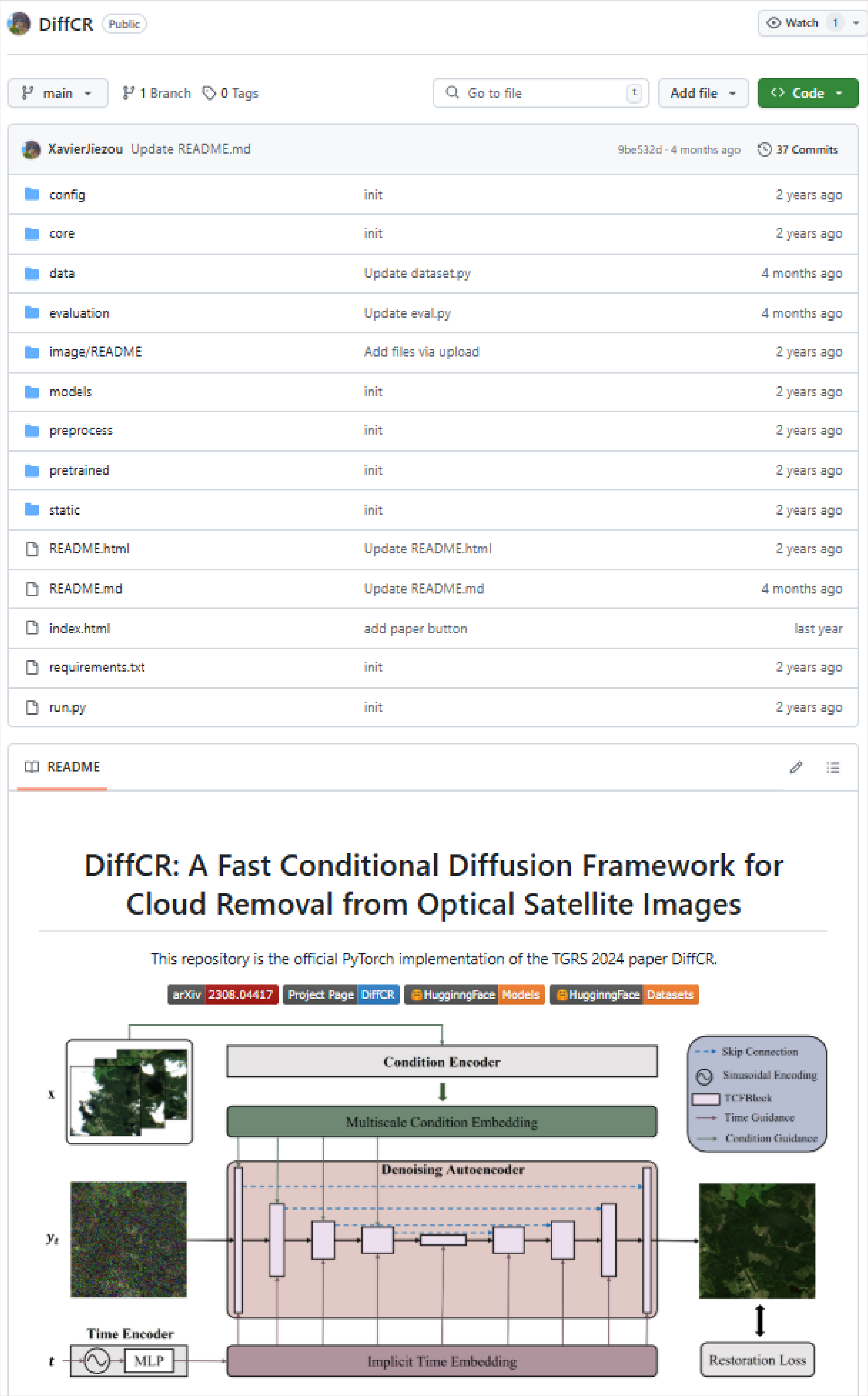

DiffCR 코드 리뷰

DiffCR 코드 리뷰에 대한 소개입니다.
논문에서 구현된 핵심 구성 요소에 집중하여 전체 코드 구조와 실행 흐름을 분석합니다.

전체 구조 개요

핵심 구성 요소

- config/: 실험 설정을 담은 JSON 파일
- core/: 공통 모듈 정의
- data/: 데이터셋 로딩 및 전처리 클래스 정의
- models/: 네트워크 구조 및 학습 파이프라인 정의
- evaluation/: FID, PSNR, LPIPS 등 평가 지표 계산 함수
- pretrained/: 사전학습 모델
- static/: 웹 기반 데모 환경 지원

전체 시스템 구조를 json 설정 기반으로 설명합니다.

실행 파일

- run.py: 프레임워크 메인 실행 파일

전체 실행 흐름

1

실행 시작

- ✓ run.py 실행
- ✓ argparse로 config 파일 및 phase 설정값을 입력받음

2

설정 파싱

- ✓ core/praser.py의 parse() 함수로 JSON 설정 파일 파싱
- ✓ phase, GPU 설정, 경로 세팅 등 전처리 수행

3

데이터셋 및 데이터로더 정의

- ✓ define_dataset()으로 Dataset 클래스(Sen2_MTC_New_Multi) 초기화
- ✓ define_dataloader()로 Dataloader 구성

4

모델 및 네트워크 생성

- ✓ create_model() 호출
- ✓ models/model.py의 Palette 클래스 로드
- ✓ 내부에서 네트워크(Network) 및 UNet 구조 초기화

5

학습 수행

- ✓ train 또는 test: phase에 따라 forward 진행
- ✓ 결과 저장 및 TensorBoard 로그 기록

```python
if __name__ == '__main__':
    args = parse_args()
    opt = parse(args)
    ...
    main_worker(...)
```

```python
# core/praser.py
def parse(args):
    ...
    opt = json.load(...)  # JSON config 파일 로드
    ...
    opt['phase'] = args.phase
```

```python
# data/__init__.py
def define_dataloader(logger, opt):
    phase_dataset, val_dataset = define_dataset(logger, opt)
    ...
    dataloader = DataLoader(phase_dataset, ...)
```

```python
# models/__init__.py
def create_model(opt, logger):
    model = init_obj(opt['model']['which_model'], ...)
    return model
```

```python
# run.py
    if opt['phase'] == 'train':
        model.train()
    else:
        model.test()
```

