2023
Samsung AI Challenge

Camera-Invariant Domain Adaptation

Public : 0.63171 (6등)
Private : 0.63704 (9등)

팀명 : 태훈_

대회 개요

- 대회 주제
  - 왜곡된 이미지에 대해서도 고성능의 이미지 분할을 수행하는 AI 알고리즘 개발

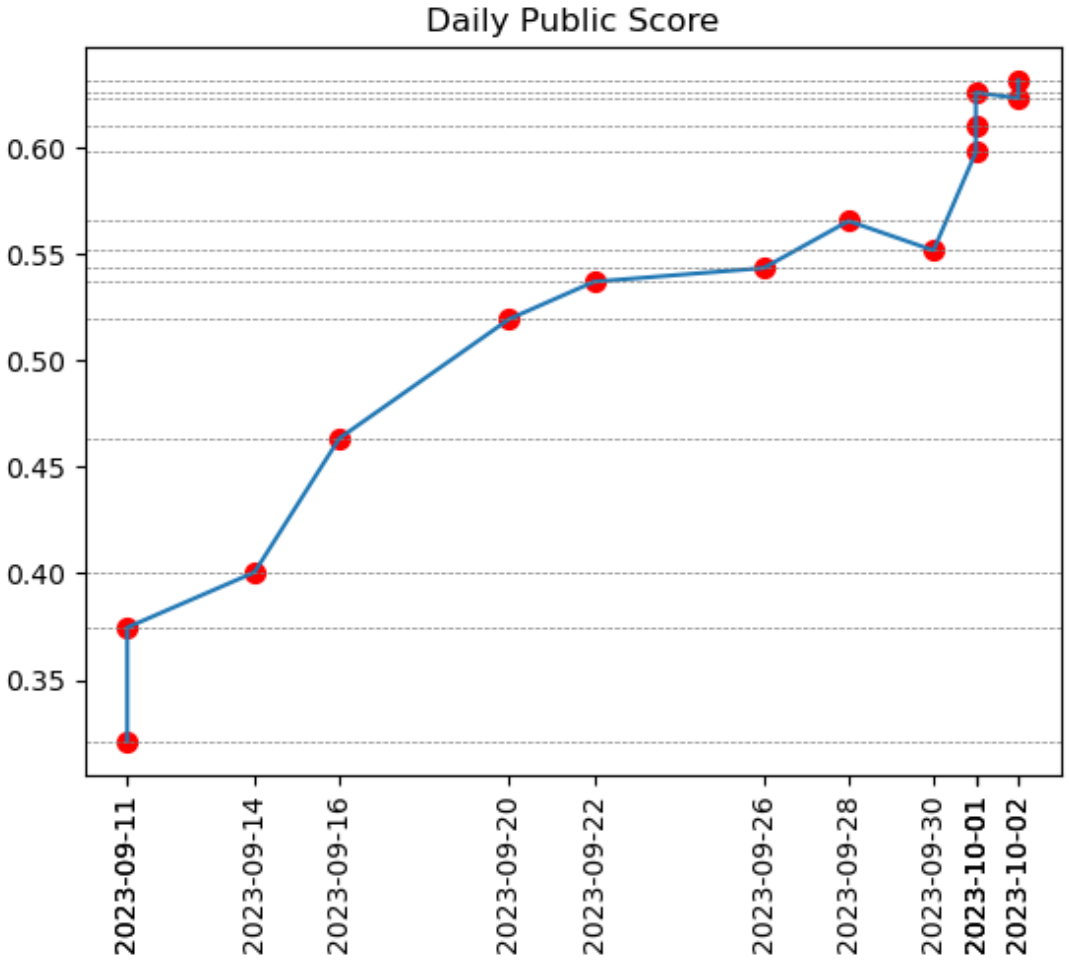
- 대회 참여 목표
  - SOTA 모델들을 참고하며 대회 데이터, 목적에 맞는 구조로 변경하고 효율적인 자원 사용을 목표로 함

평가항목

- 창의성
  - 기존의 Domain Adaptation(DA) 기법들은 왜곡에 대해 다루지 않으며 복잡한 구조를 가졌지만, 간단한 구조를 통해 왜곡에 의한 Domain Gap 문제를 효율적으로 해결할 수 있도록 데이터 증강과 DA 방법론에 대해 다룰 예정

- 적용 가능성
  - 현업에서의 적용에 있어서 가장 큰 문제점은 연산량이라고 생각하기에 본 대회를 진행하며 연산량과 성능 사이의 Trade-off 관계를 생각했고 주로 연산량 관점에서 모델 구조, 추론 방식에 대해 다룰 예정

주요 실험 과정



2023-09-11 : DeeplabV3+ (0.3212)

2023-09-11 : DAFormer(MiT-B4) (0.3746)

2023-09-14 : 데이터 증강 (0.4006)

2023-09-16 : BackGround Mix (0.4635)

2023-09-20 : Validation 데이터 사용 (0.5194)

2023-09-22 : Learning Rate 1e-4 (0.5370)

2023-09-26 : Distortion Image (0.5434)

2023-09-28 : 데이터 사용 수 1/10 → 전체 (0.5655)

2023-09-30 : Crop 사용 (0.5516)

2023-10-01 : Crop,2x Resolution(MiT-B5) (0.5978)

2023-10-01 : 2x Validation(MiT-B5) (0.6098)

2023-10-01 : MiT-B4 (0.6258)

2023-10-02 : MIT-B3 (0.6231)

2023-10-02 : Ensemble (0.6317)

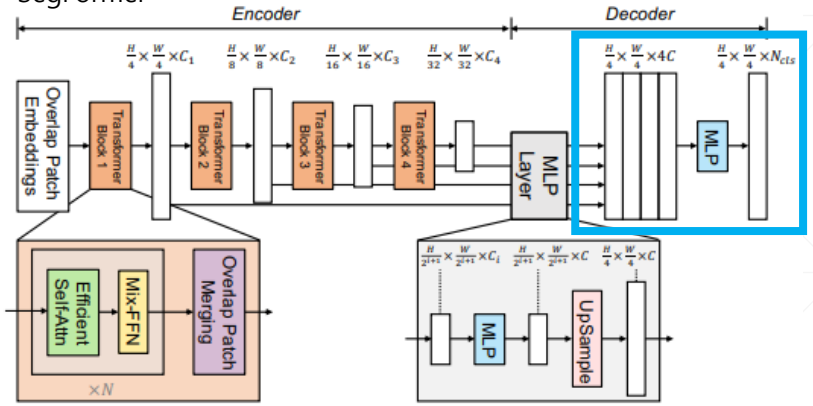
목차

1. Model

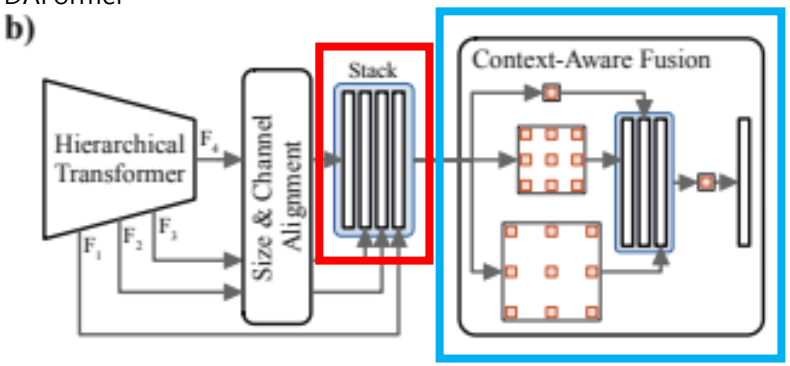
DAFormer

- SegFormer를 기반으로 한 모델
- Stack Layer의 채널 수를 줄여 연산량 감소
- Decoder의 MLP Layer 대신 ASPP Convolution 연산을 적용하여 풍부한 Context 정보를 활용

