NYCU CV2025 HW4

GitHub Link:
https://github.com/CEJiang/NYCU-Computer-Vision-2025-Spring-HW4.git

1. Introduction

In this assignment, we address the task of image restoration under challenging degradation conditions, such as synthetic rain and snow. Given the complexity of natural corruption patterns, a robust and effective model is essential for accurately recovering image details. To this end, we adopt the **PromptIR All-in-One architecture [1]**, which integrates prompt-based feature modulation with transformer-based encoding to enhance restoration across multiple degradation types.

Our model is specifically designed and optimized to perform well on the HW4 image restoration dataset, where each image is paired with a clean ground truth. PromptIR employs a multi-level prompt injection mechanism, enabling the network to adapt its feature processing dynamically, which greatly improves restoration quality by preserving structural information and reducing residual artifacts.

To further enhance performance, we incorporate **Test-Time Augmentation (TTA)[2]** during inference and use a hybrid training loss that combines **L1 and SSIM [3]**, striking a balance between pixel-wise accuracy and perceptual quality. Experimental results show that our method achieves **31.17 dB PSNR** on the public leaderboard, demonstrating its effectiveness in restoring realistic visual content under synthetic degradations.

2. Method

2.0 Data split

Since no separate validation set was provided for this task, we manually split the training set into a training and validation subset with an 80:20 ratio. This split was used for hyperparameter tuning.

2.1 Data Preprocessing

HorizontalFlip	There are eight unique combinations when combining HorizontalFlip, VerticalFlip, and RandomRotate90.
VerticalFlip	
RandomRotate90	
ToTensor	

1. HorizontalFlip

   Randomly flips the image left to right. This helps the model learn position-invariant features, especially useful when the object layout is symmetric or when degradations (like rain) are not strictly position-dependent.

2. VerticalFlip

   Randomly flips the image top to bottom. Although vertical flipping does not commonly occur in natural images, it helps regularize the model and ensures robustness to flipped degradations during training.

3. ToTensor

   We apply only 180° rotation instead of the usual 90°/180°/270° options. This is because both rain and snow in our dataset are vertically aligned — applying 90° or 270° rotation would disrupt the natural direction of degradation and confuse the model. A 180° rotation, however, preserves the vertical orientation and thus maintains the semantic consistency of the corrupted patterns.

4. ToTensor

   Converts a PIL image to a PyTorch tensor and scales pixel values to [0, 1]. This is necessary for efficient tensor computation and loss calculation during training.

