*, 2018).

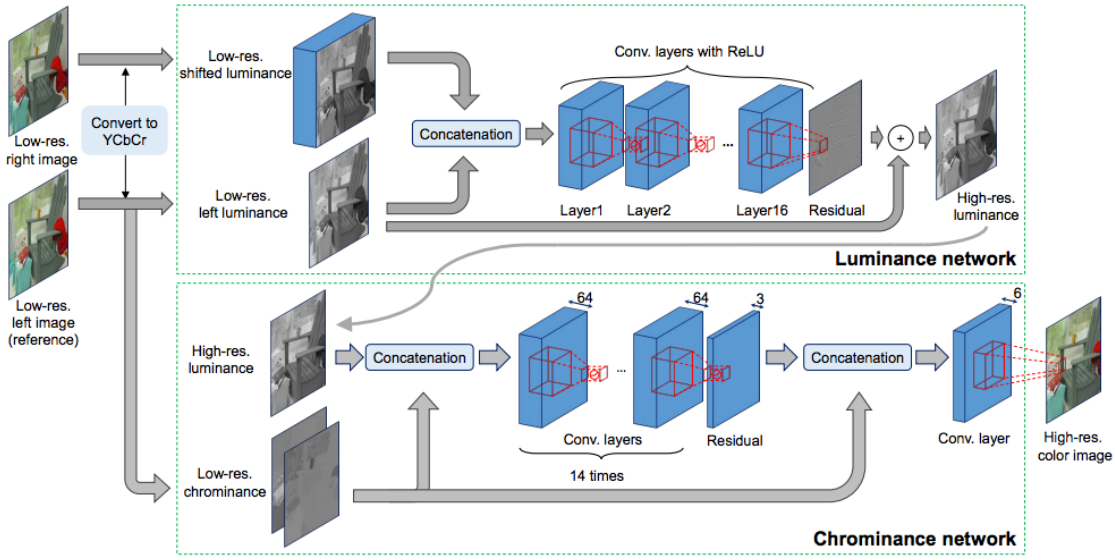


Figure 2.12: Proposed network architecture by (Jeon *et al.*, 2018)

of 64 shifted right patches as inputs. Then, it generates a high-resolution of the luminance component. The chrominance network then uses this estimated high-resolution patch and concatenates it to the low-resolution chrominance part of the left patch and generates the high-resolution full-color image.

As it can be seen in the Figure 2.12, both the luminance and chrominance networks are using a residual network (Kim *et al.*, 2016). A residual luminance of an image is $r_L = y_L - x_{1,L} \in \Re^2$, where y_L is a high resolution luminance image and $x_{1,L}$ is the luminance of left low-resolution stereo image.

The first network learns the residuals r_L^i between a high-resolution luminance y_L^i and a low-resolution luminance stereo image $x_{1,L}^i$ over training data-sets. Therefore, the input to the luminance network is:

$$\begin{aligned} \tilde{X}_{L,j}^i(x,y) &= x_{2,L}^i(x-\phi(j),y) && for \quad j \in \{1 \cdots M\}, \\ \tilde{X}_{L,j}^i(x,y) &= x_{1,L}^i(x,y) && for \quad j = M+1 \end{aligned}$$

where $\phi(x)$ is a shifted offset of the j-th channel, and M is the number of shifts.

The loss function used fo the luminance network is the mean squared errors of $\{\frac{1}{2}||r_L^i - f(\tilde{X}_L^i)||^2\}$ where f is the output of the luminance network.

The input for the chrominance network is:

$$\begin{aligned} \tilde{x}_c^i &= \hat{y}_L^i && for \quad c = 1 \\ \tilde{x}_c^i &= x_{1,c}^i && for \quad c \in \{2,3\}, \end{aligned}$$

where $\hat{y}_L^i$ is the output of luminance network, and $x_{1,c}^i$ are the two chrominance components of left patch from the training data-sets. Similar to the luminance network,

Table 2.2: Quantitative evaluation of proposed method by (Jeon *et al.*, 2018) on Middlebury data-set.

Dataset	Scale	Bicubic		SRCNN		(Jeon *et al.*, 2018)	
		PSNR	SSIM	PSNR	SSIM	PSNR	SSIM
Middleburry	x2	29.64	0.9228	31.48	0.9505	**33.05**	**0.9545**
	x3	27.20	0.8737	28.76	**0.9136**	**29.59**	0.8974
	x4	25.79	0.8344	**27.11**	**0.8814**	26.80	0.8495

the chrominance network uses the residual learning, and the loss function for this network is $\{\frac{1}{2}||r^i - g(\tilde{x}^i)||^2\}$ where function g is the residual of chrominance network.

Authors evaluate the results of their method with quantitative and qualitative comparisons. For comparison, they choose state-of-the-art single-image superresolution methods: super-resolution convolutional neural network (SRCNN) (Dong *et al.*, 2016). To quantitatively evaluate results, they first created test datasets by downsampling them by 2, 3 and 4. Then they upscale them by the magnification ratios of 2, 3 and 4. Figure 2.13 shows super-resolution results of the magnification ratio of 2 on the Middlebury test dataset. The proposed method by (Jeon *et al.*, 2018) outperforms other state-of-the-art single-image methods. Table 2.5.2 provides the average PSNRs and SSIMs on each benchmark dataset. This method achieves the highest PSNR and SSIM values in most cases when compared with the state-of-the-art methods. For more results, refer to the original paper (Jeon *et al.*, 2018).

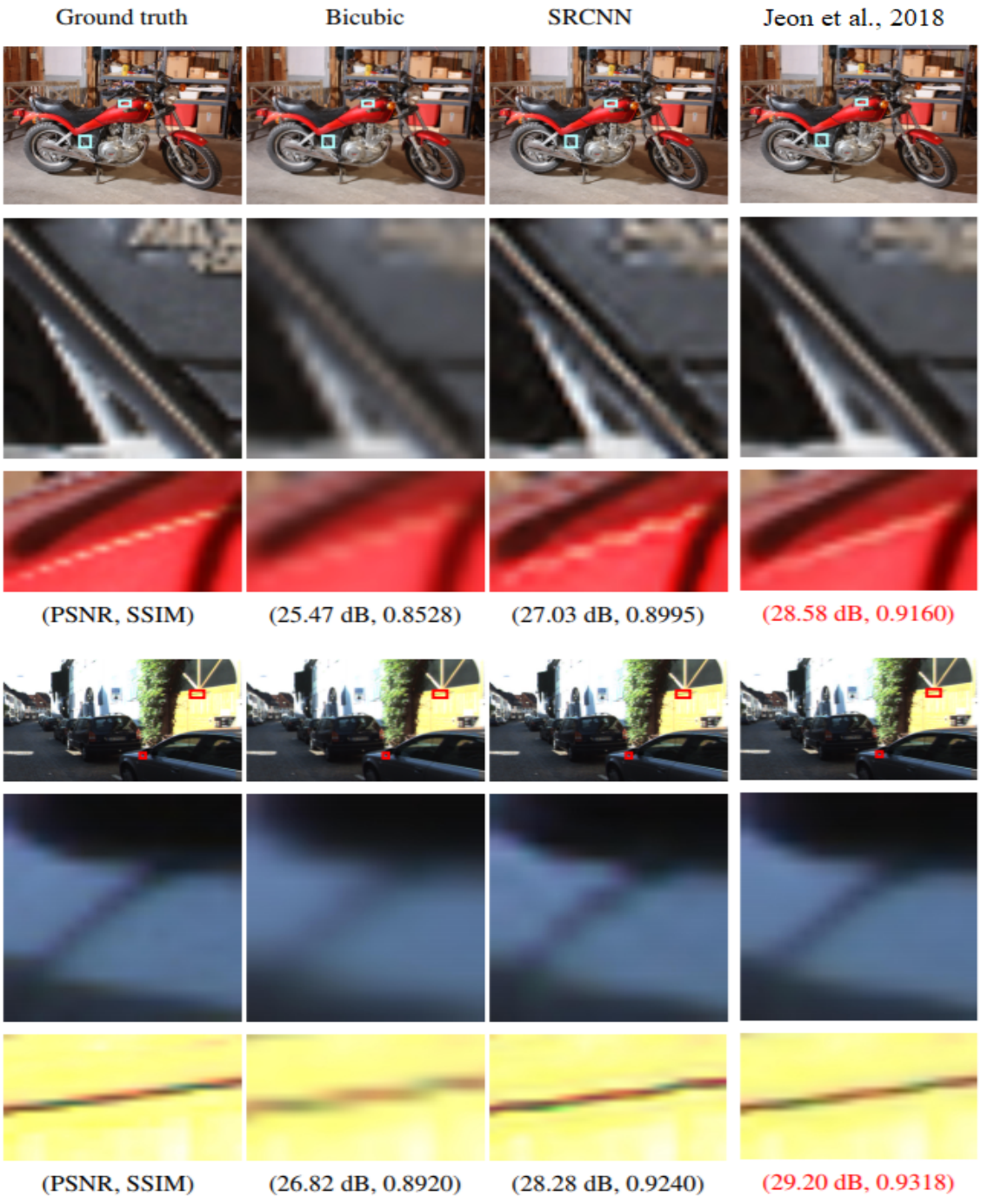


Figure 2.13: Results of super-resolution with the x2 factor (Jeon *et al.*, 2018).

