Image Enhancement Techniques : Denoising, Dehazing and Deraining

Rajat Gupta (19BM6JP17)
Vineet Kumar (19BM6JP46)
Paturu Harish (19BM6JP55)

Abstract

Image enhancement techniques are used to refine a given image so that the desired image features are more perceptible to the human visual system. Technically, Image Enhancement which deals with estimating the clean, original image given a noisy, blur and hazy one.

Keywords: Image Enhancement, Computer Vision, De-noising, Gaussian Noise, Streak Removal, Deep Learning

Highlight: The work culminating out of this project has been accepted at **CVPR Workshop 2020**. The publication is available at CVPR Proceedings Website.
• We participated in the NTIRE Denoising Challenge and secured $\mathbf{16}^{th}$ position among **250+** registrants.
• Our results for image dehazing are comparable to the NTIRE 2019 Image Dehazing Challenge. Our scores position us among the **Top 16** participants of the challenge.
• Our results of Image Deraining Model is among current **state of the art** methods available.

1. Introduction

Image enhancement techniques allow the observer to see the details in the image that may not be immediately observable at first glance; such improvements remarkably improve the visual appeal as well as enhance the usefulness of the image for other downstream tasks. Image enhancement is a tricky problem of computer vision domain as transformation or mapping of one image to another is not necessarily one-to-one, so that two different input images may transform into the same or similar output images after enhancement. Furthermore, it cannot be expected that the enhancement algorithm provides information that does not exist in the original image. If the image does not contain any function to be enhanced, it may inadvertently increase noise or other unwanted image components without any benefit to the user.

Specifically we worked on three sub-problems in this domain –

- **Image Denoising** is technique to remove noise from a noisy image, so as to restore the true image. Denoising is a classic problem and has been studied for a long time. However, it remains a challenging and open task.

- **Image Dehazing** also known as "defogging" is the technique to reduce or even remove interference due to haze by special approaches, in order to obtain satisfactory visual effects and obtain more useful information.

- **Image Deraining** is process to remove the rain streak from the images to make it look clear and more informative for other downstream applications.

2. Related Works

With the demand for highly accurate and visually pleasing images increasing with time, several images are taken every day. However, these images captured by sophisticated and high-tech cameras are inevitably degraded by noise, which leads to deterioration in image quality. Thus, works are required that can reduce noise without losing image features. Owning to environmental factors, transmission through channels and others, images get inevitably contaminated by noise because of many reasons compression, acquisition, and transmission, which lead to image distortion and loss of information. Due to this, image processing tasks like image analysis, video processing, and tracking can be severely affected. Hence, image denoising is an active area of research in the domain of computer vision and plays a vital role in image processing. Image Denoising is a technique of removing noise or distortions from an image. The underlying principle of an image denoising algorithm is that while true signal pixel intensities are correlated to each other, noise is not Goyal et al. (2020).

Image denoising Fan et al. (2019) methods can be broadly classified into two categories:

Transform domain methods. these methods first transform the noisy image to another domain, followed by applying a denoising procedure on the transformed image according to characteristics of image and noise. These methods are subdivided into data-adaptive or non-data adaptive, based on chosen basis transform functions.

Spatial domain methods. these methods aim to remove noise by calculating gray value of each pixel based on the correlation between pixels/image patches in the original image. These methods are further divided into two categories variational denoising methods and spatial domain filtering.

With the scattering of atmospheric particles, which can reduce the contrast, change the color, and make the object features challenging to identify by human vision and by some outdoor computer vision systems, the image captured in hazy or foggy weather can be heavily occluded Wang and Yuan (2017). The following figure shows a comparison between a hazy image and non-hazy image. It can be seen how the contrast and colors are heavily affected due to scattered light due to haze.

Therefore, it is important to improve the visual effects of image and highlight image features. Image dehazing deals with these kinds of problems. Image dehazing refers to procedures that aim to remove haze amount in a hazy image and grant the degraded image an overall sharpened appearance to obtain clearer visibility and smooth image. It is highly desired to enhancing images taken under bad visibility or weather to obtain clearer and sharper images. Thus, image dehazing has been an important issue and widely researched in the field of computer vision. However, the problem of dehazing is challenging, and many researchers have been devoted to solving this.

The rest of the paper is organized as follows Section 3 deals with the mathematical framework of conventional image enhancement techniques. In Section 4, we discuss the evaluation metrics we used for judging the performance of our models. Section 5 – 7 and 8 describes our various approaches for image enhancement along with their results and a brief commentary. Finally, Section 9 concludes our paper by shedding light on a few possible future directions as well.

