Image deblurring with Generative Adversarial Network (GAN)

Pham Tung Lam, Bui Thi Ngoc Anh
Hanoi University of Science and Technology
No. 1 Dai Co Viet, Hanoi, Vietnam
{lam.pt224321, anh.btn224276}@sis.hust.edu.vn

Abstract

This project focuses on the development and training of a deep learning model aimed at image deblurring using a Generative Adversarial Network (GAN). The primary objective is to reconstruct sharp images from blurred inputs, a common challenge in computer vision tasks. The architecture comprises a ResNet-based generator and a GAN discriminator, both optimized using adversarial and pixel-wise loss functions. The model is trained on a dataset of blurred and sharp image pairs, with validation performed periodically to track progress and prevent overfitting. The training process includes the use of checkpointing to save the best-performing model, based on validation loss. A thorough evaluation of the model's performance is conducted through both qualitative visual inspection and quantitative metrics, such as PSNR (Peak Signal-to-Noise Ratio) and SSIM (Structural Similarity Index). The proposed approach demonstrates the effectiveness of GANs in solving image deblurring problems, providing a solid foundation for future improvements and applications in real-world scenarios. Source code is available for research purposes at: Source code

1. Introduction

Single image deblurring aims to restore a latent sharp image from a blurry one. Despite the rapid advancements in camera technology over the past few decades, blur artifacts still occur when the camera or objects are in motion. Blurry images not only appear visually unappealing but also significantly impact the performance of vision systems, including surveillance and autonomous driving, which require precise and efficient image deblurring methods. Thanks to the success of deep learning, convolutional neural network (CNN)-based image deblurring techniques have been widely studied and have demonstrated promising results.

Early CNN-based deblurring methods typically use CNNs to estimate the blur kernel and establish a two-stage deblurring framework: a CNN-based blur kernel estimation stage followed by a kernel-based deconvolution stage. More recently, CNN-based methods aim to directly model the complex relationship between blurry and sharp image pairs in an end-to-end fashion. A pioneering technique, DeepDeblur, introduces a deep multi-scale CNN for dynamic scene deblurring, which directly predicts a sharp image from a blurry one. DeepDeblur uses multiple stacked sub-networks to address multi-scale blur, with each sub-network processing a downscaled image and progressively recovering a sharp image from coarse to fine details.

Building on the success of DeepDeblur, several CNN-based deblurring methods have been proposed, yielding significant improvements in performance. While these methods aim to enhance deblurring in various ways, they share a similar coarse-to-fine approach, stacking multiple sub-networks. This coarse-to-fine design principle has proven effective in image deblurring.

In this report, we learn about GANs and try to replicate its application to image deblurring. Main challenges that we faced were noise amplification - restorative operations often exacerbate noise, handling unknown kernels - blur kernels are typically not known in real-world blurry images, complexity of natural images - images contain intricate details such as edges, fine textures, and high-frequency components, computational efficiency - algorithms need to be optimized for real-time applications, and training GANs models also faces with devices problem, especially when working with large datasets containing high-resolution input images. To ensure image quality and details, the input size needs to be sufficiently large, but this requires a significant amount of GPU memory. Limited GPU capacity often forces reductions in image size or smaller batch sizes, which can negatively impact training efficiency and the model's generalization ability. This poses a major challenge, requiring a balance between data quality and hardware capabilities. Because of the limitations of the device, we will build the network according to the most basic theory.

2. Related works

In this section, we review the conventional image deblurring methods that adopt a coarse-to-fine strategy

2.1. DeepDeblur

DeepDeblur, as a pioneering approach, directly models the relationship between blurry and sharp image pairs in an end-to-end fashion, utilizing a coarse-to-fine strategy. Nah et al. further contributed by introducing the GoPro dataset for real-world image deblurring. This dataset was created by capturing a sequence of sharp images at 240 fps using a GoPro camera, and generating a blurry image, B, by averaging consecutive sharp images, as follows:

$$B = \frac{1}{M}\sum_{i=0}^{M-1} S[i] \tag{1}$$

where M and $S[i]$ represent the number of sampled sharp images and the i^{th} sharp image, respectively. To construct a blurry and sharp image pair for training, the ground-truth sharp image for B is chosen by selecting the middle image from the sampled sharp images.

To implement a coarse-to-fine strategy in CNNs for the progressive recovery of latent sharp images, DeepDeblur employs multiple stacked sub-networks, as illustrated in Figure 1. Each sub-network comprises a series of convolutional layers that preserve the spatial resolution of the input feature maps. Input images at various scales are processed by these sub-networks, with the output from a coarser-scale sub-network concatenated with the input of the subsequent finer-scale sub-network. This design facilitates the transfer of coarse-to-fine information across scales.

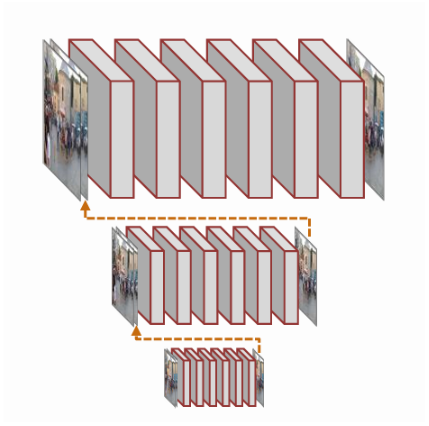


Figure 1. DeepDeblur network architectures

The reconstruction procedure of DeepDeblur is formulated as follows:

$$\hat{S}_n = \mathcal{H}^D_{\theta^D_n}\left(B_n, (\hat{S}_{n+1})^{\uparrow}\right) + B_n, \tag{2}$$

where $\mathcal{H}^D_{\theta^D_n}$ is the n^{th} sub-network of DeepDeblur parameterized by θ^D_n, B_n and $\hat{S}_n$ are blurry and deblurred images at the n^{th} scale, respectively and $\uparrow$ denotes the up-sampling operation.