Chapter 3

Low-Light Image Enhancement

3.1 Introduction

This chapter explains how to use deep learning in enhancing the low-light condition in images. A few works have been done on low-light condition enhancement and as far as we know, there is no paper or research on using stereo images on low-light image enhancement. In the next chapter we will explain our proposed method on how we can generate a normal light image using a stereo image pair which suffers from the low-light condition.

In this chapter, we discuss the state of the art methods for Gamma correction and image histogram equalization and explore their drawbacks first. Then, a recent work on low-light enhancement using single camera will be discussed.

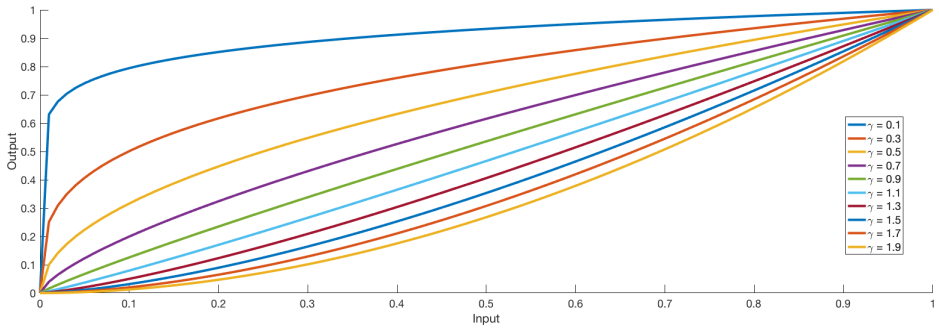


Figure 3.1: Gamma transformation function with different number of γ.

3.2 Gamma Correction

Gamma correction is a simple method and expands an image histogram to produce a brighter image. In other words, the gamma correction method enhances the low-contrast images by increasing the small values and compressing the large values to generate a higher contrast image.

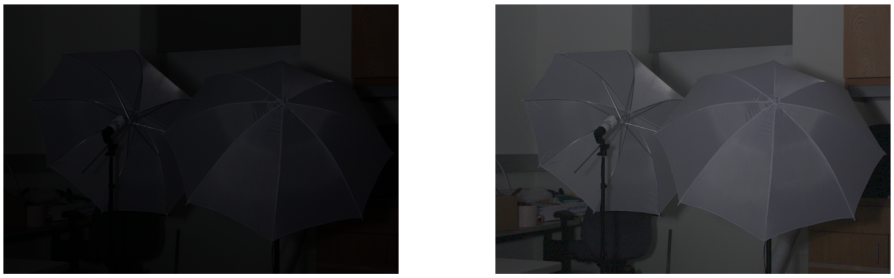
The gamma correction uses the following simple equation:

$$I_{out} = I_{in}^{\gamma} \tag{3.1}$$

where I_{out} and I_{in} represent pixel values of output and input images respectively and γ is a parameter that controls the image contrast.

Let's assume that the pixel values of an input image are in $[0, 1]$ range. If γ is less than one, the values close to zero increase more than the values close to one. Figure 3.1 shows the gamma function for different values of γ. Therefore, to have a bright input image, we select γ less than one.

Figure 3.2 shows the input and output of the gamma transformation for $\gamma = 0.5$.



(a) Input Image (b) Output Image

Figure 3.2: Input and output of gamma transformation for $\gamma = 0.5$

The gamma correction method is simple and does not consider the relation between color channels and neighbor pixels in the calculation of the output. In other words, it enhances the input image globally, which produces noise in the output image and also choosing the best gamma can be challenging.

3.3 Image Histogram Equalization

3.3.1 Histogram

A *histogram* for a digital image with gray levels in the range $[0, L]$ is a discrete function $h(r_k) = n_k$, where r_k is the kth gray level and n_k is the number of pixels in the image having gray level r_k. In addition, a normalized histogram is given by $p(r_k) = n_k/n$, for $k = 0, 1, \cdots, L-1$. Roughly, $p(r_k)$ gives us an estimate of *probability distribution function* (PDF) of the image (Gonzalez and Woods, 2006).

Histograms are the basis for many image enhancement techniques, as well as image histogram equalization. Histogram of an image is simple to calculate in software and hardware. Therefore, it is a popular tool for real-time image processing (Gonzalez

and Woods, 2006).

Basically, a histogram shows the contrast in an image. If the histogram for an image is distributed over all values in the range $[0, L-1]$, it means generally that all the pixels have a wide range value; thus the contrast in the image is high. However, all pixel values in a low contrast image belong to a narrow range of numbers.

3.3.2 Histogram Equalization

Let the variable r represent the gray levels of an image to be enhanced. It is assumed that $r \in [0, 1]$ where $r = 0$ and $r = 1$ represent black and white respectively.

Histogram equalization transform can be defined as the following

$$s = T(r) \tag{3.2}$$

where $0 \leq r \leq 1$. This transform produces a level s for every pixel in the original image.

$T(r)$ must have a few conditions to enhance an image. First, it should be a single-valued function to guarantee that the inverse transformation will exist. It is also needed to be a monotonic function because we should preserve the increasing order from black to white in the output image. The last condition on $T(r)$ specifies that $0 \leq T(r) \leq 1$ where $0 \leq r \leq 1$.

Let $p_r(r)$ and $p_s(s)$ denote the PDF of random variable r and s respectively. It can be shown that if $p_r(r)$ and $T(r)$ are known and $T^{-1}(s)$ satisfies the mentioned conditions on $T(r)$, then the PDF of $p_s(s)$ can be obtained as the following:

$$p_s(s) = p_r(r)|\frac{dr}{ds}|. \tag{3.3}$$

Letting $s = T(r) = \int_0^r p_r(w)dw$ indicates the transformation for histogram equalization. Therefore,

$$\begin{aligned} \frac{ds}{dr} &= \frac{dT(r)}{dr} \\ &= \frac{d}{dr}[\int_0^r p_r(w)dw] \\ &= p_r(r). \end{aligned}$$

Now, if we substitute the above equation into (3.3), and note that all probability values are positive, we have

$$\begin{aligned} p_s(s) &= p_r(r)|\frac{dr}{ds}| \\ &= p_r(r)|\frac{1}{p_r(r)}| \\ &= 1, 0 \leq s \leq 1 \end{aligned}$$

Thus the PDF for variable s (output image) is 1 for every $s \in [0, 1]$ which is a uniform PDF.

Since the images are digital and their pixels have discrete values, the PDF of a discrete random variable is defined as the following:

$$p_r(r_k) = \frac{n_k}{n}, \qquad k = 0, 1, 2, \cdots, L-1. \tag{3.4}$$

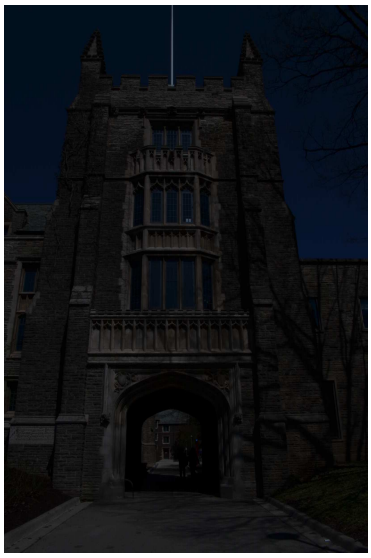
Therefore, the transformation would be:

$$\begin{aligned} s_k = T(r_k) &= \sum_{j=0}^{k} p_r(r_j) \\ &= \sum_{j=0}^{k} \frac{n_j}{n} \qquad k = 0, 1, \cdots, L-1 \end{aligned}$$

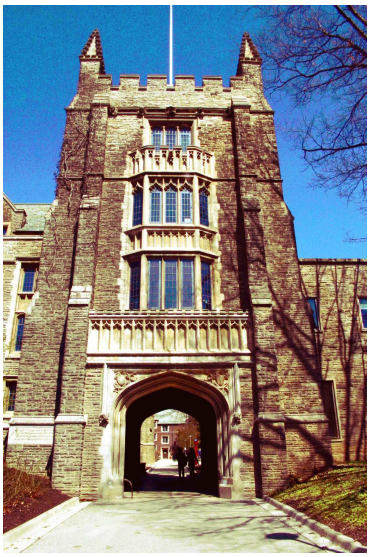
Unlike the continuous transformation, it cannot be proved in general that the discrete transformation will generate the discrete equivalent of a uniform PDF. However, it tends to create a highly distributed PDF (histogram).

Figure 3.3 shows the results for histogram equalization implemented for a sample image of McMaster University campus. As illustrated in Figure 3.3(c), the histogram concentrates on dark values. However, the histogram of the enhanced image in Figure 3.3(d) distributed in all the range of values. The red, green, and blue curves in these figures show the three main color component of the image.

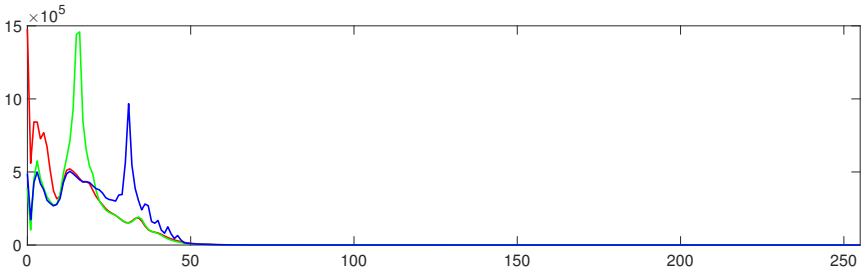
According to the Figure 3.3(b), image histogram equalization algorithm produces noise to the output image which affects the output image quality. Moreover, this algorithm enhances the input image (low-light condition) globally and does not take into account the spatial locality. This method also does not consider the relation between the color components. However, this method is a straightforward algorithm which is a baseline for other enhancement algorithms.



