Visual Recognition Using Deep Learning Lab4 多工所碩一 313553024 蘇柏叡

---

github link: https://github.com/rmd926/NYCU_CV2025_Spring/tree/main/Lab4

1. Introduction

In this lab, our objective is to **perform blind image restoration on images degraded by rain and snow conditions.** Specifically, we aim to train a unified visual model capable of simultaneously addressing restoration tasks for these two distinct scenarios, using the **Peak Signal-to-Noise Ratio (PSNR)** as the primary evaluation metric. The dataset employed in our experiments consists of two categories—rain and snow—each containing 1,600 pairs of degraded and corresponding clean images, along with an additional 100 degraded images (50 from each category) without category labels reserved for testing purposes.

Regarding the model selection, following the given specifications, we adopted **PromptIR [1]**, a network that introduces a set of **learnable prompt vectors** designed to encode and distinguish characteristics of different image degradation types. Specifically, PromptIR associates each degradation scenario with a dedicated **prompt embedding**, which is integrated—either **concatenated or summed**—with intermediate-layer feature representations throughout the restoration network. This design allows the **same set of network weights** to dynamically adapt the restoration strategy based on the embedded prompts, enabling simultaneous handling of multiple unknown degradation scenarios such as **rain streaks and snow haze** within a unified framework.

Unlike previous assignments that I primarily focused on **architectural modifications**, the current experiment, due to hardware constraints, **shifts attention to the design and combination of loss functions (also meets the requirements of the additional experiments).** Specifically, we implemented several loss functions commonly employed in the **image restoration literature**, with a focus on their adaptability to PromptIR [1]. Although the original implementation of PromptIR utilizes **L1 Loss**, it is important to note that optimizing for **Peak Signal-to-Noise Ratio (PSNR)** inherently corresponds to minimizing **Mean Squared Error (MSE)**. However, due to **MSE's sensitivity to outliers**, directly using it may hinder convergence. Therefore, we adopt **Charbonnier Loss [2]** as a compromise, serving as the **baseline loss function** in our study. This loss offers a **smooth approximation of the L2 norm**, allowing the model to better **preserve edge structures** while **mitigating excessive penalization of outliers**. Additionally, the inclusion of a small constant **ε** ensures **well-behaved gradients near zero**, thereby promoting training stability and improving the quality of restoration outcomes.

In addition to the Charbonnier Loss [2] employed as the **baseline**, this study also implements the **Guided Frequency Loss (GFL) [3]** as an **alternative loss function** in our extended experiments. The **GFL [3]** integrates spatial and frequency domain supervision by combining three core components: the **Charbonnier component [2]**, the **Laplacian Pyramid component**, and the **Gradual Frequency component**. This composite structure enables the model to jointly optimize both **spatial texture consistency** and **spectral fidelity**, substantially enhancing the recovery of high-frequency image details. Beyond conducting multi-stage fine-tuning on the **baseline model** and comparing the effects of different loss functions, we further incorporate a **Test-Time Augmentation (TTA) [4]** strategy during inference. This TTA comprises **eight directional transformations**, including horizontal and vertical flips, as well as 90-degree rotations, aiming to assess its potential in boosting restoration performance.

Experimental results demonstrate that, even without TTA [4], both the **Charbonnier Loss [2]** and the **Guided Frequency Loss [3]**—after one training phase followed by our proposed **two-stage fine-tuning strategy**—consistently surpass the **strong baseline**. When TTA [4] is additionally applied, the average **PSNR is further improved by approximately 0.6 dB**, thereby validating the **effectiveness and generalizability of TTA [4]** in blind image restoration tasks.

2. Method

2.1 Data Preprocessing

(1) Data Loader

The image restoration dataset used in this experiment includes two degradation types: **Rain** and **Snow**, with **1,600 degraded–clean pairs** per type (3,200 total). The test set comprises **100 unlabeled degraded images**, evenly split between the two categories. To ensure **balanced validation**, we perform **stratified sampling** with a fixed seed—selecting **20% (160 images)** from each category for validation and using the remaining **80% (1,280 images)** for training. This yields a **320-image validation set**, effectively avoiding class imbalance.

(2) Data Augmentation

In the data augmentation stage, we implemented five techniques as follows:

a. Horizontal Flip and Vertical Flip:

In this experiment, random **horizontal and vertical flipping** is employed to enhance the model's **robustness to mirrored spatial variations** in objects or backgrounds. These geometric transformations **preserve relative pixel relationships**, generating left-right or top-bottom mirrored versions without altering the underlying content. As a result, the model is exposed to greater directional diversity during training while **maintaining consistency between degraded–clean image mappings**, thereby mitigating overfitting and improving generalization in **unseen orientations**.

b. Random 90° Multiple Rotations:

Random rotation by multiples of 90° enables the model to **adapt to orthogonal directional variations**, preserving the pixel grid structure while increasing training diversity through multi-view perspectives. This augmentation enhances the model's **robustness to rotational degradations**, leading to more stable restoration performance across varied orientations.

c. CutBlur [5]:

CutBlur [5] is a patch-level augmentation method tailored for image restoration. It randomly exchanges same-sized regions between degraded and clean image pairs, producing samples with **both degraded and sharp areas**. This preserves **spatial alignment** while introducing **diverse restoration difficulty**, allowing the model to **learn adaptive reconstruction** across regions. As a result, CutBlur improves the model's **precision in restoring local degradations** like raindrops or snow haze.

d. Mixup [6]:

Mixup [6] generates intermediate degradation levels by performing a linear interpolation between two degraded–clean image pairs using a shared random weight. This strategy produces **mixed samples** that smooth the model's **decision boundaries**, **mitigate overfitting**, and **enhance generalization** to diverse degradation distributions.

e. Blend [7]:

Blend [7] is a global photometric perturbation technique that applies random brightness or contrast shifts to the entire image without altering its geometric structure. By synchronously adding these perturbations to both degraded and clean inputs, **pixel-wise correspondence** is preserved, enhancing the model's **robustness to varying illumination conditions** during restoration.

2.2 Model Architecture

2.2.1 Overall Pipeline

In the overall pipeline, a degraded image $\mathbf{I} \in \mathbb{R}^{H\times W\times C}$ is first projected into a low-level feature space via a **3×3 convolution**, yielding the representation $\mathbf{F_0} \in \mathbb{R}^{H\times W\times C}$. These features are then processed by a **four-stage hierarchical encoder–decoder architecture**. The **encoder** progressively reduces spatial resolution and expands channel dimensions through multiple **Transformer Blocks** and downsampling operations. Conversely, the **decoder** reconstructs the resolution using **upsampling** and **symmetric skip connections**, ultimately recovering to the original resolution **(H×W)** from a compressed latent space of **(H/8)×(W/8)×8C**. Within the encoder, the number of **Transformer Blocks** increases from top to bottom to balance **computational efficiency and representational capacity**. During decoding, **Prompt Blocks** are inserted between every two stages, allowing degradation-specific **prompt information** to guide feature restoration. Finally, a **3×3 convolution** followed by **residual addition** produces the clean output image.

2.2.2 Prompt Block

The **Prompt Block**, designed as a pluggable module, is inserted between every two decoder stages to inject degradation-specific guidance into the restoration pipeline. It enables targeted modulation by **deeply integrating dynamically generated prompts with intermediate decoder features**. The block comprises two key submodules: the **Prompt Generation Module (PGM)** and the **Prompt Interaction Module (PIM)**.

(1) Prompt Generation Module (PGM)

The **Prompt Generation Module (PGM)** first applies global average pooling to the decoder features, extracting a channel-wise vector that summarizes the degradation characteristics while preserving spatial invariance. This vector is then passed through a lightweight point-wise convolution followed by a normalization and activation mechanism to produce **attention weights** over a set of learnable prompt components. Unlike static prompts, this attention-based strategy dynamically modulates each prompt according to the input content, ensuring **adaptive and context-aware guidance**. A subsequent spatial convolution and bilinear upsampling align the prompt tensor with the decoder feature resolution, preparing it for interaction via the next module.

(2) Prompt Interaction Module (PIM)

The **Prompt Interaction Module (PIM)** concatenates the prompts generated by PGM with the decoder features along the channel dimension, forming an enriched representation that encodes both spatial visual cues and degradation priors. This combined representation is processed by a **Transformer Block** comprising **Multi-Dconv Head Transposed Attention (MDTA)** and a **Gated-Dconv Feedforward Network (GDFN)**. The **MDTA** computes self-attention across channels with linear complexity, modeling inter-channel and prompt-feature interactions. Meanwhile, the **GDFN** employs depthwise separable convolutions and gating mechanisms to selectively amplify task-relevant signals while suppressing irrelevant noise. A final **3×3 convolution** maps the output back to the original channel dimension, yielding an updated, prompt-guided feature map for the next decoding stage. Overall, the **Prompt Block**, through dynamic prompt generation (PGM) and deep prompt-feature fusion (PIM), continuously injects task-specific guidance into the multi-level decoding pipeline, significantly enhancing the model's adaptability and restoration fidelity under complex degradations.

2.3 Modification of the loss function as the additional Experiment

Given that this experiment focuses on the **design and substitution of loss functions**, we observed that early trials with **L1 Loss** and **MSE Loss** yielded suboptimal results. Motivated by prior studies [2, 3], which demonstrate the consistent and superior performance of **Charbonnier Loss** [2] and **Guided Frequency Loss** [3] across various tasks and models, we adopted these alternatives. Accordingly, we **replaced the original L1 Loss in the training code** with the aforementioned losses and conducted comparative experiments. This section provides a detailed introduction to the design of **Charbonnier Loss** and **Guided Frequency Loss**.

2.3.1 Charbonnier Loss[2]

Charbonnier Loss [2] is a smooth approximation of pixel-wise error, commonly used to improve numerical stability and robustness. Its formulation is given in **Equation (1)** as follows:

$$\mathcal{L}_{\text{Charbonnier}}(\hat{y}, y) = \frac{1}{N}\sum_{i=1}^{N}\sqrt{(\hat{y}_i - y_i)^2 + \varepsilon^2},$$

In **Charbonnier Loss** [2],ŷ and y denote the predicted and ground truth pixel values, respectively, while εis a small constant (set to 1e-3 in this work) introduced to maintain smooth gradients near zero error. Compared to **MSE Loss**, Charbonnier Loss **[2]** avoids excessive penalization of outliers due to its sub-quadratic behavior, enhancing robustness. In contrast to **L1 Loss**, it provides smooth and continuous gradients near zero by incorporating ε^2 within the square root, avoiding the non-differentiability of L1 at the origin. This hybrid behavior—approximating **L1** in high-error regions and **L2** near zero—enables Charbonnier Loss**[2]** to better balance noise suppression and edge preservation, thereby improving the visual quality and stability of image restoration.