2.2. PSS-NSC

Building on the success of DeepDeblur, Gao et al. proposed the Parameter Selective Sharing and Nested Skip Connections (PSS-NSC) framework. As depicted in Figure 2, the architecture of PSS-NSC resembles that of DeepDeblur but introduces two key innovations. First, each sub-network adopts an encoder-decoder-based U-Net structure with symmetric skip connections, enabling direct transfer of feature maps from the encoder to the decoder. Second, since all sub-networks share the common goal of recovering a sharp image from a blurry one, most network parameters are shared across sub-networks. This significantly reduces the memory requirements of PSS-NSC. However, the computational complexity remains high, as the final sharp image is generated after passing through three sub-networks.

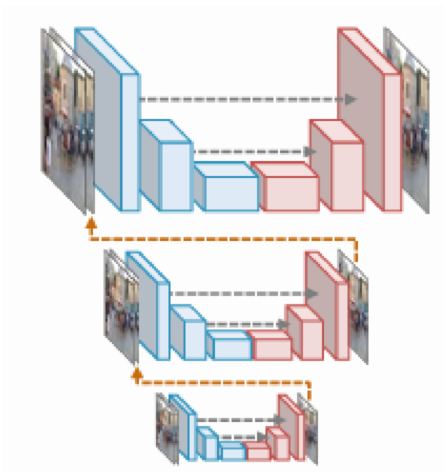


Figure 2. PSS-NSC network architectures

The reconstruction procedure of PSS-NSC is formulated as follows:

$$\hat{S}_n = \mathcal{H}^P_{(\theta^P_n,\theta^P)}\left(B_n, (\hat{S}_{n+1})^{\uparrow}\right) + B_n, \tag{3}$$

where $\mathcal{H}^P_{(\theta^P_n,\theta^P)}$ represents the n^{th} sub-network of PSS-NSC with exclusive parameter θ^P_n and shared parameter θ^P.

2.3. MT-RNN

The architecture of the Multi-Temporal Recurrent Neural Network (MT-RNN) is shown in Figure 3. In this design, a single U-shaped network is repeated seven times, with feature maps from the decoder in the previous iteration being passed to the encoder in the next iteration, as indicated by the green arrows. During each iteration, MT-RNN is trained to predict an averaged image computed with a progressively decreasing value of M in Eq. 1. This iterative process allows the model to refine its predictions over time. While the repeated use of the same U-shaped network results in low memory consumption, it comes at the cost of reduced runtime efficiency.

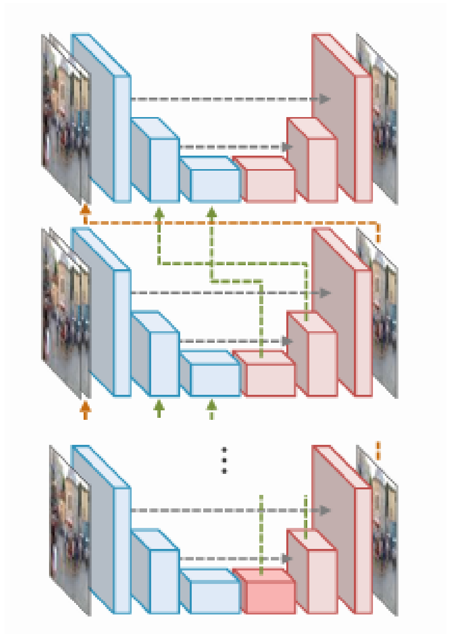


Figure 3. PSS-NSC network architectures

The reconstruction proce dure of MT-RNN is formulated as follows:

$$\left\{\hat{I}^i, F^i\right\} = \mathcal{H}^M_{\theta^M}\left(B^i; \hat{I}^{i-1}, F^{i-1}\right), \tag{4}$$

where i refers to an iteration index. $\mathcal{H}^M_{\theta^M}$ is the network of MT-RNN parameterized by θ^M. B^i, $\hat{I}^i$ and F^i are input blurry image, estimated latent image, and feature maps at the i^{th} iteration, respectively.

3. Proposed method

We propose a GAN architecture for image deblurring, where the generator adopts a ResNet structure, and the discriminator is designed as a PatchGAN. The generator, based on ResNet, consists of multiple residual blocks to effectively capture complex features and preserve image details. The PatchGAN discriminator is utilized to evaluate the realism of local image patches rather than the entire image, enabling better handling of fine details and textures. The following subsections elaborate on the key components of the architecture, including the residual block design in the generator and the patch-based evaluation strategy in the discriminator.

3.1. 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. Realistic, high-quality samples are produced as a result of this competitive interaction, which drives both networks toward advancement. GANs are proving to be highly versatile artificial intelligence tools, as evidenced by their extensive use in image synthesis, style transfer, and text-to-image synthesis. They have also revolutionized generative modeling.

Through adversarial training, these models engage in a competitive interplay until the generator becomes adept at creating realistic samples, fooling the discriminator approximately half the time.

Generative Adversarial Networks (GANs) can be broken down into three parts: Generative - To learn a generative model, which describes how data is generated in terms of a probabilistic model. Adversarial - The word adversarial refers to setting one thing up against another. This means that, in the context of GANs, the generative result is compared with the actual images in the data set. A mechanism known as a discriminator is used to apply a model that attempts to distinguish between real and fake images. Networks - Use deep neural networks as artificial intelligence (AI) algorithms for training purposes.