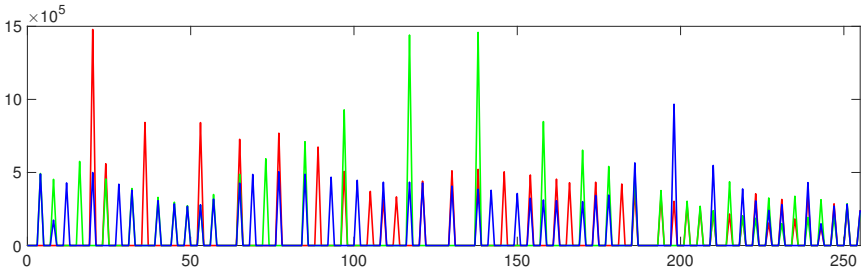
(a) Original Image



(b) Enhanced Image



(c) Histogram of the original image



(d) Histogram of the enhanced image

Figure 3.3: Histogram equalization results implemented by MATLAB.

3.4 Low-Light Image Enhancement Using Deep Convolutional Network

In this section, a method proposed by (Shen *et al.*, 2017) is discussed. Authors introduced a CNN based end-to-end model to enhance the image quality in low-light conditions.

Historically, the low-light image enhancement methods can be divided into two categories: histogram-based and Retinex-based methods. In the previous section, we explained a histogram-based method, while in this section an enhancement method based on CNN and Retinex theory (Land, 1977) is proposed.

3.4.1 Algorithm and Network Architecture

The proposed algorithm by (Shen *et al.*, 2017) has three components: *Multi-scale Logarithmic Transformation*, *Difference-of-convolution* and *Color Restoration Function*. The following paragraphs explain these three parts in details.

Formally, let X denote the input low-light image and denote the corresponding bright image as Y. Suppose f_1, f_2, f_3 represent the three mentioned components. Thus, the proposed network is the composition of three functions:

$$f(X) = f_3(f_2(f_1(X))) \tag{3.5}$$

Multi-scale Logarithmic Transformation: $f_1(X)$ takes the input low-light image X and computes the same size output. First, the input image is brightened by

several logarithmic transformations:

$$M_j = log_{v_j+1}(1 + v_j.X), j = 1, 2, \cdots, n, \tag{3.6}$$

where M_j denotes the output of j^{th} scale with the logarithmic base $v_j + 1$, and n is the number of logarithmic transformation function. Next, these 3D tensors $(height \times width \times 3channels)$ are concatenated together.

$$M = [M_1, M_2, \cdots, M_n]. \tag{3.7}$$

Now, this larger 3D tensor is fed to two consecutive convolutional layers:

$$f_1(X) = max(0, M * W_{-1} + b_{-1}) * W_0 + b_0 \tag{3.8}$$

The first convolutional layer shrinks the input channel $3n$ to 3 and uses a ReLU activation function. The second convolutional layer has no activation function.

This part is mainly designed to get a better image via weighted sums of multiple logarithmic transformations, which accelerates the convergence of the neural network (Shen *et al.*, 2017).

Difference-of-convolution: function f_2 takes the output of function f_1 as input and computes the same size output. First, the input goes through a few convolutional layers.

$$H_0 = f_1(X) \tag{3.9}$$

$$H_m = max(0, H_{m-1} * W_m + b_m), m = 1, 2, \cdots, K \tag{3.10}$$

Where K is a hyper-parameter in this algorithm and denotes the number of convolutional layers in this part. All these H_m outputs are concatenated together, and the larger tensor goes through another convolutional layer without activation function:

$$H = [H_1, H_2, \cdots, H_K] \tag{3.11}$$

$$H_{K+1} = H * W_{K+1} + b_{K+1}. \tag{3.12}$$

The last layer is equivalent to weighted averaging of these K images. The final output is:

$$f_2(f_1(X)) = f_1(X) - H_{K+1}. \tag{3.13}$$

Color Restoration Function: This step uses a convolutional layer to enhance the color components:

$$f_3(f_2(f_1(X))) = f_2(f_1(X)) * W_{K+2} + b_{K+2} \tag{3.14}$$

Figure 3.4 shows a low-light image and the results of f_1, f_2, and f_3. Figure 3.5 illustrates the proposed network by (Shen *et al.*, 2017).

Authors in this work used the regularized Frobenius norm as the loss function:

$$L = \frac{1}{N} \sum_{i=1}^{N} ||f(X_i) - Y_i||_F^2 + \lambda \sum_{i=-1}^{K+2} ||W_i||_F^2 \tag{3.15}$$

where N is the number of training samples, and λ denotes the regularization parameter.

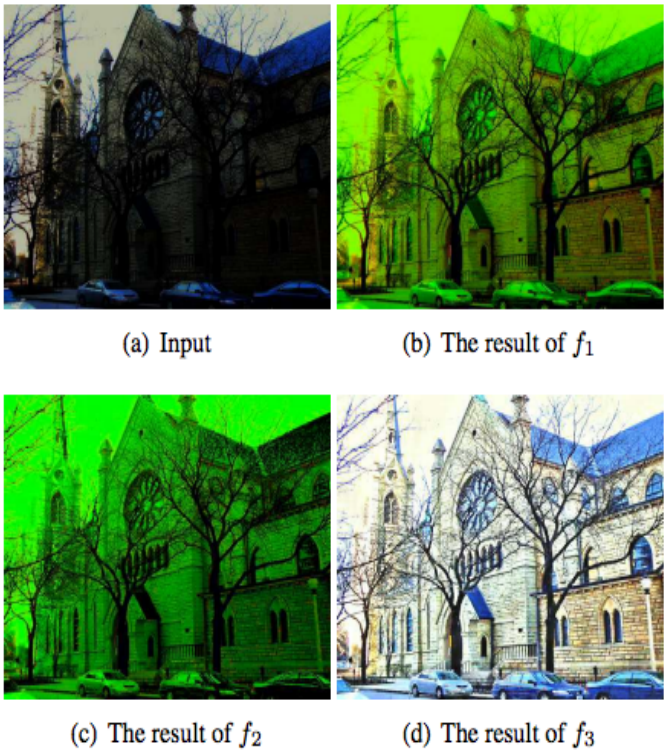


Figure 3.4: A low-light image and the results of f_1, f_2, f_3 (Shen *et al.*, 2017).

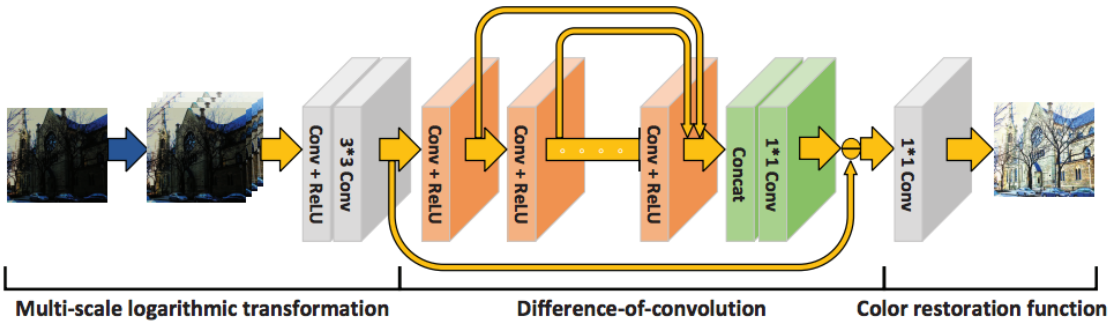


Figure 3.5: The architecture of MSR-net (Shen *et al.*, 2017).

Figure 3.6: ground truth, input, and output of the method proposed by (Shen *et al.*, 2017).

3.4.2 Traning Dataset and Results

To generate a training data-set, authors selected more than 10,000 images from the UCID data-set (Schaefer and Stich, 2003), BSD data-set (Arbelaez *et al.*, 2011), and Google image search. For each image, the brightness and contrast are reduced manually to create low-light images. Finally, about one million 64×64 high-quality/low-quality patch pair extracted.

To train the network ADAM optimization algorithm is used for 300K iterations. The number of logarithmic transformation function $n = 4$ and $v = 1, 10, 100, 300$ have been chosen.

Figure 3.6 shows the visual comparison of three synthesized low light images. The output is highly natural and has no artificial artifacts.

Chapter 4

Dual Camera Low-Light Image Enhancement

This chapter presents the technical developments of this thesis. First, we define the problem precisely, and then propose our solution and analyze our results.

4.1 Problem Definition and Motivation

Our objective is to enhance the low-light images using convolutional neural networks in dual camera setup. CNNs have been exploited in recent years for many image restoration applications such as super-resolution, denoising, and low-light enhancement (Sindagi and Patel, 2018) due to the availability of large amounts of images and high computational power.

