DEEP LEARNING FOR SIGNAL AND IMAGE PROCESSING

Single Image Dehazing using CycleGAN

Class: (III-B.Tech-CSE-AI (Semester 6)), Batch (2019-23)
Course: 21AIE312(21-22(Even))
Date: 05.05.2022

Team 3

Adhithan P	CB.EN.U4AIE19003
Anuvarshini S P	CB.EN.U4AIE19011
Kabilan N	CB.EN.U4AIE19033
Sivamaran M A C	CB.EN.U4AIE19061

Abstract:

In this project, we present an end-to-end network using CycleGAN, for a single image dehazing problem, which does not require pairs of hazy and corresponding ground truth images for training. That is, we train the network by feeding clean and hazy images in an unpaired manner. Image-to-image translation is a class of vision and graphics problems where the goal is to learn the mapping between an input image and an output image using a training set of aligned image pairs. However, for many tasks, paired training data will not be available. CycleGAN learns to translate an image from a source domain X to a target domain Y in the absence of paired examples. The goal here is to learn a mapping $G: X \rightarrow Y$ such that the distribution of images from $G(X)$ is indistinguishable from the distribution Y using an adversarial loss. Because this mapping is highly under-constrained, it is coupled with an inverse mapping $F: Y \rightarrow X$ and a cycle consistency loss to enforce $F(G(X)) \approx X$ (and vice versa). The results are evaluated with different metrics such as PSNR and SSIM.

Keywords: Generator, Discriminator, Adversarial Loss, Cycle-Consistency Loss.

Introduction:

Haze is caused by lack of transparency in air. Majorly caused by mist, fog and dust particles suspended in air. Because of these the light entering the atmosphere is absorbed and scattered. Hazing atmosphere decreases the quality of the images by reducing visibility and color fidelity. Dehazing has numerous applications in domains like astronomy, medical sciences, remote sensing, surveillance, web mapping, land-use planning, agronomy, archaeology, and environmental studies. In this project, we used CycleGAN for dehazing hazed images. Here we used a Custom generated unpaired dataset created from NYC Depth dataset [5]. Our dataset contains 51 hazed and 56 dehazed images. To evaluate the model, we created a custom dataset with just 10 hazed images out of which 5 are indoor and 5 are outdoor.

Literature:

The problem of Image dehazing has been tried to solve by using multiple different methods over the period of time. In paper [2], author introduces An End-to-End System for Single Image
Haze Removal using Convolution Neural Networks called DehazeNet. Single Image Haze Removal using a conditional GAN was done in [3]. [4], does image dehazing using variational mode decomposition technique. There have already been many image dehazing work done on Different GAN models, but most of the GAN models require paired images, sometimes it’s not possible to collect paired images for this problem, so in our work we used CycleGAN, which does not require paired images to train on.

Objective:

The main objective of our problem is to dehaze the hazed image using CycleGAN, with unpaired data for hazed and dehazed images, with just the limited data.

Theoretical Background:

GAN:

Generative Adversarial Networks, or GAN are an approach to generative modeling using deep learning methods. The generator performs an unsupervised learning task, that automatically discovers and learns the regularities or patterns in input data in such a way that the distribution of the original data and the distribution of the generated data is equal. GAN has a lot of use cases in image-to image translation, because of its ability to generate realistic images from the given input image. The GAN model architecture involves two neural networks: a generator and a discriminator.

Generator: Model that is used to generate samples from the latent samples such that it is similar to the original samples.

Discriminator: Model that is used to classify samples as real or fake. It takes in input from the real samples as well as the generated samples and predicts which class it belongs to. 0 if sample is fake and 1 if the sample is real.

Generative adversarial networks are based on a two-player minimax game where the players are the ‘generator’ and the ‘discriminator. The word ‘adversarial’ means ‘conflict’ pr ‘opposition’. The generator network directly produces samples. Its opponent, the discriminator network, distinguishes between samples drawn from the training data and samples drawn from the generator.

Discriminator tries to maximize the probability that it predicts correct whereas the generator tries to minimize the probability that the discriminator predicting correct.

Generator Model:

The generator model takes a fixed random vector (Latent vector) as input and generates a sample in the domain, such as an image. In our case it takes the hazed image as input and generates a dehazed image as output. From a random distribution ‘Z’, this model tries to generate the same distribution as close to the original distribution ‘X’.

Discriminator Model:

The discriminator model takes inputs from the generator and the real samples and predicts a binary class label of real or fake (generated). The real example comes from the training dataset. The generated examples are the output of the generator model. The discriminator is a normal classification model.

The key factor of GAN is that: one model is kept fixed or constant while training the other i.e., the training of generator is kept fixed while training the discriminator and vice versa. This is because, for ex., if the discriminator gets trained when the generator is learning, it learns on how the generator is trying to fool it and gets over smart and by doing so, the generator will never be able to win over the discriminator.

The discriminator is updated to get better at discriminating real and fake samples in the next round, and importantly, the generator is updated based on how well, or not, the generated samples fooled the discriminator.

The model parameters (weights and biases) get updated when one loses to the other, i.e., when the discriminator predicts correctly, there is no change required in change of its model parameters whereas the generator requires large updates in weights and biases. Similarly, when the generator fools the discriminators, the discriminator will require larger updates in the model parameters. At a point, generator fools the discriminator by creating perfect replicas of the original image, discriminators predict probability of [0.5,0.5] (Fake, Real). Then the discriminator is trained to correctly classify samples as real or fake.

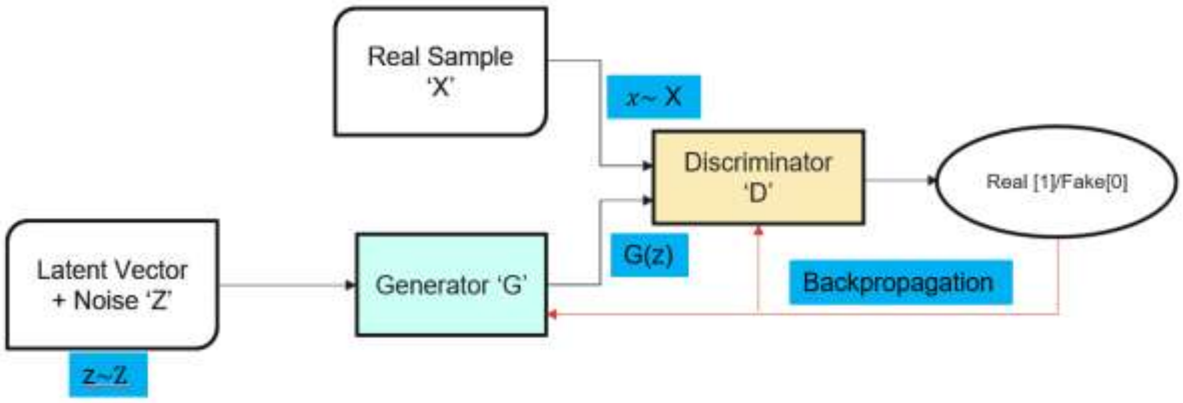


Fig 1: Flow graph for GAN

X is the random instance from real data, z is the random instance from random samples, G(z) is the generated output from random sample data. Discriminator's output can be represented in two ways based on the type of the input it takes. If it takes real data as input, its output can be written as D(x) when the input data is the output of the generator; this can be written as D(G(z)). Discriminator tries to maximize the probability that it predicts correctly, in other way, it tries to make D(G(z)) near to 0. Whereas the generator tries to minimize the probability that the discriminator predicting is correct, in other way, it tries to make D(G(z)) near to 1.

Loss function computation:

The choice of loss function is directly related to the effectiveness and robustness of the mode. As the output of the discriminator is a binary value (It predicts the image is real or fake). So, the loss function used is binary cross entropy loss. Binary cross entropy loss can be written as

$$L(\hat{y}, y) = y\, log(\hat{y}) + (1 - y)\, log(1 - \hat{y}) \rightarrow 1$$

When the input of the discriminator is from original data, y = 1 and $\hat{y}$ = D(x). then the loss function will be,

$$L(D(x),1) = \log(D(x)) \rightarrow 2$$

When the input of the discriminator is from the generator, y = 0 and $\hat{y}$= D(G(z)). then the loss function will be,

$$L(D(G(z)),0) = \log(1 - D(G(z))) \rightarrow 3$$

The objective of the discriminator is to classify the real and fake images correctly i.e ., D(x)=1 and D(G(z))=0.

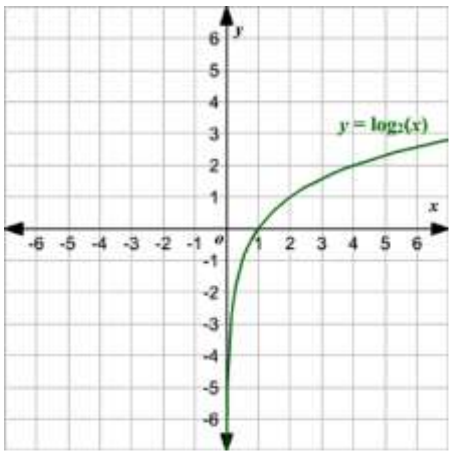


Fig 2 (a): Plot for equation (2)

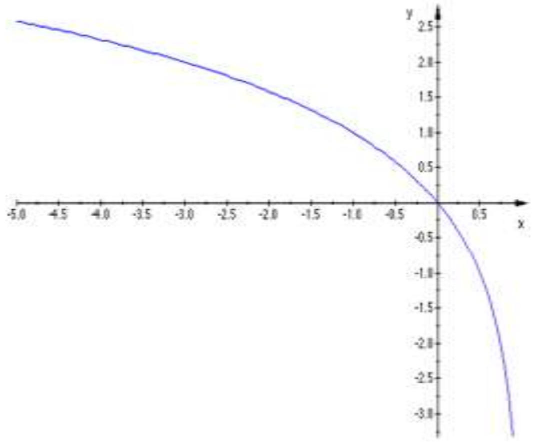


Fig 2 (b): Plot for equation (3)

From the above figure(2a), the x-axis represents L(D(x),1). The region after 1 at x-axis is not required as it is a probability function. We can see that D(x) = 1 when log(D(x)) is maximized. From the above figure(2b), we can see that D(G(z)) = 0 when log(1-D(G(z))) is maximized. To optimize the discriminator, we need to force D(x) to 1 and D(G(z)) to 0. From the plots we can say that, only if we maximize Fig 2a and Fig 2b, it will force D(x) = 1 and D(G(z)) = 0. Thus, the discriminator objective function is:

$$D^* = \max\left\{\log(D(x)) + \log(1 - D(G(z))\right\}$$

The Objective of the Generator is to fool the discriminator by generating samples that look much similar to that of the original ones, i.e., to make D(G(z)) =1. From Fig 2, we can see that D(G(z)) is 1 when log(1-D(G(z))) is minimized as the value tends to $-\infty$. Thus, the generator objective function can be written as

$$G^* = \min\left\{\log(D(x)) + \log(1 - D(G(z))\right\}$$

Clubbing the above objective function, the final equation can be written as

$$D^*, G^* = \min_G \max_D \left\{\log(D(x)) + \log(1 - D(G(z))\right\}$$

The above equation is for one instance of ‘X’. So, in order to consider all the instances of ‘X’, We will be taking ‘Expectation’ for the above argument. Thus, the equation can be written as:

$$\min_G \max_D V(D,G) = \min_G \max_D \left\{ \begin{array}{l} E_{x\sim P_{data}(x)}\left[\log(D(x))\right] + \\ E_{z\sim P_Z(z)}\left[\log(1 - D(G(z)))\right] \end{array} \right\}$$

Discriminator and Generator G play the following two-player minimax game with value function V (D, G). Where the generator tries to minimize the value function while its adversary, the discriminator tries to maximize it.