There are several different types of GANs, the GANs we discuss in this paper are Deep Convolutional GANs (DCGANs). DCGAN is one of the most popular and also the most successful implementations of GANs. It is composed of ConvNets in place of multi-layer perceptrons. The ConvNets are implemented without max pooling, which is in fact replaced by convolutional stride. Also, the layers are not fully connected.

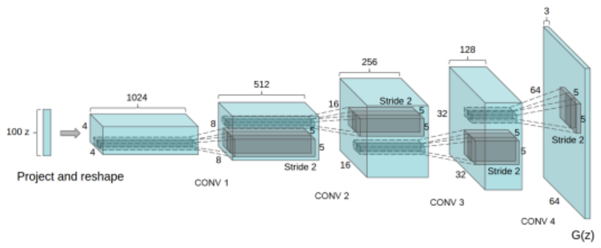


Figure 4. DCGANs - Generator architecture

3.2. Architecture of GANs

A Generative Adversarial Network (GAN) is composed of two primary parts, which are the Generator and the Discriminator.

Generator Model - A key element responsible for creating fresh, accurate data in a Generative Adversarial Network (GAN) is the generator model. The generator takes random noise as input and converts it into complex data samples, such text or images. It is commonly depicted as a deep neural network. The training data's underlying distribution is captured by layers of learnable parameters in its design through training. The generator adjusts its output to produce samples that closely mimic real data as it is being trained by using backpropagation to fine-tune its parameters. The generator's ability to generate high-quality, varied samples that can fool the discriminator is what makes it successful.

Generator Loss - The objective of the generator in a GAN is to produce synthetic samples that are realistic enough to fool the discriminator. The generator achieves this by minimizing its loss function J_G. The loss is minimized when the log probability is maximized, i.e., when the discriminator is highly likely to classify the generated samples as real. The following equation is given below:

$$J_G = -\frac{1}{m}\sum_{i=1}^{m}\log D(G(z_i)) \tag{5}$$

Where,

J_G measure how well the generator is fooling the discriminator. $logD(G(z_i))$ represents log probability of the discriminator being correct for generated samples. The generator aims to minimize this loss, encouraging the production of samples that the discriminator classifies as real ($logD(G(z_i))$), close to 1.

Discriminator Model - An artificial neural network called a discriminator model is used in Generative Adversarial Networks (GANs) to differentiate between generated and actual input. By evaluating input samples and allocating probability of authenticity, the discriminator functions as a binary classifier. Over time, the discriminator learns to differentiate between genuine data from the dataset and artificial samples created by the generator. This allows it to progressively hone its parameters and increase its level of proficiency. Convolutional layers or pertinent structures for other modalities are usually used in its architecture when dealing with picture data. Maximizing the discriminator's capacity to accurately identify generated samples as fraudulent and real samples as authentic is the aim of the adversarial training procedure. The discriminator grows increasingly discriminating as a result of the generator and discriminator's interaction, which helps the GAN produce extremely realistic-looking synthetic data overall.

Discriminator loss - The discriminator reduces the negative log likelihood of correctly classifying both produced and real samples. This loss incentivizes the discriminator to accurately categorize generated samples as fake and real samples with the following equation:

$$J_D = -\frac{1}{m}\sum_{i=1}^{m}\log D(x_i) - \frac{1}{m}\sum_{i=1}^{m}\log\left(1 - D(G(z_i))\right) \tag{6}$$

where,

J_D assesses the discriminator's ability to discern between produced and actual samples. The log likelihood that the discriminator will accurately categorize real data is represented by $logD(x_i)$. The log chance that the discriminator would correctly categorize generated samples as fake is represented by $log\big(1 - D(G(z_i))\big)$. The discriminator aims to reduce this loss by accurately identifying artificial and real samples.

MinMax Loss - In a Generative Adversarial Network (GAN), the minimax loss formula is provided by:

$$\min_G \max_D \mathcal{L}(G, D) = \mathbb{E}_{x\sim p_{\text{data}}}[\log D(x)] + \mathbb{E}_{z\sim p_z(z)}[\log(1 - D(g(z)))] \tag{7}$$

where,

G is generator network and is D is the discriminator network. Actual data samples obtained from the true data distribution $p_{data}(x)$ are represented by x. Random noise sampled from a previous distribution $p_z(z)$ (usually a normal or uniform distribution) is represented by z. $D(x)$ represents the discriminator's likelihood of correctly identifying actual data as real. $D(G(z))$ is the likelihood that the discriminator will identify generated data coming from the generator as authentic.

3.3. How does a GAN works

The steps involved in how a GAN works:

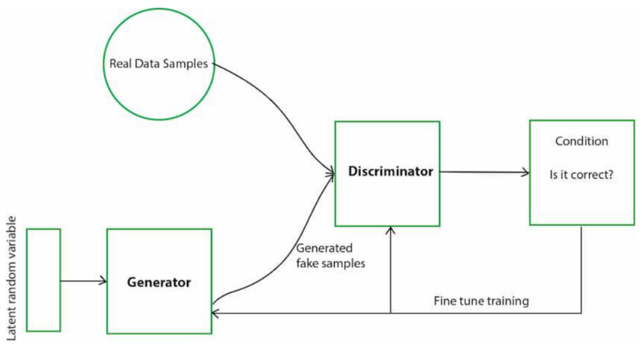


Figure 5. GAN diagram

Initialization - Two neural networks are created: a Generator (G) and a Discriminator (D). G is tasked with creating new data, like images or text, that closely resembles real data. D acts as a critic, trying to distinguish between real data (from a training dataset) and the data generated by G.

Generator's First Move - G takes a random noise vector as input. This noise vector contains random values and acts as the starting point for G's creation process. Using its internal layers and learned patterns, G transforms the noise vector into a new data sample, like a generated image.