In chapter 3, we reviewed previous works done on the enhancement of low-light images in single camera setup. However, to the best of our knowledge, this thesis is the first work of using CNNs on the low-light enhancement of stereo images in dual

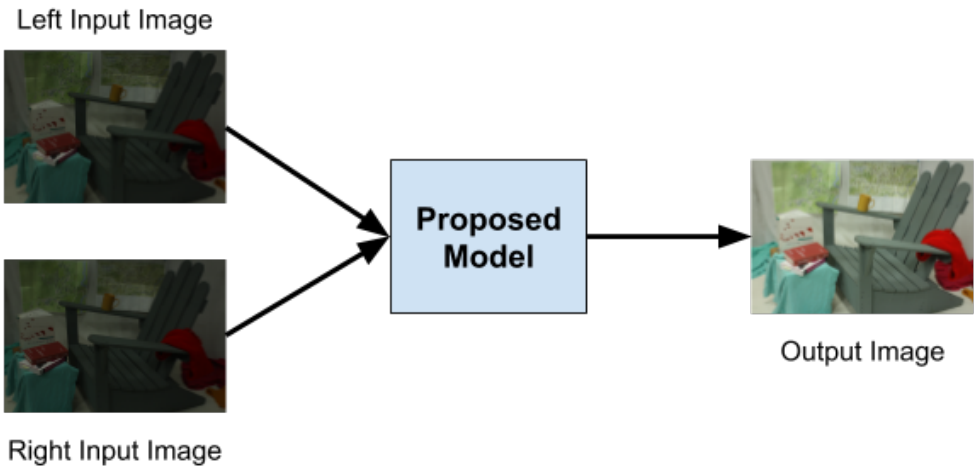


Figure 4.1: Input and output of the proposed model.

camera setup.

The reason to use a stereo image pair instead of one single image for the low-light enhancement is to exploit the high correlation between the two images in a stereo image pair. In dual camera setup, we have two different views of the same object in a scene, which are highly correlated. In our proposed method, we use this prior to enhance the low-light stereo images. Our results demonstrate that using a dual camera system in low-light enhancement leads to better results than using only a single camera system.

We design an end-to-end deep convolutional neural network system for enhancing the low-light stereo images. Therefore, the input to our system is a stereo image pair (left and right image), and the output is a left normal-light image. Figure 4.1 shows the input and output of the proposed system.

As we discussed in section 2.5.1, in a dual camera system, every point in 3D space is represented by two pixels, one in the left image and the other in the right

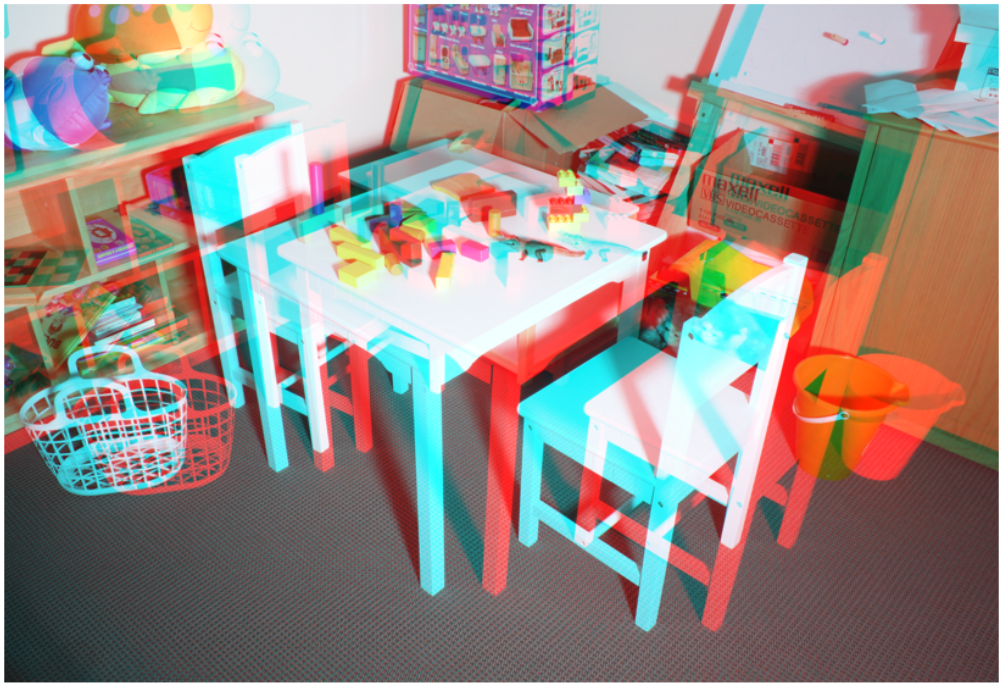


Figure 4.2: A superimposed photo of a stereo image pair. This image shows the disparity between the two left and right images.

image. The coordinates of these two pixels are different due to the distance between the two cameras. The disparity is the difference between these two corresponding coordinates. Figure 4.2 demonstrates the disparity between the left and right images by superimposing the two images.

One way to enhance the stereo images in a dual camera system is to align or register the two images first and then using an enhancement method to produce the enhanced normal-light output image. However, estimating the disparity is still an active research topic and finding an accurate disparity is a difficult task. Instead, we propose an end-to-end CNN-based method to avoid estimating the disparity in a stereo image pair.

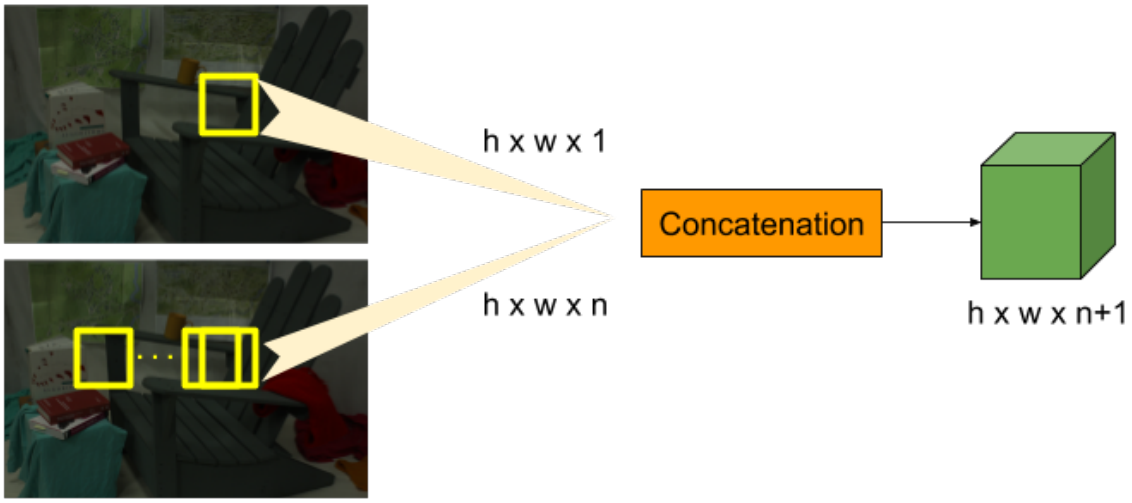


Figure 4.3: The process of how we extract patches to train the CNN.

In order to design a convolutional neural network that learns a mapping from a low-light image pair to a normal light image, we extract a patch from the left image and n patches from the right image with different shift offsets with respect to the corresponding left patch. Shifted patches are used specifically for accommodating the imprecise disparity between the left and right patch pairs. These patches are concatenated together to create a 3D tensor with size $h \times w \times (n+1)$ which is the input to the network. h and w are the height and width of the tensor respectively. In this development, we choose $n = 64$ because it is more than enough to accommodate the disparity between the two left and right patches. Figure 4.3 shows the process of extracting input patches to train the CNN. For a particular patch in the left image with center position (x, y), the shifted patches are extracted with $(x - j, y)$ center positions from the right image where j varies from 1 to n. Note that the two images are assumed rectified and have no disparity in the y direction.

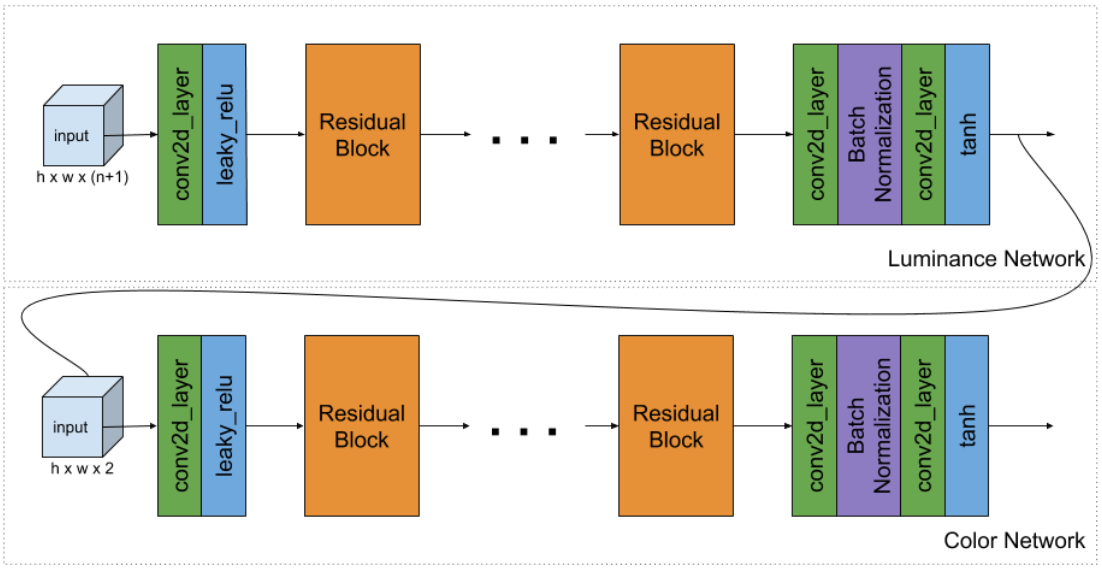


Figure 4.4: The proposed network architecture for low-light enhancement using stereo images. All the convolution layers has the same number of filters (64) with filter size 3×3, stride=1, and padding. The network has two parts: the luminance and color networks.

