Low-light Image Enhancement

By Alif Jakir, Ruijie Wang, Jiaen Wang, James Au

Index

1. Datasets
2. Data Processing
3. Low Light Enhancement(LLE) and Its Neural Network Architecture
4. Super Resolution and Its Neural Network Archtecture
5. Metric Tracking
6. Results
7. Conclusions

Low-light Images

Poor visibility, low contrast, high noise level, lack of useful information

Low light

Normal light

Datasets

- LOL_Dataset
- SR_Dataset

LOL_Datasets

Low-light and normal-lights images in pairs for machine learning: Total 500 images, with pixel size 600*400

W. Y. J. L. Chen Wei, Wenjing Wang, “Deep retinex decomposition for low-light enhancement,”

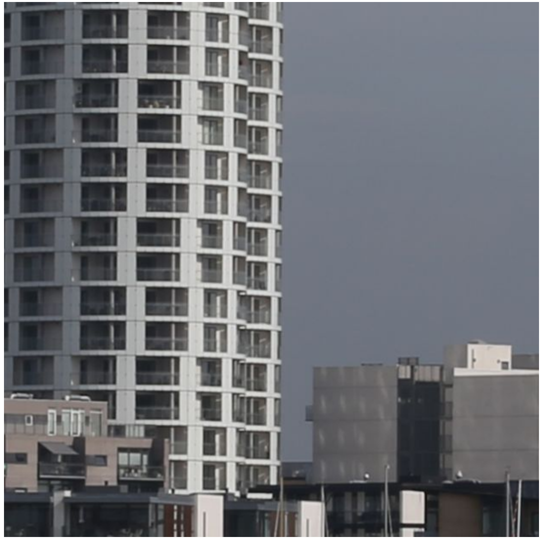
Datasets - Strategy

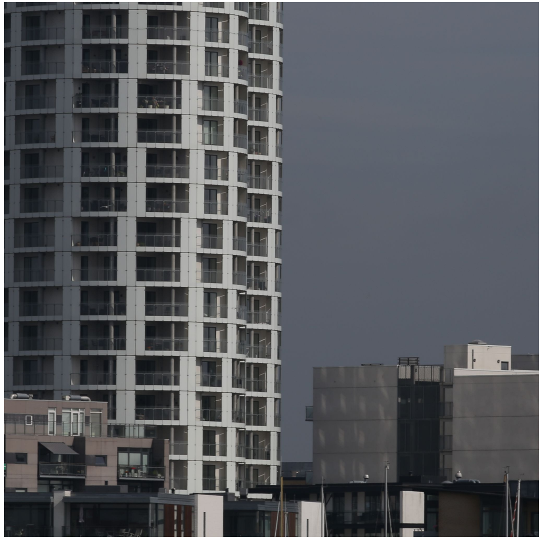
Data split strategy:

- training (70%)
- validation(20%)
- test(10%)

SR_Dataset

Low-resolution and high-resolution images in pairs for machine learning: Total 850 pages, with pixel size 625*625 of input, 2500*2500 of output