2.2 Model Architecture

```python
class PromptGenerationModule(nn.Module):
    """Generate task-aware prompts based on input features."""

    def __init__(self, num_prompts, channels):
        super().__init__()
        self.prompt_components = nn.Parameter(
            torch.randn(num_prompts, channels, 1, 1))
        self.gap = nn.AdaptiveAvgPool2d(1)
        self.selector = nn.Sequential(
            nn.Conv2d(channels, channels // 2, kernel_size=1),
            nn.ReLU(inplace=True),
            nn.Conv2d(channels // 2, num_prompts, kernel_size=1),
            nn.Flatten(start_dim=1),
            nn.Softmax(dim=1)
        )

    def forward(self, features):
        """Forward pass to generate the prompt tensor."""
        pooled = self.gap(features)
        scores = self.selector(pooled)
        prompt = torch.einsum("bn,nchw->bchw", scores, self.prompt_components)
        return prompt
```

```python
class PromptInteractionModule(nn.Module):
    """Inject prompts and apply transformer + convolutional fusion."""

    def __init__(self, channels, transformer_block):
        super().__init__()
        self.concat = nn.Conv2d(channels * 2, channels, kernel_size=1)
        self.transformer_block = transformer_block
        self.fuse = nn.Sequential(
            nn.Conv2d(channels, channels, kernel_size=1),
            nn.ReLU(inplace=True),
            nn.Conv2d(channels, channels, kernel_size=3, padding=1)
        )

    def forward(self, features, prompt):
        """Forward pass to integrate prompt into features."""
        prompt = prompt.expand_as(features)
        combined = torch.cat([features, prompt], dim=1)
        fused = self.concat(combined)
        transformed = self.transformer_block(fused)
        return self.fuse(transformed)
```

```python
class PromptBlock(nn.Module):
    """Combine Prompt Generation and Interaction modules."""

    def __init__(self, num_prompts, channels, transformer_block):
        super().__init__()
        self.pgm = PromptGenerationModule(num_prompts, channels)
        self.pim = PromptInteractionModule(channels, transformer_block)

    def forward(self, features):
        """Forward pass through PGM + PIM."""
        prompt = self.pgm(features)
        return self.pim(features, prompt)
```

```python
class TransformerBlock(nn.Module):
    """Wrap multiple SwinTransformerBlocks into a module."""

    def __init__(self, dim, input_resolution,
                 depth=4, num_heads=3, window_size=7):
        super().__init__()
        self.blocks = nn.Sequential(*[
            SwinTransformerBlock(
                dim=dim,
                input_resolution=input_resolution,
                num_heads=num_heads,
                window_size=window_size,
                shift_size=0 if (i % 2 == 0) else window_size // 2,
                mlp_ratio=4.0,
                qkv_bias=True,
                drop_path=0.0,
                norm_layer=nn.LayerNorm
            ) for i in range(depth)
        ])

    def forward(self, x):
        """Forward pass through Swin Transformer blocks."""
        x = x.permute(0, 2, 3, 1).contiguous()
        x = self.blocks(x)
        return x.permute(0, 3, 1, 2).contiguous()
```

```python
class Downsample(nn.Module):
    """Downsample block using conv + pixel unshuffle."""

    def __init__(self, in_channels):
        super().__init__()
        self.body = nn.Sequential(
            nn.Conv2d(
                in_channels,
                in_channels // 2,
                kernel_size=3,
                stride=1,
                padding=1,
                bias=False),
            nn.PixelUnshuffle(2)
        )

    def forward(self, x):
        """Forward downsampling."""
        return self.body(x)


class Upsample(nn.Module):
    """Upsample block using conv + pixel shuffle."""

    def __init__(self, in_channels):
        super().__init__()
        self.body = nn.Sequential(
            nn.Conv2d(
                in_channels,
                in_channels * 2,
                kernel_size=3,
                stride=1,
                padding=1,
                bias=False),
            nn.PixelShuffle(2)
        )

    def forward(self, x):
        """Forward upsampling."""
        return self.body(x)
```

```python
class PromptIR(nn.Module):
    """Main PromptIR architecture with Swin Transformer and prompt blocks."""

    def __init__(self, num_prompts=2):
        super().__init__()
        self.input_proj = nn.Conv2d(3, 96, 3, padding=1)

        # Encoder
        self.encoder1 = TransformerBlock(96, (256, 256))
        self.encoder2 = TransformerBlock(192, (128, 128))
        self.encoder3 = TransformerBlock(384, (64, 64))
        self.encoder4 = TransformerBlock(768, (32, 32))

        self.downsample1_2 = Downsample(96)
        self.downsample2_3 = Downsample(192)
        self.downsample3_4 = Downsample(384)

        # Decoder
        self.upsample4_3 = Upsample(768)
        self.upsample3_2 = Upsample(384)
        self.upsample2_1 = Upsample(192)

        # Prompt blocks
        self.prompt4 = PromptBlock(
            num_prompts, 768, TransformerBlock(768, (32, 32), depth=2))
        self.prompt3 = PromptBlock(
            num_prompts, 384, TransformerBlock(384, (64, 64), depth=2))
        self.prompt2 = PromptBlock(
            num_prompts, 192, TransformerBlock(192, (128, 128), depth=2))
        self.prompt1 = PromptBlock(
            num_prompts, 96, TransformerBlock(96, (256, 256), depth=2))

        # Fusion + decoder
        self.fuse3 = nn.Conv2d(384 + 384, 384, 1)
        self.fuse2 = nn.Conv2d(192 + 192, 192, 1)
        self.fuse1 = nn.Conv2d(96 + 96, 96, 1)

        self.decoder3 = TransformerBlock(384, (64, 64))
        self.decoder2 = TransformerBlock(192, (128, 128))
        self.decoder1 = TransformerBlock(96, (256, 256))

        self.refinement = nn.Sequential(
            TransformerBlock(96, (256, 256)),
            nn.GroupNorm(8, 96)
        )

        self.final = nn.Sequential(
            nn.Conv2d(96, 96, 3, padding=1),
            nn.ReLU(inplace=True),
            nn.Conv2d(96, 3, 1)
        )
```

```python
    def forward(self, x):
        """Forward pass through PromptIR network."""
        residual = x
        x = self.input_proj(x)

        # Encode
        feat_1 = self.encoder1(x)
        feat_2 = self.encoder2(self.downsample1_2(feat_1))
        feat_3 = self.encoder3(self.downsample2_3(feat_2))
        feat_4 = self.encoder4(self.downsample3_4(feat_3))

        # Prompt injection
        prompt_4 = self.prompt4(feat_4)

        fusion_3 = torch.cat([feat_3, self.upsample4_3(prompt_4)], dim=1)
        fusion_3 = self.fuse3(fusion_3)
        fusion_3 = self.decoder3(fusion_3)
        fusion_3 = self.prompt3(fusion_3) + fusion_3

        fusion_2 = torch.cat([feat_2, self.upsample3_2(fusion_3)], dim=1)
        fusion_2 = self.fuse2(fusion_2)
        fusion_2 = self.decoder2(fusion_2)
        fusion_2 = self.prompt2(fusion_2) + fusion_2

        fusion_1 = torch.cat([feat_1, self.upsample2_1(fusion_2)], dim=1)
        fusion_1 = self.fuse1(fusion_1)
        fusion_1 = self.decoder1(fusion_1) + fusion_1

        # Final refinement
        refined = self.refinement(fusion_1) + fusion_1
        output = self.final(refined)

        return torch.sigmoid(output + residual)
```

