**A blue shield with a cross and a book

Description automatically generated**

**Project 4 For Diploma in AI and its application in Business:**

**Image Dehaze with Conditional Generative Adversarial Network**

**Project Report**

*Image Generation and Example Implementation on cGAN Architecture*

Prepared by: Chen Yijie

Supervisor: Zhou Changxin

Table of Contents

[Project Overview: 3](#_Toc164174475)

[Covering Concepts: 3](#_Toc164174476)

[Potential Use Case: 5](#_Toc164174477)

[Environment Used: 7](#_Toc164174478)

[Dataset Used 7](#_Toc164174479)

[Preprocessing: 8](#_Toc164174480)

[cGAN Implementation: 11](#_Toc164174481)

[Training Parameter: 16](#_Toc164174482)

[Initial Results: 19](#_Toc164174483)

[Transpose Convolution 21](#_Toc164174484)

[Final Result 22](#_Toc164174485)

[Potential Improvements 24](#_Toc164174486)

[Conclusion: 24](#_Toc164174487)

**Project Overview:**   
The project aims to develop an image dehazing system using Conditional Generative Adversarial Networks (CGANs) to enhance visibility and clarity in hazy images. CGANs generate realistic, dehazed images from hazy inputs, effectively removing atmospheric haze. Unlike traditional methods, CGANs learn the mapping between hazy and clear images, resulting in visually pleasing dehazed images without loss of detail. This approach holds promise for various applications, including surveillance and autonomous driving, where clear visibility is crucial for accurate decision-making.

## **Covering Concepts:**

1. **Convolutional Neural Network (CNN):**

CNNs are a class of deep neural networks commonly used for processing visual data, such as images. They consist of convolutional layers that apply learnable filters to input images to extract spatial features. These networks are particularly effective for image processing tasks due to their ability to capture local patterns and hierarchies of features.

1. **Pooling:**

Pooling layers in CNNs reduce the spatial dimensions of feature maps, thereby reducing computational complexity while retaining important information. Max pooling and average pooling are commonly used techniques to downsample feature maps by selecting the maximum or average value within each pooling region.

1. **Upsampling:**

Upsampling layers in CNNs increase the spatial dimensions of feature maps, allowing for the reconstruction of high-resolution images from low-resolution representations. Techniques like nearest neighbor interpolation, bilinear interpolation, and transposed convolution (deconvolution) are employed for upsampling.

1. **Generative Adversarial Networks (GAN):**

GANs consist of two neural networks, a generator and a discriminator, trained simultaneously in a competitive manner. The generator generates realistic data samples, while the discriminator distinguishes between real and generated samples. Through adversarial training, GANs learn to generate high-quality data samples that are indistinguishable from real data.

1. **Conditional Generative Adversarial Networks (cGAN):**

cGANs extend the GAN framework by conditioning both the generator and discriminator networks on additional information, such as class labels or input images. In the context of image dehazing, the generator takes hazy images as input and generates corresponding dehazed images, while the discriminator learns to distinguish between real clear images and generated dehazed images.

1. **Loss Functions in cGAN:**

Loss functions in cGANs play a crucial role in guiding the training process. Commonly used loss functions include the adversarial loss, which encourages the generator to generate realistic images that fool the discriminator, and additional perceptual losses such as L1 or L2 loss, which ensure pixel-wise similarity between generated and ground truth images.

1. **Image Dehazing:**

Image dehazing is the process of removing the effects of atmospheric haze from images to enhance visibility and clarity. In the context of cGANs, dehazing involves training a generator network to learn the mapping between hazy and clear images, effectively restoring visual fidelity by generating dehazed images from hazy input images.

1. **Conditional Image Generation:**

Conditional image generation refers to the generation of images conditioned on additional information, such as input images or context labels. In the case of cGAN-based image dehazing, the generator network learns to produce dehazed images conditioned on hazy input images, enabling the removal of atmospheric haze while preserving important visual features.

1. **Transfer Learning:**

Transfer learning involves leveraging knowledge learned from one task or domain to improve performance on a related task or domain. In the context of cGAN dehazing, transfer learning techniques may be applied to leverage pre-trained models or feature extractors trained on large-scale image datasets, enhancing the dehazing performance and generalization capability of the model.

1. **Evaluation Metrics:**

Evaluation metrics are used to assess the performance of cGAN-based image dehazing algorithms. Commonly used metrics include Peak Signal-to-Noise Ratio (PSNR), Structural Similarity Index (SSIM), and perceptual quality metrics such as the Fréchet Inception Distance (FID). These metrics quantify the visual quality and similarity between generated dehazed images and ground truth clear images.

## **Potential Use Case:**

A well-trained Conditional Generative Adversarial Network (cGAN) for image dehazing can find application in various domains where visibility and clarity in images are crucial. Here are some potential use cases:

1. **Autonomous Vehicles**:

Autonomous vehicles heavily rely on visual data for navigation and obstacle detection. A cGAN-based dehazing system can enhance the visibility of road scenes captured by onboard cameras, improving the vehicle's perception of the environment and ensuring safer navigation, especially in adverse weather conditions like fog or haze.

1. **Surveillance Systems**:

Surveillance systems often encounter challenges in capturing clear images under varying lighting and environmental conditions. By deploying a cGAN-based dehazing system, surveillance cameras can produce clearer images, enabling better detection and identification of objects, people, and activities, thereby enhancing security monitoring capabilities.

1. **Remote Sensing**:

Remote sensing applications, such as satellite imagery and aerial photography, are essential for environmental monitoring, disaster management, and urban planning. A cGAN-based dehazing system can improve the quality of satellite images by removing atmospheric haze, allowing for more accurate analysis of land cover, vegetation health, and environmental changes.

1. **Medical Imaging**:

Medical imaging technologies, such as X-rays, CT scans, and MRI, often suffer from image degradation due to factors like noise and artifacts. By integrating a cGAN-based dehazing system into medical imaging pipelines, healthcare professionals can obtain clearer and more detailed images, leading to more accurate diagnoses and treatment planning.

1. **Art Restoration and Preservation**:

Cultural heritage institutions and art conservators can benefit from cGAN-based dehazing techniques for restoring and preserving historical artworks and artifacts. By removing haze and enhancing image clarity, these systems can reveal hidden details and restore the original appearance of deteriorated artworks, contributing to cultural heritage conservation efforts.

1. **Environmental Monitoring**:

Environmental monitoring applications, such as air quality assessment and pollution detection, rely on image data collected from outdoor sensors and cameras. A cGAN-based dehazing system can improve the visibility of environmental images, enabling more accurate analysis of atmospheric conditions, pollutant concentrations, and environmental changes over time.

1. **Real Estate and Property Management**:

Real estate agencies and property management companies can use cGAN-based dehazing systems to enhance property photographs for marketing purposes. Clearer and more visually appealing images can attract potential buyers or tenants by showcasing properties in their best light, leading to faster transactions and higher customer satisfaction.

## **Environment Used:**

Python programming language with:

1. tensorflow = ^2.15.0

2. keras = ^2.15.0

3. tensorboard = ^2.15.0

4. numpy = ^1.26.3

5. matplotlib = ^3.8.2

## **Dataset Used**

For this project, the dataset used will be the Dehaze dataset available on Kaggle consists of a collection of hazy images paired with their corresponding ground truth clear images. The dataset contains 840 original images as ground truth and 13,000 artificially generated hazed images from the original. These images are diverse, covering various scenes and environments affected by atmospheric haze, such as urban landscapes, natural settings, and indoor scenes. The dataset is suitable for training and evaluating image dehazing algorithms, including deep learning approaches.

## **Preprocessing:**

The dehaze dataset images needs to be prepared into original images and hazy image pair for training to take place. Some of the common preprocessing practices are used to prepare the datasets:

1. **Defined a data\_path function:**

This function takes in the paths to the original images and hazy images as inputs. It's responsible for organizing the dataset into train and validation sets. Typically, it reads the image files, pairs each hazy image with its corresponding original clear image, and splits the dataset into training and validation subsets based on a predefined ratio.

A screen shot of a computer program

Description automatically generated

Fig 1. Code Implementation that Defines data\_path Function

1. **Visualize a sample of the image pair:**

After loading the dataset, it's important to visualize a sample pair of hazy and original clear images to ensure that the data has been loaded correctly and that the pairs correspond appropriately. Visualization are done using Matplotlib to display the images side by side or in a grid format.



Fig 2. Code Implementation and Visualization of Image Pair

1. **Define a function to translate a jpeg image given its path:**

This function takes the path to an image file as input and returns the corresponding image data. It's responsible for reading the image file, decoding it into a suitable format (e.g., JPEG, PNG), and converting it into a numerical representation that can be processed by machine learning algorithms. This function involves tasks such as resizing images to a uniform size, normalizing pixel values, and will be used within the data loading pipeline to load individual images as needed.

A black background with text

Description automatically generated

Fig 3. Code Implementation on load\_image Function Defination

1. **Define a dataloader function:**

The dataloader function is responsible for loading all translated data and preparing it in a format suitable for training machine learning models, particularly deep learning models like convolutional neural networks (CNNs). By calling the load\_image iteratively, the function is able to generate ground truth and hazed image pairs, zip them together and batch the data to facilitate efficient processing. It prepares the dataset for training by converting it into TensorFlow-compatible data structures (e.g., TensorFlow Dataset objects) that can be fed into the model during training.

A screen shot of a computer code

Description automatically generated

Fig 4. Code Implementation on dataloader Function that Calls load\_image and Batch the Data

1. **Define a batch size and load data into train and validation datasets:**

Once the data has been preprocessed and organized, a batch size is defined to specify the number of image pairs to be processed simultaneously during training. Typically, smaller batch sizes are used for training, as they allow for more frequent updates to the model parameters and can help improve convergence, more details will be discussed during the training parameters section. After defining the batch size, the preprocessed data is loaded into separate train and validation datasets, which are then used for training and evaluating the model, respectively.

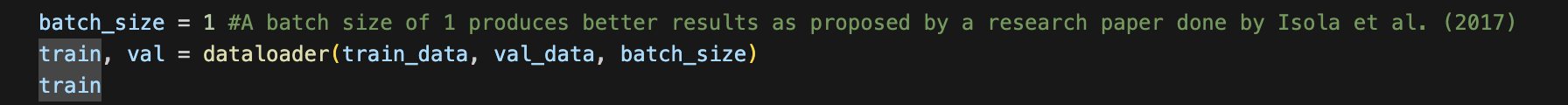
Bottom of Form

Fig 5. Code Implementation that Loads Data into Trai

ning and Validation

## **cGAN Implementation:**

Implementing a Conditional Generative Adversarial Network (cGAN) for image dehazing requires careful consideration of its architecture and components. The cGAN structure incorporates distinct features tailored to this task, ensuring effective extraction of relevant image features and generation of high-quality, dehazed outputs.

1. **DownSampling Block**: The downsampling block is one out of the 2 foundation building blocks in the GAN structure. It consists of a convolutional layer, a batch normalization layer on demand and a Leaky Rectified Linear Unit(ReLU) function. The downsampling block’s function is to extract features from an image or feature map input and output feature map that is smaller in dimension.

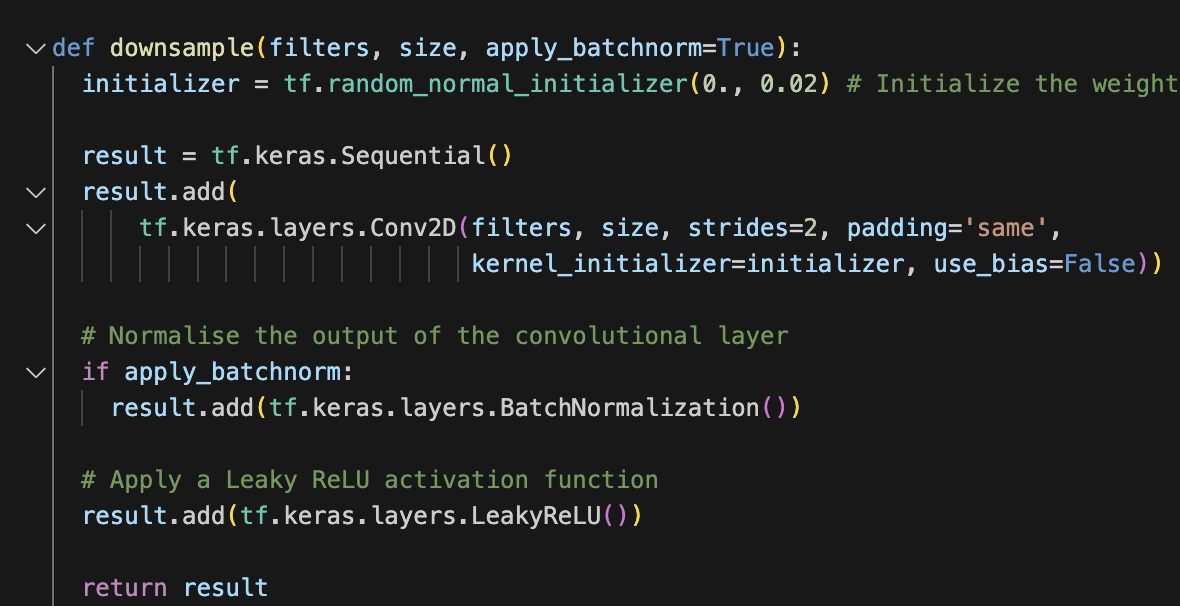


Fig 6. Code Implementation for downsampling Block

1. **UpSampling Block**: The second building block is the upsampling block consist of a bilinear upsampling layer, a convolutional layer, a batch normalization layer, a drop out layer on demand and a normal ReLU activation function. The Upsampling block’s function is to return a bigger size image given a smaller size feature map.

A screen shot of a computer program

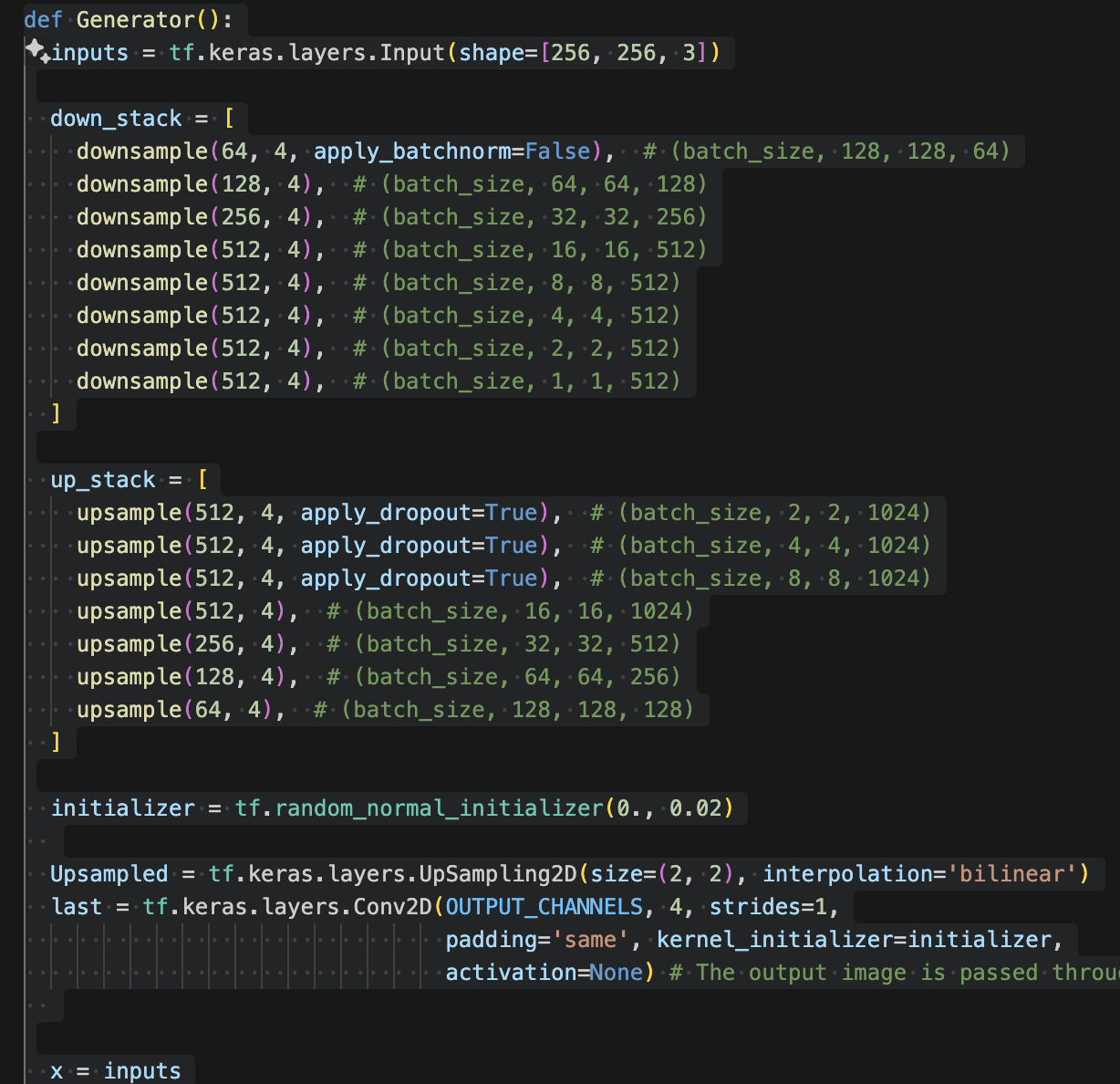
Description automatically generated

Fig 7. Code Implementation for upsampling Block

1. **Generator**: In a conditional Generative Adversarial Network (cGAN), the generator takes as input a random noise vector along with conditional information, such as a low-quality or hazy image in the case of an image dehazing task. The generator's role is to transform this input into a high-quality, dehazed version of the image. In the context of a basic cGAN architecture with stacks of downsampling and upsampling blocks, the generator comprises stacks of downsampling and upsampling blocks.

The downsampling blocks extracts low to high level features from the given hazy image into smaller feature maps and these upsampling blocks progressively increase the spatial dimensions of the feature maps while refining the details of the image. The feature maps along the process will be saved separately in a list which will be referenced later by upsampling layers to ensure maximum information retainment.

Each upsampling block takes in a lower resolution feature map from the previous block together with the feature map output from the corresponding downsampling layer that is saved, then applies upsampling, convolution, batch normalization, and activation operations to generate a higher resolution feature map. The final output of the generator is the dehazed image.



A screen shot of a computer program

Description automatically generated

Fig 8&9 Code Implementation for Generator model and Skip Connection Establishment

1. **Discriminator**: The discriminator in a cGAN serves as the adversary to the generator. Its objective is to distinguish between real high-quality images and fake images generated by the generator. The discriminator's architecture typically consists of downsampling blocks.

Similar to the generator, the discriminator is composed of multiple stacks of downsampling blocks. These blocks progressively reduce the spatial dimensions of the feature maps while extracting increasingly abstract and high-level features. The final output of the discriminator is a probability score indicating the likelihood that the input image is real (as opposed to generated).

A computer screen shot of a program code

Description automatically generated

Fig 10. Code Implementation for Discriminator

## **Training Parameter:**

In training the Q-learning agent for reinforcement learning, specific hyperparameters are chosen based on their impact on the learning process and the characteristics of the environment. Here's an explanation of the reasons for the chosen hyperparameters:

1. **Batch Size = 1:** The utilization of a batch size of 1 is chosen according to a research paper done by Isola et al. (2017) to maximize the precision of parameter updates during training. For image dehazing tasks, where quality is paramount, this approach ensures that each sample receives undivided attention, leading to more accurate adjustments in the model parameters. By processing individual samples per iteration, the model can better capture nuanced features in the data, facilitating superior convergence and stability. Furthermore, the reduction in memory requirements associated with a batch size of 1 allows for the exploration of larger and more complex models without compromising performance.