1. 실행 시작

#run.py

```python
import argparse
import os
import warnings

import torch
import torch.multiprocessing as mp

import core.praser as Praser
import core.util as Util
from core.logger import VisualWriter, InfoLogger
from data import define_dataloader
from models import create_model, define_network, define_loss, define_metric


# 각 GPU에서 실행될 train/test 프로세스 정의
def main_worker(gpu, ngpus_per_node, opt):
    if 'local_rank' not in opt:
        opt['local_rank'] = opt['global_rank'] = gpu
    # 멀티 GPU 훈련 지원
    if opt['distributed']:
        torch.cuda.set_device(int(opt['local_rank']))
        print('using GPU {} for training'.format(int(opt['local_rank'])))
        torch.distributed.init_process_group(
            backend = 'nccl',
            init_method = opt['init_method'],
            world_size = opt['world_size'],
            rank = opt['global_rank'],
            group_name = 'mtorch'
        )

    # 난수 시드 고정 및 cuDNN 최적화
    torch.backends.cudnn.enabled = True
    Util.set_seed(opt['seed'])

    # 로그 및 시각화 설정
    phase_logger = InfoLogger(opt)
    phase_writer = VisualWriter(opt, phase_logger)
    phase_logger.info('Creat the log file in directory {}.\n'.format(opt['path']['experiments_root']))

    # 데이터 로더, 네트워크 설정
    phase_loader, val_loader = define_dataloader(phase_logger, opt)
    networks = [define_network(phase_logger, opt, item_opt) for item_opt in opt['model']['which_networks']]

    # 평가 지표, loss 설정
    metrics = [define_metric(phase_logger, item_opt) for item_opt in opt['model']['which_metrics']]
    losses = [define_loss(phase_logger, item_opt) for item_opt in opt['model']['which losses']]

    model = create_model(
        opt = opt,
        networks = networks,
        phase_loader = phase_loader,
        val_loader = val_loader,
        losses = losses,
        metrics = metrics,
        logger = phase_logger,
        writer = phase_writer
    )

    phase_logger.info('Begin model {}.'.format(opt['phase']))
    try:
        if opt['phase'] == 'train':
            model.train()
        else:
            model.test()
    finally:
        phase_writer.close()
```

```python
if __name__ == '__main__':
    parser = argparse.ArgumentParser()
    parser.add_argument('-c', '--config', type=str, default='config/colorization_mirflickr25k.json',
help='JSON file for configuration')
    parser.add_argument('-p', '--phase', type=str, choices=['train','test'], help='Run train or test',
default='train')
    parser.add_argument('-b', '--batch', type=int, default=None, help='Batch size in every gpu')
    parser.add_argument('-gpu', '--gpu_ids', type=str, default=None)
    parser.add_argument('-d', '--debug', action='store_true')
    parser.add_argument('-P', '--port', default='21012', type=str)

    # 인자 파싱 및 설정 로드
    args = parser.parse_args()
    opt = Praser.parse(args)

    # GPU 설정
    gpu_str = ','.join(str(x) for x in opt['gpu_ids'])
    os.environ['CUDA_VISIBLE_DEVICES'] = gpu_str
    print('export CUDA_VISIBLE_DEVICES={}'.format(gpu_str))

    # 멀티 GPU 사용 - DistributedDataParallel(DDP) and multiprocessing
    if opt['distributed']:
        ngpus_per_node = len(opt['gpu_ids'])
        opt['world_size'] = ngpus_per_node
        opt['init_method'] = 'tcp://127.0.0.1:' + args.port
        mp.spawn(main_worker, nprocs=ngpus_per_node, args=(ngpus_per_node, opt))
    else:
        opt['world_size'] = 1
        main_worker(0, 1, opt)
```

2. 설정 파싱

```python
# core/praser.py
def parse(args):
    json_str = ''
    with open(args.config, 'r') as f:
        for line in f:
            line = line.split('//')[0] + '\n'
            json_str += line
    opt = json.loads(json_str, object_pairs_hook=OrderedDict)

    # 인자 값으로 config 값 덮어쓰기
    opt['phase'] = args.phase
    if args.gpu_ids is not None:
        opt['gpu_ids'] = [int(id) for id in args.gpu_ids.split(',')]
    if args.batch is not None:
        opt['datasets'][opt['phase']]['dataloader']['args']['batch_size'] = args.batch

    ...
    return dict_to_nonedict(opt)
```

- parse 함수 – json 파일 파싱

- json 설정 파일 기반 실행

✓ 데이터셋

```
# datasets
"datasets": {
  "train": {
    "which_dataset": {
      "name": ["data.dataset", "Sen2_MTC_New_Multi"],
      "args": {
        "data_root": "../pmaa/data",
        "mode": "train"
      }
    },
    "dataloader": {
      "args": {
        "batch_size": 8,
        "num_workers": 8,
        "shuffle": true
      }
    }
  }
}
```

✓ 모델

```
# model
"model": {
  "which_model": {
    "name": ["models.model", "Palette"],
    "args": {
      "task": "decloud",
      "sample_num": 8
    }
  }
}
```

✓ 네트워크

```
# network
"which_networks": [
  {
    "name": ["models.network_x0_dpm_solver", "Network"],
    "args": {
      "module_name": "ours_double_encoder_splitcaCond_splitcaUnet",
      "unet": {
        "img_channel": 3,
        "width": 64
      },
      "beta_schedule": {
        "train": { "schedule": "sigmoid", "n_timestep": 2000 },
        "test": { "schedule": "sigmoid", "n_timestep": 1000 }
      }
    }
  }
]
```