All learning components in the model are designed following the principle of residual learning [4]. In addition to the global residual connection that adds the original input image to the final output, local residual connections are also used—for instance, in the decoder stages where the output of each Prompt Block is added back to the feature map. This strategy helps the model focus on learning the residual (differences) instead of reconstructing the entire image, thereby improving both restoration performance and training stability.

Before entering the encoder, the input image is first passed through a 3×3 convolution layer to project it into a 96-channel feature map. This serves as the initial embedding layer, matching the channel dimension of the first encoder stage.

Encoder

I designed the encoder based on the structure of Swin Transformer Tiny [5], dividing it into four stages with channel dimensions of 96, 192, 384, and 768 respectively. Each stage uses a custom Transformer Block, which wraps multiple Swin Transformer Blocks for feature extraction.

Between each stage, I apply a Downsample module to reduce spatial resolution while increasing channel dimensions. This module first uses a 3×3 convolution followed by PixelUnshuffle, effectively halving the spatial size and doubling the channels. This design allows the model to capture increasingly higher-level semantic features in deeper layers.

Prompt Block (PGM + PIM)

- PGM (Prompt Generation Module)

  I first apply global average pooling to the feature map, then pass it through two 1×1 convolution layers to generate a set of weights. These weights are used to compute a weighted sum of learnable prompt vectors, producing a prompt tensor that has the same shape as the input feature map.

- PIM (Prompt Interaction Module)

  I concatenate this prompt tensor with the original feature map along the channel dimension, then use a 1×1 convolution to merge them. The result is passed through a Swin Transformer Block for enhancement, followed by

additional convolutions for feature fusion. Each decoder stage uses one Prompt Block like this.

Encoder

The decoder mirrors the structure of the encoder. I first use an Upsample module (convolution + PixelShuffle) to enlarge the feature maps, then concatenate them with the corresponding encoder features as skip connections.

Each stage passes through a fuse convolution to merge channels, followed by a Transformer Block for feature integration. A Prompt Block is then applied to inject task-specific guidance. Finally, the output goes through a refinement module to enhance details and produces the final three-channel restored image.