3. Mathematical Framework

We define some notations first. Let C be a clean natural image and N be either some noise, haze(fog) or rain streak depending on the problem. Also, we assume M is given image. Hence we may write

$$M = C \oplus N$$

In our setting we will have to find C from M. However, the inversion problem is not unique. In other words, $\oplus$ is not invertible. The ultimate objective of image enhancement in a broad sense is to improve the degraded image that can express all the information of the scene. As we have already seen above, mathematically it is impossible to solve the inverse problem in general.

4. Evaluation Metrics

In literature these tasks are evaluated on following two metrics:

Peak signal-to-noise ratio (PSNR). It is an expression for the ratio between maximum possible power of a signal and the power of corrupting noise that affects the fidelity of its representation. It is usually expressed in terms of logarithmic decibel scale, due to its wide dynamic range.

Structural Similarity (SSIM) Index. SSIM is used for measuring the similarity between two images. The SSIM Index is a method for predicting the perceived quality of digital television and cinematic pictures, as well as other kinds of digital images and videos. It is a full reference metric, which means that the measurement or prediction of image quality is based on an initial uncompressed or distortion-free image as reference.

5. Methodology & Technique

Common workflow for all three problem is common first we preprocess the dataset then train our model and finally evaluate the model through PSNR and SSIM Index.

6. Image Denoising

The aim is to obtain a denoised image in standard RGB (sRGB) color space with the best peak signal-to-noise-ration (PSNR) and structural similarity (SSIM) with reference to the ground truth images.

6.1. Dataset

The dataset made available has been derived from the (Smartphone Image Denoising Dataset) SIDD Dataset Abdelhamed et al. (2018). Due to the small aperture and sensor size, smartphone images have notably more noise than their DSLR counterparts. With denoising for smartphone images being an active research area, the authors have proposed a dataset of 30,000 noisy images from 10 scenes under different lighting conditions using five representative smartphone cameras. The "Data" directory contains 320 image pairs (noisy and ground-truth, in sRGB space) representing 160 scene instances, two images from each scene instance, gamma corrected. The images have been taken by the following smartphones under Low light, Normal brightness and High exposure setting –

- Google Pixel
- iPhone 7
- Samsung Galaxy S6 Edge
- Motorola Nexus 6
- LG G4

The validation set consists of 1024 noisy image blocks from both rawRGB and sRGB images, with each block of 256 × 256 pixels. Similarly, the SIDD+ testing set consists of 1024 noisy image blocks from a different set of images, but following the same format as validation set. Some sample images are shown in Figure 1.

Figure 1: SIDD Sample Images

6.2. Dataset Preprocessing

The original images are extremely high-quality images of dimensions 5328 × 3000. However, due to lack of computational resources, the images have been resized to 256 × 256 pixels using OpenCV library in Python.

6.3. Training the Model

A PyTorch implementation of **Stacked AutoEncoder**, originally based on MNIST Digit Recognition has been used. We used the reference code from here and modified it to suit our purpose.

	Encoder	Decoder
Total parameters	582,496	582,241
Trainable Parameter	582,496	582,241
Non-trainable Parameter	0	0

Table 1: Number of Parameters

While training, under each epoch, image pairs (original and noisy) were randomly shuffled and a Gaussian noise with mean and standard deviation of the difference between original and noisy image for all 160 image pairs was added.

6.3.1. Model Architecture

The following explains the architecture of the encoder and decoder modules used respectively in Figure 2 and 3:

Layer (type)	Output Shape	Param #
Conv2d-1	[-1, 32, 256, 256]	320
ReLU-2	[-1, 32, 256, 256]	0
BatchNorm2d-3	[-1, 32, 256, 256]	64
Conv2d-4	[-1, 32, 256, 256]	9,248
ReLU-5	[-1, 32, 256, 256]	0
BatchNorm2d-6	[-1, 32, 256, 256]	64
Conv2d-7	[-1, 64, 256, 256]	18,496
ReLU-8	[-1, 64, 256, 256]	0
BatchNorm2d-9	[-1, 64, 256, 256]	128
Conv2d-10	[-1, 64, 256, 256]	36,928
ReLU-11	[-1, 64, 256, 256]	0
BatchNorm2d-12	[-1, 64, 256, 256]	128
MaxPool2d-13	[-1, 64, 128, 128]	0
Conv2d-14	[-1, 128, 128, 128]	73,856
ReLU-15	[-1, 128, 128, 128]	0
BatchNorm2d-16	[-1, 128, 128, 128]	256
Conv2d-17	[-1, 128, 128, 128]	147,584
ReLU-18	[-1, 128, 128, 128]	0
BatchNorm2d-19	[-1, 128, 128, 128]	256
MaxPool2d-20	[-1, 128, 64, 64]	0
Conv2d-21	[-1, 256, 64, 64]	295,168
ReLU-22	[-1, 256, 64, 64]	0

Figure 2: Encoder Architecture

Table 1 shows the statistics regarding the volume of model in terms of learnable parameters.