3. 데이터셋 및 데이터로더 정의

- data/__init__.py

```
def define_dataloader(logger, opt):
    phase_dataset, val_dataset = define_dataset(logger, opt)
    dataloader = DataLoader(phase_dataset, ...)
    return dataloader, val_dataloader
```

✓ DataLoader 구성

```
def define_dataset(logger, opt):
    dataset_opt = opt['datasets'][opt['phase']]['which_dataset']
    phase_dataset = init_obj(dataset_opt, logger, ...)
    ...
    return phase_dataset, val_dataset
```

✓ 데이터셋 클래스 정의

- data/dataset.py

```
def __getitem__(self, index):
    # 파일 경로들을 불러와 이미지 읽기 수행
    cloud_image_path0, cloud_image_path1, cloud_image_path2 = \
        self.filepair[index][0], self.filepair[index][1], self.filepair[index][2]
    cloudless_image_path = self.filepair[index][3]
    image_cloud0 = self.image_read(cloud_image_path0)
    image_cloud1 = self.image_read(cloud_image_path1)
    image_cloud2 = self.image_read(cloud_image_path2)
    image_cloudless = self.image_read(cloudless_image_path)

    ret = {}
    ret['gt_image'] = image_cloudless[:3, :, :]
    ret['cond_image'] = torch.cat([image_cloud0[:3, :, :], image_cloud1[:3, :, :], image_cloud2[:3, :, :]])
    ret['path'] = self.image_name[index] + ".png"
    return ret
```

✓ cloud image 3장 + cloudless img 1장 구성

```
# tiff 파일
def image_read(self, image_path):
    img = tiff.imread(image_path)
    img = (img / 1.0).transpose((2, 0, 1))  # HWC → CHW

    if self.mode == 'train':
        if self.augment_flip_param[self.index // 4] != 0:
            img = np.flip(img, self.augment_flip_param[self.index // 4])
        if self.augment_rotation_param[self.index // 4] != 0:
            img = np.rot90(img, self.augment_rotation_param[self.index // 4], (1, 2))
        self.index += 1
    ...
    image = torch.from_numpy(img.copy()).float() / 10000.0
    mean = torch.as_tensor([0.5]*4).view(-1, 1, 1)
    std = torch.as_tensor([0.5]*4).view(-1, 1, 1)
    image.sub_(mean).div_(std)

    return image
```

✓ tiff 파일 로드 – augment, 정규화

4. 모델 및 네트워크 생성

- 모델 생성

```
# model
"model": {
  "which_model": {
    "name": ["models.model", "Palette"],
    "args": {
      "task": "decloud",
      "sample_num": 8
    }
  }
}
```

- models/__init__.py

```
def create_model(**cfg_model):
    ...
    model = init_obj(model_opt, logger, default_file_name='models.model', init_type='Model')
    return model
```

✓ init_obj()를 통해 모델 객체 생성

- models/model.py

```
class Palette(BaseModel):
    def __init__(self, networks, losses, sample_num, task, optimizers, ema_scheduler=None, **kwargs):
        # Basemodel 초기화를 위해 kwargs 반드시 전달 필요
        super(Palette, self).__init__(**kwargs)

        # networks, dataloader, optimizers, losses, etc.
        self.loss_fn = losses[0]
        self.netG = networks[0]
        if ema_scheduler is not None:
            self.ema_scheduler = ema_scheduler
            self.netG_EMA = copy.deepcopy(self.netG) # netG -> EMA용으로 따로 보관
            self.EMA = EMA(beta=self.ema_scheduler['ema_decay'])
```

✓ Palette 클래스 초기화

```
    def train_step(self):
        self.netG.train()
        self.train_metrics.reset()
        for train_data in tqdm.tqdm(self.phase_loader):
            self.set_input(train_data)
            self.optG.zero_grad()
            loss = self.netG(self.gt_image, self.cond_image, mask=self.mask)
            loss.backward()
            self.optG.step()

            self.iter += self.batch_size
            self.writer.set_iter(self.epoch, self.iter, phase='train')
            self.train_metrics.update(self.loss_fn.__name__, loss.item())
            if self.iter % self.opt['train']['log_iter'] == 0:
                for key, value in self.train_metrics.result().items():
                    self.logger.info('{:5s}: {}\t'.format(str(key), value))
                    self.writer.add_scalar(key, value)
                for key, value in self.get_current_visuals().items():
                    self.writer.add_images(key, value, dataformats='CHW')
            if self.ema_scheduler is not None:
                if self.iter > self.ema_scheduler['ema_start'] and self.iter % self.ema_scheduler['ema_iter'] == 0:
                    self.EMA.update_model_average(self.netG_EMA, self.netG)


            for scheduler in self.schedulers:
                scheduler.step()
            return self.train_metrics.result()
```

✓ 학습 루프

4. 모델 및 네트워크 생성

- 네트워크 생성

```
# network
"which_networks": [
  {
    "name": ["models.network_x0_dpm_solver", "Network"],
    "args": {
      "module_name": "ours_double_encoder_splitcaCond_splitcaUnet",
      "unet": {
        "img_channel": 3,
        "width": 64
      },
      "beta_schedule": {
        "train": { "schedule": "sigmoid", "n_timestep": 2000 },
        "test": { "schedule": "sigmoid", "n_timestep": 1000 }
      }
    }
  }
]
```

- models/__init__.py

```python
def define_network(logger, opt, network_opt):
    # define network with weights initialization
    net = init_obj(network_opt, logger, default_file_name='models.network', init_type='Network')

    if opt['phase'] == 'train':
        logger.info('Network [{}] weights initialize using [{:s}] method.'.format(net.__class__.__name__,
network_opt['args'].get('init_type', 'default')))
        net.init_weights()
    return net
```