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

The Learning Process - Now, the adversarial part comes in. 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. This is because they're both doing their jobs well. However, the key is to continuously improve. If D consistently identifies everything correctly, it won't learn much. So, the goal is for G to eventually trick D.

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. This feedback helps G improve its generation process to create more realistic data.

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.

3.4. ResNet - Based generator

After the first CNN-based architecture (AlexNet) that win the ImageNet 2012 competition, Every subsequent winning architecture uses more layers in a deep neural network to reduce the error rate. This works for less number of layers, but when we increase the number of layers, there is a common problem in deep learning associated with that called the Vanishing/Exploding gradient. This causes the gradient to become 0 or too large. Thus when we increases number of layers, the training and test error rate also increases.

In the following plot, we can observe that a 56-layer CNN gives more error rate on both training and testing dataset than a 20-layer CNN architecture. After analyzing more on error rate the authors were able to reach conclusion that it is caused by vanishing/exploding gradient.

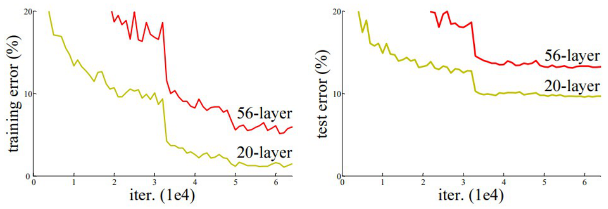


Figure 6. Comparison of 20-layer vs 56-layer architecture

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. This forms a residual block. Resnets are made by stacking these residual blocks together. The approach behind this network is instead of layers learning the underlying mapping, they allow the network to fit the residual mapping.

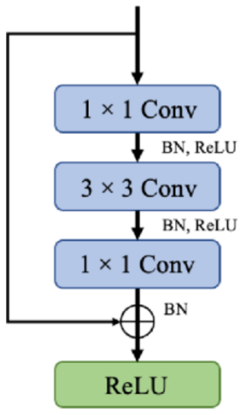


Figure 7. Basic residual block

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. This generator employs the Residual Network (ResNet) architecture, renowned for its effectiveness in learning complex features in deep image processing tasks. The network consists of three main components: an encoder, a bottleneck with Residual Blocks, and a decoder.

The encoder begins with a convolutional layer (Conv2d) that uses a kernel size of 7x7, a stride of 1, and padding of 3. This layer transforms the input blurred image into a feature map with 64 channels. It is followed by Batch Normalization and a ReLU activation function to normalize the feature map, reduce internal covariate shifts, and ensure stable and efficient learning. Subsequently, two additional convolutional layers, with kernel sizes of 3x3 and strides of 2, progressively increase the feature channels to 128 and then 256 while halving the spatial dimensions, creating a compact feature representation.

The bottleneck of the network comprises a sequence of residual blocks. Each block contains two Conv2d layers with 3x3 kernels, accompanied by Batch Normalization and ReLU activation. Crucially, the residual connections within these blocks enable the network to preserve low-level details and efficiently learn transformations, even as the network depth increases. This structure is particularly advantageous in image deblurring, where retaining fine details while reconstructing sharp edges is essential.

The decoder reconstructs the image using transposed convolutional layers (ConvTranspose2d) with kernel sizes of 4x4, strides of 2, and padding of 1. These layers upscale the feature map back to the original spatial dimensions while reducing the channel size to 128, then 64. A final convolutional layer with a kernel size of 7x7, stride 1, and padding 3 transforms the features into an output image with 3 channels. A Tanh activation function is applied to scale the pixel values to the range [-1, 1], ensuring compatibility with image restoration tasks.

This ResNet Generator, by leveraging the power of residual learning, is highly effective in addressing image deblurring challenges. The residual connections prevent degradation in network performance and facilitate the recovery of subtle image details, producing sharper and more accurate reconstructions. The architecture excels in handling motion blur, out-of-focus images, and other challenging scenarios.

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)

3.5. 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. This design allows the network to effectively capture and analyze critical image features.

The Discriminator begins with a Conv2d layer, which processes the input image (size B×3×H×W) using a kernel size of 4x4, a stride of 2, and a padding of 1. This layer outputs a feature map with 64 channels and reduces the spatial resolution by half. A LeakyReLU activation function (with a negative slope of 0.2) is then applied to introduce nonlinearity and mitigate the vanishing gradient problem.

Subsequent layers in the Discriminator follow a similar structure, progressively increasing the number of feature channels (128, 256, and 512) while reducing the spatial resolution. Each of these layers is accompanied by BatchNorm to normalize activations, thereby improving convergence and stabilizing training, and LeakyReLU to enhance learning capabilities.

The final layer of the Discriminator is a Conv2d layer with one output channel, a kernel size of 4x4, and a stride of 1. This layer maps the feature space into a single scalar value representing the validity score of the input image. A Sigmoid activation function is applied to this scalar output, ensuring that the validity score lies in the range [0, 1], which is suitable for binary classification tasks (real vs. fake).

This architecture is particularly well-suited for the image deblurring task, as it focuses on distinguishing realistic and unrealistic images based on their overall appearance. By learning to differentiate between real and generated images, the Discriminator effectively guides the Generator to produce sharper and more visually coherent results.

3.6. Loss function

In this study, two primary loss functions are employed to train the model effectively: adversarial loss and pixel loss.

The adversarial loss ($\mathcal{L}_{adv}$) is defined using Binary Cross-Entropy with Logits Loss (BCEWithLogitsLoss), which combines a sigmoid activation with binary cross-entropy. This loss is calculated between the Discriminator's output logits and the target labels, encouraging the Generator to produce images that the Discriminator cannot distinguish from real ones. Mathematically, the adversarial loss is expressed as:

$$\begin{aligned}\mathcal{L}_{adv}(G, D) = &\ \mathbb{E}_{x\sim\text{data}}[\log D(x)] \\ &+ \mathbb{E}_{z\sim\text{noise}}[\log(1 - D(G(z)))]\end{aligned} \tag{8}$$

where G is the Generator, D is the Discriminator, x represents real images, and $G(z)$ represents generated images.

