Report

1. Theoretical Justification: Accelerating DDPM with DIP-based Initial Priors

解釋:

該方法使用 DIP 生成一開始的去噪圖像，將其作為 DDPM 的起點。
利用 DIP 中 CNN 的內在結構快速捕捉高層次的圖像結構，並使用這些來加速 DDPM 的收斂。

結合 DDPM 和 DIP 的優勢:

DDPM:

- 透過反向步驟從噪聲中生成較好的圖像
- 學習原圖的分佈
- 提供了很好的生成能力，捕捉細節並生成更逼真的輸出

DIP

- 不需大量訓練數據(簡單)
- 利用 CNN 結構施加自然圖像先驗
- 快速捕捉圖像的高層次結構
- 不會過度擬合噪聲，提供快速有效的去噪機制

通過兩者優勢，Accelerating DDPM with DIP-based Initial Priors 利用 DIP 的快速去噪提供 DDPM 良好的起點，以減少計算負擔並加速整體去噪過程。

設計選擇和假設:

1. 使用 DIP 作為初始先驗
   - 設計選擇：使用 DIP 生成原始的去噪圖像，然後再應用 DDPM。
   - 假設：由 DIP 生成的初始先驗將足夠接近乾淨圖像，使得 DDPM 需要更少的 iteration 次數去細化圖像，以此達到高質量
2. 修改 DDPM 訓練
   - 設計選擇：修改 DDPM 訓練過程，使從 DIP 輸出的圖像作為起點(而非純噪聲)
   - 假設：從信息豐富的先驗（DIP 輸出）開始，將減少 DDPM 收斂所需的擴散步驟數。
3. DIP 訓練時長的實驗:

- 設計選擇：實驗不同 DIP 的訓練時長，以便平衡捕捉有意義的圖像先驗和避免過度擬合噪聲之間的取捨。
- 假設：存在最佳訓練時長，能夠在計算效率和初始先驗質量之間提供最佳平衡。

潛在好處:

1. 加速收斂速度: DIP 所生成的初始先驗為 DDPM 提供了一個良好起點，可能有效減少收斂所需的擴散步驟數。而更快的收斂意味著可以減少了計算成本和時間
2. 靈活、穩健性: 該方法可應用於各種圖像處理任務(如去噪、超分辨率、修復)，因為 DIP 和 DDPM 都具有靈活性
3. 提高圖像質量: DIP 階段確保初始圖像捕捉到重要的結構，DDPM 可以進一步細化以生成更高質量的輸出

潛在限制:

1. 實施的複雜性: 因為同時實現 DIP 跟 DDPM，需要更複雜的設計，同時也需要更多實驗去檢驗設計
2. 依賴於 DDPM 訓練數據: DIP 不需 dataset，但 DDPM 依賴於大量 data 學習圖像分佈，也因此整體方法的有效性依賴於 DDPM 訓練數據的質量和數量
3. 推理時間: 如上面潛在好處的(1)中所述，可能減少擴散步驟數，但兩階段過程（先 DIP、後 DDPM）可能還是會比單獨使用一個模型更慢

比較分析

DIP+DDPM vs 單獨使用 DDPM 相比：
好處： DIP 提供的更好初始猜測可以加速收斂速度，並且因為 DIP 階段有效捕捉高層次結構，可能可以產生更高質量的圖像。
限制：增加了複雜性，並且可能依賴於大量 data。

DIP+DDPM vs 單獨使用 DIP 相比：
好處：DDPM 細化由 DIP 生成的初始去噪圖像，提高圖像質量，利用 DDPM 的生成能力來增強真實感。
限制：結合方法可能需要比單獨使用 DIP 需要更多的計算資源和時間，因為 DIP 本身的優勢在於簡單和不需要訓練。

