Index

Chapter 1

Introduction

In bad weather conditions like fog and haze, the particles present in the atmosphere scatter incident light in different directions. As a result, image taken under these conditions suffers from reduced visibility, lack of contrast, as a result, it appears colorless. Image dehazing method tries to recover a haze-free portrayal of the given hazy image. In this paper we propose a method that dehazes a given image by comparing various output patches with the original hazy version and choosing the best one.

Any image received from a communication system is usually noisy. Image denoising is one of the most important and crucial part of image processing and data sampling. Many important data sets picked by the sensors are usually affected by noise. It is contaminated either due to the data acquisition process, or due to naturally occurring phenomenon. One of the most prevalent cases is due to the additive white gaussian noise caused by poor image acquisition or by communicating the image data through noisy channels. The goal of denoising algorithm is to remove the unwanted noise while preserving the important signal features as much as possible.

The goal of image defogging is to recover a clean image x from a noisy observation y which follows an image degradation model $y = x + v$. Once denoising is done, we try to remove fog from the image with the help of a CNN based patch quality comparator and obtain a clean and dehazed image.

1.1 Aim of the Project

Our goal is to remove fog and noise from a given (hazy and noisy) image using denoising and dehazing techniques.

The proposed denoising convolutional neural network is referred as DnCNN. Rather than directly outputing the denoised image **x**, the proposed DnCNN is designed to predict the residual image **v**, i.e., the difference between the noisy observation and the latent clean image. In other words, the proposed DnCNN implicitly removes the latent clean image with the operations in the hidden layers. The batch normalization technique is further introduced to stabilize and enhance the training performance of DnCNN. It turns out that residual learning and batch normalization can benefit from each other, and their integration is effective in speeding up the training and boosting the denoising performance.

While this project aims to design a more effective denoiser, we observe that when **v** is the difference between the ground truth high resolution image and the upsampling of the low resolution image, the image degradation model for Gaussian denoising can be converted to a single image super-resolution (SISR) problem; analogously, the JPEG image deblocking problem can be modeled by the same image degradation model by taking v as the difference between the original image and the compressed image. In this sense, SISR and JPEG image deblocking can be treated as two special cases of a “general” image denoising problem, though in SISR and JPEG deblocking the noise vs are much different from AWGN. It is natural to ask whether is it possible to train a CNN model to handle such general image denoising problem? By analyzing the connection between DnCNN and TNRD, we propose to extend DnCNN for handling several general image denoising tasks, including Gaussian denoising.

Furthermore, in this paper we have proposed a daytime image dehazing method that finds the transmittance at each patch (and subsequently at each pixel) by comparing the dehazed version of the patch with the hazy one. This is motivated by the fact that comparing two patches and saying which one has more haze is easier than saying the haze level of a given patch. Now, this comparison is performed by our proposed module called patch quality

comparator. This comparator, when given two patches as input, can indicate which one is of better quality in terms of haziness. We build the comparator in such a way, that the natural looking patches (e.g., without saturated colors, noise etc.) and the patches with less haze are declared as the better one. So, with the help of this comparator, we search for a transmittance value that can dehaze the given hazy patch and at the same time does not degrade the dehazed output by overdoing. Our main contribution is designing this comparator that efficiently guides this search and exploiting this comparator to solve single image dehazing problem.

Chapter 2

Review of Literature

2.1 Previous Works (Models)

- Nonlocal Self-Similarity (NSS)
- Bayesian Model
- Markov Random Field (MRF) Models
- Other State-of-the-Art methods are **BM3D, LSSC, NCSR and WNNM**
- Blind Haze Separation
- Single Image Haze Removal Using DarkChannel Prior

2.2 Drawbacks

2.2.1 Denoising

Despite their high denoising quality, most of the image prior-based methods typically suffer from two major drawbacks:

First, those methods generally involve a complex optimization problem in the testing stage, making the denoising process time-consuming. Thus, most of the prior based methods can hardly achieve high performance without sacrificing computational efficiency.

Second, the models in general are non-convex and involve several manually chosen parameters, providing some leeway to boost denoising performance.

2.2.2 Dehazing

The most challenging aspect of the problem is its depth dependent degradation, that means with increasing depth, the amount of degradation increases. As estimating depth from a single image is an ill-posed problem, a few researchers have tried to dehaze and image from the point of view of contrast restoration, assuming haze is uniform in the image. These methods work without taking into account how images form under haze. For this reason, these methods can’t handle color changes due to haze.

Chapter 3

Convolutional Neural Network

3.1 Overview

In deep learning, a convolutional neural network (CNN, or ConvNet) is a class of deep neural networks, most commonly applied to analyzing visual imagery. CNNs, like neural networks, are made up of neurons with learnable weights and biases.