6.4. Results

The results for the model trained on 1000 epochs on training set are shown in 2.

The Figure 4 highlights the model performance in removing noise from images. The comparison between original, noisy and denoised images are shown in Appendix .12.

Layer (type)	Output Shape	Param #
ConvTranspose2d-1	[-1, 128, 128, 128]	295,040
ReLU-2	[-1, 128, 128, 128]	0
BatchNorm2d-3	[-1, 128, 128, 128]	256
ConvTranspose2d-4	[-1, 128, 128, 128]	147,584
ReLU-5	[-1, 128, 128, 128]	0
BatchNorm2d-6	[-1, 128, 128, 128]	256
ConvTranspose2d-7	[-1, 64, 128, 128]	73,792
ReLU-8	[-1, 64, 128, 128]	0
BatchNorm2d-9	[-1, 64, 128, 128]	128
ConvTranspose2d-10	[-1, 64, 128, 128]	36,928
ReLU-11	[-1, 64, 128, 128]	0
BatchNorm2d-12	[-1, 64, 128, 128]	128
ConvTranspose2d-13	[-1, 32, 128, 128]	18,464
ReLU-14	[-1, 32, 128, 128]	0
BatchNorm2d-15	[-1, 32, 128, 128]	64
ConvTranspose2d-16	[-1, 32, 128, 128]	9,248
ReLU-17	[-1, 32, 128, 128]	0
BatchNorm2d-18	[-1, 32, 128, 128]	64
ConvTranspose2d-19	[-1, 1, 256, 256]	289
ReLU-20	[-1, 1, 256, 256]	0

Figure 3: Decoder Architecture

Epochs	Mean PSNR	Mean SSIM
200	22.814	0.6773
400	23.408	0.6955
600	25.966	0.7705
800	**26.952**	**0.7796**
1000	26.466	0.7541

Table 2: Model Performance

7. Image Dehazing

In image dehazing too, our goal is obatain better SSIM and PSNR metrics. We first preprocessed the Dense-Haze dataset, Trained and implemented **Denoising AutoEncoder** and evaluate the model through PSNR and SSIM Index finally Extending work on **CycleGANs** Zhu et al. (2017) and **Pix2Pix** Isola et al. (2017) and comparing their performances.

7.1. Training using Autoencoder

The identical data preprocesing method and autoencoder as described in previous Section 6.3.1 was used for initial dehazing attempt.

Figure 4: removing noise from images

Description	Mean PSNR	Mean SSIM
Train	23.546	0.738
Test	10.244	0.537

Table 3: Dehazing results for the model trained on 500 epochs

7.1.1. Results

Figure 5 shows the performance of the model on our dataset set. Model performance on a random imagetaken from internet is shown in Appendix Figure .13.

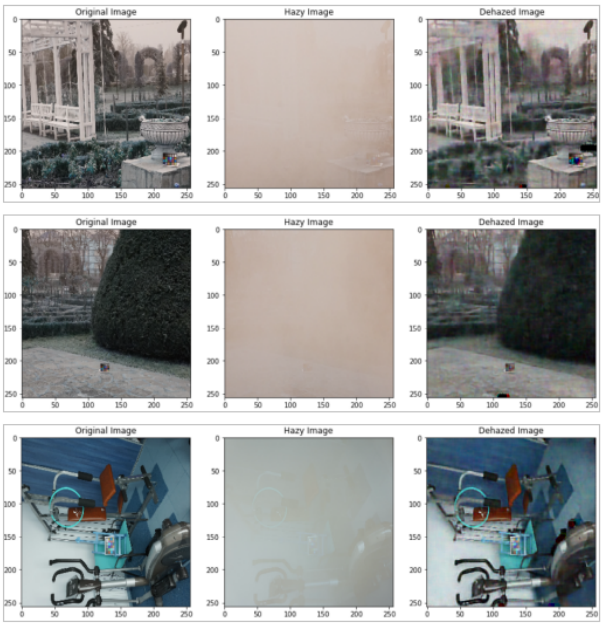

Figure 5: Model performance on images

7.2. Training CycleGANs and Pix2Pix

Image Dehazing can also be looked as an Image Translation task where the objective is to transform an image in one style to another. This transformation is also commonly known as a Style Transfer effect. Two state-of-the-art approaches that perform this effect are — CycleGANs and Pix2Pix.
Often in many style transfer applications, paired data is required for training. While Pix2Pix requires paired data, CycleGANs do not. A random collection of images in style A and another random collection of images in style B is all what is required to perform the image translation in CycleGANs and hence, it induces more flexibility. Common applications of these approaches are shown in Figure 6.