4.2 Network Architecture

We devise a deep CNN network architecture to improve the image quality in low-lighting condition of stereo images inspired by (Jeon *et al.*, 2018). The network processes the luminance and chrominance components of the input image in a cascade architecture. The first part of this network learns the luminance mapping of a low-light stereo image pair to a normal-light image, while the second part learns a color transformation mapping from both the enhanced luminance and the chrominance of the low-light left image to a normal-image. The input tensor for the luminance network has $n + 1$ channels and the input tensor for the color network has only two chrominance channels from the left image and one estimated luminance channel of the left image. See figure 4.4 for an overview.

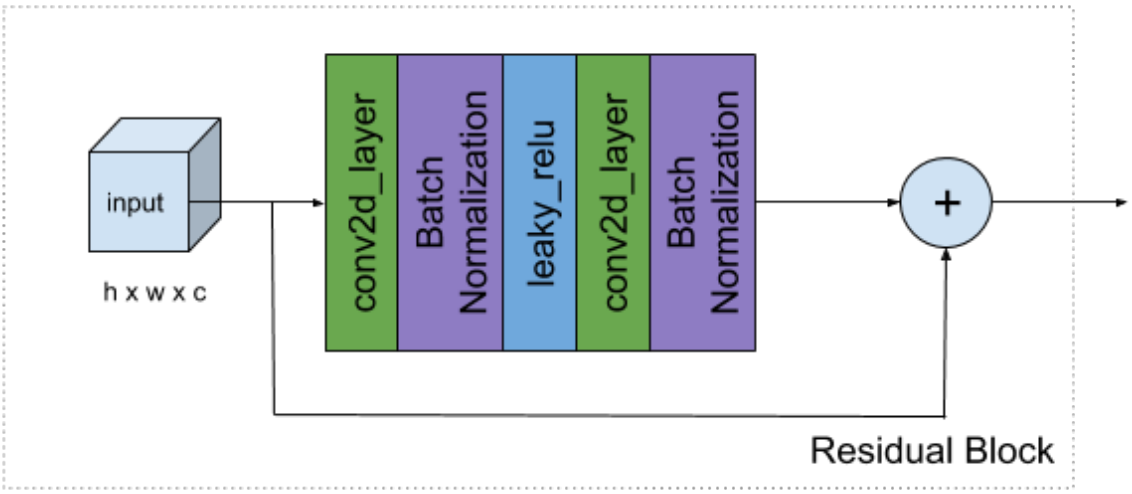


Figure 4.5: Residual block used in the proposed network. Both convolutional layers have 64 filters with filter size 3×3, stride=1, and padding.

In Figure 4.4, there are 32 *Residual Blocks* in each of the luminance and color parts. We use the the residual blocks in a similar way as (Kim *et al.*, 2016). The residual learning makes networks deeper without suffering from the gradient vanishing problem. Figure 4.5 shows the architecture of each residual block shown in Figure 4.4.

4.2.1 Network Construction

Instead of using device-oriented color representation of red, green, blue, we convert three color channels into YC_bC_r coefficients of luminance Y and chrominance C_bC_r as a 3D tensor of size $H \times W \times C \in \Re^3$, where H is the height, W is the width, and C is the number of channels in an input image. We design the first part of the network that learns a luminance mapping between a normal-light patch and a stack of the left patch and shifted right patches. Suppose a training data-set $\{X^i, Y^i\}_{i=1}^N$ is given from N number of training images where X^i represents a stereo image pair

and $Y^i \in H \times W \times 3$ denotes the YC_bC_r representation of ground truth normal-light image for i^{th} image. $X = \{X_l, X_r\}$ contains the YC_bC_r representation of the two left $X_l \in H \times W \times 3$ and right $X_r \in H \times W \times 3$ low-light stereo images. The main objective of our network is to learn a model F that can predict a normal-light image $\hat{Y} = F(X)$ from given stereo input X. As entitled at the beginning of this section, our model consists of two parts: a luminance part and a color part. Note that we jointly train both parts as one to learn an end-to-end mapping. In this section, we denote images with a capital letter (X) and patches with small letters, i.e., x.

Luminance Part: The luminance part detects similar patches in the input stereo tensor with shifted patches using deep convolutional networks, rather than using traditional block matching that determines the discrete disparity. It means that our luminance network detects patch similarities in stereo channels that contain patches with different shift offsets. Finding similar patches is a more effective way to enhance patch low-light condition regardless of patch correspondences for the disparity. Note that no disparity map is required for our enhancement of low-light condition.

In the training phase, the first network learns the mapping between a normal-light luminance patch $l_y \in h \times w \times 1$ and a low-light luminance stereo patch stack $l_x \in h \times w \times (n+1)$ over training data-set where h and w are the height and width of patches. The input tensor to the luminance network can be defined as follows:

$$\begin{aligned} l_{x,k}(i,j) &= l_{x,r}(i-k,j) \qquad for \qquad k \in \{1, \cdots, n\} \\ l_{x,k}(i,j) &= l_{x,l}(i,j) \qquad for \qquad k = n+1 \end{aligned}$$

where $l_{x,l}(i,j) \in h \times w \times 1$ and $l_{x,r}(i,j) \in h \times w \times 1$ are the luminance channel of the two left and right patches with the (i,j) center position coordinates. The output

of the luminance network $\hat{l}_y$ is the enhanced normal-light luminance patch with size $h \times w \times 1$.

Color Part The input of the color network c_x is the concatenation of reconstructed normal-light luminance patch $\hat{l}_y$ and the low-light chrominance channels from C_b and C_r channels in the low-light left patch $b_l \in h \times w \times 1$ and $r_l \in h \times w \times 1$.

The main objective of the color network is to reconstruct a final normal-light color image. The output of the color network $\hat{y}$ is the enhanced normal-light full color patch with size $h \times w \times 3$.

All the images in the training data-set are normalized to $[-1, 1]$ range by:

$$y = \frac{x}{127.5} - 1 \tag{4.1}$$

where y is the normalized value, x is the integer pixel value which is in range $[0, 255]$.

The last layer in both networks uses a *tanh* activation function which outputs data in the range $[-1, 1]$. We denormalized the output values to the range $[0, 255]$ for showing and storing the results.

4.3 Loss Function

In the training phase, we use the following loss function to train both the luminance and color parts simultaneously:

$$J = \frac{1}{N}\sum_{i=1}^{N} ||l_y^i - F_{lum}(l_x^i)||_2^2 + \lambda \frac{1}{N}\sum_{i=1}^{N} ||y^i - F_{color}(c_x^i)||_2^2 \tag{4.2}$$

where i represents the i^{th} training sample, N is the number of training samples, l_y represents the luminance component of the ground-truth left patch, y denotes the ground-truth YC_bC_r coefficients of the left patch, and l_x, c_x represent the input for the luminance and color network, respectively, while $F_{lum}()$ and $F_{color}()$ are the output of the luminance and color networks. λ is a weight parameter which controls the importance of the color network compare to the luminance network.

4.4 Training Data-set

In this development, we use the publicly available Middlebury data-set (Scharstein *et al.*, 2014). Originally, this data-set has been introduced for the depth estimation topic. However, we use this data-set for our stereo enhancement problem because it contains over-exposed (bright), under-exposed (dark), and normal light stereo images. Thus, we can construct a training data-set to train our CNN network as discussed in section 4.2.1.

Middlebury data-set has 33 different indoor scenes, where for every scene, four different ambient lights are available. For every ambient light, there are eight stereo image pairs (left and right) with different exposure time from too bright to too dark. Moreover, a ground truth disparity mapping is also available for all these scenes. Figure 4.6 shows different scenes available in this data-set. Figure 4.7 illustrates different ambient lights and exposure times for one of the Middlebury data-set scenes. The disparity between the two images is also shown in Figure 4.8.

