

Single Image Reflection Removal with Mamba

Seyed Shayan (Shay) Daneshvar

007960634

MAMBA Lab

Image-based Generative Methods in Machine Learning

COMP7950 – Winter 2024

Dr. Christopher Henry

April 26, 2024

Introduction

In computer vision and image processing, addressing the issue of disentangling images affected by diverse superimposition scenarios remains a considerable challenge. These scenarios include tasks such as image deblurring, rain removal, dehazing, reflection removal, and more. In reflection removal, images that are taken through glass-like surfaces frequently capture unwanted reflections that degrade the visual quality of the image and present obstacles for downstream image processing and computer vision tasks [31].

Single Image Reflection Removal (SIRR) involves the separation of a surface (transmission layer T) from the reflection layer R using a single image. A comprehensive formulation for this task, as proposed by Wan et al. [1] is $I = \alpha T + \beta R + \Phi(T, R)$, where I is the blended image, α and β are coefficients that scale the layers, and $\Phi(T, R) = I - R - T$ captures the residue of the reconstruction process, encompassing various functions to model different residual situations such as overexposure and attenuation. Fig. 1 displays the comprehensive formulation visually.

Recent advances in research have sparked a renewed interest in the state space model (SSM). Building upon the classic Kalman filter model [15], modern SSMs exhibit proficiency in capturing long-range connections and benefit from parallel training approaches. The structured state space sequence model (S4) [29] has emerged to handle sequence data with an emphasis on modeling long-range dependencies.

Mamba [30] builds on top of S4 and incorporates time-varying parameters and proposes a hardware-aware algorithm to enhance the efficiency of training and inference processes. The significant scalability of Mamba suggests it could serve as a viable alternative to Transformer architectures in language modeling tasks. Mamba has found success in various vision tasks, such as dehazing [16] and segmentation [26, 27, 28, 33], yet its potential application in reflection removal remains unexplored.

Motivation, Problem Definition, and Contribution

Inspired by the recent accomplishments of Mamba in vision tasks, in this project we aim to integrate the state-of-the-art model proposed by Wan et al. [1] with the original Mamba module at various stages of their architecture. Through this endeavor, we seek to evaluate the efficacy of Mamba for the task of Single Image Reflection Removal (SIRR), results of which may be applicable to many other image restoration tasks. Additionally, we intend to explore the applicability of weight decay and Cosine Annealing Learning Rate scheduler [8], as demonstrated by Andriushchenko et al. [6] for Large Language Models (LLMs), in the context of Convolutional Neural Networks (CNNs) and a CNN employing Mamba at certain points in the network. This investigation will shed light on whether the findings observed in LLMs extend to CNNs and Mamba-utilized CNN architectures.

Given our decision to integrate the original Mamba within an existing model architecture rather than opting for a newer variant like Vision Mamba, it's crucial to develop our own ImageMamba module. A single Mamba module has a lot more parameters than a standard convolutional filter; therefore, we need to carefully select where to incorporate Mamba in the network to prevent the model from becoming too large for our GPU to handle.

Furthermore, the chosen architecture is already big in size and complex, yet offers various integration points for Mamba, thus we plan to experiment with placing Mamba in different parts

of the architecture to evaluate its performance. This approach will help us understand how effective the original Mamba is specifically for reflection removal.

The contribution of our work is as follows:

- Introduction of a new, straightforward Mamba-based module (Image Mamba) that can be seamlessly integrated at any point within a convolutional network, eliminating the requirement for input embedding dimensions.
- Enhancement of a prior model, surpassing state-of-the-art (SOTA) models in performance.
- A concise examination of the efficacy of weight decay regularization with AdamW, along with Cosine Annealing learning rate scheduler on two networks.

Additionally, all code written for the conducted experiments can be found [here](#).

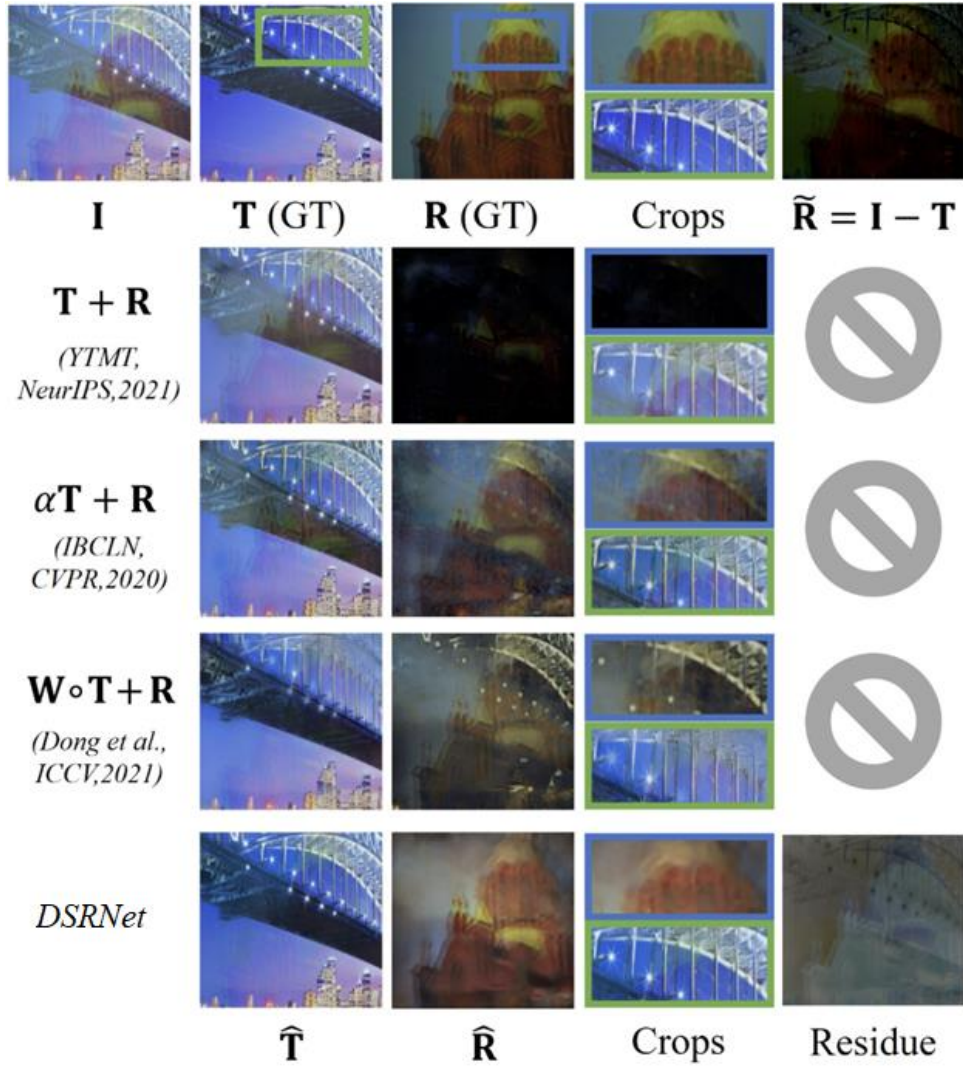


Figure 1: A visual example from [1] showing a real input **I**, ground truth **T** and **R**. The next rows show how different architectures extract **T**, **R**, and the residue that DSRNet generates and is the nonlinear part of blended reflection components.

