HUST

ĐẠI HỌC BÁCH KHOA HÀ NỘI

HANOI UNIVERSITY OF SCIENCE AND TECHNOLOGY

ONE LOVE. ONE FUTURE.

Image Deblurring

ĐẠI HỌC
BÁCH KHOA HÀ NỘI
HANOI UNIVERSITY
OF SCIENCE AND TECHNOLOGY

Digital Image Processing

Image Deburring

Instructors: Tran Thi Thanh Hai
Le Thi Lan
Student 1: Pham Tung Lam - 20224321
Student 2: Bui Thi Ngoc Anh - 20224276

Table of contents

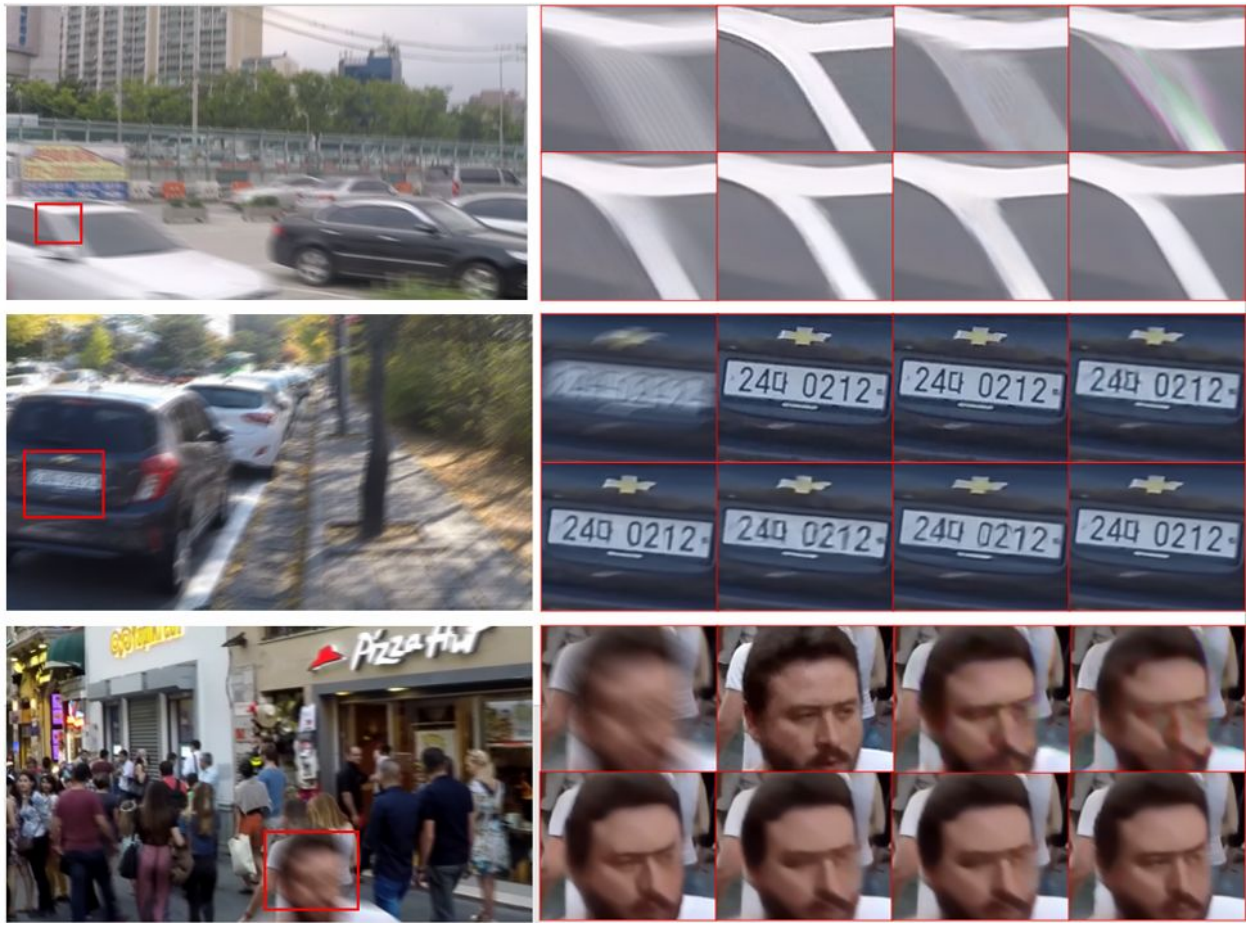
HUST

hust.edu.vn fb.com/dhbkhn

1. Introduction

1. Introduction

- Single image deblurring aims to restore a latent sharp image from a blurry one.



1. Introduction

- There are several methods for this work

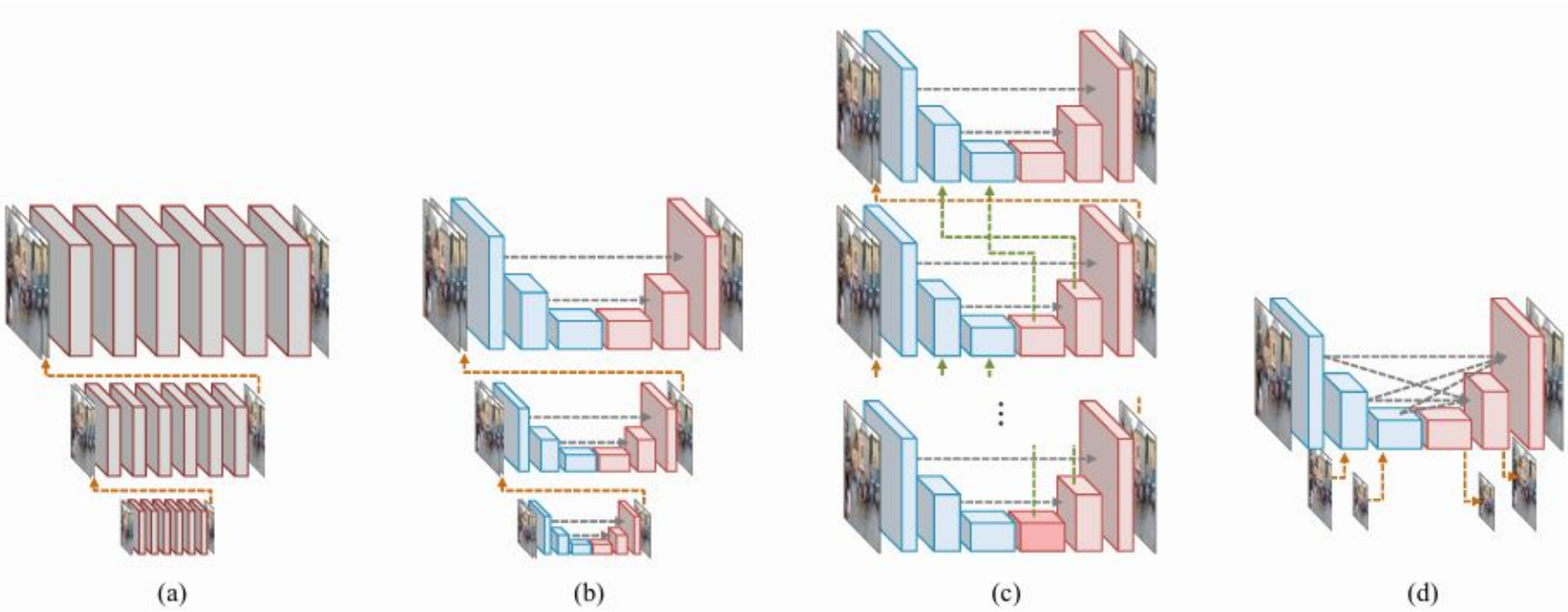


Figure 2. Comparison of coarse-to-fine image deblurring network architectures: (a) DeepDeblur, (b) PSS-NSC, (c) MT-RNN, and (d) proposed MIMO-UNet.

1. Introduction

- In this report, we learn about GANs and try to replicate its application to image deblurring.
- Because of the limitations of the device, we will build the network according to the most basic theory.

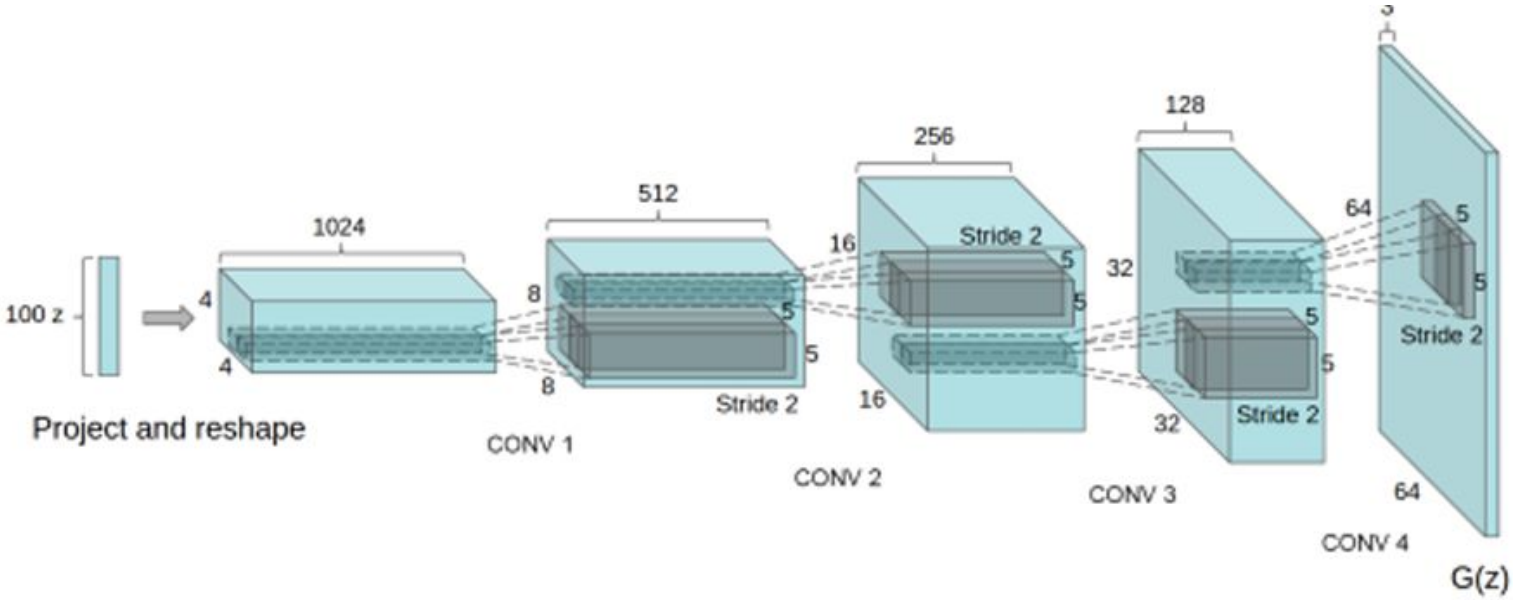


Figure 4. DCGANs - Generator architecture

HUST

hust.edu.vn fb.com/dhbkhn

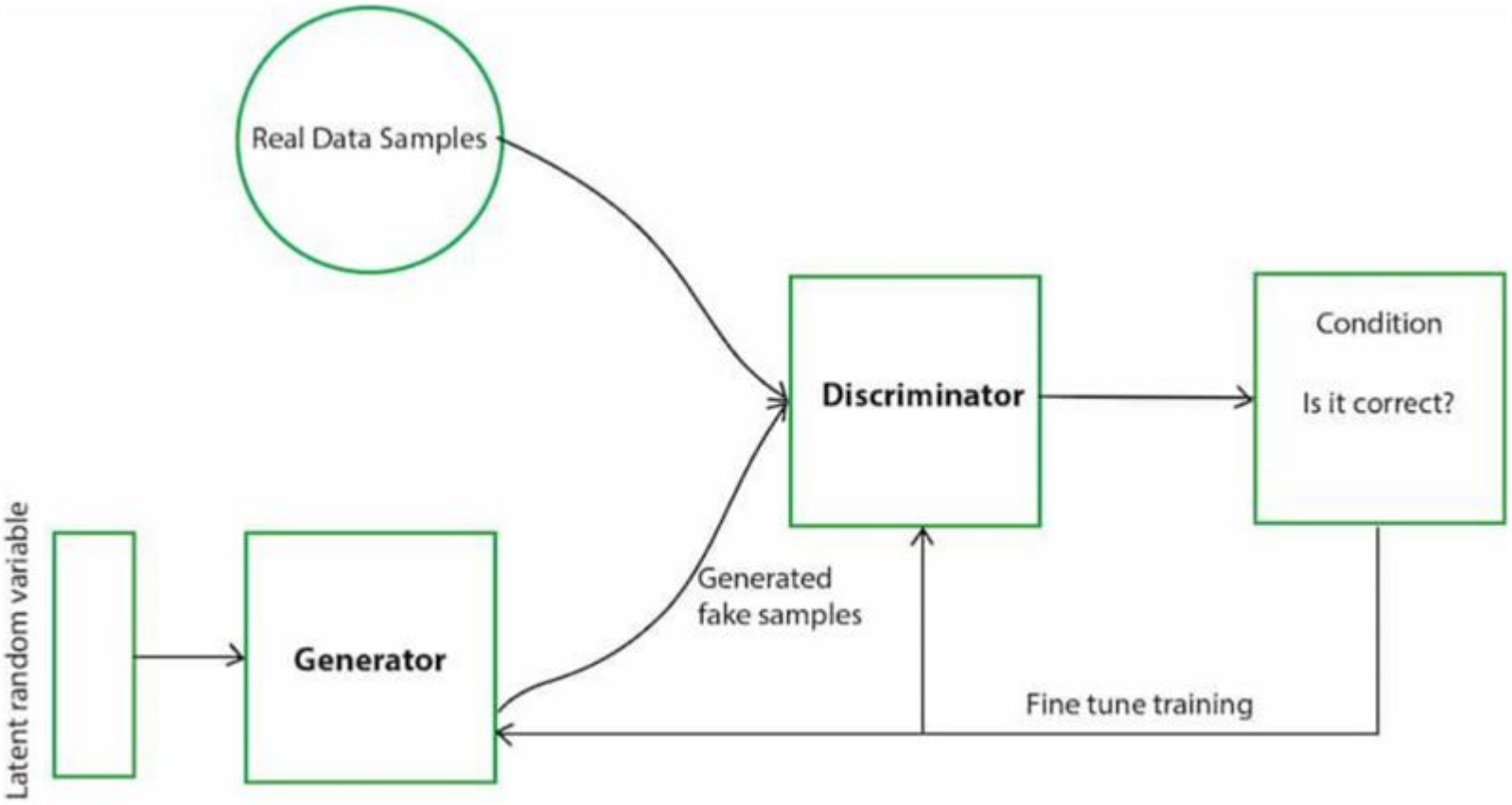
2. Proposed method

2.1. Generative Adversarial Network (GAN)

- **Generative Adversarial Networks (GANs)** are a powerful class of ***neural networks*** that are used for ***an unsupervised learning.***
- GANs are made up of two neural networks, a ***discriminator*** and a ***generator.***
- They use ***adversarial training*** to produce artificial data that is identical to actual data.
- The ***Generator*** attempts to ***fool*** the ***Discriminator,*** which is tasked with accurately distinguishing between ***produced*** and ***genuine data***, by producing ***random noise samples***.
- Through ***adversarial training***, these models engage in a ***competitive interplay*** until the ***generator*** becomes adept at creating ***realistic samples,*** fooling the ***discriminator*** approx imately ***half the time.***

2.2. How does a GAN works

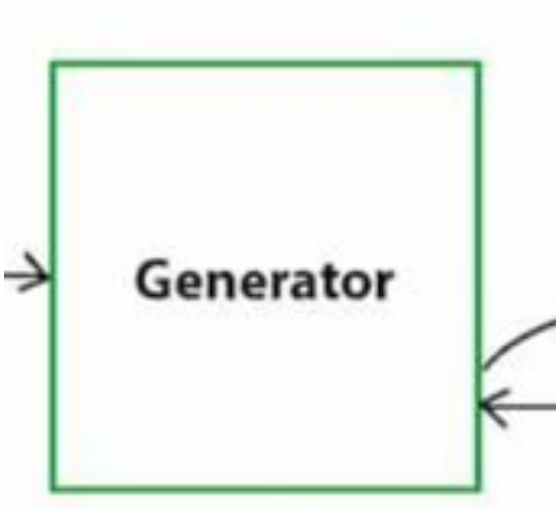
The steps involved in how a GAN works:



2.2. How does a GAN works

1. **Initialization**:

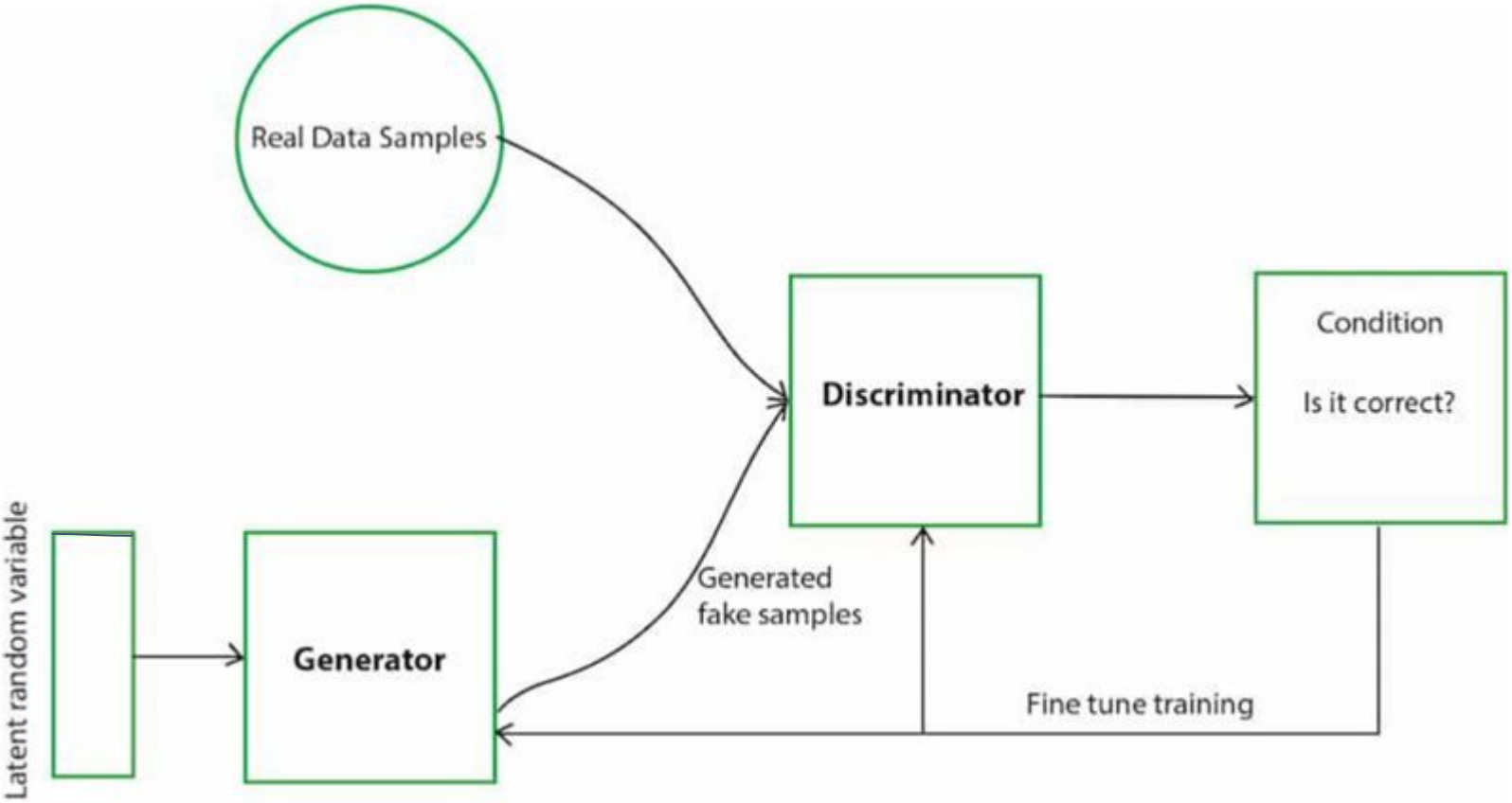
Two neural networks are created: a Generator (G) and a Discriminator (D).



2.2. How does a GAN works

2. Generator's First Move:

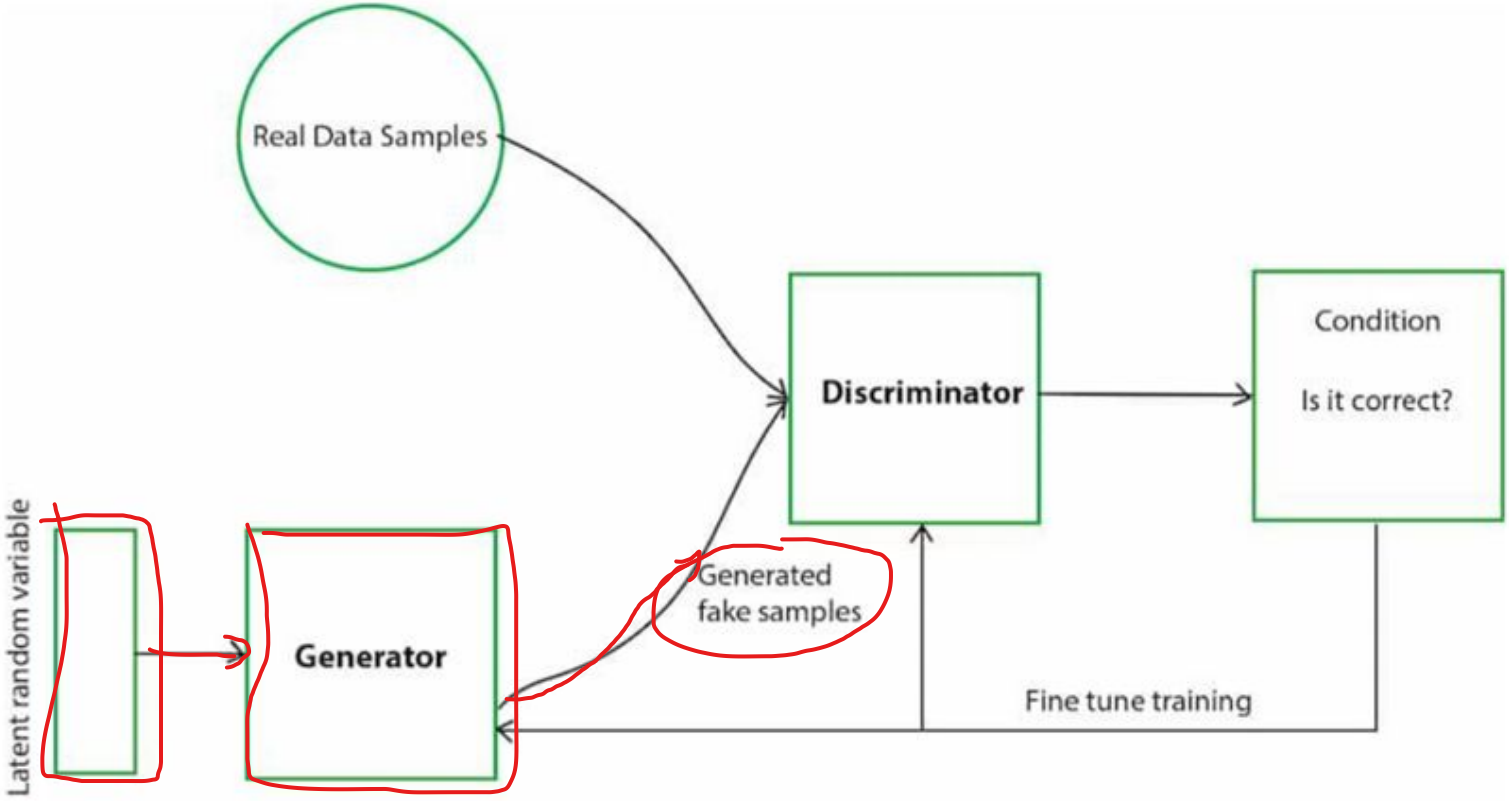
- G takes a random noise vector as input.



2.2. How does a GAN works

2. Generator's First Move:

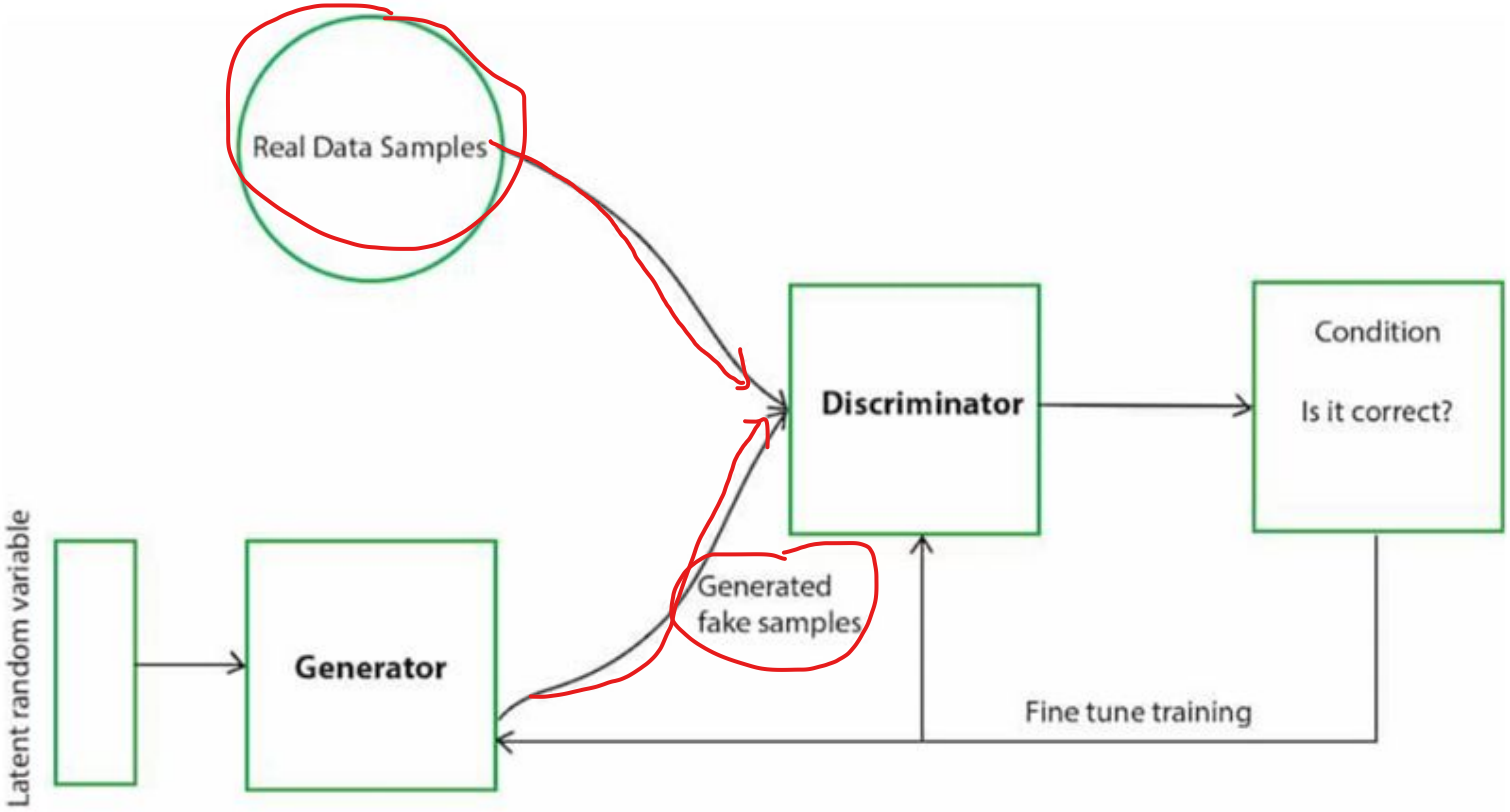
- Using its internal layers and learned patterns, G transforms the noise vector into a new data sample, like a generated image.



2.2. How does a GAN works

3. Discriminator's Turn:

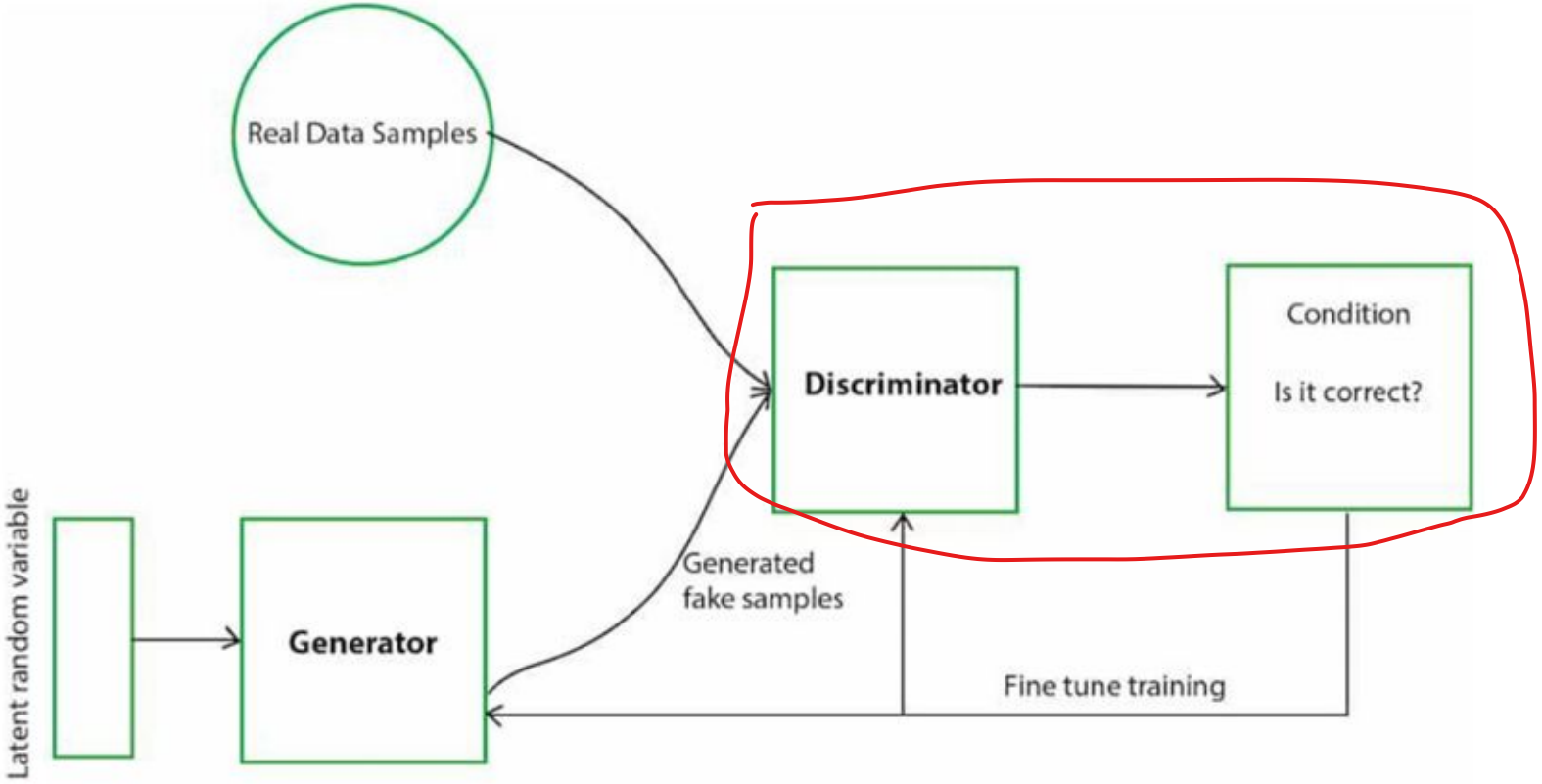
- D receives two kinds of inputs: Real data samples from the training dataset and the data samples generated by G in the previous step.



2.2. How does a GAN works

3. Discriminator's Turn:

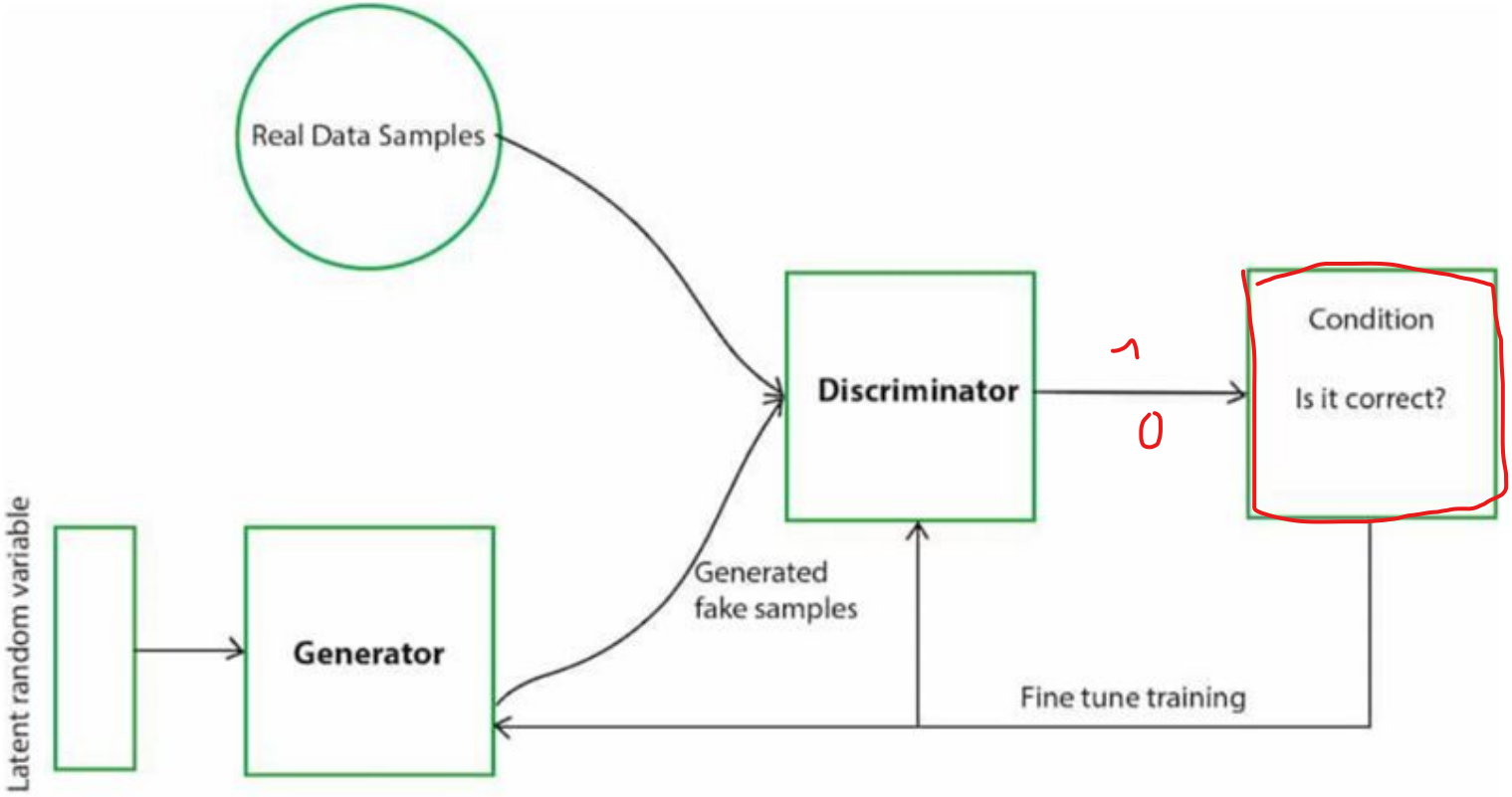
- D's job is to analyze each input and determine whether it's real data or something G cooked up.



2.2. How does a GAN works

3. Discriminator's Turn:

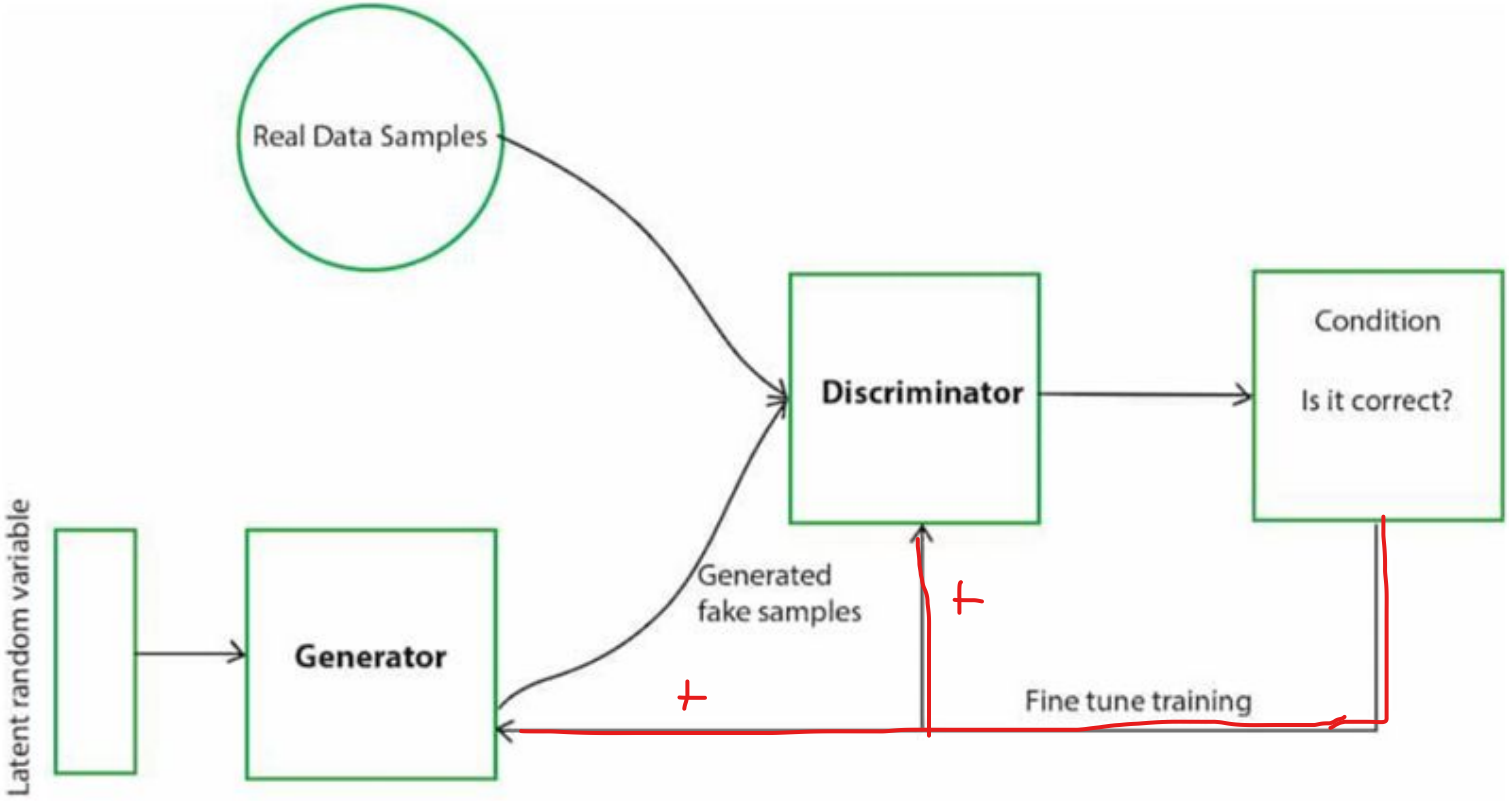
- It outputs a probability score between 0 and 1. A score of 1 indicates the data is likely real, and 0 suggests it's fake.



2.2. How does a GAN works

4. **The Learning Process:**

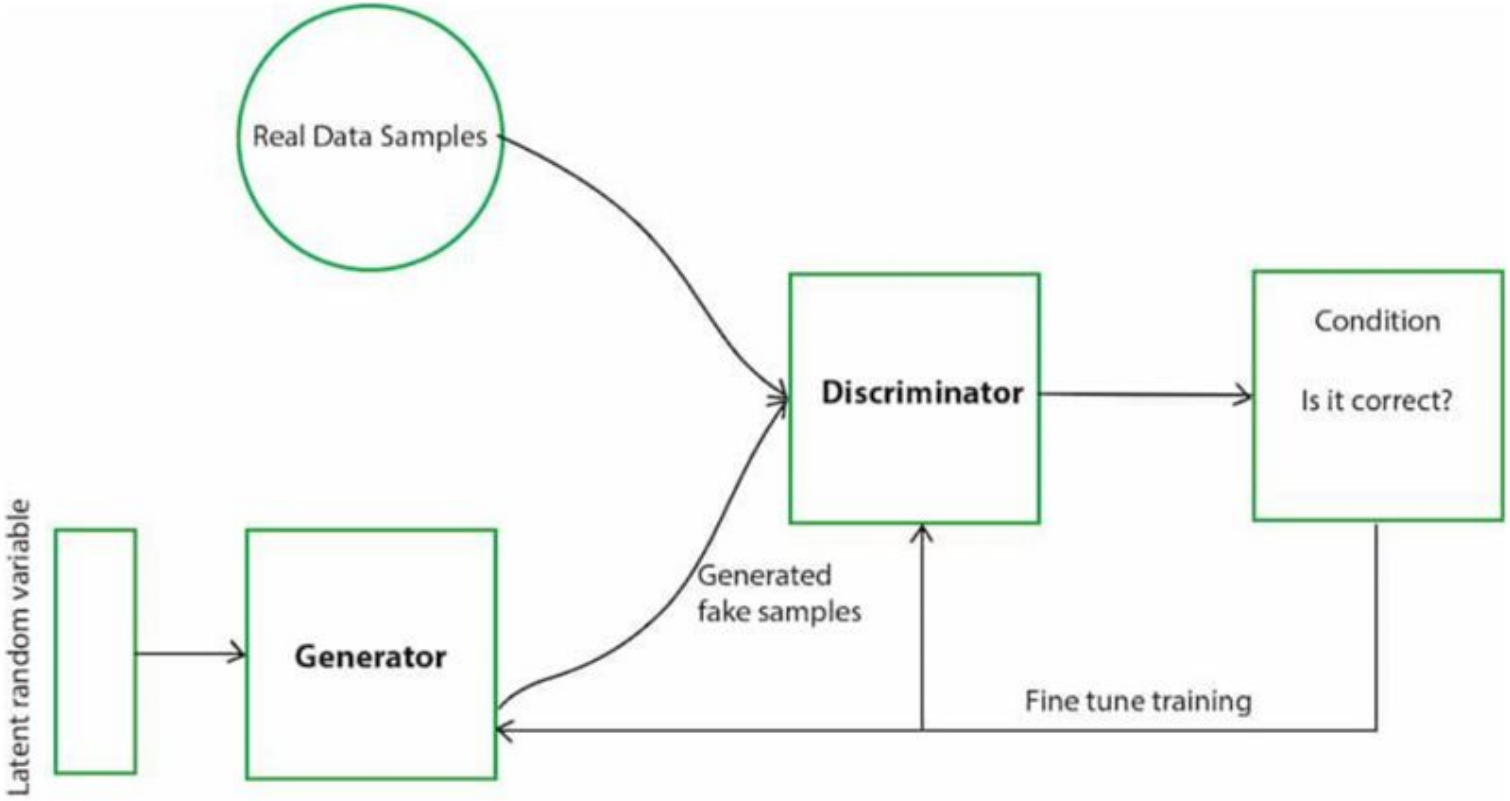
- If D correctly identifies real data as real (score close to 1) and generated data as fake (score close to 0), both G and D are rewarded to a small degree.



2.2. How does a GAN works

4. The Learning Process:

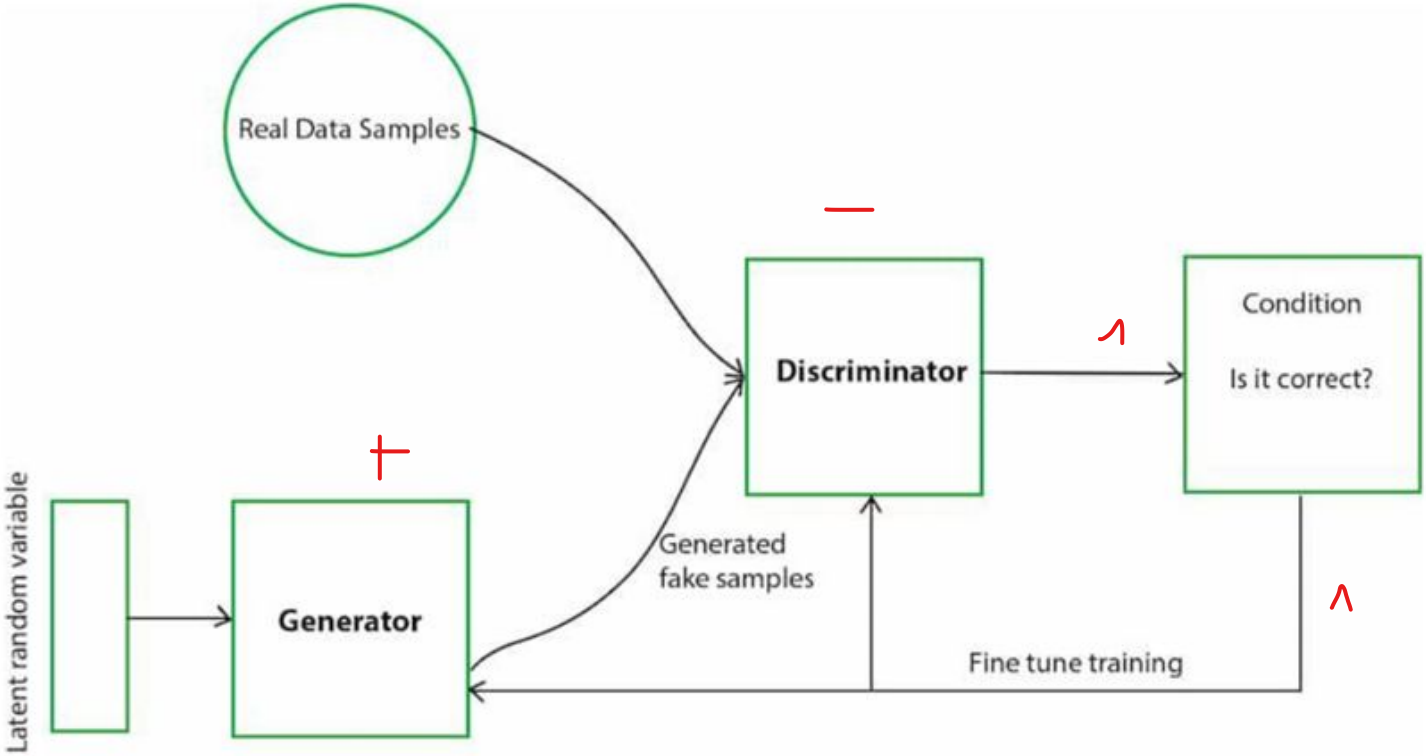
- However, the key is to continuously improve. If D consistently identifies every thing correctly, it won't learn much. So, the goal is for G to eventually trick D.



2.2. How does a GAN works

5. Generator's Improvement:

- When D mistakenly labels G's creation as real (score close to 1), it's a sign that G is on the right track.
- In this case, G receives a significant positive update, while D receives a penalty for being fooled.



2.2. How does a GAN works

6. **Discriminator's Adaptation:**

- Conversely, if D correctly identifies G's fake data (score close to 0), but G receives no reward, D is further **strengthened** in its discrimination abilities. This ongoing duel between G and D refines both networks over time.

- As training progresses, G gets better at generating realistic data, making it harder for D to tell the difference. Ideally, G becomes so adept that D can't reliably distinguish real from fake data.

- At this point, G is considered well-trained and can be used to generate new, realistic data samples

2.3. ResNet-based Generator

- **Residual Network** - In order to solve the problem of the vanishing/exploding gradient, this architecture introduced the concept called Residual Blocks.
- In this network, they use a technique called skip connections
- The skip connection connects activations of a layer to further layers by skipping some layers in between.

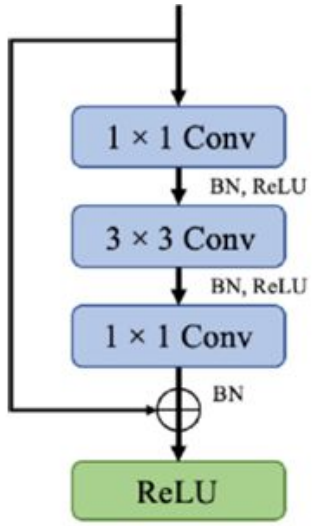


Figure 7. Basic residual block

```
class ResidualBlock(nn.Module):
    def __init__(self, channels):
        super(ResidualBlock, self).__init__()
        self.block = nn.Sequential(
            nn.Conv2d(channels, channels, kernel_size=3, stride=1, padding=1),
            nn.BatchNorm2d(channels),
            nn.ReLU(inplace=True),
            nn.Conv2d(channels, channels, kernel_size=3, stride=1, padding=1),
            nn.BatchNorm2d(channels),
        )

    def forward(self, x):
        return x + self.block(x)
```

2.3. ResNet-based Generator

- In this study, we implement a ResNet Generator network to address the problem of image deblurring, aiming to reconstruct sharp and detailed images from blurred inputs.

- The network consists of three main components: an encoder, a bottleneck with Residual Blocks, and a decoder.

```python
class Generator(nn.Module):
    def __init__(self, num_residual_blocks=6):
        super(Generator, self).__init__()

        # Encoder
        self.encoder = nn.Sequential(
            nn.Conv2d(3, 64, kernel_size=7, stride=1, padding=3),
            nn.BatchNorm2d(64),
            nn.ReLU(inplace=True),
            nn.Conv2d(64, 128, kernel_size=3, stride=2, padding=1),
            nn.BatchNorm2d(128),
            nn.ReLU(inplace=True),
            nn.Conv2d(128, 256, kernel_size=3, stride=2, padding=1),
            nn.BatchNorm2d(256),
            nn.ReLU(inplace=True),
        )

        # Bottleneck with residual blocks
        self.bottleneck = nn.Sequential(
            *[ResidualBlock(256) for _ in range(num_residual_blocks)]
        )
```

```python
# Decoder
self.decoder = nn.Sequential(
    nn.ConvTranspose2d(256, 128, kernel_size=4, stride=2, padding=1),
    nn.BatchNorm2d(128),
    nn.ReLU(inplace=True),
    nn.ConvTranspose2d(128, 64, kernel_size=4, stride=2, padding=1),
    nn.BatchNorm2d(64),
    nn.ReLU(inplace=True),
    nn.Conv2d(64, 3, kernel_size=7, stride=1, padding=3),
    nn.Tanh()
)
```

2.3. ResNet-based Generator

- Summary of how is works:

1.Input: Original image (CxHxW)

2.Basic feature extraction: Encoder transforms input through convolution layers, BatchNorm, and ReLU.

3.Learning complex features: Bottleneck processes features via a sequence of residual blocks.

4.Image reconstruction: Decoder reconstructs the output using transposed convolutions and Tanh.

5. Output: Fake image (C'xHxW)

```python
class Generator(nn.Module):
    def __init__(self, num_residual_blocks=6):
        super(Generator, self).__init__()

        # Encoder
        self.encoder = nn.Sequential(
            nn.Conv2d(3, 64, kernel_size=7, stride=1, padding=3),
            nn.BatchNorm2d(64),
            nn.ReLU(inplace=True),
            nn.Conv2d(64, 128, kernel_size=3, stride=2, padding=1),
            nn.BatchNorm2d(128),
            nn.ReLU(inplace=True),
            nn.Conv2d(128, 256, kernel_size=3, stride=2, padding=1),
            nn.BatchNorm2d(256),
            nn.ReLU(inplace=True),
        )

        # Bottleneck with residual blocks
        self.bottleneck = nn.Sequential(
            *[ResidualBlock(256) for _ in range(num_residual_blocks)]
        )
```

```python
# Decoder
self.decoder = nn.Sequential(
    nn.ConvTranspose2d(256, 128, kernel_size=4, stride=2, padding=1),
    nn.BatchNorm2d(128),
    nn.ReLU(inplace=True),
    nn.ConvTranspose2d(128, 64, kernel_size=4, stride=2, padding=1),
    nn.BatchNorm2d(64),
    nn.ReLU(inplace=True),
    nn.Conv2d(64, 3, kernel_size=7, stride=1, padding=3),
    nn.Tanh()
)
```

2.4. Discriminator

- In this study, we employ a Convolutional Neural Network based Discriminator to evaluate the realism of reconstructed images in the image deblurring task.
- The architecture consists of multiple Convolutional layers applied sequentially to extract features and reduce the spatial dimensions of the input image while increasing the feature depth.

```python
class Discriminator(nn.Module):
    def __init__(self):
        super(Discriminator, self).__init__()

        self.model = nn.Sequential(
            # Input: (B, 3, H, W)
            nn.Conv2d(3, 64, kernel_size=4, stride=2, padding=1),  # Downsample
            nn.LeakyReLU(0.2, inplace=True),

            nn.Conv2d(64, 128, kernel_size=4, stride=2, padding=1),  # Downsample
            nn.BatchNorm2d(128),
            nn.LeakyReLU(0.2, inplace=True),

            nn.Conv2d(128, 256, kernel_size=4, stride=2, padding=1),  # Downsample
            nn.BatchNorm2d(256),
            nn.LeakyReLU(0.2, inplace=True),

            nn.Conv2d(256, 512, kernel_size=4, stride=2, padding=1),  # Downsample
            nn.BatchNorm2d(512),
            nn.LeakyReLU(0.2, inplace=True),

            nn.Conv2d(512, 1, kernel_size=4, stride=1, padding=0),  # Final classification
            nn.Sigmoid()  # Output: Validity score
        )
```

2.5. Loss function

- Adversarial loss is expressed as:

$$\mathcal{L}_{adv}(G, D) = \mathbb{E}_{x\sim\text{data}}[\log D(x)] + \mathbb{E}_{z\sim\text{noise}}[\log(1 - D(G(z)))]$$

- Validation loss:

$$\text{val\_loss} = \frac{1}{N}\sum_{i=1}^{N}|\text{fake\_images}_i - \text{sharp}_i|$$

- The pixel loss:

$$\mathcal{L}_{\text{pixel}}(G) = \mathbb{E}_{x,y\sim\text{data}}\left[\|y - G(x)\|_1\right]$$

- The Discriminator loss:

$$\mathcal{L}_{\text{real}} = \mathcal{L}_{\text{adv}}(D(x_{real}), y_{real})$$

$$\mathcal{L}_{\text{fake}} = \mathcal{L}_{\text{adv}}(D(G(z), y_{fake})$$

$$\mathcal{L}_D = \frac{\mathcal{L}_{\text{real}} + \mathcal{L}_{\text{fake}}}{2}$$

- The total loss for the Generator:

$$\mathcal{L}_{G_{adv}} = \mathcal{L}_{\text{adv}}(D(G(z)), y_{real})$$

$$\mathcal{L}_{G_{\text{pix}}} = \mathbb{E}\left[\|G(z) - x_{\text{sharp}}\|_1\right]$$

$$\mathcal{L}_G = \mathcal{L}_{G_{adv}} + 100.\mathcal{L}_{G_{\text{pix}}}$$

HUST

hust.edu.vn fb.com/dhbkhn

3. Experiment

3. Experiment

Dataset:

- *GOPro: 2,902 images (Training, Validation, Test sets).*
- *Training: 1,029 blurry & 1,029 sharp images (Resized: 1280x720 → 640x360).*
- *Validation: 422 blurry & 422 sharp images.*

Preprocessing:

- *Align blurry and sharp images for synchronization.*
- *Maintain original aspect ratio during resizing.*

Hardware:

- *AMD Ryzen 7, NVIDIA RTX 3040Ti.*

3. Experiment

3.1. Discriminator Training

Input

- Real sharp images (labeled as 1).
- Fake images (labeled as 0).

Loss:

- Minimize adversarial loss to distinguish real vs. fake images.
- Optimizer updates weights accordingly.

3.2. Generator Training

Input:

- Blurry images → Generate fake sharp images.

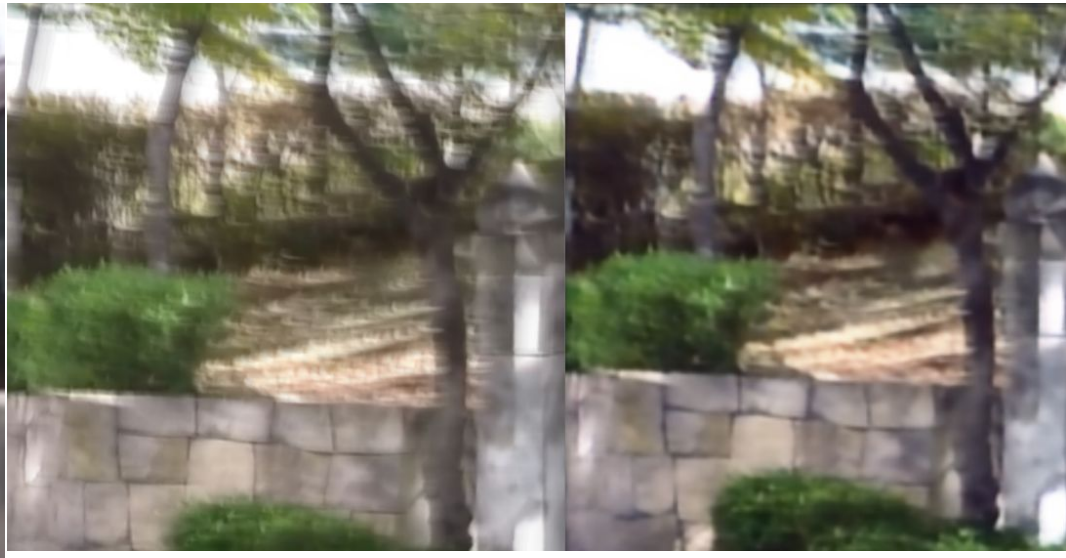
Loss:

- **Adversarial Loss:** Fool discriminator (fake → real).
- **Pixel Loss:** Difference between generated & ground-truth images.
- **Total Loss:** Weighted combination (Pixel Loss × 100).

3. Experiment

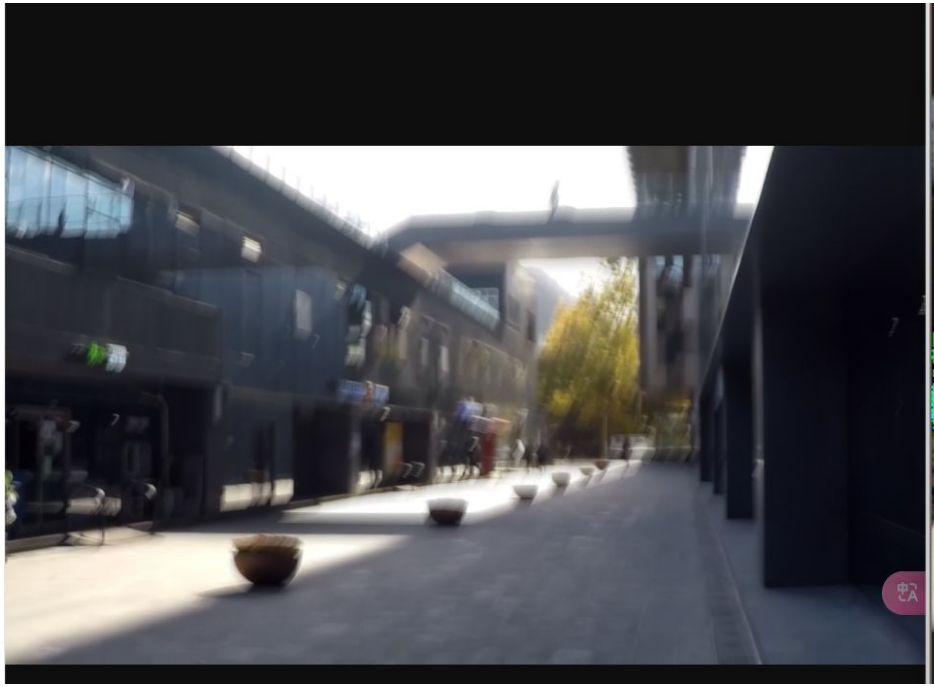
```
Epoch [4/80] Validation Loss: 0.0444
Epoch [5/80], Step [1/129], D Loss: 0.7126, G Loss: 4.7720
Epoch [5/80], Step [6/129], D Loss: 0.6887, G Loss: 5.1953
Epoch [5/80], Step [11/129], D Loss: 0.5895, G Loss: 4.0249
Epoch [5/80], Step [16/129], D Loss: 0.6529, G Loss: 4.4837
Epoch [5/80], Step [21/129], D Loss: 0.6723, G Loss: 4.2209
Epoch [5/80], Step [26/129], D Loss: 0.6310, G Loss: 4.4986
Epoch [5/80], Step [31/129], D Loss: 0.6585, G Loss: 4.3607
Epoch [5/80], Step [36/129], D Loss: 0.5742, G Loss: 4.3218
Epoch [5/80], Step [41/129], D Loss: 0.5834, G Loss: 5.1320
Epoch [5/80], Step [46/129], D Loss: 0.6790, G Loss: 4.6118
Epoch [5/80], Step [51/129], D Loss: 0.5871, G Loss: 4.8405
Epoch [5/80], Step [56/129], D Loss: 0.7221, G Loss: 4.4310
Epoch [5/80], Step [61/129], D Loss: 0.6745, G Loss: 4.5070
Epoch [5/80], Step [66/129], D Loss: 0.6075, G Loss: 4.0721
Epoch [5/80], Step [71/129], D Loss: 0.6622, G Loss: 5.0257
Epoch [5/80], Step [76/129], D Loss: 0.6814, G Loss: 4.5499
Epoch [5/80], Step [81/129], D Loss: 0.7209, G Loss: 3.5788
Epoch [5/80], Step [86/129], D Loss: 0.6617, G Loss: 3.6682
Epoch [5/80], Step [91/129], D Loss: 0.8094, G Loss: 3.3183
Epoch [5/80], Step [96/129], D Loss: 0.7283, G Loss: 4.8690
Epoch [5/80], Step [101/129], D Loss: 0.6495, G Loss: 4.5557
Epoch [5/80], Step [106/129], D Loss: 0.6794, G Loss: 5.1889
Epoch [5/80], Step [111/129], D Loss: 0.6878, G Loss: 4.7568
Epoch [5/80], Step [116/129], D Loss: 0.6923, G Loss: 4.1153
Epoch [5/80], Step [121/129], D Loss: 0.6914, G Loss: 4.2176
Epoch [5/80], Step [126/129], D Loss: 0.6512, G Loss: 4.5315
Epoch [5/80] Validation Loss: 0.0391
```