Steps for optimizing value function:

Learning for Generator:

1. Fix the Learning of 'G' and for Inner Loop for 'D':
2. Take 'm' noise samples from generated data $P_g(x)$
3. Take 'm' samples from original data $P_{data}(x)$
4. Update Discriminator 'D' (θ_d) by Gradient Ascent (because 'D' should be carried out with maximum optimization)

$$\nabla_{\theta_d} \frac{1}{m} \sum_{i=1}^{m} \left[\log(D(x^{(i)})) + \log(1 - D(G(z^{(i)}))) \right]$$

In the above equation θ_d is the parameters i.e., weights and bias. And the expectation operator is converted into averaging operator.

First, the learning starts with the discriminator as the backpropagation starts from the output predicted by the discriminator. The 'generator' is fixed while the 'discriminator' is being trained and vice versa.

During the training of the discriminator, there is an inner loop which runs for 'k' loops where k is a hyperparameter.

- The first step is to take 'm' real samples from original data and 'm' fake samples from the latent vector with noisy data.
- Next step is to update the discriminator by using gradient ascent. It is ascent because the discriminator's objective is to maximize.

Learning for Discriminator:

1. Fix the Learning of 'D'
2. Take 'm' noise samples from generated data $P_g(x)$
3. Update Generator 'G' (θ_g) by Gradient Descent (because 'G' should be carried out with minimum optimization)

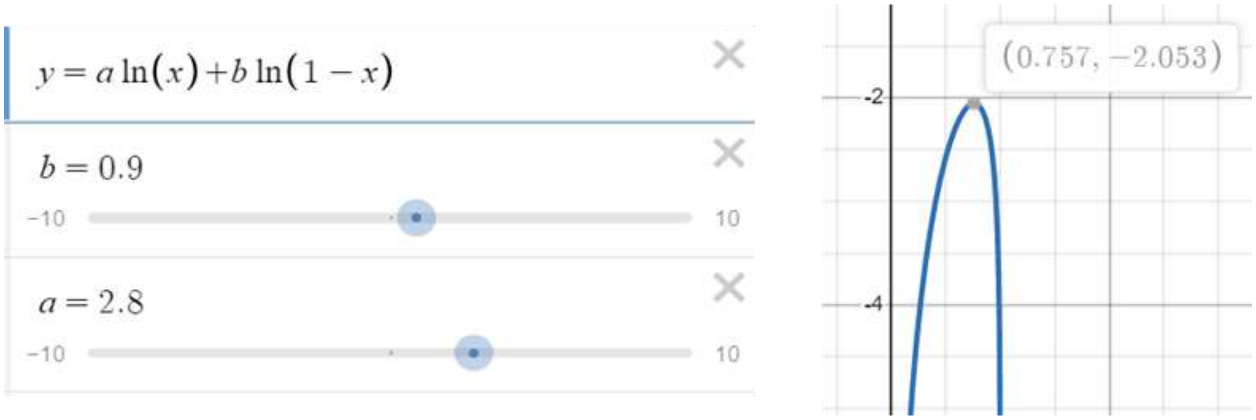
$$\nabla_{\theta_g} \frac{1}{m} \sum_{i=1}^{m} \left[\log(1 - D(G(z^{(i)}))) \right]$$

Now that the discriminator has trained for k loops, we fix the learning of 'D' and train the generator. For each 'k' loop training of the discriminator, the generator gets updated once.
Here, the first step is to take 'm' fake samples as the generator only takes in the fake samples.

Next step is to update the generator by using gradient descent. It is decent because the generator's objective is to minimize.

We held one network constant while training the other because the ultimate goal of the generator is to fool the discriminator, but if the discriminator keeps learning even during the training of the generator, it becomes smart enough to not get fooled by the generator. By doing so, the generator will never be able to win over the discriminator. What is the guarantee that the generator will replicate the same distribution as the original distribution? In other words, we have to prove that $P_g(x)$ will converge to $P_{data}(x)$ when the generator achieves the global minimum of the value function. This is a two-step process. First, for fixed G, for which value of the Discriminator, the Value Function is maximum.

$$V(G,D) = \int_x P_{data}(x)\ln[D(x)] + P_g(x)\ln[1-D(x)]dx$$



$y = a\ln(x) + b\ln(1-x)$

$b = 0.9$

$a = 2.8$

Fig 3: Plot for equation y=a ln(x)+b ln(1-x)

From the figure 3, we can tell that the equation of 'y' is similar to that of our value function.
y=a ln(x)+b ln(1-x)
We can map the above equation to out value function where:

- a=$P_{data}(x)$
- b=$P_g(x)$
- x=D(x)

Thus, the maximum value of 'x' for any 'a' and 'b' is $\frac{a}{a+b}$, which can be inferred from the above plot.
Thus, the value function is maximum when, D(x) = $\frac{P_{data}(x)}{P_{data}(x)+P_g(x)}$.
By substituting D(x) in value function, it becomes

$$\min_G V = E_{x\sim P_{data}(x)} \ln\left(\frac{P_{data}(x)}{P_{data}(x)+P_g(x)}\right) + E_{x\sim P_g(x)} \ln\left(1-\frac{P_{data}(x)}{P_{data}(x)+P_g(x)}\right)$$

$$\min_G V = E_{x\sim P_{data}(x)} \ln\left(\frac{P_{data}(x)}{P_{data}(x)+P_g(x)}\right) + E_{x\sim P_g(x)} \ln\left(\frac{P_g(x)}{P_{data}(x)+P_g(x)}\right)$$

Since, we have to prove $P_{data}(x) = P_g(x)$, we use a method that measures the difference between two distributions called **Jenson Shannon Divergence** (JS Divergence).

$$JS(P_1 \| P_2) = \frac{1}{2} E_{x\sim P_1} \ln\left(\frac{P_1}{\frac{P_1+P_2}{2}}\right) + \frac{1}{2} E_{x\sim P_2} \ln\left(\frac{P_2}{\frac{P_1+P_2}{2}}\right)$$

Now we have to modify our value function such that we get it in terms of the JS Divergence Equation, as we have to compare two probability distributions.

Adding and substracting 2ln2 in the value function, we get

$$\min_G V = E_{x\sim P_{data}(x)} \ln\left(\frac{P_{data}(x)}{\frac{P_{data}(x)+P_g(x)}{2}}\right) + E_{x\sim P_g(x)} \ln\left(\frac{P_g(x)}{\frac{P_{data}(x)+P_g(x)}{2}}\right) - 2\ln 2$$

$$\min_G V = 2JS(P_{data} \| P_g) - 2\ln 2$$

The JS Divergence between two distributions cannot be negative and the minimum it can get is 0. It will be zero only when $P_{data}(x) = P_g(x)$. **Hence, it is proved that at the global minimum, $\boldsymbol{P_{data}(x) = P_g(x)}$.**

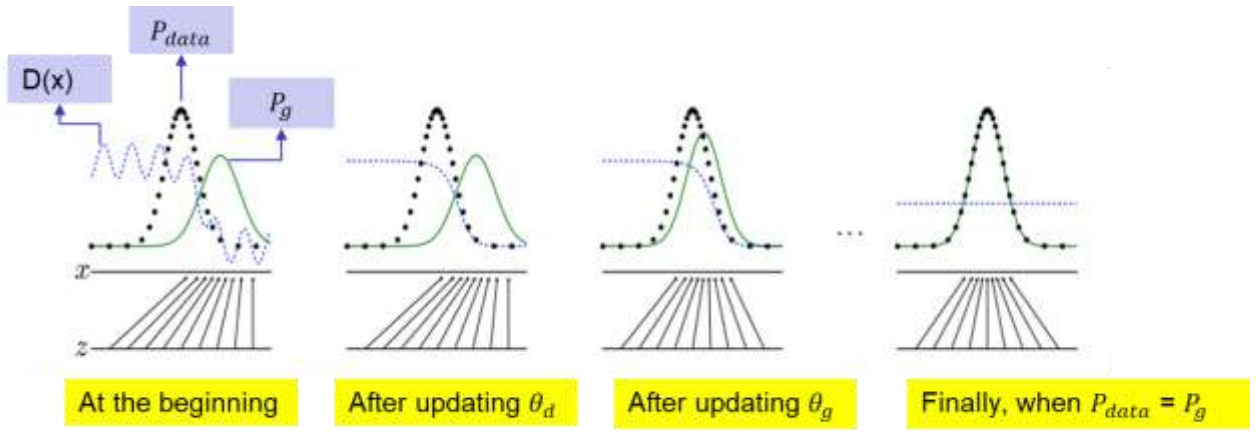

P_{data}
D(x)
P_g
x
z
At the beginning
After updating θ_d
After updating θ_g
Finally, when $P_{data} = P_g$

Fig 4: Illustration of Training

From figure 4, at the beginning, the distribution of generator and original data vary highly such that the discriminator is able to classify. In the next step when backpropagation starts, the discriminator gets updated first and it classifies if it is real or fake. Next step in the training process is to update generator, we can see that the distribution of generator converges to the original data gradually. After certain number of iterations, in the last part of the figure, we can observe that $P_{data}(x) = P_g(x)$. Now, the discriminator will not be able to distinguish between the real image and the fake image and thus produces probability of 0.5.

Applications of GAN:

GAN has various number of use cases in real life scenarios,

- Generate new data samples from available data – It means generating new samples from an available sample that is not similar to a real one.
- Generate realistic pictures of people (Deepfakes) that have never existed.
- GAN’s can generate not only image it can generate text, poems, musics, songs etc.,
- GAN can also generate captions for images and videos by text to image generation using (ObjectGAN and Object Driven GAN).
- GAN can generate anime characters in Game Development and animation production on its own.
- GAN can translate one image to another without disturbing the background. (In this project we have translated hazed images into dehazed images)
- GAN also can be used to increase the resolution of the image or video (SR-GAN and ESR-GAN).

Different Types of GAN:

Deep Convolution GAN (DC GAN):
It is a Deep convolutional GAN, is one of the most used, powerful, and successful types of GAN architecture. It is implemented with the help of ConvNets in place of a Multi-layered perceptron. The ConvNets use a convolutional stride. ConvNets are built without max pooling and layers in this network are not fully connected.