✓ init_obj()를 통해 모델 객체 생성

- models/network_x0_dpm_solver.py

```python
class Network(BaseNetwork):
    def __init__(self, unet, beta_schedule, module_name='ours_double_encoder_splitcaCond_splitcaUnet', **kwargs):
        super(Network, self).__init__(**kwargs)

        if module_name == "ours_double_encoder_splitcaCond_splitcaUnet":
            from .ours.nafnet_double_encoder_splitcaCond_splitcaUnet import UNet
        else:
            raise NotImplementedError(f"Unknown module_name: {module_name}")

        self.denoise_fn = UNet(**unet)
        self.beta_schedule = beta_schedule
```

✓ json 설정의 "module_name" 값을 기준으로 UNet 구조 import

```python
    # 노이즈 수준 설정 (beta_schedule)
    def set_new_noise_schedule(self, device=torch.device('cuda'), phase='train'):
        # PyTorch Tensor 기본 설정 - to_torch 호출 시 해당 설정으로 자동 변환됨됨
        to_torch = partial(torch.tensor, dtype=torch.float32, device=device)

        betas = make_beta_schedule(**self.beta_schedule[phase])
        betas = betas.detach().cpu().numpy() if isinstance(betas, torch.Tensor) else betas
        alphas = 1. - betas

        # betas = np.linspace(1e-6, 1e-2, 1000)
        # betas.shape (1000,)
        timesteps, = betas.shape
        self.num_timesteps = int(timesteps)

        gammas = np.cumprod(alphas, axis=0)
        gammas_prev = np.append(1., gammas[:-1])

        # 학습되지 않는 변수(고정값) 모델에 저장 - q(x_t | x_{t-1}) 과정
        self.registered_buffer('gammas', to_torch(gammas))
        self.registered_buffer('sqrt_recip_gammas', to_torch(np.sqrt(1. / gammas)))
        self.registered_buffer('sqrt_recipm1_gammas', to_torch(np.sqrt(1. / gammas - 1)))

        # p(x_{t-1} | x_t, x_0)
        posterior_variance = betas * (1. - gammas_prev) / (1. - gammas)
        self.registered_buffer('posterior_log_variance_clipped', to_torch(np.log(np.maximum(posterior_variance, 1e-20))))
        self.registered_buffer('posterior_mean_coef1', to_torch(betas * np.sqrt(gammas_prev) / (1. - gammas)))
        self.registered_buffer('posterior_mean_coef2', to_torch((1. - gammas_prev) * np.sqrt(alphas) / (1. - gammas)))
```

✓ beta_schedule에 따라 노이즈 분포 파라미터 계산하여 등록

4. 모델 및 네트워크 생성

```python
# Reverse Process - μ_θ(y_t) 와 (σ_t)^2를 구하는 함수
def p_mean_variance(self, y_t, t, clip_denoised: bool, y_cond=None):
    noise_level = extract(self.gammas, t, x_shape=(1, 1)).to(y_t.device)
    # x_0을 직접 예측하도록 바뀐 부분
    # y_0_hat = self.predict_start_from_noise(
    #     y_t, t=t, noise=self.denoise_fn(torch.cat([y_cond, y_t], dim=1), noise_level)
    # )

    y_0_hat = self.denoise_fn(
        torch.cat([y_cond, y_t], dim=1), noise_level
    )
    if clip_denoised:
        y_0_hat.clamp(-1., 1.)

    model_mean, posterior_log_variance = self.q_posterior(
        y_0_hat=y_0_hat, y_t=y_t, t=t
    )
    return model_mean, posterior_log_variance
```

✓ y_0를 예측하여 posterior 평균 및 분산 계산

- models/network.py

```python
def predict_start_from_noise(self, y_t, t, noise):
    return(
        extract(self.sqrt_recip_gammas, t, y_t.shape) * y_t -
        extract(self.sqrt_recipm1_gammas, t, y_t.shape) * noise
    )
```

$$\hat{x}_0 = \frac{1}{\sqrt{\bar{\alpha}_t}} \cdot y_t - \frac{\sqrt{1-\bar{\alpha}_t}}{\sqrt{\bar{\alpha}_t}} \cdot \hat{\epsilon}$$

```python
# Reverse Process - y_T에서 y_0까지 반복 수행 (전체 복원 과정) / p_sample 여러 번 호출
@torch.no_grad()
def restoration(self, y_cond, y_t=None, y_0=None, mask=None, sample_num=8):
    b, *_ = y_cond.shape

    assert self.num_timesteps > sample_num, "num_timesteps must be greater than sample_num"
    sample_inter = (self.num_timesteps // sample_num)

    y_t = default(y_t, lambda: torch.randn_like(y_0))
    ret_arr = y_t
    # for i in tqdm(reversed(range(0, self.num_timesteps)), desc='sampling loop time step', total=self.num_timesteps):
    #     t = torch.full((b,), i, device=y_cond.device, dtype=torch.long)
    #     y_t = self.p_sample(y_t, t, y_cond=y_cond)
    #     if mask is not None:
    #         y_t = y_0 * (1. - mask) + mask * y_t
    #     if i % sample_inter == 0:
    #         ret_arr = torch.cat([ret_arr, y_t], dim=0)

    # DPM-Solver++ 기반 고속 복원 샘플링 수행
    # 기존의 timestep 반복 샘플링을 대체하며, 더 적은 step 수로 빠르고 정밀하게 복원 가능
    noise_schedule = NoiseScheduleVP(schedule='discrete', betas=torch.from_numpy(self.betas))
    model_fn = model_wrapper(
        self.denoise_fn,
        noise_schedule,
        model_type='x_start',
        model_kwargs={},
        guidance_type='classifier-free',
        condition=y_cond,
        unconditional_condition=None,
        guidance_scale=1.,
    )
    dpm_solver = DPM_Solver(
        model_fn,
        noise_schedule,
        algorithm_type='dpmsolver++',
        corecting_x0_fn='dynamic_thresholding',
    )
    y_t = dpm_solver.sample(
        y_t,
        steps=20, # 10, 12, 15, 20, 25, 50, 100
        order=2,
        skip_type='time_uniform',
        method='multistep',
        denoise_to_zero=True,
    )
    return y_t, ret_arr
```