SegFormer



DAFormer



[2105.15203] SegFormer: Simple and Efficient Design for Semantic Segmentation with Transformers (arxiv.org)

[2111.14887] DAFormer: Improving Network Architectures and Training Strategies for Domain-Adaptive Semantic Segmentation (arxiv.org)

DAFormer 문제점

1. DAFormer의 출력 해상도는 입력 이미지 해상도의 절반이 됨

2. Transformer의 특성상 연산량이 입력이미지 크기의 제곱배로 늘어남
   (주로 Attention 연산에서 발생)

→ 고해상도의 출력을 얻기 위해서는 연산량 증가는 불가피하므로 적절한 조정이 필요함

고해상도 출력이 필요한 이유 : 보간을 통해 채워진 부분이 실제 예측값과 다를 가능성이 높음

Prediction

1	**2**
3	4

Nearest Resize →

Prediction(Resize)

1	**1**	**2**	**2**
1	1	2	2
3	3	4	4
3	3	4	4

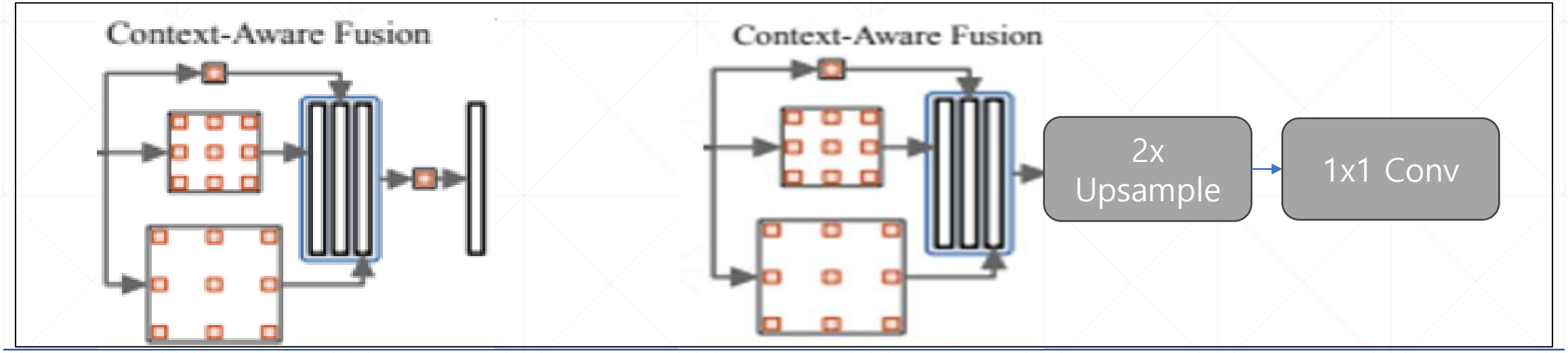
IoU 0.25 —

Ground Truth

1	**2**	**2**	**4**
3	3	4	4
3	4	4	2
1	4	2	2

해결방법

- 1x1 Conv -> 2x Upsample, 1x1 Conv의 간단한 구조 변경
- Upsample로 인해 전체 픽셀 수가 4배가 되었기에 Conv 연산의 연산량 또한 4배가 됨
- 입력 이미지의 크기 증가로 인한 연산량 증가에 비해 무시할 수 있는 연산량 증가



Pretrain Weight - CityScape

- 50에폭 기준 다른 사전학습 모델 TRAIN_TARGET_0000.png 추론 결과
- ImageNet(좌) : 비슷한 학습량인 경우 대부분 잘 예측하지 못함
- ADE(중) : 학습은 잘 되지만 Fence를 잘 구분하지 못함
- CityScape(우) : 중앙 분리대가 아닌 Fence를 잘 구분하는 모습을 보이며 학습도 원활히 잘 됨



NVlabs/SegFormer: Official PyTorch implementation of SegFormer (github.com)

2. Data

데이터 분포 분석

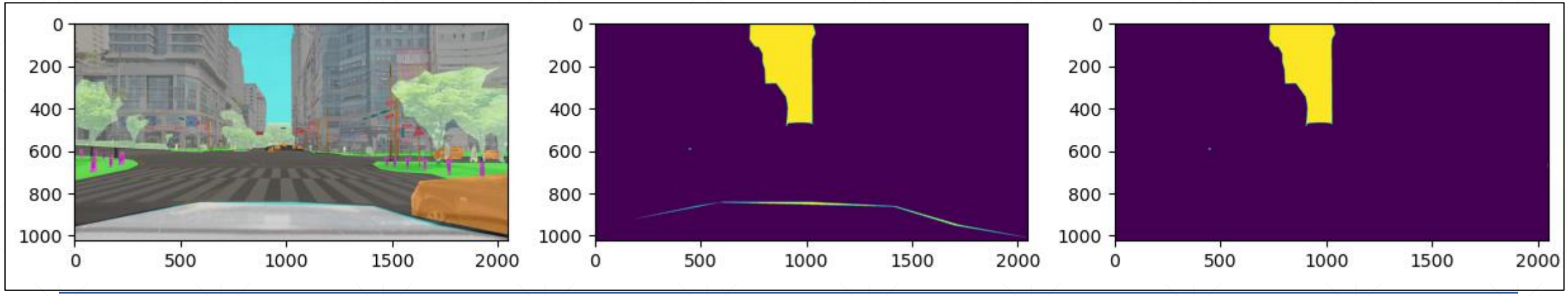
- 데이터의 Mean과 Standard를 보았을 때 Source와 Target 사이의 차이가 큼
- 학습 과정에서 서로 다른 Normalization 적용

Train/Valid Source Mean = [0.58971057 0.59526609 0.57897422]
Train/Valid Source Std = [0.16688787 0.15721062 0.1589595]

Train Target Mean = [0.47146652 0.47141413 0.49733914]
Train Target Std = [0.23908237 0.24033972 0.25281718]

잘못된 라벨 분석

- 대부분의 라벨에 차량 보닛 부분에 하늘 클래스가 존재
- 이미지 높이에서 1/3에 해당하는 아래 영역에 있는 하늘 클래스를 배경 클래스로 변경
- 가운데 사진은 하늘 클래스는 1 아닌 부분은 0으로 이진화 한 것이며 우측 사진은 변경 후 모습