2. Experimental Verification

實驗設置

我們以 CIFAR-10 dataset 進行實驗，並且透過 PSNR、SSIM 來評估圖像的質量

```
def prepare_cifar10_dataset(batch_size):
    transform = transforms.Compose([
        transforms.ToTensor(),
        transforms.Normalize((0.5, 0.5, 0.5), (0.5, 0.5, 0.5))
    ])

    train_dataset = CIFAR10(root='./data', train=True, download=True, transform=transform)
    test_dataset = CIFAR10(root='./data', train=False, download=True, transform=transform)

    train_loader = DataLoader(train_dataset, batch_size=batch_size, shuffle=True)
    test_loader = DataLoader(test_dataset, batch_size=batch_size, shuffle=False)

    return train_loader, test_loader
```

```
# DDPM Block
class DDPMBlock(nn.Module):
    def __init__(self, channels, noise_schedule):
        super(DDPMBlock, self).__init__()
        self.net = nn.Sequential(
            nn.Conv2d(channels, channels, kernel_size=3, padding=1),
            nn.ReLU(),
            nn.Conv2d(channels, channels, kernel_size=3, padding=1),
            nn.Dropout(p=0.2)
        )
        self.sigma_schedule = nn.Parameter(torch.tensor(noise_schedule), requires_grad=False)

    def forward(self, x, t):
        sigma_t = self.sigma_schedule[t]
        noise = torch.randn_like(x) * sigma_t
        return x + noise + self.net(x)

# DDPM Model
class DDPM(nn.Module):
    def __init__(self, num_blocks, channels, noise_schedule):
        super(DDPM, self).__init__()
        self.blocks = nn.ModuleList([DDPMBlock(channels, noise_schedule) for _ in range(num_blocks)])

    def forward(self, x, t):
        for block in self.blocks:
            x = block(x, t)
        return x
```

```
# Simplified DIP model using nn.Sequential
class DIP(nn.Module):
    def __init__(self):
        super(DIP, self).__init__()
        self.model = nn.Sequential(
            nn.Linear(3 * 32 * 32, 1024),
            nn.LeakyReLU(),
            nn.Linear(1024, 3 * 32 * 32),
            nn.Sigmoid()
        )

    def forward(self, x):
        x = x.view(-1, 3 * 32 * 32)
        x = self.model(x)
        x = x.view(-1, 3, 32, 32)
        return x
```

定量評估

實驗結果

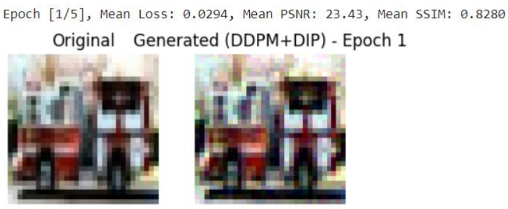
PNSR 和 SSIM: 在 5 個 epoch 的訓練下，DIP+PPDM 和 DDPM 都顯示逐步提升的趨勢而且驗證也很穩定，說明模型訓練良好

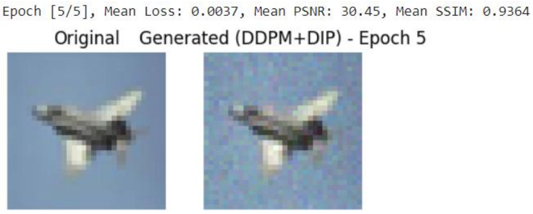
比較結果: DIP+PPDM 和 DDPM 相比，DIP+PPDM 在 PSNR 和 SSIM 表現上提升較為明顯，但 DDPM 從一開始到最後的數值較高，可能是因為構造較簡單，所以在訓練時適應性更好

定性評估

透過視覺化展示生成的圖片，可以發現 DIP+PPDM 在生成的圖像在細節跟全局一

致性都有提升





(因為此圖顏色較混雜，單看背景會比較明顯)

實驗數據

DIP+PPDM

=========I am training DDPM+DIP=========

Epoch [1/5], Mean Loss: 0.0294, Mean PSNR: 23.43, Mean SSIM: 0.8280

Epoch [2/5], Mean Loss: 0.0064, Mean PSNR: 28.10, Mean SSIM: 0.9085

Epoch [3/5], Mean Loss: 0.0047, Mean PSNR: 29.41, Mean SSIM: 0.9252

Epoch [4/5], Mean Loss: 0.0041, Mean PSNR: 30.05, Mean SSIM: 0.9323

Epoch [5/5], Mean Loss: 0.0037, Mean PSNR: 30.45, Mean SSIM: 0.9364

=========I am testing DDPM+DIP=========

Validation Mean PSNR: 31.07, Mean SSIM: 0.9417

DIP only

=========I am training DDPM only=========

Epoch [1/5], Mean Loss: 0.0027, Mean PSNR: 31.74, Mean SSIM: 0.9458

Epoch [2/5], Mean Loss: 0.0022, Mean PSNR: 32.57, Mean SSIM: 0.9528

Epoch [3/5], Mean Loss: 0.0021, Mean PSNR: 32.77, Mean SSIM: 0.9545

Epoch [4/5], Mean Loss: 0.0021, Mean PSNR: 32.84, Mean SSIM: 0.9553

Epoch [5/5], Mean Loss: 0.0021, Mean PSNR: 32.89, Mean SSIM: 0.9558

=========I am testing DDPM only=========

Validation Mean PSNR: 33.17, Mean SSIM: 0.9580

3. Ablation Studies and Analysis

消融研究設置: 為研究不同組件或超參數設置隊性能影響，進行以下幾種實驗:

噪聲水平

設置

```python
def noise_level_experiment(ddpm, dip, train_loader, test_loader, noise_levels):
    results = []
    for noise_level in noise_levels:
        print(f"=========Noise Level: {noise_level}=========")
        # Adjust noise level in DDPM
        ddpm = DDPM(num_blocks=3, channels=3, noise_schedule=np.linspace(noise_level, 0.1, 1000))
        train_combined_model_ddpm_dip(ddpm, dip, train_loader, num_epochs=3, lr=0.0001)
        mean_psnr, mean_ssim = validate_model(ddpm, dip, test_loader)
        results.append((noise_level, mean_psnr, mean_ssim))
    return results
```

實驗結果

=========Noise Level: 0.01=========

Epoch [1/3], Mean Loss: 0.0179, Mean PSNR: 25.44, Mean SSIM: 0.8850

Epoch [2/3], Mean Loss: 0.0027, Mean PSNR: 32.18, Mean SSIM: 0.9662

Epoch [3/3], Mean Loss: 0.0016, Mean PSNR: 34.37, Mean SSIM: 0.9797

Validation Mean PSNR: 36.26, Mean SSIM: 0.9865

=========Noise Level: 0.05=========

Epoch [1/3], Mean Loss: 0.0205, Mean PSNR: 23.32, Mean SSIM: 0.7900

Epoch [2/3], Mean Loss: 0.0097, Mean PSNR: 26.21, Mean SSIM: 0.8603

Epoch [3/3], Mean Loss: 0.0081, Mean PSNR: 26.99, Mean SSIM: 0.8750

Validation Mean PSNR: 27.39, Mean SSIM: 0.8816

=========Noise Level: 0.1=========

Epoch [1/3], Mean Loss: 0.0524, Mean PSNR: 19.13, Mean SSIM: 0.6588

Epoch [2/3], Mean Loss: 0.0323, Mean PSNR: 20.95, Mean SSIM: 0.7025

Epoch [3/3], Mean Loss: 0.0276, Mean PSNR: 21.62, Mean SSIM: 0.7235

Validation Mean PSNR: 22.02, Mean SSIM: 0.7371

Noise Level Experiment Results: [(0.01, 36.264279711114675, 0.9865010433895572), (0.05, 27.385407429371668, 0.8815636585472496), (0.1, 22.022560449740656, 0.7370819794903894)]

結論: 可以看出在噪聲越小，生成圖像的細節會更清晰，但過低的噪聲水平可能會讓模型欠擬合，而噪聲越高能生成更多的細節，但圖像質量卻會下降

去噪計畫

設置

```python
def denoising_steps_experiment(ddpm, dip, train_loader, test_loader, denoising_steps):
    results = []
    for steps in denoising_steps:
        print(f"=========Denoising Steps: {steps}=========")
        # Adjust denoising steps in DDPM
        ddpm = DDPM(num_blocks=3, channels=3, noise_schedule=np.linspace(0.01, 0.1, steps))
        train_combined_model_ddpm_dip(ddpm, dip, train_loader, num_epochs=3, lr=0.0001)
        mean_psnr, mean_ssim = validate_model(ddpm, dip, test_loader)
        results.append((steps, mean_psnr, mean_ssim))
    return results
```

實驗結果

=========Denoising Steps: 5=========

Epoch [1/3], Mean Loss: 0.0203, Mean PSNR: 23.83, Mean SSIM: 0.8159

Epoch [2/3], Mean Loss: 0.0103, Mean PSNR: 26.34, Mean SSIM: 0.8633

Epoch [3/3], Mean Loss: 0.0089, Mean PSNR: 27.06, Mean SSIM: 0.8741

Validation Mean PSNR: 27.52, Mean SSIM: 0.8797

=========Denoising Steps: 10=========

Epoch [1/3], Mean Loss: 0.0430, Mean PSNR: 20.86, Mean SSIM: 0.7851

Epoch [2/3], Mean Loss: 0.0096, Mean PSNR: 26.36, Mean SSIM: 0.8745

Epoch [3/3], Mean Loss: 0.0069, Mean PSNR: 27.75, Mean SSIM: 0.8933

Validation Mean PSNR: 28.92, Mean SSIM: 0.9120

=========Denoising Steps: 20=========

Epoch [1/3], Mean Loss: 0.0152, Mean PSNR: 24.95, Mean SSIM: 0.8464

Epoch [2/3], Mean Loss: 0.0057, Mean PSNR: 28.59, Mean SSIM: 0.9119

Epoch [3/3], Mean Loss: 0.0045, Mean PSNR: 29.57, Mean SSIM: 0.9238

Validation Mean PSNR: 30.12, Mean SSIM: 0.9289

Denoising Steps Experiment Results: [(5, 27.520944448736877, 0.8797296862693349), (10, 28.915344831497414, 0.9119954363555666), (20, 30.116753341364934, 0.9289215529800221)]

結論: 當去噪步數越大時，模型的 PSNR 和 SSIM 越大，在 Denoising Steps=10 時，成長的幅度最顯著，到了 20 時，雖然生成圖像質量有所提升，但是增加幅度就沒有那麼高，而且會增加訓練時間