Conditional GAN (CGAN):
Conditional GAN is an important extension to GAN. CGAN used to generate the output samples with given conditions passed as additional inputs. In CGAN, the output labels are based on the class we want. The value function is conditioned on a particular class rather than training for all classes such that the output is solely dependent on that particular given class. For e.g., male or female in the generation of photographs of people, or a digit, in the case of generating images of handwritten digits. The discriminator is also conditioned, meaning that, it is provided both with an input image, generated image and the class the output should belong to.

Super Resolution GAN (SR GAN):
The motive of Super Resolution GAN is to recover finer textures from the image when we upscale it so that its quality cannot be compromised. There are other methods such as Bilinear, Bicubic Interpolation methods for up-sampling that can be used to perform this task but they suffer from image information loss and smoothing and also, they have pre-defined weights. These methods will not learn anything when the model is being trained.

There are few other Types of GANs like InfoGAN, ESR GAN, LSGAN, CycleGAN (which we'll discuss below) etc.

Methodology:

CycleGAN:

The CycleGAN is an extension of the GAN architecture that involves the simultaneous training of two generator models and two discriminator models. One generator takes images from the first domain (Hazed Images) as input and outputs images for the second domain (Dehazed Images), and the other generator takes images from the second domain (Dehazed Images) as input and generates images from the first domain (Hazed Images). Discriminator models are then used to determine how close the generated images are to the original image and update the generator models accordingly. Usually, GAN requires a dataset of paired examples to train an image-to-image translation model. It's difficult to get pairwise datasets in real world datasets. In many cases such a dataset does not exist. To overcome this limitation, a technique is employed where we do not require pairwise images; instead, un-paired images can be used and the general characteristics are extracted from each collection and used in the image translation process. Unpaired image-to-image translation is successfully done by cycle GAN. CycleGAN is well known for its application of image-to-image translation in the absence of paired data. CycleGAN can dela with unpaired data which makes the model more flexible. CycleGAN is trained in an unsupervised manner as it uses un-paired images.

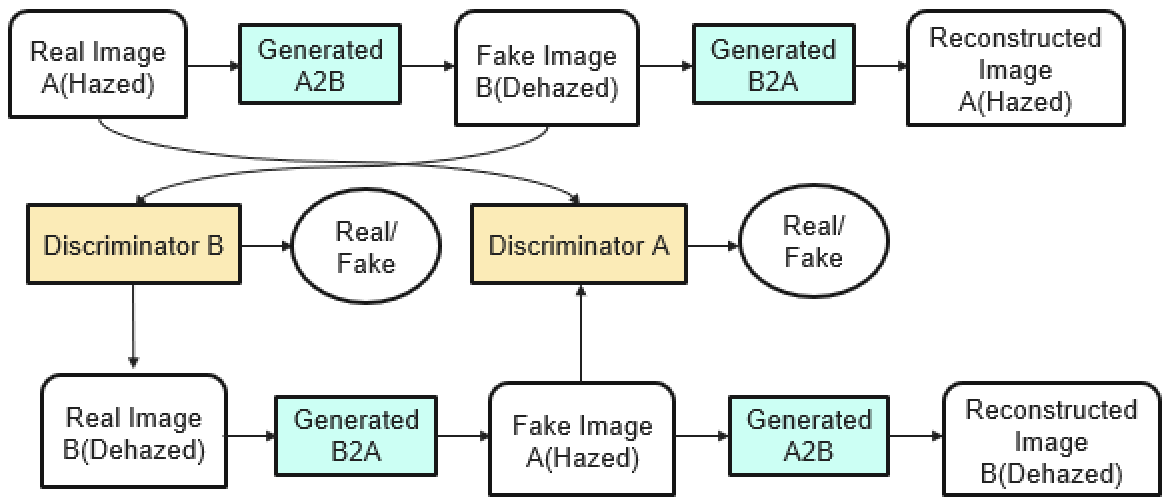


Fig 5: Cycle GAN architecture

In our project we have two different collections of data (Hazed image and Dehazed image). For this we develop an architecture of two GANs, and each GAN has a discriminator and a generator model, meaning there are four models (Generator A2B, Generator B2A, Discriminator B and Discriminator A) in total in the architecture. To summarize

- GAN 1: Translates dehazed Images to hazed Images.
- GAN 2: Translates hazed Images to dehazed Images.

We can summarize the generator and discriminator models from GAN 1 as follows:

Generator Model 1:
– Input: dehazed Images.
– Output: Generated hazed Images.
Discriminator Model 1:
– Input: hazed Images and output from Generator Model 1.
– Output: Likelihood of image is a hazed Image.

Similarly, we can summarize the generator and discriminator models from GAN 2 as follows:

Generator Model 2:
– Input: hazed Images.
– Output: Generated hazed Images.
Discriminator Model 2:
– Input: dehazed Images and output from Generator Model 2.
– Output: Likelihood of image is a dehazed Image.

The key point of CycleGAN is the loss that it calculates between the re-constructed image and the original image. The loss is called **cycle consistency loss**. This is designed to encourage the synthesized images in the target domain that are to be translations of the input image. Cycle consistency loss compares an input image to the generated image and calculates the difference between them using the L1 norm or summed absolute difference in pixel values.

The cycle consistency loss calculates the difference between the image input to GAN 1 and the image output by GAN 2 and the generator models are updated accordingly to reduce the difference in the images. This is a **forward-cycle consistency loss**. The same process is related in reverse for a **backward cycle-consistency loss** from generator 2 to generator 1 and comparing the real hazed image to the generated hazed image.

Forward Cycle-Consistency Loss: $x \rightarrow G(x) \rightarrow F(G(x)) \approx x$. Image A to reconstructed Image A.
Backward Cycle-Consistency Loss: $y \rightarrow F(y) \rightarrow G(F(y)) \approx y$. Image B to reconstructed Image B.

The discriminator and generator models are trained in an adversarial zero-sum process, like normal GAN models. The generators learn to better fool the discriminators and the discriminators learn to better detect fake images. Together, the models find an equilibrium during the training process.

Conventions:

•G – Generates Dehazed image

•F – Generates Hazed image

•D_Y - Discriminator B

•D_X - Discriminator A

•X – Real Image A (Hazed)

•Y– Real Image B (Dehazed)

•$\hat{X}$– Fake Image A (Hazed)

•$\hat{Y}$– Fake Image B (Dehazed)

Loss Functions:

Adversarial Loss and Forward Cycle-Consistency Loss for Hazed(X) to Dehazed(Y):

$$L_{GAN_1}(G, D_Y, X, Y) = E_{y \sim Y}\, log[\, D_Y(y)] + E_{x \sim X}[log(\, 1 - D_Y(G(x)))]$$
$$L_{cyc_1}(G, F) = E_{x \sim X}[||F(G(x)) - x||_1]$$

Adversarial Loss and Backward Cycle-Consistency Loss for Dehazed(Y) to Hazed(X):

$$L_{GAN_2}(F, D_X, Y, X) = E_{x \sim X}\, log[\, D_X(x)] + E_{y \sim Y}[log(\,1 - D_X(F(y)))]$$
$$L_{cyc_2}(F, G) = E_{y \sim Y}[||G(F(y)) - y||_1]$$

For cycle-consistency loss, L1 norm is taken between the reconstructed image and the original image. By combining the loss functions above, we can write the objective function as:

$$L(G, F, D_X, D_Y) = \underbrace{L_{GAN_1}(G, D_Y, X, Y) + L_{cyc_1}(G, F)}_{hazed-dehazed} + \underbrace{L_{GAN_2}(F, D_X, Y, X) + L_{cyc_2}(F, G)}_{dehazed-hazed}$$

Optimization:

$$G^*, F^* = arg \min_{G,F} \max_{D_X, D_Y} L(G, F, D_X, D_Y)$$

Implementation of Cycle GAN for single image dehazing:

At first, we load the input images from both haze and dehaze folders, then the data is preprocessed in the way all the images are of the same shape i.e. (255,256,3) as shown in Figure 6

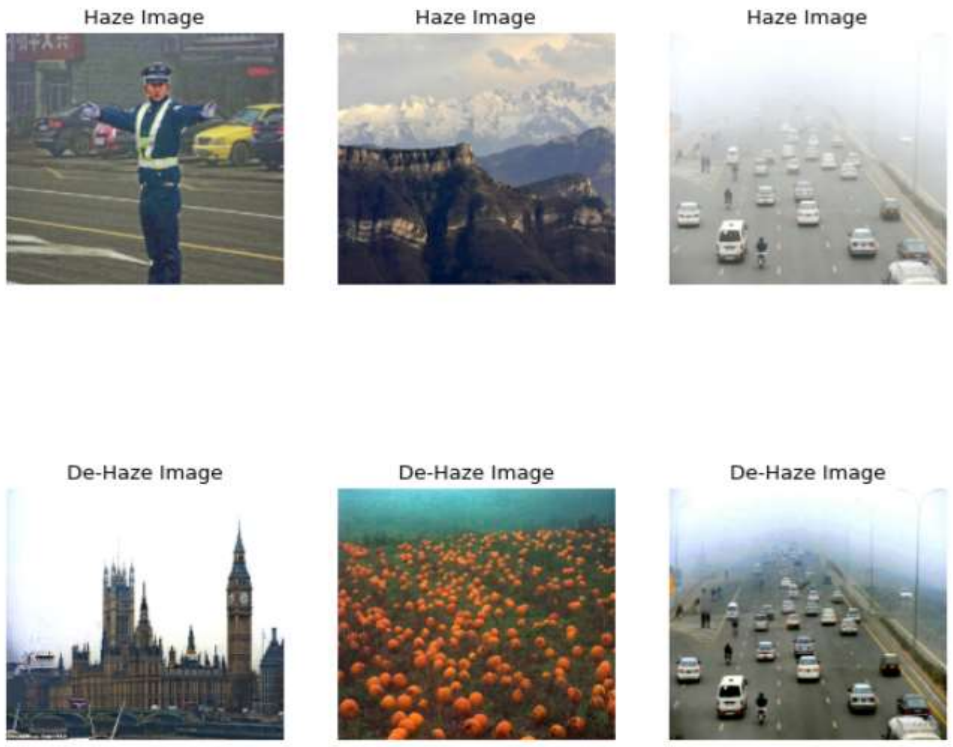


Fig 6: Sample preprocessed input hazed and dehazed images

The discriminator of our model takes a 256×256 sized image as input and outputs a patch of predictions (70 x 70), thus this model is called PatchGAN discriminator. The model is optimized using least squares loss (L2) implemented as mean squared error,and a weighting is used so that the model has half (i.e. 0.5) the usual effect. The authors of the CycleGAN paper recommend this weighting of model updates to slow down changes to the discriminator, relative to the generator model during training.