2.3.2 Guided Frequency Loss [3]

Guided Frequency Loss [3] is a unified loss function that simultaneously imposes constraints across three complementary domains: the **spatial error term** (Charbonnier Component [2]), the **multi-scale structural term** (Laplacian Pyramid Component), and the **frequency-guided term** (Gradual Frequency Component). By integrating these components into a single objective, the loss effectively guides the model to balance pixel-level fidelity, structural consistency across scales, and progressive recovery of high-frequency details. Its formulation is given in Equation (2).

$$GFL\ (I_R,\ I_{HQ}) = \sqrt{Ch_C + \Pi_C\ + \Theta_C}$$

(1) Charbonnier Component [2] (ChC)

As discussed in the previous section, the **Charbonnier Component** [2] leverages a hybrid behavior—approximating **L1** loss in large-error regions and **L2** loss near zero—to achieve a more balanced trade-off between denoising and edge preservation. This design enhances both the **visual fidelity** and **training stability** of the image restoration process.

(2) Laplacian Pyramid Component (ΠC)

The **Laplacian Pyramid Component** decomposes an image into **multi-scale residual representations** through a sequence of **downsampling → difference computation**

→ **upsampling** operations. Each level isolates **structural and textural details** at a specific resolution. Specifically, both the **restored** and **ground truth images** are transformed into **Laplacian pyramids**, and **Charbonnier loss** is computed at each scale between corresponding residuals. This **multi-scale supervision** not only reinforces **pixel-level consistency** at the original resolution but also enhances recovery of **coarse shapes**, **mid-level structures**, and **fine textures** simultaneously—achieving an **optimal balance** between **edge sharpness** and **local detail fidelity**. The Equation (3) is:

$$\Pi_C = \|\Pi(I_R) - \Pi(I_{HQ})\|^2$$

(3) Gradual Frequency Component (ΘC)

In **GFL Loss**, the **ΘC component** is designed to **progressively guide the model in learning and restoring high-frequency details**. The Equation (4) is:

$$\Theta_C = \|\Theta(I_R) - \Theta(I_{HQ})\|^2$$

In this formulation, IR and IHQ represent the model output and ground-truth high-quality image, respectively, while **Θ** denotes a **high-pass filter with a dynamically adjustable passband** throughout training. **Initially**, the filter allows only the **most extreme high-frequency components**—as defined by the starting threshold **ω_0**—to pass. As training progresses, or when the **GFL loss drops below a predefined threshold**, the passband is **gradually expanded toward lower frequencies**, ultimately reaching a final threshold **ωF**, which includes all high-frequency content.

This **progressive expansion** can be scheduled either via a **static strategy** (based on epoch counts) or a **dynamic strategy** (based on loss convergence). The **ΘC component computes the squared difference** between the filtered outputs of the restored and ground-truth images, thereby **focusing supervision on reconstruction errors within the current frequency band**. Since **high-frequency details are inherently harder to recover**, this **gradual spectrum-guided mechanism** enables the model to first **stabilize its learning on sharp textures and edges**, and then **progressively adapt to broader frequency ranges**, leading to **improved precision and robustness** in recovering structural and textural fidelity in image restoration.

(4) Summary

In summary, **Guided Frequency Loss (GFL)** achieves joint optimization across the **pixel**, **structural**, and **frequency domains** through the synergy of its three core components,

thereby significantly improving both the **stability** and **accuracy** of image restoration. The **Charbonnier loss**, as a smooth approximation of the L2 norm, provides a **robust pixel-wise residual estimation**, effectively suppressing the influence of outliers. The **Laplacian Pyramid Component** enforces **multi-scale structural consistency** by comparing residuals between the restored and ground truth images at coarse, mid-level, and fine resolutions. Meanwhile, the **Gradual Frequency Component (ΘC)** introduces high-frequency information **progressively** via a **dynamic high-pass filter**, beginning with extremely high-frequency details and expanding its passband as training proceeds or as loss converges. This enables the model to **learn edge and texture features more effectively**. Collectively, these components empower the network to attain **superior reconstruction quality across diverse degradation scenarios**, such as denoising, deraining, and desnowing.

(5) The parameter design

The hyperparameter settings for the GFL Loss [3] are summarized in Figure 1. The smoothing factor $\varepsilon = 1 \times 10^{-3}$ is incorporated into the Charbonnier component [2] to regularize the residual calculation by adding a small constant, thereby preventing gradient explosion or vanishing as the residual approaches zero and enhancing numerical stability. At the outset of training, the Gradual Frequency term is constrained to only the highest frequency components above an initial threshold $\omega_0 = 0.1$, encouraging the model to prioritize learning fine edge details. As training progresses, the high-pass filter's passband is incrementally expanded from ω_0 to a final threshold $\omega\_F = 0.5$, ensuring comprehensive coverage of all critical high-frequency information and facilitating complete detail recovery. Under static scheduling (static = True), with num_epochs and num_stages = 5, the lower bound of the passband is elevated by one sub-band every (num_epochs/num_stages) epochs, implementing a spectrum-progressive learning strategy that is solely determined by epoch count and stage division, and remains unaffected by loss convergence dynamics throughout training.

```
criterion = GFLLoss(
    eps=1e-3,
    omega0=0.1,
    omegaF=0.5,
    num_epochs=args.epochs,
    num_stages=5,
    static=True
)
```

Fig 1. The snapshot of Hyperparameters of GFL Loss settings

2.4 Hyperparameters Settings and Training Configurations

2.4.1 Hyperparameters Settings

Given that the original experimental setting involves cropping 256×256 images into 128×128 patches with a **batch size of 32**, the effective input per iteration is equivalent to **eight full-resolution images**. However, in this study, we aim to train using the **full-resolution images**, and due to GPU constraints (RTX 4060-Ti 16GB), the **batch size is reduced to 2**. To mitigate this limitation, we apply **gradient accumulation**, enabling an effective batch size of 8 and preserving the computational behavior of larger batches. Moreover, the training pipeline adopts a **one-stage training followed by a two-stage fine-tuning** strategy. Both the training and fine-tuning phases utilize a **Cosine Annealing Learning Rate Scheduler with Warmup**, allowing for stable convergence and improved performance. In the following sections, we detail the configuration of training stages and the corresponding **hyperparameter settings**.

(1) Training Phase:

During training, we adopt the **hyperparameter configuration** summarized in **Table 1**. Regarding the **weight decay** strategy, we do not apply uniform regularization across all trainable parameters. Instead, we selectively apply a **weight decay of 1e-4** to components such as **convolutional and linear layers**, while **excluding bias terms and normalization-related parameters**. This design choice is motivated by the need to **avoid imposing excessive L2 constraints** on sensitive scaling parameters in **bias and normalization layers**, thereby **preserving their flexibility and stability** during gradient updates.

Hyperparameter	Value
Batch size	**8**
Initial learning rate (LR)	2×10^{-4}
η(min)	1×10^{-6}
Epochs & Warmup Epochs	(250, 10)
Weight decay	1×10^{-4}
Optimizer	AdamW

Table 1. Hyperparameters Settings of Training Phase

(2) First stage Fine-Tuning Phase:

In the **first-stage fine-tuning**, we adopt the **hyperparameter configuration** summarized in **Table 2**. Unlike the initial training phase, this stage **disables aggressive data augmentations**, retaining only **flipping and rotation**. Additionally, we reduce the **total number of epochs to 100**, lower the **learning rate to 1.5×10^{-4}**, and set the **minimum learning rate η_{min} to 5×10^{-6}**, reflecting a more conservative optimization strategy.

Hyperparameter	Value
Batch size	**8**
Initial learning rate (LR)	1.5×10^{-4}
η(min)	5×10^{-6}
Epochs & Warmup Epochs	(100, 5)
Weight decay	1×10^{-4}
Optimizer	AdamW

Table 2. Hyperparameters Settings of First stage Fine-Tuning Phase

(3) Second stage Fine-Tuning Phase:

Distinct from the training and first-stage fine-tuning phases, the **second-stage fine-tuning** freezes most components of the **PromptIR [1]** architecture, allowing **gradient updates only for six specific submodules**: **refinement**, **output**, **decoder_level1**, **noise_level1**, **reduce_noise_level1**, and the **upsampling branch up2_1** (including the associated convolution and **PixelShuffle** layers). The corresponding **hyperparameter configuration** is detailed in **Table 3**.

Hyperparameter	Value
Batch size	**8**
Initial learning rate (LR)	1.5×10^{-4}
η(min)	5×10^{-6}
Epochs & Warmup Epochs	(100, 5)
Weight decay	1×10^{-4}
Optimizer	AdamW

Table 3. Hyperparameters Settings of First stage Fine-Tuning Phase

3. Results & Additional experiments

3.1 The baseline experiment

(1) Using Charbonnier Loss[2] as the loss function of PromptIR[1] w/o Fine-Tuning

In this experiment, we adhered to the **hyperparameter configuration outlined in the previous section**. The **training and validation loss curves** exhibit a **smooth convergence**, with the **training loss dropping below the validation loss around epoch 100**, eventually stabilizing at approximately **0.019** and **0.023**, respectively. According to the **validation PSNR curve**, the PSNR surpassed **29 dB at epoch 100**, reaching a peak of **29.534 dB** near epoch 210. During inference without **TTA [4]**, we submitted the results to **CodaBench**, achieving a **public score of 30.093 dB** and a **private score of 29.508 dB**, which **did not surpass the strong baseline** of 30 dB. However, with **TTA [4] enabled during inference**, the same model weights yielded a **public score of 30.618 dB** and a **private score of 30.001 dB**, **narrowly surpassing the strong baseline**.