Validation 데이터 사용

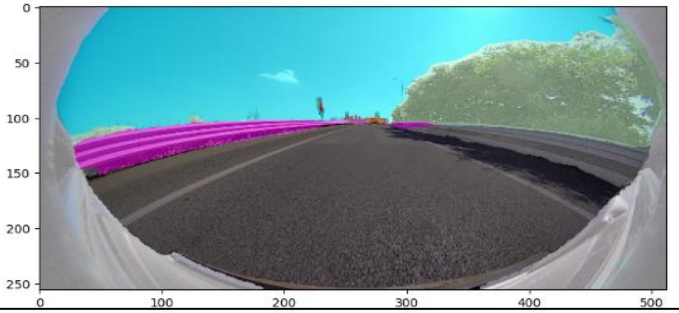
- Target Data를 설명할 수 있는 데이터가 Train Source에 존재하지 않지만 Validation 데이터에는 존재함

VALID_SOURCE_000.png

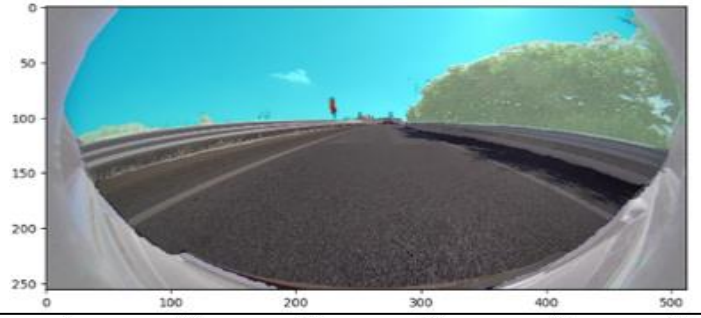
TRAIN_TARGET_0019.png

Input

No Validation



Use Validation



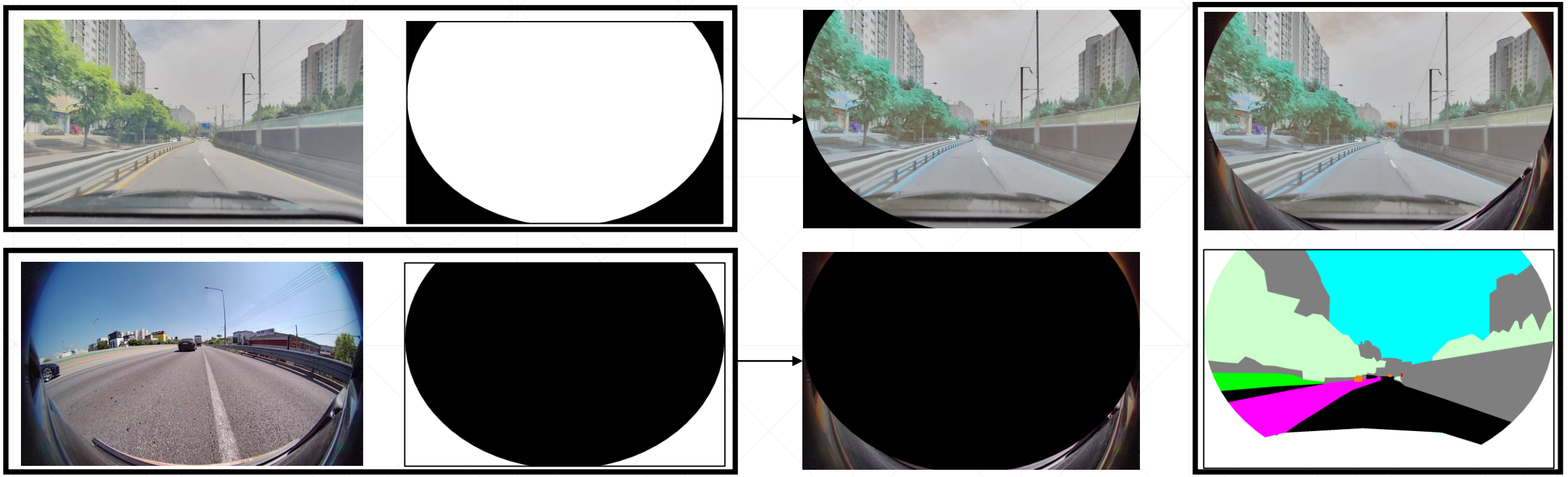
데이터 증강 - Mix BackGround

- Target Data의 가장 자리 부분을 설명할 수 있는 데이터가 Train,Validation 데이터에서 존재하지 않음

- 타원 모양의 이진화와 Bitwise 연산을 통해 배경 부분을 Source Image에 섞는 방식 선택

데이터 증강 - Mix BackGround

배경 혼합 과정



데이터 증강 - Mix BackGround

실제 추론 결과 비교

No Mix

Mix

데이터 증강 - Flip

- Fence 클래스에 해당하는 중앙분리대는 Train과 Validation 데이터에서 주로 좌측에 위치함 (차량 전면 카메라)
- Target Data에서는 그렇지 않은 경우 다수 존재(차량 후면 카메라)
- 이를 반영하기 위해 Flip을 적용하여 중앙 분리대가 반대에 있는 상황도 잘 예측할 수 있도록 함

```
l = []
r = []
lr = []
for filename in tqdm.tqdm(glob.glob('D:/open/*_source_gt/*')):
    filename = filename.replace('\\','/')
    mask = cv2.imread(filename,cv2.IMREAD_GRAYSCALE)
    h,w = mask.shape

    lcount = np.sum(mask[:,:w//2][mask[:,:w//2]==3])
    rcount = np.sum(mask[:,-w//2:][mask[:,-w//2:]==3])

    if lcount+rcount > 0:
        if np.abs(lcount-rcount)/(lcount+rcount) < 0.5:

            lr.append(filename)

        else:
            if (lcount>rcount):
                l.append(filename)
            else:
                r.append(filename)

print(len(l),len(r))
print(len(lr))
```

✓ 35.0s

```
  0%|          | 0/2660 [00:00<?, ?it/s]
C:\Users\fhsdn\AppData\Local\Temp\ipykernel_16356\4101995137.py:13:
  if np.abs(lcount-rcount)/(lcount+rcount) < 0.5:
100%|██████████| 2660/2660 [00:35<00:00, 75.91it/s]
1417 410
78
```

410장의 이미지도 대부분 중앙 분리대가 아닌 콘 형태의 Fence임

데이터 증강 - Jitter

- 현실의 데이터에서는 밝기, 대조 등의 차이 등이 여러 존재하며 이를 반영하기 위함

TRAIN_SOURCE_0012.png

TRAIN_SOURCE_0013.png

데이터 증강 - Elastic, Affine

- Train Target의 왜곡에 대한 정보를 반영할 수 있도록 이미지의 왜곡을 주고자 함

왜곡 전

왜곡 후

최종 Augmentation

```python
self.augmentation = A.Compose([
        A.ColorJitter(p=0.25),
        A.HorizontalFlip(p=0.5),
        A.RandomCrop(width = CROP_WIDTH, height = CROP_HEIGHT)
    ])
```