學習率

設置

```
def learning_rate_experiment(ddpm, dip, train_loader, test_loader, learning_rates):
    results = []
    for lr in learning_rates:
        print(f"=========Learning Rate: {lr}=========")
        train_combined_model_ddpm_dip(ddpm, dip, train_loader, num_epochs=3, lr=lr)
        mean_psnr, mean_ssim = validate_model(ddpm, dip, test_loader)
        results.append((lr, mean_psnr, mean_ssim))
    return results
```

實驗結果

=========Learning Rate: 0.0001=========

Epoch [1/3], Mean Loss: 0.0023, Mean PSNR: 32.64, Mean SSIM: 0.9559

Epoch [2/3], Mean Loss: 0.0022, Mean PSNR: 32.68, Mean SSIM: 0.9563

Epoch [3/3], Mean Loss: 0.0022, Mean PSNR: 32.71, Mean SSIM: 0.9566

Validation Mean PSNR: 33.03, Mean SSIM: 0.9591

=========Learning Rate: 1e-05=========

Epoch [1/3], Mean Loss: 0.0022, Mean PSNR: 32.72, Mean SSIM: 0.9567

Epoch [2/3], Mean Loss: 0.0022, Mean PSNR: 32.72, Mean SSIM: 0.9567

Epoch [3/3], Mean Loss: 0.0022, Mean PSNR: 32.73, Mean SSIM: 0.9568

Validation Mean PSNR: 33.02, Mean SSIM: 0.9592

Learning Rate Experiment Results: [(0.0001, 33.03078034479308, 0.9591380760168574), (1e-05, 33.02404811010825, 0.9592050803694755)]

結論: 學習率的調整對於數值來說沒有明顯差異，不過仍可以看出，越小的學習率，會使的生成的圖像更穩定(前後數值差異不大)

批量大小

設置

```python
def batch_size_experiment(dip, ddpm, batch_sizes):
    results = []
    for batch_size in batch_sizes:
        print(f"=========Batch Size: {batch_size}=========")
        train_loader, test_loader = prepare_cifar10_dataset(batch_size)
        # Train DDPM + DIP model
        noise_schedule = np.linspace(0.01, 0.1, 1000)
        ddpm = DDPM(num_blocks=3, channels=3, noise_schedule=noise_schedule)
        print("=========I am training DDPM+DIP=========")
        train_combined_model_ddpm_dip(ddpm, dip, train_loader, num_epochs=3, lr=0.001)
        # Validate model on test data
        print("=========I am testing DDPM+DIP=========")
        mean_psnr_ddpm_dip, mean_ssim_ddpm_dip = validate_model(ddpm, dip, test_loader)
        results.append((batch_size, mean_psnr_ddpm_dip, mean_ssim_ddpm_dip))
    return results
```

實驗結果

=========Batch Size: 32=========

Files already downloaded and verified

Files already downloaded and verified

=========I am training DDPM+DIP=========

Epoch [1/3], Mean Loss: 0.0035, Mean PSNR: 31.30, Mean SSIM: 0.9409

Epoch [2/3], Mean Loss: 0.0024, Mean PSNR: 32.38, Mean SSIM: 0.9530

Epoch [3/3], Mean Loss: 0.0023, Mean PSNR: 32.52, Mean SSIM: 0.9545

=========I am testing DDPM+DIP=========

Validation Mean PSNR: 32.75, Mean SSIM: 0.9568

=========Batch Size: 64=========

Files already downloaded and verified

Files already downloaded and verified

=========I am training DDPM+DIP=========

Epoch [1/3], Mean Loss: 0.0077, Mean PSNR: 28.85, Mean SSIM: 0.9092

Epoch [2/3], Mean Loss: 0.0028, Mean PSNR: 31.67, Mean SSIM: 0.9467

Epoch [3/3], Mean Loss: 0.0026, Mean PSNR: 32.03, Mean SSIM: 0.9500

=========I am testing DDPM+DIP=========

Validation Mean PSNR: 32.27, Mean SSIM: 0.9519

Batch Size Experiment Results: [(32, 32.75372899736111, 0.9567597546516516), (64, 32.2707672291599, 0.9518791805407044)]

結論: 可以看出較高的批次量可以得到更好的圖形質量，當 batch size=32 時，雖然質量沒那麼好，但是訓練時間較快

總體結論

1. 噪聲水平: 較低的噪聲水平可以生成更清晰的圖像，較高噪稱水平增加圖形噪聲
2. 去噪步數: 去噪步數為 10 時，生成質量很好而且訓練時間也相對合理

3. 學習率: 越小的學習率可以讓收斂速度降低，使模型質量更穩定
4. 批量大小: 當批量大小是 64 時，模型穩定性跟質量更好