The pixel loss ($\mathcal{L}_{pixel}$) is defined using the Mean Absolute Error (L1 loss). This loss measures the average absolute difference between the pixel values of the generated image and the ground truth, ensuring that the output images closely match the target images in pixel space. It is mathematically represented as:

$$\mathcal{L}_{\text{pixel}}(G) = \mathbb{E}_{x,y\sim\text{data}}\left[\|y - G(x)\|_1\right] \tag{9}$$

where y is the ground truth image and $G(x)$ is the generated image. By combining these two losses, the model optimizes both the perceptual realism and pixel-wise accuracy of the reconstructed images.

To train the Discriminator, the total loss ($\mathcal{L}_D$) is computed as the average of two components: the real loss ($\mathcal{L}_{real}$) and the fake loss ($\mathcal{L}_{fake}$). The real loss is calculated using the adversarial loss function applied to the Discriminator's output for real images ($D(x_{real})$) and their corresponding real labels (y_{real}). This encourages the Discriminator to correctly classify real images as authentic. The fake loss, on the other hand, is calculated using the Discriminator's output for generated (fake) images ($D(G(Z))$) and their corresponding fake labels (y_{fake}). This penalizes the Discriminator for misclassifying fake images as real. Mathematically, these losses are expressed as:

$$\mathcal{L}_{\text{real}} = \mathcal{L}_{\text{adv}}(D(x_{real}), y_{real}) \tag{10}$$

$$\mathcal{L}_{\text{fake}} = \mathcal{L}_{\text{adv}}(D(G(z), y_{fake}) \tag{11}$$

The Discriminator loss is then defined as:

$$\mathcal{L}_D = \frac{\mathcal{L}_{\text{real}} + \mathcal{L}_{\text{fake}}}{2} \tag{12}$$

By balancing these two components, the Discriminator learns to differentiate effectively between real and generated images, providing meaningful feedback to the Generator during the adversarial training process.

The total loss for the Generator ($\mathcal{L}_G$) is a combination of two components: the generator adversarial loss ($\mathcal{L}_{G_{adv}}$) and the generator pixel loss ($\mathcal{L}_{G_{pix}}$). This design ensures that the Generator not only produces images that appear realistic to the Discriminator but also maintains pixel-wise similarity to the ground-truth sharp images.

Generator Adversarial Loss ($\mathcal{L}_{G_{adv}}$): This loss is computed by applying the adversarial loss function to the Discriminator's output for generated (fake) images ($D(G(z))$) and the real labels (y_{real}). It encourages the Generator to produce images that the Discriminator cannot distinguish from real ones. Formally, it is expressed as:

$$\mathcal{L}_{G_{adv}} = \mathcal{L}_{\text{adv}}(D(G(z)), y_{real}) \tag{13}$$

Generator Pixel Loss ($\mathcal{L}_{G_{pix}}$): This loss, calculated using the L1 loss function, ensures that the generated images ($G(z)$) are close to the ground-truth sharp images (x_{sharp}) in terms of pixel values:

$$\mathcal{L}_{G_{\text{pix}}} = \mathbb{E}\left[\|G(z) - x_{\text{sharp}}\|_1\right] \tag{14}$$

The total loss for the Generator combines these two components, with the pixel loss scaled by a weight ($\lambda = 100$) to emphasize the importance of preserving image fidelity. The final loss function is shown below. By optimizing this loss, the Generator learns to produce sharp and realistic images, effectively addressing the image deblurring task.

$$\mathcal{L}_G = \mathcal{L}_{G_{adv}} + 100.\mathcal{L}_{G_{\text{pix}}} \tag{15}$$

In our model, the validation loss is calculated using L1 loss (pixel-wise loss) to measure the difference between the generated images and the corresponding sharp images in the validation set. After each epoch, we set the generator to evaluation mode and compute the validation loss by iterating through the validation data. For each batch of blurry images, the generator produces fake sharp images, and the pixel-wise difference between these fake images and the true sharp images is computed using the formula:

$$\text{val\_loss} = \frac{1}{N}\sum_{i=1}^{N}|\text{fake\_images}_i - \text{sharp}_i| \tag{16}$$

Where N is the number of images in the validation set, $fake\_images_i$ are the images generated by the model, and $sharp_i$ are the true sharp images. The loss for each batch is accumulated and then averaged over the entire validation set. This approach allows us to track the performance of the model in generating sharper images and its effectiveness in reducing image blurriness.

3.7. Complexity

3.7.1. Computational Complexity (Time Complexity)

The complexity of the proposed Generative Adversarial Network (GAN) model, designed for image deblurring, can be evaluated in terms of computational complexity, memory usage, and algorithmic structure. Below, we present a detailed analysis of these aspects.

