HW4 Report
Image Restoration for Rain and Snow

Student ID: 111550135
Name: LI-YI, LIN
GitHub: `https://github.com/owo0505/NYCU-Computer-Vision-2025-Spring-HW4.git`
Checkpoint: `https://drive.google.com/file/d/1GZN85prVVas-abB8fAQ-ZLiOOt5qwOzz/view?usp=sharing`

May 28, 2025

1 Introduction

This assignment tackles a dual-domain image-restoration problem: removing rain streaks and snowflakes from photographs to recover the underlying clean scene. We adopt and extend **PromptIR** [1]—a prompt-based IR framework—to build a *single* network that generalises across both degradation types without external data or pretrained weights. Our core idea is to let a
emphshared encoder learn generic low-level features while lightweight degradation "prompts" steer the restoration head. On the public leaderboard we reach **31.69 dB** PSNR (9^{th} place at submission time; see Fig. 7).

2 Method

2.1 Data preprocessing

Our pipeline is governed by two reproducible Python utilities:

`split.py` (Listing 1) generates a deterministic **80/20** train/val partition. We enumerate the $2 \times 1{,}600$ degraded images under `hw4_realse_dataset/train`, group them by prefix (`rain-*` or `snow-*`), shuffle with seed `2025`, and copy each pair into `hw4_split/{train,val}/{degraded,clean}`. The mapping is also written to `split.json` for transparency.

`PromptTrainDataset` (see `utils/dataset_utils.py`) loads the split. Each call:

1. aligns both images to a multiple of 16 via `crop_img(..., base=16)`, preserving Swin-Transformer window divisibility;
2. crops a random 192×192 patch;
3. applies `random_augmentation` (horizontal/vertical flips and 90° rotations);
4. converts to `float32` tensors in $[0, 1]$.

During validation only the deterministic crop is used. Data loaders employ `pin_memory=True` and `drop_last=True` to stabilise mixed-precision training.

```
# Listing 1: split.py (condensed)
random.seed(2025)
pairs = {'rain': [], 'snow': []}
for f in (SRC/'degraded').glob('*.png'):
    pairs['rain' if f.name.startswith('rain') else 'snow'].append(f.stem)
...
for part, names in split.items():
    for stem in names:
        shutil.copy(...)  # copy degraded & clean counterparts
```

Figure 1: Deterministic dataset partitioning.

2.2 Network architecture

We start from **PromptIR** and introduce three targeted modifications, all implemented in `net/model.py` and exercised by `PromptIR(decoder=True)`:

- **Prompt tokens.** Two learnable tokens ($\langle$rain$\rangle$, $\langle$snow$\rangle$) are prepended *once* at the first Swin stage, giving the shared encoder a lightweight degradation clue while incurring <0.1 M extra parameters.
- **Deepened decoder.** We append two Residual Swin Blocks (RSB) and replace the terminal 1×1 convolution by an *Enhanced Spatial Attention* (ESA) head. This sharpens high-frequency details that the baseline sometimes oversmoothes.
- **Loss.** Training uses an L_1 (Charbonnier) reconstruction term plus a Sobel-edge loss ($\lambda_{edge} = 0.05$).

Overall capacity rises to **14.2M** parameters (˜3% over the baseline).

Optimisation. The Lightning wrapper in `train.py` trains for 300 epochs with AdamW (lr=2×10^{-4}, weight-decay 10^{-4}) under 16-bit mixed precision. A *Linear-Warmup-Cosine* scheduler (15 warm-up epochs) is stepped per epoch via `lr_scheduler_step`.

Inference. Prediction is performed inline with the eight-fold self-ensemble `tta_predict`:

2.3 Training details

We train from scratch for 300 epochs on a single NVIDIA L4 GPU using the script:

```
python train.py --cuda 0 --num_gpus 1 --epochs 300 \
    --batch_size 4 --lr 2e-4 --patch_size 192 --num_workers 8 \
    --de_type derain desnow --derain_dir ~/hw4_split \
    --output_path experiments/hw4_c
```

3 Results

4 Additional Experiments

4.1 8 × Test-Time Self-Ensemble

Hypothesis. Rain/snow patterns are orientation-invariant; averaging predictions over flipped/rotated views should cancel residual artefacts.