Background and Related Work

The task of Single Image Reflection Removal (SIRR) involves separating two layers from a single image, making it an inherently ill-posed problem. This is because we have only one equation but at least two unknowns if we model the problem using the simplest form of $I=T+R$.

To address this inherent ill-posedness, additional knowledge or priors are required, such as manual annotations [17] or gradient-based constraints [17, 18, 19, 23]. Specifically, Levin et al. [19, 18] impose sparse constraints on gradients to obtain separations with fewer edges. Additionally, Levin and Weiss [17] require several manual annotations to obtain favorable decompositions, a manual process.

By leveraging Deep Learning (DL) models, such priors can be embedded inside the network given enough data is provided. IBCLN [20] and BDN [22] both utilize a simple form of the reflection model weighted by different scalars and iteratively estimate both layers. While this approach aims to prevent reflections from weakening, it may struggle to entirely remove transmission artifacts, as the reflection intensity varies across the image, making scalar estimation challenging for all pixels. In contrast, Wen et al. [21] propose a nonlinear model by predicting an alpha blending weight map in three channels with adversarial guidance. Additionally, Dong et al. [3] propose an iterative network that estimates a confidence map for the reflection layer in each iteration.

Of more recent models, CEILNet [5] applies a smoothness prior to the synthesis of reflection components and mixes them with transmission components via addition. However, the model's edge-aware approach to capturing transmission components disregards potentially valuable high-level semantics for the task of Single Image Reflection Removal (SIRR). To address this limitation, Zhang et al. [14] introduce HyperColumn features [11] which is a pre-trained VGG-19 network to extract semantic information from images, besides the perceptual, adversarial, and exclusion losses that they utilize. Li et al. [24] propose a two-stage network that estimates reflection layers and transmission components guided by them, while the reflection and transmission estimations are isolated in this way.

Furthermore, Hu and Guo [12] present the (Your Trash is My Treasure) YTMT method, employing a dual-stream interactive network to simultaneously restore both layers. Nevertheless, their assumption of a linear relationship between the components often results in weak reflection estimation.

Hu and Guo [1] introduce an end-to-end two-stage network, named Dual-stream Semantic-aware network with Residual correction (DSRNet), accompanied by a learnable residue module (LRM). The LRM learns the residue term $\Phi(T,R)$ in the more general formulation of SIRR, which is also introduced by the same paper, thereby generalizing the residue that should be removed from the image after the estimated approximate reflection has been eliminated. The first stage of their network, inspired by HyperColumn features [11], is a dual-stream pyramid fusion network (DSFNet). This stage decomposes and fuses multiscale semantic information in a hierarchical fashion using mutually gated interactive (MuGI) blocks and dual-stream fusion (DSF) blocks. The relatively decomposed features of layers are then passed into the second stage, namely the dual-stream fine-grained decomposition phase network (DSDNet), which incorporates the LRM module before the final layer. The architecture of this network is depicted in Fig 2.

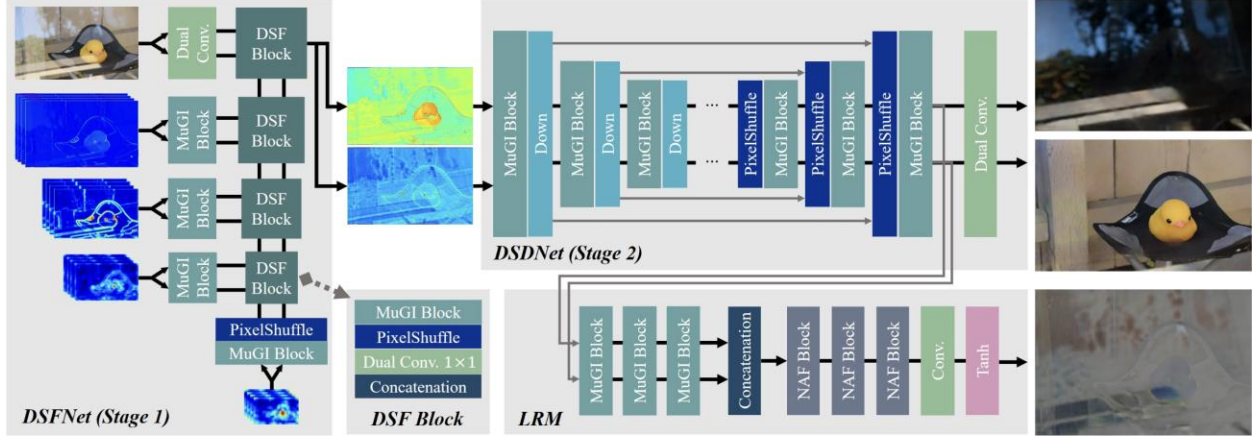


Figure 2: Illustrates the architecture of DSRNet [1], a two-stage activation-free network designed under the assumption that the layers can be separated linearly, attached to the LRM module that employs a single Tanh activation function to capture the residue for components that deviate from this linear assumption.

Theoretical framework

In this section, we provide the necessary information to comprehend the implementations used in this project. For brevity, we assume the reader is familiar with fundamentals typically covered in graduate-level or standard deep learning courses. Specifically, we won't delve into detailed explanations of the architecture of Adam optimizer, VGG 19, UNets, CNNs, Depthwise Separable Convolution, Encoder-Decoder Architecture, Activation Functions, Upsampling methods (including Bilinear interpolation), and other foundational concepts such as backpropagation.

Pixel Shuffle

Pixel Shuffle [4] is an upsampling operation commonly employed in super-resolution and image regeneration models to implement efficient sub-pixel convolutions with a stride of $1/r$. Essentially, it rearranges elements in a tensor of shape $(B, C * r^2, H, W)$ to $(B, C, H * r, W * r)$. In other words, each r^2 channels are used to create the new channel which will now have the size of $H * r$ and $W * r$. This operation's output dimensions are analogous to bilinear interpolation when the scale factor is r , as it does not involve learnable parameters. It, however, differs from bilinear upsampling in the sense that it reduces the size of channels while bilinear upsampling does not.

Dual Stream Modules

In DSRNet [1] and throughout this project, Dual Stream Modules share the same architectural design as their single-stream counterparts. However, they differ in that a dual stream module processes a data pair (x, y) , with both x and y passing through the module M sequentially. Consequently, the output of the dual stream module M is equivalent to $M(x)$ and $M(y)$ when M is a single stream standard module. An example of this is the Dual Conv modules depicted in Fig 2. In some figures, the two streams are visually represented separately, while in instances where space is limited, the module may be labeled as a dual-stream module without visually depicting two separate streams. However, it is important to note that they represent the same concept.

$$\hat{F}_T = G_1(F_T) \circ G_2(F_R); \hat{F}_R = G_1(F_R) \circ G_2(F_T), \quad (1)$$

Mutually Gated Interactive (MuGI) Block and MuGI Mechanism

This is the primary building block of DSRNet [1], operating under the linear assumption that if the nonlinear components can be estimated using the LRM module, then the equation $I - \Phi(T, R) = \alpha T + \beta R$ becomes linear. Consequently, solving the equation and obtaining the Transmission and Reflection layers requires few nonlinearities. As depicted in Fig. 3 by Hu and Guo [1], they formulate the synergy between the layers and develop the MuGI mechanism. In this mechanism, the transmission F_T and reflection F_R streams are fed into two simple functions G_1 and G_2 , outputs of which are then multiplied according to Eq. 1. This multiplication ensures that the output retains the same dimension, while the number of channels is halved. This reduction in the number of channels is achieved by designing the G functions to produce outputs half the size of the input. Specifically, they opted to keep these functions static, with G_1 taking the first half of channels and G_2 taking the second half.

