CS F425 Deep Learning Term Project - Task 2

Bhuman Pandita 2021A2PS2605P

M N Vignesh Kumar 2023H1120192P

Ojasva Goyal 2021A2PS2378P

April 26, 2024

Abstract

The project explores the application of CycleGAN for image dehazing, comparing its performance with and without data augmentation. By training the model on a provided dataset and introducing a training hack involving 90-degree anticlockwise image rotations, the study assesses the impact on image quality and generalization. Evaluation metrics such as SSIM and PSNR are used to quantify the results. CycleGAN proves effective in improving image clarity, with data augmentation enhancing its performance further. The experimentation provides insights into practical techniques for image dehazing and opens avenues for future research in enhancing visibility in visual data. We have trained the model from scratch and we tried to restrain from using any pretrained model.

1 Literature Review

Our initial literature survey started with the influential Skygan paper by Aditya Mehta, Harsh Sinha, Murari Mandal, Pratik Narang: "Domain-Aware Unsupervised Hyperspectral Reconstruction for Aerial Image Dehazing" [1]. SkyGAN tackles aerial image dehazing with a two-pronged approach: 1) Hazy-to-Hyperspectral (H2H) Module: This module creates a helpful data source (hyperspectral catalyst) from regular RGB images, even without paired hazy-hyperspectral training data. 2) Image-to-Image Translation (I2I) Module: This module utilizes the H2H output along with image data to perform dehazing by exploiting the full color spectrum for better results.However, due to limitations in hardware and technical resources, we determined that utilizing hyperspectral imagery, as employed in Skygan, would not be feasible for our project. Therefore, we sought alternative published research that leveraged the more readily available RGB (three-channel) format for the image dehazing task.

2 Model Description

We selected CycleGAN for this task. CycleGAN is well-suited for the task of image dehazing. We selected CycleGAN because of their ability to learn mappings between unpaired image domains while enforcing cycle-consistency and adversarial training makes it a powerful tool for image dehazing, offering promising results in generating clear images from hazy inputs. Even with paired data available, CycleGAN can offer benefits such as data augmentation, robustness to missing data, flexibility in fine-tuning and adaptation, reduced annotation effort, and support for domain adaptation and transfer learning, making it a valuable tool for image dehazing tasks.

This code implements a CycleGAN model for dehazing images. The components are documented below:

1. **CustomImageFolder and CustomDataset Classes:** These classes are used to load the dataset. CustomImageFolder extends PyTorch's ImageFolder class to also return the index of each image alongside the image itself. CustomDataset is a custom dataset class that loads hazy and ground truth images from the specified directories.
2. **Generator and Discriminator Classes:** The Generator class defines the architecture of the generator network, which aims to generate dehazed images from hazy ones. It consists of convolutional layers followed by residual blocks and transposed convolutional layers. The Discriminator class defines the discriminator network architecture, which discriminates between real and fake images. It uses a series of convolutional layers.
3. **Data Preprocessing:** Images are resized to (256, 256), converted to tensors, and normalized to have a mean and standard deviation of (0.5, 0.5, 0.5) for each channel.
4. **Training Loop:** The training loop consists of alternating updates between the discriminators and the generators. Discriminator loss is computed using a Mean Squared Error (MSE) loss between real and fake images. Generator loss includes adversarial loss (MSE loss between generated images and ones), and cycle consistency loss (L1 loss between reconstructed images and original images). These losses are weighted and summed to form the total generator loss.
5. **Optimizers and Learning Rate Scheduling:** Adam optimizers are used for both generators and discriminators with a learning rate of 0.0002 and betas (0.5, 0.999). No explicit learning rate scheduling is implemented in this code.
6. **Model Saving:** Models are saved after each epoch along with optimizer states and training statistics.
7. **Validation Loop:** After each epoch, the model is evaluated on a validation set. Validation loss is computed using cycle consistency loss between dehazed images and ground truth images.