CNNs are regularized versions of multilayer perceptrons. Multilayer perceptrons usually refer to fully connected networks, that is, each neuron in one layer is connected to all neurons in the next layer. The "fully-connectedness" of these networks make them prone to overfitting data. Typical ways of regularization includes adding some form of magnitude measurement of weights to the loss function. However, CNNs take a different approach towards regularization: they take advantage of the hierarchical pattern in data and assemble more complex patterns using smaller and simpler patterns. Therefore, on the scale of connectedness and complexity, CNNs are on the lower extreme.Each neuron receives several inputs, takes a weighted sum over them, pass it through an activation function and responds with an output.

3.2 Design / Architecture

A convolutional neural network consists of an input and an output layer, as well as multiple hidden layers. The hidden layers of a CNN typically consist of convolutional layers, RELU layer i.e. activation function, pooling layers, fully connected layers and normalization layers.

3.2.1 Convolutional Layer

The Convolution layer is the core building block of a Convolutional Network that does most of the computational heavy lifting. The element involved in carrying out the convolution operation in the first part of a Convolutional Layer is called the **Kernel/Filter**.

The **filter** moves to the right with a certain **Stride** value till it parses the complete width. Moving on, it hops down to the beginning (left) of the input image with the same Stride value and repeats the process until the entire image is traversed.

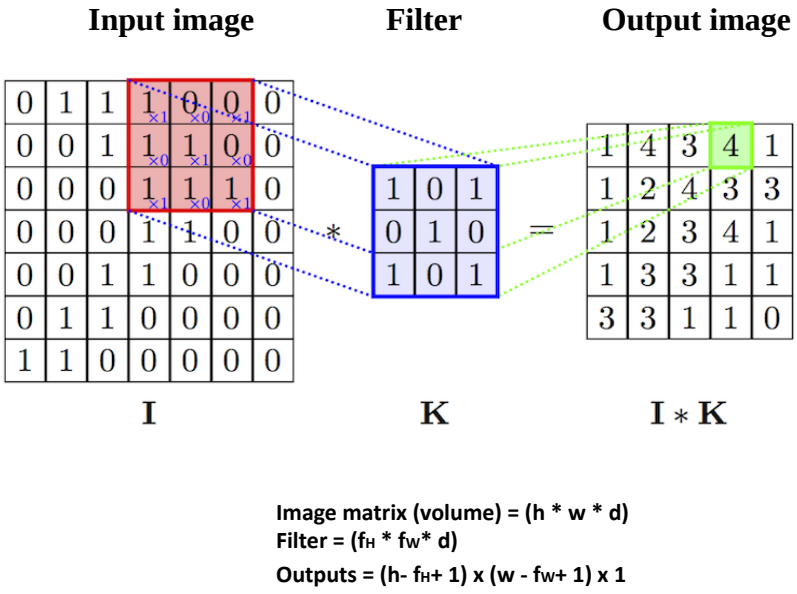


Figure 3.1: Illustration of convolution layer

3.2.2 Pooling Layer

Similar to the Convolutional Layer, the Pooling layer is responsible for reducing the spatial size of the Convolved Feature. This is to decrease the computational power required to process the data through dimensionality reduction. Furthermore, it is useful for extracting dominant features which are rotational and positional invariant, thus maintaining the process of effectively training of the model.

There are two types of Pooling:

- **Max Pooling** returns the maximum value from the portion of the image covered by the Kernel.
- **Average Pooling** returns the average of all the values from the portion of the image covered by the Kernel.

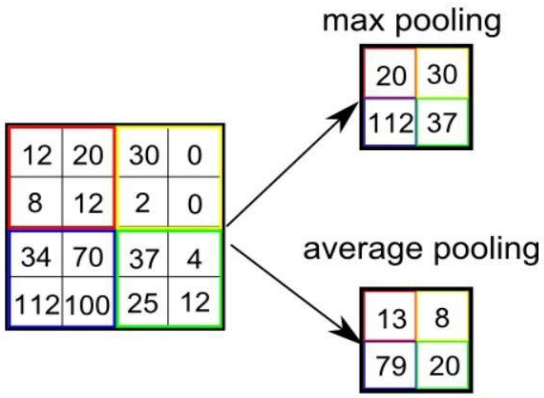


Figure 3.2: Illustration of pooling layer

3.2.3 Fully Connected Layer

Fully connected layers connect every neuron in one layer to every neuron in another layer. It is in principle the same as the traditional multi-layer perceptron neural network (MLP).