공통적으로 사용하는 Transform

```python
if self.target_data is not None:
    target_img_path = self.target_data.iloc[idx % self.len_target, 1].replace('./',data_path+'/')
    target_image = cv2.imread(target_img_path)
    target_image = cv2.cvtColor(target_image, cv2.COLOR_BGR2RGB)

    if self.mix_bg_prob > np.random.uniform(0,1):
        distortion_image,distortion_mask = self.mix_bg(distortion_image.astype(np.uint8),distortion_mask.astype(np.uint8),target_image)
```

기본적으로 0.25값 사용

```python
source_tensor = A.Compose([self.augmentation,
                           self.source_norm,
                           ToTensorV2()])(image=image,mask=mask)
```

Source Transform

```python
target_image = A.Compose([self.augmentation,
                          self.target_norm,
                          ToTensorV2()])(image=target_image)['image']
```

Target Transform

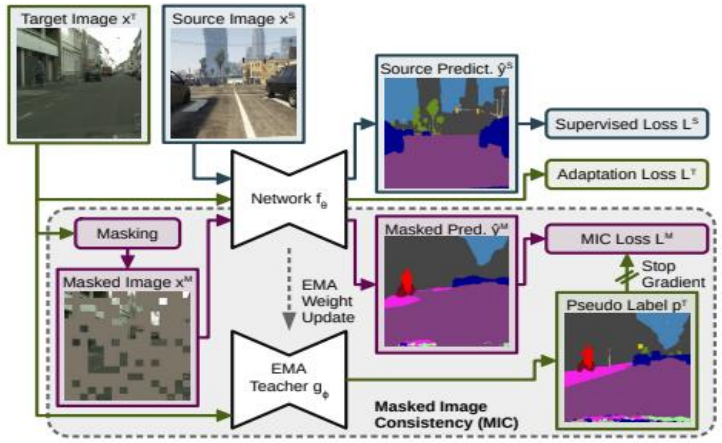
```python
distortion_tensor = A.Compose([self.augmentation,
                               A.ElasticTransform(alpha=100, sigma=10, alpha_affine=25,border_mode = 1,p=1),
                               self.source_norm,
                               ToTensorV2()])(image=distortion_image,mask= distortion_mask)
```

Distortion Transform

3. Domain Adaptation Method

Base Method - MIC

- Masking 된 부분을 예측하게 함으로써 각 패치 사이의 관계를 파악하도록 하는 방식
- DAFormer에서 사용한 UDA 방식을 기반으로 함
- 다른 데이터 셋에서도 대부분 2~3위의 성능을 달성



Unsupervised Domain Adaptation

652 papers with code • 33 benchmarks • 34 datasets

Unsupervised Domain Adaptation is a learning framework to transfer knowledge learned from source domains with a large number of annotated training examples to target domains with unlabeled data only.

Source: Domain-Specific Batch Normalization for Unsupervised Domain Adaptation

Benchmarks

Add a Result

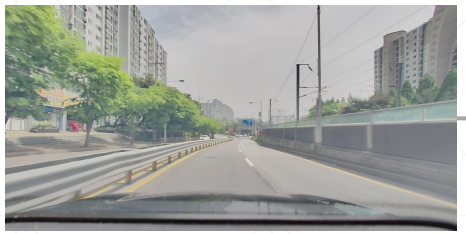
These leaderboards are used to track progress in Unsupervised Domain Adaptation

Trend	Dataset	Best Model	Paper	Code	Compare
	Duke to Market	EvoADA			See all
	Market to Duke	LF2			See all
	GTAV-to-Cityscapes Labels	MIC			See all
	ImageNet-C	EfficientNet-L2+RPL			See all
	SYNTHIA-to-Cityscapes	HRDA + PiPa			See all
	Cityscapes to Foggy Cityscapes	MIC			See all
	Market to MSMT	CCTSE			See all
	Duke to MSMT	CCTSE			See all
	Office-Home	PMTrans			See all

[2212.01322] MIC: Masked Image Consistency for Context-Enhanced Domain Adaptation (arxiv.org)

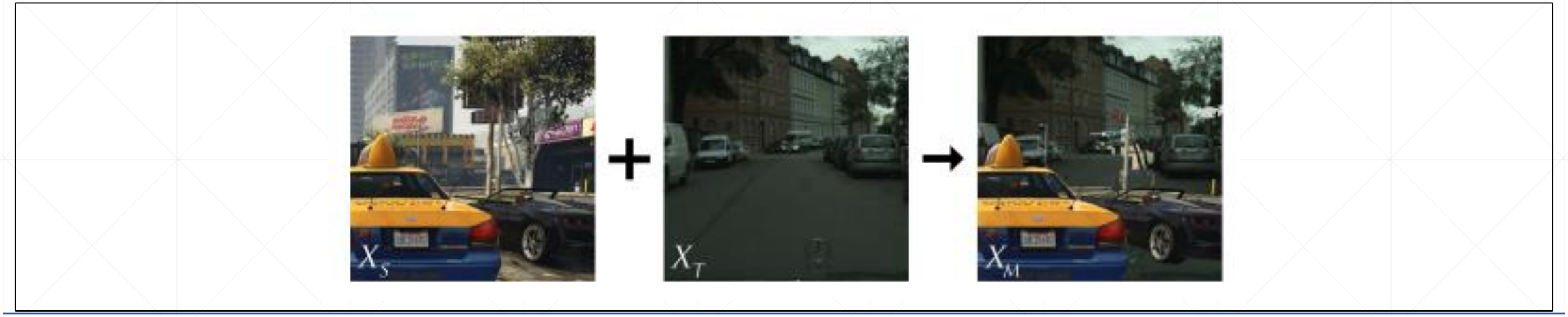
MIC 상세구조

- 매우 복잡한 구조



DACS

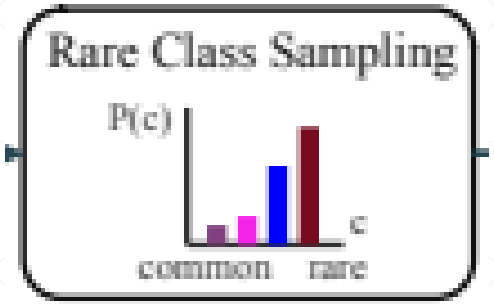
- DACS는 Target Domain에 Source Domain의 일부를 Class별로 섞어 도메인 간의 차이를 줄이고자 하는 목적
- 실제 작동은 하나 학습하는데 많은 에폭이 필요한 것 같아 실제로 사용하지는 못함
- 시간, 자원의 제한으로 추가하지 않음



[2007.08702] DACS: Domain Adaptation via Cross-domain Mixed Sampling (arxiv.org)