A. Aakerberg, K. Nasrollahi, and T. B. Moeslund, “RELLISUR: A Real Low-Light Image Super-Resolution Dataset,”

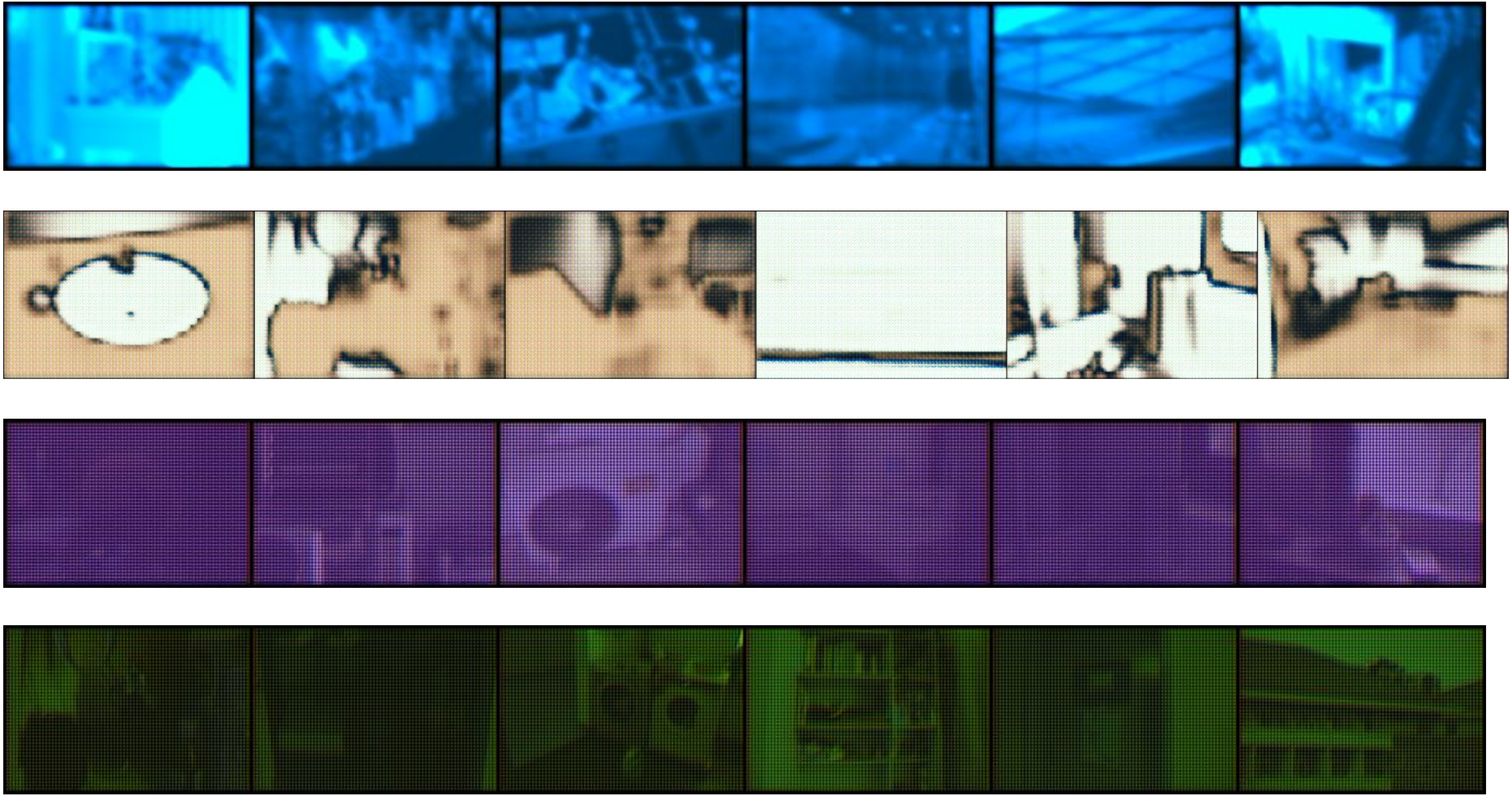
Datasets - Processing

Choice of color space: YCrCb or RGB

Firstly, We adapted RGB with three color channels, using the raw images.

However, pictures are of random colors.

The reasons for the problem: primitive loss function and mismatch of color channels between generator and discriminator

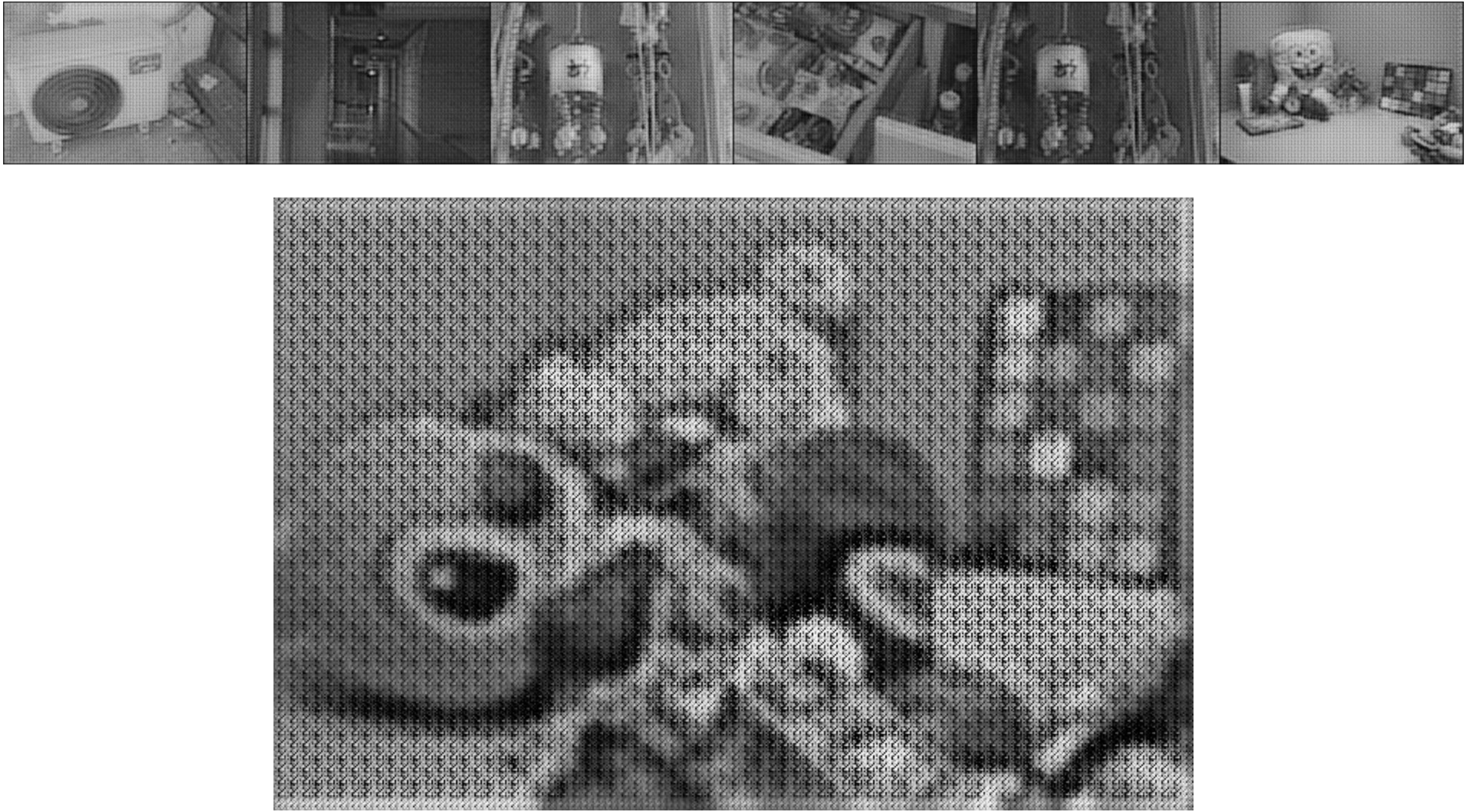


Datasets - Processing

Afterwards, we adapted YCrCb color space instead. In the YCrCb space, Y is of the greatest importance. Thus, it is easier to process the channels.

However, it is complicated to maintain the raw colors.

Spots, boxes, and lines still exist in the images.



Datasets- Processing

Data Augumentation

Increase the diversity of our data - using rotations, flips, translations, scaling, brightness adjustments, and more.

To create new training examples that are variations of the original data, making the model more robust and capable of generalizing better to unseen data.