✓ DPM-Solver++ 기반으로 빠르고 정밀한 복원을 수행

4. 모델 및 네트워크 생성

- models/ours/nafnet_double_encoder_splitcaCond_splitcaUnet.py

```python
# Sinusoidal Encoding
def gamma_embedding(gammas, dim, max_period=10000):
    #
    half = dim // 2
    freqs = torch.exp(
        -math.log(max_period) * torch.arange(start=0,
                                              end=half,
                                              dtype=torch.float32) / half
    ).to(device=gammas.device)
    args = gammas[:, None].float() * freqs[None]
    embedding = torch.cat([torch.cos(args), torch.sin(args)], dim=-1)
    if dim % 2:
        embedding = torch.cat(
            [embedding, torch.zeros_like(embedding[:, :1])], dim=-1
        )
    return embedding


class LayerNorm2d(nn.Module):
    def __init__(self, channels, eps=1e-6):
        super(LayerNorm2d, self).__init__()
        self.register_parameter('weight', nn.Parameter(torch.ones(channels)))
        self.register_parameter('bias', nn.Parameter(torch.zeros(channels)))
        self.eps = eps

    def forward(self, x):
        return LayerNormFunction.apply(x, self.weight, self.bias, self.eps)


# 채널 dimension 반으로 나눈 후, elementwise 곱셈
class SimpleGate(nn.Module):
    def forward(self, x):
        x1, x2 = x.chunk(2, dim=1)
        return x1 * x2
```

✓ Block 구현에 필요한 클래스 및 함수

```python
# Condition Block
class CondNAFBlock(nn.Module):
    def __init__(self, c, DW_Expand=2, FFN_Expand=2, drop_out_rate=0.):
        super().__init__()
        dw_channel = c * DW_Expand
        self.conv1 = nn.Conv2d(in_channels=c, out_channels=dw_channel,
                               kernel_size=1, padding=0, stride=1, groups=1, bias=True)
        self.conv2 = nn.Conv2d(in_channels=dw_channel, out_channels=dw_channel,
                               kernel_size=3, padding=1, stride=1, groups=dw_channel, bias=True)
        self.conv3 = nn.Conv2d(in_channels=dw_channel // 2, out_channels=c,
                               kernel_size=1, padding=0, stride=1, groups=1, bias=True)

        self.sca_avg = nn.Sequential(
            nn.AdaptiveAvgPool2d(1),
            nn.Conv2d(in_channels=dw_channel // 4, out_channels=dw_channel // 4,
                      kernel_size=1, padding=0, stride=1, groups=1, bias=True),
        )
        self.sca_max = nn.Sequential(
            nn.AdaptiveMaxPool2d(1),
            nn.Conv2d(in_channels=dw_channel // 4, out_channels=dw_channel // 4,
                      kernel_size=1, padding=0, stride=1, groups=1, bias=True),
        )
        self.sg = SimpleGate()

        ffn_channel = FFN_Expand * c
        self.conv4 = nn.Conv2d(in_channels=c, out_channels=ffn_channel,
                               kernel_size=1, padding=0, stride=1, groups=1, bias=True)
        self.conv5 = nn.Conv2d(in_channels=ffn_channel // 2, out_channels = c,
                               kernel_size=1, padding=0, stride=1, groups=1, bias=True)

        self.norm1 = LayerNorm2d(c)
        self.norm2 = LayerNorm2d(c)

        self.dropout1 = nn.Dropout(drop_out_rate) if drop_out_rate > 0. else nn.Identity()
        self.dropout2 = nn.Dropout(drop_out_rate) if drop_out_rate > 0. else nn.Identity()

        self.beta = nn.Parameter(torch.zeros((1, c, 1, 1)), requires_grad=True)
        self.gamma = nn.Parameter(torch.zeros((1, c, 1, 1)), requires_grad=True)

    def forward(self, input):
        x = input

        x = self.norm1(x)

        x = self.conv1(x)
        x = self.conv2(x)
        x = self.sg(x)
        x_avg, x_max = x.chunk(2, dim=1)
        x_avg = self.sca_avg(x_avg) * x_avg
        x_max = self.sca_max(x_max) * x_max
        x = torch.cat([x_avg, x_max], dim=1)

        x = self.conv3(x)
        x = self.dropout1(x)
        y = input + x * self.beta

        x = self.conv4(self.norm2(y))
        x = self.sg(x)
        x = self.conv5(x)

        x = self.dropout2(x)

        return y + x * self.gamma
```

4. 모델 및 네트워크 생성

```python
# Time Block
class NAFBlock(EmbedBlock):
    def __init__(self, c, DW_Expand=2, FFN_Expand=2, drop_out_rate=0.):
        ...
        # MLP - 최종 Time
        self.time_emb = nn.Sequential(
            nn.SiLU(),
            nn.Linear(256, c),
        )

    def forward(self, input, t):
        ...
        y = input + x * self.beta
        y = y + self.time_emb(t)[..., None, None]

        ...

        return y + x * self.gamma
```