{"private_psnr": 29.50838244629658, "public_psnr": 30.09308865966024}

Fig 2. The PSNR score using Charbonnier Loss[2] as the loss function of PromptIR[1] w/o Fine-Tuning w/o TTA[4].

{"private_psnr": 30.00112613944801, "public_psnr": 30.618397681629034}

Fig 3. The PSNR score using Charbonnier Loss[2] as the loss function of PromptIR[1] w/o Fine-Tuning w/ TTA[4].

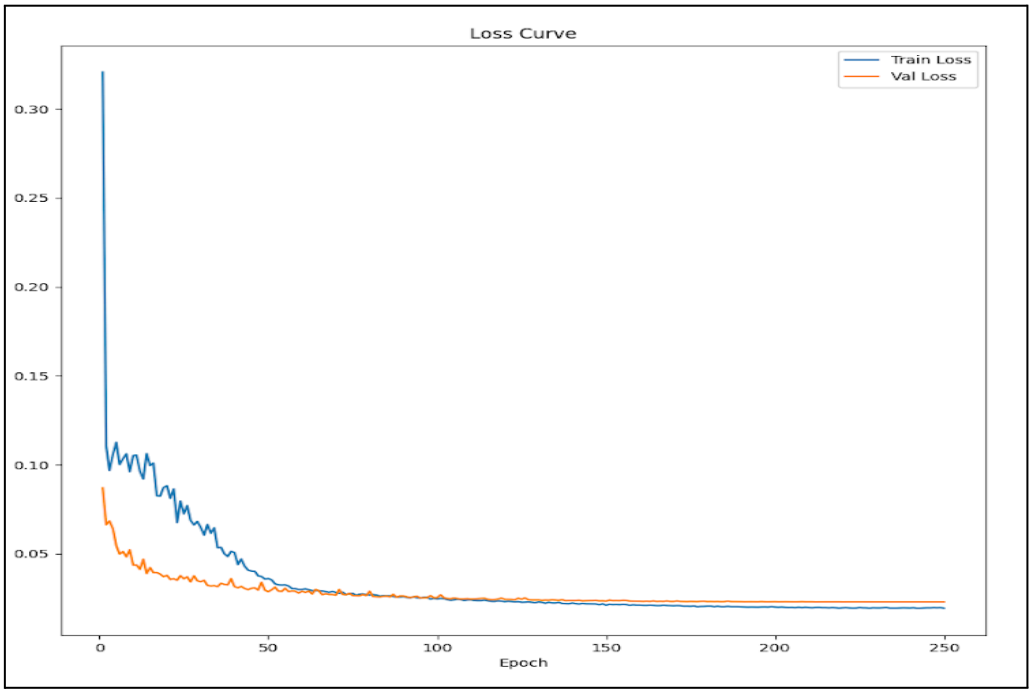

Fig 4. The plot of Training and Validation Loss using Charbonnier Loss[2] as the loss function of PromptIR[1] w/o Fine-Tuning.

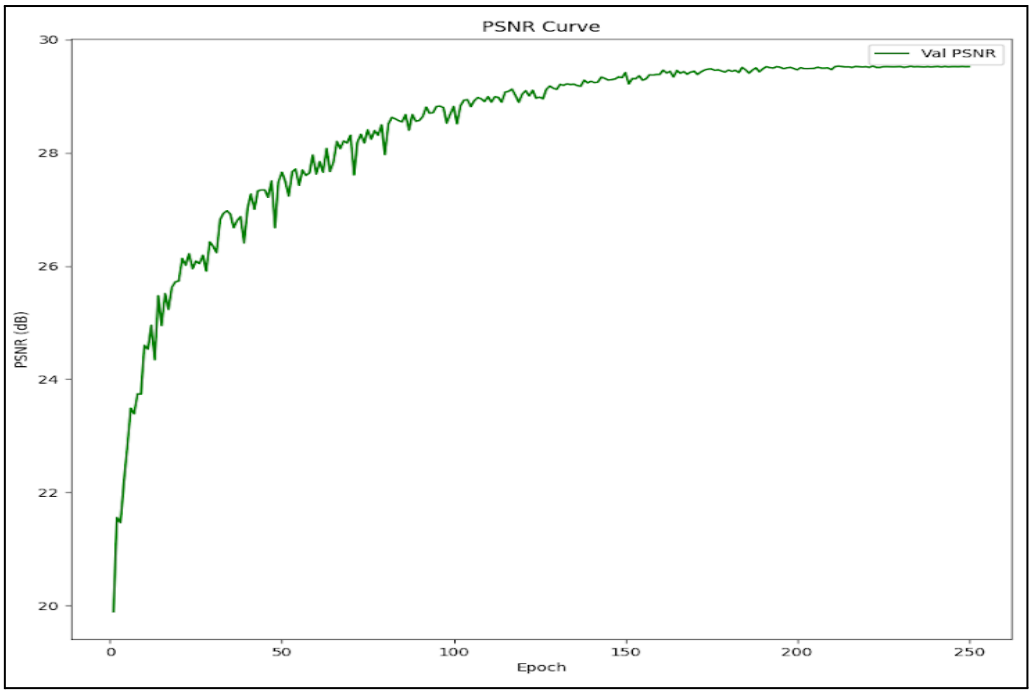

Fig 5. The plot of Validation PSNR using Charbonnier Loss[2] as the loss function of PromptIR[1] w/o Fine-Tuning.

(2) Using Charbonnier Loss[2] as the loss function of PromptIR[1] w/ First stage Fine-Tuning

In this fine-tuning experiment, we initialized the model with the **previously best-performing checkpoint** and **disabled high-intensity augmentation methods**, retaining only **flip and rotation transformations**. We followed the **hyperparameter settings described in the previous section**. Initially, the **validation loss exhibited substantial fluctuation**, but began to **stabilize and decline after epoch 60**, ultimately converging to **approximately 0.022**. Meanwhile, the **training loss decreased more rapidly**, settling around **0.017**, suggesting a **potential risk of overfitting** despite continued validation improvement.

As shown in the **validation PSNR curve**, the model surpassed the **PSNR of 29.534 dB** as early as **epoch 30**, though the curve remained somewhat oscillatory. The **best PSNR of 29.831 dB** was recorded at **epoch 77**. During inference **without TTA [4]**, the model achieved a **public score of 30.369 dB** and a **private score of 29.833 dB** on **CodaBench**, which **did not surpass the strong baseline** of 30 dB. However, with **TTA [4] enabled**, the same weights yielded a **public score of 30.960 dB** and a **private score of 30.385 dB**, thus **successfully exceeding the strong baseline**.

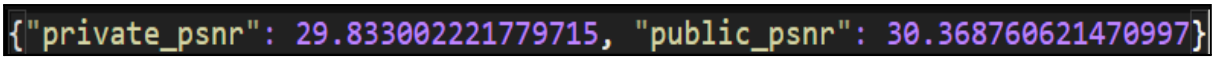

Fig 6. The PSNR score using Charbonnier Loss[2] as the loss function of PromptIR[1] w/ First stage Fine-Tuning and w/o TTA[4].

Fig 7. The PSNR score using Charbonnier Loss[2] as the loss function of PromptIR[1] w/ First stage Fine-Tuning and w/ TTA[4].

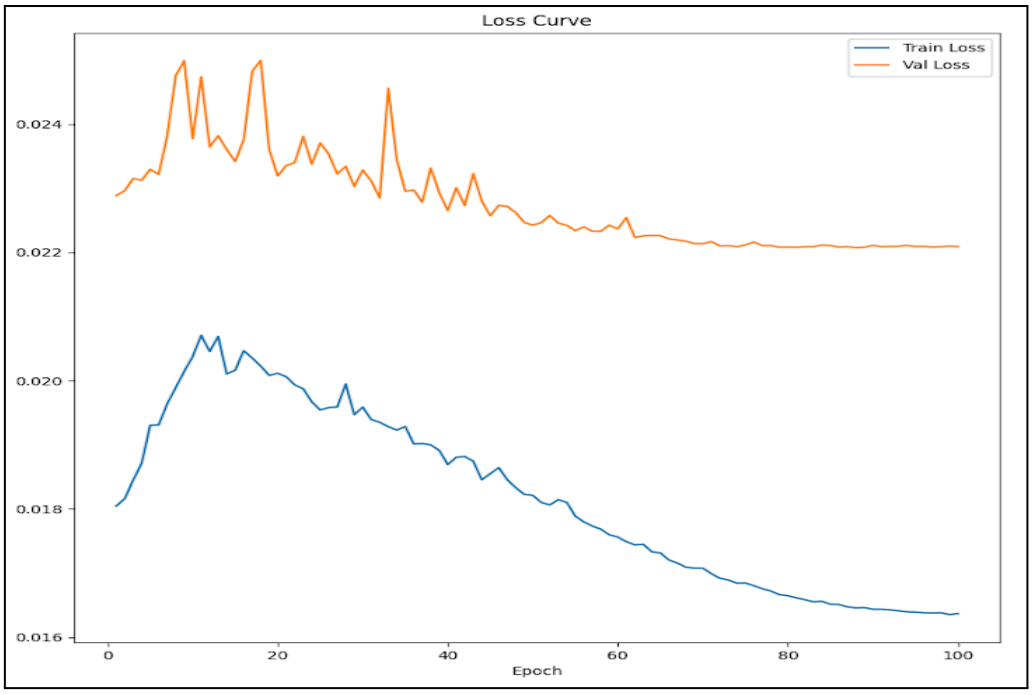

Fig 8. The plot of Training and Validation Loss using Charbonnier Loss[2] as the loss function of PromptIR[1] w/ First stage Fine-Tuning.