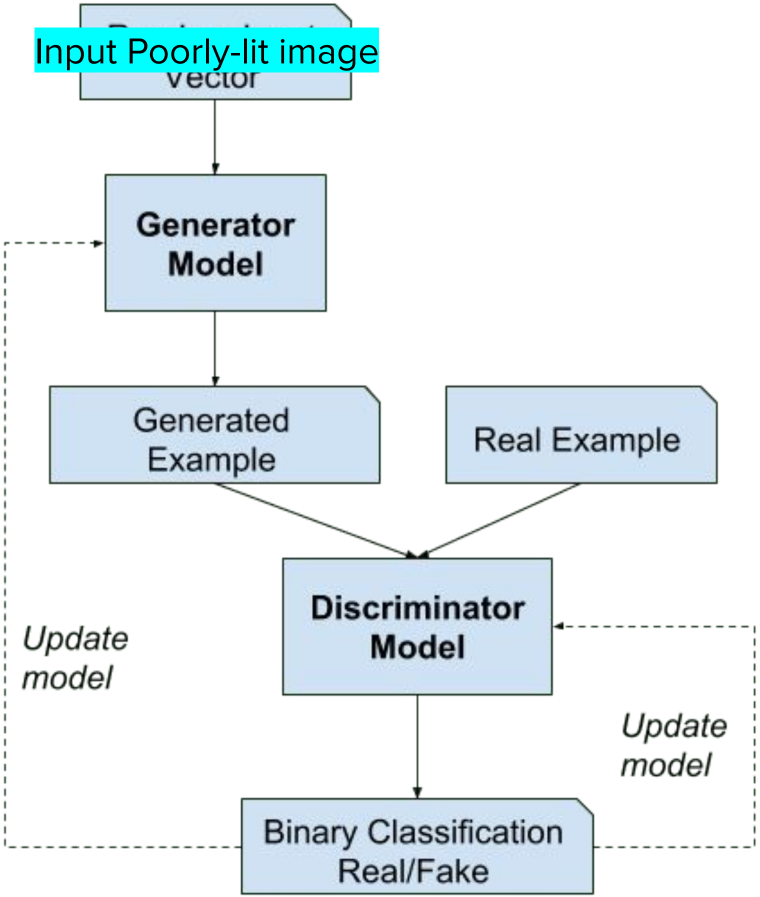
Get the Images Well-lit: Low Light Enhancement(LLE)

We adapt a GAN network:

Generator: learning from the pairs of images and transform the low-light images into brighter and contrast-enhanced versions.

Discriminator: learning to differentiate between the well-lit and the generated processed images.

Architecture



Define Transformation

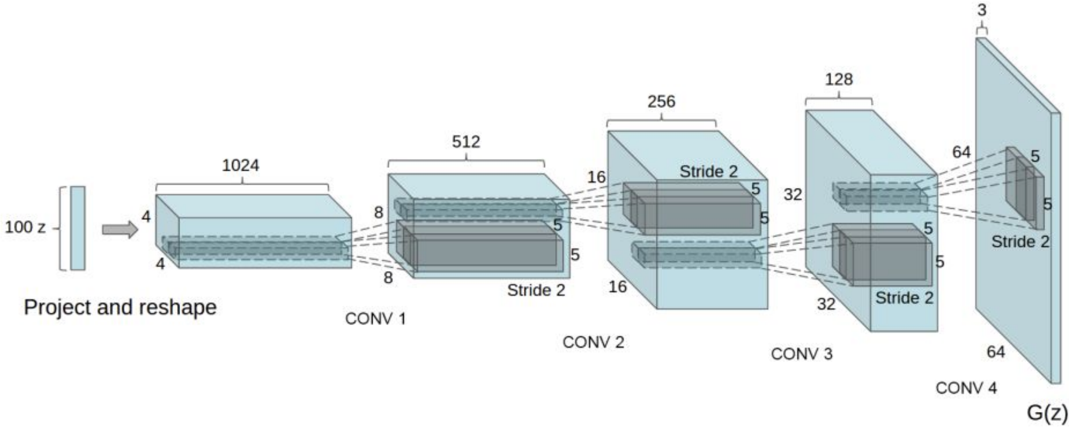
- Convert image to tensor
- Data augumentation adds more data
    - Horizontal/Vertical Flip
    - Rotation
    - Random color jitter(brightness, contrast, saturation, and hue adjustments.)
    - Takes more time to converge to generalization during training

Generator Structure - 1

- U-Net convolutional neural network (CNN) architecture.
    - encoder: downsampling, capturing hierarchical feature
    - decoder: upsampling to reconstruct the spatial resolution
    - skip connections: preserve fine-grained details
    - Apply sigmoid to output (maps to (0,1)

Generator Structure - 2

The Generator would undergo convolution to turn any input image into a two-dimensional image matrix, and then processing it through Convolution and transpose, with each convolution kernel cycle the width and height are doubled while depth are being halved.



Nathan Inkawhich

https://pytorch.org/tutorials/beginner/dcgan_faces_tutorial.html#results

Generator Structure - 3

Limitations with the generator architecture

```
# Generator (U-Net-like)
class Generator(nn.Module):
    def __init__(self, in_channels=3, out_channels=3):
        super().__init__()

        # Encoder
        self.conv1 = nn.Conv2d(in_channels, 64, kernel_size=3, padding=1)
        self.conv2 = nn.Conv2d(64, 128, kernel_size=3, padding=1)
        self.conv3 = nn.Conv2d(128, 256, kernel_size=3, padding=1)

        # Decoder
        self.tconv1 = nn.ConvTranspose2d(256, 128, kernel_size=3, padding=1)
        self.tconv2 = nn.ConvTranspose2d(128, 64, kernel_size=3, padding=1)
        self.tconv3 = nn.ConvTranspose2d(64, out_channels, kernel_size=3, padding=1)

    def forward(self, x):
        # Encoder
        x1 = F.relu(self.conv1(x))
        x2 = F.relu(self.conv2(x1))
        x3 = F.relu(self.conv3(x2))

        # Decoder with skip connections
        x = F.relu(self.tconv1(x3))
        x = F.relu(self.tconv2(x + x2))
        x = self.tconv3(x + x1)
        # Apply sigmoid to output
        x = torch.sigmoid(x)
        return x
```

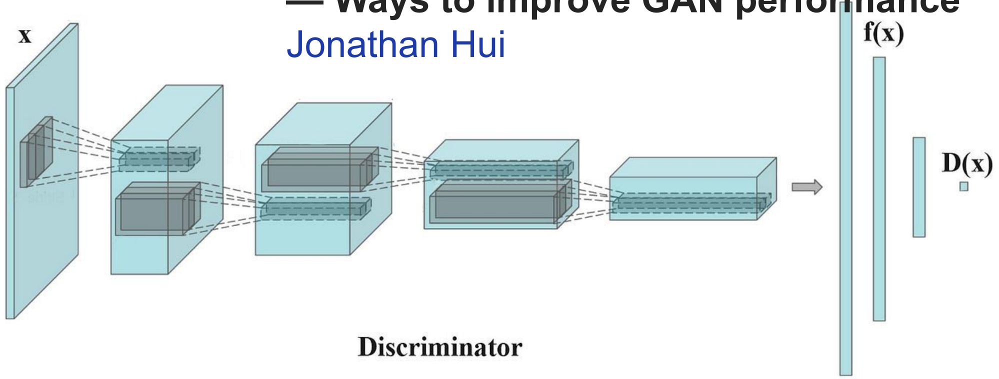
Discriminator Structure - 1

- Distinguish between real and generated images
  - Five convolutional layers
    - The five layers of the Discriminator form a hierarchical feature extraction process.
    - They progressively downsample the input image, reducing its spatial dimensions while increasing the number of feature maps.
  - Activatation Function
    - The LeakyReLU activation function is applied after the first four Conv2d layers to introduce non-linearity and enhance feature learning.