Adding a Fully-Connected layer is a (usually) cheap way of learning non-linear combinations of the high-level features as represented by the output of the convolutional layer. The Fully-Connected layer is learning a possibly non-linear function in that space. The flattened matrix then goes through a fully connected layer to classify the images.

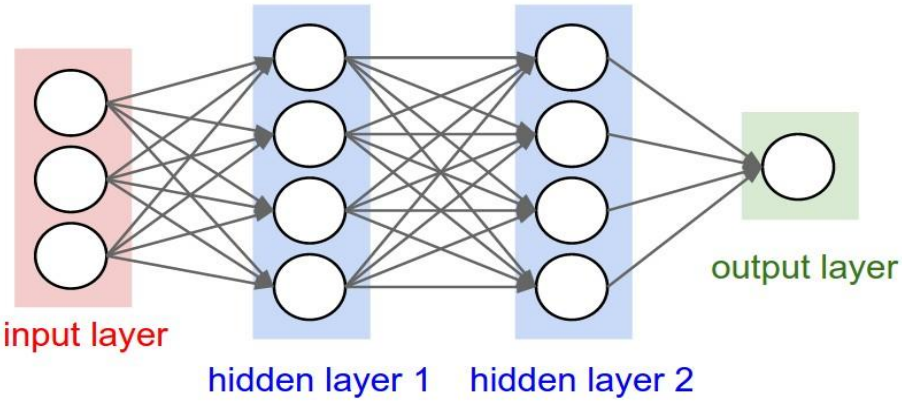


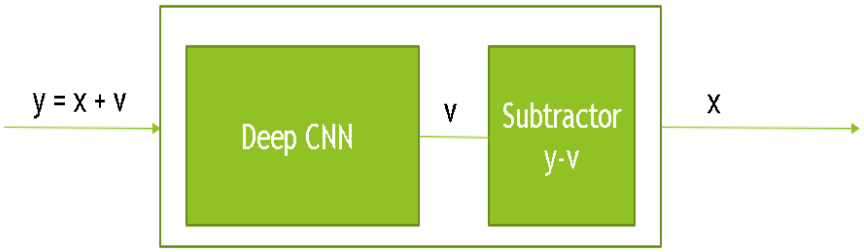
Figure 3.3: Illustration of fully connected layer

Chapter 4

Optimizing Techniques

4.1 Residual Learning

Residual learning of CNN was originally proposed to solve the performance degradation problem, i.e., even the training accuracy begins to degrade along with the increasing of network depth. By assuming that the residual mapping is much easier to be learned than the original unreferenced mapping, residual network explicitly learns a residual mapping for a few stacked layers. With such a residual learning strategy, extremely deep CNN can be easily trained and improved accuracy has been achieved for image classification and object detection. The proposed DnCNN model also adopts the residual learning formulation. Unlike the residual network that uses many residual units (i.e., identity shortcuts), our DnCNN employs a single residual unit to predict the residual image. We further explain the rationale of residual learning formulation by analyzing its connection with TNRD, and extend it to solve several general image denoising tasks. It should be noted that, prior to the residual network, the strategy of predicting the residual image has already been adopted in some low-level vision problems such as single image superresolution and color image demosaicking. However, to the best of our knowledge, there is no work which directly predicts the residual image for denoising.



where,
y = input noisy image
v = learned noise
x = predicted clean image

Figure 4.1: Model showing residual learning

4.2 Batch Normalization

Mini-batch stochastic gradient descent (SGD) has been widely used in training CNN models. Despite the simplicity and effectiveness of mini-batch SGD, its training efficiency is largely reduced by internal covariate shift, i.e., changes in the distributions of internal nonlinearity inputs during training. Batch normalization is proposed to alleviate the internal covariate shift by incorporating a normalization step and a scale and shift step before the nonlinearity in each layer. For batch normalization, only two parameters per activation are added, and they can be updated with back-propagation. Batch normalization enjoys several merits, such as fast training, better performance, and low sensitivity to initialization.

By far, no work has been done on studying batch normalization for CNN-based image denoising. We empirically find that, the integration of residual learning and batch normalization can result in fast and stable training and better denoising performance.

4.3 Merits of above methods

Residual learning and batch normalization are utilized to speed up the training process as well as boost the denoising performance. Different from the existing discriminative denoising models which usually train a specific model for additive white Gaussian noise (AWGN) at a certain noise level, our DnCNN model is able to handle Gaussian denoising with unknown noise level (i.e., blind Gaussian denoising). With the residual learning strategy, DnCNN implicitly removes the latent clean image in the hidden layers. This property motivates us to train a single DnCNN model to tackle with several general image denoising tasks such as Gaussian denoising, single image super-resolution and JPEG image deblocking. Our extensive experiments demonstrate that our DnCNN model can not only exhibit high effectiveness in several general image denoising tasks, but also be efficiently implemented by benefiting from GPU computing.