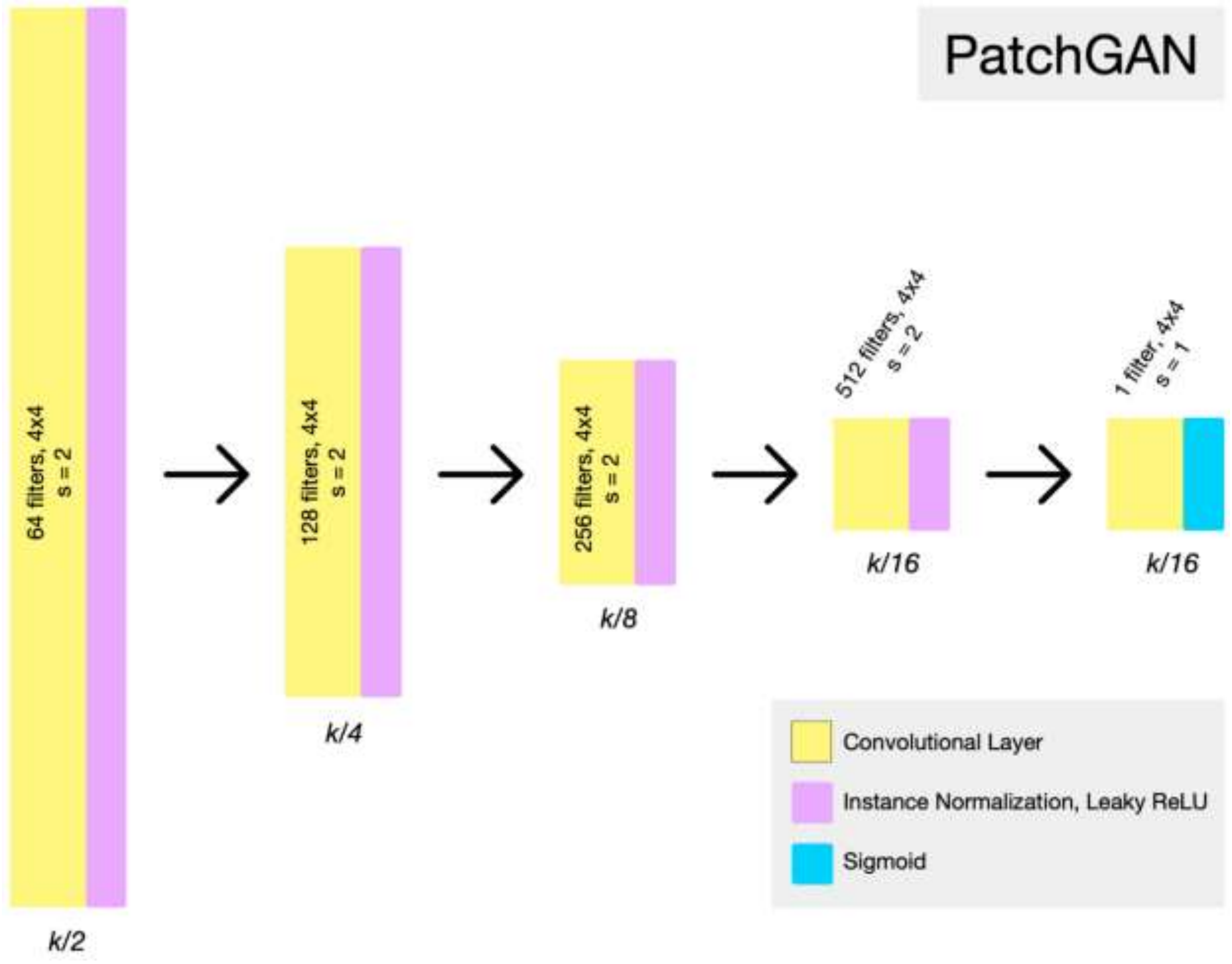


Fig 7: Discriminator architecture

This model does not use batch normalization; instead, instance normalization is used. It is a type of normalization that standardizes the values on each feature map. Instance normalization is used in place of batch normalization in order to remove instance-specific contrast information that simplifies generation.

The Generator of our model consists of an encoder consisting of convolution layers, a transformer with residual blocks and a decoder consisting of transpose convolution layers. The resnet block used in the generator consists of two Convolution-InstanceNorm blocks with 3×3 filters and 1×1 stride and without a ReLU activation after the second block.These are blocks comprised of two 3×3 CNN layers where the input to the block is added to the output of the block, channel-wise.

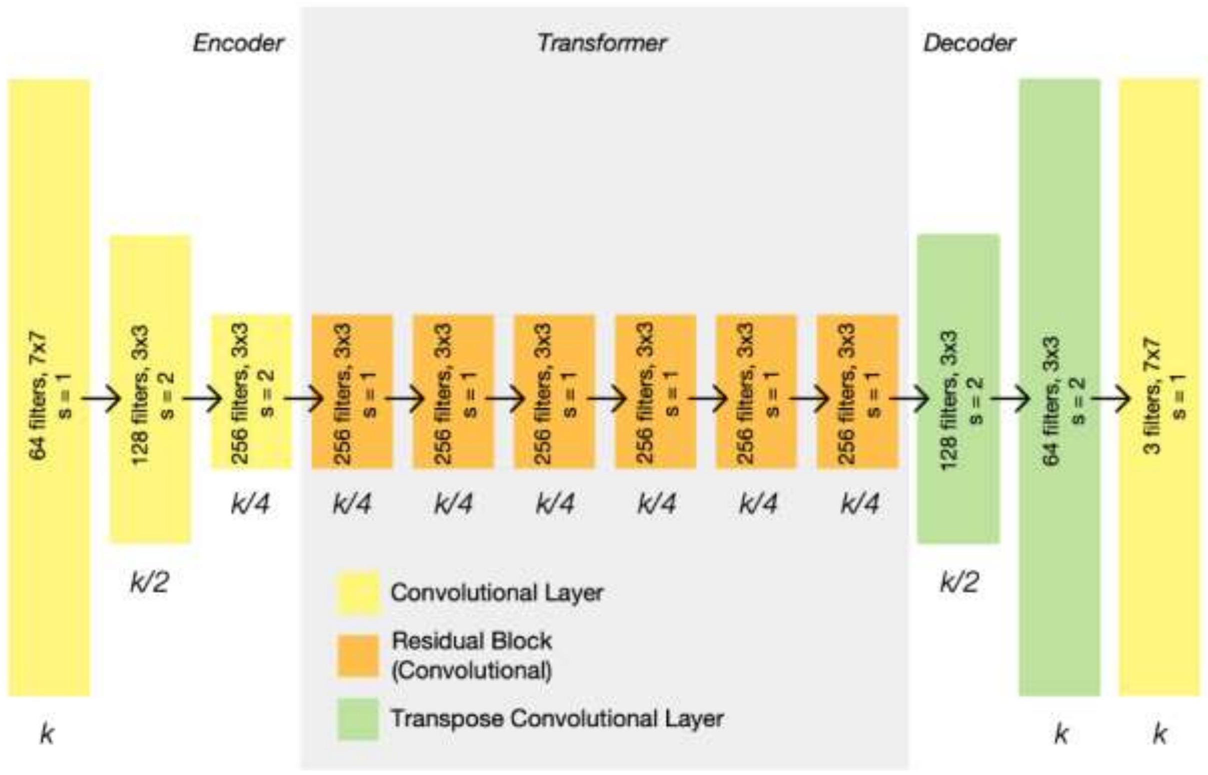


Fig 8: Generator architecture

The generator models are not updated directly. Instead, the generator models are updated via composite models. An update to each generator model involves changes to the model weights based on four concerns:

- Adversarial loss (L2 or mean squared error).
- Identity loss (L1 or mean absolute error).
- Forward cycle loss (L1 or mean absolute error).
- Backward cycle loss (L1 or mean absolute error).

The adversarial loss is the standard approach for updating the generator via the discriminator, although in this case, the least squares loss function is used instead of the negative log likelihood (e.g., binary cross-entropy).
Composite model is used to train the generate by making all other models non-trainable with the help of the above-described losses. The generator is updated as a weighted average of the four loss values. The adversarial loss is weighted normally, whereas the forward and backward cycle loss is weighted using a parameter called lambda and is set to 10, e.g., 10 times more important than adversarial loss. The identity loss is also weighted as a fraction of the lambda parameter and is set to 0.5×10 or 5 in the official Torch implementation.

A batch of fake images are generated from the generator. Similarly, a batch of real images are also extracted from the dataset and used to train the discriminator model. The loss is reported each training iteration, including the Discriminator-A loss on real and fake examples (dA), Discriminator-B loss on real and fake examples (dB), and Generator-AtoB and Generator-BtoA loss, each of which is a weighted average of adversarial, identity, forward, and backward cycle loss (g).

Evaluation metrics:

A haze removal algorithm's performance can be evaluated on several factors, among them, two of the most frequently used factors are the PSNR and SSIM. Peak Signal to Noise Ratio (PSNR) measures the ability of the algorithm to remove noise from a noisy image. PSNR is a full reference evaluation method, which requires reference to clean images and images to be evaluated.

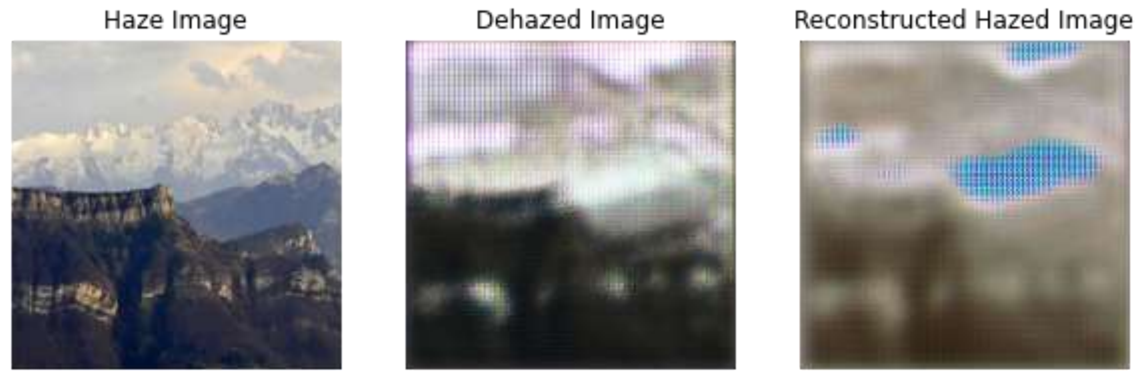
$$PSNR = 10\log_{10}\frac{255}{\sqrt{|X_{in} - X_{out}|^2}}$$