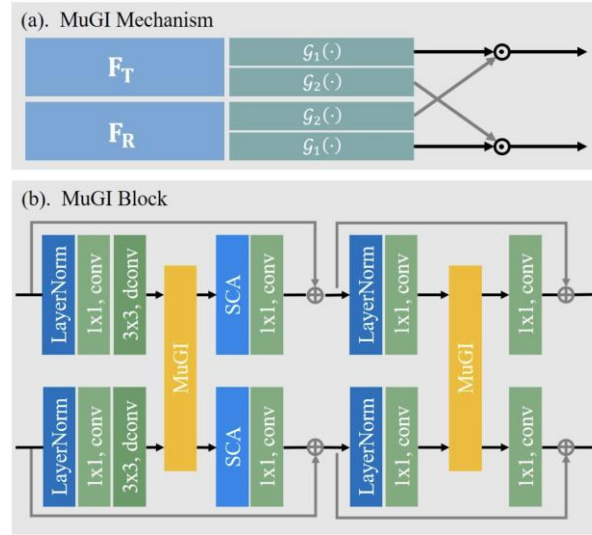


Figure 3: MuGI Block of DSRNet [1] and the MuGI mechanism used inside the MuGI block. The module dconv is a depth-wise separable convolution with a group size equal to the number of input channels, and output channels are twice the size of input channels.

2D Layer Normalization (LayerNorm2D)

LayerNorm2D, a normalization technique akin to BatchNorm2D, is specifically tailored for convolutional neural networks (CNNs) and offers nuanced advantages over batch normalization. Unlike batch normalization, which computes statistics over batches, LayerNorm normalizes activations within each sample independently across spatial dimensions. This finer granularity makes it particularly useful for scenarios with smaller batch sizes or when batch statistics might be unreliable [2]. By promoting stable training dynamics and enhanced generalization, LayerNorm2D contributes to the robustness and efficiency of CNNs.

Non-linear Activation Free (NAF) Block

Chen et al. [2] proposed NAFNet, comprising stacked NAF blocks as depicted in Fig. 4 of their work. The NAF block differs from traditional architectures by utilizing SimpleGate instead of activation functions to introduce non-linearity. As shown in Fig. 5, SimpleGate divides the input

into two halves, each with half the original number of channels, and multiplies them to produce an output with reduced channel dimensions. It's worth noting that MuGI mechanisms can be seen as a generalized version of SimpleGate for dual streams, while the entire NAF Block resembles a MuGI block when only one stream is utilized.

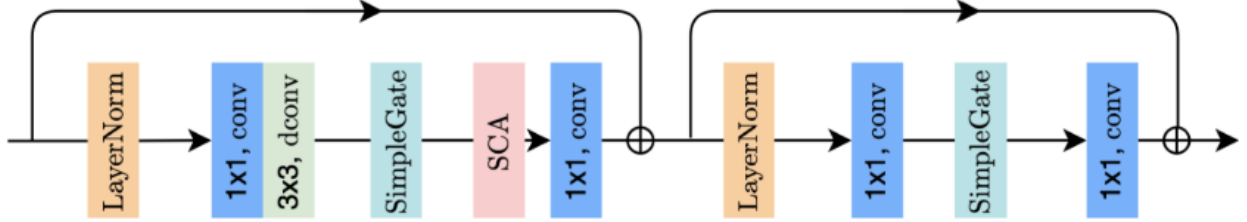


Figure 4: The architecture of NAFBlock from [2]. If we use the MuGI block with a single stream it will turn into a simple NAFBlock, and MuGI mechanism would act the same as the SimpleGate

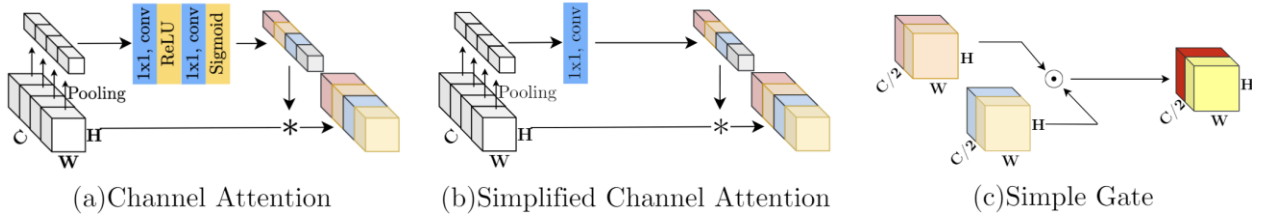


Figure 5: (From [2]) Illustrates (a) a from of Channel Attention (CA) which requires two 1x1 convolutions with an activation function in between, and a sigmoid to calculate the attention score similar to how softmax is used in Transformers, (b) Simplified Channel Attention (SCA), which is similar to CA, yet the attention score is calculated with a single 1x1 attention before being multiplied by the value vector, (c) The architecture of Simple Gate, from which MuGI was inspired

Transformers

Transformers [9] have emerged as a pivotal innovation in deep learning, particularly in natural language processing and more recently in computer vision. At its core lies the self-attention mechanism, which enables the model to weigh the importance of different input elements by computing attention scores between them. This attention mechanism allows transformers to capture long-range dependencies in data without relying on fixed-length context windows, making them highly effective for tasks requiring global context understanding. In vision transformers (ViTs) [10], this same attention mechanism is applied to image patches, allowing for efficient processing of images as sequences of tokens. By replacing traditional convolutional layers with self-attention mechanisms, vision transformers achieve impressive performance on various image-related tasks while maintaining flexibility and scalability in model architecture.

$$A = \text{Softmax}\left(\frac{QK^T}{\sqrt{d_k}}\right), \quad (2)$$

The attention mechanism in transformers involves computing attention scores between each pair of input elements and utilizing these scores to determine the significance of different elements. Mathematically, given a sequence of input vectors X , the attention scores A are computed according to Eq. 2. Here, Q , K , and V represent the query, key, and value matrices, respectively, obtained through linear transformations of the input vectors, with d_k representing a hyperparameter

akin to the dimension of key vectors. Subsequently, the attention scores are employed to calculate the weighted sum of the value vectors V , resulting in the output $Y = A V$. In vision transformers, this attention mechanism is adeptly applied to image patches, facilitating the efficient processing of images as sequences of tokens.

Simplified Channel Attention (SCA)

Inspired by the attention mechanism in Transformers, researchers have explored the application of similar principles in convolutional neural networks (CNNs). Chen et al. introduced a linear and straightforward form of attention, as illustrated in Fig. 5, which utilizes a 1×1 convolutional layer to process the input. The result of this convolutional operation is then multiplied elementwise with the input to produce the output, allowing the network to focus on relevant features while minimizing the focus on less significant information.

DSFNet

In the initial stage, illustrated in Fig. 2, the input (F_0) is passed through a VGG 19 network. Features generated at specific layers [F_1 : 2, F_2 : 7, F_3 :12, F_4 : 21, F_5 : 30] are extracted. Subsequently, F_5 undergoes processing through a MuGI block and then an Upsampling method (Bilinear Upsampling in the final implementation). Similarly, F_4 passes through a MuGI block, followed by a DSF block where the output of F_4 and F_5 are combined to yield a new F_4 . This architectural flow, as depicted accurately in Fig. 2, continues as F_3 is directed through a MuGI block and then to the DSF block alongside F_4 , resulting in a new F_3 . This process iterates until a new F_1 is generated. Finally, F_1 is passed through a DSF block alongside the new F_0 , which is the input processed through a MuGI block. The resulting output which are two images, is then forwarded to the second stage of the network.