Chapter 5

Image Denoising using DnCNN

5.1 Proposed Work

The summary of this work is as follows:

1. We propose an end-to-end trainable deep CNN for Gaussian denoising. In contrast to the existing deep neural network-based methods which directly estimate the latent clean image, the network adopts the residual learning strategy to remove the latent clean image from noisy observation.

2. We find that residual learning and batch normalization can greatly benefit the CNN learning as they can not only speed up the training but also boost the denoising performance. For Gaussian denoising with a certain noise level, DnCNN outperforms state-of-the-art methods in terms of both quantitative metrics and visual quality

3. Our DnCNN can be easily extended to handle general image denoising tasks. We can train a single DnCNN model for blind Gaussian denoising, and achieve better performance than the competing methods trained for a specific noise level.

5.2 Denoising CNN Model

For network architecture design, we modify the VGG network to make it suitable for image denoising, and set the depth of the network based on the effective patch sizes used in state-of-the-art denoising methods.

5.2.1 Network Depth

Following the standards, we set the size of convolutional filters to be 3 × 3 but remove all pooling layers. Therefore, the receptive field of DnCNN with depth of d should be (2d+1)×(2d+1). Increasing receptive field size can make use of the context information in larger image region. For better tradeoff between performance and efficiency, one important issue in architecture design is to set a proper depth for DnCNN.

It has been pointed out that the receptive field size of denoising neural networks correlates with the effective patch size of denoising methods. Moreover, high noise level usually requires larger effective patch size to capture more context information for restoration. Thus, by fixing the noise level σ = 25, we found the effective patch size of several leading denoising methods to guide the depth design of our DnCNN. In BM3D, the non-local similar patches are adaptively searched in a local widow of size 25 × 25 for two times, and thus the final effective patch size is 49×49. Similar to BM3D, WNNM uses a larger searching window and performs non-local searching iteratively, resulting in a quite large effective patch size (361 × 361). MLP first uses a patch of size 39×39 to generate the predicted patch, and then adopts a filter of size 9×9 to average the output patches, thus its effective patch size is 47×47. The CSF and TNRD with five stages involves a total of ten convolutional layers with filter size of 7 × 7, and their effective patch size is 61 × 61. **Table 5.1** summarizes the effective patch sizes adopted in different methods with noise level σ = 25. It can be seen that the effective patch size used in EPLL is the smallest, i.e., 36×36. It is interesting to verify whether DnCNN with the receptive field size similar to EPLL can compete against the leading denoising methods. Thus, for Gaussian denoising with a certain noise level, we set the receptive field size of DnCNN to 35 × 35 with the corresponding depth of 17. For other general image denoising tasks, we adopt a larger receptive field and set the depth to be 20.

THE EFFECTIVE PATCH SIZES OF DIFFERENT METHODS WITH NOISE LEVEL $\sigma = 25$.

Methods	BM3D	WNNM	EPLL	MLP	CSF	TNRD
Effective Patch Size	49 × 49	361 × 361	36 × 36	47 × 47	61 × 61	61 × 61

Table 5.1: Effective patch sizes

5.2.2 Network Architecture

For DnCNN, we adopt the residual learning formulation to train a residual mapping R(y) ≈ v, and then we have x = y − R(y). Formally, the averaged mean squared error between the desired residual images and estimated ones from noisy input

$$\ell(\Theta) = \frac{1}{2N}\sum_{i=1}^{N} \|\mathcal{R}(\mathbf{y}_i;\Theta) - (\mathbf{y}_i - \mathbf{x}_i)\|_F^2 \qquad (5.1)$$

can be adopted as the loss function to learn the trainable parameters Θ in DnCNN.

Here $\{(yi , xi)\}_{i=1}^{N}$ represents N noisy-clean training image (patch) pairs. **Fig. 5.1** illustrates the architecture of the proposed DnCNN for learning R(y). In the following, we explain the architecture of DnCNN and the strategy for reducing boundary artifacts.