7.3. CycleGANs Methodology

The dataset has been splitted with 45 images (80%) in training set and 10 images (20%) in test set. In Table 4 detailed information about the hyperparameters is reproduced.

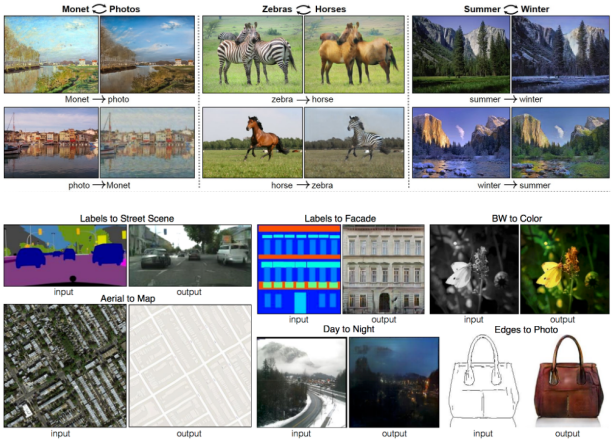

Figure 6: Common applications of CycleGANs and Pix2Pix

Hyperparameters	Value
Epochs	1000
GAN-mode/objective	Least Square GAN
Learning Rate	0.0002
Batch Size	1
Image Size	256

Table 4: Hyperparameter for CycleGANs

7.4. CycleGAN Results

Results for CycleGAN is shown in Table 7. Images

Description	Mean PSNR	Mean SSIM
Training	15.6	0.503
Testing	13.44	0.438

Table 5: results for CycleGAN

that show the performance of cycleGAN model on the test dataset/Holdout set is shown in Figure 7.

7.5. Pix2Pix Methodology

The dataset has been splitted with 45 images in training set (80%), 5 images in validation set (10%) and 5 images in test set (10%). Combined images from both styles (occluded/hazy and non-occluded/non-hazy) are needed to be created to be fed as input to the model for training. In Table 6 detailed information about the hyperparameters is reproduced.

Figure 7: CycleGAN performance

Hyperparameters	Value
Epochs	1000
GAN-mode/objective	Cross Entropy
Learning Rate	0.0002
Batch Size	1
Image Size	256

Table 6: Hyperparameter for Pix2Pix

7.6. Pix2Pix Results

Results for Pix2Pix is shown in Table 7. Images that show the performance of Pix2Pix model on the test dataset/Holdout set is shown in Figure 8.

Description	Mean PSNR	Mean SSIM
Training	22.434	0.722
Testing	15.963	0.602

Table 7: results for CycleGAN

Figure 8: Pix2Pix performance

8. Image Deraining

We used GANs based Generator and Discriminator coupled architecture for our deraining model. Specifically Figure 9 shows the generator and Figure 10 shows discriminator architecture. Also statistics regarding the volume of model in terms of learnable parameters is shown in Table 10.

Model: "Generator"