2.1 Generator Model

The generator network is a convolutional neural network (CNN). It uses convolutional layers, batch normalization, and activation functions like ReLU to transform hazy images into dehazed ones. The inclusion of residual blocks further enhances its ability to capture and reconstruct details effectively. The parmeters of the model are :

- Total params: 11,383,424
- Trainable params: 11,383,424
- Non-trainable params: 0
- Input size (MB): 0.75
- Forward/backward pass size (MB): 819.00
- Params size (MB): 43.42

- Estimated Total Size (MB): 863.17

The generator model implements a CycleGAN architecture tailored for dehazing images. It comprises convolutional layers, residual blocks, and transposed convolutional layers. The generator takes hazy images as input and aims to generate dehazed images. Key hyperparameters include the input and output channels, both set to 3 for RGB images. The number of residual blocks is set to 9, contributing to the model's ability to capture and reconstruct complex features from hazy images. Convolutional kernel sizes vary across layers, with the first and last layers employing larger kernels (7x7) to capture global context, while intermediate layers utilize smaller kernels (3x3) for feature extraction. The model architecture is depicted below :

Layer (type)	**Output Shape**	**Param #**
Conv2d-1	[-1, 64, 256, 256]	9,408
BatchNorm2d-2	[-1, 64, 256, 256]	128
ReLU-3	[-1, 64, 256, 256]	0
Conv2d-4	[-1, 128, 128, 128]	73,728
BatchNorm2d-5	[-1, 128, 128, 128]	256
ReLU-6	[-1, 128, 128, 128]	0
Conv2d-7	[-1, 256, 64, 64]	294,912
BatchNorm2d-8	[-1, 256, 64, 64]	512
ReLU-9	[-1, 256, 64, 64]	0
Conv2d-10	[-1, 256, 64, 64]	589,824
BatchNorm2d-11	[-1, 256, 64, 64]	512
ReLU-12	[-1, 256, 64, 64]	0
Conv2d-13	[-1, 256, 64, 64]	589,824
BatchNorm2d-14	[-1, 256, 64, 64]	512
ReLU-15	[-1, 256, 64, 64]	0
ResidualBlock-16	[-1, 256, 64, 64]	0
...	...	...
ConvTranspose2d-76	[-1, 64, 256, 256]	73,728
BatchNorm2d-77	[-1, 64, 256, 256]	128
ReLU-78	[-1, 64, 256, 256]	0
Conv2d-79	[-1, 3, 256, 256]	9,408
Tanh-80	[-1, 3, 256, 256]	0

2.2 Justification for Selections

In the code, a residual block is used within the generator network. The purpose of a residual block is to facilitate the training of deeper networks by mitigating the vanishing gradient problem. In deep neural networks, especially those with many layers, the gradients can vanish during backpropagation, making it difficult to train effectively. Residual connections help mitigate this issue by allowing gradients to flow more easily through the network.

ReLU is used after each convolutional layer and before each batch normalization layer. It's applied to the output of each convolutional layer to introduce non-linearity into the network and help the model learn complex patterns in the data.

2.3 Discriminator Model

The discriminator network is also a CNN. Its purpose is to discriminate between real and fake images. It takes hazy or dehazed images as input and outputs a single value indicating the likelihood of the input being real. The discriminatoralso consists of convolutional layers, batch normalization, and ReLU activation functions to learn discriminative features from the images. The parmeters of the model are :

- Total params: 2,765,696
- Trainable params: 2,765,696
- Non-trainable params: 0
- Input size (MB): 0.75
- Forward/backward pass size (MB): 53.27
- Params size (MB): 10.55
- Estimated Total Size (MB): 64.57

In contrast, the discriminator model plays a crucial role in distinguishing between real and fake images generated by the generator. It consists of multiple convolutional layers with increasing numbers of channels. The discriminator takes either dehazed or hazy images as input and outputs a single scalar value representing the probability of the input being real. Hyperparameters for the discriminator include the input channels, set to 3 to match the RGB image format. Convolutional kernel sizes are predominantly 4x4 across layers, with the last layer employing a 1x1 kernel to produce a single scalar output. The model architecture is depicted below :