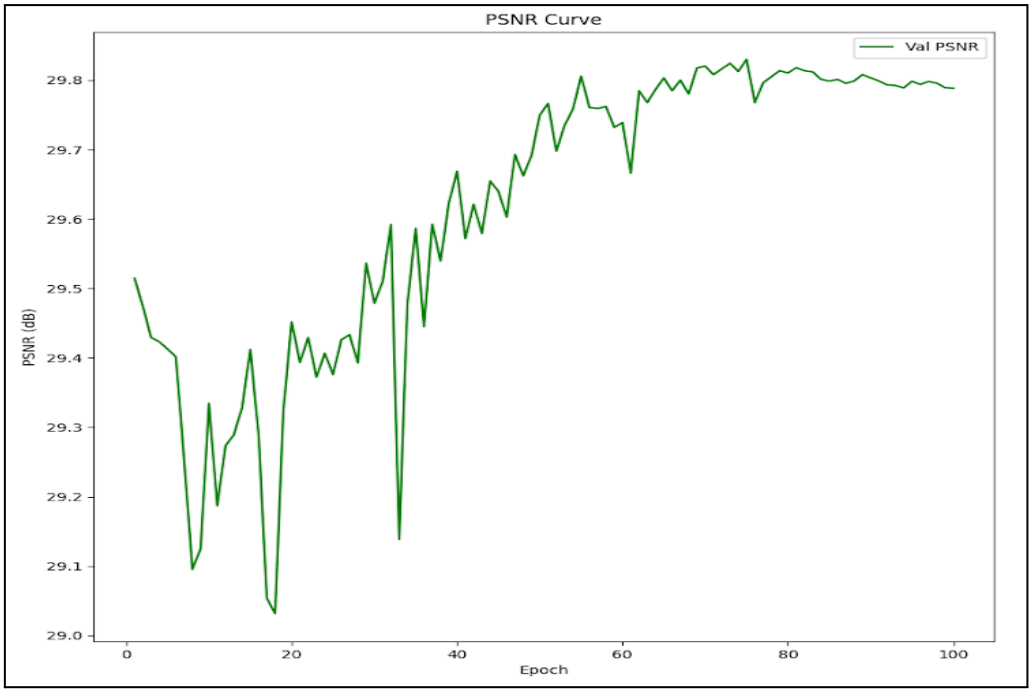

Fig 9. The plot of Validation PSNR using Charbonnier Loss[2] as the loss function of PromptIR[1] w/ First stage Fine-Tuning.

(3) Using Charbonnier Loss[2] as the loss function of PromptIR[1] w/ Second stage Fine-Tuning

In the **second-stage fine-tuning** experiment, we initialized the model with the **best checkpoint obtained from the first-stage fine-tuning**. While continuing to **disable high-intensity data augmentations**, we **retained only basic flip transformations**. Furthermore, we **froze the majority of the PromptIR [1] architecture**, selectively **enabling gradient updates** for only the following **six submodules**: **refinement**, **output**, the **first decoder layer (decoder_level1)**, the **first noise branch (noise_level1)**, the **first denoising reduction branch (reduce_noise_level1)**, and the **upsampling pathway up2_1**, which includes the associated **convolution and PixelShuffle layers**.

Subsequently, we performed fine-tuning according to the **hyperparameter settings outlined in the previous section**. As illustrated by the loss curves, the **validation loss exhibited notable fluctuations during the early training epochs**, but began to **stabilize and decline around epoch 50**, eventually **converging smoothly** by the end of the fine-tuning process. Following **second-stage fine-tuning**, the **training loss converged to approximately 0.0165**, while the **validation loss settled around 0.0215**.From the PSNR curve on the validation set, it is evident that by **epoch 30**, the model **surpassed the first-stage best PSNR of 29.831**, despite minor oscillations. The **peak PSNR of 29.949** was achieved at **epoch 97**. During inference without **Test-Time Augmentation (TTA) [4]**, the model achieved a **public score of approximately 30.539 dB** and a **private score of 29.963 dB** on CodaBench, **slightly below the strong baseline of 30.0 dB**. However, after applying **TTA during inference**, the same model weights yielded a **public score of 31.123 dB** and a **private score of 30.517 dB**, **successfully surpassing the strong baseline**.

Fig 10. The PSNR score using Charbonnier Loss[2] as the loss function of PromptIR[1] w/ Second stage Fine-Tuning and w/o TTA[4].

Fig 11. The PSNR score using Charbonnier Loss[2] as the loss function of PromptIR[1] w/ Second stage Fine-Tuning and w/ TTA[4].

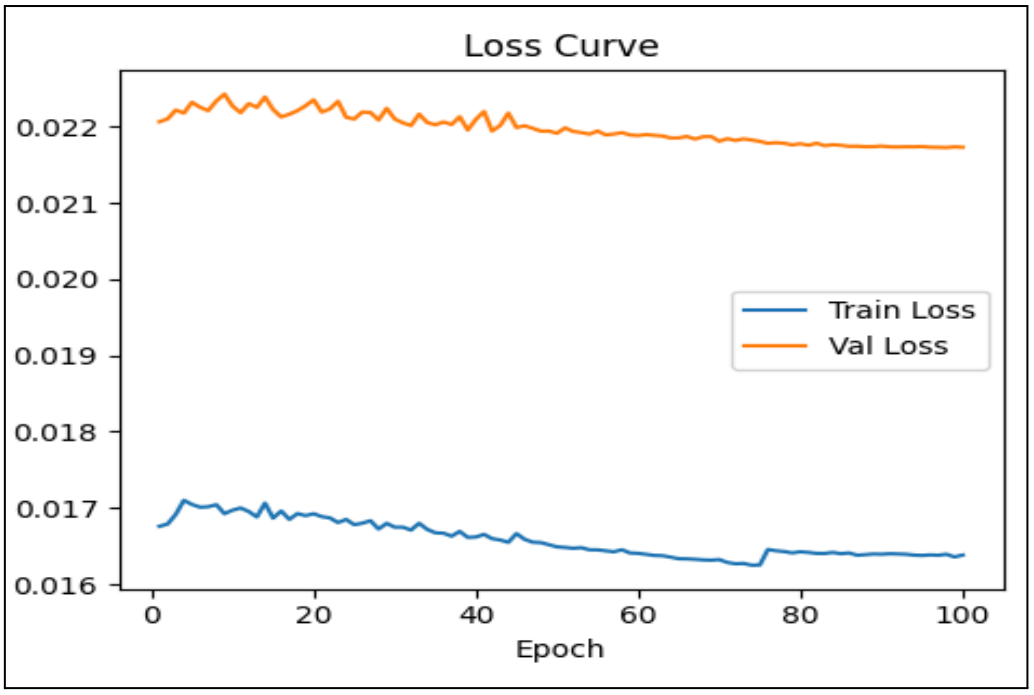

Fig 12. The plot of Training and Validation Loss using Charbonnier Loss[2] as the loss function of PromptIR[1] w/ Second stage Fine-Tuning.

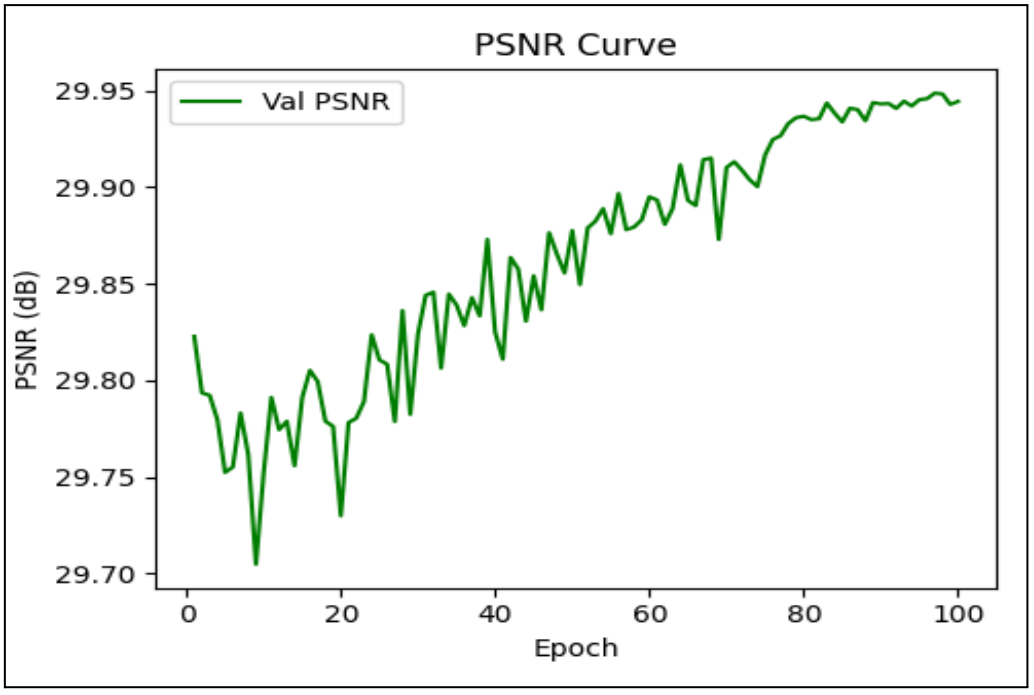

Fig 13. The plot of Validation PSNR using Charbonnier Loss[2] as the loss function of PromptIR[1] w/ Second stage Fine-Tuning.

(4) Using Charbonnier Loss[2] as the loss function of PromptIR[1] w/ Second stage Fine-Tuning [Fine-Tuning Again]

Building upon the promising results observed in the **Second-Stage Fine-Tuning**, where **validation loss continued to decrease** and **PSNR steadily improved** near the end of training, **we extended the fine-tuning process using the same methodology.** As illustrated by the **loss curves**, the **validation loss exhibited noticeable fluctuations during the early epochs**, but began to **stabilize around epoch 50** and **converged smoothly by the end** of training. The final **training loss converged to approximately 0.0161**, and the **validation loss settled at 0.0213**. From the **validation PSNR curve**, the PSNR exceeded the previous best of **29.949** by **epoch 30**, despite some oscillations. The experiment ultimately reached a **new best PSNR of 30.242 at epoch 97**.

When evaluated on CodaBench without **Test-Time Augmentation (TTA) [4]**, the model achieved a **public score of 30.626 dB** and a **private score of 30.042 dB**, **finally surpassing the strong baseline of 30.0 dB**. Furthermore, with **TTA enabled during inference**, the same weights yielded a **public score of 31.204 dB** and a **private score of 30.586 dB**, thereby **consistently outperforming the baseline**.

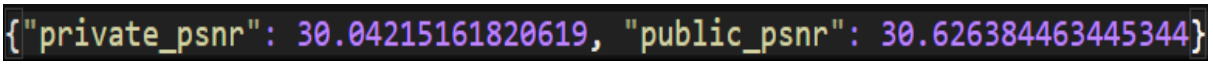

Fig 14. The PSNR score using Charbonnier Loss[2] as the loss function of PromptIR[1] w/ Second stage Fine-Tuning [Fine-Tuning Again] and w/o TTA[4].