Layer (type)	Output Shape	Param #	Connected to
input_1 (InputLayer)	(None, 256, 256, 3)	0	
1stconv (Conv2D)	(None, 256, 256, 64)	1792	input_1[0][0]
batch_normalization_1 (BatchNor	(None, 256, 256, 64)	256	1stconv[0][0]
activation_1 (Activation)	(None, 256, 256, 64)	0	batch_normalization_1[0][0]
conv2d_2 (Conv2D)	(None, 256, 256, 64)	102464	activation_1[0][0]
activation_3 (Activation)	(None, 256, 256, 64)	0	conv2d_2[0][0]
conv2d_1 (Conv2D)	(None, 256, 256, 64)	36928	activation_1[0][0]
batch_normalization_3 (BatchNor	(None, 256, 256, 64)	256	activation_3[0][0]
activation_2 (Activation)	(None, 256, 256, 64)	0	conv2d_1[0][0]
activation_4 (Activation)	(None, 256, 256, 64)	0	batch_normalization_3[0][0]
batch_normalization_2 (BatchNor	(None, 256, 256, 64)	256	activation_2[0][0]
batch_normalization_4 (BatchNor	(None, 256, 256, 64)	256	activation_4[0][0]
average_1 (Average)	(None, 256, 256, 64)	0	batch_normalization_2[0][0] batch_normalization_3[0][0] batch_normalization_4[0][0]
add_1 (Add)	(None, 256, 256, 64)	0	activation_1[0][0] average_1[0][0]
conv2d_5 (Conv2D)	(None, 256, 256, 64)	102464	add_1[0][0]
activation_6 (Activation)	(None, 256, 256, 64)	0	conv2d_5[0][0]
conv2d_4 (Conv2D)	(None, 256, 256, 64)	36928	add_1[0][0]
batch_normalization_6 (BatchNor	(None, 256, 256, 64)	256	activation_6[0][0]
activation_5 (Activation)	(None, 256, 256, 64)	0	conv2d_4[0][0]
activation_7 (Activation)	(None, 256, 256, 64)	0	batch_normalization_6[0][0]
batch_normalization_5 (BatchNor	(None, 256, 256, 64)	256	activation_5[0][0]
batch_normalization_7 (BatchNor	(None, 256, 256, 64)	256	activation_7[0][0]
average_2 (Average)	(None, 256, 256, 64)	0	batch_normalization_5[0][0] batch_normalization_6[0][0] batch_normalization_7[0][0]
add_2 (Add)	(None, 256, 256, 64)	0	add_1[0][0] average_2[0][0]
conv2d_8 (Conv2D)	(None, 256, 256, 64)	102464	add_2[0][0]
activation_9 (Activation)	(None, 256, 256, 64)	0	conv2d_8[0][0]
conv2d_7 (Conv2D)	(None, 256, 256, 64)	36928	add_2[0][0]
batch_normalization_9 (BatchNor	(None, 256, 256, 64)	256	activation_9[0][0]
activation_8 (Activation)	(None, 256, 256, 64)	0	conv2d_7[0][0]
activation_10 (Activation)	(None, 256, 256, 64)	0	batch_normalization_9[0][0]
batch_normalization_8 (BatchNor	(None, 256, 256, 64)	256	activation_8[0][0]
batch_normalization_10 (BatchNo	(None, 256, 256, 64)	256	activation_10[0][0]
average_3 (Average)	(None, 256, 256, 64)	0	batch_normalization_8[0][0] batch_normalization_9[0][0] batch_normalization_10[0][0]
add_3 (Add)	(None, 256, 256, 64)	0	add_2[0][0] average_3[0][0]
conv2d_10 (Conv2D)	(None, 256, 256, 64)	4160	add_3[0][0]
activation_11 (Activation)	(None, 256, 256, 64)	0	conv2d_10[0][0]
add_4 (Add)	(None, 256, 256, 64)	0	activation_11[0][0] add_3[0][0]
conv2d_11 (Conv2D)	(None, 256, 256, 64)	4160	add_4[0][0]
activation_12 (Activation)	(None, 256, 256, 64)	0	conv2d_11[0][0]
add_5 (Add)	(None, 256, 256, 64)	0	activation_12[0][0] add_4[0][0]
conv2d_12 (Conv2D)	(None, 256, 256, 64)	4160	add_5[0][0]
activation_13 (Activation)	(None, 256, 256, 64)	0	conv2d_12[0][0]
add_6 (Add)	(None, 256, 256, 64)	0	activation_13[0][0] add_5[0][0]
conv2d_13 (Conv2D)	(None, 128, 128, 64)	4160	add_6[0][0]
batch_normalization_11 (BatchNo	(None, 128, 128, 64)	256	conv2d_13[0][0]
conv2d_14 (Conv2D)	(None, 64, 64, 64)	4160	batch_normalization_11[0][0]
batch_normalization_12 (BatchNo	(None, 64, 64, 64)	256	conv2d_14[0][0]
conv2d_15 (Conv2D)	(None, 64, 64, 3)	1731	batch_normalization_12[0][0]
up_sampling2d_1 (UpSampling2D)	(None, 128, 128, 3)	0	conv2d_15[0][0]
conv2d_16 (Conv2D)	(None, 128, 128, 256	7168	up_sampling2d_1[0][0]
activation_14 (Activation)	(None, 128, 128, 256	0	conv2d_16[0][0]
conv2d_17 (Conv2D)	(None, 128, 128, 3)	6915	activation_14[0][0]
up_sampling2d_2 (UpSampling2D)	(None, 256, 256, 3)	0	conv2d_17[0][0]
conv2d_18 (Conv2D)	(None, 256, 256, 256	7168	up_sampling2d_2[0][0]
activation_15 (Activation)	(None, 256, 256, 256	0	conv2d_18[0][0]
conv2d_19 (Conv2D)	(None, 256, 256, 3)	6915	activation_15[0][0]

