Project Report
Image Dehazing Using GAN based architecture

Deep Learning CS F425
Group:
Rudra Jewalikar : 2021A7PS0450P
Dhruv Shrimali : 2021A7PS0008P
Salil Godbole : 2021A7PS2004P

Data Preprocessing and Augmentation:

1. **Data handling:** Data handling posed a significant challenge in our project, particularly in managing a large volume of image data. Initially, we attempted to store the images in either a single or multiple NumPy arrays. However, we encountered memory constraints when working within the Google Colab environment. To address this limitation, we adopted a batch processing approach, where we imported the images directly in batches of 32 during the training phase. This allowed us to efficiently train our model on the available data while circumventing the memory limitations imposed by the environment.
2. **Data Augmentation**: To improve the model's colour accuracy, brightness and contrast adjustments were randomly applied to augment the training data. Given the model's focus on dehazing, techniques like flipping and zooming were omitted, as they were deemed less relevant for improving dehazing performance.

Model Choice:

We chose Pix2Pix over CycleGAN for dehazing images primarily because our dataset comprised paired hazy and clear images, aligning well with Pix2Pix's strength in tasks with clear input-output mappings. Pix2Pix's faster training and convergence, along with its simpler architecture and ease of implementation, made it a better choice given our resource constraints on Google Colab. Its suitability for structured image translation tasks and ability to preserve fine details further solidified our decision. Overall, Pix2Pix emerged as the preferred option for our image dehazing task due to its compatibility with our dataset, training efficiency, and effectiveness in preserving image details.

Model Architecture :

1. **Generator :**
    - The generator is implemented using a U-Net architecture, which is a common choice for image-to-image translation tasks like Pix2Pix. It consists of downsampling and upsampling blocks.
    - Downsampling layers reduce the spatial dimensions of the input image while increasing the number of channels, capturing abstract features.
    - Upsampling layers aim to reconstruct the output image from the learned features.
    - There are total 14 layers, 7 in downsampling and 7 in upsampling

2. **Discriminator (D):**
    - The Discriminator follows a PatchGAN architecture, where it classifies image patches as real or fake. This approach allows the discriminator to provide spatially-localized feedback, which encourages the generator to produce realistic textures.
    - The Discriminator consists of 4 convolutional layers. These layers analyze image patches and provide spatially-localized feedback for guiding the Generator's training.
    - The discriminator trains on a single image along with its label (Label =0 for generated images and Label=1 for GT images) as input.

3. **Adversarial Loss:**
    - In this implementation, the adversarial loss is implemented conventionally. The discriminator is trained to distinguish real and fake image pairs, while the generator aims to generate realistic images to fool the discriminator.
    - The discriminator uses mean squared error (MSE) loss, and the generator's loss is a combination of MSE and mean absolute error (MAE) losses. The MAE loss penalizes the pixel-wise difference between the generated and target images.

Training Process :

1. **Initial Training Phase:** Both the Generator and Discriminator underwent an initial training phase of 2 epochs.
2. **Freezing the Discriminator:** Upon observing the Discriminator achieving close to 100% accuracy, indicative of effective discrimination, it was decided to freeze its weight to avoid overfitting by preventing it from becoming overly specialized on the training data.
3. **Focus on Generator Training:** By pausing Discriminator updates, the focus shifted solely to training the Generator for 5 epochs, allowing it to catch up and refine its image generation capabilities without interference.
4. **Unfreezing the Discriminator:** After the dedicated training phase for the Generator, the Discriminator was unfrozen.
5. **Continued Training:** Both networks underwent an additional 5 epochs of training after the Discriminator was unfrozen.
6. **Early Stopping**: After the 12th epoch the model was not improving significantly. So for further training we tried training both generator and the discriminator on the augmented data, but we found that it degraded the models performance on the validation data. So we decided to stop at 11th epoch