Rare Class Sampling

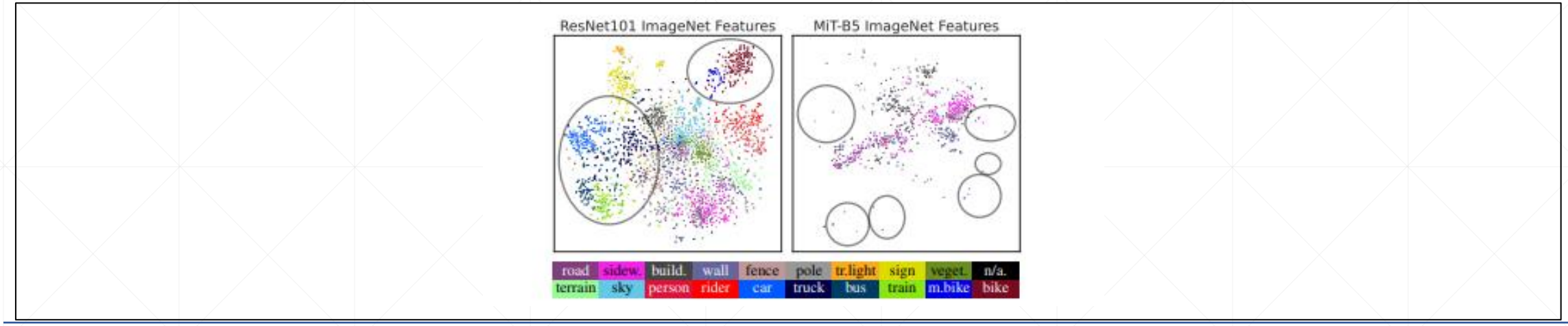
- 자주 나오지 않는 Class(Rider, Person 등)들이 포함된 데이터를 더 자주 뽑아 학습을 하게 함으로써 골고루 학습 시키고자 하는 방식
- 데이터의 양이 충분하지 않아 추출된 데이터가 중복되는 경우가 많이 발생하여 학습이 잘 되지 않는 경우가 발생
- 시간, 자원의 제한으로 추가하지 않음



[2111.14887] DAFormer: Improving Network Architectures and Training Strategies for Domain-Adaptive Semantic Segmentation (arxiv.org)

Feature Distance

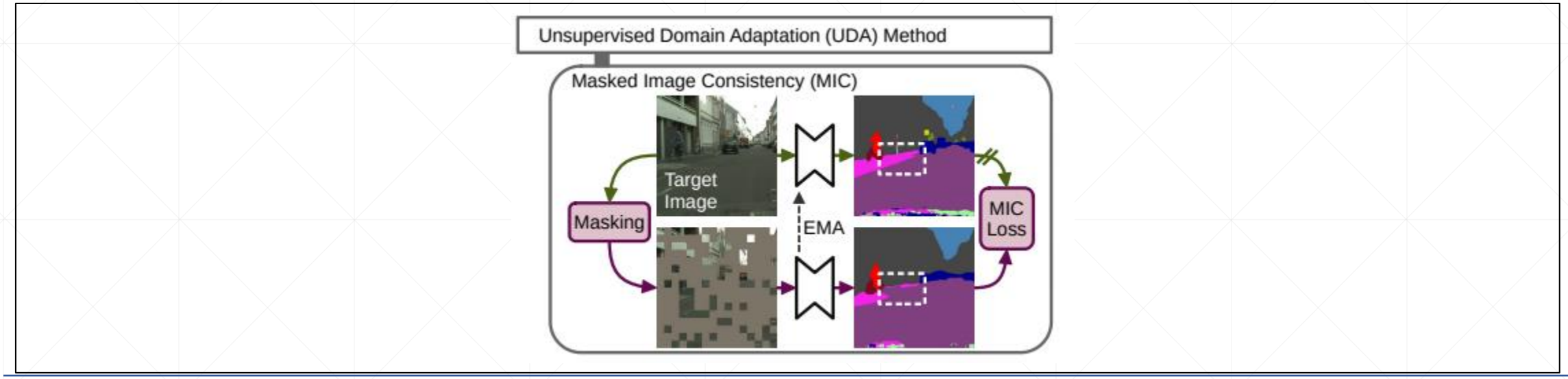
- ImageNet으로 사전 학습된 모델은 객체들에 대한 중요한 정보를 잘 담고 있다는 판단
- 학습을 하며 ImageNet의 정보가 완전히 사라지는 것을 방지하기 위해 Feature간 거리를 최소화 시킴
- City Scape로 FineTuning된 가중치를 사용하기에 불필요하다고 느낌
- 시간, 자원의 제한으로 추가하지 않음



[2111.14887] DAFormer: Improving Network Architectures and Training Strategies for Domain-Adaptive Semantic Segmentation (arxiv.org)

Masking

- Masking 방식을 통해 가려진 부분을 예측하게 함으로써 Class들 사이의 관계를 더 잘 파악할 수 있도록 함
- Masking 함으로써 Domain Adaptaion 속도 또한 크게 상승



[2212.01322] MIC: Masked Image Consistency for Context-Enhanced Domain Adaptation (arxiv.org)

EMA Weight Update

- Teacher Model의 가중치를 업데이트 하는 기법으로 지수이동평균(Exponential Moving Average)을 활용한 기법
- 매 Epoch마다 Teacher Model의 가중치 정보가 남아있어 일관적인 Domain Adaptation이 가능하게 함

$$a = Min\left(Max\ Alpha, 1 - \frac{1}{Epoch}\right)$$

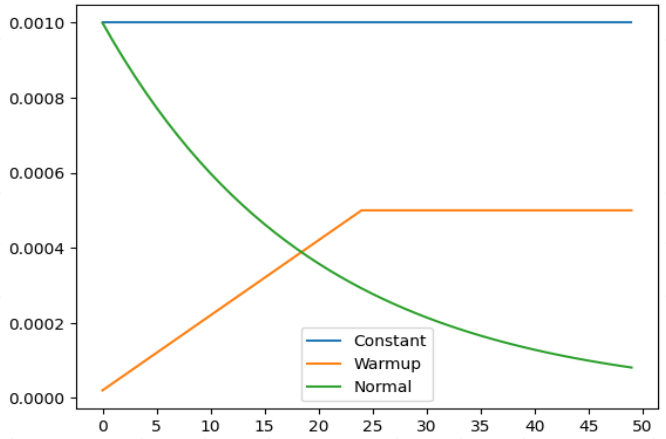
$$W_t^T = a \times W_{t-1}^T + (1 - a) \times W_t^S$$

[2212.01322] MIC: Masked Image Consistency for Context-Enhanced Domain Adaptation (arxiv.org)

Warmup Scheduler

- 특정 Epoch까지 학습률을 선형적으로 증가시키는 방식
- 학습 초기 단계에서 학습의 안정성을 향상시키는 데 사용됨

$$lr_{warmup} = lr_{base} \times \frac{Epoch}{Warmup}$$



[2212.01322] MIC: Masked Image Consistency for Context-Enhanced Domain Adaptation (arxiv.org)