DSDNet

In the second stage of the network, a UNet-like architecture is employed, featuring a sequence of MuGI blocks accompanied by Down blocks, repeated four times. The encoder comprises layers with 2, 2, 4, and 8 MuGI blocks, respectively, before passing through a Down module, which is a convolutional layer that doubles the number of channels and halves the dimensions. Subsequently, the bottleneck consists of 12 MuGI blocks connected to each other. In the decoder, the same Down module is utilized, but with a 1×1 convolutional filter instead of 3×3 , doubling the number of channels and feeding the output to a pixel shuffle operation with $r = 2$. The output of the UNet then undergoes processing through both a final convolution layer and the LRM module to generate the Transmission and Reflection layers, along with the residue, which is utilized in the calculation of the \mathcal{L}_{rec} loss function.

LRM

This module initially doubles the number of channels by feeding both streams into a 1×1 convolutional layer. Subsequently, the output is passed through two MuGI blocks connected to each other, with their outputs concatenated before being fed into four NAF blocks. Finally, a 3×3 convolutional layer accompanied by a Tanh activation function is applied to generate an image with three channels. The resulting output (residue) maintains the same dimensionality as the input, representing both the estimated transmission and reflection images.

$$\mathcal{L}_{\text{all}} = \mathcal{L}_{\text{pix}} + 0.01 * \mathcal{L}_{\text{per}} + \mathcal{L}_{\text{exc}} + 0.2 * \mathcal{L}_{\text{rec}} \quad (3)$$

Loss functions

In this section we cover all the loss functions used in the architecture of DSRNet. These loss functions are finally added together using as shown in Eq. 3.

$$\mathcal{L}_{\text{pix}} = \|\hat{T} - T\|_2^2 + \|\hat{R} - R\|_2^2 + 2 * (\|\nabla_x \hat{T} - \nabla_x T\|_2^2 + \|\nabla_x \hat{R} - \nabla_x R\|_2^2 + \|\nabla_y \hat{T} - \nabla_y T\|_2^2 + \|\nabla_y \hat{R} - \nabla_y R\|_2^2) \quad (4)$$

Pixel Loss: The pixel loss serves as a pivotal constraint in ensuring the consistency between the reconstructed layer and the ground truth across both the natural image domain and the gradient domain. Eq. 4 shows the equation for this loss, where ∇ is the gradient operator in the specified direction and $\|\cdot\|_2$ is the l_2 norm.

$$\mathcal{L}_{\text{rec}} = \|I - \hat{T} - \hat{R} - \phi(\hat{T}, \hat{R})\|_1 \quad (5)$$

Reconstruction Loss with Residual Rectification: With the use of the LRM we can write our main objective as a loss function that is depicted in Eq. 4. With the help of this loss, the nonlinear additive will flow to the LRM and the model can focus on the linear part of the equation.

$$\mathcal{L}_{\text{exc}} = \frac{1}{N} \sum_{n=0}^{N-1} \tanh(\eta_1 |\nabla \hat{T}^n|) * \tanh(\eta_2 |\nabla R^n|) \quad (6)$$

Exclusion Loss: To reinforce the gradient independence prior and mitigate structural coupling in the predicted layers, the loss is formulated as described in Eq. 6. The rationale behind this loss is that each layer possesses its own edges corresponding to the gradients of the image. When these layers are blended, the edges diminish due to smoothing, necessitating the optimal prediction to maintain the highest gradient intensity across both images. Here, the η_1 and η_2 are hyperparameters set like [14], and the superscript n represents downsampling 2^n times.

$$\mathcal{L}_{\text{per}} = \sum_{i \in \{2,7,12,21,30\}} \omega_i \|\phi_i(\hat{T}) - \phi_i(T)\|_1 \quad (7)$$

Perceptual Loss: Pixel-wise losses may fall short in guaranteeing multi-scale consistency between predictions and their ground truths. This limitation can lead to disproportionate penalization of entire images due to minor discrepancies in local brightness. Hence, we use Eq. 7 to formulate this loss. ω_i is the hyperparameter that is set using the same settings as [25], and ϕ_i is the output of VGG19 at the i th layer.

$$\begin{aligned} h'(+)&= Ah(t) + Bx(t) \\ y(t)&= Ch(t) + Dx(t) \end{aligned} \quad (8) \quad \begin{aligned} h_k &= \bar{A}h_{k-1} + \bar{B}x_k \\ y_k &= \bar{C}h_k \end{aligned} \quad (9)$$

$$\begin{aligned} \bar{K} &= (\bar{C}\bar{B}, \bar{C}\bar{A}\bar{B}, \dots, \bar{C}\bar{A}^k\bar{B}) \\ y &= x * \bar{K} \end{aligned} \quad (10)$$

Mamba

Also known as Structured State-Space models for Sequences with Selective Scan (S6), this approach builds upon Structured State-Space models for Sequences (S4), which, in turn, is rooted in State-Space models originating from Kalman Filters.

A state-space model, as defined in Eq. 8, maps a 1D input $x(t)$ into an ND latent $h(t)$, representing the system's state, before being transformed into a 1D output $y(t)$. Discretization, achieved through methods like the bilinear method or Zero-order hold, enables modeling within computer programs, as illustrated in Eq. 9, where \bar{A} , \bar{B} , and C are learnable matrices. Parallelization of computations is feasible by reformulating equations and employing convolutional filters, as shown in Eq. 10.

S4 integrates High-order Polynomial Projection Operators (HiPPO) to compress past inputs into coefficient vectors, influencing the initialization of matrix \bar{A} . Specifically, \bar{A} is structured to zero out values above the diagonal, set diagonal values to $n+1$, and adjust values below the diagonal to $(2i+1)^2(2j+1)^2$, where i and j are row and column indices, facilitating the model's ability to memorize its history, exploit long-range dependencies, and avoid random initialization biases.

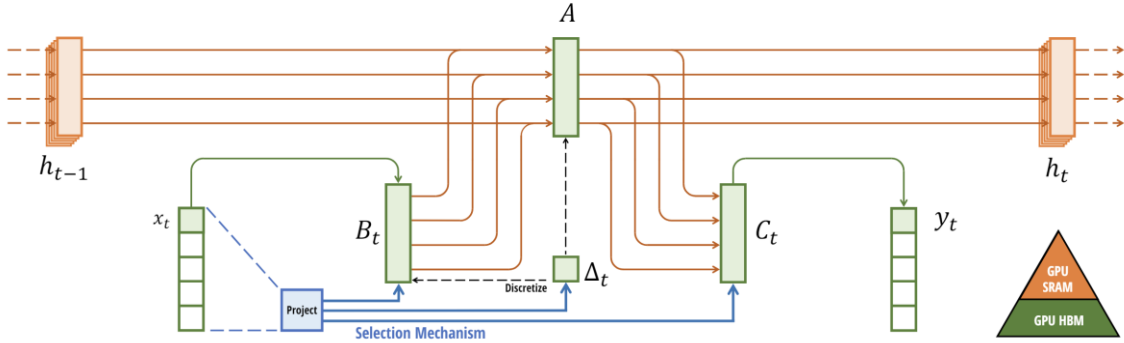


Figure 6: (from [30]) Illustration of the selection mechanism of Mamba, and how it directly affects Δ_t , B_t and C_t matrices.