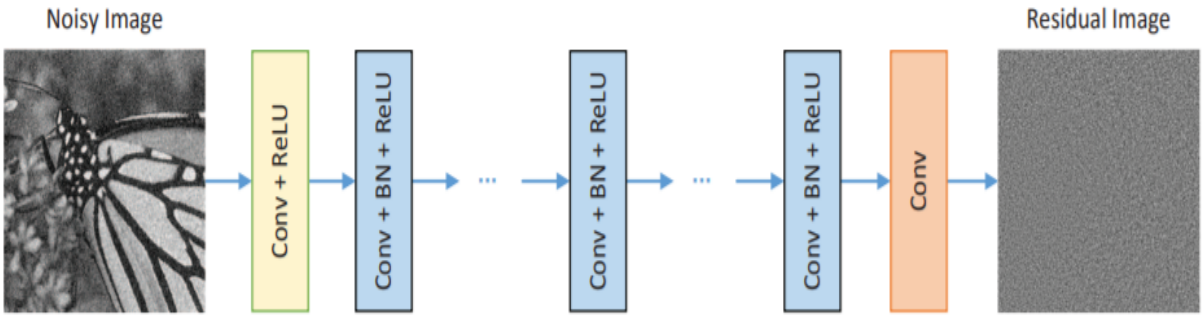


Fig 5.1: The architecture of DnCNN network

Given the DnCNN with depth D, there are three types of layers, shown in **Fig. 1** with three different colors.

(i) Conv + ReLU: for the first layer, 64 filters of size $3 \times 3 \times c$ are used to generate 64 feature maps, and rectified linear units (ReLU, $\max(0, \cdot)$) are then utilized for nonlinearity. Here c represents the number of image channels, i.e., $c = 1$ for gray image and $c = 3$ for color image.

(ii) Conv + BN + ReLU: for layers $2 \sim (D - 1)$, 64 filters of size $3 \times 3 \times 64$ are used, and batch normalization is added between convolution and ReLU.

(iii) Conv: for the last layer, c filters of size $3 \times 3 \times 64$ are used to reconstruct the output.

Chapter 6

Image Dehazing

6.1 Background and Motivation

Light propagating through a scattering medium undergoes certain changes. Transformation of intensity is one of them. This change in intensity is modeled using the following equation:

$$\mathbf{I(x) = J(x)t(x) + (1 - t(x))A} \qquad \mathbf{(6.1)}$$

$$\mathbf{t(x) = e^{-\beta d(x)}} \qquad \mathbf{(6.2)}$$

where I(x) is the observed intensity, J(x) is the intensity of light coming from the scene objects and before getting scattered, t(x) is the scene transmittance denoting the amount of light that reaches the observer after getting scattered and A denotes the global environmental illumination. The scene transmittance t(x) depends on the depth at position x and the scattering coefficient (β) (Eq. 2). This scattering coefficient depends on the size of the scattering particle and the wave- length of the light.

We assume t(x) to be constant across the channels. So, for RGB images the Eq. (1) is treated as a vector equation, with I(x), J(x), and A as 3 × 1 vector and t(x) as a scalar. Now, given an image I(x) the image dehazing methods try to recover J(x) at each pixel. The methods usually compute the value of 'A' for the whole image and estimate t(x) at each pixel. Then the imaging model is inverted to compute J(x). Since the mapping J(x) → I(x) is not one-to-one, as t(x) varies from pixel to pixel, the estimation of J(x) independently at each pixel x can be inconclusive. The value of I(x) could be due to only J(x) when t(x) = 1 or due to only 'A' when t(x) = 0. This confusion exists even if we know 'A' for the given image. To surmount

this hurdle a simple assumption is adopted that within a small patch of the image the depth of the scene and consequently, the transmittance t is constant. This assumption is valid because a patch of the image corresponds to a small part of a single surface in the scene, which may be approximated by a relatively smooth surface except at the places where the patch is on the boundary between two surfaces. Therefore the following equation is utilized to estimate transmittance n a patch.

I(x) = J(x)t + (1 − t)A **(6.3)**

6.2 Proposed Solution

Our approach is built on the principle that for a given hazy patch there is a t = t ′ that properly dehazes this patch. Dehazing this patch with t > t ′ retains some haze in the dehazed output, and on the other hand, using t < t ′ produces over contrasted, bad looking, unnatural output. So, dehazing a given patch with t = t ′ , we get the actual J(x) by the following equation:

$$J^c_{t'}(\mathbf{x}) = A - \frac{A - I^c(\mathbf{x})}{t'} \quad \textbf{(6.4)}$$

Where c ∈ {R, G, B} is one of the color channels. If the same patch is dehazed with t (≠ t ′), we can write:

$$J^c_t(\mathbf{x}) = A - \frac{A - I^c(\mathbf{x})}{t} \quad \textbf{(6.5)}$$

From these two equations the following can be written:

$$\Delta(\mathbf{x}) = J_t^c(\mathbf{x}) - J_{t'}^c(\mathbf{x}) = (A - I^c(\mathbf{x}))\left(\frac{t - t'}{tt'}\right) \quad \textbf{(6.6)}$$