To create a training data-set for our low-light image enhancement method, we need an under-exposed image pair and the corresponding normal-light stereo image pairs. For every ambient light, we choose the best normal image among all the

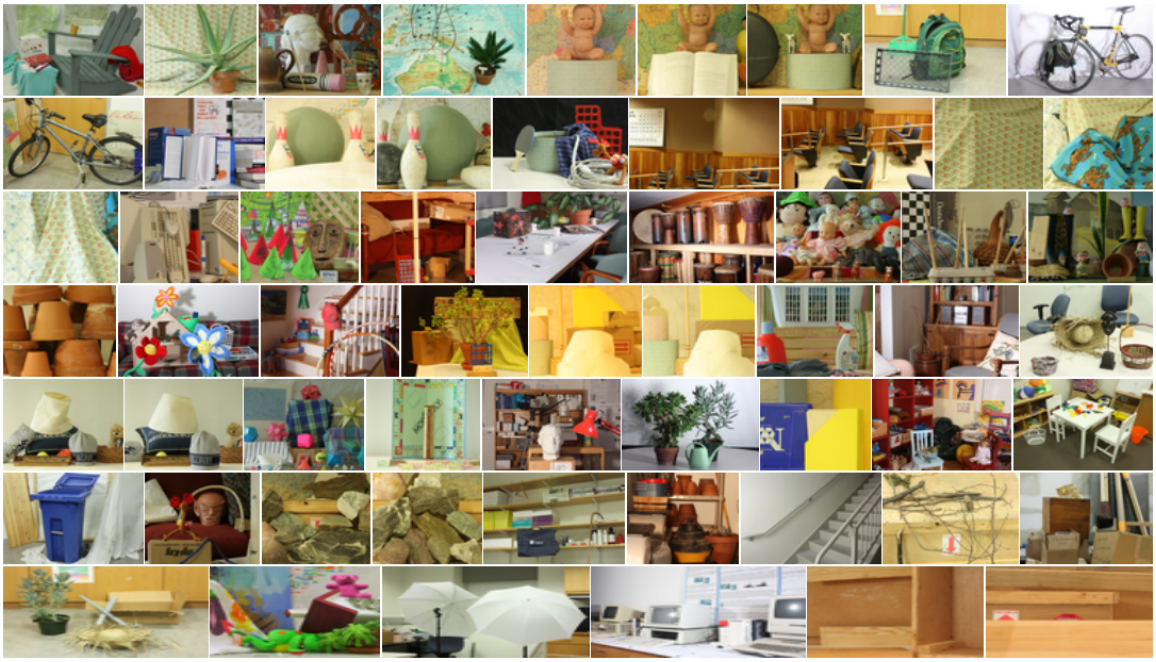


Figure 4.6: Different scenes of Middlebury data-set used in training our CNN network.

different exposure time image pairs. For instance, in Figure 4.7, we select the fourth row as the normal-light ground truth stereo image pair. We also removed all the first three rows because we aim to enhance only low-light stereo images. Then, in this example, we have four ground truth image pairs and 4 low-light image pairs to create the training data-set. The same procedure is done for all other scenes in this data-set.

In total, we have 78 normal-light (ground truth) and 230 low-light stereo images. We extracted $517,586$ patches with size 33×33 with stride 24 from these images. Table 4.1 shows all the parameters used in creating the training data-set.

Figure 4.7: Different ambient lights (every column) and exposure times (every row) for one of the Middlebury images.

Table 4.1: Parameters used in creating the training data-set. Stride is the jump value for going to the next patch in both horizontal and vertical axes.

Parameter	Value
Patch size	33×33
stride	24
Number of shifted patches in the right image	64
Total number of patches	$517,586$

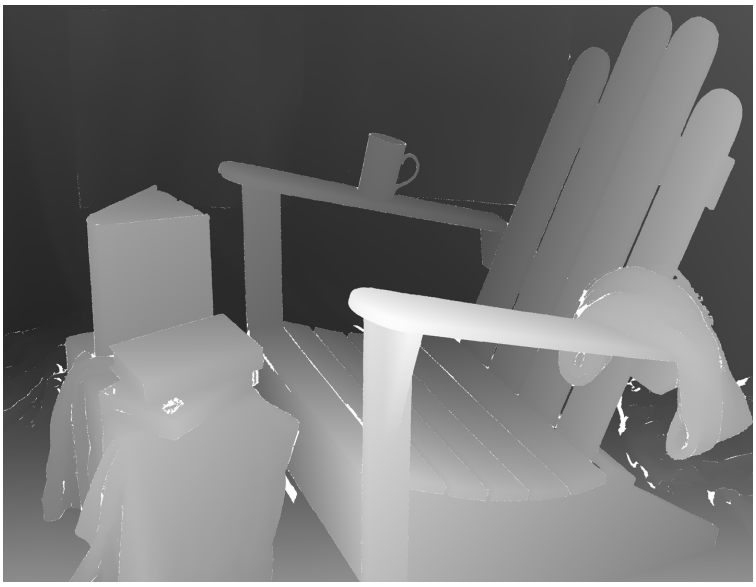


Figure 4.8: The disparity mapping for image pair shown in Figure 4.7. Bright intensities are corresponding to the close objects to the dual camera setup while dark intensities represent the far objects.

4.5 Evaluation Criteria

Commonly used criteria for image quality assessment are *Mean Squared Error* (MSE), *Peak Signal-to-Noise Ratio* (PSNR) and *Structural similarity* (SSIM) (Wang *et al.*, 2004). All these criteria compare a reference image I_{ref} with a target image I_{tar}. Here we explain these three evaluation metrics.

MSE: MSE is basically a weighted function of deviations in images, or square difference between compared images (Huynh-Thu and Ghanbari, 2008). The MSE for two images I_{ref} and I_{tar} with size $M \times N$ is calculated as follows:

$$MSE = \frac{1}{MN} \sum_{i=1}^{M} \sum_{j=1}^{N} (I_{ref}(i,j) - I_{tar}(i,j))^2. \tag{4.3}$$

PSNR: Another measure, which is strongly related to MSE is PSNR, defined by

equation (4.4). It indicates the level of losses or signals integrity (Wang *et al.*, 2004).

$$PSNR = 10log(\frac{max(I)^2}{MSE}), \tag{4.4}$$

where $max(I)$ is the maximum value for all pixels in the two images. For instance, this value for 8-bit images is 255.

SSIM is more closely related to the human visual system as it extracts useful information as luminance (l), contrast (c), and structure (s). It can be applied to evaluate structure preservation and noise removal (Wang *et al.*, 2004). The SSIM index is calculated on various windows of the two images. The measure between two windows x and y of common size $N \times N$ is:

$$\begin{aligned} l(x,y) &= \frac{2\mu_x\mu_y + c_1}{\mu_x^2 + \mu_y^2 + c_1} \\ c(x,y) &= \frac{2\sigma_x\sigma_y + c_2}{\sigma_x^2 + \sigma_y^2 + c_2} \\ s(x,y) &= \frac{\sigma_{xy} + \frac{c_2}{2}}{\sigma_x\sigma_y + \frac{c_2}{2}} \end{aligned}$$

where $c_1 = (k_1L)^2$, $c_2 = (k_2L)^2$ are two variables to stabilize the division with weak denominator; L is the dynamic range of the pixel values and typically it is $L = 2^{\#bitsperpixel} - 1$ and $K_1 = 0.01$, $k_2 = 0.03$.

SSIM is then a weighted combination of those comparative measures:

$$SSIM(x,y) = [l(x,y)^\alpha.c(x,y)^\beta.s(x,y)^\gamma] \tag{4.5}$$

The resultant SSIM index is a decimal value between -1 and 1, and value 1 is only

reachable in the case of two identical images. Typically it is calculated on window sizes of 8×8. The window can be displaced pixel-by-pixel on the images, but the authors propose to use only a subgroup of the possible windows to reduce the complexity of the calculation.

4.6 Results

We employed the TensorFlow deep-learning framework (Abadi *et al.*, 2015) to implement our proposed solution to low-light stereo image enhancement. All our implementation code can be found on GitHub[1]. For computing and applying gradients to weights, we choose the ADAM optimizer (Kingma and Ba, 2014) with an initial learning rate of 0.001. We set the exponential decay rate for the first momentum as $\beta_1 = 0.9$ and the second momentum as $\beta_2 = 0.999$ and divide the learning rate by 10 after every three epochs. In each epoch, the optimization algorithm iterates through all the training samples.

The training of the two luminance and color networks takes about 12 hours on a machine with a 4.00 GHz Intel i7-6700L CPU and NVIDIA Titan XP GPU card with a batch size of 64 for 10 epochs. Table 4.2 shows all the parameters used to train the proposed network. We found that learning more than ten epochs does not improve the results significantly.

Figure 4.9 and 4.10 show the loss function plot for all the iterations in the training phase for both the luminance and color networks. The total loss is the weighted summation of these two losses with $\lambda = 0.5$.

To show that using a dual camera system leads to better results rather than using

[1] `https://github.com/hassanisaadi/dark_enh_rgb`

Table 4.2: All the parameters used to train the proposed network.

Parameter	Value
Number of Epochs	10
Batch size	64
Initial learning rate	0.001
Number of neighbor patches in the right image	64
λ used in the loss function	0.5
Number of Residual Blocks in each network	32
Feature map size for every convolutional layer	64

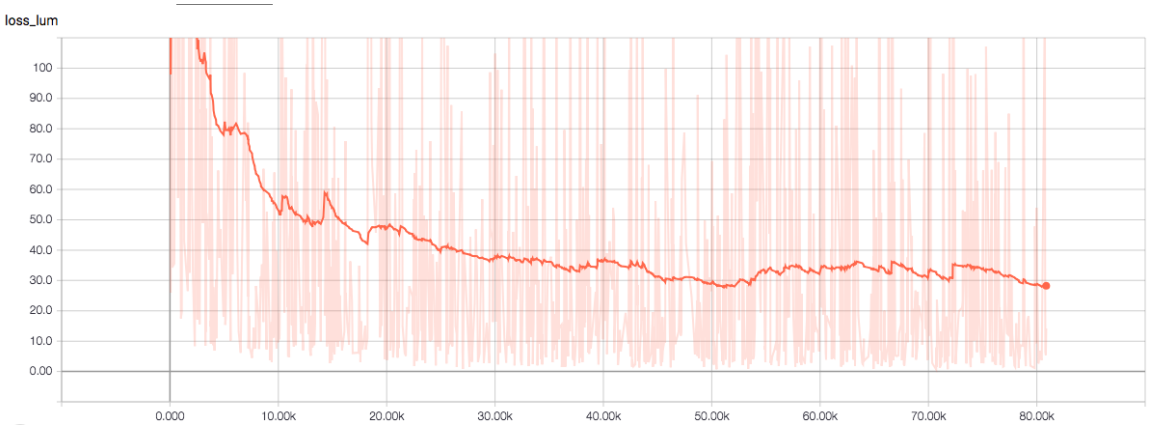


Figure 4.9: Luminance loss function for the proposed network.