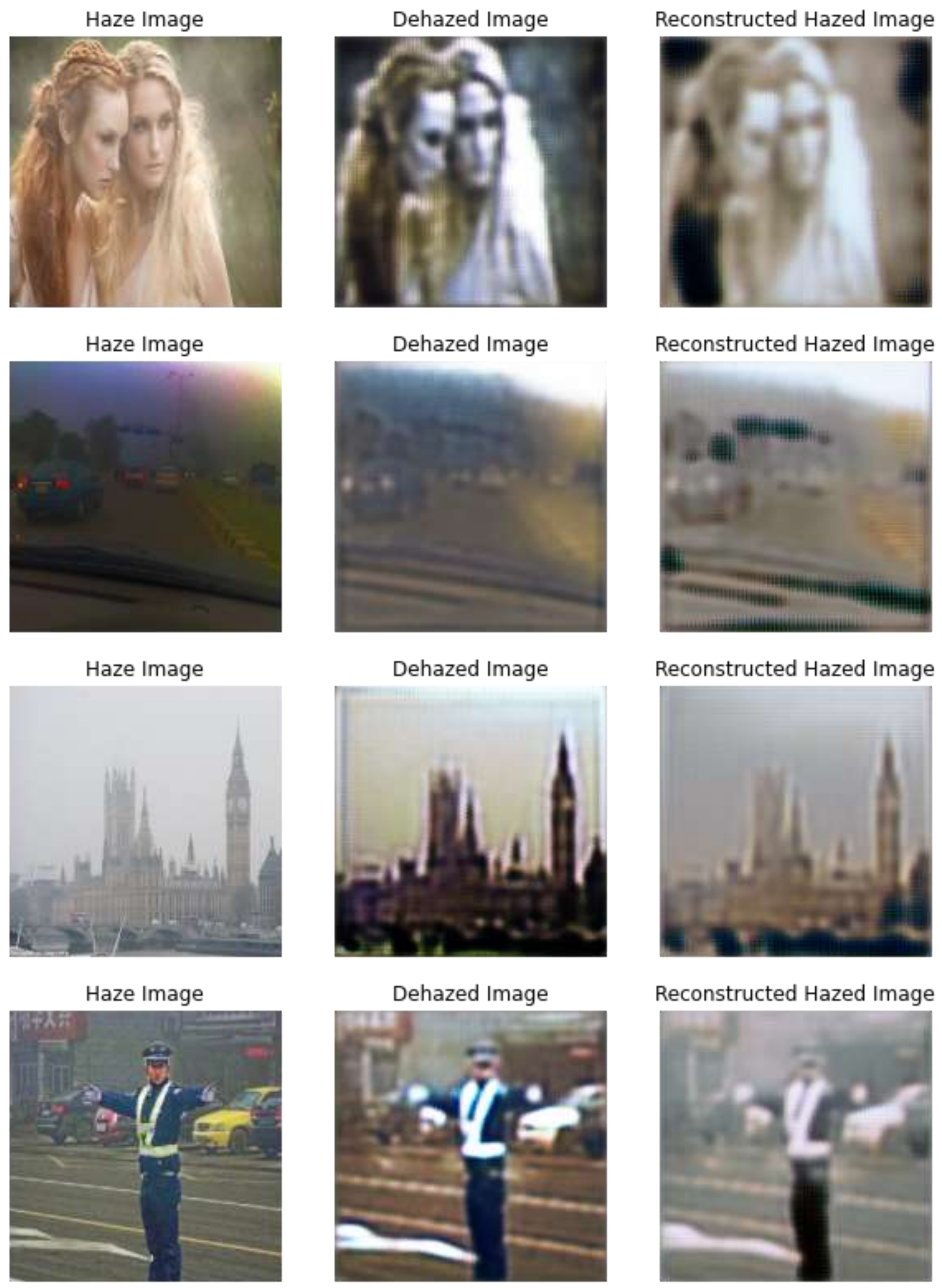
where X_{in} is a hazy image output Xout hazy model image, the larger the PSNR value, indicating the lower degree of distortion of the image, the higher the image quality, better dehazing. Two identical images will have a PSNR value of infinity.
SSIM is also a full-reference evaluation method, which measures the similarity between the clean image and the image to be evaluated in terms of brightness, contrast, and structure. Two identical images will have a SSIM of 1. Two identical images will have a SSIM of 1.

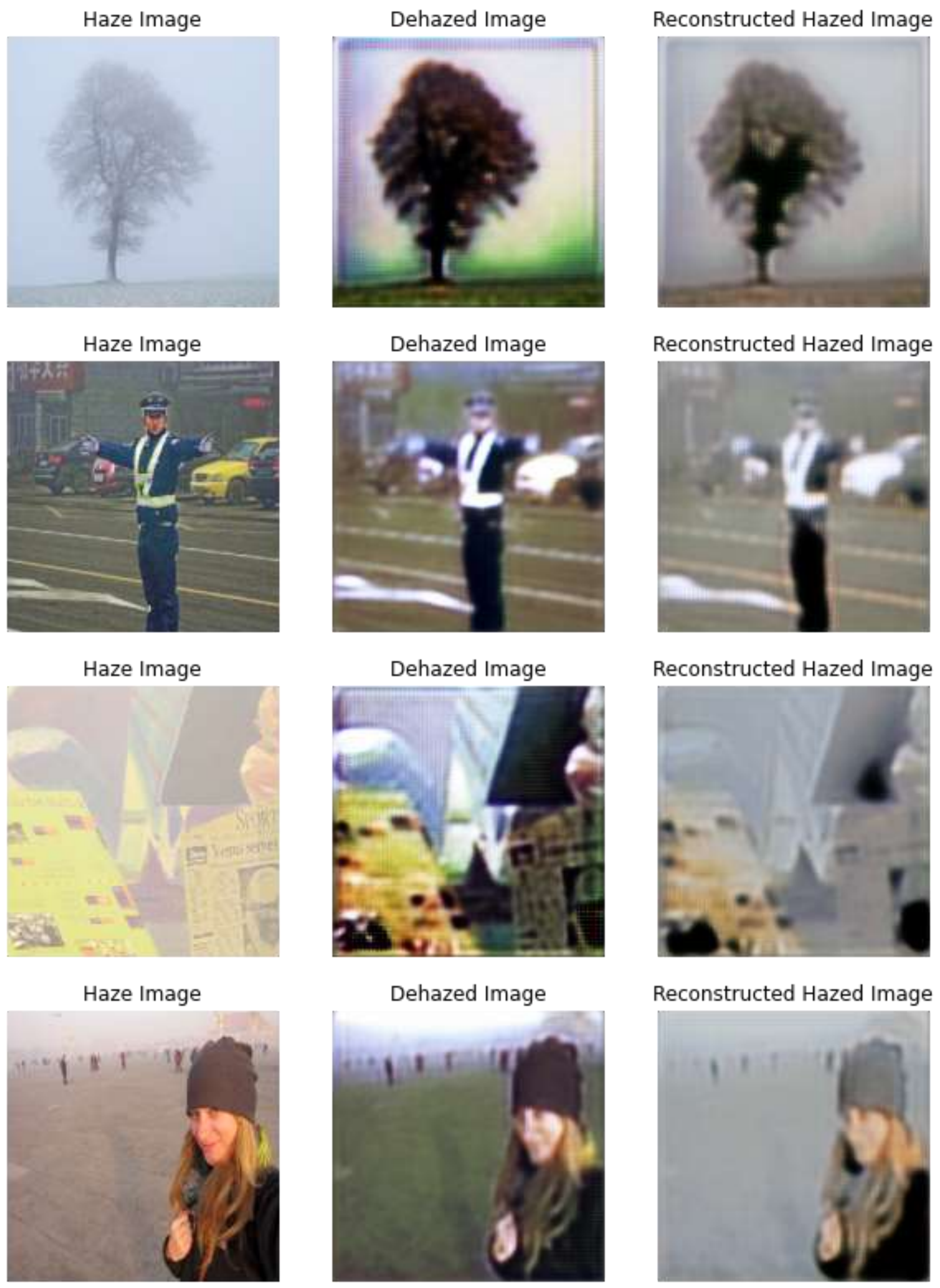
$$\mathrm{SSIM} = \frac{\left(2\mu_{X_{\mathrm{in}}}\mu_{X_{\mathrm{out}}} + \theta_1\right)\left(2\sigma_{X_{\mathrm{in}}X_{\mathrm{out}}} + \theta_2\right)}{\left(\mu_{X_{\mathrm{in}}}^2 + \mu_{X_{\mathrm{out}}}^2 + \theta_1\right)\left(\sigma_{X_{\mathrm{in}}}^2 + \sigma_{X_{\mathrm{out}}}^2 + \theta_2\right)}$$

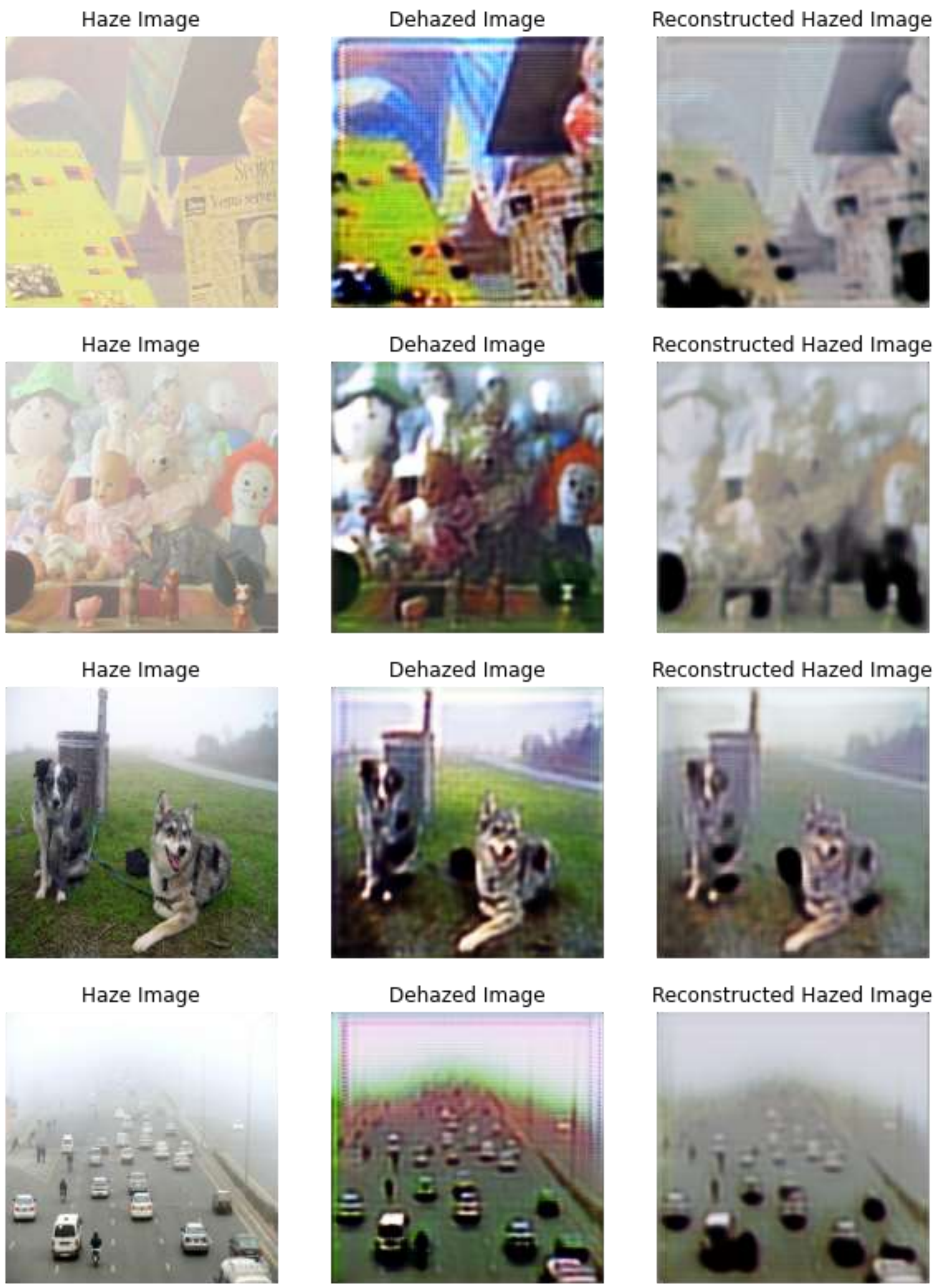
Where $\mu_{X_{\mathrm{in}}}$ is the average of X_{in}, $\mu_{X_{\mathrm{out}}}$ is the average ofX_{out}, $\sigma_{X_{\mathrm{in}}X_{\mathrm{out}}}$ is the covariance of X_{in} and X_{out}, $\sigma_{X_{\mathrm{in}}}^2$ is the variance of X_{in}, and $\sigma_{X_{\mathrm{out}}}^2$ is the variance of X_{out}.