✓ NAFBlock – time emb

```python
# Condition Block
class CondNAFBlock(nn.Module):
    def __init__(self, c, DW_Expand=2, FFN_Expand=2, drop_out_rate=0.):
        super().__init__()
        dw_channel = c * DW_Expand
        self.conv1 = nn.Conv2d(in_channels=c, out_channels=dw_channel,
                               kernel_size=1, padding=0, stride=1, groups=1, bias=True)
        self.conv2 = nn.Conv2d(in_channels=dw_channel, out_channels=dw_channel,
                               kernel_size=3, padding=1, stride=1, groups=dw_channel, bias=True)
        self.conv3 = nn.Conv2d(in_channels=dw_channel // 2, out_channels=c,
                               kernel_size=1, padding=0, stride=1, groups=1, bias=True)

        self.sca_avg = nn.Sequential(
            nn.AdaptiveAvgPool2d(1),
            nn.Conv2d(in_channels=dw_channel // 4, out_channels=dw_channel // 4,
                      kernel_size=1, padding=0, stride=1, groups=1, bias=True),
        )
        self.sca_max = nn.Sequential(
            nn.AdaptiveMaxPool2d(1),
            nn.Conv2d(in_channels=dw_channel // 4, out_channels=dw_channel // 4,
                      kernel_size=1, padding=0, stride=1, groups=1, bias=True),
        )
        self.sg = SimpleGate()

        ffn_channel = FFN_Expand * c
        self.conv4 = nn.Conv2d(in_channels=c, out_channels=ffn_channel,
                               kernel_size=1, padding=0, stride=1, groups=1, bias=True)
        self.conv5 = nn.Conv2d(in_channels=ffn_channel // 2, out_channels = c,
                               kernel_size=1, padding=0, stride=1, groups=1, bias=True)

        self.norm1 = LayerNorm2d(c)
        self.norm2 = LayerNorm2d(c)

        self.dropout1 = nn.Dropout(drop_out_rate) if drop_out_rate > 0. else nn.Identity()
        self.dropout2 = nn.Dropout(drop_out_rate) if drop_out_rate > 0. else nn.Identity()

        self.beta = nn.Parameter(torch.zeros((1, c, 1, 1)), requires_grad=True)
        self.gamma = nn.Parameter(torch.zeros((1, c, 1, 1)), requires_grad=True)

    def forward(self, input):
        x = input

        x = self.norm1(x)

        x = self.conv1(x)
        x = self.conv2(x)
        x = self.sg(x)
        x_avg, x_max = x.chunk(2, dim=1)
        x_avg = self.sca_avg(x_avg) * x_avg
        x_max = self.sca_max(x_max) * x_max
        x = torch.cat([x_avg, x_max], dim=1)

        x = self.conv3(x)
        x = self.dropout1(x)
        y = input + x * self.beta

        x = self.conv4(self.norm2(y))
        x = self.sg(x)
        x = self.conv5(x)

        x = self.dropout2(x)

        return y + x * self.gamma
```

✓ CondNAFBlock – condition emb

4. 모델 및 네트워크 생성

```python
# Condition Block
class CondNAFBlock(nn.Module):
    def __init__(self, c, DW_Expand=2, FFN_Expand=2, drop_out_rate=0.):
      ...

    def forward(self, input):
        x = input

        x = self.norm1(x)
        x = self.conv1(x)
        x = self.conv2(x)
        x = self.sg(x)
        x_avg, x_max = x.chunk(2, dim=1)
        x_avg = self.sca_avg(x_avg) * x_avg
        x_max = self.sca_max(x_max) * x_max
        x = torch.cat([x_avg, x_max], dim=1)

        x = self.conv3(x)
        x = self.dropout1(x)
        y = input + x * self.beta

        x = self.conv4(self.norm2(y))
        x = self.sg(x)
        x = self.conv5(x)

        x = self.dropout2(x)

        return y + x * self.gamma
```

