GAI HW4

F74095322 黃渝凌

Theoretical Justification

Use the output of a DIP as the starting point for the reverse diffusion process in DDPM. DDPM's probabilistic modeling can enhance the image-specific priors learned by DIP.

Potential Benefits:

DIP leverages the architectural bias of CNNs to generate high-quality initial estimates of images without needing a large dataset for training. Starting the reverse diffusion process with these estimates can reduce the number of denoising steps required, which reduce training duration, and potentially improve the quality of the generated images.

With the high-quality initial estimate from DIP, DDPM can focus on refining the image through its probabilistic denoising steps. This can reduce the number of steps needed and enhance the final image quality.

Limitations:

Combining both models increases computational complexity, as it involves iterative optimization processes from both DIP and DDPM.

Experimental Verification

Experiment Design

Build a DIP model to generate initial priors.

```python
# Simple convolutional neural network
class DeepImagePriorNet(nn.Module):
    def __init__(self):
        super(DeepImagePriorNet, self).__init__()

        # Define the layers of the model
        self.layers = nn.Sequential(
            nn.Conv2d(in_channels=3, out_channels=64, kernel_size=3, stride=1, padding=1),
            nn.BatchNorm2d(64),
            nn.ReLU(inplace=True),
            nn.Conv2d(in_channels=64, out_channels=64, kernel_size=3, stride=1, padding=1),
            nn.BatchNorm2d(64),
            nn.ReLU(inplace=True),
            nn.Conv2d(in_channels=64, out_channels=3, kernel_size=3, stride=1, padding=1),
            nn.Sigmoid()
        )

    def forward(self, x):
        # Forward pass through the model
        return self.layers(x)
```

Train a DIP model to generate one initial priors. Repeat the process.

```
for i in range (450):
  if i > 399:
    print(i)
    net = DeepImagePriorNet()
    initial_prior = train_dip_model(net,images[i])
    initial_priors.append(initial_prior)
```

Train a DDPM model

Use the output of DIP model as the input of DDPM. I use UNet2DModel as base here.

```
from diffusers import UNet2DModel

model = UNet2DModel(
    sample_size=config.image_size,
    in_channels=3,
    out_channels=3,
    layers_per_block=2,
    block_out_channels=(128, 128, 256, 256, 512, 512),
    down_block_types=(
        "DownBlock2D",
        "DownBlock2D",
        "DownBlock2D",
        "DownBlock2D",
        "AttnDownBlock2D",
        "DownBlock2D",
    ),
    up_block_types=(
        "UpBlock2D",
        "AttnUpBlock2D",
        "UpBlock2D",
        "UpBlock2D",
        "UpBlock2D",
        "UpBlock2D",
    ),
)
```

Evaluation

I use PSNR and training time as metrics. I had tried using SSIM before, but due to the lack of data, it always showed zero. Therefore, I removed it from my metrics after attempting it a few times.

I implemented a double for-loop in the evaluation to take the average results. There are 16 test cases in this evaluation. Each generated output image is compared with 16 original images to calculate the PSNR

```
total_psnr = 0
total_ssim = 0
for image in images:
  # PIL(size) to numpy(shape)
  output = np.array(image)/255
  single_psnr = 0
  single_ssim = 0
  for i in range(16):
    # tensor to a NumPy array
    # Transpose from (C, H, W) to (H, W, C)
    input = dataset[i]['images'].squeeze(0)
    input = input.permute(1, 2, 0).numpy()
    # input = dataset[i].squeeze(0).permute(1,2,0).numpy()
    psnr = calculate_psnr(input,output)
    ssim = calculate_ssim(input,output)
    single_psnr += psnr
    single_ssim += ssim
  print('SINGLE PSNR: ',single_psnr/16,'SINGLE SSIM: ', single_ssim/16)
  total_psnr += single_psnr/16
  total_ssim += single_ssim/16
print('PSNR:',round(total_psnr/16,3))
print('SSIM:',round(total_ssim/16,3))
```

Results

DDPM 10 epoch vs 20 epoch (Dataset size = 1000)

10 epoch

PSNR: 7.997

20 epoch

PSNR: 8.194

The DDPM model are able to generate similar images as input with enough size of dataset.