The generator consists of three main components: encoder, bottleneck with residual blocks, and decoder. Each convolutional or transposed convolutional operation within these components has a computational complexity of:

$$O(K^2 \cdot C_{\text{in}} \cdot C_{\text{out}} \cdot H \cdot W) \tag{17}$$

where, K is the kernel size, C_{in},C_{out} are number of input and output channels and H, W are patial dimensions of the input tensor (height and width). The bottleneck contains 6 residual blocks, with each block comprising two convolutional layers. Thus, the overall complexity of the generator can be expressed with the complexity of encoder, residual block and decoder as:

$$O(E + nB + D) \tag{18}$$

The discriminator is a convolutional network with multiple downsampling layers, where each layer reduces the spatial dimensions of the input tensor by a factor of two. Complexity to each convolutional layer is similar above $O(K^2 \cdot C_{\text{in}} \cdot C_{\text{out}} \cdot H \cdot W)$

The computational cost of one epoch involves training both the generator and the discriminator for all batches in the dataset. Discriminator Training: Process real and fake images $\rightarrow$ 2 times through Discriminator, Generator Training: Pass through Generator and one more pass through

Discriminator. For a batch size of m, the complexity for one epoch is given by:

$$O(m \cdot (T_D + T_G)) \tag{19}$$

where, T_D is Discriminator calculation time, T_G is Generator calculation time. Hence, the complexity of training loop:

$$O(Epochs \cdot Batches \cdot m \cdot (T_D + T_G)) \tag{20}$$

3.7.2. Memory Complexity (Space Complexity)

Each convolutional layer stores weights and biases, contributing to a memory complexity of $O(K^2 \cdot C_{\text{in}} \cdot C_{\text{out}})$. This applies to all layers in both the generator and the discriminator.Residual blocks store additional intermediate tensors. Intermediate feature maps generated during the forward pass are stored in memory. For a batch size of B, and input tensor dimensions , the memory usage is $O(B \cdot C \cdot H \cdot W \cdot$ .The total memory requirement can be expressed as: $O(Params + B \cdot C \cdot H \cdot W \cdot)$ where includes the memory used for weights and biases, and refers to memory occupied by feature maps.

3.7.3. Algorithmic Complexity

The algorithmic complexity arises from the iterative training process of GANs, which involves alternating updates for the generator and discriminator. The residual blocks in the generator contribute to increased depth and improved feature learning but also add computational and memory overhead.

4. Experiment

In this project, we use a GOPro dataset consisting of a total of 2,902 images, divided into two main sets: the training set and the validation set (we mention about test set later). The training set includes 1,029 blurry images and 1,029 sharp images. The original size of the images in this set is 1280x720. However, due to GPU memory limitations, we scaled the images down to 640x360 to ensure efficient model training. The validation set, used to evaluate the model's performance, includes 422 blurry images and 422 sharp images. The data preprocessing involves aligning the blurry and sharp images to ensure proper synchronization between the input and output for the model. The image resizing is done with a ratio that maintains the original aspect ratio, minimizing the loss of information during the resizing process.Our experiments were conducted on AMD Ryzen 7 5000 series and NVIDIA RTX 3040ti.

The training loop in this model is designed to iteratively optimize both the generator and the discriminator using a combination of adversarial loss and pixel-wise loss. The loop runs for 80 epochs (this should be higher, but as i said before, our devices are not strong enough, so we have to use online tools with the cost of time limited), where the generator and discriminator are updated alternately in each iteration.

4.1. Discriminator Training

For each batch, the discriminator is first trained. It evaluates both real sharp images (from the dataset) and fake images (generated by the generator). The real sharp images are labeled with ones, and the fake images are labeled with zeros. The discriminator is then updated to minimize the loss for distinguishing real from fake images. The adversarial loss for real and fake images is computed, and the optimizer updates the discriminator's weights accordingly.

4.2. Generator Training

After training the discriminator, the generator is updated. The generator produces fake images from blurry input, and the discriminator evaluates these fake images. The generator's goal is to minimize the adversarial loss by making the discriminator think the fake images are real (using labels of ones). Additionally, the generator minimizes the pixel loss, which measures the difference between the generated images and the sharp ground truth images. The total generator loss is a weighted combination of these two losses, with the pixel loss scaled by a factor of 100.

4.3. Logging and Validation

Every 5 steps during training, the loss for both the discriminator and generator is logged to monitor the progress. After each epoch, the model is evaluated on the validation set to calculate the validation loss, which helps track the model's generalization performance. The validation loss is computed using pixel loss between the generated images and the sharp images in the validation set. If the validation loss improves (i.e., it decreases), the model checkpoint for the generator is saved as the best one. At the end of each epoch, the model checkpoint for the generator is saved. If the validation loss is the best so far, the checkpoint is updated as the best model.

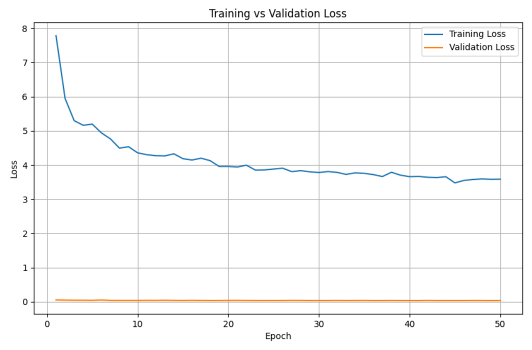
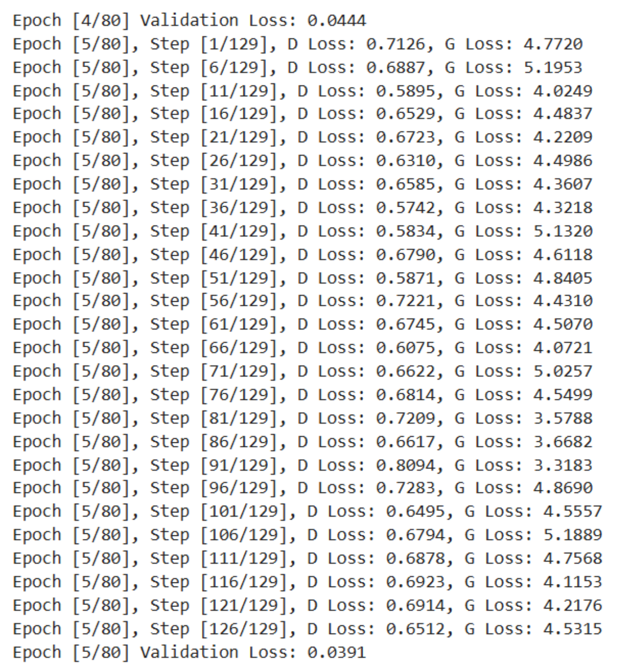


Figure 8. Loss plot



```
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
```

Figure 9. Training loop output

Finally, after training completes, the best generator model with the lowest validation loss is identified and saved. This process ensures that both the generator and discriminator are optimized in tandem, improving the generator's ability to produce realistic sharp images from blurry inputs.