Discriminator Structure - 2

The discriminator would do the exact opposite to the Generator. The discriminator receives a 2D image matrix, then it would pass the image though a series of convolution, With each cycle, the image shrink by half in Width and Height, but increase in depth. The cycle repeats itself until it was made into a one-dimensional number array by the last function of sigmoid.

https://towardsdatascience.com/gan-ways-to-improve-gan-performance-acf37f9f59b **GAN — Ways to improve GAN performance**
Jonathan Hui



Discriminator Structure - 3

```
class Discriminator(nn.Module):
    def __init__(self, in_channels=3):
        super().__init__()

        self.main = nn.Sequential(
            # Resize input to consistent size
            nn.AdaptiveAvgPool2d((64, 64)),
            # input is (in_channels) x 64 x 64
            nn.Conv2d(in_channels, 64, 4, 2, 1, bias=False),
            nn.LeakyReLU(0.2, inplace=True),
            # state size. (64) x 32 x 32
            nn.Conv2d(64, 128, 4, 2, 1, bias=False),
            nn.BatchNorm2d(128),
            nn.LeakyReLU(0.2, inplace=True),
            # state size. (128) x 16 x 16
            nn.Conv2d(128, 256, 4, 2, 1, bias=False),
            nn.BatchNorm2d(256),
            nn.LeakyReLU(0.2, inplace=True),
            # state size. (256) x 8 x 8
            nn.Conv2d(256, 512, 4, 2, 1, bias=False),
            nn.BatchNorm2d(512),
            nn.LeakyReLU(0.2, inplace=True),
            # state size. (512) x 4 x 4
            nn.Conv2d(512, 3, 4, 1, 0, bias=False),
            nn.Sigmoid()
        )

    def forward(self, input):
        return self.main(input)
```

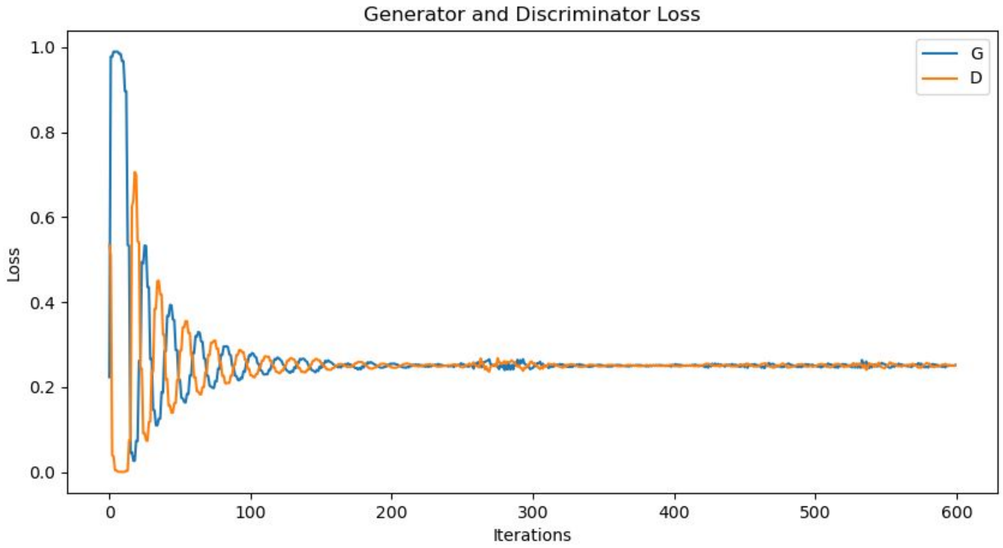
Loss Function

There are many loss functions to choose.

- Mean Squared Error (MSE) Loss: Used for regression tasks, where the output is a continuous value, and the loss measures the mean squared difference between predicted and target values.
- Mean Absolute Error (MAE) Loss: used for regression tasks, which measures the mean absolute difference between predicted and target values.
- Binary Cross Entropy Loss (BCE Loss): Used for binary classification tasks, where the output is a single probability value between 0 and 1.

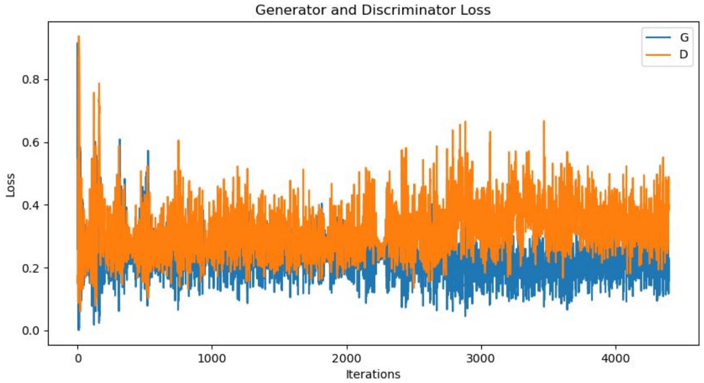
Results for Different Loss Function

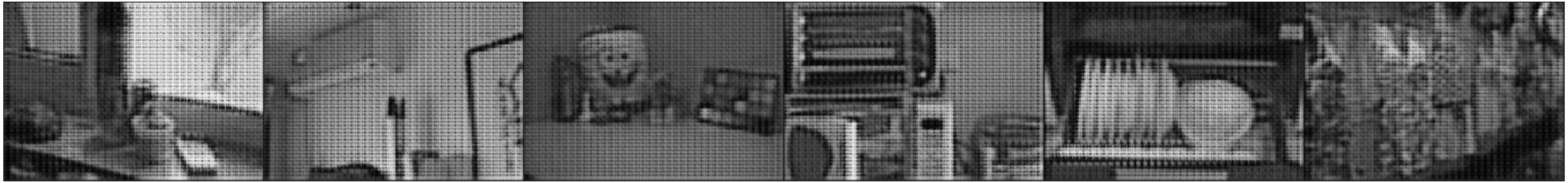
MSE Loss: The MSE loss function is the first one to use, but after serval training epochs, the output will overlearn and result in pure white pictures.