Layer (type)	**Output Shape**	**Param #**
Conv2d-1	[-1, 64, 128, 128]	3,072
BatchNorm2d-2	[-1, 64, 128, 128]	128
ReLU-3	[-1, 64, 128, 128]	0
Conv2d-4	[-1, 128, 64, 64]	131,072
BatchNorm2d-5	[-1, 128, 64, 64]	256
ReLU-6	[-1, 128, 64, 64]	0
Conv2d-7	[-1, 256, 32, 32]	524,288
BatchNorm2d-8	[-1, 256, 32, 32]	512
ReLU-9	[-1, 256, 32, 32]	0
Conv2d-10	[-1, 512, 31, 31]	2,097,152
BatchNorm2d-11	[-1, 512, 31, 31]	1,024
ReLU-12	[-1, 512, 31, 31]	0
Conv2d-13	[-1, 1, 30, 30]	8,192

Table 1: Model Architecture

2.4 Justification for Selections

The Rectified Linear Unit (ReLU) activation function is commonly used in the discriminator model for several reasons:Overall, the ReLU activation function is well-suited for the discriminator model in a GAN because of its efficiency, non-linearity, sparsity, and ability to avoid saturation, all of which contribute to better discriminative performance and faster convergence during training.

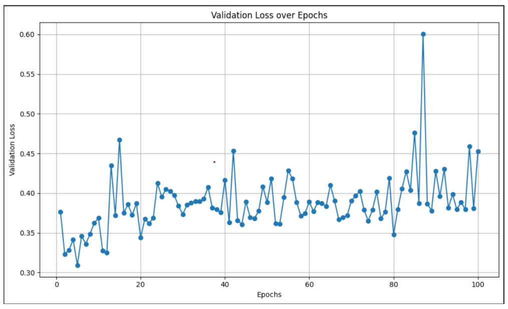
3 Experimentation Results

As discussed in previous section, we have considered cycleGAN for our project. We used pyTorch framework for the training and validation phases. We have experimented by training the CycleGAN in 2 ways. The 1st approach is to directly train the model on the provided training data. And the 2nd approach was to introduce Data Augmentation Technique on the same model and observe its results. The summarization of both of the experiments are documented below:

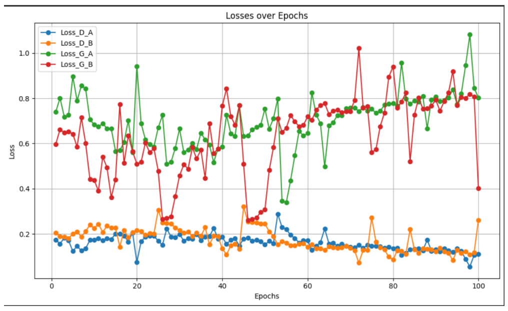
3.1 Basic Cycle GAN

For this approach, we trained the model for 100 epochs, we have provided the plots for Validation loss and GAN Consistency losses. With Loss D A at 0.1104, Loss D B at 0.2611, Loss G A at 0.8028, Loss G B at 0.4013, Loss Cycle A at 0.0574, and Loss Cycle B at 0.0504, the model achieved a Validation Loss of 0.4525 after 100 epochs.

In the first approach without data augmentation, the model is trained solely on the original dataset, which might have limited diversity and variability. As a result, the model may have overfit to the training data, capturing specific patterns and features present only in the training set but not representative of the entire dataset or real-world variations. Consequently, during validation, when the model encounters unseen data or slightly different variations of images (such as different lighting conditions, angles, or perspectives), it may have struggled to generalize well, leading to an increase in validation loss. The model's inability to adapt to these variations results in poorer performance on unseen data.