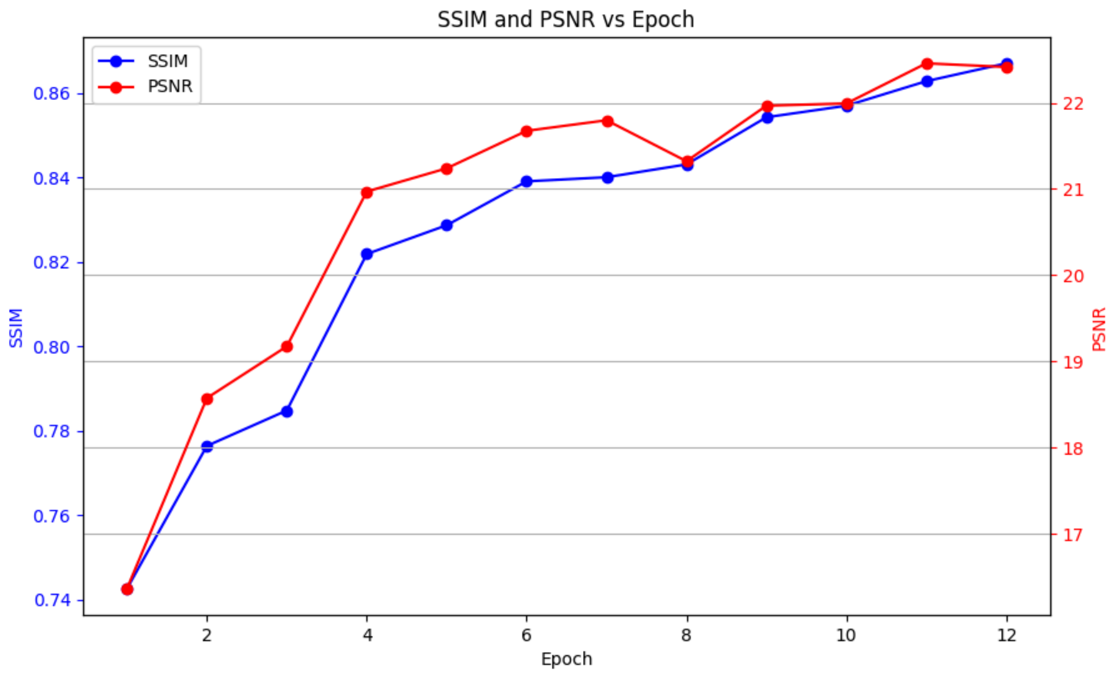
   The pause and subsequent continuation facilitated a more balanced learning process, ensuring the refinement of learned representations and overall performance improvement. Early stopping prevented the model from overfitting.

Hyperparameters:

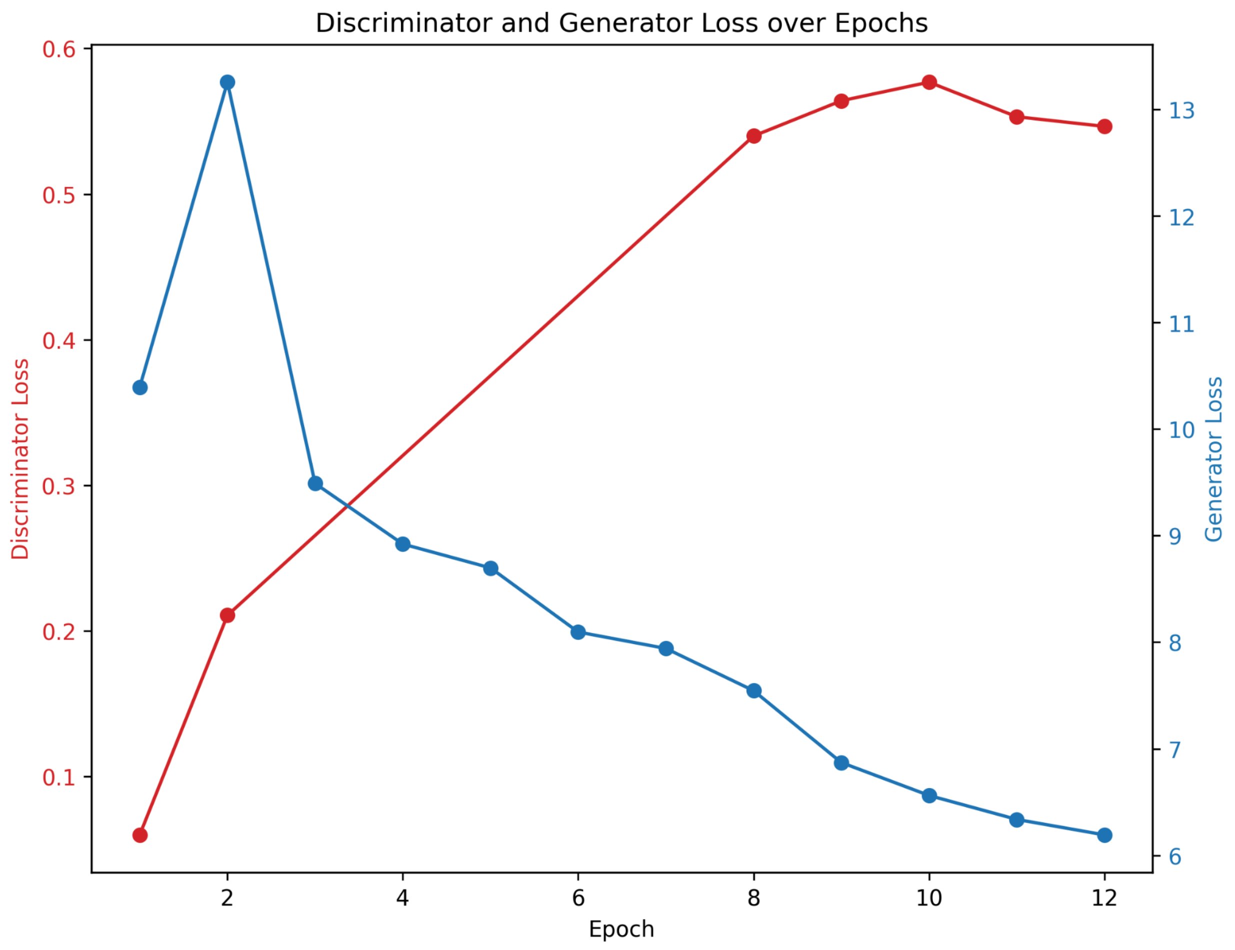
1. **Image Resolution**: Set to 256x256 pixels (`img_rows`, `img_cols`), determining the input image size.
2. **Number of Channels**: Defined as 3 (`channels`), representing RGB color images.
3. **Generator Filters:** The number of filters in the first layer of the generator (`self.gf`) is set to 64, influencing the architecture's depth and complexity.
4. **Discriminator Filters:** Similarly, the number of filters in the first layer of the discriminator (`self.df`) is also set to 64.
5. **Optimizer Parameters:** The learning rate (`0.0002`) and momentum (`0.5`) for the Adam optimizer, to control the rate of parameter updates during training.
6. **Loss Weights:** In the combined model's compilation, the loss weights (`[1, 100]`) determine the balance between adversarial and pixel-wise losses. The latter, weighted higher (100), emphasizes image fidelity during training.

Results and Conclusion:

On validation data : SSIM= 0.84003   PSNR= 21.7970



It can be observed from this graph that rate of improvement had really slowed down in the last 3 epochs.



From the graph it is observed that when the Discriminator was frozen after the 2nd epoch the generator started improving much more rapidly. And when the Discriminator was unfrozen at the 8th epoch, Discriminator loss shot up which indicates that generator's progress caught up with the discriminator.

Here are few example images generated by the model: