CS512 - Project - Report

Sarvesh Shroff | Akshay Jain

A20488681 | A20502846

Department of Computer Science

Illinois Institute of Technology

April 18, 2022

Single Image Haze Removal using GANs

Abstract

When working with a single image, it becomes a challenging problem to remove haze as a single image lacks depth knowledge. So in this project we would use modified conditional Generative Adversarial Network to directly remove haze from the image.

Problem Statement

Haze is traditionally an atmospheric phenomenon in which dust, smoke, and other dry particulates obscure the clarity of the sky. Due to this light gets scattered and from the viewer's perspective, the field of view is minimized. Also, the hazy image has reduced visibility and reduced contrast. So when visibility is of main importance for example in autonomous vehicles this becomes a cause of concern.

Proposed Solution

We propose to implement this using the paper Single Image Haze Removal using a Generative Adversarial Network (Raj & N, 2018). With respect to paper, we will implement an end-to-end learning-based approach that uses a modified conditional Generative Adversarial Network to directly remove haze from a single image without explicitly estimating the transmission map. We would use two different models Tiramisu (Jégou et al., 2016) and U-Net (Ronneberger et al., 2015) model for the Generator and use the one which performs best.

Generative Adversarial Networks (GANs)

GANs (Goodfellow et al., 2014) is an approach for generative modeling using deep learning methods for our case using convolutional neural networks. This is an unsupervised approach where discovering and learning the regularities or patterns in input data is automated/ not supervised. It creates new data similar to the one in the dataset used for training the model. Here we use two models namely generator and discriminator where generator tries to create realistic images as close to the images in the dataset and discriminator tries to classify if the images are real or fake. They both are trained together in a zero-sum game until the discriminator model is fooled half of the time by the generated images from the generator.

In our model we have used U-Net (Ronneberger et al., 2015) and Tiramisu (Jégou et al., 2016) models as our generator and patch wise discriminator. Here a hazed image is given as input to the generator and it produces an output image where the haze is removed. Now the pair of haze image and generated dehazed image and pair of hazed image and ground truth image is given to the discriminator and the discriminator tries to predict if the given image is real or fake/generated. The loss from the discriminator and the loss from the generated image from the generator is added in the ratio of 1:100 and given back to the generator model for updating the weights, and the loss of discriminator is given back to the discriminator to update weights in the parameter space. We used 1:100 for the generator because as per pix2pix gans (Isola et al., 2016) this ratio helps to generate much better results in the generator at a faster rate hence making learning faster.

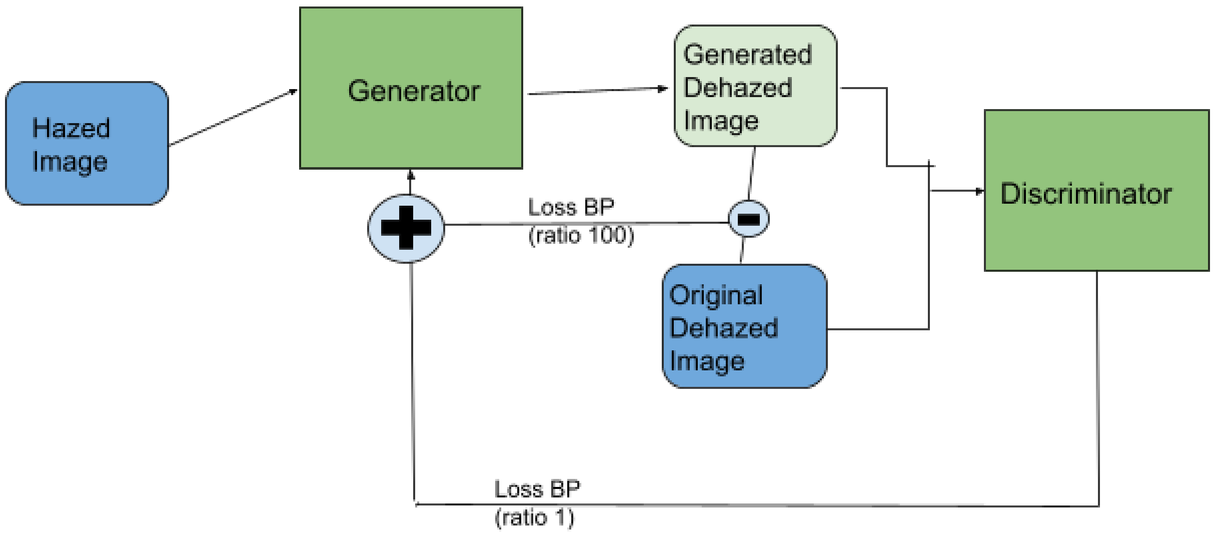


Fig. GANs Architecture Diagram

Tiramisu as generator

We used 56 layer Tiramisu (Jégou et al., 2016) architecture for the generator model. In this we have taken the haze image as input with (256,256,3) shape and then passed it through a Convolution layer having 32 feature maps/depth then passed it through a Dense block (Huang et al., 2016) having 4 layers followed by a transition down block. We made this setup 5 times and then had a dense block with 15 layers in it as the bottleneck. After which to get back the image we passed the tensor obtained from the bottleneck block through a transition up block followed by a skip connection from the same depth of the transition down path and passed it through a dense block. This was repeated for 5 times and then finally the output tensor from the dense block on the up path was passed through a Conv2DTranspose layer having feature maps same as that of the input image of 3. This resulted in the same shape of (256,256,3) for the output image that was generated.

Dense Block (Huang et al., 2016) comprised 4 layers each layer comprising of a batch normalization layer followed by ReLU and a conv2d layer of (1,1) kernel having 4 times the feature maps than the input of 32, and then again batch normalization layer followed by ReLU and a conv2d layer of (3,3) kernel having feature maps of 32 making a layer of the dense block. After each layer of dense block the feature maps of the above layer's feature maps are concatenated and passed through the next layer of the dense block.

Transition Down block comprises of a batch normalization followed by Relu layers and then conv2d layer having kernel size of (1, 1) and changing feature maps to 32 , followed by dropout layer with dropout probability set to 0.2, and finally average pool is applied on the output of the dropout layer. This is used to decrease the dimensions of the image.

Transition Up block only has a conv2dTranspose layer with kernel size of (3, 3) and having feature maps of 32. This is needed for increasing the dimensions of the image.

Bottleneck is a dense block comprising 15 layers where all previous layers output is concatenated and given to the layer.

All these layers had 0-padding enabled and stride set to (2, 2). Due to this setup in the down path applying conv2d reduces the spatial resolution of the image to half while increasing the feature maps and on the up path applying the conv2dTranspose doubles the spatial resolution of the image tensor. As the size is upscaled/downscaled by factor of 2 it enables us to concatenate the feature maps from the down path to up path where spatial resolution is the same.Due to this concatenations or the skip connections information of the feature maps from the initial layers is passed to the later layer hence making sure overfitting does not take place and losses are back propagated to all layers with almost equal weights.

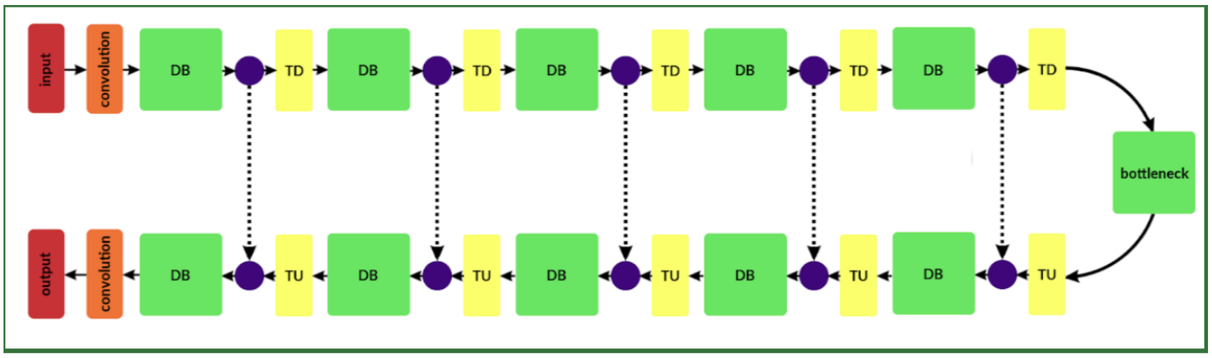


Fig . Generator of the Tiramisu model. It consists of Dense Blocks (DB), Transition Down layers (TD), Transition Up layers (TU) and a Bottleneck layer. Additionally, it contains an input convolution and an output convolution. Dotted lines imply a concatenation operation. (Raj & N, 2018)

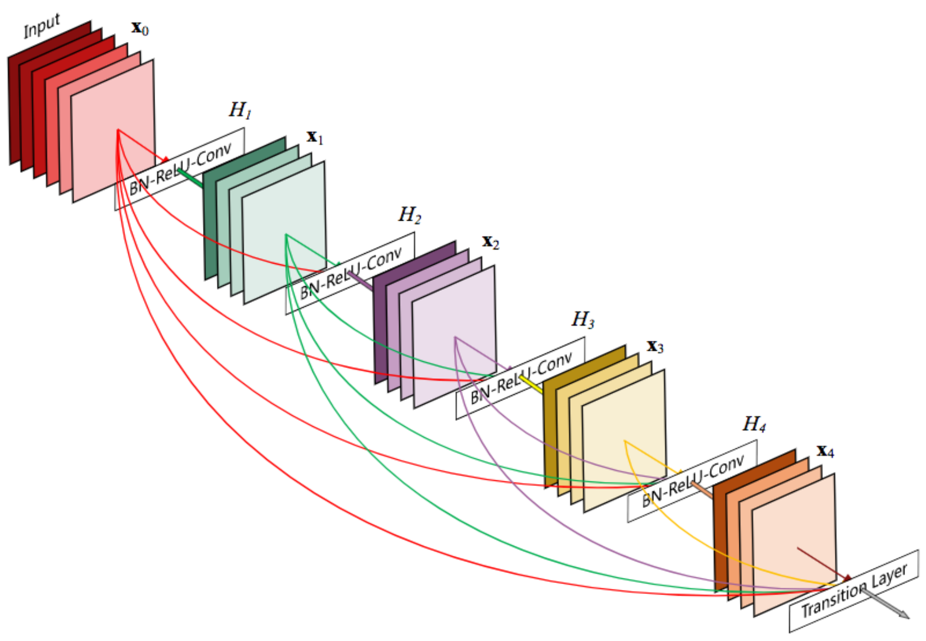


Fig. A 5-layer dense block with a growth rate of k = 4. Each layer takes all preceding feature-maps as input. (Huang et al., 2016)

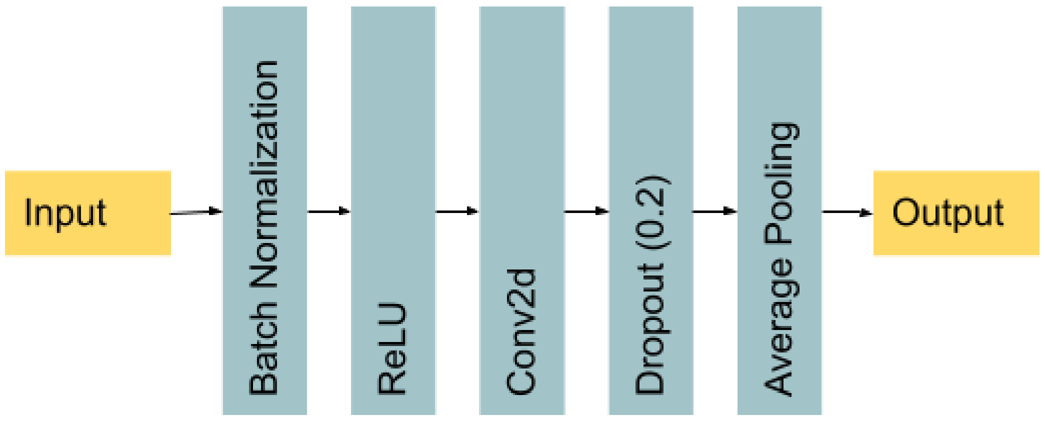


Fig. Transition Down Block

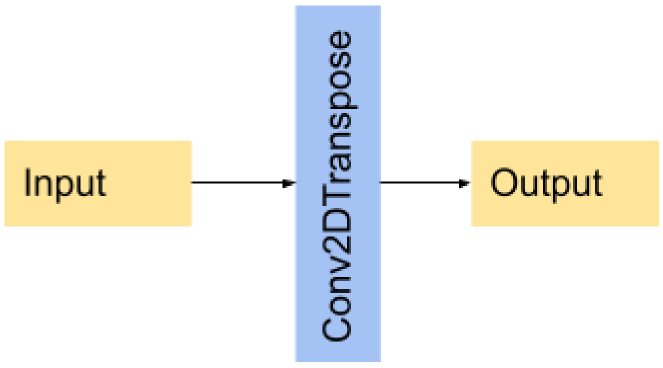


Fig. Transition Up Block

UNet as generator

In UNet (Ronneberger et al., 2015) as a generator, we followed the configuration of the generator set by pix2pix Gans. Here we started with an input tensor of the shape (256, 256, 3) to a Conv2D layer with feature maps of 64 with 0-padding enabled. This was passed through a leakyRelu layer with alpha value set to 0.2. After this it was passed through 6 blocks of conv2D layer followed by batch normalization and then through a LeakyRelu with alpha value of 0.2. In each of these block feature maps were increased from 64 to 128-256-512-512-512-512. After that in the bottleneck the output tensor from the previous layers was passed through a conv2d layer with feature map of 1024 and activation layer of Relu. The output of the bottleneck block was then forwarded to the Conv2dTranspose layer with 512 feature map; this was used to increase the spatial resolution of the tensor to generate the image by upscaling. Following the conv2dtranspose layer was batchnorm layer followed by dropout with probability of 0.5. The output of the same depth in the down path was concatenated with the output of this dropout and was passed to the activation layer of Relu. This up-scaling block was repeated 6 times with feature maps following the order of 512-512-512-256-128-64 and finally a Conv2dTranspose layer having the same feature maps as that of the input image. And finally passing the output of this layer through a sigmoid activation layer to finally get our desired output shape.

All these layers had kernel size of (3, 3) with 0-padding enabled and stride set to (2, 2). Due to this setup in the down path applying conv2d reduces the spatial resolution of the image to half while increasing the feature maps and on the up path applying the conv2dTranspose doubles the spatial resolution of the image tensor. As the size is upscaled/downscaled by factor of 2 it enables us to concatenate the feature maps from the down path to up path where spatial resolution is the same. Due to these concatenations or the skip connections information of the feature maps from the initial layers is passed to the later layer hence making sure overfitting does not take place and losses are back propagated to all layers with almost equal weights.

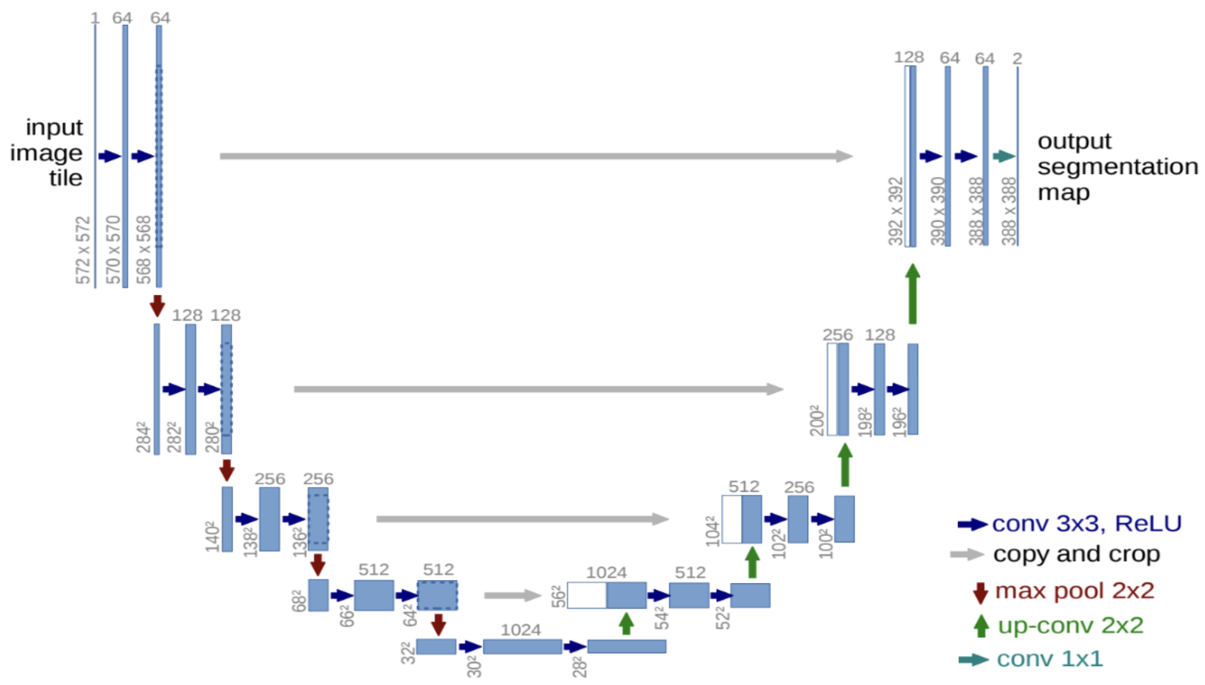


Fig . U-net architecture (example for 32x32 pixels in the lowest resolution). Each blue box corresponds to a multi-channel feature map. The number of channels is denoted on top of the box. The x-y-size is provided at the lower left edge of the box. White boxes represent copied feature maps. The arrows denote the different operations. (Ronneberger et al., 2015)

Patch Discriminator in GANs

The discriminator in a GAN is simply a classifier (Isola et al., 2016). It tries to distinguish real data from the data created by the generator. It could use any network architecture appropriate to the type of data it's classifying. We are using the Patch Discriminator network. This network performs patch-wise comparison of the target and generated images, rather than pixel wise comparison.

Like Unet (Ronneberger et al., 2015) as generator, we have started with an input tensor of the shape (256, 256, 3) to a Conv2D layer with feature maps of 64.This was passed through a leakyRelu layer with alpha value set to 0.2. After this it was passed through 4 blocks of conv2D layer followed by batch normalization and then through a LeakyRelu with alpha value of 0.2 and in the last block the output image from 5th block is convolved through 1*1 filter with sigmoid activation function.

In the starting 5 blocks, the size of feature maps were increased from 64 to 128-256-512-512 but in the 6th block it decreased to one for predicting if the image is real or fake.

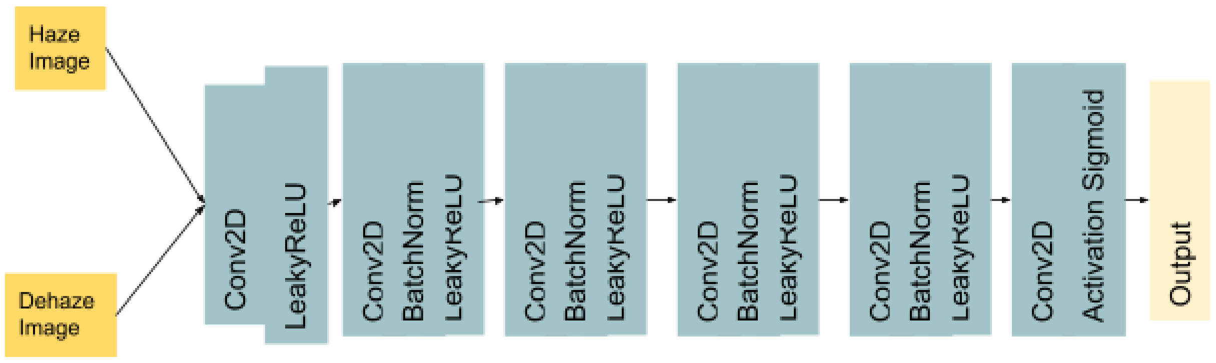


Fig. Discriminator of the proposed models. It takes in an input of an hazy image concatenated with the ground truth or the generated image. It outputs a 16x16 matrix, which is used to test if the image is real or fake.

As mentioned in the above figure the discriminator takes 2 input images for training which comes from 2 sources:

- Real data instances, such as real pictures of people. The discriminator uses these instances as positive examples during training.
- Fake data instances created by the generator. The discriminator uses these instances as negative examples during training.

Training Data

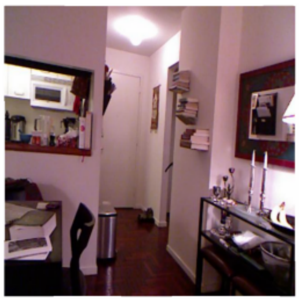
To create a realistic haze dataset, we need depth information. Since it is difficult to find a large, realistic haze/ground-truth image pair dataset, it was synthetically created and named as NYU Depth dataset (Silberman et al., 2012).

The NYU data set is composed of video sequences from a variety of indoor scenes as recorded by both the RGB and Depth cameras from the Microsoft Kinect. It features:

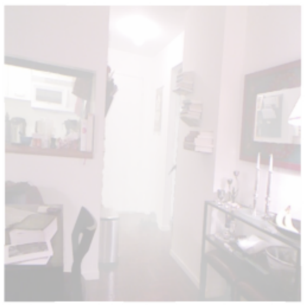
- 1449 densely labeled pairs of aligned RGB and depth images
- 464 new scenes taken from 3 cities

Few samples from the dataset are given below

Image from NYU dataset (Silberman et al., 2012)



Generated Hazed Image from the dataset (more information in implementation details part)



Implementation details

The input for the training of GANs (Goodfellow et al., 2014) is the dataset of hazed and unhazed images pairs. These pairs have the shape of (256, 256, 3) each. The generator takes in a hazed image and tries to create a dehazed image. Now this dehaze image from the generator along with the haze image makes a pair for the discriminator to take in. Discriminator is trained with a haze image and ground truth image pair along with a haze image and generated dehaze image from the generator. The labels for the ground truth image is set to 1 while for the generated image it is set to 0. This information is fed to the discriminator which tries to classify the given pair as real or fake/generated. The loss from the discriminator is passed back into the discriminator for changing weights. While for the generator loss from generator and loss from the ground truth image with respect to the generated image is taken in the ratio of 1:100, and we took a batch size equal to 1 as stated in pix2pix gans as this allows the model to converge faster. For training we train the model alternately once the generator is trained at that time discriminator is set to untrainable and vice versa for every step.

We trained our model with U-Net (Ronneberger et al., 2015) once and another time with Tiramisu (Jégou et al., 2016) as the generator. It was done for [1, 5, 10, 20] epochs and the results are discussed below.

For recreating these results

First run the data_download.py file this will download a mat file from the NYU Depth dataset (Silberman et al., 2012). *After this run the haze_image_generation.py* file in which the mat file is loaded and original images are extracted in folder named image and using the formula:

$I(x) = J(x)T(x) + A(x)(1 - T(x))$

where, $I(x)$ is the generated hazed image, $J(x)$ is the haze-free counterpart, $T(x)$ is the transmission map and $A(x)$ is global atmospheric light. For our convenience we have kept global atmospheric light fact to 1. For transmission map we used the formula:

$T(x) = e^{-\beta d(x)}$

where β being the attenuation coefficient for our convenience we have taken this as 1 and $d(x)$ being the scene depth of the image.

So more depth leads to more distortion leading to more hazed parts in depth and low haze distortion closer to the camera. By this we generated the haze image from the mat file as it contains the depth knowledge as well. This will create a new folder named hazed_image in which all the hazed images are placed. For making sure the pair of images with haze images is not mismatched we numbered them the same in both the files by enumerating in 5 digit places.

Now for training the model we need to run train.py where gans.py is a dependent file.

When we run the train.py file it will ask you if you want to train with UNet as a generator model or Tiramisu as a generated model. After selecting that it will ask for the number of epochs to be trained on. Now this program will load the images and hazed images from the generated folder, resize them in shape of (256, 256, 3) and then are passed through the model. After the model is trained it will store the model with the name g_model.h5 in the same directory.

Now we can *run the predict.py file which takes in argument a haze image and the model's saved file* and it will generate an unhazed image from the given hazed image using the model we are passing as argument. Now the generated dehaze image is in the shape of (256, 256, 3) and is stored in the same directory with name out.jpg

The *accuracy.py file is used to generate the performance measures from the generated image with respect to the ground truth image.* It takes in two arguments one is the generated image and another is the ground truth image from which we need to compare the generated image.

Click here to view and download pretrained model and created dataset along with the architecture's plots.

Results and Discussions

Performance Metrics

Haze removal performance can be evaluated on several factors, among them, two of the most frequently used factors are SSIM and PSNR.

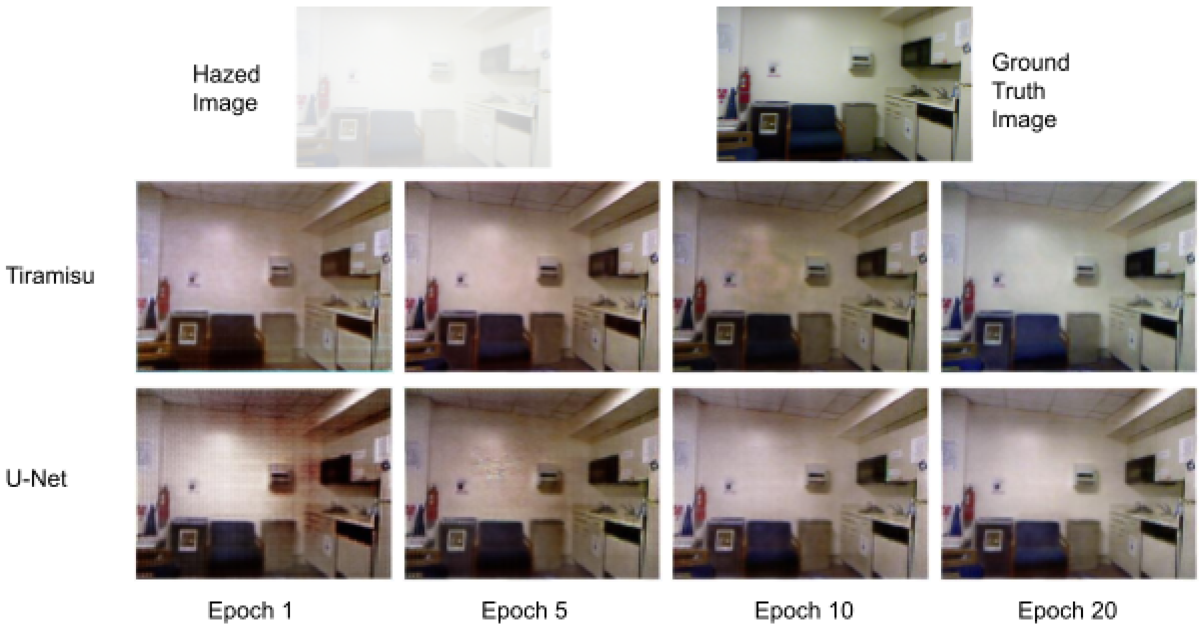
Structural Similarity Index Measure

- Structural Similarity Index Measure (SSIM), measures how similar two images are.
- Its range is from 0-1
- Two identical images will have a SSIM of 1

Peak Signal to Noise Ratio

- Peak Signal to Noise Ratio (PSNR) measures the ability of the algorithm to remove noise from a noisy image.
- Two identical images will have a PSNR value of infinity.
- For measuring haze removal capability, a higher PSNR value indicates better performance.

Below is the generated image from the image in the dataset used for training and performance metrics values



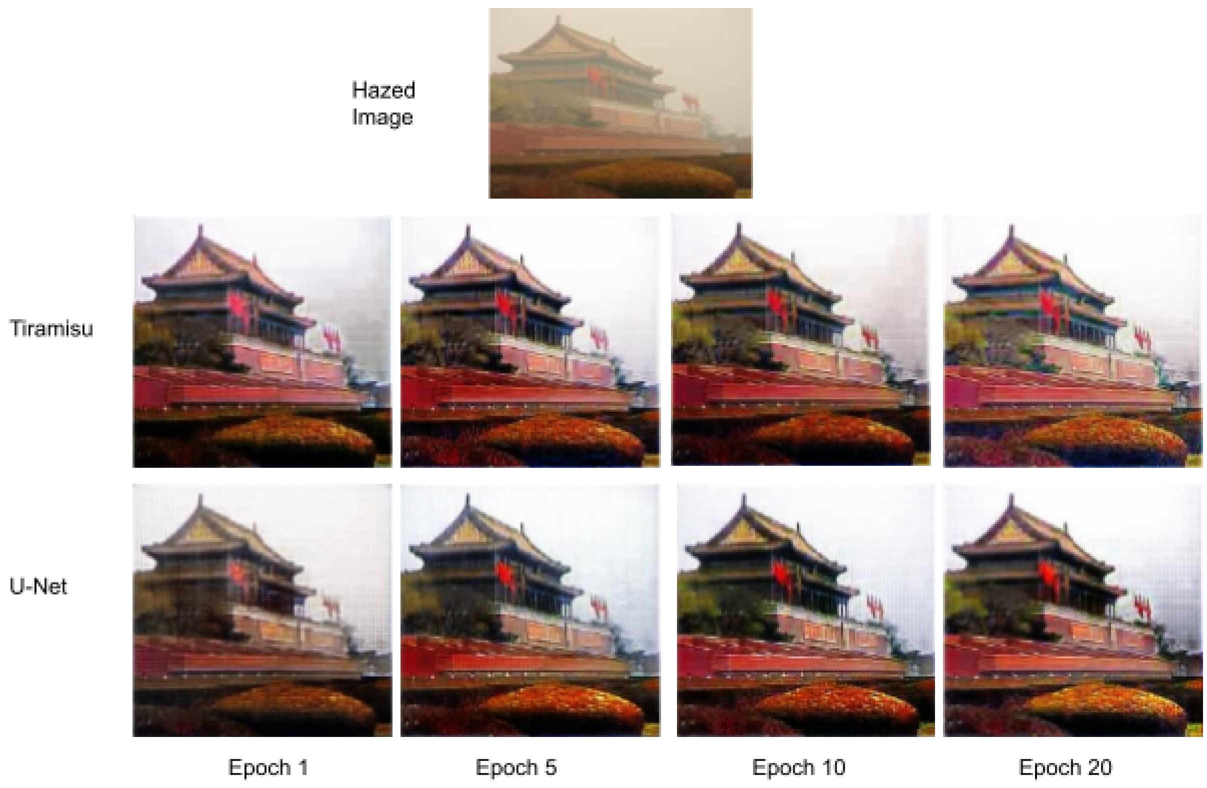
SSIM and PSNR values for UNet as generator

U-Net	**epoch 1**	**epoch 5**	**epoch 10**	**epoch 20**
SSIM	0.6385	0.7152	0.8320	0.8566
PSNR	18.1459	17.2186	23.4061	25.0133

SSIM and PSNR values for Tiramisu as generator

Tiramisu	**epoch 1**	**epoch 5**	**epoch 10**	**epoch 20**
SSIM	0.7389	0.8489	0.7954	0.8746
PSNR	20.6417	22.5325	16.651	25.0761

Below is the generated image of a image which is not dataset used for training



Team Member Names and Contribution

Tasks	Akshay Jain	Sarvesh Shroff
Data Collection/Creation	60	40
Tiramisu as generator	30	70
UNet as generator	70	30
Patch Discriminator in GANs	50	50
Integrating Generator and Discriminator	50	50
Testing and Validation	50	50
Report and Presentation	50	50

References

Goodfellow, I. J., Pouget-Abadie, J., Mirza, M., Xu, B., Warde-Farley, D., Ozair, S., Courville, A., & Bengio, Y. (2014, June 10). *Generative Adversarial Networks*. arXiv. https://arxiv.org/abs/1406.2661

Huang, G., Liu, Z., Maaten, L. v. d., & Weinberger, K. Q. (2016, August 25). *Densely Connected Convolutional Networks*. arXiv. https://arxiv.org/abs/1608.06993

Isola, P., Zhu, J.-Y., Zhou, T., & Efros, A. A. (2016, November 21). *Image-to-Image Translation with Conditional Adversarial Networks*. arXiv. https://arxiv.org/abs/1611.07004

Jégou, S., Drozdzal, M., Vazquez, D., Romero, A., & Bengio, Y. (2016, November 28). *The One Hundred Layers Tiramisu: Fully Convolutional DenseNets for Semantic Segmentation*. arXiv. https://arxiv.org/abs/1611.09326

Raj, B., & N, V. (2018, October 22). *Single Image Haze Removal using a Generative Adversarial Network*. arXiv. https://arxiv.org/abs/1810.09479

Ronneberger, O., Fischer, P., & Brox, T. (2015, May 18). *U-Net: Convolutional Networks for Biomedical Image Segmentation*. arXiv. https://arxiv.org/abs/1505.04597

Silberman, N., Hoiem, D., Kohli, P., & Fergus, R. (2012). *Indoor Segmentation and Support Inference from RGBD Images*. ECCV. https://cs.nyu.edu/~silberman/datasets/nyu_depth_v2.html