Total params: 473,737
Trainable params: 472,201
Non-trainable params: 1,536

Figure 9: Generator Architecture

```
Model: "model_1"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
input_2 (InputLayer)         (None, 256, 256, 3)       0
_________________________________________________________________
conv2d_20 (Conv2D)           (None, 256, 256, 64)      1792
_________________________________________________________________
leaky_re_lu_1 (LeakyReLU)    (None, 256, 256, 64)      0
_________________________________________________________________
conv2d_21 (Conv2D)           (None, 128, 128, 64)      36928
_________________________________________________________________
leaky_re_lu_2 (LeakyReLU)    (None, 128, 128, 64)      0
_________________________________________________________________
batch_normalization_13 (Batc (None, 128, 128, 64)      256
_________________________________________________________________
conv2d_22 (Conv2D)           (None, 128, 128, 128)     73856
_________________________________________________________________
leaky_re_lu_3 (LeakyReLU)    (None, 128, 128, 128)     0
_________________________________________________________________
batch_normalization_14 (Batc (None, 128, 128, 128)     512
_________________________________________________________________
conv2d_23 (Conv2D)           (None, 64, 64, 128)       147584
_________________________________________________________________
leaky_re_lu_4 (LeakyReLU)    (None, 64, 64, 128)       0
_________________________________________________________________
batch_normalization_15 (Batc (None, 64, 64, 128)       512
_________________________________________________________________
conv2d_24 (Conv2D)           (None, 64, 64, 256)       295168
_________________________________________________________________
leaky_re_lu_5 (LeakyReLU)    (None, 64, 64, 256)       0
_________________________________________________________________
batch_normalization_16 (Batc (None, 64, 64, 256)       1024
_________________________________________________________________
conv2d_25 (Conv2D)           (None, 32, 32, 256)       590080
_________________________________________________________________
leaky_re_lu_6 (LeakyReLU)    (None, 32, 32, 256)       0
_________________________________________________________________
batch_normalization_17 (Batc (None, 32, 32, 256)       1024
_________________________________________________________________
conv2d_26 (Conv2D)           (None, 32, 32, 512)       1180160
_________________________________________________________________
leaky_re_lu_7 (LeakyReLU)    (None, 32, 32, 512)       0
_________________________________________________________________
batch_normalization_18 (Batc (None, 32, 32, 512)       2048
_________________________________________________________________
conv2d_27 (Conv2D)           (None, 16, 16, 512)       2359808
_________________________________________________________________
leaky_re_lu_8 (LeakyReLU)    (None, 16, 16, 512)       0
_________________________________________________________________
batch_normalization_19 (Batc (None, 16, 16, 512)       2048
_________________________________________________________________
dense_1 (Dense)              (None, 16, 16, 1024)      525312
_________________________________________________________________
leaky_re_lu_9 (LeakyReLU)    (None, 16, 16, 1024)      0
_________________________________________________________________
flatten_1 (Flatten)          (None, 262144)            0
_________________________________________________________________
dense_2 (Dense)              (None, 1)                 262145
=================================================================
Total params: 5,480,257
Trainable params: 5,476,545
Non-trainable params: 3,712
_________________________________________________________________
```

Figure 10: Discriminator Architecture

8.1. Dataset preprocessing

For our experiments we used two datasets 100H and 100L having heavy as well as light rain streaks respectively. Both datasets have been synthesized from BSD200. However, the size of dataset is 320px $\times$ 480px. We did a uniform downsizing of all images and converted the images to numpy pickle dump for faster processing. Description of training data is shown in Table 8. For test set we had similarly, 235 images in each category. The size of thus created numpy dump was 1.32 GB each.

Set	type	category	number
Train	Heavy	Rain	1800
Train	Heavy	Norain	1800
Train	Light	Rain	1800
Train	Light	Norain	1800

Table 8: Dataset Size

Parameters	Generator	Discriminator
Total	473,737	5,480,257
Trainable	472,201	5,476,545
Non-trainable	1,536	3,712

Table 9: Model Statistics & Parameter

8.2. Methodology

The hyperparameters related with model is shown in Table 10. We also used transfer learning for getting the weights from standard VGG model, which was used subsequently in building our generator.

Hyper-parameters	Value
Epochs	100
GAN-mode/objective	Cross Entropy
Learning Rate	0.0001
Optimizer	Adam
Batch Size	1
Image Size	256px

Table 10: Hyperparameter for Derain Model

8.3. Results

Results from our Deraining model is shown in Table 11. Sample output from our deraining model is shown Figure 11.