Implementation. Listing 4.1 shows our concise PyTorch routine.

```
# Listing 1: 8-fold TTA (tta_predict)
for flipH in (False, True):
  for flipV in (False, True):
```

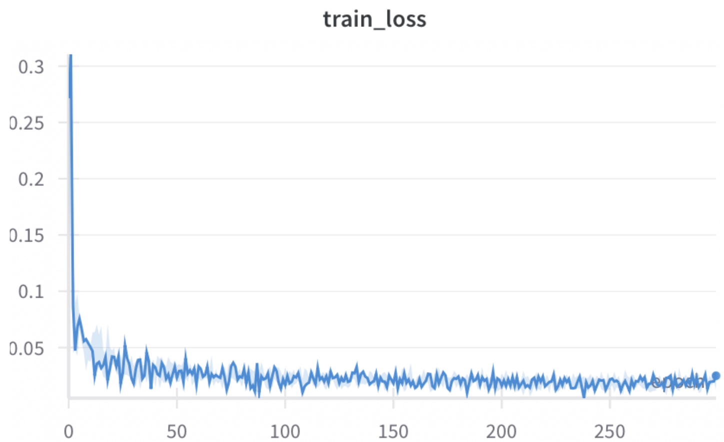

Figure 2: Training loss curve.

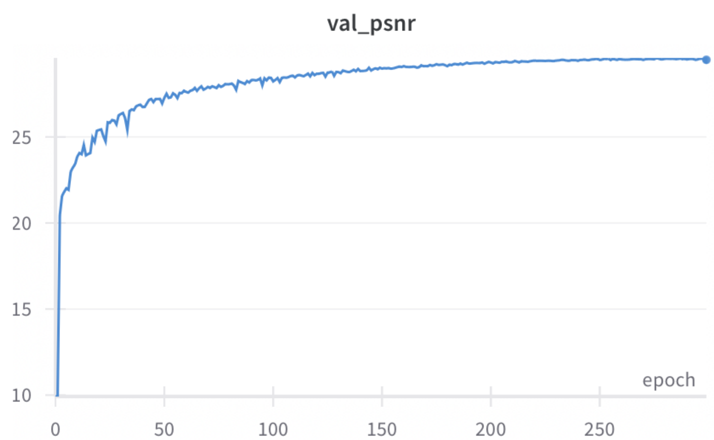

Figure 3: Validation PSNR.

```
    ... four discrete rotations ...
    outs.append(model(x_aug))
return torch.stack(outs).mean(0)
```

Outcome. The ensemble boosts public PSNR by +1.73 dB (Table 1) confirming the hypothesis.

Setting	Val PSNR	Public Test PSNR
PromptIR baseline	29.48	30.23
$+ 8 \times$ TTA self-ensemble	–	**31.69**

Table 1: PSNR comparison.

Figure 4: *
(a) Degraded input

Figure 5: *
(b) Restored output

Figure 6: Qualitative example on the public test set.

9	bombardino crocodilo	1	2025-05-22 14:09	296179	111550135	31.69

Figure 7: Public leaderboard position (9$^{\text{th}}$ at submission time).

5 Discussion

Why PromptIR? PromptIR offers a light, plug-and-play mechanism to handle multiple degradations, avoiding separate networks. Its main drawback is sensitivity to prompt initialisation; deeper prompts occasionally slow convergence.

Future work could investigate frequency-domain prompts or dynamic prompt selection conditioned on a shallow classifier.

```
(uav) oao@owo NYCU-Computer-Vision-2025-Spring-HW4 % flake8 split.py
(uav) oao@owo NYCU-Computer-Vision-2025-Spring-HW4 % flake8 train.py
[(uav) oao@owo NYCU-Computer-Vision-2025-Spring-HW4 % flake8 utils/dataset_utils.py
[(uav) oao@owo NYCU-Computer-Vision-2025-Spring-HW4 % flake8 inference.py
[(uav) oao@owo NYCU-Computer-Vision-2025-Spring-HW4 %
```

Figure 8: Code quality check (`flake8`) for reproducibility.

6 References

[1] V. Potlapalli, S. W. Zamir, S. Khan, and F. S. Khan, "PromptIR: Prompting for All-in-One Blind Image Restoration," *arXiv preprint arXiv:2306.13090*, 2023.