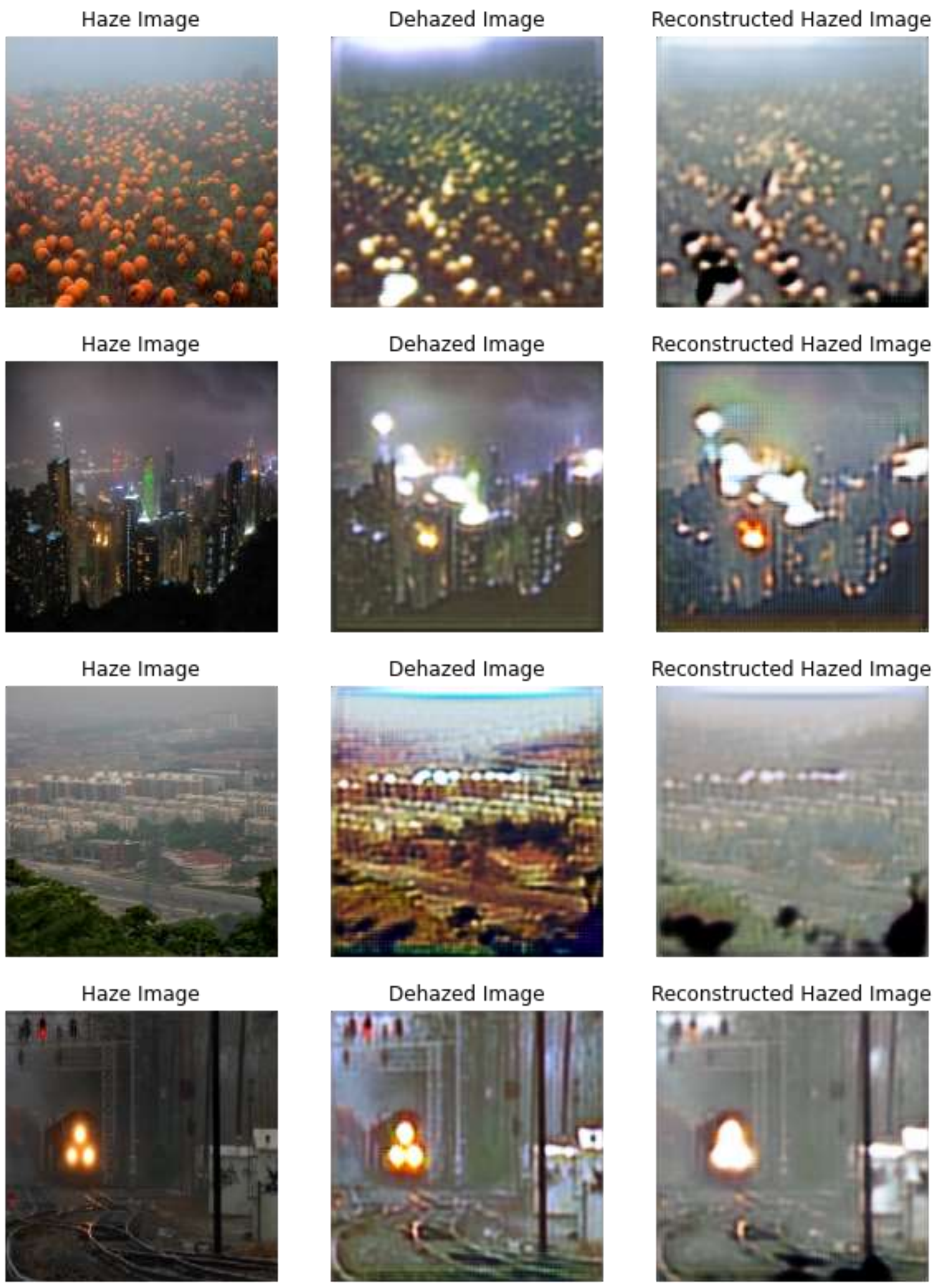
Results and discussion:

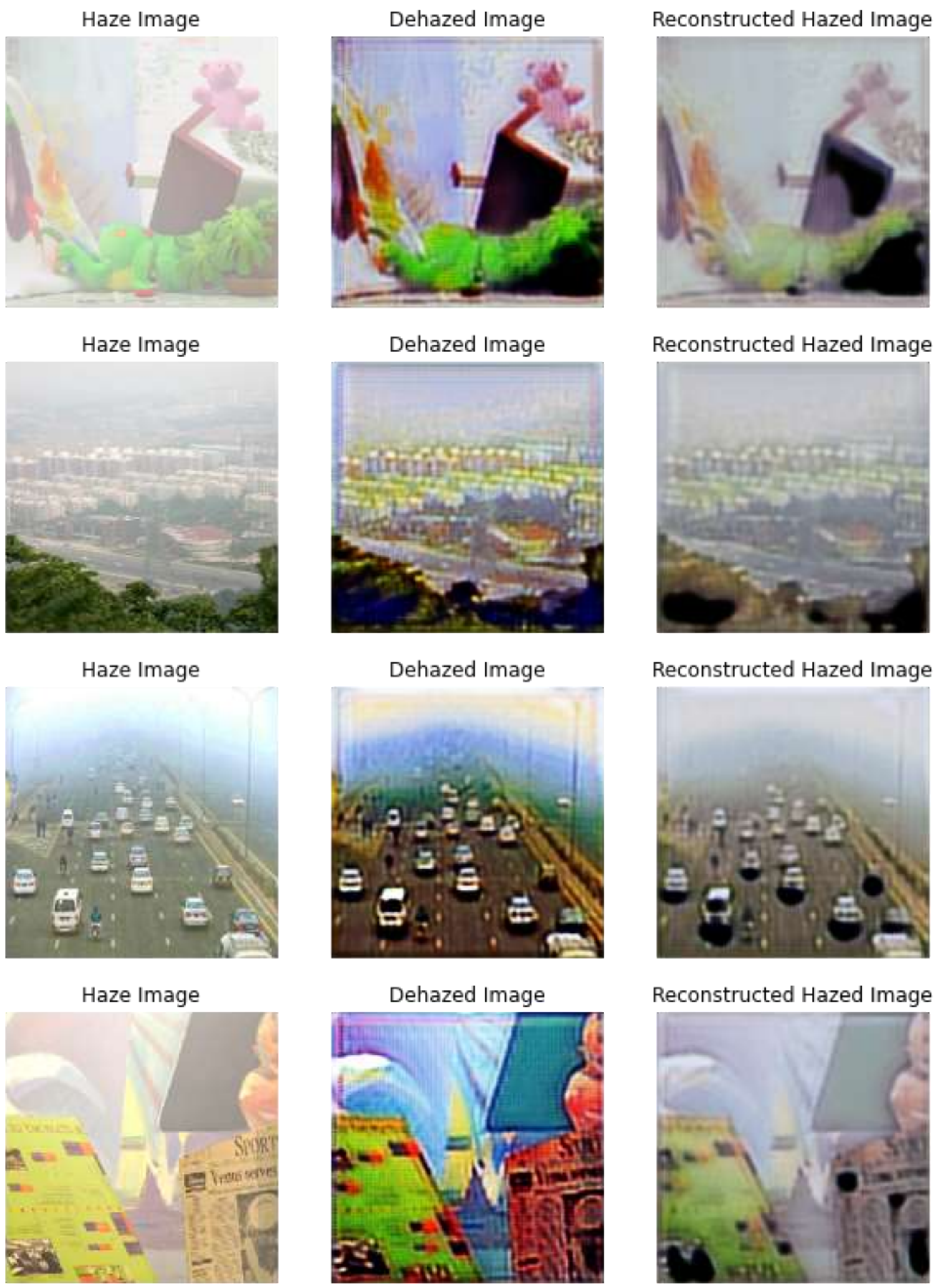


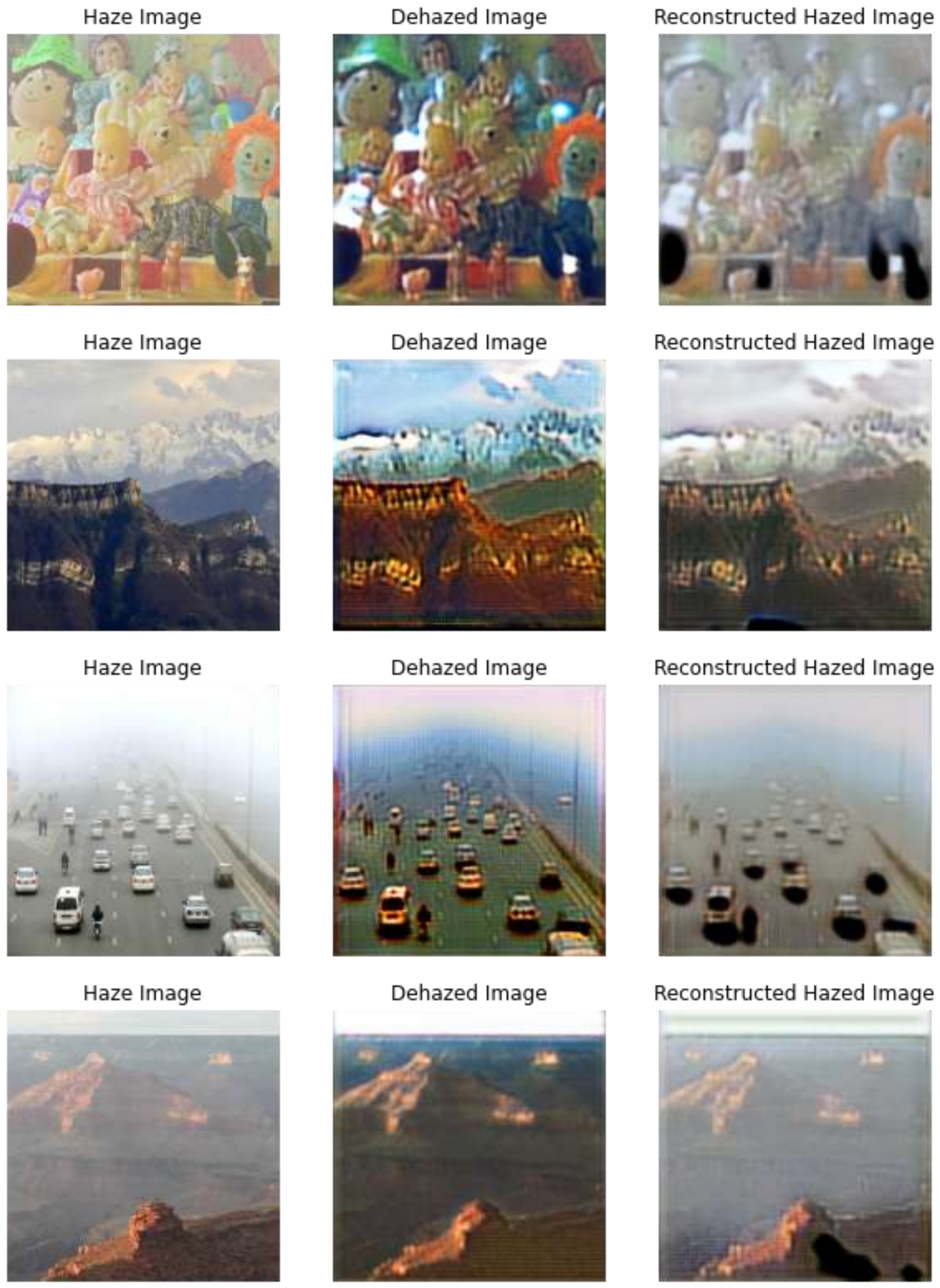

Haze Image
Dehazed Image
Reconstructed Hazed Image
Haze Image
Dehazed Image
Reconstructed Hazed Image
Haze Image
Dehazed Image
Reconstructed Hazed Image
Haze Image
Dehazed Image
Reconstructed Hazed Image

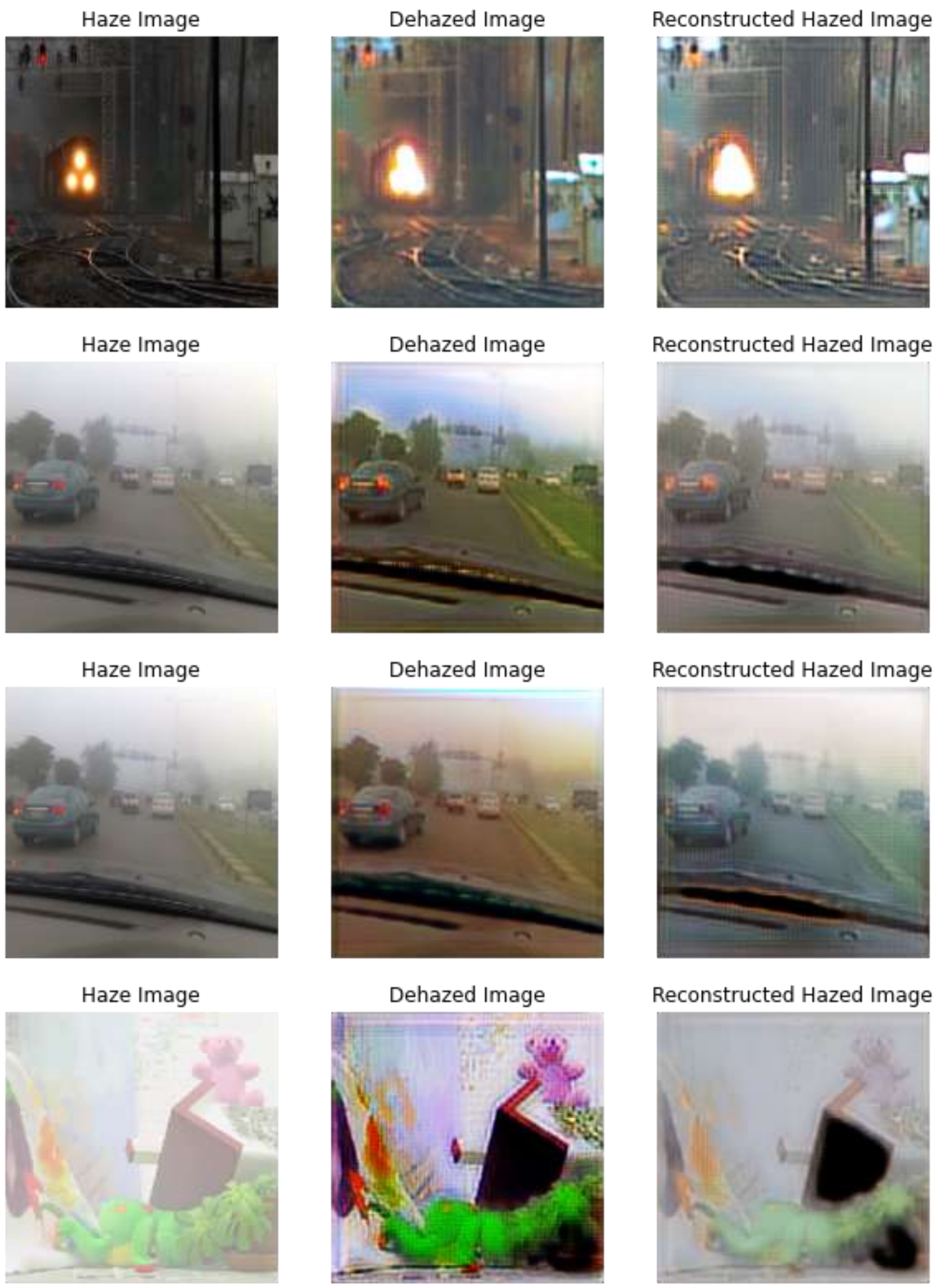

Haze Image
Dehazed Image
Reconstructed Hazed Image
Haze Image
Dehazed Image
Reconstructed Hazed Image
Haze Image
Dehazed Image
Reconstructed Hazed Image
Haze Image
Dehazed Image
Reconstructed Hazed Image

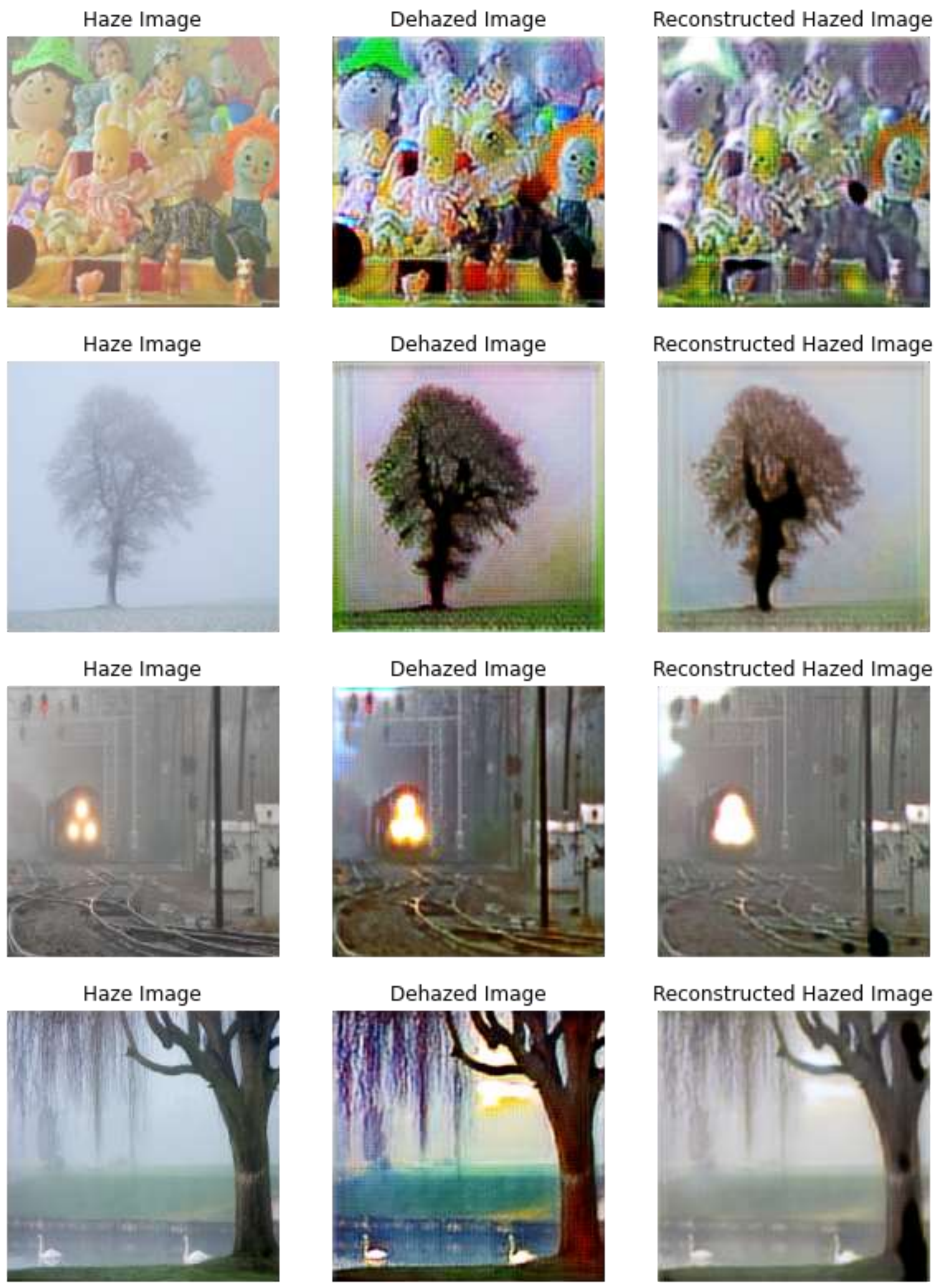

Haze Image
Dehazed Image
Reconstructed Hazed Image
Haze Image
Dehazed Image
Reconstructed Hazed Image
Haze Image
Dehazed Image
Reconstructed Hazed Image
Haze Image
Dehazed Image
Reconstructed Hazed Image

