GAI Project 4 Report

AN4106068 田容甄

1. 結合 DDPM、DIP

採用範例二的作法。主要想法是在 DIP 訓練過程中引入逐步去噪的步驟，類似於 DDPM 中的去噪步驟。通過在不同級別添加噪聲到目標影像，並使用這些帶有噪聲的版本作為中間目標，可以引導 DIP 模型學習影像的階層表示，最後使用 PSNR、SSIM 的指標來衡量好壞。

2. Dataset

本研究使用的是 MNIST 資料集，資料集中的影像都是手寫數字。每張影像都是灰階的，大小為 28x28 像素。為了將像素值調整到[-1, 1]的範圍內，我們對資料集進行了正規化處理，使其平均值為 0.5，標準差為 0.5。

3. DDPM

```
# DDPM 模型
model = Unet(dim=28, channels=1, dim_mults=(1, 2, 4)).cuda()
diffusion = GaussianDiffusion(
        model,
        image_size=28,
        timesteps=1000,
).cuda()
```

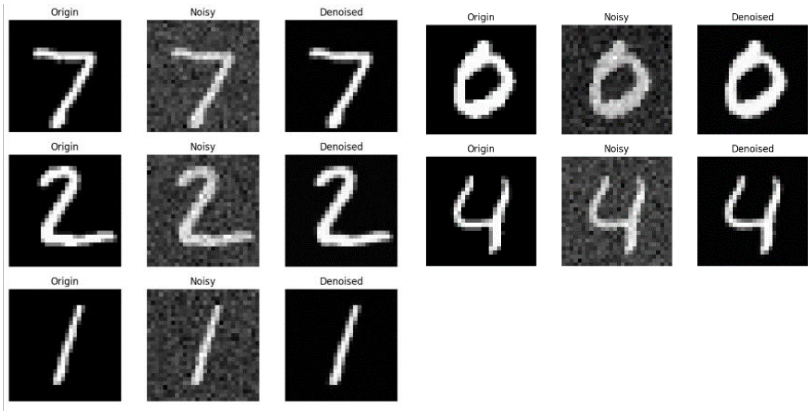
4. DIP

```
class DIP(nn.Module):
    def __init__(self):
        super(DIP, self).__init__()
        self.conv1 = nn.Conv2d(1, 64, kernel_size=3, padding=1)
        self.conv2 = nn.Conv2d(64, 64, kernel_size=3, padding=1)
        self.conv3 = nn.Conv2d(64, 1, kernel_size=3, padding=1)
        self.relu = nn.ReLU()

    def forward(self, x):
        x = self.relu(self.conv1(x))
        x = self.relu(self.conv2(x))
        x = self.conv3(x)
        return x
```

5. Combined

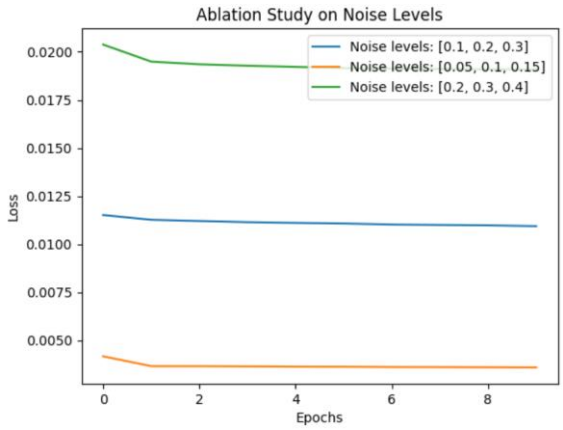
```python
def add_noise(img, noise_level):
    noise = torch.randn_like(img) * noise_level
    return img + noise
```

```python
def train_dip_with_ddpm_guidance(dip_model, trainloader, epochs=10, noise_levels=[0.1, 0.2, 0.3], display_interval=5):
    criterion = nn.MSELoss()
    optimizer = optim.Adam(dip_model.parameters(), lr=0.001)

    losses = []

    for epoch in range(epochs):
        running_loss = 0.0
        for i, (images, _) in enumerate(trainloader):
            images = images.cuda()
            optimizer.zero_grad()

            noisy_images = [add_noise(images, nl) for nl in noise_levels]
            loss = 0.0

            for noisy_img in noisy_images:
                output = dip_model(noisy_img)
                loss += criterion(output, images)

            loss.backward()
            optimizer.step()
            running_loss += loss.item()

        avg_loss = running_loss / len(trainloader)
        losses.append(avg_loss)
        print(f'Epoch [{epoch+1}/{epochs}], Loss: {avg_loss:.4f}')

    return losses
```



6. 測試不同的 noise level

noise level 越小，loss 也越小。



7. 評估和性能比較

- PSNR：PSNR 越高，去噪後的圖像質量越好。它表示去噪圖像與原始圖像的相似程度。
- SSIM：SSIM 越高，去噪後的圖像结构信息保留得越好。它表示去噪圖像與原始图像在亮度、對比度和結構方面的相似性。
- Time：運行時間越短，模型越高效。

```python
def evaluate_model(model, dataloader, noise_level):
    model.eval()
    total_psnr = 0
    total_ssim = 0
    num_images = 0
    start_time = time.time()

    with torch.no_grad():
        for images, _ in dataloader:
            images = images.cuda()
            noisy_images = add_noise(images, noise_level)
            denoised_images = model(noisy_images)

            for i in range(images.size(0)):
                clean_img = images[i].cpu().numpy().squeeze()
                denoised_img = denoised_images[i].cpu().numpy().squeeze()
                total_psnr += psnr(clean_img, denoised_img, data_range=clean_img.max() - clean_img.min())
                total_ssim += ssim(clean_img, denoised_img, data_range=clean_img.max() - clean_img.min())
                num_images += 1

    avg_psnr = total_psnr / num_images
    avg_ssim = total_ssim / num_images
    total_time = time.time() - start_time
    return avg_psnr, avg_ssim, total_time
```

8. 結論

```
DIP - PSNR: 28.425744420036494, SSIM: 0.966815156301428, Time: 7.393675327301025
DDPM - PSNR: 1.838413918314804, SSIM: -0.23770180582089323, Time: 12.910062074661255
Combined - PSNR: 30.83165863574036, SSIM: 0.9800663372988161, Time: 7.129502296447754
```

- Combined 模型在 PSNR 和 SSIM 方面表現最好，表明它在圖像去噪任務中能夠生成質量最高的圖像，同時它的運行時間也較短，表現出很好的效率。
- DIP 模型雖然在 PSNR 和 SSIM 方面略遜於 Combined 模型，但它的性能依然不錯，並且運行時間相對較短。
- DDPM 模型在 PSNR 和 SSIM 方面表現很差，表明它在圖像去噪任務中生成的圖像質量較低。此外，它的運行時間最長，效率最差。