Results for Different Loss Function

- Binary Cross Entropy Loss (BCE Loss): this is what we used afterwards, but the loss function keeps oscillating and does not converge at all, the output is not good as well.

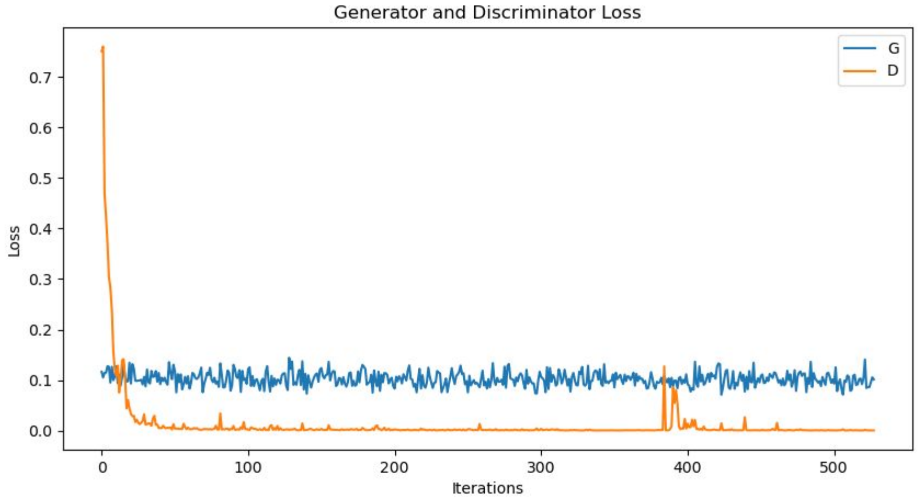




Combination of Multiple Loss Functions?

We use the combination of Custom_loss = mae_loss + beta * gan_loss

The Beta as a weight hyperparameter that controls the trade-off between the MAE loss and the GAN loss.(=0.01)



BENEFITS

GANs alone may produce visually appealing images, but they might lack fine-grained details and sharpness.

MAE loss penalizes the pixel-wise differences between the generated and target (real) images.

By properly combining them together, we can control how realistic and how close it is to the reference picutre.

More training needed to improve the efficiency and outcome

Loss Function

```
def loss_fn(real_outputs, fake_outputs, real_labels, fake_labels, beta):
    # MAE loss
    mae_loss = criterion_mae(real_outputs, fake_outputs)

    # Resize fake_outputs tensor to match the size of real_labels tensor
    fake_outputs_resized = F.interpolate(fake_outputs, size=real_labels.size()[2:], mode='bilinear')

    # GAN loss - calculate separately for each channel and then average
    gan_loss = criterion_gan(fake_outputs_resized, real_labels)
    gan_loss = gan_loss.mean()

    # Combined loss
    loss = mae_loss + beta * gan_loss

    return loss, mae_loss, gan_loss
```

Optimization Algorithm

We're using the Adam (Adaptive Moment Estimation) optimization algorithm for both the generator and the discriminator.

Set the learning rate to be 0.0001. A smaller learning rate results in slower but more precise convergence, while a larger learning rate may lead to faster but less stable convergence.

We're also trying different value of the learning rate together with the batch size to improve the efficiency and stability of the NN.

Training Processes

- Epoch Loop: each epoch represents one pass through the entire dataset. 100 Epoches for now.
- Batch Loop: each batch contains a fixed number of low-light and corresponding well-lite images. batch size = 8
- Discriminator Training
- Generator Training.
- Backpropagation and Optimization

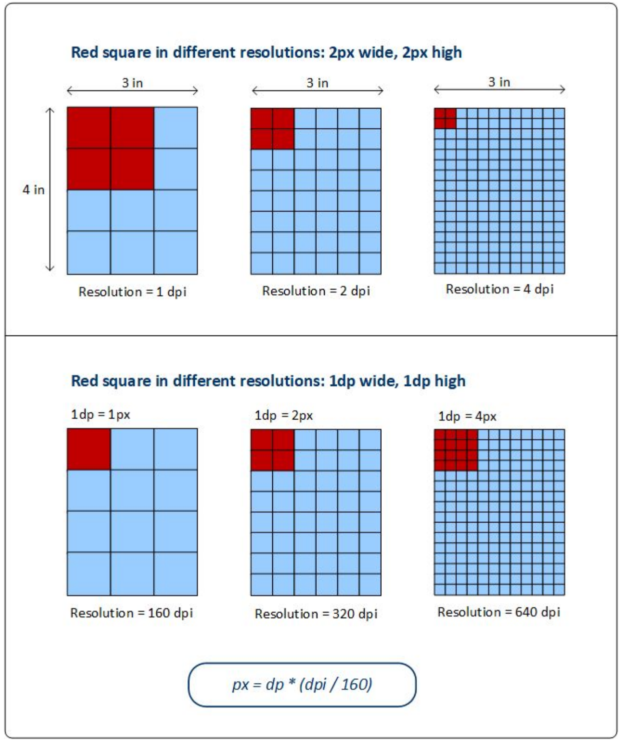
Training Processes

- Discriminator Training:
  - Real well-lit images input. label =1
  - Fake well-lit images input (generated by The Generator) . label = 0
  - Discriminator Loss: both real loss and fake loss. The goal is to minimize this loss so that the discriminator can correctly distinguish real from fake images.
- Generator Training:
  - Generate Fake output .
  - Two Loss Functions combined for the generator's loss : mae_loss and gan_loss. The mae_loss is the mean absolute error between the generated well-lit images and the corresponding real well-lit images. The gan_loss is the loss related to the discriminator's output for the generated images.
- Backpropagation and Optimization
  - backpropagation is performed to update the model parameters. The Adam optimizer is used to optimize the model parameters, and gradients are accumulated over several batches

Super Resolution

Aims:
In the result image from the low light enhancement, the image have in low in resolution, which make detail viewing by users being difficult

Super resolution is the process to upsampling the image from low resolution image. Which makes the resultant image contain smoother curves which make the image content more recognisable.