So, depending on the value of Δ(x) we can say whether the dehazed output J t (x) is more than the actual J(x) or less. Now this equation has two terms. The first term will always be positive. As, A is the environmental illumination, everything in the scene is illuminated by it. So, the value of I(x) (= r(x)A, where r(x) ∈ [0, 1] denotes the reflectance property of the scene object) can't be more than A. Therefore the value of Δ(x) depends only on the relation between t and t ′ . If t < t ′ then Δ(x) is negative. That means the dehazed output is less than the actual one, therefore darker. On the other hand, if t > t ′ then the dehazed output is more than actual one: it can be further cleaned. However, note that to be able to find the point of transition, we must be able to tell, without the knowledge of desired value of t, i.e. t ′ , whether a dehazed patch is a good dehazed patch or a bad one. For this purpose we build the patch quality comparator.

6.2.1 Patch Quality Comparator

The patch quality comparator we propose here compares two patches, say, a given patch and its dehazed version, and reports whether the dehazed patch is good or bad. Instead of using some handcrafted features and employing a two-class (good and bad) classifier to do this task, we use a CNN to learn the features and do the classification simultaneously. The proposed network takes two patches as input and produces two outputs to denote which one of the input is better. The ideal output is [1, 0] T if the first input is better and [0, 1] T otherwise. Here the assumption is that the two patches differ only in the amount of haze, and represent the same part of the same scene. The basic structure of the network is inspired from the common CNN based classifiers i.e. convolutional layers for feature extraction followed by dense layers for classification based on the extracted features. Our network is designed in the same way.

In our case, we are processing small image patches (10 × 10). Therefore, we need only small number of convolutional layers to extract features. On the other hand, we have to take small

number of dense layers to avoid overfitting of the classifier. We have used tanh function as the non-linear activation throughout the network except the last layer. In the last layer we use softmax activation function as we are training the network as a classifier. The use of tanh results in relatively large gradients (during backpropagation) leading to speed up of the optimization process. Moreover, the vanishing gradients problem associated with tanh is not likely to occur as the network is not very deep. We train the comparator distinguish the haziness of patches making sure the following conditions are met:

1) The haze patch is better than a bad dehazed patch.
2) A good dehazed patch is better than the haze patch.

Using this comparator we find at what value of t the transition from good dehazed patch to bad dehazed patch occurs, by repeatedly dehazing a given patch using different values of t. As we have already mentioned, this point of transition gives the desired value of t for this patch. Now, Instead of arbitrary search for this point, we employ binary search to do this computationiently.

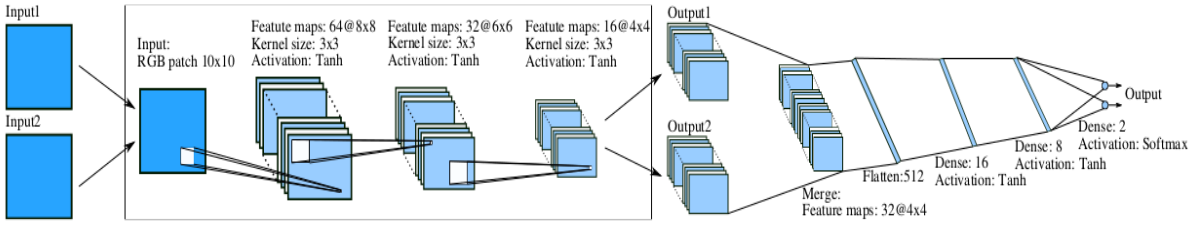


Fig 6.1: Architecture of our Patch Quality Comparator

6.3 Implementation of the proposed solution

The proposed dehazing method has the following 4 main steps:

1) Computation of atmospheric light A.
2) Binary search for the '*t*' for each patch, based on the response of the comparator.
3) Aggregation and Interpolation of the estimated *t*.
4) Haze-free image recovery.

6.3.1 Computation of Environmental Illumination

The environmental illumination can be estimated from the color of the most haze opaque region, i.e., where value of the t is least or, in other words, depth is large. This region is detected with the help of the dark channel of the hazy image. Dark channel of an image I is given by::

$$D(\mathbf{x}) = \min_{\mathbf{y} \in \Omega(\mathbf{x})} \left(\min_{c \in \{R,G,B\}} I^c(\mathbf{y}) \right) \quad \textbf{(6.7)}$$

where *I* is a color channel of *I* and Ω(x) is a local patch centered at x.