✓ TCFBlock 구현

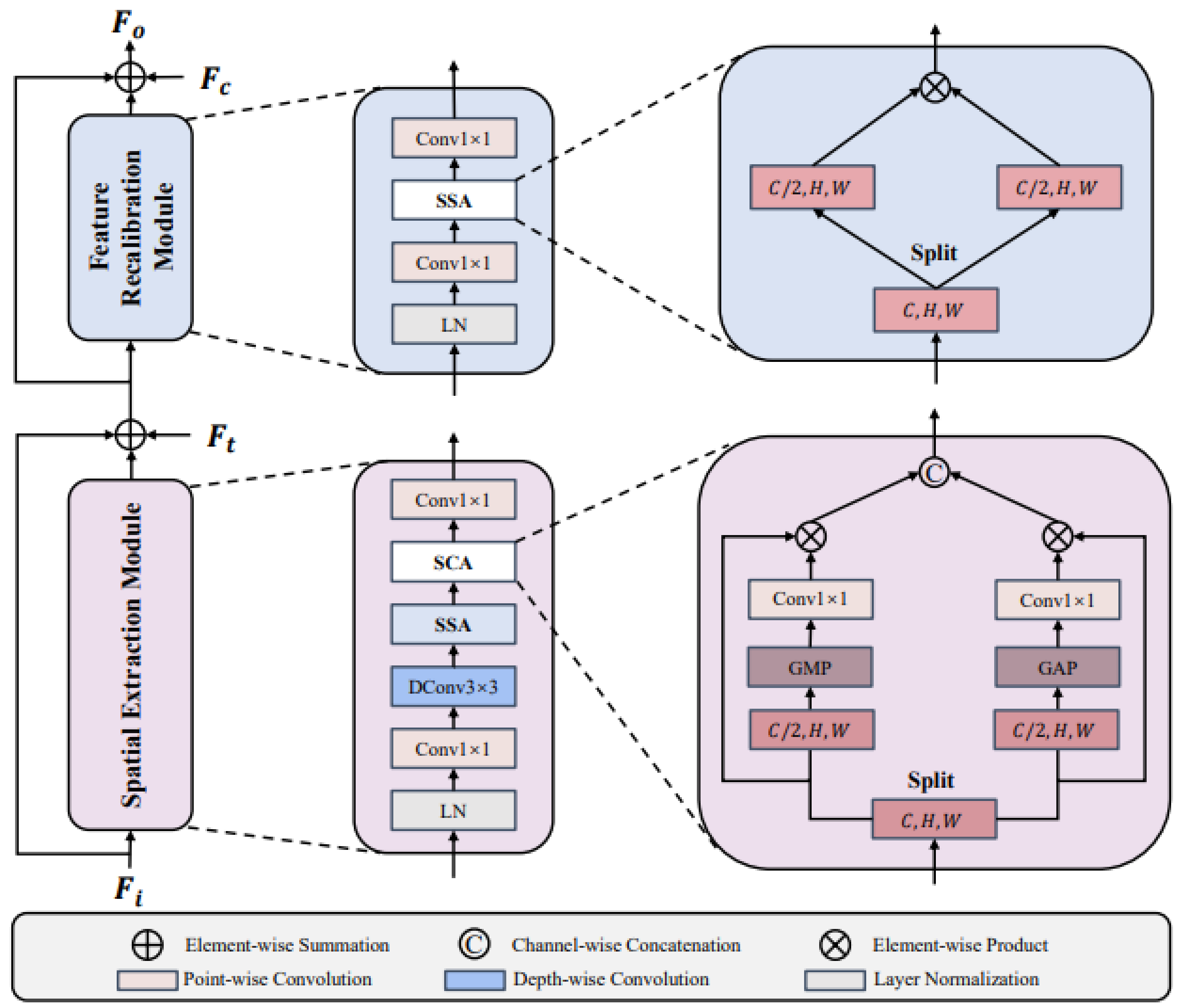

Fig. 4. Schematic diagram of our proposed time and condition fusion block.

4. 모델 및 네트워크 생성

```python
class UNet(nn.Module):
    def __init__(self,
                 img_channel=3,
                 width=64,
                 middle_blk_num=1,
                 enc_blk_nums=[1, 1, 1, 1],
                 dec_blk_nums=[1, 1, 1, 1],
    ):
        super().__init__()
        self.intro = nn.Conv2d(in_channels=img_channel, out_channels=width,
                               kernel_size=3, padding=1, stride=1, groups=1, bias=True)
        self.cond_intro = nn.Conv2d(in_channels=img_channel, out_channels=width,
                               kernel_size=3, padding=1, stride=1, groups=1, bias=True)
        self.ending = nn.Conv2d(in_channels=width, out_channels=3,
                               kernel_size=3, padding=1, stride=1, groups=1, bias=True)

        self.encoders = nn.ModuleList()
        self.cond_encoders = nn.ModuleList()
        self.decoders = nn.ModuleList()

        self.middle_blks = nn.ModuleList()

        self.ups = nn.ModuleList()
        self.downs = nn.ModuleList()
        self.cond_downs = nn.ModuleList()
```