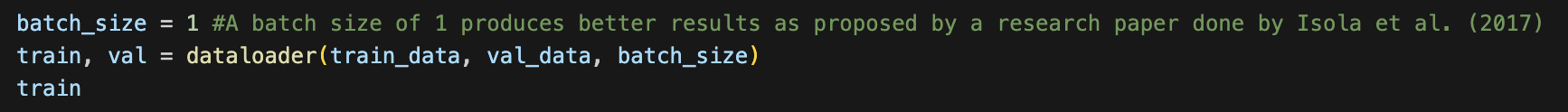


Fig 11. Code Implementation to Set the Batch Size

1. **Downsample Stack = 8:** The decision to incorporate 8 downsampling blocks within the generator architecture is driven by the necessity to capture hierarchical features from the input hazy images comprehensively. By stacking multiple downsampling blocks, the generator can extract increasingly abstract representations of the input, enabling a more effective transformation process towards clearer, dehazed images. This depth in feature extraction is critical for accurately removing haze and restoring intricate image details, ultimately resulting in visually appealing outputs.
2. **Upsample Stack = 8**: In alignment with the downsampling stack in order to establish skipping connection, the inclusion of 8 upsampling blocks within the generator architecture aims to restore finer details and enhance image resolution. Each upsampling block contributes to enlarging the spatial dimensions of the feature maps, progressively refining the generated images. By leveraging multiple upsampling blocks, the generator can better reconstruct high-frequency components lost during the hazing process, leading to a significant improvement in the perceptual quality and realism of the dehazed images.
3. **Lambda for L1 Gen Loss = 100**: A lambda value of 100 for the L1 loss term underscores the paramount importance placed on preserving image details and minimizing reconstruction error during training. By assigning a high weight to the L1 loss, the generator is incentivized to produce dehazed images that closely resemble the ground truth images in terms of pixel-wise intensity values. This strategic emphasis ensures that the generated images retain crucial structural and textural characteristics, resulting in outputs of superior quality and fidelity.

A screen shot of a computer code

Description automatically generated

Fig 12. Code implementation to Set L1 Loss Weight

1. **Learning Rate = 2e-4:** The selection of a learning rate of 2e-4 strikes an optimal balance between convergence speed and stability during training. This moderate learning rate facilitates smooth updates to the model parameters, preventing abrupt fluctuations that may impede convergence. By carefully calibrating the learning rate, the optimization process can efficiently navigate the loss landscape, progressively enhancing the model's performance and convergence behavior.
2. **Beta\_1 = 0.5:** With a beta\_1 value of 0.5 for the Adam optimizer, equal weight is given to the current gradient and historical gradient estimates during parameter updates. This deliberate choice ensures a balanced consideration of gradient information, stabilizing the optimization process and mitigating excessive oscillations in the parameter space. By maintaining stability throughout training, the optimizer can effectively guide the model towards the optimal solution, facilitating smoother convergence and improved performance.

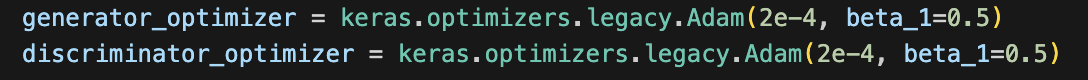


Fig 13. Code Implementation to Set Learning Rate and Beta

1. **Steps = 50001:** By setting the total number of training steps to 50001, a specific training duration or budget is established for the model. This deliberate decision allows for ample iterations to capture underlying patterns in the data and refine the model parameters accordingly. With a defined training duration, the training process becomes more manageable and conducive to monitoring progress and evaluating performance effectively, ultimately leading to the generation of high-quality, dehazed images.

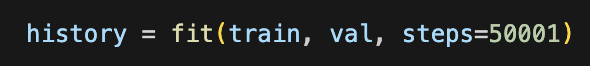


Fig 14. Code Implementation to Set Training Steps

## **Initial Results:**

The model achieved PSNR of 20 shortly after only 4000 steps of training and its results has been fluctuating around 20 afterwards. Not only the PSNR, but the generator loss also fluctuates showing an unfavorable training outcome. The model is able to introduce more color vibrancy to the hazy images but there are often exploding pixels, jeopardizing the picture quality of the restored image.