Mamba introduces a selective scan mechanism to S4 while optimizing all operations for hardware, making stacking multiple Mamba blocks computationally more feasible than stacking self-attention modules of Transformers. Consequently, Mamba outperforms transformers in various applications, including medical semantic segmentation [26, 27, 28, 33], dehazing [16], and various other tasks [30]. Unlike S4, which exhibits limited learning capabilities due to small, static, and input-independent learnable matrices, Mamba dynamically adjusts matrices \bar{B} , and C based on the input length, thereby enhancing its adaptability. This dynamic adjustment, termed the scan mechanism, ensures unique matrices \bar{B} , and C , for each input token, as illustrated in Fig. 6.

Additionally, Mamba revolutionizes the discretization process by adding a selection mechanism, depicted in Fig. 6, and making Δ_t the discretization step, learnable. In contrast to previous methods, such as S4, where Δ_t remains fixed, Mamba's selective scan mechanism enables the model to adaptively focus on different scales. Smaller Δ_t values broaden the model's perspective to encompass larger landscapes, while larger Δ_t values prioritize individual tokens, effectively skipping more tokens in each iteration. Fig. 7 shows the distinction between Mamba's algorithm and S4, highlighting how Mamba loses its linear-time invariance property primarily due to the selection mechanism. Additionally, it illustrates that Mamba's performance is contingent upon the size of the input and the batch size.

Algorithm 1 SSM (S4)	Algorithm 2 SSM + Selection (S6)
Input: $x : (B, L, D)$ Output: $y : (B, L, D)$ 1: $A : (D, N) \leftarrow \text{Parameter}$ \triangleright Represents structured $N \times N$ matrix 2: $B : (D, N) \leftarrow \text{Parameter}$ 3: $C : (D, N) \leftarrow \text{Parameter}$ 4: $\Delta : (D) \leftarrow \tau_\Delta(\text{Parameter})$ 5: $\overline{A}, \overline{B} : (D, N) \leftarrow \text{discretize}(\Delta, A, B)$ 6: $y \leftarrow \text{SSM}(\overline{A}, \overline{B}, C)(x)$ \triangleright Time-invariant: recurrence or convolution 7: return y	Input: $x : (B, L, D)$ Output: $y : (B, L, D)$ 1: $A : (D, N) \leftarrow \text{Parameter}$ \triangleright Represents structured $N \times N$ matrix 2: $B : (B, L, N) \leftarrow s_B(x)$ 3: $C : (B, L, N) \leftarrow s_C(x)$ 4: $\Delta : (B, L, D) \leftarrow \tau_\Delta(\text{Parameter} + s_\Delta(x))$ 5: $\overline{A}, \overline{B} : (B, L, D, N) \leftarrow \text{discretize}(\Delta, A, B)$ 6: $y \leftarrow \text{SSM}(\overline{A}, \overline{B}, C)(x)$ \triangleright Time-varying: recurrence (<i>scan</i>) only 7: return y

Figure 7: (from [30]) Illustration of the selective scan mechanism used in Mamba, and how by making Δ_t , B_t and C_t input-dependent the model changes from time-invariant to time-varying.

AdamW and Cosine Annealing LR

AdamW [32], an extension of Adam, decouples weight decay from adaptive learning rates. Unlike Adam, which applies weight decay directly to both model weights and adaptive learning rates, AdamW applies weight decay exclusively to model weights. This separation prevents weight decay from influencing the adaptive moments of the gradients. As a result, AdamW ensures more stable optimization and improved generalization in deep learning models.

CosineAnnealingLR [8] is a learning rate scheduler that adjusts the learning rate according to a cosine function over epochs, gradually decreasing it from a maximum to a minimum value in a smooth curve. This scheduler can also be utilized to create oscillations in the learning rate (depicted in Fig. 8), resembling CosineAnnealing with Warm Restarts [8], albeit with a more gradual and colder restart process. This feature allows for versatile training strategies, aiding in better convergence and potentially escaping local minima, ultimately leading to improved performance and faster convergence compared to traditional learning rate schedules.

Weight decay, though less prevalent in modern deep learning practices, proves to be advantageous when combined with a CosineAnnealing learning rate schedule, as demonstrated by Andriushchenko et al. [6]. It is worth noting that Andriushchenko et al. utilized the CosineAnnealing learning rate schedule in its default configuration, where the learning rate is minimized at each epoch and eventually reaches 0 by the end of training epochs.

$$\text{SSIM}(x, y) = \frac{(2\mu_x\mu_y + c_1)(2\sigma_{xy} + c_2)}{(\mu_x^2 + \mu_y^2 + c_1)(\sigma_x^2 + \sigma_y^2 + c_2)} \quad (11)$$

Metrics

SSIM: Mean Squared Error (MSE) is commonly utilized for image comparison due to its simplicity and computational efficiency. However, MSE overlooks texture, which can lead to inaccuracies in assessing perceived similarity between images. To tackle this limitation, the Structured Similarity Index Measure (SSIM) was developed. SSI incorporates not only the mean and variance of two images but also their covariance, resulting in a more comprehensive measure of similarity. As a result, SSIM is frequently employed in image restoration tasks, where accurately gauging similarity is crucial for achieving high-quality outcomes. The formula for calculating SSIM is shown in Eq. 11, where μ and σ are the mean and variance (or covariance) and c_1 and c_2 are constant variables. Maximum value for SSIM is 1 which indicates that two images are the same.

$$\text{PSNR}(x, y) = 10 \log_{10} \left(\frac{\text{MAX}^2}{\text{MSE}(x, y)} \right) \quad (12)$$

PSNR: Peak Signal-to-Noise Ratio is a widely-used metric in deep learning for evaluating the quality of data reconstruction. It measures the ratio between the maximum possible power of a signal and the power of corrupting noise that affects the fidelity of its representation. Typically expressed in decibels (dB), higher PSNR values indicate better image quality. It's a straightforward measure often used in tasks like image compression, where minimizing information loss is crucial. PSNR is directly related to MSE and is calculated via Eq. 12, where MAX is the maximum value of a pixel which would be 255 or 1 if normalized.

Implementation details

In this section, we delve into the specifics of the architectural modifications that we applied to DSRNet, incorporating the original Mamba module. Additionally, we discuss the additional adjustments implemented to leverage AdamW and CosineAnnealingLR, deviating from the conventional fixed learning rate approach.

ImageMamba

To utilize the original Mamba module for processing images, it's necessary to reshape the image into a 1D vector before feeding it into the module, and then reshape it back to its original dimensions after processing. This functionality is encapsulated in our module, ImageMamba. ImageMamba reshapes the input image into a 1D vector with a dimension D equal to the number of output channels. It then passes this vector through a Mamba module with default settings (typically 16 states) before reshaping the output back into its original height, width, and number of channels. This approach enables the application of the original Mamba module to images without the need to rewrite a new module from scratch, incorporating a hardware-friendly 2D selective scan mechanism tailored for image processing.