(a) Validation Loss Plot



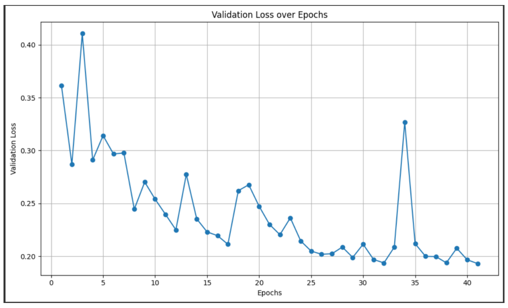
(b) Cycle GAN consistency loss

Figure 1: Various

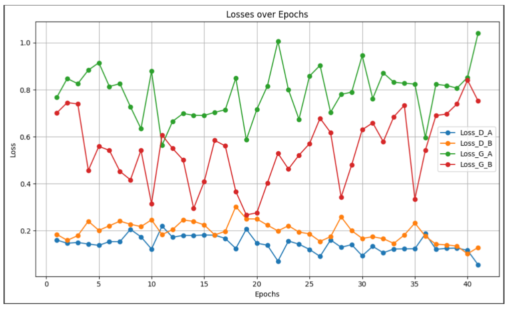
3.2 Cycle GAN With Data Augmentation

Epoch 41 saw Loss D A at 0.0553, Loss D B at 0.1289, Loss G A at 1.0402, Loss G B at 0.7524, Loss Cycle A at 0.0685, and Loss Cycle B at 0.0718, with a Validation Loss of 0.1931.

In this method incorporates data augmentation by rotating the images by 90 degrees anticlockwise during training. Data augmentation introduces additional variations and diversity into the training dataset, effectively increasing its size and diversity without collecting new data. By exposing the model to rotated versions of the images, it learns to recognize and generalize better to variations in orientation and perspective. As a result, the model trained with data augmentation becomes more robust and invariant to such transformations, leading to improved generalization performance on unseen data.During validation, when the model encounters similar but rotated versions of images, it can still perform well due to its exposure to such variations during training. This results in a decrease in validation loss compared to the model trained without data augmentation.



(a) Validation Loss Plot



(b) Cycle GAN consistency loss

Figure 2: Various

4 Other Experimentations

While looking for other options to develop a dehazing GAN for this project, we came across an interesting implementation of CycleGAN which we want to discuss in this section. Authors Deniz Engin, Anıl Genc¸Hazım Kemal Ekenel published a paper "Cycle-Dehaze: Enhanced CycleGAN for Single Image Dehazing" [2]. In ths paper, they proposed an end to end image dehazing pipeline which they termed as Cycle-Dehaze. It is for single image dehazing problem and the authors claim that they do not require a pair of images which constitute ground truth and corresponding images. The proposed network architecture of their model is similar to original CycleGAN [3]but they are having cyclic perceptual-consistency loss and Laplacian pyramid as a post-processing step. The authors have trained their model on the following datasets:

- **I-HAZE:** I-HAZE is a dataset for single image dehazing, primarily used for evaluating algorithms aimed at removing haze and enhancing visibility in single images.
- **O-HAZE:** O-HAZE is another dataset used for single image dehazing research. It consists of hazy images captured under various weather conditions and contains ground truth clear images for evaluation.
- **NYU-Depth:** The NYU-Depth dataset is commonly used for depth estimation tasks. It contains RGB-D images and is often used for training and evaluating depth prediction models.

The model of the authors was comparitively large and we encountered problems while implementing their code on free version of google colab. However, the authors have provided the weights of their model. As a part of our experimentation we have used their pre-trained weights and made them run on the dataset provided for this project.

5 Conclusion and Future Scope

With these experimentations we can conclude that the cycle GAN is a better is better model for image dehazing. We tried to encorporate a training hack (data augmentation). The 90 degree