A graph of different colored lines

Description automatically generated

Fig 15. Training History of First Model

A street with buildings and cars

Description automatically generated

A car driving down a street

Description automatically generated

Fig 16&17. First Model performance

## **Transpose Convolution**

The Model’s suboptimal performance might be a result of over-utilising Bilinear Upsampling, causing nonsense pixels due to the interpolation process. Bilinear upsampling calculates new pixel values by taking a weighted average of neighboring pixels in the lower-resolution input image. While this method is simple and computationally efficient, it may introduce artifacts and "nonsense" pixels, particularly in regions with sharp transitions or fine details. This project looked into using transpose convolution to replace some bilinear upsampling blocks and investigate the effect.

Transpose convolution, also known as deconvolution or fractionally strided convolution, offers a solution to mitigate the generation of nonsense pixels. Instead of relying solely on interpolation, transpose convolution learns to upsample feature maps through trainable filters, allowing the network to adaptively generate pixel values based on the surrounding context. By learning these filters during training, the network can better capture spatial dependencies and produce more accurate upsampling results.

Additionally, transpose convolution enables the network to learn complex patterns and structures in the data, which may not be adequately captured by simple interpolation methods like bilinear upsampling. This flexibility makes it particularly beneficial for tasks that require high-fidelity reconstruction, such as image generation or super-resolution.

A screen shot of a computer program

Description automatically generated

Fig 18. Hyperparameters and Code Implementation for Re-training

## **Final Result**

The updated model with 7 bilinear upsampling block and 1 transpose convolution upsampling block achieved highest PSNR of 36 after 18000 steps of training. The training results fluctuates at around PSNR of 30, which is a great improvement.

Human rendering of the generated pictures exposes no obvious flaw and thus the project can be deemed successful. When viewing the magnified images, checker box like artifacts can be found showing that more improvements can be done to finetune the project.

A graph of different colored lines

Description automatically generated

Fig 19. Training History of the Updated Model

A group of buildings in a city

Description automatically generated

Fig 20. Example Model Performance

A blue sky with clouds

Description automatically generated

Fig 20. Checker Box Like Artifact that Can be Found After Magifying

## **Potential Improvements**

Some potential improvements that can be implemented includes:

* 1. Further finetune the model by changing hyperparameters.
  2. The checker box artifact should be a result of uneven (Overlapping) upsampling, different filter size can be tested to find out the appropriate size.

## **Conclusion:**

In conclusion, this project has delved deeply into the realm of computer vision, particularly focusing on the innovative application of conditional Generative Adversarial Networks (cGANs) for image dehazing. Beginning with foundational concepts, the project elucidated the principles underlying image dehazing, where models iteratively learn to transform hazy images into clear, visually appealing representations.

The study progressed to the implementation of cGANs, a powerful framework that leverages the adversarial training paradigm to generate high-quality, dehazed images. By incorporating stacks of downsampling and upsampling blocks within the generator architecture, the project enabled the extraction of intricate features from hazy inputs and the restoration of fine details in the output images.

Through meticulous design and training, the cGANs were equipped to effectively address the challenges posed by haze, producing dehazed images that closely resemble the ground truth. The utilization of hyperparameters such as batch size, network depth, and loss weighting facilitated the optimization process, leading to the generation of visually compelling results.

Moreover, the project elucidated critical concepts such as adversarial training, feature extraction, and image reconstruction, which are fundamental to understanding and implementing cGAN methodologies proficiently. By bridging theory with practical implementation, learners gained valuable experience in developing and training cGAN models, equipped with the skills to tackle complex challenges in image dehazing tasks.

In summary, this project has offered a comprehensive understanding of cGANs for image dehazing, providing a robust foundation for further exploration and application of computer vision techniques in various real-world scenarios. From fundamental principles to advanced implementations, learners are now prepared to engage in the field of computer vision with confidence and proficiency, contributing to advancements in image processing and restoration.Bottom of Form