6.3.2 Transmittance finding using binary search

In this step we estimate t from a given patch. We find the ideal t (= t′) for a hazy patch using patch quality comparator following binary search strategy. We begin the process with $t_e = 1$ and $t_b = 0$. We compute $t_m = (t_b + t_e)/2$. Then the input patch is dehazed with the value t_m, and the input hazy patch and the dehazed patch is compared. If the dehazed patch is bad then we can say that $t_m < t'$. Therefore t′ lies in the range (t_m, t_e). So, we set t_b to t_m. On the other hand, if the obtained dehazed patch is good, then $t_m > t'$. So, we set t_e to t_m as t′ lies in the range (t_b, t_m). This process is repeated (i.e. computing t_m, dehazing with new t_m,

comparing new dehazed patch with the hazy one, and finally updating t_e or t_b) until ($t_e - t_b$) becomes small enough. When the search stops, $t_m = (t_b + t_e)/2$ is declared as the desired t' for this patch.

Algorithm 1 t searching algorithm

Input: I_p, A, θ_t
Output: t_m

```
t_e ← 1
t_b ← 0
while (t_e − t_b) > θ_t and t_e > t_b do
    t_m ← (t_e + t_b)/2
    I_d = dehaze(I_p, t_m, A)
    (a, b) = haze_patch_comparator(I_p, I_d)
    if a > b then {dehazed patch is bad}
        t_b ← t_m
    else {dehazed patch is good}
        t_e ← t_m
    end if
end while
t_m ← (t_e + t_b)/2
```

6.3.3 Aggregation and Interpolation of t(x)

Thus the transmittance parameter t is estimated from the patches as described in the previous step. If we consider overlapping patches, a pixel is likely to receive more than one estimate of t. These values are combined to obtain a single value of t at each pixel.

6.3.4 Haze-free image recovery

Once we obtain t(x) at every pixel, we can dehaze the input image. We use this computed t(x) along with the atmospheric light 'A' to obtain the dehazed result. Using the following equation we calculate the estimated dehaze image J_e^c (x) as:

$$J_e^c(\mathbf{x}) = A - \frac{A - I^c(\mathbf{x})}{\max\{t(\mathbf{x}), 0.0001\}} \qquad \textbf{(6.8)}$$

Note that, J_e^c (x) values lying beyond the valid intensity range are clipped to the valid range. Second, we assume that t(x) should be greater than zero, otherwise no scene information would reach the observer or the sensor (camera). To ensure this, we clip the value of t(x) arbitrarily at 0.0001 from lower. Third, unlike many other methods we do not employ any kind of post-processing technique.

Chapter 7

Results and Discussion

7.1 Results

Table 7.1: Denoising results with noise level 25

GreyScale Image	PSNR (in dB)	SSIM (in %)
01	30.24	93.15
02	33.09	94.34
03	30.84	94.00
04	29.35	92.43
05	30.42	95.56
06	29.08	93.16
07	29.44	92.66
08	32.39	94.15
09	30.09	93.46
10	30.20	90.78
11	30.08	90.68
12	30.10	91.35
Average	**30.45**	**92.98**

Table 7.2: PSNR and SSIM value of predicted clean images

Table 7.3: Denoising results with noise level 25

Color Image	PSNR (in dB)	SSIM (in %)
01	27.57	86.92
02	26.43	87.34
03	27.44	87.71
04	24.76	89.96
05	29.88	92.18
06	28.43	90.23
07	24.67	84.95
08	29.46	89.42
09	30.08	93.08
10	29.76	90.65
11	30.92	92.23
12	29.55	95.48
Average	**28.20**	**90.02**

Table 7.4: PSNR and SSIM value of predicted clean images

Denoising Model	PSNR(dB)	SSIM(%)
BM3D	28.57	80.13
WNNM	28.33	80.84
EPLL	28.68	82.86
MLP	28.96	80.94
CSF	28.74	81.68
TNRD	28.92	81.52
DnCNN (Our)	**30.45**	**92.98**