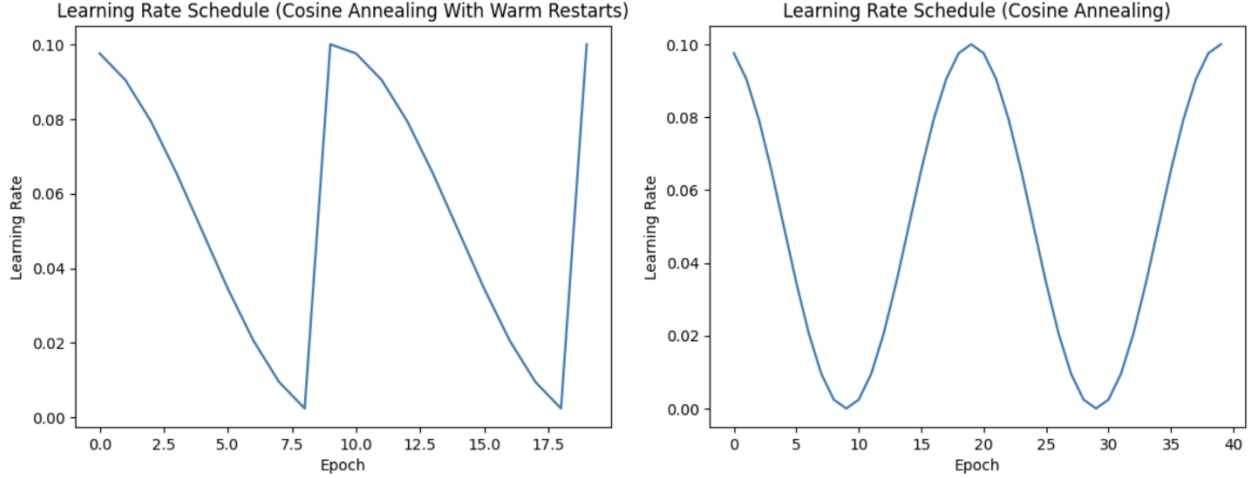


Figure 8: Illustrates the distinction between CosineAnnealing and CosineAnnealing With Warm Restarts when the number of epochs is set to 10. CosineAnnealing With Warm Restarts resets the learning rate when the target epoch is reached, whereas CosineAnnealing follows a standard Cosine curve to gradually restore the initial value for the learning rate.

Successful Models

In this section we list and explain the modules that were designed and trained successfully while adhering to the original hyperparameters of DSRNet.

Mamba DSRNet – V2 (M2DSRNet):

In this model, we replace the MuGI block in the second stage with our custom designed MuGI block, which we refer to as MuGI with Mamba v2 (MuGIM2). All MuGI blocks in the second stage are replaced with MuGIM2 blocks. Specifically, we integrate the ImageMamba module in place of the SCA module, with no other alterations made. This modification increases the number of parameters from 125M to 209M, as the DSDNetwork consists of 36 MuGI blocks. It's important to note that this parameter count pertains solely to the trainable model and excludes the parameters of VGG19, which are used both in the DSFNet and in the calculation of the Perceptual loss. The rationale behind this modification stems from the similarity between Mamba's selective scan mechanism and the attention mechanism of Transformers, which inspired the SCA module. Additionally, the SCA block employs a 1x1 convolution, which does not contribute to the network's receptive field. Therefore, replacing it with the MuGIM2 block, which integrates Mamba's selective scan mechanism, appears to be a logical decision.

Mamba2X DSRNet (MXDSRNet):

This extension builds upon MDSRNet-V2 by incorporating the MuGIM2 block originally designed for the DSDNet into the DSFNet as well. Consequently, both stages of the network now utilize MuGIM2 blocks. This enhancement results in an increase in the number of parameters to 213M.

DSRNet + AdamW + CosineAnnealingLR (DSRNet-W):

DSRNet-W involves training the original model with several adjustments: Adam is replaced with AdamW, which includes a weight decay coefficient of $1e-3$. Additionally, a CosineAnnealing scheduler is invoked in every iteration to adjust the learning rate for the subsequent sample. With a batch size of 1 and 300 epochs set, the learning rate starts at $1e-4$ and gradually decreases to the

minimum of $1e-5$ over 300 epochs. Subsequently, it returns to the original learning rate, repeating this cycle.

Each epoch involves feeding 7932 batches of 1 image to the network. Choosing 300 epochs ensures that every 600 batches are trained with a different learning rate. This setup allows each batch to apply its own effect over time. By selecting 300 as the number of epochs for the CosineAnnealingLR of Pytorch, after the first epoch, the shift becomes $7932 \% 600 = 132$, close to 150, which corresponds to approximately $\pi/4$ of a phase shift. This choice ensures that nearly every 4 epochs, each sample in the training set applies its effect. However, the slightly less than $\pi/4$ shift ensures slight changes in the learning rate, reducing the risk of overfitting.

It's important to note that while this approach shares similarities with CosineAnnealing with Warm Restarts, it differs in several aspects. Warm Restart typically increases the learning rate after it diminishes to the minimum in a semi-linear fashion and updates after every epoch. In contrast, DSRNet-W incorporates cold restarts and updates after every iteration. Hence this approach is a personal innovation completely based on intuition, differing from Warm Restarts in its update frequency and manner.

MDSRNet + AdamW + CosineAnnealingLR (MDSRNet-W):

MDSRNet-W is like DSRNet-W with the only difference being that instead of the original model we use M2DSRNet to see whether the settings mentioned also benefits the model when Mamba, which is a relatively new and unstable module, is used.

Failed Models

In this section, we list and explain the modules that were designed but encountered training failure due to model instability, sizing issues, or other reasons. These modules won't have their results presented in the experiments section, as they require larger GPUs or further experimentation. However, they're included here for completeness.

MDSRNet – V1: In the MuGI architecture, we made a significant adjustment by replacing two consecutive LayerNorms with an ImageMamba block and removing the convolutional filter that came after the LayerNorms, effectively introducing two Mamba blocks into the MuGI module. This alteration was implemented within the DSDNet *network*, allowing the model to remain trainable on an RTX3090 GPU. However, during the training process, a notable issue emerged: after approximately 15-17 epochs, there was a sudden spike in the loss function, resulting in the model consistently producing blank images. Despite experimenting with various learning rates and schedules *to* address this issue, no viable solution was found. Consequently, further investigation is *required* to determine the root cause of this instability.

MambaGated DSRNet (MGDSR): The MuGI mechanism, being straightforward and devoid of learnable parameters, prompted us to experiment with its architecture. We configured G1 as a Mamba module and G2 as a 1×1 convolutional filter to explore the potential of the MuGI mechanism. However, this adjustment substantially increased the model's size, rendering it untrainable with the same number of MuGI blocks as in the original DSRNet. Consequently, this configuration necessitates a larger GPU for training and testing purposes, and since we did not have any bigger GPUs, this model was left untested. Unfortunately, due to the unavailability of larger GPUs, this model remained untested.