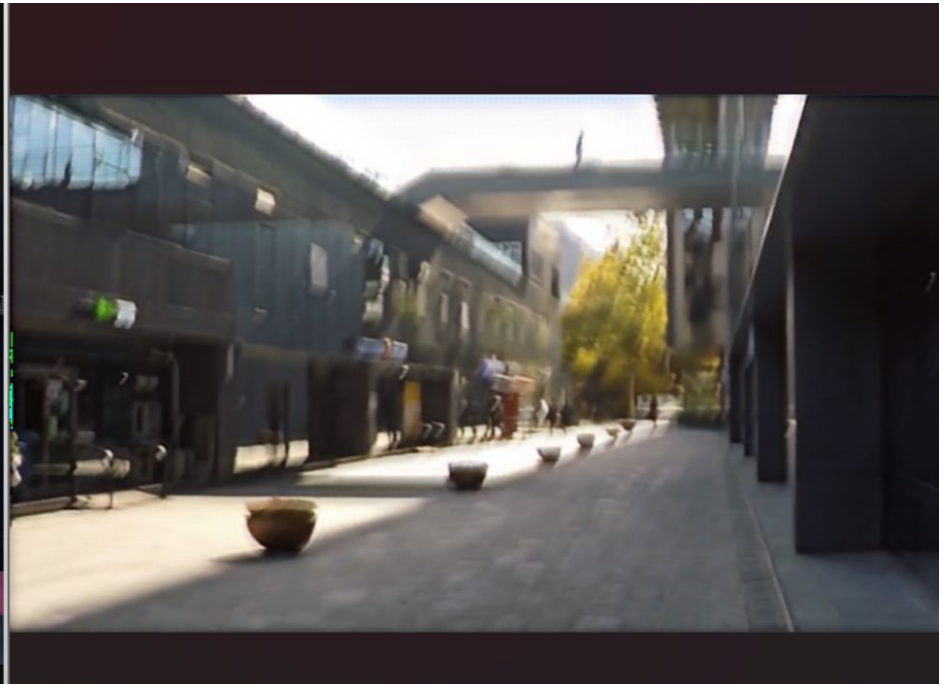
3. Experiment - Results



3. Experiment - Results

3. Experiment - Results





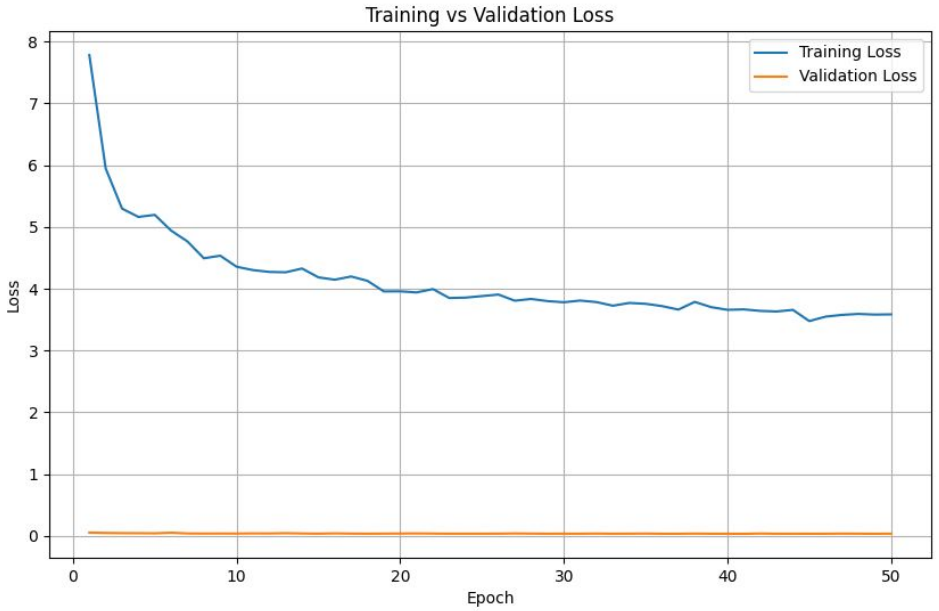
3.3. Logging and Validation

Logging:

- ***Every 5 steps:*** Generator & Discriminator loss recorded.
- ***After each epoch:*** Validate on the validation set.

Validation Loss:

- Measure pixel loss on validation data.
- Save best generator model based on lowest validation loss.



4. Model Evaluation

Evaluation Process

- **Pre-Trained Model:**
  - *Load best checkpoint weights.*
- **Dataset:**
  - *GOPro & RealBlur test sets.*
- **Evaluation:**
  - *Use DataLoader (batch size = 1) to process images one by one.*
  - *Generate sharp images from blurry inputs.*

4. Model Evaluation

General statistics

Metric	Mean	Std	Min	Max
PSNR	*23.979077*	*4.747543*	*7.644042*	*34.360462*
SSIM	*0.760050*	*0.218749*	*0.054897*	*0.976034*

- ***Peak Signal-to-Noise Ratio (PSNR)***:
  - Evaluates pixel-wise differences.
  - Higher PSNR = better similarity to ground truth.
- ***Structural Similarity Index Measure (SSIM)***:
  - Considers perceptual quality: luminance, contrast, structure.
  - Higher SSIM = better structural similarity.

4. Model Evaluation

- PSNR and SSIM computed for **each test image**.
- Results **highlight** the model's ability to **recover sharp images**.
- Metrics printed to the **console for analysis**.

Model	PSNR	SSIM
DeepDeblur [11]	29.23	0.916
SRN [21]	30.26	0.934
PSS-NSC [3]	30.92	0.942
DMPHN [23]	31.20	0.945
RADN [13]	31.76	0.953
SVDN [24]	29.81	0.937
MIMO-UNet [22]	31.73	0.951
MIMO-UNET++ [22]	32.68	0.959
Proposed model	23.97	0.7601

5. Discussion and Conclusion

Performance Summary

- Metrics Comparison
  - ***PSNR:*** Proposed Model (23.97) vs. State-of-the-art (e.g., MIMO-UNET++: 32.68)
  - ***SSIM:*** Proposed Model (0.7601) vs. MIMO-UNET++ (0.959), RADN (0.953)

5. Discussion and Conclusion

Challenges

- ***Low PSNR:*** Difficulty in reconstructing high-frequency details.
- ***Low SSIM:*** Limited structural similarity to ground-truth images.
- Comparison to Other Models
  - ***MIMO-UNET++*:** Advanced multi-scale & attention mechanisms.
  - ***SRN*, *PSS-NSC*:** Higher robustness to diverse blur patterns

5. Discussion and Conclusion

Improvement Directions

- **Model Architecture:** *Multi-scale processing, attention mechanisms.*
- **Loss Functions:** *Perceptual loss, refined adversarial loss.*
- **Training**
  - *Larger datasets.*
  - *Hyperparameter tuning.*
  - *Transfer learning with pre-trained models.*

5. Discussion and Conclusion

Conclusion

- **Achievements:**
  - *Implemented GAN-based deblurring model.*
  - *Valuable insights gained despite challenges.*
- **Limitations:**
  - *Performance below expectations.*
  - *Struggled with complex blur patterns.*
- **Future Directions:**
  - *Explore alternative architectures (e.g., encoder-decoder).*
  - *Optimize loss functions and training processes.*
  - *Investigate noise handling and dataset augmentation.*

6. Reference

[11] Seungjun Nah, Tae Hyun Kim, and Kyoung Mu Lee. Deep multi-scale convolutional neural network for dy namic scene deblurring. In IEEE Conf. Comput. Vis. Pat tern Recog., pages 3883–3891, 2017.

[21] Xin Tao, Hongyun Gao, Xiaoyong Shen, Jue Wang, and Ji aya Jia. Scale-recurrent network for deep image de blurring. In IEEE Conf. Comput. Vis. Pattern Recog., pages 8174 8182, 2018.

[3] Hongyun Gao, Xin Tao, Xiaoyong Shen, and Jiaya Jia. Dy namic scene deblurring with parameter selective sharing and nested skip connections. In IEEE Conf. Com put. Vis. Pattern Recog., pages 3848–3856, 2019.

[13] Kuldeep Purohit and AN Rajagopalan. Region adaptive dense network for efficient motion deblurring. In AAAI, pages 11882–11889, 2020.

[23] Yuan Yuan, Wei Su, and Dandan Ma. Efficient dy namic scene deblurring using spatially variant deconvolu tion net work with optical flow guided training. In IEEE Conf. Com put. Vis. Pattern Recog., pages 3555–3564, 2020.

[24] HongguangZhang, YuchaoDai, HongdongLi, and Piotr Ko niusz. Deep stacked hierarchical multi-patch net work for im age deblurring. In IEEE Conf. Comput. Vis. Pattern Recog., pages 5978–5986, 2019.

[22] Sung-Jin Cho, Seo-Won Ji, Jun-Pyo Hong, Seung Won Jung, Sung-Jea Ko. Rethinking Coarse-to-Fine Ap proach in Single Image Deblurring.

Division of Labor

Student	Tasks
Phạm Tùng Lâm	Studied about Generator and reconstructed it
	Dataset works
Bùi Thị Ngọc Anh	Studied about Discriminator and reconstructed it
	Construct utils functions
Both	Training model
	Write research
	Testing
	Slide making

HUST

THANK YOU !