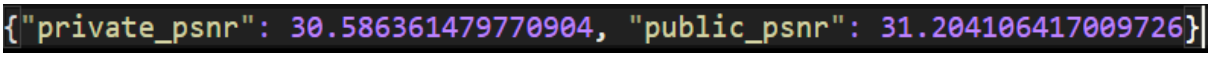

Fig 15. The PSNR score using Charbonnier Loss[2] as the loss function of PromptIR[1] w/ Second stage Fine-Tuning [Fine-Tuning Again] and w/ TTA[4].

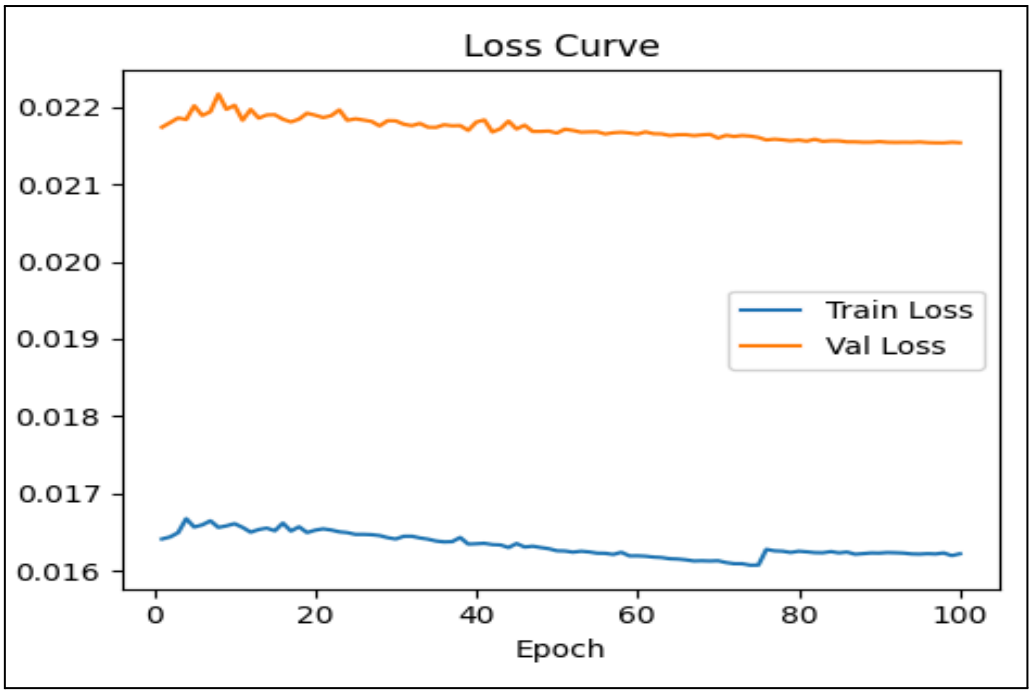

Fig 16. The plot of Training and Validation Loss using Charbonnier Loss[2] as the loss function of PromptIR[1] w/ Second stage Fine-Tuning.

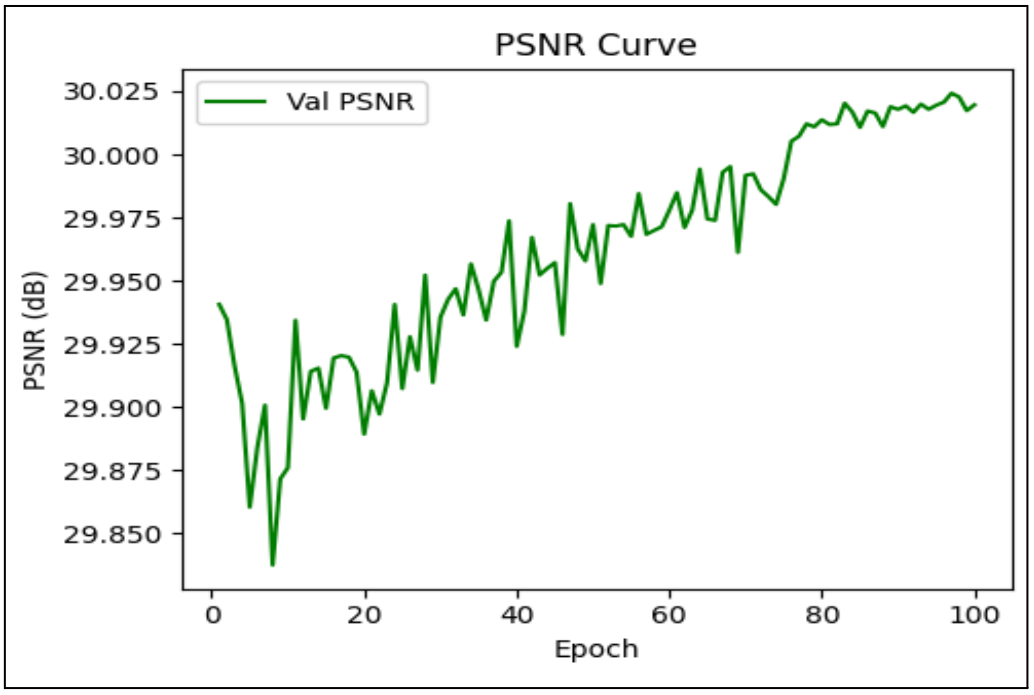

Fig 17. The plot of Validation PSNR using Charbonnier Loss[2] as the loss function of PromptIR[1] w/ Second stage Fine-Tuning.

3.2 Visualization of baseline Experiment Results

In this section, we showcase two snow and two rain examples: the top row displays the degraded inputs, while the bottom row shows their restored outputs produced by our trained model. These four image pairs correspond to the four **distinct Charbonnier-loss configurations [2] evaluated in our earlier experiments (in 3.1 section).**

I. **Using Charbonnier Loss[2] as the loss function of PromptIR[1] w/o Fine-Tuning and got a 30.001 PSNR in private score.**

Fig 18. Visualization of the baseline result (Charbonnier Loss[2]) w/o Fine-Tuning

II. **Using Charbonnier Loss[2] as the loss function of PromptIR[1] w/ First stage Fine-Tuning and got a 30.385 PSNR in private score.**

Fig 19. Visualization of the baseline result (Charbonnier Loss[2]) w/ First stage Fine-Tuning

III. Using Charbonnier Loss[2] as the loss function of PromptIR[1] w/ Second stage Fine-Tuning and got a 30.517 PSNR in private score.

Fig 20. Visualization of the baseline result (Charbonnier Loss[2]) w/ Second stage Fine-Tuning

IV. Using Charbonnier Loss[2] as the loss function of PromptIR[1] w/ Second stage Fine-Tuning [Fine-Tuning Again] and got a 30.586 PSNR in private score.

Fig 21. Visualization of the baseline result (Charbonnier Loss[2]) w/ Second stage Fine-Tuning [Fine-Tuning Again]

3.3 The additional experiment (which replaced the Charbonnier Loss[2] with Guided Frequency Loss [3] to meet the requirement of the spec)

(1) Using GFL Loss[3] as the loss function of PromptIR[1] w/o Fine-Tuning

In this experiment, we **followed the hyperparameter configuration described in the previous section**. As shown in the **loss curves**, the training process exhibited **smooth convergence**, with the **training loss dropping below the validation loss around epoch 95**. Both losses continued to decline, with the **training loss ultimately converging to approximately 0.027** and the **validation loss to 0.031**.

The **PSNR curve on the validation set** showed that the **PSNR surpassed 29 dB at epoch 80**, and eventually reached a **peak value of 29.620 at epoch 193**, marking the **best PSNR in this setting**. During inference, we first submitted results to CodaBench **without applying Test-Time Augmentation (TTA) [4]**, yielding a **public score of 30.124 dB** and a **private score of 29.556 dB**, which **did not surpass the strong baseline of 30 dB**. However, after **applying TTA** and re-submitting with the **same model weights**, the results improved to a **public score of 30.734 dB** and a **private score of 30.229 dB**, thereby **successfully surpassing the strong baseline**.

Fig 22. The PSNR score using GFL Loss[3] as the loss function of PromptIR[1] w/o Fine-Tuning w/o TTA[4].

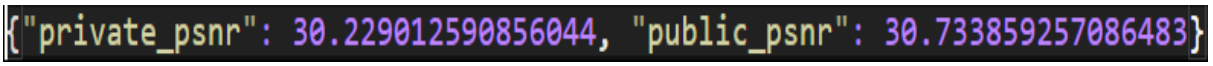

Fig 23. The PSNR score using GFL Loss[3] as the loss function of PromptIR[1] w/o Fine-Tuning w/ TTA[4].

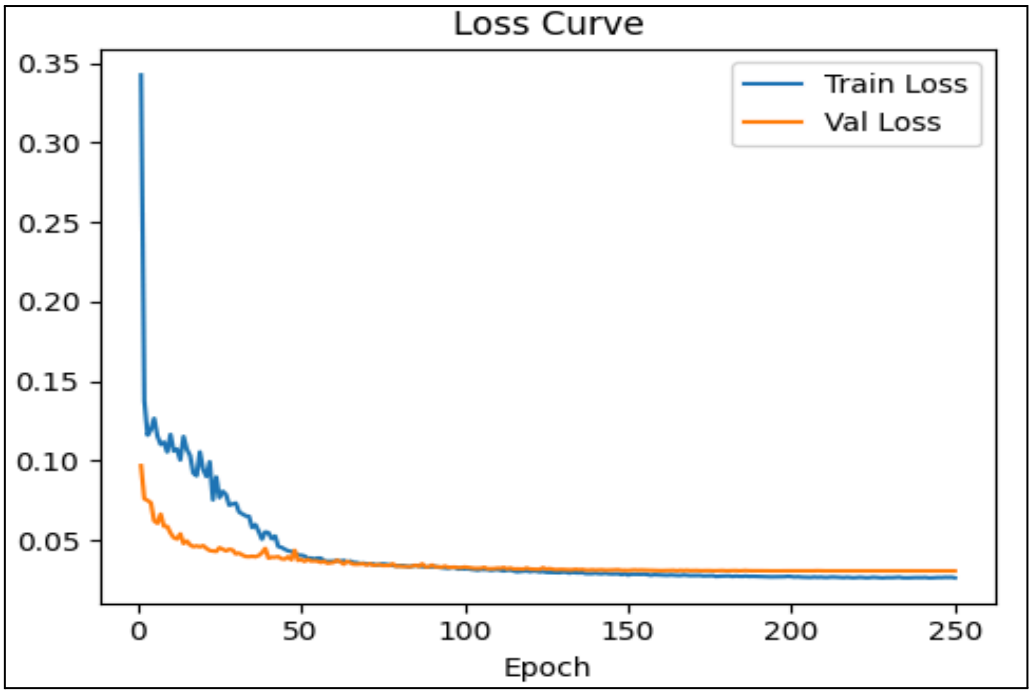

Fig 24. The plot of Training and Validation Loss using GFL Loss[3] as the loss function of PromptIR[1] w/o Fine-Tuning.