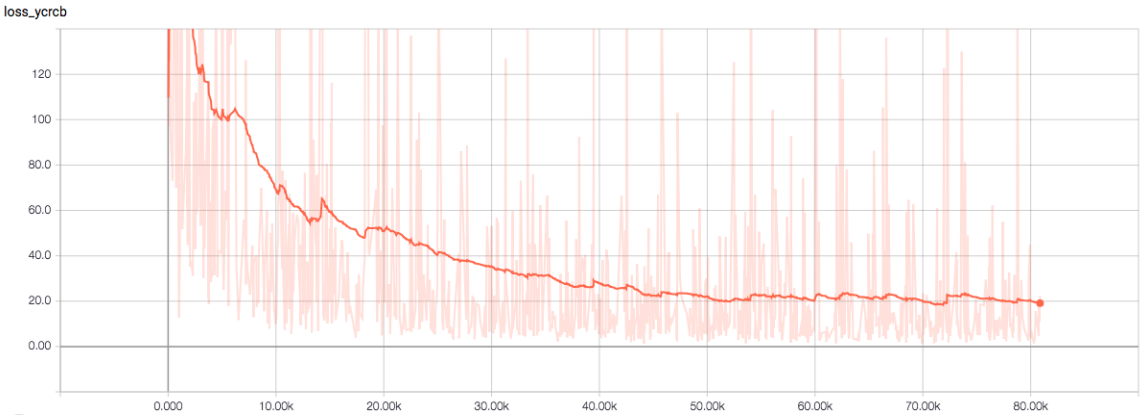


Figure 4.10: Color loss function for the proposed network.

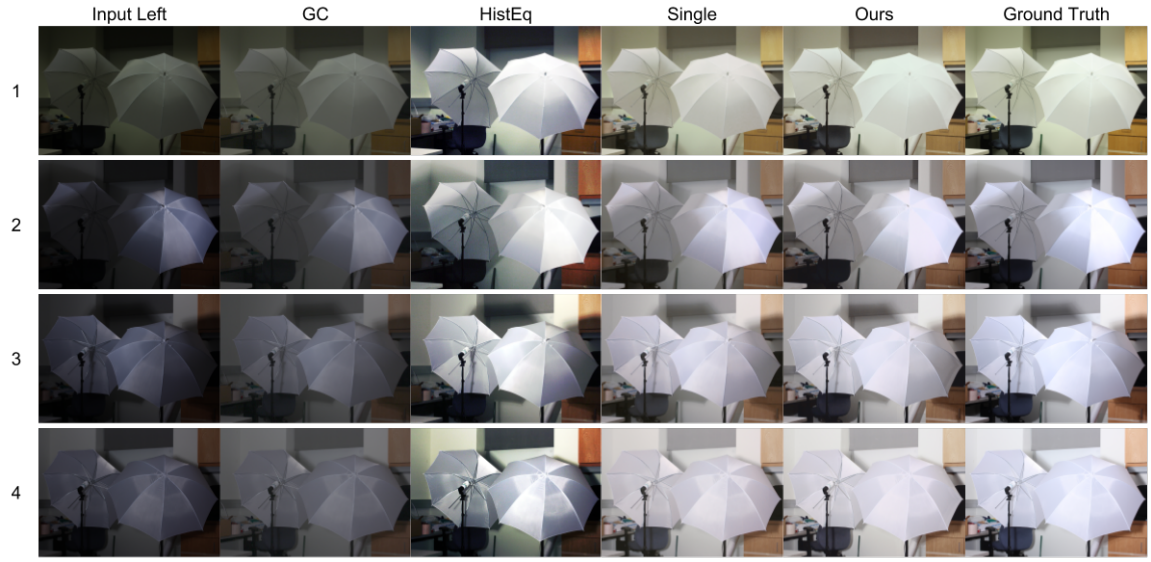


Figure 4.11: Results for different methods compared to our method. Gamma correction (GC) with $\lambda = 0.5$, Histogram equalization (HistEq), Single left image (Single), Ours.

a single camera setup, we train another network which is exactly the same as the proposed network with a difference in the input layer of the luminance network. This single camera network takes only the luminance part of the left patch and maps the low-light left image to the normal-light left image.

For a better comparison, we also evaluate our method with the two basic methods discussed in the previous chapter: Gamma Correction and Histogram Equalization. We evaluate our low-light enhancement method on 8 test images from the Middlebury data-set. Figures 4.11 and 4.12 show the results for four different low-light enhancement methods: 1) Gamma correction, 2) Histogram equalization, 3) Single camera, 4) Our method. Table 4.3 also shows the PSNR and SSIM values for the images in the previous figures.

The images reconstructed by both gamma correction and histogram equalization

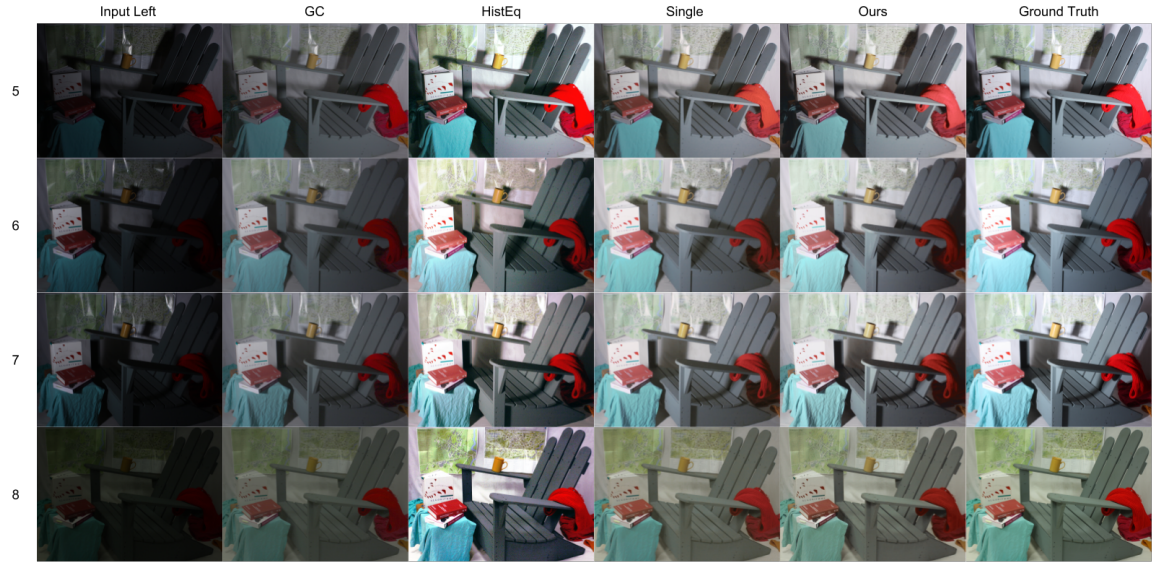


Figure 4.12: Results for different methods compare to our method. Gamma correction (GC) with $\lambda = 0.5$, Histogram equalization (HistEq), Single left image (Single), Ours.

Table 4.3: PSNR values for different methods compare to our method. Gamma correction (GC) with $\lambda = 0.5$, Histogram equalization (HistEq), Single left image (Single), Ours.

Test Image	GC		HistEq		Single		**Ours**	
	PSNR	SSIM	PSNR	SSIM	PSNR	SSIM	PSNR	SSIM
1	11.33	0.7515	15.95	0.3449	28.34	0.9183	**35.88**	**0.9233**
2	11.91	0.7403	19.43	0.5482	28.88	0.8369	**35.13**	**0.8517**
3	10.85	0.7618	15.59	0.5862	28.04	0.8258	**34.61**	**0.8526**
4	9.87	0.7784	10.73	0.5154	27.45	0.7692	**34.21**	**0.8326**
5	11.78	0.6770	17.18	0.7460	26.31	0.8296	**29.51**	**0.8480**
6	11.44	0.7043	22.35	0.6346	26.34	0.8291	**30.56**	**0.8417**
7	11.47	0.7082	21.62	0.6732	26.43	0.8284	**30.79**	**0.8481**
8	13.16	0.6822	24.04	0.4561	27.51	**0.8859**	**31.56**	0.8814
Average	11.48	0.7255	18.36	0.5631	27.41	0.8404	**32.78**	**0.8599**

methods suffer from a global noise, resulting in low PSNR. Comparing to the single camera method, our method outperforms the single camera regarding PSNR and SSIM which shows the effectiveness of the proposed method empirically.