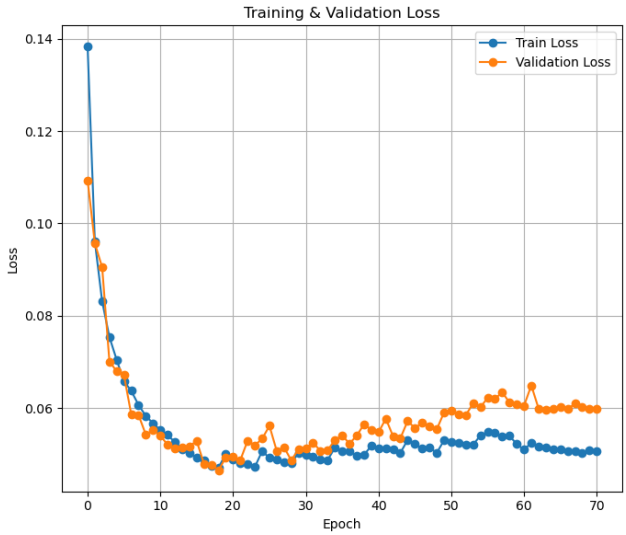
After the final decoding stage, a refinement module is applied, which includes a Transformer Block and GroupNorm [6]. This helps refine the output feature before generating the final image.

The final output is produced by two convolution layers: a 3×3 followed by a 1×1 to reduce the feature map back to 3 channels (RGB image). A sigmoid is applied at the end to ensure the output pixel values are in the [0, 1] range.

2.3 Hyperparameters

Pretrained Weight	None
Learning rate	1×10^{-4}
Batch size	1
Epochs	80
decay	5×10^{-3}
Optimizer	AdamW
Eta_min	1×10^{-6}
T_max	80
Scheduler	CosineAnnealing
Ratio	0.3 → 0.7
Criterion	(1 - ratio) * L1 Loss ratio * SSIM Loss

3. Results

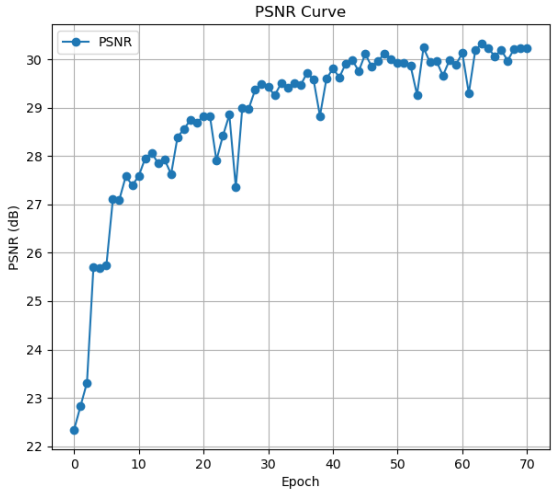


As shown in the figure, the training and validation losses decrease steadily during the first 20 epochs. In this phase, I used a loss combination of 0.7 × L1 + 0.3 × SSIM to help the model focus on pixel-level accuracy.

Starting from epoch 20, I gradually adjusted the weighting between L1 and SSIM losses. Specifically, every 5 epochs, I increased the SSIM weight by 0.05, up to a maximum of 0.7 × SSIM + 0.3 × L1. This strategy allows the model to focus more

on low-level detail in the early stage and gradually shift attention to structural consistency and perceptual quality.

During this adjustment phase, the training loss continued to decrease, but the validation loss began to fluctuate slightly and even increase slightly around epochs 30 to 70. This indicates a mild overfitting effect, likely due to SSIM being more sensitive and the learned structural features not generalizing perfectly on the validation set. Nonetheless, the overall loss remains within a reasonable range, and training remains stable.



From the curve, we can observe several noticeable drops in PSNR that do not directly correspond to any changes in the loss curve. These drops likely indicate that the model is shifting its learning focus or adopting new feature representations at certain stages of training.

Such behavior typically arises when:

- The model begins transitioning from low-level texture reconstruction to more structure-aware restoration,
- Or when Prompt Blocks influence the model to break out of its previous learning pattern and explore new semantic features.

These temporary fluctuations can be seen as part of an exploration phase, rather than signs of degradation or overfitting. Notably, the PSNR consistently recovers and continues to improve afterward, suggesting that such exploration helps the model

discover more effective restoration strategies.

From the figure, it can be observed that the predicted image is almost indistinguishable from the original clean image in terms of overall visual quality. The model successfully restores the degraded image—affected by rain streaks—back to its normal appearance.

However, a closer examination of the red-boxed area reveals minor discrepancies at the detail level. For example, the bricks and the fine lines between them appear slightly blurred in the predicted image, and some texture details are not fully recovered.