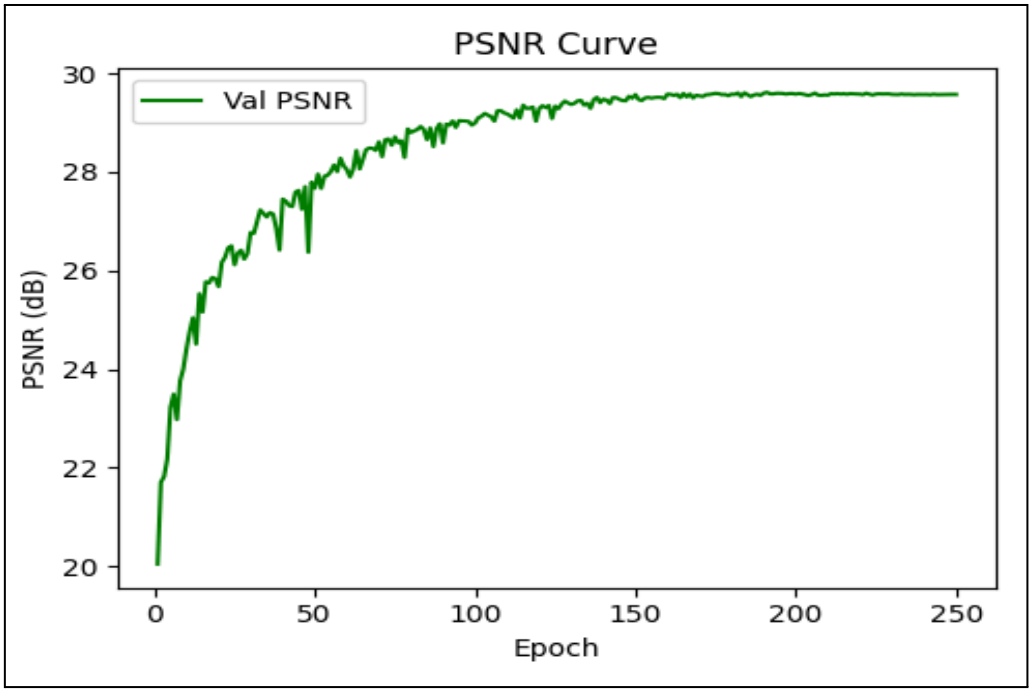

Fig 25. The plot of Validation PSNR using GFL Loss[3] as the loss function of PromptIR[1] w/o Fine-Tuning.

(2) Using GFL Loss[3] as the loss function of PromptIR[1] w/ First stage Fine-Tuning

In this **fine-tuning experiment**, we utilized the **best-performing checkpoint from the previous stage** and **disabled aggressive augmentation strategies**, retaining only **image flipping**. The model was fine-tuned according to the **hyperparameter configuration described in the previous section**. From the **training curves**, we observed that the **validation loss exhibited significant fluctuation during the early phase**, but began to **stabilize and decrease after epoch 40**, ultimately **converging to approximately 0.0303**. Meanwhile, the **training loss decreased more rapidly after epoch 50**, and although the **validation loss continued to decline**, there were signs of a **slight overfitting trend**, particularly beyond **epoch 75**. The **PSNR curve on the validation set** showed that the **PSNR surpassed the pre-fine-tuning level of 29.620 as early as epoch 17**, and ultimately **reached a peak of 29.879 at epoch 63**, marking the **best result in this phase**.

For inference, we first evaluated the model **without applying Test-Time Augmentation (TTA) [4]**, which yielded a **public PSNR of 30.407** and a **private PSNR of 29.878**, **failing to exceed the strong baseline of 30.0 dB**. However, after **enabling TTA and using the same model weights**, we achieved a **public PSNR of 31.069** and a **private PSNR of 30.491**, thereby **successfully surpassing the strong baseline**.

Fig 26. The PSNR score using GFL Loss[3] as the loss function of PromptIR[1] w/ First stage Fine-Tuning and w/o TTA[4].

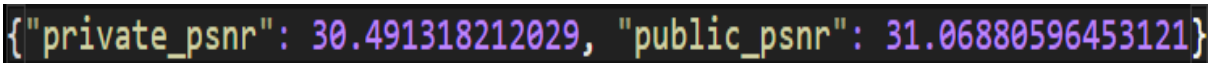

Fig 27. The PSNR score using GFL Loss[3] as the loss function of PromptIR[1] w/ First stage Fine-Tuning and w/ TTA[4].

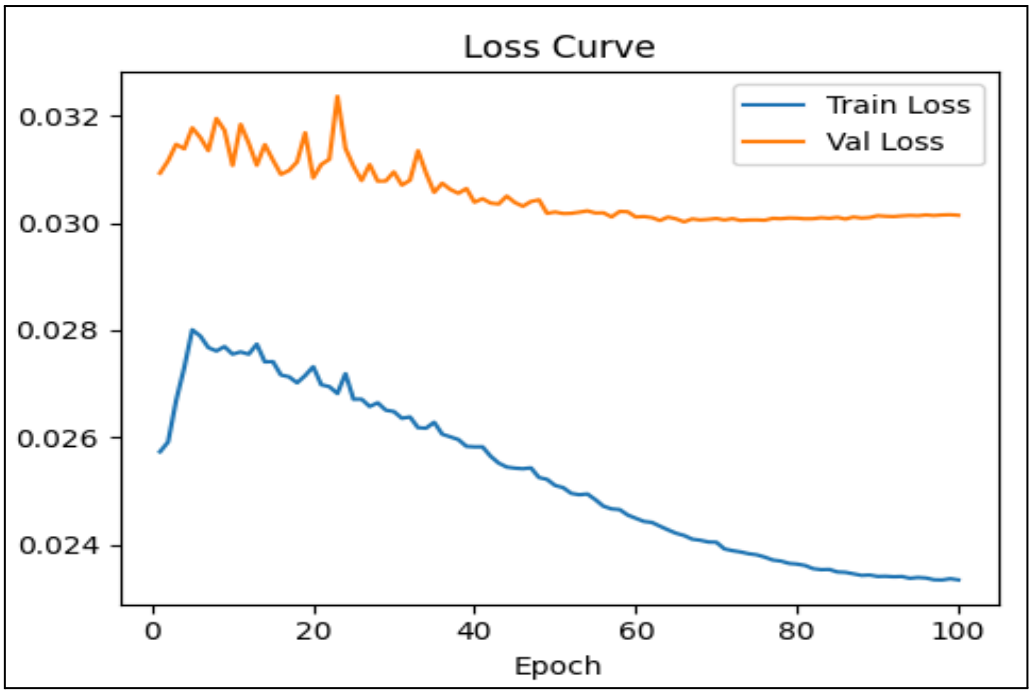

Fig 28. The plot of Training and Validation Loss using GFL Loss[3] as the loss function of PromptIR[1] w/ First stage Fine-Tuning.

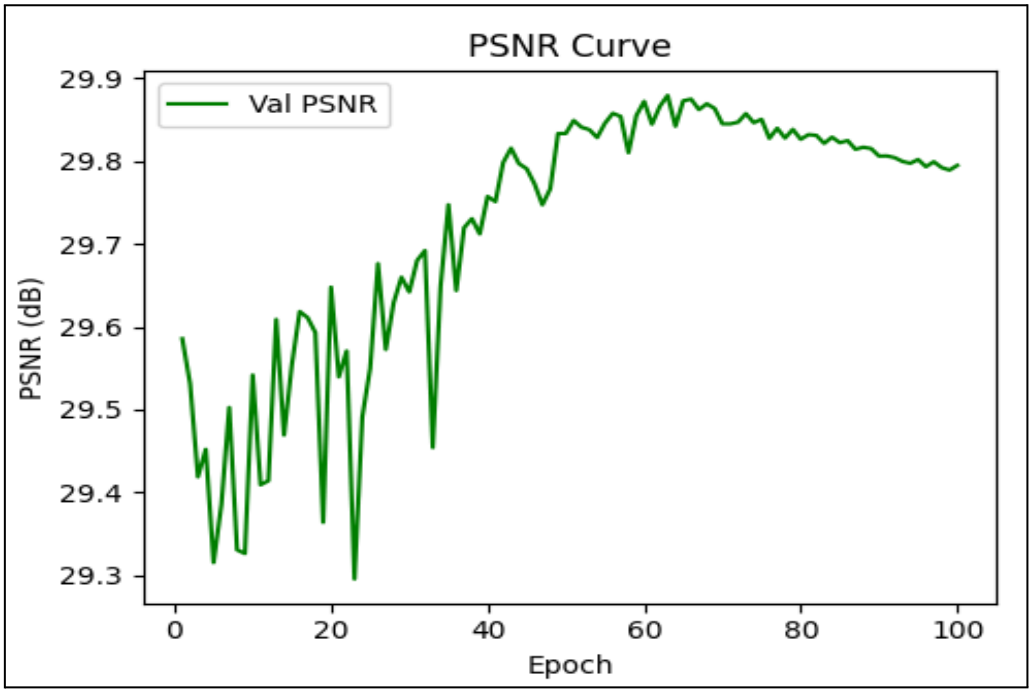

Fig 29. The plot of Validation PSNR using GFL Loss[3] as the loss function of PromptIR[1] w/ First stage Fine-Tuning.