Chapter 5

Conclusion and Future Work

5.1 Conclusion

We have proposed a method that can enhance the low-light condition of stereo images, which comprises two convolutional neural networks for the luminance and color, respectively. Even though our method does not calculate the disparity directly, it utilizes a shifted neighbor patches in stereo to reconstruct a normal-light image. Our method can outperform the state-of-the-art methods such as gamma correction and histogram equalization. We also show that the proposed dual camera system performs better than the single camera setup enhancement empirically.

5.2 Future Work

In our development, we use a public data-set (Middlebury) which has a limited number of images to train our model. All the images in this data-set are captured in an indoor space. To create a larger and more general training data-set, we could use

smartphones that have a dual camera system. Capturing thousands of indoor and outdoor images using these smartphones provide a larger training data-set. Moreover, since the two sensors in smartphones have a smaller distance compared to the dual camera used in the Middlebury data-set, the disparity between the two images would be much smaller. The lower disparity leads us to extract fewer neighbor patches and probably higher quality in the output.

Using more complicated models such as Auto-Encoders (AE), Generative Adversarial Networks (GAN) could help us in increasing the evaluation criteria in the low-light enhancement. In addition, we can also use pre-trained networks to improve our model performance. Pre-trained networks such as VGG (Simonyan and Zisserman, 2014) and ImageNet (Krizhevsky *et al.*, 2012) have been trained on a large number of images to perform the object detection task. Many researchers apply these networks on different tasks such as image enhancement which is called *transfer learning*. In other words, transfer learning helps models to benefit pre-trained networks in providing more useful feature maps rather than image pixel values. We can also use the pre-trained networks in our model to improve the performance.

In chapter 1, dual camera smartphones with a Bayer and Monochrome sensor are discussed. Another possibility to improve the performance of the proposed model could be using a Bayer and monochrome image stereo pair. Since the monochrome image captures all the existing light in the space, it contains more details, especially in the low-light conditions. Moreover, a raw mosaiced image without demosaicing step in the image processing pipeline does not include demosaicing artifacts. Therefore, using a Bayer and monochrome stereo image pair can lead us to a better enhanced image.

Bibliography

Abadi, M., Agarwal, A., Barham, P., Brevdo, E., Chen, Z., Citro, C., Corrado, G. S., Davis, A., Dean, J., Devin, M., Ghemawat, S., Goodfellow, I., Harp, A., Irving, G., Isard, M., Jia, Y., Jozefowicz, R., Kaiser, L., Kudlur, M., Levenberg, J., Mané, D., Monga, R., Moore, S., Murray, D., Olah, C., Schuster, M., Shlens, J., Steiner, B., Sutskever, I., Talwar, K., Tucker, P., Vanhoucke, V., Vasudevan, V., Viégas, F., Vinyals, O., Warden, P., Wattenberg, M., Wicke, M., Yu, Y., and Zheng, X. (2015). TensorFlow: Large-scale machine learning on heterogeneous systems. Software available from tensorflow.org.

Arbelaez, P., Maire, M., Fowlkes, C., and Malik, J. (2011). Contour detection and hierarchical image segmentation. *IEEE transactions on pattern analysis and machine intelligence*, **33**(5), 898–916.

Bhavsar, A. V. and Rajagopalan, A. (2010). Resolution enhancement in multi-image stereo. *IEEE transactions on pattern analysis and machine intelligence*, **32**(9), 1721–1728.

Boyd, S., Boyd, S., Vandenberghe, L., and Press, C. U. (2004). *Convex Optimization*. Berichte über verteilte messysteme. Cambridge University Press.

Bromley, J., Guyon, I., LeCun, Y., Säckinger, E., and Shah, R. (1994). Signature verification using a" siamese" time delay neural network. In *Advances in neural information processing systems*, pages 737–744.

Dong, C., Loy, C. C., He, K., and Tang, X. (2016). Image super-resolution using deep convolutional networks. *IEEE transactions on pattern analysis and machine intelligence*, **38**(2), 295–307.

Duchi, J., Hazan, E., and Singer, Y. (2011). Adaptive subgradient methods for online learning and stochastic optimization. *Journal of Machine Learning Research*, **12**(Jul), 2121–2159.

Fukushima, K. (1988). Neocognitron: A hierarchical neural network capable of visual pattern recognition. *Neural networks*, **1**(2), 119–130.

Geiger, A., Lenz, P., and Urtasun, R. (2012). Are we ready for autonomous driving? the kitti vision benchmark suite. In *Conference on Computer Vision and Pattern Recognition (CVPR)*.

Gonzalez, R. C. and Woods, R. E. (2006). *Digital Image Processing (3rd Edition)*. Prentice-Hall, Inc., Upper Saddle River, NJ, USA.

Goodfellow, I., Bengio, Y., and Courville, A. (2016). *Deep Learning*. MIT Press.

Hinton, G., Srivastava, N., and Swersky, K. (2014). Lecture 6a overview of mini-batch gradient descent.

Huynh-Thu, Q. and Ghanbari, M. (2008). Scope of validity of psnr in image/video quality assessment. *Electronics letters*, **44**(13), 800–801.

Jeon, D. S., Baek, S.-H., Choi, I., and Kim, M. H. (2018). Enhancing the spatial resolution of stereo images using a parallax prior. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 1721–1730.

Kim, J., Kwon Lee, J., and Mu Lee, K. (2016). Accurate image super-resolution using very deep convolutional networks. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 1646–1654.

Kingma, D. P. and Ba, J. (2014). Adam: A method for stochastic optimization. *CoRR*, **abs/1412.6980**.

Krizhevsky, A., Sutskever, I., and Hinton, G. E. (2012). Imagenet classification with deep convolutional neural networks. In *Advances in neural information processing systems*, pages 1097–1105.

Kurenkov, A. (2015). A 'brief' history of neural nets and deep learning. `http://www.andreykurenkov.com/writing/ai/a-brief-history-of-neural-nets-and-deep-learning/`. Accessed: 2018-08-27.

Land, E. H. (1977). The retinex theory of color vision. *Scientific american*, **237**(6), 108–129.

Li, X., Gunturk, B., and Zhang, L. (2008). Image demosaicing: A systematic survey. In *Visual Communications and Image Processing 2008*, volume 6822, page 68221J. International Society for Optics and Photonics.

Long, L. N. and Gupta, A. (2008). Scalable massively parallel artificial neural networks. *Journal of Aerospace Computing, Information, and Communication*, **5**(1), 3–15.

Losson, O., Macaire, L., and Yang, Y. (2010). Comparison of color demosaicing methods. In *Advances in Imaging and Electron Physics*, volume 162, pages 173–265. Elsevier.

Park, H., Lee, K. M., and Lee, S. U. (2012). Combining multi-view stereo and super resolution in a unified framework. In *Signal & Information Processing Association Annual Summit and Conference (APSIPA ASC), 2012 Asia-Pacific*, pages 1–4. IEEE.

Park, S. C., Park, M. K., and Kang, M. G. (2003). Super-resolution image reconstruction: a technical overview. *IEEE signal processing magazine*, **20**(3), 21–36.

Ruder, S. (2016). An overview of gradient descent optimization algorithms. *arXiv preprint arXiv:1609.04747*.

Schaefer, G. and Stich, M. (2003). Ucid: an uncompressed color image database. In *Storage and Retrieval Methods and Applications for Multimedia 2004*, volume 5307, pages 472–480.

Scharstein, D. and Szeliski, R. (2002). A taxonomy and evaluation of dense two-frame stereo correspondence algorithms. *International journal of computer vision*, **47**(1-3), 7–42.

Scharstein, D., Hirschmüller, H., Kitajima, Y., Krathwohl, G., Nešić, N., Wang, X., and Westling, P. (2014). High-resolution stereo datasets with subpixel-accurate

ground truth. In *German Conference on Pattern Recognition*, pages 31–42. Springer.

Shen, L., Yue, Z., Feng, F., Chen, Q., Liu, S., and Ma, J. (2017). Msr-net: Low-light image enhancement using deep convolutional network. *arXiv preprint arXiv:1711.02488*.

Simonyan, K. and Zisserman, A. (2014). Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556*.

Sindagi, V. A. and Patel, V. M. (2018). A survey of recent advances in cnn-based single image crowd counting and density estimation. *Pattern Recognition Letters*, **107**, 3–16.

Srivastava, N., Hinton, G., Krizhevsky, A., Sutskever, I., and Salakhutdinov, R. (2014). Dropout: a simple way to prevent neural networks from overfitting. *The Journal of Machine Learning Research*, **15**(1), 1929–1958.

Stenroos, O. (2017). *Object detection from images using convolutional neural networks; Kohteentunnistus kuvista konvoluutioneuroverkoilla*. G2 pro gradu, diplomity.

Wang, Z., Bovik, A. C., Sheikh, H. R., and Simoncelli, E. P. (2004). Image quality assessment: from error visibility to structural similarity. *IEEE Transactions on Image Processing*, **13**(4), 600–612.

Yang, J., Wright, J., Huang, T. S., and Ma, Y. (2010). Image super-resolution via sparse representation. *IEEE transactions on image processing*, **19**(11), 2861–2873.

Zbontar, J. and LeCun, Y. (2016). Stereo matching by training a convolutional neural network to compare image patches. *Journal of Machine Learning Research*, **17**(1-32), 2.