Models	Real20 (20)		Objects (200)		Postcard (199)		Wild (55)		Average	
	PSNR	SSIM	PSNR	SSIM	PSNR	SSIM	PSNR	SSIM	PSNR	SSIM
Zh. et al. [14]	22.55	0.788	22.68	0.879	16.81	0.797	21.52	0.832	20.08	0.835
BDN [22]	18.41	0.726	22.72	0.856	20.71	0.859	22.36	0.830	21.65	0.849
ERRNet [25]	22.89	0.803	24.87	0.896	22.04	0.876	24.25	0.853	23.53	0.879
IBCLN [20]	21.86	0.762	24.87	0.893	23.39	0.875	24.71	0.886	24.10	0.879
RAGNet [24]	22.95	0.793	26.15	0.903	23.67	0.879	25.53	0.880	24.90	0.886
DMGN [7]	20.71	0.770	24.98	0.899	22.92	0.877	23.81	0.835	23.80	0.877
Zh. Et al. [13]	20.17	0.755	25.20	0.880	23.26	0.905	25.39	0.878	24.19	0.885
YTMT [12]	23.26	0.806	24.87	0.896	22.91	0.884	25.48	0.890	24.05	0.886
DSRNet [1]	24.23	0.820	26.28	0.914	24.56	0.908	25.68	0.896	25.40	0.905
M2DSRNet	23.24	0.801	26.83	0.920	24.0	0.904	26.23	0.907	25.42 ↑	0.907 ↑
MXDSRNet	23.01	0.793	26.33	0.917	24.06	0.901	24.96	0.9	25.07 ↓	0.903 ↓
DSRNet-W	23.58	0.802	26.76	0.919	25.01	0.91	26.36	0.908	25.84 ↑	0.909 ↑
MDSRNet-W	23.65	0.814	25.93	0.914	24.89	0.914	27.14	0.914	25.54 ↑↓	0.91 ↑↑

Table 1: Quantitative results on all four real benchmark datasets of previous methods as well as our 4 models. ↑↓ indicates whether the average performance increased or decreased compared to that of the original DSRNet. ↓↑ indicates the change in average performance compared to the original DSRNet trained with AdamW and CosineAnnealingLR.

Experiments, results, and discussion

All reported results stem from training the models for 25 epochs with a batch size of 1, followed by selecting the best-performing models based on evaluation results on the testing set, following the precedent set by previous works. All training, except for DSRNet-W which was trained on the Wrybill server with a single Nvidia A4000 16GB GPU, was conducted on the Quail server, utilizing a single Nvidia RTX3090 24GB GPU, as only the original DSRNet model would fit in a GPU with 16GB of memory and all the other models would take up at least 17GB of memory up to almost 24GB for MXDSRNet. Each model's training duration ranged from 64 to 68 hours. Hyperparameter settings are consistent with DSRNet, except for the number of epochs, which was increased from 20 to 25.

The dataset utilized is identical to the one employed in DSRNet. Our implementation is based on their code, with necessary modifications. The final training dataset consists of 7932 samples. Some samples originate from two datasets, Nature and Real, which contain both blended images and ground truth transmission layers. Others are generated by blending two random images from a third dataset, VOCDevkit, comprising only clean images. These datasets are fused together using predefined ratios (0.6 for generated samples and 0.2 for others), consistent with prior literature. Fusion entails pooling all data, but during sampling, proportions are adhered to as specified by the ratios. The testing set is sourced from four categories: Real, Objects, Postcard, and Wild, consisting of 20, 200, 199, and 55 items, respectively.

As shown in Table 1, all our models except for MXDSRNet outperform DSRNet on both metrics. Additionally, models trained with AdamW and CosineAnnealingLR surpass their counterparts trained with Adam and without weight decay. Notably, MDSRNet-W outperforms DSRNet-W on SSIM, the more crucial metric, although it slightly underperforms in terms of PSNR. Thus, the overall superiority of our MDSRNet is evident.

However, when visually comparing these methods in Table 2 and especially in Table 3, the superiority may not be immediately evident. This could be attributed to the fact that the baseline model already performs well, and our improvements, albeit significant, might not be visually perceivable to the human eye.





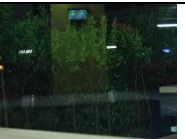






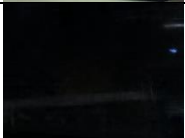
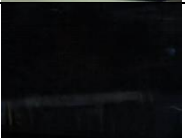
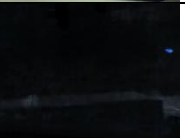
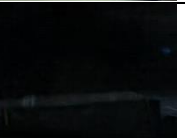










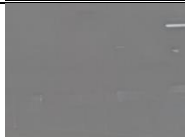



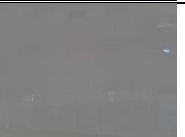
	MDSRNet-W	DSRNet-W	M2DSRNet	DSRNet [1]	MXDSRNet
Input					
Estimated T					
Estimated R					
Ground Truth T					
Ground Truth R					
Estimated RR					

Table 2: Visual Comparison of image wild_014 and its predicted transmission T, reflection R, and residual of reflection RR images. The superior performance of DSRNet-W and MDSRNet-W is clear and visible to the naked eye.

Discussion

In this section, our goal is to analyze the improvements and deteriorations in performance observed in our models and provide insights into the underlying reasons behind their performance.

M2DSRNet and MXDSRNet

The performance enhancement observed in MDSRNet underscores the utility of Mamba as a replacement for previous attention-based or self-attention-like modules. Conversely, the inferior performance of MXDSRNet, like M2DSRNet but with Mamba utilized in the first stage of the network, suggests that Mamba may not surpass simple convolution in extracting low-level data. In essence, Mamba excels when handling already extracted and high-level data. Therefore, we infer that Mamba is most beneficial when employed in later stages of the network, particularly as a substitute for more conventional attention modules such as SCA.

MDSRNet-W and DSRNet-W

The results clearly demonstrate the efficacy of our approach, utilizing CosineAnnealingLR with AdamW's weight decay, in enhancing the model's performance by enabling it to explore various learning rates at every iteration. However, this approach behaves slightly differently when applied to a fully convolutional network like DSRNet compared to a network containing Mamba. Specifically, although the Mamba-based model still outperforms the other model, the increase in SSIM achieved for DSRNet is significantly greater than for the Mamba-based model. We attribute this difference to the fact that Mamba is a more hardware-friendly and stable module, and is less sensitive to changes in the learning rate.

In conclusion, the combination of CosineAnnealingLR updated per iteration alongside AdamW with weight decay can indeed benefit the SIRR models and probably image restoration models, and Mamba exhibits less sensitivity to changes in the learning rate compared to traditional convolutional networks.


	MDSRNet-W	DSRNet-W	M2DSRNet	DSRNet [1]	MXDSRNet
Input					
Estimated T					
Estimated R					
Ground Truth T					
Ground Truth R					
Estimated RR					

Table 3: Visual Comparison of image wild_006 and its predicted transmission T, reflection R, and residual of reflection RR images.

EDI

In our research, Equity, Diversity, and Inclusion (EDI) considerations were integrated across three key stages: the design of the study (Stage 2), methodology and data collection (Stage 3), and analysis and interpretation (Stage 4).

During the study's design phase, we deliberately chose to utilize a widely used dataset that encompasses images from diverse sources and environments. This approach aimed to foster the development of a more generalized model capable of robust performance across various conditions. By adhering to this strategy, we aimed to ensure the reliability and generalizability of our testing results, paralleling the principles highlighted in Example 2 on the NSERC website.

In terms of methodology and data collection, we proactively sought to minimize resource inequities by prioritizing the use of less-utilized servers. We only resorted to the more powerful server when absolutely necessary, thereby allowing other researchers to access computing resources more readily. Additionally, we restricted our study to the use of a single GPU and one training session at a time, further promoting equity in resource access among researchers.

During the critical analysis and interpretation stage, we conducted thorough testing of our model across various datasets to evaluate its performance under diverse conditions. This comprehensive evaluation process, akin to the methodology outlined in Example 8 on the NSERC website, ensured the robustness and reliability of our findings, reinforcing the applicability of our research outcomes.