Target Loss

- 모델이 예측한 Target 의 Class 확률 값이 일정값이 넘긴 픽셀의 비율만큼 Loss를 적용하는 방식

$$p_{ij}^{T,cls/seg} = [c = \arg\max_{c'} g_\phi(x^T)_{ijc'}] .$$

$$q^{T,cls} = \max_{c'} g_\phi(x^T)_{c'}$$

$$q^{T,seg} = \frac{\sum_{i=1}^{H}\sum_{j=1}^{W}[\max_{c'} g_\phi(x^T)_{(ijc')} > \tau]}{H \cdot W} .$$

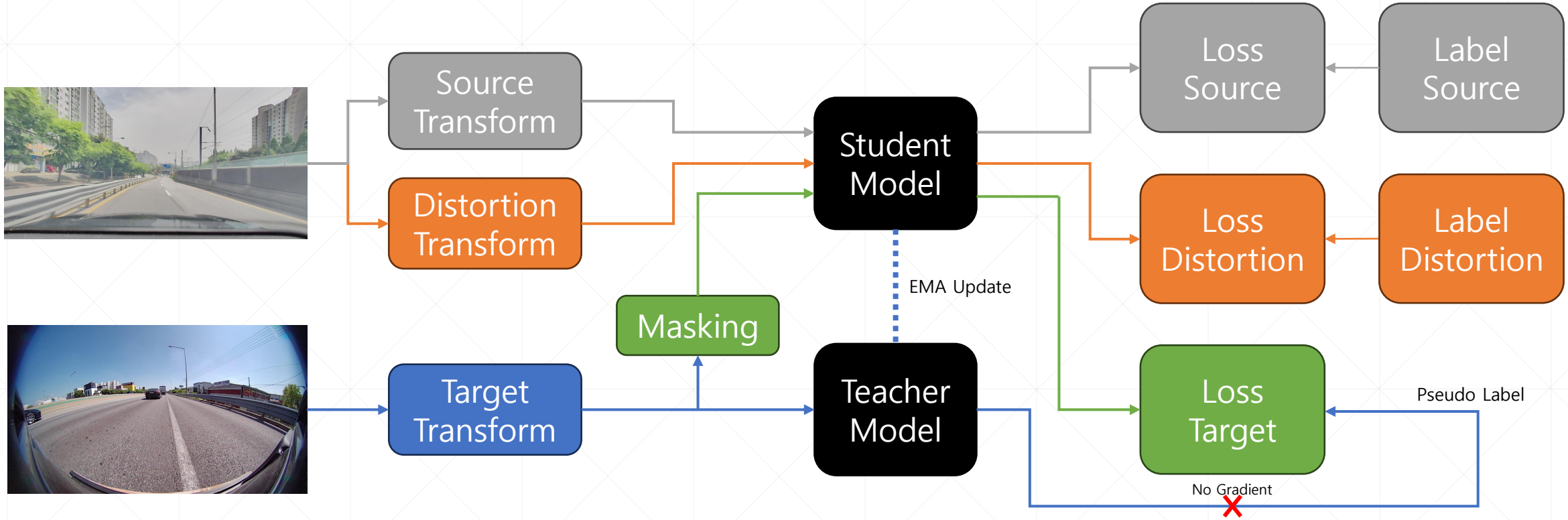
$$\mathcal{L}^M = q^T \mathcal{H}(\hat{y}^M, p^T) ,$$

```python
class target_loss(nn.Module):
    def __init__(self,t=0.968):
        super(target_loss,self).__init__()
        self.t = t
        self.celoss = nn.CrossEntropyLoss(reduction='none')

    def forward(self,pred, pseudo_label):

        b,c,h,w = pseudo_label.shape

        confidence, pt = torch.max(pseudo_label, dim=1) # b,h,w

        qt = (torch.sum(confidence.view(b,-1) > self.t,1) / (h*w)).view(b,1,1) # b,1,1

        loss = torch.mean(self.celoss(pred, pt) * qt) # b,h,w * b,1,1 -> b,h,w -> 1

        return loss
```

[2212.01322] MIC: Masked Image Consistency for Context-Enhanced Domain Adaptation (arxiv.org)

변경된 구조

- 비교적 간단한 구조의 모델(EMA, Warmup, Masking만 유지)
- 계산할 Loss의 수와 Prediction의 수의 감소로 인해 적은 메모리 사용과 기존보다 빠른 학습 가능



Transform 종류

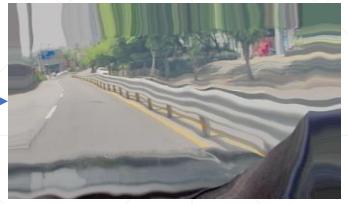
구조 설명

Loss Source	카메라 왜곡을 적용하지 않은 이미지에 대한 Loss이며 왜곡으로 인한 정보 손실을 방지하는 목적의 Loss
Loss Distortion	카메라 왜곡을 적용한 이미지에 대한 Loss이며 Source Domain과 Target Domain 사이의 왜곡에 대한 Gap을 줄이기 위한 Loss
Loss Target	라벨이 없는 Target Domain에 대해서 학습을 진행하기 위한 Loss