This indicates that while the model performs excellently in overall restoration, there is still room for improvement in reconstructing high-frequency details.

4. References

[1] Potlapalli, V., Zamir, S. W., Khan, S., & Khan, F. S. Promptir: Prompting for all-in-one blind image restoration. arxiv 2023. arXiv preprint arXiv:2306.13090, 7.

[2] Kimura, M. (2021, December). Understanding test-time augmentation. In International Conference on Neural Information Processing (pp. 558-569). Cham: Springer International Publishing.

[3] Zhao, H., Gallo, O., Frosio, I., & Kautz, J. (2016). Loss functions for image restoration with neural networks. IEEE Transactions on computational imaging, 3(1), 47-57.

[4] Bae, W., Yoo, J., & Chul Ye, J. (2017). Beyond deep residual learning for image restoration: Persistent homology-guided manifold simplification. In Proceedings of the IEEE conference on computer vision and pattern recognition workshops (pp. 145-153).

[5] Liu, Z., Lin, Y., Cao, Y., Hu, H., Wei, Y., Zhang, Z., ... & Guo, B. (2021). Swin transformer: Hierarchical vision transformer using shifted windows. In Proceedings of the IEEE/CVF international conference on computer vision (pp. 10012-10022).

[6] Wu, Y., & He, K. (2018). Group normalization. In Proceedings of the European conference on computer vision (ECCV) (pp. 3-19).

5. Additional experiments

Pros

1. **Hierarchical Feature Extraction (Swin Transformer)**

   Swin Transformer's hierarchical structure with shifted window attention enables efficient learning of both local and global features. This is particularly beneficial for image restoration, which often requires capturing fine details and contextual structure.

2. **Task-Aware Adaptation (Prompt Block)**

   By injecting prompts at multiple decoding stages, the model dynamically adapts to image content and structural cues. This improves the model's ability to restore complex patterns and textures.

3. **Residual Learning Design:**

   The entire model is built upon residual connections, which help stabilize training and focus the learning on residual differences instead of reconstructing the full image from scratch.

Cons

1. **Higher Computational Cost (Transformer)**

   Compared to CNN-based methods, Transformer blocks, even with local windowing, are still computationally heavier, especially during decoding stages with larger feature maps.

2. **Prompt Injection Complexity**

   Designing and integrating effective prompt injection requires careful tuning, and improper usage can lead to unstable training or degraded performance.

3. **PSNR-Sensitive Behavior**

   Since the model shifts its focus to structural similarity (SSIM) over time, PSNR may fluctuate, especially during mid-training phases when the loss balance changes.

Hypothesis: The hypothesis is that increasing the depth of the Transformer Block (i.e., stacking more Swin Transformer Blocks) will allow the model to capture richer contextual and long-range dependencies, which may benefit image restoration performance. Therefore, I compare the performance of depth = 2 vs. depth = 4.

Why it may (or may not) work: While deeper transformers theoretically improve representational capacity, they also increase training difficulty and risk of overfitting. This is especially problematic when training data is limited or the image structures are relatively simple, where the extra depth may not translate to better results and might even degrade stability.

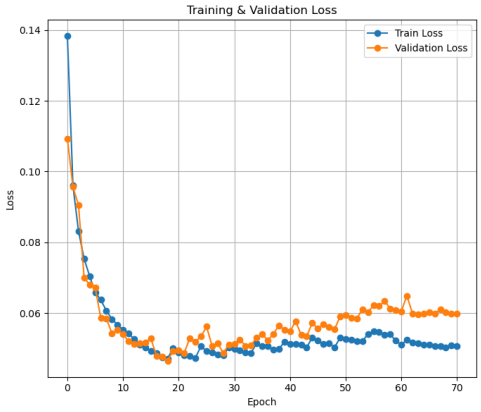
Experimental results & implications: According to the results, both Depth = 4 and Depth = 2 models exhibit similar convergence patterns in early epochs. However, after around epoch 30, the Depth = 4 model shows more stable validation loss, indicating slightly better generalization. On the PSNR curve, both models reach around 30 dB eventually. However, the Depth = 4 model climbs faster and exhibits smaller fluctuations, especially in the later stages of training. These observations suggest that increasing the depth of Swin Transformer blocks provides minor advantages in restoring complex patterns and stabilizing performance, at the cost of increased computational load.