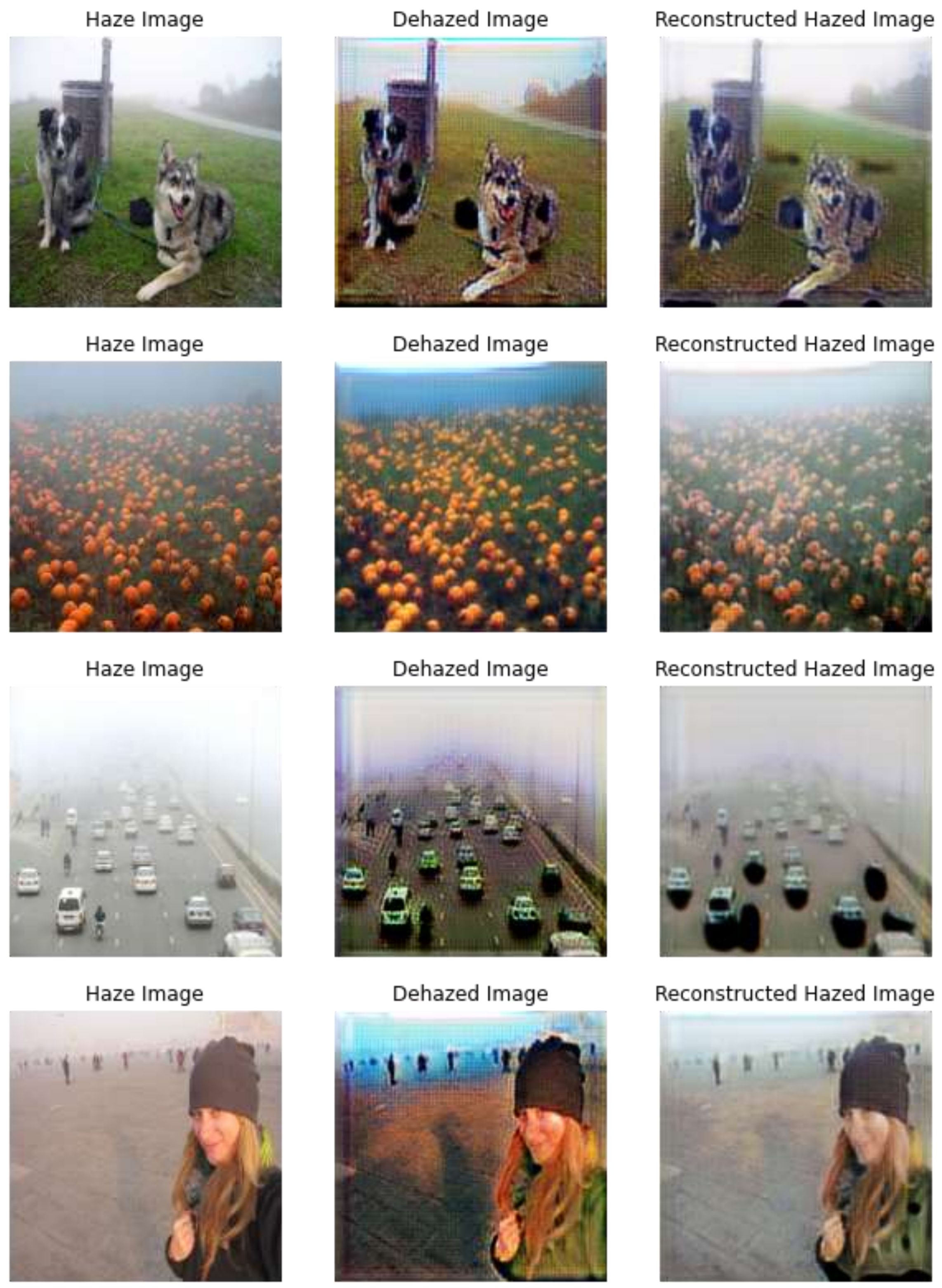

Haze Image
Dehazed Image
Reconstructed Hazed Image
Haze Image
Dehazed Image
Reconstructed Hazed Image
Haze Image
Dehazed Image
Reconstructed Hazed Image
Haze Image
Dehazed Image
Reconstructed Hazed Image


Haze Image
Dehazed Image
Reconstructed Hazed Image
Haze Image
Dehazed Image
Reconstructed Hazed Image
Haze Image
Dehazed Image
Reconstructed Hazed Image
Haze Image
Dehazed Image
Reconstructed Hazed Image


Haze Image
Dehazed Image
Reconstructed Hazed Image
Haze Image
Dehazed Image
Reconstructed Hazed Image
Haze Image
Dehazed Image
Reconstructed Hazed Image
Haze Image
Dehazed Image
Reconstructed Hazed Image


Haze Image
Dehazed Image
Reconstructed Hazed Image
Haze Image
Dehazed Image
Reconstructed Hazed Image
Haze Image
Dehazed Image
Reconstructed Hazed Image
Haze Image
Dehazed Image
Reconstructed Hazed Image


Haze Image
Dehazed Image
Reconstructed Hazed Image
Haze Image
Dehazed Image
Reconstructed Hazed Image
Haze Image
Dehazed Image
Reconstructed Hazed Image
Haze Image
Dehazed Image
Reconstructed Hazed Image


Haze Image
Dehazed Image
Reconstructed Hazed Image
Haze Image
Dehazed Image
Reconstructed Hazed Image
Haze Image
Dehazed Image
Reconstructed Hazed Image
Haze Image
Dehazed Image
Reconstructed Hazed Image

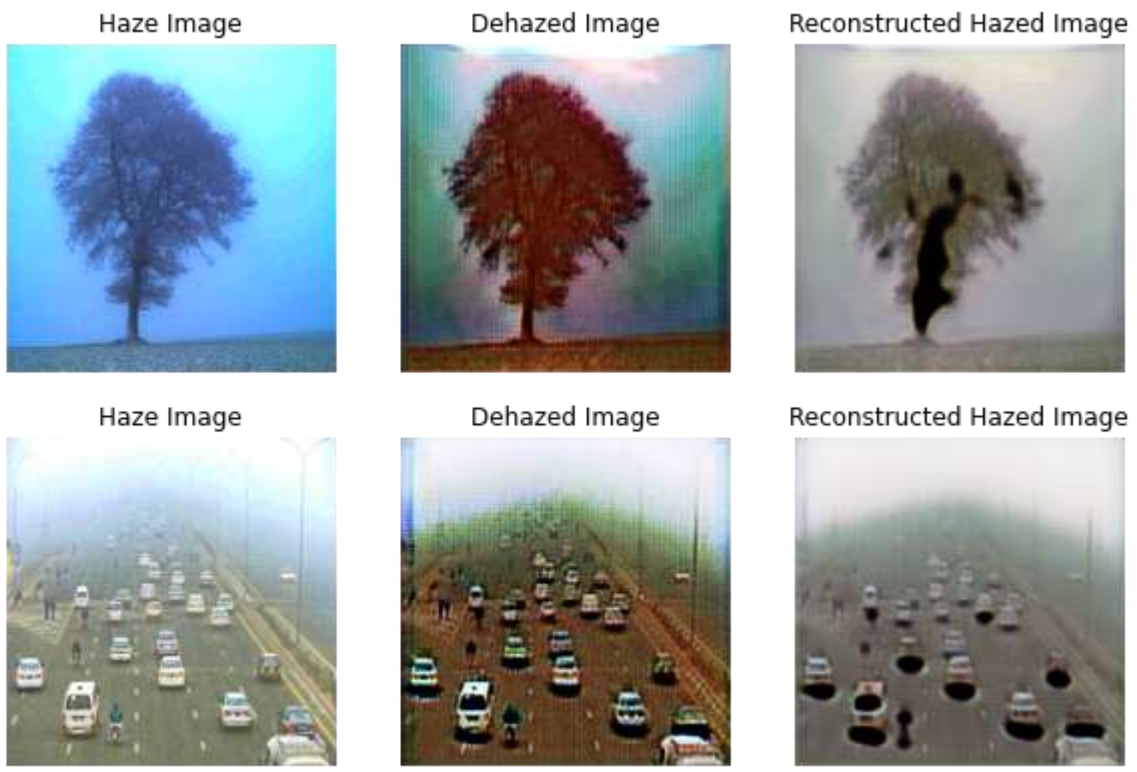


Fig 9: Dehazed and reconstructed output of a training sample for each epoch

The trained GAN was tested on the test dataset with 10 images. The model is trained upto 40 epochs. From figure 9 it's observed that from the 29th epoch, we can see, the model performs pretty well. The haze is almost de-hazed while keeping the generic features the same. The model is evaluated on PSNR (Peak Signal-to-Noise Ratio) and SSIM (Structural Similarity Index Measure). It can be observed from the PSNR values of the test images that the noise, which is the haze, is reduced. From the SSIM for the test images, we can infer that the dehazed images retained the spatial properties of the hazed image.

PSNR and SSIM values for test images:

	PSNR	SSIM
0	48.921653	0.880655
1	49.252494	0.888642
2	49.371817	0.890013
3	49.802184	0.901209
4	50.192865	0.912729

5	49.371817	0.890013
6	49.371817	0.890013
7	48.921653	0.880655
8	49.371817	0.890013
9	49.374240	0.892475
Average	**49.395235**	**0.891641**

Table 1: Results of the quantitative analysis conducted on our test set

Above table shows the evaluation metrics (PSNR and SSIM values) of the test images. The average Peak Signal-to-Noise Ratio is 49.39523569512697 and Structural Similarity Index Measure is 0.8916414893280891.

Conclusion

This project presented a Cycle generative adversarial network that directly removes haze given a single image without any mapping. We used Patch Discriminator, Cycle-Consistency Loss, L2 Loss instead of the classic Adversarial Loss (log likelihood) which reduced the presence of artifacts significantly. From these observations above, we can say our model works pretty well for dehazing the image without any loss in its spatial properties while removing the haze (noise), with just 107 training samples. This can be further improved by increasing the resolution of the image by passing it to SRGAN. This model can also be tried on Remote Sensing Data along with super resolution GAN to remove the noise (haze) from the data.

Reference

[1] Salehi, Pegah & Chalechale, Abdolah & Taghizadeh, Maryam. (2020). Generative Adversarial Networks (GANs): An Overview of Theoretical Model, Evaluation Metrics, and Recent Developments.

[2] B. Cai, X. Xu, K. Jia, C. Qing and D. Tao, "DehazeNet: An End-to-End System for Single Image Haze Removal," in IEEE Transactions on Image Processing, vol. 25, no. 11, pp. 5187-5198, Nov. 2016, doi: 10.1109/TIP.2016.2598681.

[3] N., Bharath & Narasimhan, Venkateswaran. (2018). Single Image Haze Removal using a Generative Adversarial Network.

[4] H. T. Suseelan, V. Sowmya and K. P. Soman, "Image dehazing using variational mode decomposition," 2017 International Conference on Wireless Communications, Signal Processing and Networking (WiSPNET), 2017, pp. 200-205, doi: 10.1109/WiSPNET.2017.8299748.

[5] Silberman, Nathan, Derek Hoiem, Pushmeet Kohli, and Rob Fergus. "Indoor segmentation and support inference from rgbd images." In European Conference on Computer Vision, pp. 746-760. Springer, Berlin, Heidelberg, 2012

[6] Deep Learning Generative Models for Image Synthesis and Image Translation by Jason Brownlee