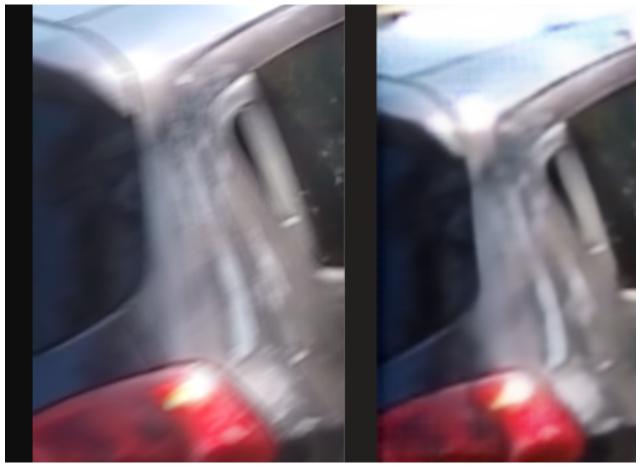


Figure 10. Example result 1

4.4. Model Evaluation

To evaluate the performance of the trained generator model, we first load the pre-trained model weights from the best checkpoint. .Next, we prepare the GOPro and RealBlur test

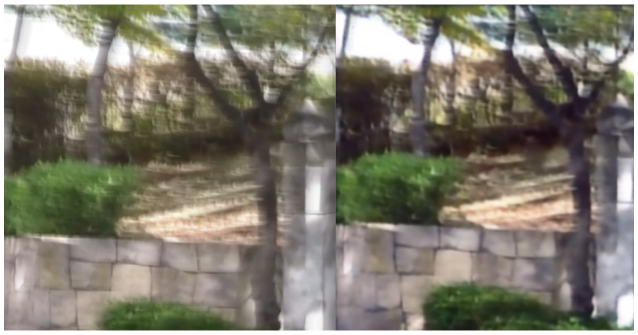
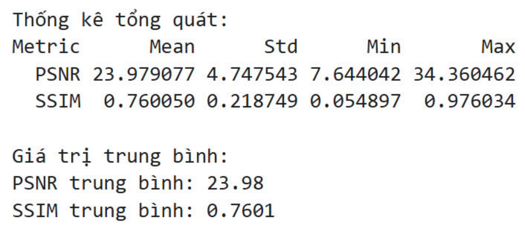


Figure 11. Example result 2

dataset and DataLoader. A DataLoader is created with a batch size of 1 to process each image in the test set one by one. During the evaluation loop, the generator model is used to produce sharp images from the blurry inputs. The output of the generator is then compared with the corresponding sharp images using two commonly used image quality metrics: Peak Signal-to-Noise Ratio (PSNR) and Structural Similarity Index Measure (SSIM).



```
Thống kê tổng quát:
Metric      Mean       Std       Min       Max
  PSNR 23.979077 4.747543 7.644042 34.360462
  SSIM  0.760050 0.218749 0.054897  0.976034

Giá trị trung bình:
PSNR trung bình: 23.98
SSIM trung bình: 0.7601
```

Figure 12. Average PSNR,SSIM on GoPRO test dataset

PSNR is calculated to assess the overall image quality, specifically focusing on the pixel-wise differences between the generated image and the ground truth sharp image. A higher PSNR indicates that the generated image is more similar to the sharp image. SSIM is used to evaluate the structural similarity between the generated and ground truth images, considering perceptual aspects of image quality such as luminance, contrast, and structure. Both PSNR and SSIM values are computed for each image in the test set and printed to the console.

This evaluation allows us to quantitatively assess the performance of the model in terms of how well it recovers sharp images from blurry inputs, with higher PSNR and SSIM values indicating better performance.

4.5. Discussion

The performance of the proposed image deblurring model using GANs, as indicated by the metrics in the table, falls short of the state-of-the-art methods in the field. Specifically, the PSNR of the proposed model is 23.97, which

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

Table 1. Performance comparison.

is significantly lower than the highest PSNR achieved by MIMO-UNET++ [22] at 32.68. Similarly, the SSIM of our model is 0.7601, which is far below the top-performing models, such as MIMO-UNET++ [22] with 0.959 and RADN [13] with 0.953.

These results highlight several challenges in the implementation of our GAN-based approach. Firstly, the relatively low PSNR suggests that our generator struggles to reconstruct high-frequency details from the blurred images. Secondly, the SSIM score indicates that the structural similarity between the generated and ground-truth images is inadequate, likely due to the limitations in the adversarial training process or the simplicity of the network architecture.

Comparing our model with others, such as SRN [21] or PSS-NSC [3], which achieve PSNR scores above 30 and SSIM scores above 0.93, suggests that our approach may lack the robustness or complexity needed to handle diverse blur patterns effectively. Models like MIMO-UNET++ likely benefit from more advanced architectures, such as multi-scale or attention mechanisms, which enhance their ability to capture intricate details in images.

The results underscore the necessity for further improvements in our approach. Incorporating features like multi-scale processing, more sophisticated loss functions (e.g., perceptual loss), or enhancing the discriminator's ability to guide the generator more effectively could potentially improve the model's performance. Additionally, fine-tuning hyperparameters, increasing the size of the training dataset, or leveraging transfer learning from pre-trained models might also yield better results.