Table 7.5: Comparison of PSNR and SSIM values of different models with DnCNN

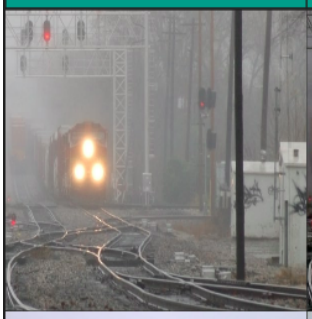
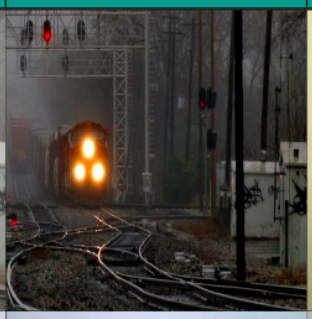
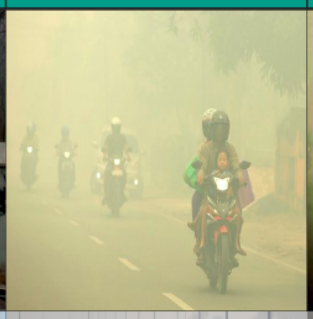
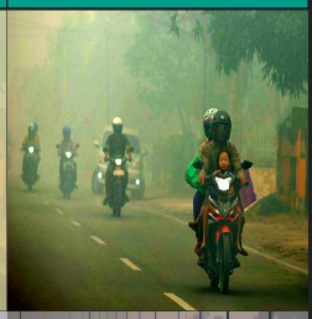
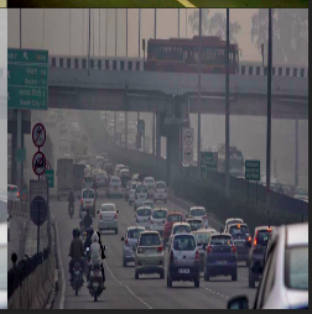
INPUT IMAGE	DEHAZED IMAGE	INPUT IMAGE	DEHAZED IMAGE
			
			

Table 7.6: Dehazing results of foggy images

Image	SSIM (in %)
01	93.0
02	95.2
03	85.6
04	94.8
05	96.7
06	96.8
07	93.5
08	93.4
09	96.9
10	94.8
11	96.9
12	95.2
Average	**94.4**

Table 7.7: SSIM value of predicted dehazed images

Dehazing Model	SSIM(%)
Pierre et al.	85.4
Fattal Color Line	96.8
Berman et al.	96.2
Ren et al.	84.5
(Our)	**96.9**

Table 7.8: Comparison of SSIM values of different dehazing models

7.2 Discussion

Training and Testing: For Gaussian denoising with either known or unknown noise level, we use standard data set for training. It is speculated that using a larger training dataset can only bring negligible improvements. To train DnCNN for Gaussian denoising with known noise level, we consider three noise levels, i.e., $\sigma = 15$ and 25. We set the patch size as 40×40, and crop $128 \times 1,600$ patches to train the model. We refer to our DnCNN model for Gaussian denoising with known specific noise level as DnCNN-S.

To train our patch comparator we synthesize hazy patches from clean haze-free patches and use standard data sets. Each pair consists of a haze patch and its dehazed version. We have taken patches of size 10×10 to train our comparator and also to dehaze the test image. We have evaluated our method on synthetic dataset as well as outdoor hazy images and compared the results with that of the state-of-the-art dehazing methods and one contrast enhancement based method.

Chapter 8

Summary and Conclusion

To sum up, our DnCNN model has two main features:

- the residual learning formulation is adopted to learn R(y), and
- batch normalization is incorporated to speed up training as well as boost the denoising performance.

By incorporating convolution with ReLU, DnCNN can gradually separate image structure from the noisy observation through the hidden layers.

Furthermore, the image dehazing method tries to estimate transmittance in each patch by comparing the dehazed version with the input hazy one. The comparison is done by the patch quality comparator. With this CNN based comparator in our hand, we employ binary search to find transmittance in each patch. Once transmittance is obtained we can recover the dehazed image.

The results show that our model performs better than other conventional denoising / dehazing techniques.

References

[1] Kai Zhang, Wangmeng Zuo, Yunjin Chen, Deyu Meng, and Lei Zhang, ***"Beyond a Gaussian Denoiser:*** *Residual Learning of Deep CNN for Image Denoising*," IEEE Transactions on Image Processing, vol. 26, pp. 3142 - 3155, July 2017.

[2] K. He, J. Sun, and X. Tang, "**Single Image Haze Removal Using Dark Channel Prior**," IEEE Transactions on Pattern Analysis and Machine Intelligence, vol. 33, pp. 2341–2353, 2011.

[3] Medium, "Towards Data Science", *https://towardsdatascience.com*.

[4] Wikipedia, "Convolutional Neural Network".

[5] K. He, J. Sun, and X. Tang, "**Single Image Haze Removal Using Dark Channel Prior,**" IEEE Transactions on Pattern Analysis and Machine Intelligence, vol. 33, pp. 2341–2353, 2011.

[6] Sanchayan Santra, Ranjan Mondal, and Bhabatosh Chanda, "**Learning a Patch Quality Comparator for Single Image Dehazing,**" IEEE Transactions on Image Processing, vol. 27, pp. 4598 - 4607, Sept. 2018.