Training process for Super Resolution

For super resolution, we have designed a training process that requires a real image along with its lower resolution counterpart.

The Generator needs to train itself to convert the low resolution image back into the higher resolution original image.

The training process continues until the Generator successfully fools the Discriminator.

Setup

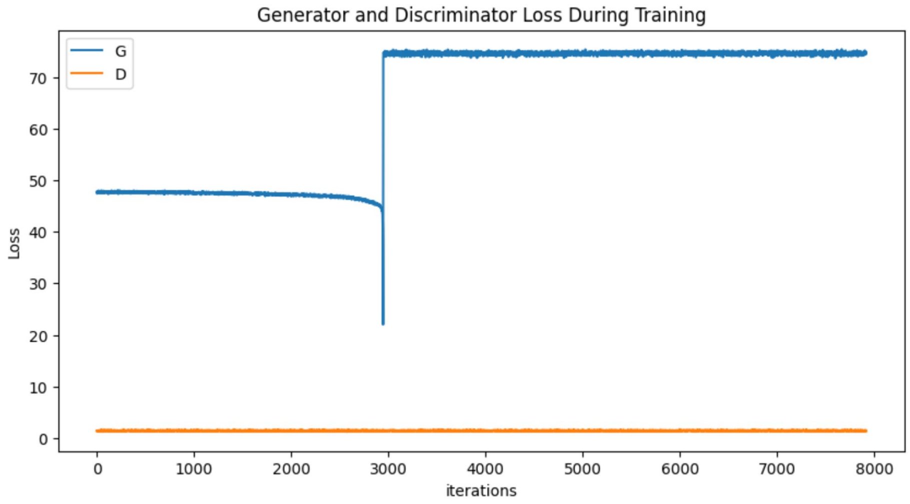
The training process consists of 10 epochs, and the learning rate is set to 0.0002

The generator features 3 convolution layer and

The discriminator have five convolution layer , 4 Relu layer,3 batchnorm layer and a Sigmoid layer at the last.

Attempt: Uneven Size train Data generated

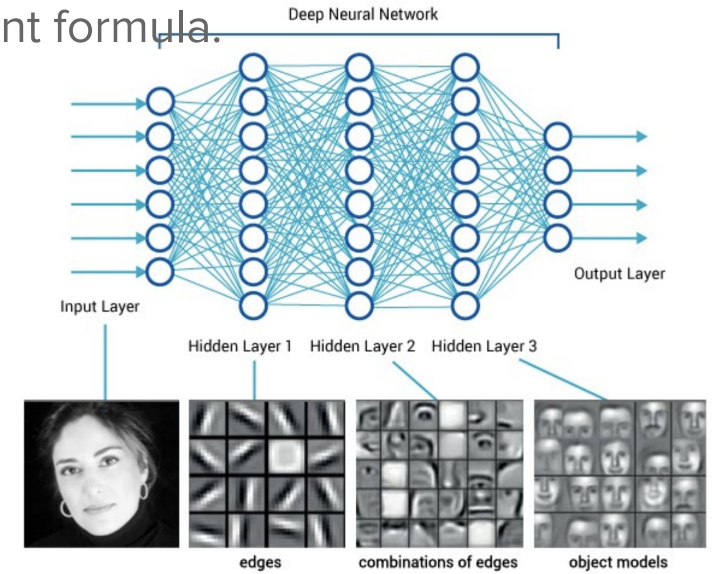
We have attempted double the size of the generated image versus the training data. This was executed by having a separate discriminator with an extra convolution data, while generator get an extra convolution cycle. This attempted failed since it would just alter the whole discriminator formula, render the whole algorithm to fail.



Why it doesn't work

For each layer of Convolution in Code, it would represent a entirely new layer of selection algorithm.

You cannot compare result generated with different formula.



Desmond Tsoi HKUST COMP2211 Course

Dataset problem

Sometimes, we might not have a set of image pairs of High resolution and low resolution.

If we don't have a specific low resolution dataset, we intentionally downsample any high resolution image to create a low resolution image for training the algorithm.

```
avg_pool = nn.AvgPool2d(kernel_size=Blur_size,
stride=Blur_size)
downsampled_tensor = avg_pool(real_images)
# Upsample the downsampled tensor using bilinear
interpolation
low_img = F.interpolate(downsampled_tensor,
scale_factor=Blur_size, mode='bilinear')
fake_images = generator(low_img)
```

Downsampling process

Downsampling the image have one golden rules: Do not alter the image size.

Therefore,After we conducted an average pooling function, we would use an interpolate function to scale up the downsampled image back into it’s original shape.