Transform 결과

Source Transform

Distortion Transform



Target Transform

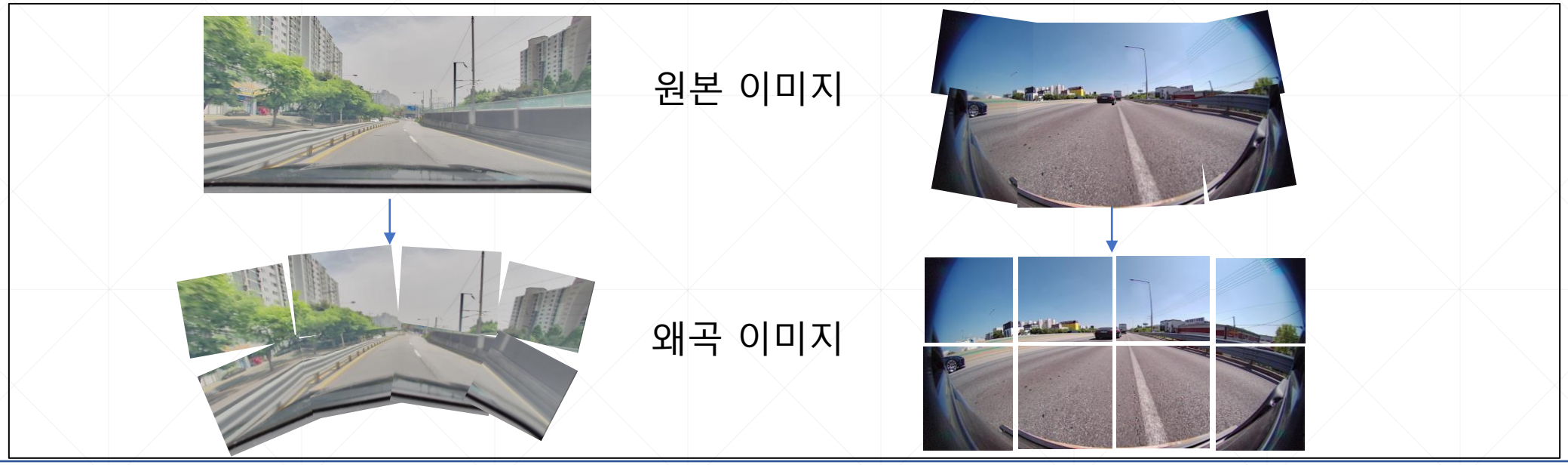
Masking

Crop의 장점 1 - 연산량

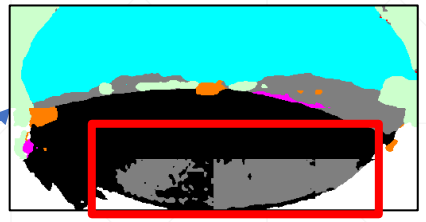
- Transformer의 Attention 연산은 O(n^2)의 시간 복잡도를 가짐
  (n은 패치의 수)
- 이미지 크기가 두배로 늘어나면 연산량은 제곱배가 됨
- Crop을 통해 이미지를 줄이기 때문에 연산량이 감소할 뿐만 아니라 학습 속도도 증가

Crop의 장점 2 - 왜곡 대응

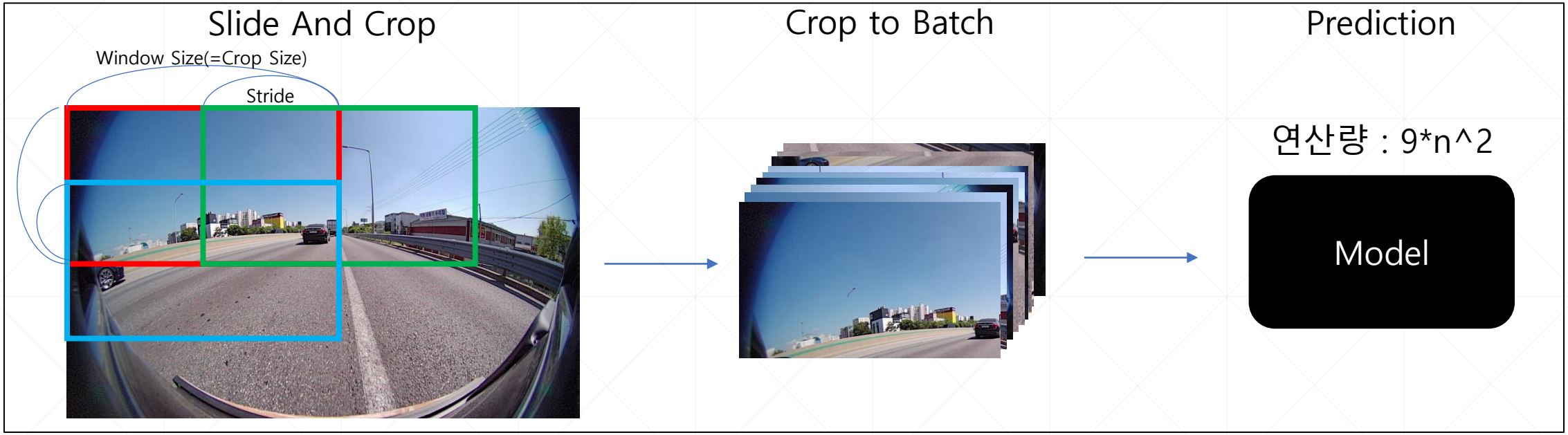
- 원본이미지에 Crop과 Affine을 적용하면 왜곡에 대한 효과를 줄 수 있다.
- 왜곡이미지를 Crop하면 모델에겐 Augmentation 된 이미지로 인식
- Affine으로 Fisheye 뿐만 아니라 파노라마 왜곡에 대해서도 대응이 가능할 것으로 예상됨



Slide Prediction - 전처리

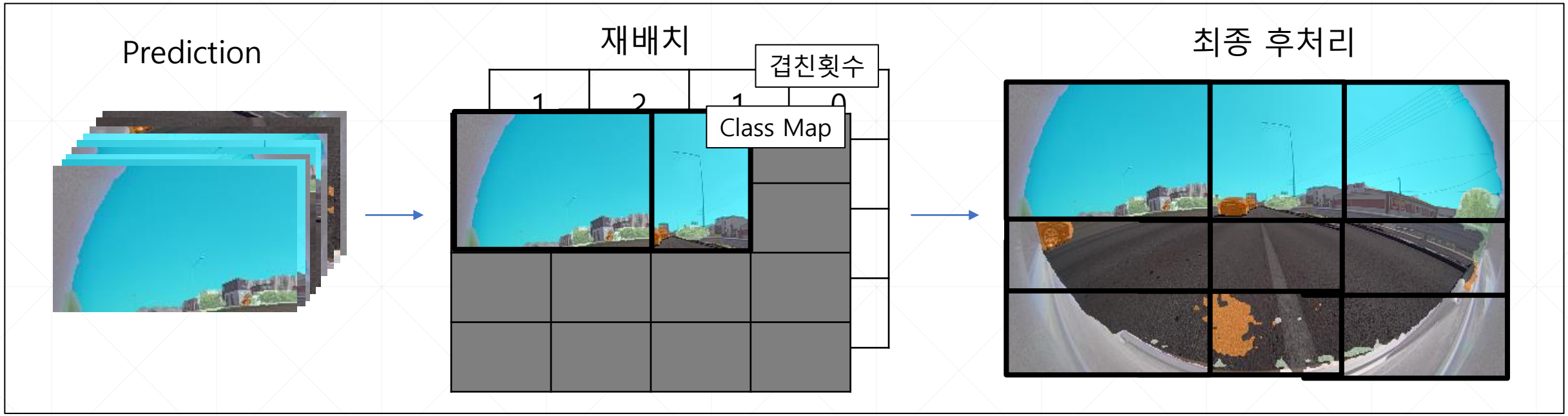


- 추론과정에서 Convolution 연산과 비슷한 방식으로 작동
- Window Size와 동일한 Stride를 사용하면 각이 진 결과를 얻을 수 있음
- 나누어진 Window를 Batch로 만들어 추론하기 때문에 병렬연산의 장점을 극대화 시킬 수 있음



Slide Prediction - 후처리

- $R \in h \times w \times (cls + 1)$의 Map을 0으로 초기화 후 된 실제 위치에 대응되는 부분에 각 클래스에 대한 확률과 겹친횟수 1 더하기
- 최종적으로 각 픽셀에 대해서 클래스의 확률 값을 겹친 횟수를 나누어준 후 Resize나 Argmax 적용하여 최종 Class Map 획득