✓ UNet 구조

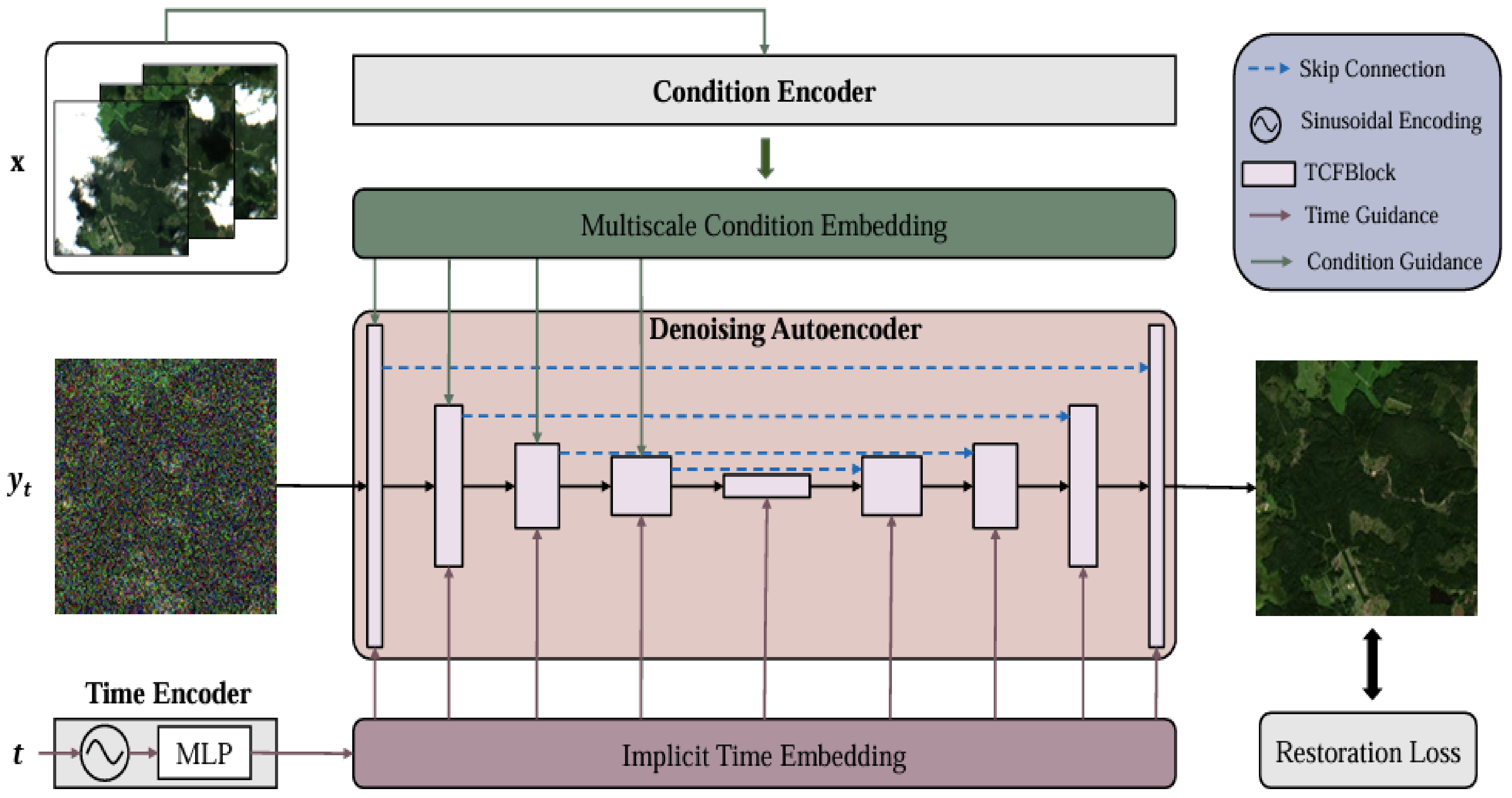

Fig. 3. Overall architecture of the conditional denoising model of DiffCR. The model consists of three parts: 1) a condition encoder, 2) a time encoder, and 3) a denoising autoencoder. The condition encoder and time encoder are responsible for extracting the spatial features of cloudy images $\mathbf{x}$ and temporal features of noise level t, respectively, which are then fed into the denoising autoencoder to guide the entire denoising process. The condition encoder and denoising autoencoder comprise several of our developed TCFBlocks (refer to Fig. 4). The time encoder comprises a sinusoidal encoding function and a multilayer perceptron (MLP). Ultimately, the data distribution of cloud-free images $\mathbf{y}_0$ is estimated under the supervision of the restoration loss.

4. 모델 및 네트워크 생성

```python
chan = width
for num in enc_blk_nums:
    self.encoders.append(
        EmbedSequential(
            *[NAFBlock(chan) for _ in range(num)]
        )
    )
    self.cond_encoders.append(
        nn.Sequential(
            *[CondNAFBlock(chan) for _ in range(num)]
        )
    )
    self.downs.append(
        nn.Conv2d(chan, 2*chan, 2, 2)
    )
    self.cond_downs.append(
        nn.Conv2d(chan, 2*chan, 2, 2)
    )
    chan = chan * 2

self.middle_blks = EmbedSequential(
    *[NAFBlock(chan) for _ in range(middle_blk_num)]
)

for num in dec_blk_nums:
    self.ups.append(
        nn.Sequential(
            nn.Conv2d(chan, chan * 2, 1, bias=False),
            nn.PixelShuffle(2),
        )
    )
    chan = chan // 2
    self.decoders.append(
        EmbedSequential(
            *[NAFBlock(chan) for _ in range(num)]
        )
    )

self.padder_size = 2 ** len(self.encoders)
self.emb = partial(gamma_embedding, dim=64)
self.map = nn.Sequential(
    nn.Linear(64, 256),
    nn.SiLU(),
    nn.Linear(256, 256),
)
```

```python
def forward(self, input, gammas):
    t = self.map(self.emb(gammas.view(-1,)))
    input = self.check_image_size(input)

    x1, x2, x3, x = input.chunk(4, dim=1)
    cond = torch.stack([x1, x2, x3], dim=1)
    b, n, c, h, w = cond.shape
    cond = cond.view(b*n, c, h, w)
    x = self.intro(x)
    cond = self.cond_intro(cond)

    encs = []

    for encoder, down, cond_encoder, cond_down in zip(self.encoders, self.downs, self.cond_encoders, self.cond_downs):
        x = encoder(x, t)
        cond = cond_encoder(cond)
        b, c, h, w = cond.shape
        tmp_cond = cond.view(b//3, 3, c, h, w).sum(dim=1)
        x = x + tmp_cond
        encs.append(x)
        x = down(x)
        cond = cond_down(cond)

    x = self.middle_blks(x, t)

    for decoder, up, enc_skip in zip(self.decoders, self.ups, encs[::-1]):
        x = up(x)
        x = x + enc_skip
        x = decoder(x, t)

    x = self.ending(x)

    return x
```

- ✓ CondNAFBlock – condition emb

5. 학습 수행

- models/model.py

```
class Palette(BaseModel):
    ...
    def train_step(self):
            self.netG.train()
            self.train_metrics.reset()
            for train_data in tqdm.tqdm(self.phase_loader):
                self.set_input(train_data)
                self.optG.zero_grad()
                loss = self.netG(self.gt_image, self.cond_image, mask=self.mask)
                loss.backward()
                self.optG.step()

                self.iter += self.batch_size
                self.writer.set_iter(self.epoch, self.iter, phase='train')
                self.train_metrics.update(self.loss_fn.__name__, loss.item())
                if self.iter % self.opt['train']['log_iter'] == 0:
                    for key, value in self.train_metrics.result().items():
                        self.logger.info('{:5s}: {}\t'.format(str(key), value))
                        self.writer.add_scalar(key, value)
                    for key, value in self.get_current_visuals().items():
                        self.writer.add_images(key, value, dataformats='CHW')
                if self.ema_scheduler is not None:
                    if self.iter > self.ema_scheduler['ema_start'] and self.iter % self.ema_scheduler['ema_iter'] == 0:
                        self.EMA.update_model_average(self.netG_EMA, self.netG)


                for scheduler in self.schedulers:
                    scheduler.step()
                return self.train_metrics.result()
```

✓ loss = self.netG(self.gt_image, self.cond_image, mask=self.mask)

기타 참고 구성 요소

항목	위치	간단한 설명
Core BaseModel 정의	`core/base_model.py`	모델 공통 학습 루프, EMA 및 로깅 인터페이스 제공
Core BaseNetwork 정의	`core/base_network.py`	UNet 등 네트워크 공통 인터페이스 관리
DPM-Solver++	`core/dpm_solver_pytorch.py`	고속 복원을 위한 고차 미분 기반 솔버
Logger 및 TensorBoard 연동	`core/logger.py`	학습 중 로그 기록, 이미지 시각화 지원
EMA (Exponential Moving Average)	`models/model.py`	안정적인 추론을 위한 가중치 평균화
Split Condition Encoder 구조	`models/nafnet_double_encoder_splitcaCond_splitcaUnet.py`	조건 이미지들을 분리 처리 후 합산
PixelShuffle Upsampling	`UNet.ups` 모듈 내	해상도 복원 시 interpolation 대신 shuffle 사용
마스크 관련 모듈 정리	`data/util/mask.py`, `Palette`, `Network` 등	마스크 생성, 손실 계산, 복원 시 마스크 적용까지 포함