Interpolate scale up by replicating each pixels NxN

Data Handling for Colab and Pytorch

Google colab data upload are complicated compared to run locally

Google drive mounting technique

Pytorch Dataloader

```
dataset = ImageFolder(root=dataroot, transform=transform)
dataloader = DataLoader(dataset, batch_size=batch_size, shuffle=True)
# print(np.shape())# high_res
print(dataloader)
# Create the generator and discriminator networks, and define the loss functions and optimizers
generator = Generator().to(device)
discriminator = Discriminator().to(device)
criterion = nn.BCEWithLogitsLoss()
l1_loss = nn.L1Loss()
optimizer_g = optim.Adam(generator.parameters(), lr=learning_rate, betas=(0.5, 0.999))
optimizer_d = optim.Adam(discriminator.parameters(), lr=learning_rate, betas=(0.5, 0.999))

# Train the GAN
train(generator, discriminator, dataloader, optimizer_g, optimizer_d, criterion)


# Generate a high-resolution image from a low-resolution input image
# img_pth = "path/to/low_res_image.jpg"
img_pth = "/content/1_copy.png"
low_res = Image.open(img_pth).convert("RGB")
low_res_tensor = transform(low_res).unsqueeze(0).to(device)
high_res_tensor = generator(low_res_tensor).squeeze(0).cpu()
high_res = transforms.ToPILImage()(high_res_tensor)
# high_res.save("path/to/high_res_image.jpg")
high_res.save("/content/high_res_image.jpg")
```

```
# drive.mount('/content/gdrive')
drive.mount('/content/gdrive', force_remount=True)

    Mounted at /content/gdrive

!ls gdrive/MyDrive/'Computational Imaging Project - DTU 34269 - Low Light Super Resolution'

# !ls gdrive/MyDrive/'Computational Imaging Project - DTU 34269 - Low Light Super Resolution'/dataset3
!ls gdrive/MyDrive/'Computational Imaging Project - DTU 34269 - Low Light Super Resolution'/dataset3

# #path that contains folder you want to copy
# %cd gdrive/MyDrive/'Computational Imaging Project - DTU 34269 - Low Light Super Resolution'/
# # %cp -av YOUR_FOLDER NEW_FOLDER_COPY dataset3
# %cp -av dataset3 dataset3_cpy

!gdown --folder gdrive/MyDrive/'Computational Imaging Project - DTU 34269 - Low Light Super Resolution'/dataset3

!unzip gdrive/MyDrive/'Computational Imaging Project - DTU 34269 - Low Light Super Resolution'/Data.zip

# "path/to/dataset" /Users/jamesau/Library/CloudStorage/GoogleDrive-jamesau2810@gmail.com/.shortcut-targets-by-id/1D_3mIrLJdP9
# Root directory for dataset
# For new Dataset Computational Imaging Project - DTU 34269 - Low Light Super Resolution/dataset3/super_dataset
dataroot = "gdrive/MyDrive/Computational Imaging Project - DTU 34269 - Low Light Super Resolution/dataset3/super_dataset"
# Original Zipped Data
# dataroot = "Data"
```

ChatGPT usage

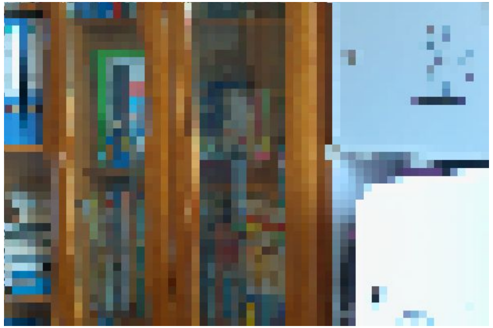
We have used it when we haven’t figure out direction, but we viewed other conventional CNN/GAN tutorial website also to understand coding better.

But for super-resolution, The discriminator from GPT was replaced since it was non-functional

Super Resolution Output

The trained superresolution model can then be used by inputting a low-resolution image into the Generator, which transforms it into the most probable high-resolution result determined by the Generator.

However, due to the complexity of iteration and training for superresolution, we have not been able to successfully convert a low-resolution image into an accurate output



Future Attempt for Super Resolution

We decided that we will need to upgrade the initial architecture to integrate the aspects of Enhanced Super-resolution generative adversarial network.

The architecture are slight different of having Residual Network instead of deep convolutional network

Plus they have entirely different Loss Calculation Method which we are still figuring it out.

https://www.geeksforgeeks.org/super-resolution-gan-sr
gan/

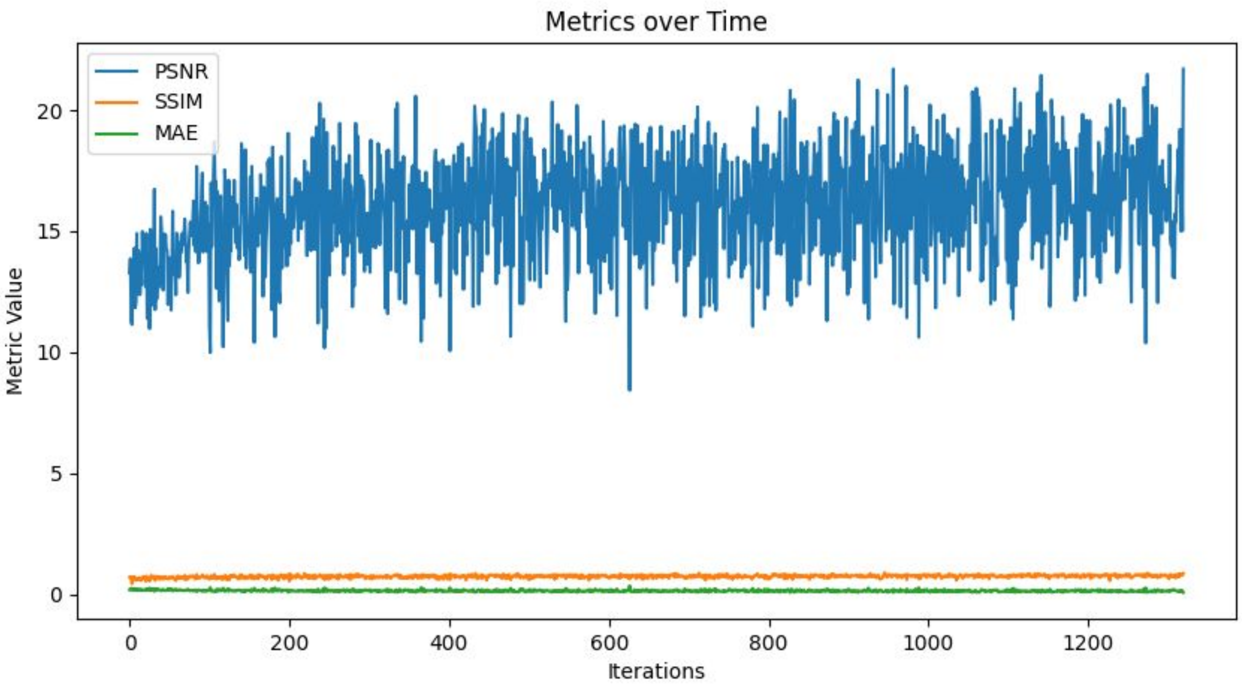
Residual Network

Residual Network are network which contain Skip connection which makes connection skipping possible once the specific Convolutional layers have been underperforming.