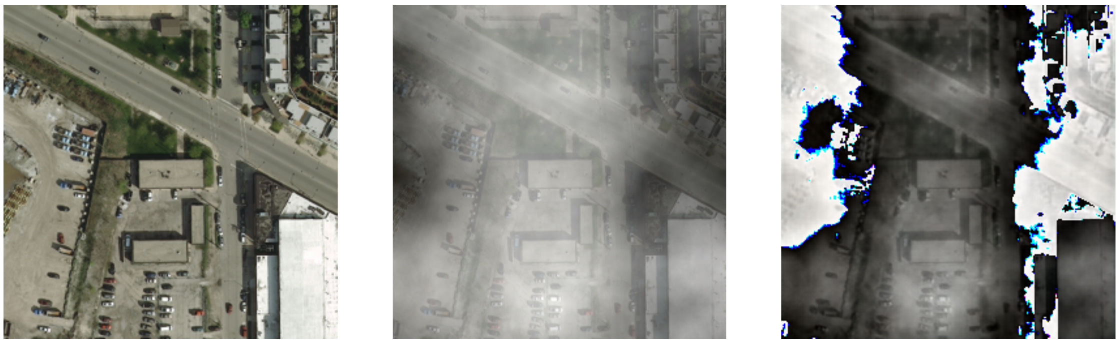

(a) Ground Truth (b) Hazy Image (c) Generated Image

Figure 3: Comparison of Image Outputs

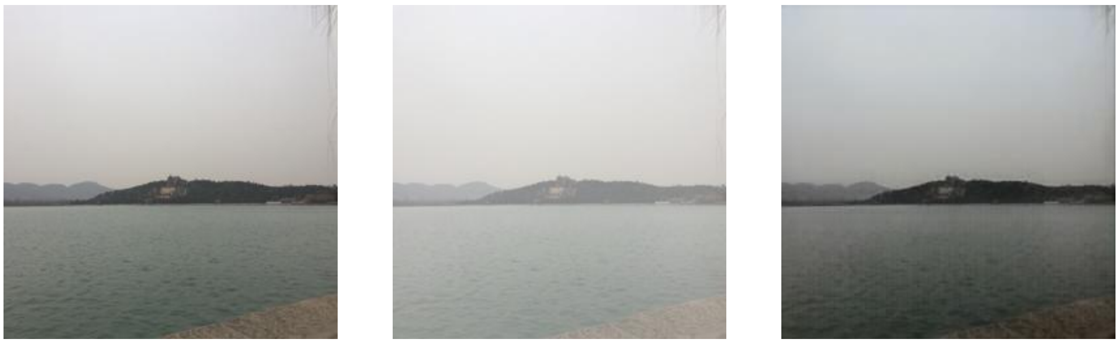

(a) Ground Truth (b) Hazy Image (c) Generated Image

Figure 4: Comparison of Image Outputs

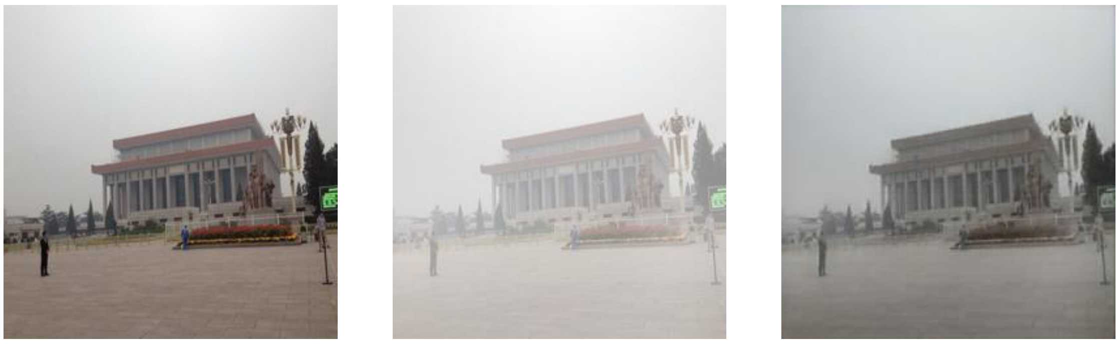

(a) Ground Truth (b) Hazy Image (c) Generated Image

Figure 5: Comparison of Image Outputs

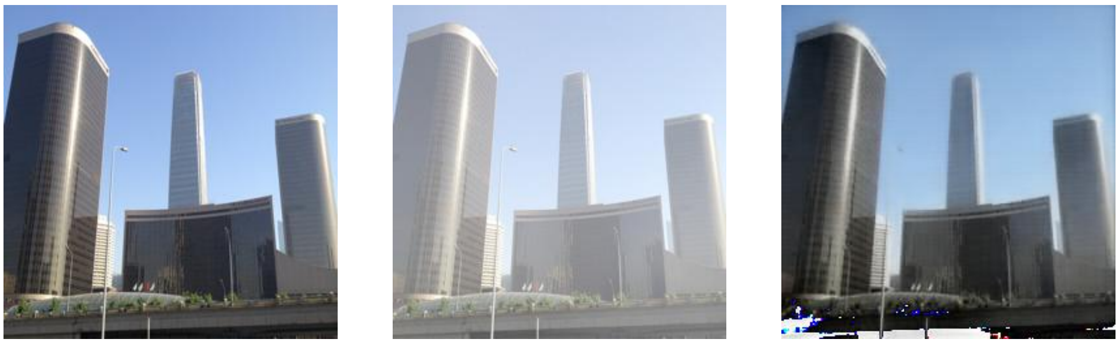

(a) Ground Truth (b) Hazy Image (c) Generated Image

Figure 6: Comparison of Image Outputs

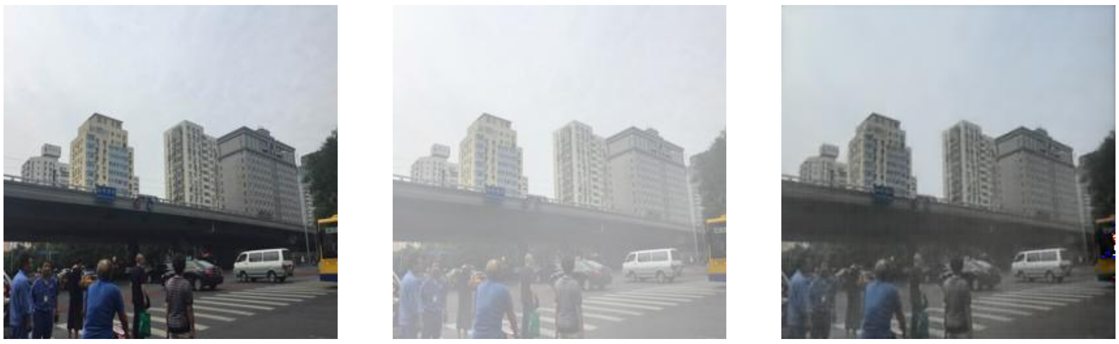

(a) Ground Truth (b) Hazy Image (c) Generated Image

Figure 7: Comparison of Image Outputs

anti-clockwise rotation helped the CGAN model to produce better images. We have also reported the plots of the various loss for both approaches. We have also computed the performance metrics values for both approaches which are:

- For Cycle GAN without augmentation : SSIM = 0.2165
- For Cycle GAN without augmentation : PSNR = 8.6764
- For Cycle GAN with augmentation : SSIM = 0.4627
- For Cycle GAN with augmentation : PSNR = 10.4554

Because of our limitations to access to high end GPU and hardware, we could not train these models for more epochs. But we assume that upon extrapolation of obtained results, we could have achieved more significant metrics values.

References

[1] Aditya Mehta, Harsh Sinha, Murari Mandal, Pratik Narang: ”Domain-Aware Unsupervised Hyperspectral Reconstruction for Aerial Image Dehazing”

[2] Deniz Engin, Anıl Genc¸Hazım Kemal Ekenel: ”Cycle-Dehaze: Enhanced CycleGAN for Single Image Dehazing”

[3] Cycle-Dehaze: Enhanced CycleGAN for Single Image Dehazinghttps://github.com/engindeniz/Cycle-Dehaze

[4] Cycle-Dehaze: Enhanced CycleGAN for Single Image Dehazing code is based on CycleGAN-TensorFlow implementation.https://github.com/vanhuyz/CycleGAN-TensorFlow