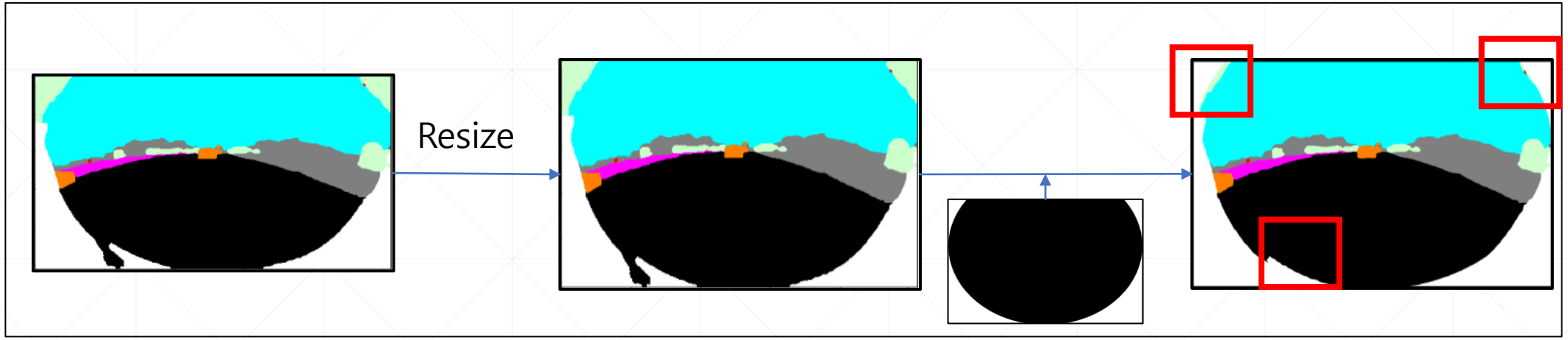
Crop과 Slide Prediction의 장단점

- 장점
  - Crop에 의해 연산량 감소되어 학습/추론 시간 감소
  - Slide Prediction 방식을 통해 전체 이미지 예측 가능
  - Local 지역만을 보기 때문에 Local Context 파악에 용이
  - 추론과정에서의 병렬성 증대

- 단점
  - Global한 Context 파악에 어려움 발생(적절한 Crop Size 필요)

최종 후처리

- 최종 추론 결과를 Resize 후 Mix Background 방식을 활용해 배경을 한번 더 없애는 방식



UDA Hyper Parameter

- Encoder Model : MiT-B3, MiT-B4, MiT-B5
- Image Size(h, w) = (512, 1024)
- Crop Size(h, w) = (256, 512)
  Context 정보를 유지할 수 있도록 이미지 크기의 절반이 적절
- Slide Pred Stride(h, w) = (128, 256)
- Mask Ratio = 0.75
- Mask Size = 32
- Max EMA Alpha = 0.9
- Warmup Epoch = 50
- Learning rate(Encoder, Decoder) = (1e-4, 1e-3)

성능 실험

Patch Size \ Ratio	0.5	0.75
16	X	0.4769513254
32	0.4745660445	0.4778109493
64	X	0.4651144239

Max EMA Alpha	0.9	0.99
Public Score	0.5729644644	0.5599661686

Warmup	25	50	75
Public Score	X	0.5415989714	0.5302718139

Encoder	MiT-B3	MiT-B4	MiT-B5
Public Score	0.6231142597	0.6258260075	0.6098966898

Crop 성능 실험

- Encoder는 B4로 실험 진행
- Crop Size가 너무 작으면 Context 정보를 너무 많이 잃게 되어 성능이 급감하는 모습을 보임

Image Size	Crop Size	Relative Time Complexity (Attention)	GPU Memory Usage (8 Batch)	GPU	Time per Epoch	Public Score
(256, 512)	X	n^2	32GB	Tesla A100	8 Minute	0.565478254
(256, 512)	(128, 128)	(n/8)^2	4GB	Tesla T4	5 Minute	0.490008843
(256, 512)	(128, 256)	(n/4)^2	8GB	Tesla T4	9 Minute	0.551691867
(512, 1024)	(256, 512)	n^2	32GB	Tesla A100	8 Minute	0.625826007

Stride 성능 실험

- Encoder는 B4로 실험 진행
- Stride가 너무 작아지면 Window의 수가 증가하므로 연산량 크게 증가

Image Size	Crop Size	Stride	Relative Time Complexity (Attention)	Public Score
(256, 512)	X	X	n^2	0.565478254
	(128, 256)	(128, 256) [x1]	(n/4)^2	0.5420129
		(64, 128) [x0.5]	9*(n/4)^2	0.551691867
		(32, 64) [x0.25]	25*(n/4)^2	0.5537311567

Ensemble

- 동일 조건으로 MiT-B3, MiT-B4, MiT-B5의 각 클래스에 대한 확률을 (0.9, 1.0, 0.8)로 가중합하는 방식을 사용

Ensemble 모델 준비 과정

```
torch.cuda.empty_cache()
model_list = [daformer(mit_b3(),daformerhead()),
              daformer(mit_b4(),daformerhead()),
              daformer(mit_b5(),daformerhead())]

state_dict = [torch.load('/content/drive/MyDrive/samsung_seg(512,1024)/model/DAformer(B3,crop2x,32seed,valid2x,1e-4)/self_training/Epoch(065).pth','cuda'),
              torch.load('/content/drive/MyDrive/samsung_seg(512,1024)/model/DAformer(B4,crop2x,32seed,valid2x,1e-4)/self_training/Epoch(065).pth','cuda'),
              torch.load('/content/drive/MyDrive/samsung_seg(512,1024)/model/DAformer(B5,crop2x,32seed,valid2x,1e-4)/self_training/Epoch(060).pth','cuda')]

weight_list = torch.tensor([0.9 ,1.0 ,0.8]).reshape(len(model_list),1,1,1,1).to(device)

for i in range(len(model_list)):
    model_list[i].load_state_dict(state_dict[i]['model'])
    model_list[i].to(device)
    model_list[i].eval()
```

Ensemble 과정

```
        input = imgs.float().to(device)
        preds_list = []
        for model in model_list:
            preds_list.append(slide_pred(model,input,stride=(CROP_HEIGHT//2,CROP_WIDTH//2),softmax = True ,padding=False))

        preds_list = torch.stack(preds_list) # len(model), b,c,h,w
        ensemble = torch.sum(preds_list*weight_list,0).unsqueeze(0) # 1,b,c,h,w
```

4. Conclusion

결론

- 모델의 Encoder를 MiT-B3를 사용한다고 하여도 성능 차이는 크게 발생하지는 않음
- Crop과 Slide Prediction 방식으로 더 적은 연산량으로 전체 이미지를 한번에 예측하는 것과 비슷한 성능을 낼 수 있음
- Distortion Loss를 통해 모델에게 왜곡에 대한 정보를 제공하여 왜곡으로 인한 Domain Gap을 더 쉽게 줄일 수 있음

느낀 점 및 아쉬운 점

- Domain Adaptation을 처음 접해보았지만 대회 중 매우 흥미로운 주제라고 생각했다.
- 시간과 자원의 제한이 커서 섬세한 실험을 하지는 못하였지만 기회가 된다면 여러 시도를 해보고 싶다.
- Train Dataset과 Validation Dataset 모두 학습에 사용하기도 하였고 Fisheye Distortion에 대한 검증 데이터가 없어 육안으로 확인하는 방식을 사용한 점이 매우 아쉬웠다.