DIP+DDPM vs DDPM

epoch = 10

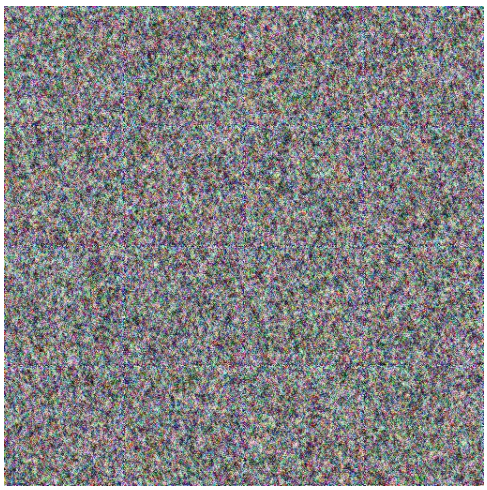
data size=100

With DIP

Timesteps = 1000

Training time: 47.78 seconds

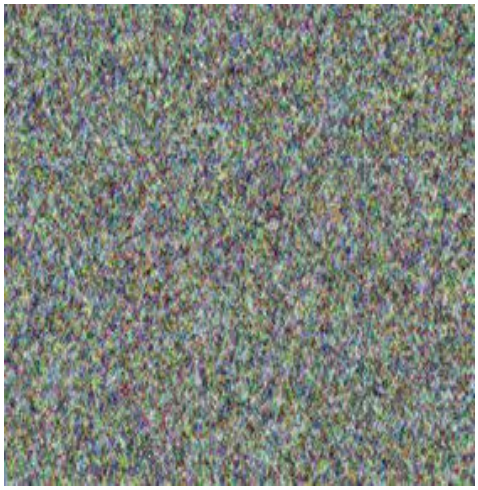
PSNR: 6.92



Timesteps = 500

Training time: 46.96 seconds

PSNR: 6.963



Without DIP

Timesteps = 1000

Training time: 52.79 seconds

PSNR: 6.966

Timesteps = 500

Training time: 50.99 seconds

PSNR: 6.979

We can't see too much difference from the pictures, but the duration time for the DIP + DDPM architecture is shorter than DDPM.

DIP+DDPM vs DDPM

epoch = 30

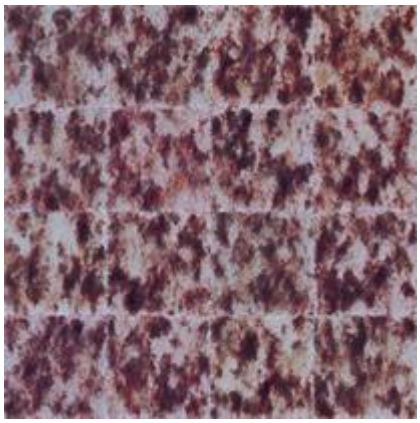
data size=100

With DIP

Timesteps = 1000

Training time: 147.57 seconds

PSNR: 7.616



Timesteps = 500

Training time: 133.33 seconds

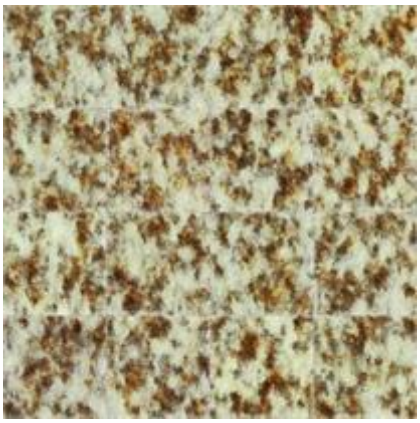
PSNR: 7.485

Without DIP

Timesteps = 1000

Training time: 166.78 seconds

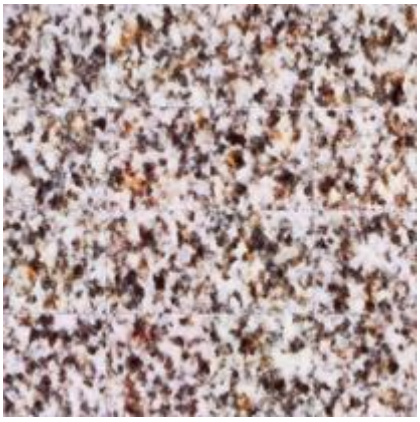
PSNR: 6.96



Timesteps = 500

Training time: 160.66 seconds

PSNR: 6.654



Obviously, the DIP + DDPM architecture has a shorter training duration compared to DDPM under the same parameters. The performance of DIP + DDPM also surpasses that of DDPM, achieving a higher PSNR. With smaller timesteps, the duration is further reduced, but this comes at the cost of performance.

DIP + DDPM

Epoch = 30

Data set = 500

Timesteps = 500

With DIP

Add noise: linear

Training time: 667.84 seconds

PSNR: 7.169

Add noise: squaredcos_cap_v2

Training time: 696.27 seconds

PSNR: 7.399

Without DIP

Add noise: linear

Training time: 785.79 seconds

PSNR: 7.727

Add noise: squaredcos_cap_v2

Training time: 802.26 seconds

PSNR: 7.868

With the increasing data size, the results are contrary to initial expectations. DDPM has better image quality compared to the hybrid system, but the DIP + DDPM combination still has a shorter duration time. Additionally, the denoising schedule 'squaredcos_cap_v2' generates better image quality than the default 'linear' schedule. ‘

Ablation Studies and Analysis

I trained a DDPM model with 1000 images has the best performance, however using DIP to generate image takes a lot of time, so I initially reduced the dataset to 100 images and later increased it to 500 images.

Experiments proved that hybrid DIP and DDPM outperforms the standard DDPM in terms of image reconstruction quality at less training data, and the duration times are always shorter than DDPM.

Reducing the timesteps makes the duration shorter but worsens performance.

Denoising schedules also impact the performance and duration. The default denoising schedule, “linear,” requires less time compared to “squaredcos_cap_v2,” but “squaredcos_cap_v2” yields better results.

In summary, the hybrid DIP and DDPM system surpasses the limitations of standard DDPM. However, it requires lots of time and resources to generate the initial priors.