Description	Mean PSNR	Mean SSIM
Training	35.605	0.7669
Testing	35.582	0.6817

Table 11: Deraining Results

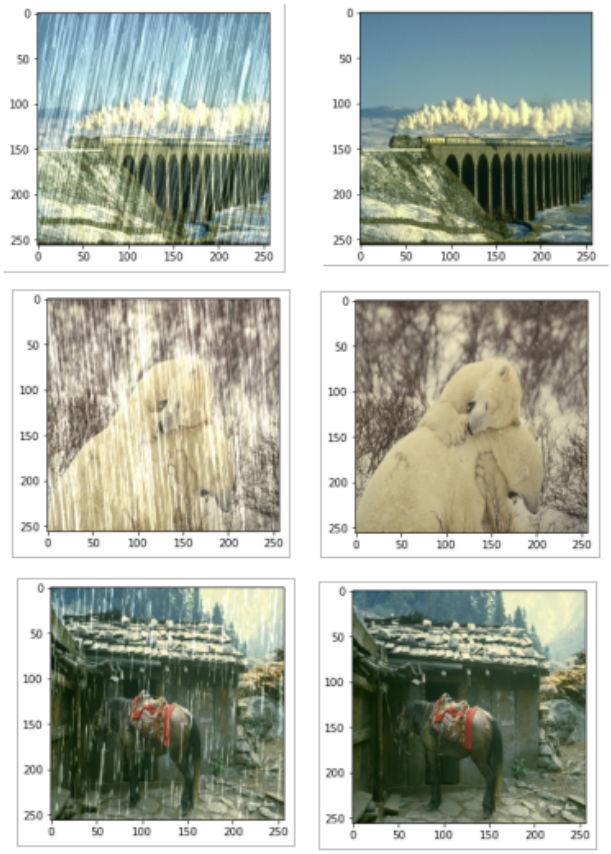

Figure 11: Sample Derained Image

9. Conclusion & Discussions

Comparison between CycleGAN and Pix2Pix. The comparison between the performance of CycleGANs and Pix2Pix is shown in Figure .14. Under same number of epochs, Pix2Pix seems to have a better performance as compared to CycleGANs.
Table 12 presents an overview of the results.

Conclusion. Although there are many novel techniques and state-of-the-art approaches (for e.g. Residual Dense Network (RDN)) which particularly deal with the problem of removing noise from image (**Image Denoising**) or converting to high-resolution images (Super-Resolution), our basic PyTorch implementation has been able to perform quite well without the usage of huge computing resources.
The auto-encoder performed extremely well for the training of **dehazing** but poor for testing set, which means it wasnt able to generalize well. The model has likely overfitted, reasons to which can be mainly attributed to the small size of dataset (only 55 images). The novel image translation approaches like CycleGAN and Pix2Pix have been able to generalize the task of dehazing well even with the small dataset. However, Generative Adversarial Networks (GANs) in general, are quite slow in learning and extremely hectic to train, which is why the models have been trained for as low as 1000 epochs, in order to show noteworthy results.
For **deraining** we have introduced a new deep learning based method to effectively remove streaks from a single image, even in the presence of rain streak accumulation and heavy rain. Experiments on synthetic dataset demonstrate that the proposed method competes equally with many state-of-the-art methods.

Future Directions. **Image Denoising:** Residual Dense Networks (RDNs) have known to be the current state-of-the-art approach in terms of dealing with tasks like Denoising and Super-Resolution. One could try such approaches and compare performance with the currently used approaches.
Image Dehazing: As dehazing can be seen as an image translation task, so can one also think about converting a non-hazy image to a hazy image as another image translation task. With CycleGANs not requiring paired data in general, one can try generating hazy images from non-hazy images using the model originally trained for the purpose of dehazing.

Technique	Model	Epochs	Mean PSNR	Mean SSIM
Denoising	Stacked Autoencoder	1000	26.952	0.7796
Dehazing	Stacked Autoencoder	500	10.244	0.537
	Pix2Pix	1000	15.963	0.602
	CycleGAN	1000	13.44	0.538
Deraining	GAN based Model	100	35.582	0.6817

Table 12: Result Summary

Image Deraining: Although our method perform very well but this model is not suitable where computational cost is limited given the model volume (almost 58 million parameters). We can try for reducing the number of parameters without compromising the quality. There has been multiple advances in this direction specifically LPNET, SPA Net.

10. Acknowledgement

We would like to thank Prof. Sujoy Bhattacharya for allowing us to pursue such an interesting topic for our course project.

References

Abdelhamed A, Lin S, Brown MS. A high-quality denoising dataset for smartphone cameras. In: IEEE Conference on Computer Vision and Pattern Recognition (CVPR). 2018. .

Fan L, Zhang F, Fan H, Zhang C. Brief review of image denoising techniques. Visual Computing for Industry, Biomedicine, and Art 2019;2(1):7.

Goyal B, Dogra A, Agrawal S, Sohi B, Sharma A. Image denoising review: From classical to state-of-the-art approaches. Information Fusion 2020;55:220–44.

Isola P, Zhu JY, Zhou T, Efros AA. Image-to-image translation with conditional adversarial networks. In: Proceedings of the IEEE conference on computer vision and pattern recognition. 2017. p. 1125–34.