(3) Using GFL Loss[3] as the loss function of PromptIR[1] w/ Second stage Fine-Tuning

In the Second Stage Fine-Tuning experiment, we adopted the best-performing checkpoint obtained from the First Stage Fine-Tuning. In this phase, strong data augmentations were disabled, retaining only image flipping, and most of the PromptIR [1] architecture was frozen. Only six submodules were unfrozen for gradient updates: refinement, output, decoder_level1, noise_level1, reduce_noise_level1, and the up2_1 upsampling branch, including both the convolutional and PixelShuffle layers. Following the predefined hyperparameter settings, fine-tuning was conducted. The training loss reached approximately 0.0237, and the validation loss converged to around 0.0296, with early fluctuations stabilizing after epoch 50. The validation PSNR surpassed the First Stage result (29.879) at epoch 27 and peaked at 29.995 by epoch 95.

For inference, we first evaluated the model **without applying Test-Time Augmentation (TTA) [4]**, which resulted in a **public PSNR of 30.516** and a **private PSNR of 30.0279**, **slightly surpassing the strong baseline of 30.0 dB**. After applying **TTA [4]**, the same model checkpoint achieved a **public PSNR of 31.185** and a **private PSNR of 30.630**, thereby **significantly outperforming the strong baseline**.

Fig 30. The PSNR score using GFL Loss[3] as the loss function of PromptIR[1] w/ Second stage Fine-Tuning and w/o TTA[4].

Fig 31. The PSNR score using GFL Loss[3] as the loss function of PromptIR[1] w/ Second stage Fine-Tuning and w/ TTA[4].

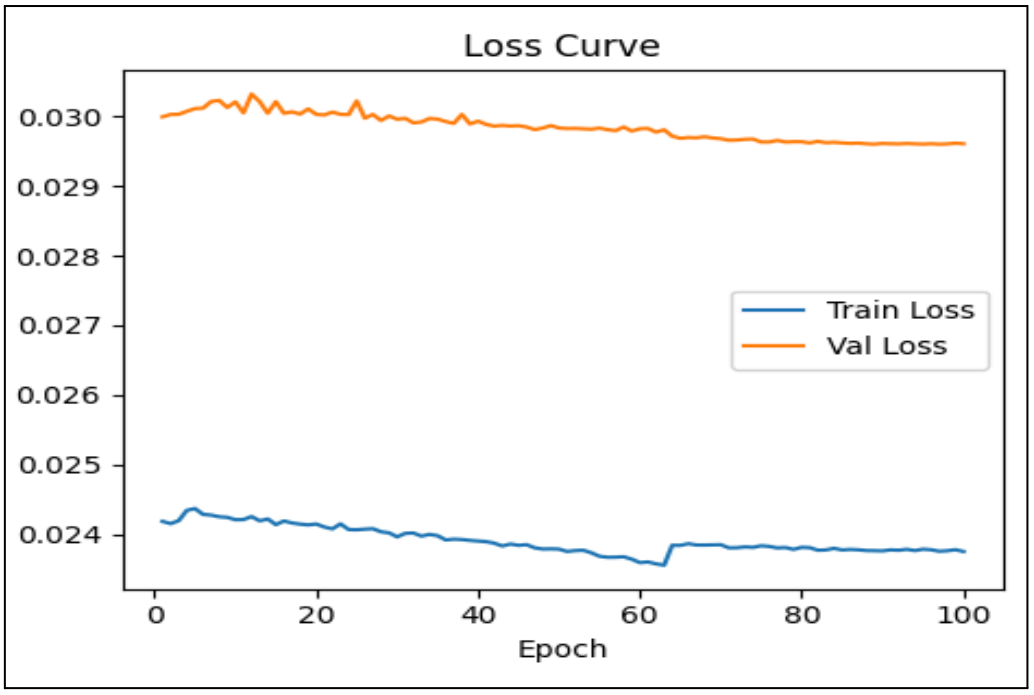

Fig 32. The plot of Training and Validation Loss using GFL Loss[3] as the loss function of PromptIR[1] w/ Second stage Fine-Tuning.

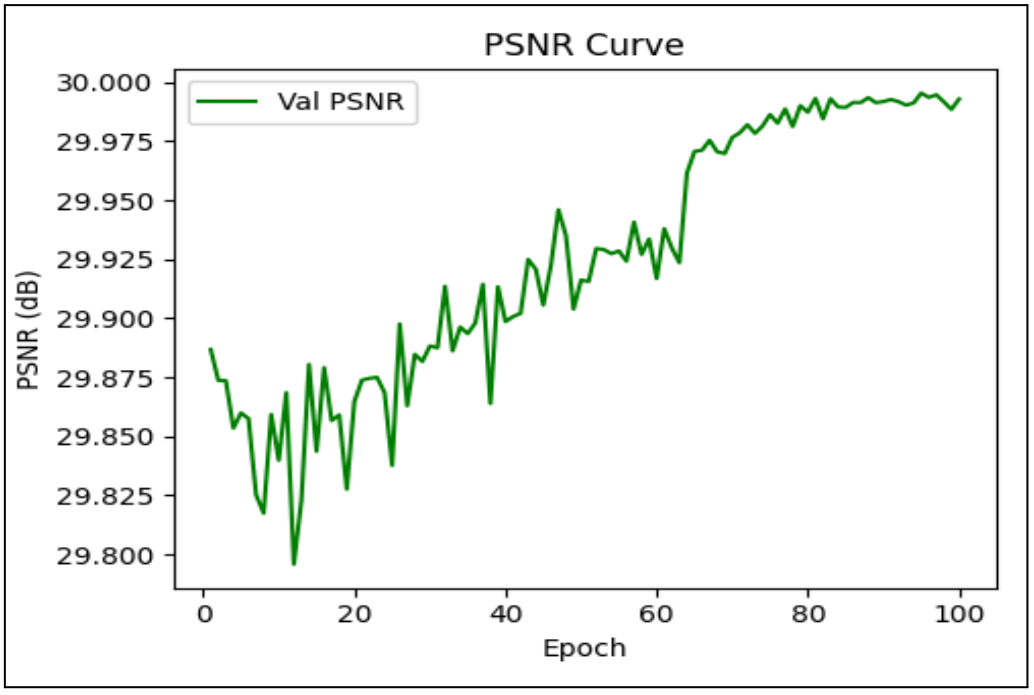

Fig 33. The plot of Validation PSNR using GFL Loss[3] as the loss function of PromptIR[1] w/ Second stage Fine-Tuning.

(4) Using GFL Loss[3] as the loss function of PromptIR[1] w/ Second stage Fine-Tuning [Fine-Tuning Again]

In light of the observation from the Second Stage Fine-Tuning experiment that validation loss continued to decrease and PSNR steadily improved toward the end of training, we proceeded to further fine-tune the model using the same strategy. Initially, the validation loss exhibited notable oscillations, but after approximately epoch 50, it began to decline consistently and eventually stabilized. The final training loss converged at approximately **0.0234**, while the validation loss settled around **0.0292**. As shown in the PSNR curve on the validation set, the performance exceeded the previous Second Stage result of **29.995** at **epoch 33**, and peaked at **30.616** by **epoch 96**, marking the best PSNR value in this experimental sequence.

During inference, we first evaluated the model **without applying Test-Time Augmentation (TTA) [4]**, achieving a **public PSNR of 30.638** and a **private PSNR of 30.140**, thereby surpassing the **30.0 dB strong baseline by 0.14 dB**. Subsequently, we applied **TTA [4]** using the same model checkpoint and re-submitted the predictions to **CodaBench**, where the model achieved a **public PSNR of 31.292** and a **private PSNR of 30.733**. These results **demonstrate a significant improvement** over the strong baseline, reaffirming the efficacy of our extended fine-tuning and the contribution of TTA to performance enhancement.

Fig 34. The PSNR score using GFL Loss[3] as the loss function of PromptIR[1] w/ Second stage Fine-Tuning [Fine-Tuning Again] and w/o TTA[4].

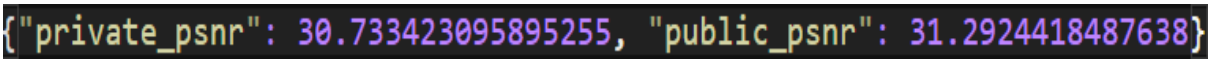

Fig 35. The PSNR score using GFL Loss[3] as the loss function of PromptIR[1] w/ Second stage Fine-Tuning [Fine-Tuning Again] and w/ TTA[4].

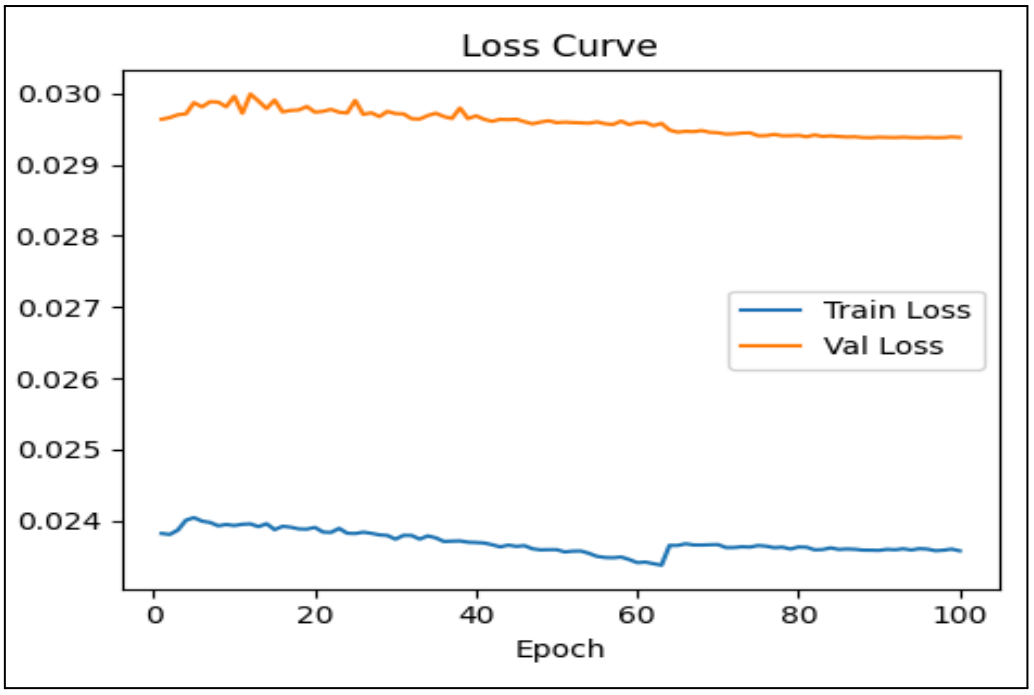

Fig 36. The plot of Training and Validation Loss using GFL Loss[3] as the loss function of PromptIR[1] w/ Second stage Fine-Tuning.