Despite the current limitations, this project serves as a foundation for future work in applying GANs to image deblurring and provides valuable insights into the challenges of this complex task.

5. Conclusions

In this project, we attempted to develop an image deblurring system using Generative Adversarial Networks (GANs). While we were able to successfully implement the basic structure of the model, which involved training a generator to restore blurred images and a discriminator to distinguish between real and generated images, the results did not meet the desired level of performance. Despite extensive training and experimentation with various architectures and loss functions, the model struggled to produce sharp images that were visually comparable to the ground truth. Challenges such as the difficulty in handling complex blur patterns and the balance between adversarial and pixel loss affected the overall quality of the generated images.

The lack of success in achieving satisfactory results highlights the complexity of the image deblurring task, particularly when using GANs. However, this experience has provided valuable insights into the limitations of the current approach and opens up potential areas for future work. Future improvements could include exploring alternative network architectures, such as using encoder-decoder structures or refining the loss function to better capture image details. Additionally, further research into different types of blur, noise handling, and optimizing the training process could help address the current challenges and lead to more effective solutions in the future. Although the project did not fully meet expectations, it has provided a foundation for continued exploration in the field of image deblurring using deep learning techniques.

References

[1] Ayan Chakrabarti. A neural approach to blind motion de blurring. In Eur. Conf. Comput. Vis., pages 221–235, 2016.

[2] RobFergus, Barun Singh, Aaron Hertzmann, Sam TRoweis, and William T Freeman. Removing camera shake from a single photograph. In ACM SIGGRAPH 2006 Papers, pages 787–794. 2006.

[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. Comput. Vis. Pattern Recog., pages 3848–3856, 2019.

[4] Michal Hradi ˇ s, Jan Kotera, Pavel Zemcık, and Filip ˇ Sroubek. Convolutional neural networks for direct text deblurring. In Brit. Mach. Vis. Conf., volume 10, page 2, 2015.

[5] Naoyuki Ichimura. Spatial frequency loss for learning con volutional autoencoders. arXiv preprint arXiv:1806.02336, 2018.

[6] Orest Kupyn, Volodymyr Budzan, Mykola Mykhailych, Dmytro Mishkin, and Ji ˇ r ́ı Matas. Deblurgan: Blind motion deblurring using conditional adversarial networks. In IEEE Conf. Comput. Vis. Pattern Recog., pages 8183–8192, 2018.

[7] Orest Kupyn, Tetiana Martyniuk, Junru Wu, and Zhangyang Wang. Deblurgan-v2: Deblurring (orders-of-

magnitude) faster and better. In Int. Conf. Comput. Vis., pages 8878 8887, 2019.

[8] Bee Lim, Sanghyun Son, Heewon Kim, Seungjun Nah, and Kyoung Mu Lee. Enhanced deep residual networks for single image super-resolution. In IEEE Conf. Comput. Vis. Pattern Recog. Worksh., pages 136–144, 2017.

[9] S. Liu, H. Wang, J. Wang, and C. Pan. Blur-kernel bound estimation from pyramid statistics. IEEE Trans. Circuit Syst. Video Technol., 26(5):1012–1016, 2016.

[10] Tomer Michaeli and Michal Irani. Blind deblurring using internal patch recurrence. In Eur. Conf. Comput. Vis., pages 783–798, 2014.

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

[12] Dongwon Park, Dong Un Kang, Jisoo Kim, and Se Young Chun. Multi-temporal recurrent neural networks for progres sive non-uniform single image deblurring with incremental temporal training. In Eur. Conf. Comput. Vis., pages 327 343, 2020.

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

[14] Joseph Redmon, Santosh Divvala, Ross Girshick, and Ali Farhadi. You only look once: Unified, real-time object de tection. In IEEE Conf. Comput. Vis. Pattern Recog., pages 779–788, June 2016.

[15] JaesungRim, Haeyun-Lee,JucheolWon,andSunghyunCho. Real-world blur dataset for learning and bench marking de blurring algorithms. In Eur. Conf. Comput. Vis., pages 184 201, 2020.

[16] Olaf Ronneberger, Philipp Fischer, and Thomas Brox. U-net: Convolutional networks for biomedical image segmentation. In International Conference on Medical Image Computing and Computer-assisted Intervention, pages 234–241, 2015.

[17] Christian J Schuler, Michael Hirsch, Stefan Harmeling, and Bernhard Sch¨ olkopf. Learning to deblur. IEEE Trans. Pat tern Anal. Mach. Intell., 38(7):1439–1451, 2015.

[18] Ziyi Shen, Wenguan Wang, Xiankai Lu, Jianbing Shen, Haibin Ling, Tingfa Xu, and Ling Shao. Human-aware mo tion deblurring. In Int. Conf. Comput. Vis., pages 5572–5581, 2019.

[19] Maitreya Suin, Kuldeep Purohit, and AN Rajagopalan. Spatially-attentive patch-hierarchical network for adaptive motion deblurring. In IEEE Conf. Comput. Vis. Pattern Recog., pages 3606–3615, 2020.

[20] Jian Sun, Wenfei Cao, Zongben Xu, and Jean Ponce. Learn ing a convolutional neural network for non-uniform motion blur removal. In IEEE Conf. Comput. Vis. Pattern Recog., pages 769–777, 2015.

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

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

[23] Yuan Yuan, Wei Su, and Dandan Ma. Efficient dynamic scene deblurring using spatially variant deconvolution 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 network for im age deblurring. In IEEE Conf. Comput. Vis. Pattern Recog., pages 5978–5986, 2019.

References