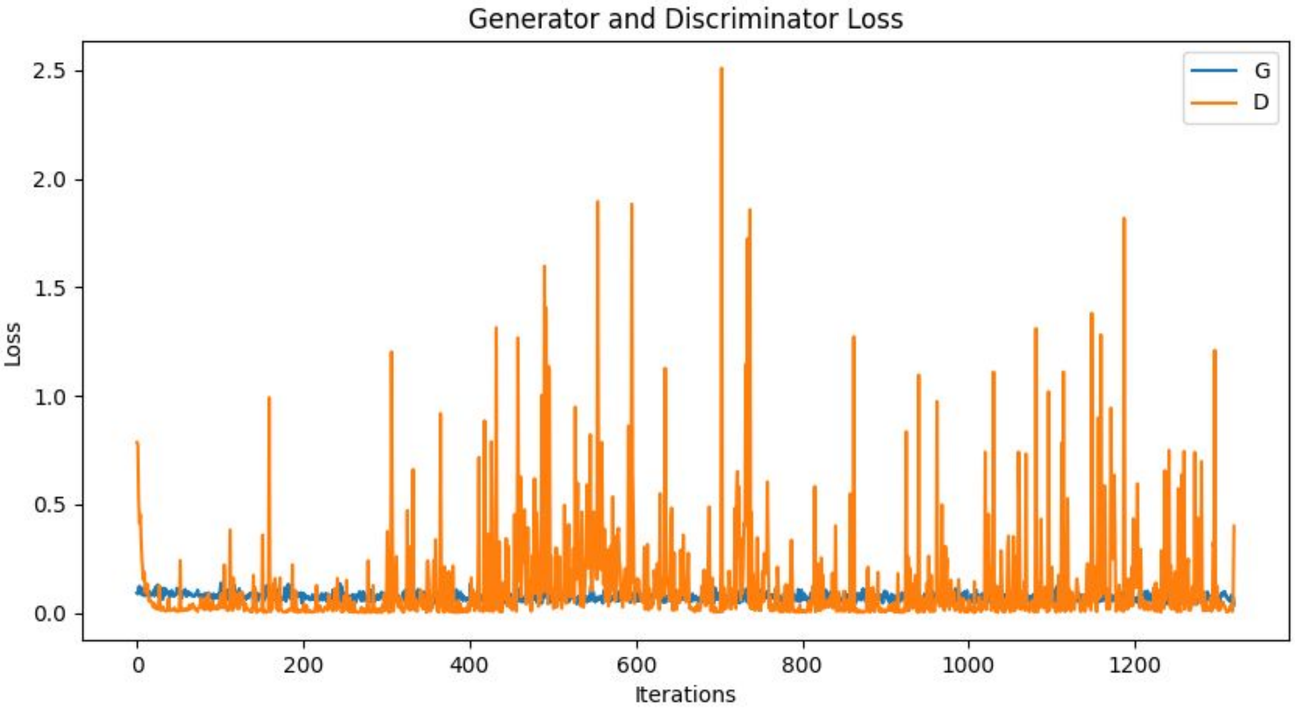
Metric Tracking

- We used three different metrics to evaluate the performance: PSNR,SSIM,MAE
  - PSNR:PSNR is a popular metric to assess the quality of images or signals by measuring the ratio of the maximum possible power of the signal to the power of the noise that affects it.
  - SSIM: It measures the perceived change in structural information, luminance, and contrast between the original and the generated images.
  - MAE: MAE is a regression metric used to evaluate the average absolute difference between the predicted and target values in a numerical prediction task.

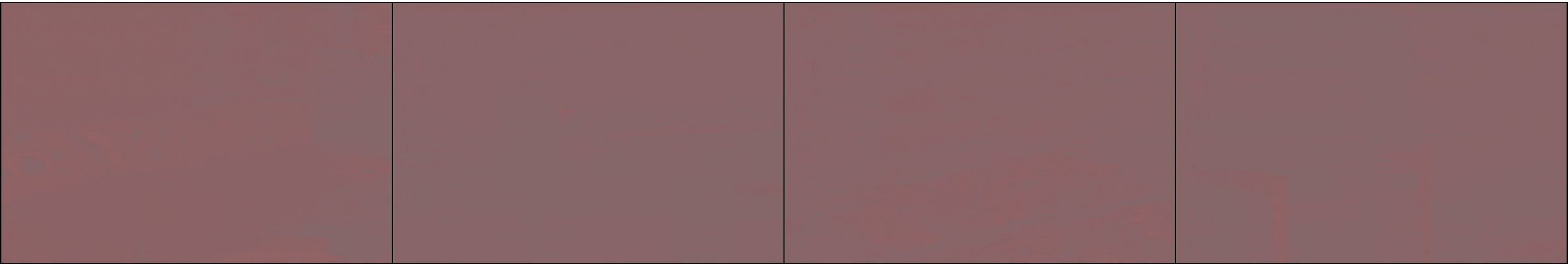
Metrics

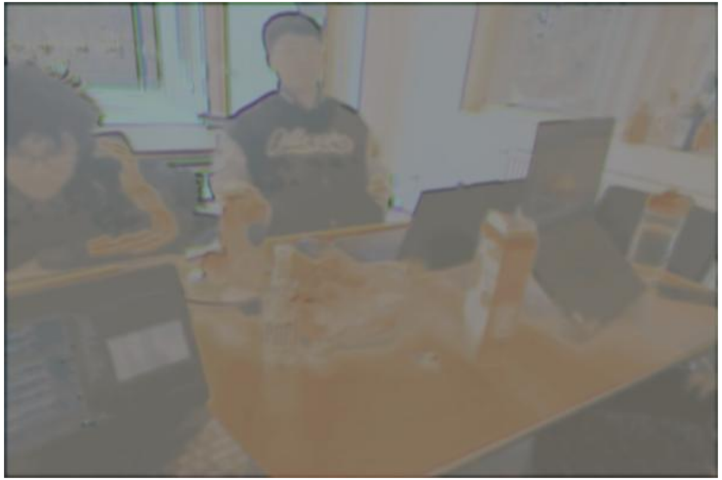

Metrics over Time
PSNR
SSIM
MAE
Metric Value
20
15
10
5
0
0
200
400
600
800
1000
1200
Iterations

Loss Function



Results





Low-lit input, ground truth, generated - RESULTS