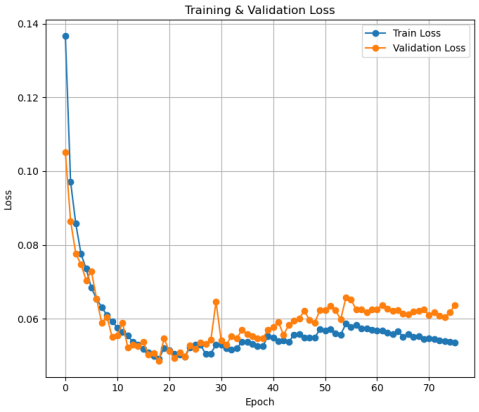
```python
class TransformerBlock(nn.Module):
    """Wrap multiple SwinTransformerBlocks into a module."""

    def __init__(self, dim, input_resolution,
                 depth=4, num_heads=3, window_size=7):
        super().__init__()
        self.blocks = nn.Sequential(*[
            SwinTransformerBlock(
                dim=dim,
                input_resolution=input_resolution,
                num_heads=num_heads,
                window_size=window_size,
                shift_size=0 if (i % 2 == 0) else window_size // 2,
                mlp_ratio=4.0,
                qkv_bias=True,
                drop_path=0.0,
                norm_layer=nn.LayerNorm
            ) for i in range(depth)
        ])

    def forward(self, x):
        """Forward pass through Swin Transformer blocks."""
        x = x.permute(0, 2, 3, 1).contiguous()
        x = self.blocks(x)
        return x.permute(0, 3, 1, 2).contiguous()
```

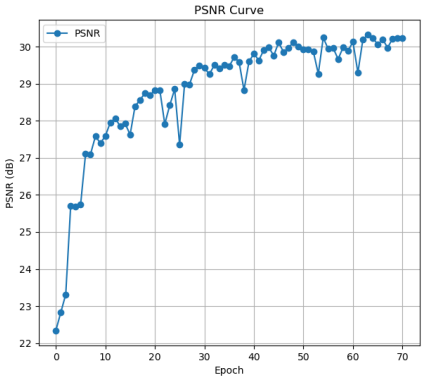
Depth = 4

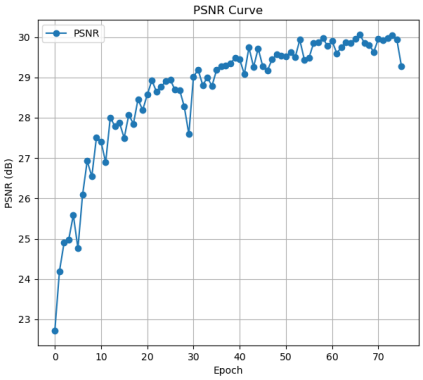
```python
class TransformerBlock(nn.Module):
    """Wrap multiple SwinTransformerBlocks into a module."""

    def __init__(self, dim, input_resolution,
                 depth=2, num_heads=3, window_size=7):
        super().__init__()
        self.blocks = nn.Sequential(*[
            SwinTransformerBlock(
                dim=dim,
                input_resolution=input_resolution,
                num_heads=num_heads,
                window_size=window_size,
                shift_size=0 if (i % 2 == 0) else window_size // 2,
                mlp_ratio=4.0,
                qkv_bias=True,
                drop_path=0.0,
                norm_layer=nn.LayerNorm
            ) for i in range(depth)
        ])

    def forward(self, x):
        """Forward pass through Swin Transformer blocks."""
        x = x.permute(0, 2, 3, 1).contiguous()
        x = self.blocks(x)
        return x.permute(0, 3, 1, 2).contiguous()
```

Depth = 2









Depth = 4

Depth = 2

Depth = 4

Depth = 2

From the comparison in the red-boxed region, the deeper Transformer model (depth = 4) demonstrates superior high-frequency detail preservation. The water ripples and bubbles around the fish are more clearly retained, resulting in a sharper and more natural reconstruction. In contrast, the depth = 2 model produces a smoother output in these areas, indicating a relatively weaker capability in capturing fine-grained details.