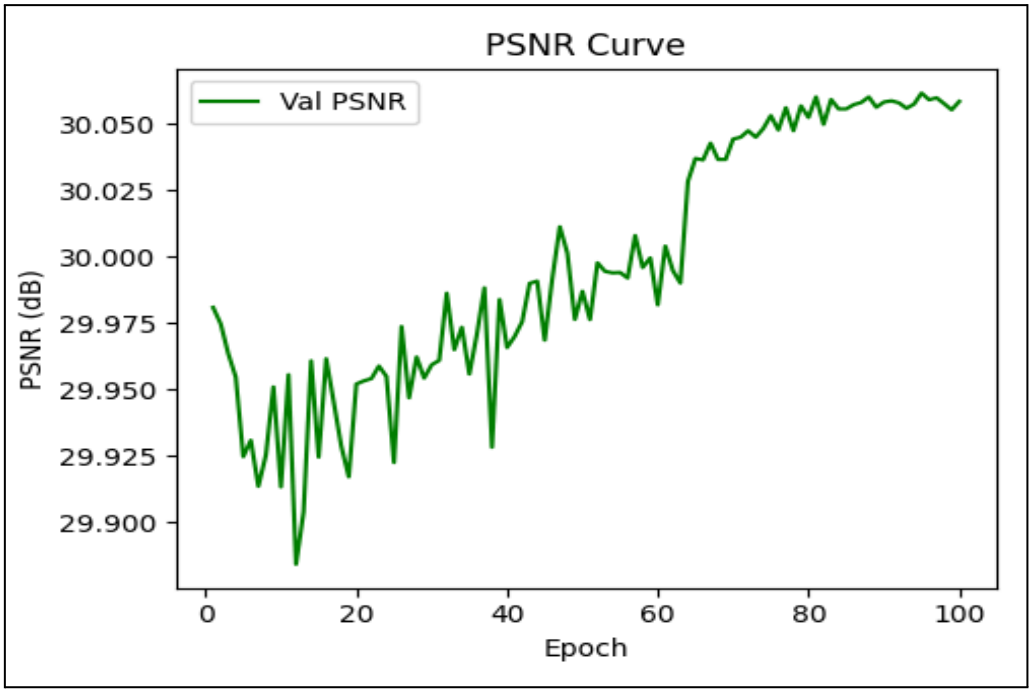

Fig 37. The plot of Validation PSNR using GFL Loss[3] as the loss function of PromptIR[1] w/ Second stage Fine-Tuning.

3.4 Visualization of additional Experiment Results

In this section, we showcase two snow and two rain examples: the top row displays the degraded inputs, while the bottom row shows their restored outputs produced by our trained model. These four image pairs correspond to the four **distinct GFL Loss[3] evaluated in our earlier experiments (in 3.3 section).**

I. **Using GFL Loss[3] as the loss function of PromptIR[1] w/o Fine-Tuning and got a 30.229 PSNR in private score.**

Fig 38. Visualization of the baseline result (GFL Loss[3]) w/o Fine-Tuning

II. **Using GFL Loss[3] as the loss function of PromptIR[1] w/ First stage Fine-Tuning and got a 30.491 PSNR in private score.**

Fig 39. Visualization of the baseline result (GFL Loss[3]) w/ First stage Fine-Tuning

III. Using GFL Loss[3] as the loss function of PromptIR[1] w/ Second stage Fine-Tuning and got a 30.630 PSNR in private score.

Fig 40. Visualization of the baseline result (GFL Loss[3]) w/ Second stage Fine-Tuning

IV. Using GFL Loss[3] as the loss function of PromptIR[1] w/ Second stage Fine-Tuning [Fine-Tuning Again] and got a 30.733 PSNR in private score.

Fig 41. Visualization of the baseline result (GFL Loss[3]) w/ Second stage Fine-Tuning [Fine-Tuning Again]

4. Discussion and Conclusion

4.1 Experiment Results

PromptIR[1] **w/ Loss Function & Fine-Tuning Settings**	**val PSNR**	**test PSNR** **(w/o TTA[4])**	**test PSNR** **(w/ TTA[4])**
Charbonnier Loss[2] w/o FT	29.534	29.481	**30.001**
Charbonnier Loss[2] w/ first-stage FT	29.830	29.833	**30.385**
Charbonnier Loss[2] w/ second-stage FT	29.949	29.963	**30.517**
Charbonnier Loss[2] w/ second-stage FT [FT again]	30.024	**30.042**	**30.586**
GFL Loss[3] w/o FT	29.620	29.556	**30.229**
GFL Loss[3] w/ first-stage FT	29.879	29.878	**30.491**
GFL Loss[3] w/ second-stage FT	29.995	**30.028**	**30.630**
GFL Loss[3] w/ second-stage FT [FT again]	30.062	**30.140**	**30.733**

Table 4. All experiments results

As shown in **Table 4**, this experiment adopts **Charbonnier Loss [2]** as the loss function for the **baseline model**. Additionally, we implement and reproduce the formulation of **Guided Frequency Loss (GFL) [3]** (refer to **GFL_Loss.py**) for comparative analysis as part of the **additional experiments**. We further conduct a detailed comparison regarding the presence or absence of **stage-wise fine-tuning**, and during inference, we evaluate the impact of applying **Test-Time Augmentation (TTA) [4]**. In total, this results in a comparison of **eight configurations and sixteen outcome variations**, covering loss type, fine-tuning strategy, and inference augmentation.

From the perspective of **loss function comparison**, experimental results demonstrate that adopting **GFL Loss** yields a consistent **PSNR improvement of approximately 0.1 to 0.22** over using **Charbonnier Loss** alone across all configurations. This performance gain can be attributed to the **three-fold design** of the GFL Loss. First, the **Laplacian Pyramid component** imposes multi-scale residual constraints at **coarse, mid-level, and fine-grained resolutions**, enabling the model to reconstruct global structure while preserving local texture. Second, the **Charbonnier component**, with its **smooth L2-like behavior**, offers **robust**

pixel-wise error estimation that mitigates the influence of outliers and stabilizes training. Most critically, the **Gradual Frequency (ΘC) component** incrementally expands the **high-pass filter bandwidth**, guiding the model to focus on the **most challenging high-frequency details early in training**, and gradually incorporating broader frequency bands to enhance **edge sharpness and fine detail recovery**.

Second, when interpreting the table from the perspective of **fine-tuning**, we observe that after applying **three stages of training, including two rounds of fine-tuning**, the model consistently achieves a **PSNR improvement of approximately 0.5–0.6 dB** compared to the baseline without fine-tuning, regardless of whether **TTA [4]** is applied during inference. This improvement is likely due to the initial use of **high-difficulty augmentations**, such as **CutBlur [5]**, **Mixup [6]**, and **Blend [7]**, which hindered optimal convergence during the initial training phase. Through **First Stage Fine-Tuning**, PSNR improved by at least **0.25 dB**, and in the **Second Stage**, where **only a subset of modules were unfrozen** to mitigate potential overfitting, an additional **0.2–0.25 dB** improvement was observed. These results affirm that our **multi-stage fine-tuning framework** effectively enhances restoration quality.

Third, regarding the effect of **Test-Time Augmentation (TTA) [4]**, the results clearly show that applying TTA yields an additional **0.5–0.65 dB PSNR gain**, irrespective of the loss function used (**Charbonnier [2]** or **GFL [3]**). This gain stems from the **weak ensemble** effect of TTA, where predictions from multiple transformed versions of the input (e.g., flips and rotations) are **inverted and averaged**, reducing the influence of network variance or noise while leveraging **spatial invariance** to enhance detail consistency. Thus, TTA consistently contributes to **more stable and accurate inference**, making it a **universally beneficial strategy** across different loss configurations.

Finally, experimental results demonstrate that the **optimal configuration** in our study involves training **PromptIR [1]** with the proposed **GFL Loss [3]**, combined with our **two-stage fine-tuning strategy**, and most critically, applying **Test-Time Augmentation (TTA) [4]** during inference. This setup achieved a **PSNR of 30.733 dB** on the **private leaderboard**, thereby **successfully surpassing** the **30 dB strong baseline**.

Reference

[1] Potlapalli, V., Zamir, S. W., Khan, S. H., & Shahbaz Khan, F. (2023). Promptir: Prompting for all-in-one image restoration. Advances in Neural Information Processing Systems, 36, 71275-71293.

[2] Charbonnier, P., Blanc-Feraud, L., Aubert, G., & Barlaud, M. (1994, November). Two deterministic half-quadratic regularization algorithms for computed imaging. In Proceedings of 1st international conference on image processing (Vol. 2, pp. 168-172). IEEE.

[3] Benjdira, B., Ali, A. M., & Koubaa, A. (2023). Guided frequency loss for image restoration. arXiv preprint arXiv:2309.15563.

[4] Shanmugam, D., Blalock, D., Balakrishnan, G., & Guttag, J. (2021). Better aggregation in test-time augmentation. In Proceedings of the IEEE/CVF international conference on computer vision (pp. 1214-1223).

[5] Yoo, J., Ahn, N., & Sohn, K. A. (2020). Rethinking data augmentation for image super-resolution: A comprehensive analysis and a new strategy. In Proceedings of the IEEE/CVF conference on computer vision and pattern recognition (pp. 8375-8384).

[6] Zhang, H., Cisse, M., Dauphin, Y. N., & Lopez-Paz, D. (2017). mixup: Beyond empirical risk minimization. arXiv preprint arXiv:1710.09412.

[7] Shyam, P., Sengar, S. S., Yoon, K. J., & Kim, K. S. (2021). Evaluating copy-blend augmentation for low level vision tasks. arXiv preprint arXiv:2103.05889.