Conclusion

In this project, we devised a Mamba-based module tailored for image input and integrated it in lieu of SCA modules across both stages of DSRNet. Our investigation revealed the efficacy of Mamba when employed as an attention module, particularly in later network layers. Additionally, we explored the effects of using a CosineAnnealing learning rate schedule and AdamW with weight decay. Our findings suggest that this setup offers advantages for fully convolutional networks like DSRNet and Mamba-based models alike. Notably, we observed that Mamba demonstrates reduced sensitivity to changes in the learning rate compared to CNNs akin to DSRNet.

References

- [1] Hu, Q., & Guo, X. (2023). Single Image Reflection Separation via Component Synergy. 2023 IEEE/CVF International Conference on Computer Vision (ICCV), 13092-13101.
- [2] Chen, L., Chu, X., Zhang, X., & Sun, J. (2022). Simple Baselines for Image Restoration. European Conference on Computer Vision.
- [3] Dong, Z., Xu, K., Yang, Y., Bao, H., Xu, W., & Lau, R.W. (2020). Location-aware Single Image Reflection Removal. 2021 IEEE/CVF International Conference on Computer Vision (ICCV), 4997-5006.
- [4] Shi, W., Caballero, J., Huszár, F., Totz, J., Aitken, A.P., Bishop, R., Rueckert, D., & Wang, Z. (2016). Real-Time Single Image and Video Super-Resolution Using an Efficient Sub-Pixel Convolutional Neural Network. 2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR), 1874-1883.

- [5] Fan, Q., Yang, J., Hua, G., Chen, B., & Wipf, D.P. (2017). A Generic Deep Architecture for Single Image Reflection Removal and Image Smoothing. 2017 IEEE International Conference on Computer Vision (ICCV), 3258-3267.
- [6] Andriushchenko, M., D'Angelo, F., Varre, A., & Flammarion, N. (2023). Why Do We Need Weight Decay in Modern Deep Learning? ArXiv, abs/2310.04415.
- [7] Feng, X., Pei, W., Jia, Z., Zhang, D., & Lu, G. (2020). Deep-Masking Generative Network: A Unified Framework for Background Restoration From Superimposed Images. IEEE Transactions on Image Processing, 30, 4867-4882.
- [8] Loshchilov, I., & Hutter, F. (2016). SGDR: Stochastic Gradient Descent with Warm Restarts. arXiv: Learning.
- [9] Vaswani, A., Shazeer, N.M., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A.N., Kaiser, L., & Polosukhin, I. (2017). Attention is All you Need. Neural Information Processing Systems.
- [10] Dosovitskiy, A., Beyer, L., Kolesnikov, A., Weissenborn, D., Zhai, X., Unterthiner, T., Dehghani, M., Minderer, M., Heigold, G., Gelly, S., Uszkoreit, J., & Houlsby, N. (2020). An Image is Worth 16x16 Words: Transformers for Image Recognition at Scale. ArXiv, abs/2010.11929.
- [11] Hariharan, B., Arbeláez, P., Girshick, R.B., & Malik, J. (2014). Hypercolumns for object segmentation and fine-grained localization. 2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR), 447-456.
- [12] Hu, Q., & Guo, X. (2021). Trash or Treasure? An Interactive Dual-Stream Strategy for Single Image Reflection Separation. ArXiv, abs/2110.10546.
- [13] Zheng, Q., Shi, B., Chen, J., Jiang, X., Duan, L., & Kot, A.C. (2021). Single Image Reflection Removal with Absorption Effect. 2021 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR), 13390-13399.
- [14] Zhang, X.C., Ng, R., & Chen, Q. (2018). Single Image Reflection Separation with Perceptual Losses. 2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition, 4786-4794.
- [15] Kálmán, R.E. (1960). A new approach to linear filtering and prediction problems" transaction of the asme~journal of basic.
- [16] Zheng, Z., & Wu, C. (2024). U-shaped Vision Mamba for Single Image Dehazing. ArXiv, abs/2402.04139.
- [17] Levin, A., & Weiss, Y. (2004). User Assisted Separation of Reflections from a Single Image Using a Sparsity Prior. IEEE Transactions on Pattern Analysis and Machine Intelligence, 29.
- [18] Levin, A., Zomet, A., & Weiss, Y. (2002). Learning to Perceive Transparency from the Statistics of Natural Scenes. Neural Information Processing Systems.
- [19] Levin, A., Zomet, A., & Weiss, Y. (2004). Separating reflections from a single image using local features. Proceedings of the 2004 IEEE Computer Society Conference on Computer Vision and Pattern Recognition, 2004. CVPR 2004., 1, I-I.

- [20] Li, C., Yang, Y., He, K., Lin, S., & Hopcroft, J.E. (2019). Single Image Reflection Removal Through Cascaded Refinement. 2020 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR), 3562-3571.
- [21] Wen, Q., Tan, Y., Qin, J., Liu, W., Han, G., & He, S. (2019). Single Image Reflection Removal Beyond Linearity. 2019 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR), 3766-3774.
- [22] Yang, J., Gong, D., Liu, L., & Shi, J. (2018). Seeing Deeply and Bidirectionally: A Deep Learning Approach for Single Image Reflection Removal. European Conference on Computer Vision.
- [23] Li, Y., & Brown, M.S. (2014). Single Image Layer Separation Using Relative Smoothness. 2014 IEEE Conference on Computer Vision and Pattern Recognition, 2752-2759.
- [24] Li, Y., Liu, M., Yi, Y., Li, Q., Ren, D., & Zuo, W. (2020). Two-stage single image reflection removal with reflection-aware guidance. Applied Intelligence, 1-16.
- [25] Wei, K., Yang, J., Fu, Y., Wipf, D.P., & Huang, H. (2019). Single Image Reflection Removal Exploiting Misaligned Training Data and Network Enhancements. 2019 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR), 8170-8179.
- [26] Liu, Y., Tian, Y., Zhao, Y., Yu, H., Xie, L., Wang, Y., Ye, Q., & Liu, Y. (2024). VMamba: Visual State Space Model. ArXiv, abs/2401.10166.
- [27] Wang, Z., Zheng, J., Zhang, Y., Cui, G., & Li, L. (2024). Mamba-UNet: UNet-Like Pure Visual Mamba for Medical Image Segmentation. ArXiv, abs/2402.05079.
- [28] Wu, R., Liu, Y., Liang, P., & Chang, Q. (2024). H-vmunet: High-order Vision Mamba UNet for Medical Image Segmentation. ArXiv, abs/2403.13642.
- [29] Gu, A., Goel, K., & R'e, C. (2021). Efficiently Modeling Long Sequences with Structured State Spaces. ArXiv, abs/2111.00396.
- [30] Gu, A., & Dao, T. (2023). Mamba: Linear-Time Sequence Modeling with Selective State Spaces. ArXiv, abs/2312.00752.
- [31] Sinha, S.N., Kopf, J., Goesele, M., Scharstein, D., & Szeliski, R. (2012). Image-based rendering for scenes with reflections. ACM Transactions on Graphics (TOG), 31, 1 - 10.
- [32] Loshchilov, I., & Hutter, F. (2017). Decoupled Weight Decay Regularization. International Conference on Learning Representations.
- [33] Xing, Z., Ye, T., Yang, Y., Liu, G., & Zhu, L. (2024). SegMamba: Long-range Sequential Modeling Mamba For 3D Medical Image Segmentation. ArXiv, abs/2401.13560.