Wang W, Yuan X. Recent advances in image dehazing 2017;.

Zhu JY, Park T, Isola P, Efros AA. Unpaired image-to-image translation using cycle-consistent adversarial networks. In: Proceedings of the IEEE international conference on computer vision. 2017. p. 2223–32.

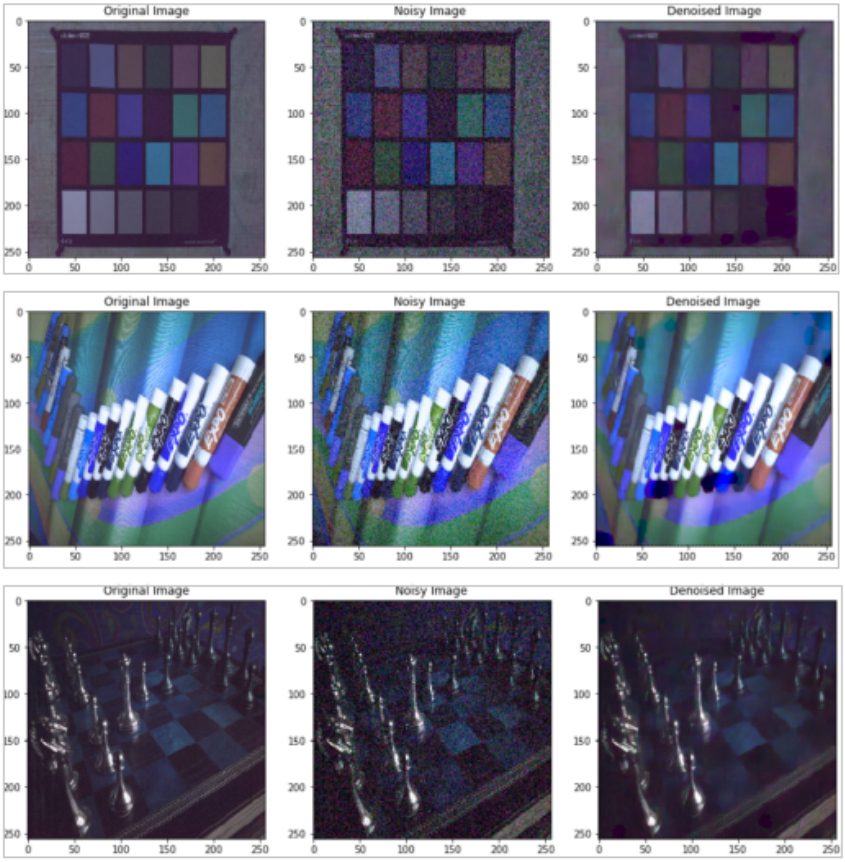

Figure .12: comparison between images (original, noisy and denoised)

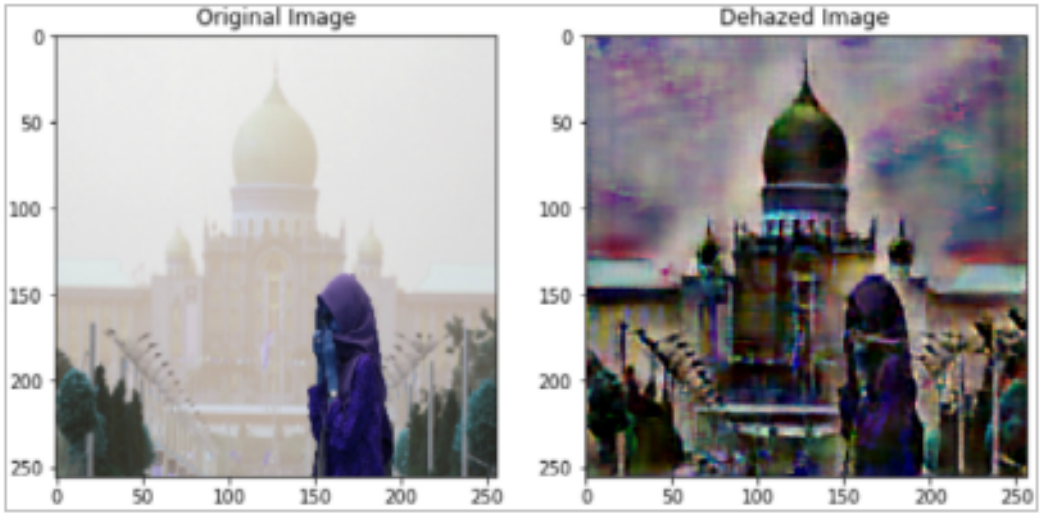

Figure .13: Random image from internet for dehazing

Figure .14: Comparison of CycleGAN and Pix2Pix