2-TEAM

Motorcycle Night Ride

- 발표일 : 2026. 02. 13.
- 발표자 : 최사랑, 고대홍, 김하늘
- 구성원 :
  고대홍, 김하늘, 박정선, 임태민, 최사랑

CONTENTS

Part 1

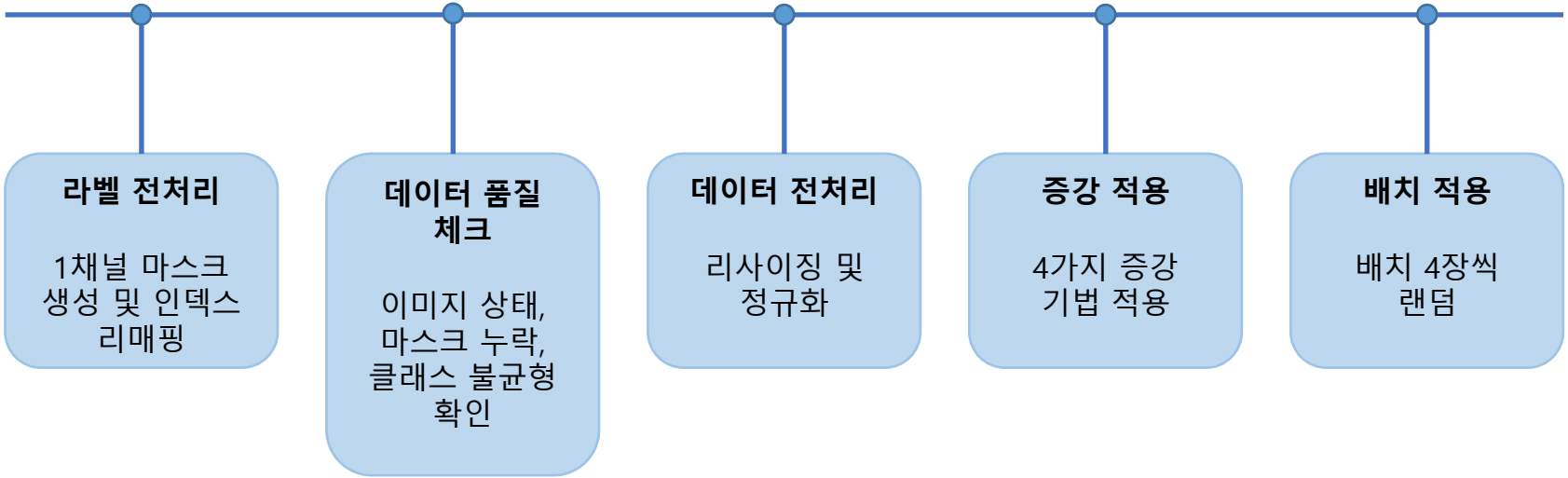
기획 동기: 주행 환경 분석의 실제 활용 방법 탐색

도로 이미지를 통해 주행 환경을 파악할 수 있다는 점에서 출발해, 이를 안전과 연결해보고자 했다.
Segmentation으로 도로 상황을 구조적으로 분석하고, 이미지별로 위험도를 수치화한 안전 점수 모델을 설계하였다.

COCO JSON을 선택한 이유

PNG 마스크는 이미 픽셀 값이 정해진 완성된 결과물이라 구조를 바꾸거나 확장 분석하기 어렵다.
반면 COCO JSON은 객체 정보가 담긴 설계도이기 때문에,
클래스를 합치거나 나누거나, 같은 클래스 안의 객체를 개별로 분석하는 등 더 다양한 실험과 분석을 할 수 있다.

진행 과정


라벨 전처리
1채널 마스크 생성 및 인덱스 리매핑
데이터 품질 체크
이미지 상태, 마스크 누락, 클래스 불균형 확인
데이터 전처리
리사이징 및 정규화
증강 적용
4가지 증강 기법 적용
배치 적용
배치 4장씩 랜덤

라벨 전처리

RGB 마스크를 1채널 마스크로 바꿔주기

RGB 상태에서 resize하면 색이 섞여서 모델이 제대로 판단을 못 한다.
그래서 1채널 마스크로 바꾸는 작업을 했다.

```
h, w = img_info['height'], img_info['width']
mask = np.zeros((h, w), dtype=np.uint8)
```

```
binary_mask = coco_api.annToMask(ann)
```

인덱스 리매핑

Background 포함해서 총 7개 클래스 만듦. (인덱스 0~6까지)

COCO annotation에 없는 픽셀을 처리하기 위해 Background(0번)를 추가한 것.

세그멘테이션에서는 일반적으로 Background를 포함시켜서 클래스 수를 잡는다.

라벨 전처리

인덱스 리매핑 확인 코드

```
# "일단 모든 픽셀을 0번으로 채워놓는다"
mask = np.zeros((h, w), dtype=np.uint8)

...

# 이 코드를 통해서 최종 인덱스가 0 ~ 6이 된 것
cat_remap = {0: 0}    # 0은 0이다 (Background는 그대로 둔다)
# 원래 COCO의 category_id를 1부터 차례대로 다시 번호 매긴다
for new_id, orig_id in enumerate(sorted_cat_ids, start=1):
    cat_remap[orig_id] = new_id
```

데이터 품질 체크

깨진 이미지 체크

이미지 파일이 정상적으로 열리고 픽셀로 변환되는지 확인 필수.

PNG/JPG 손상 시 학습 중 에러 발생.

```
for f in original_files:
    img_path = IMAGE_DIR / f
    try:
        with Image.open(img_path) as im:
            im.verify()
    except Exception:
        broken_images.append(f)
```

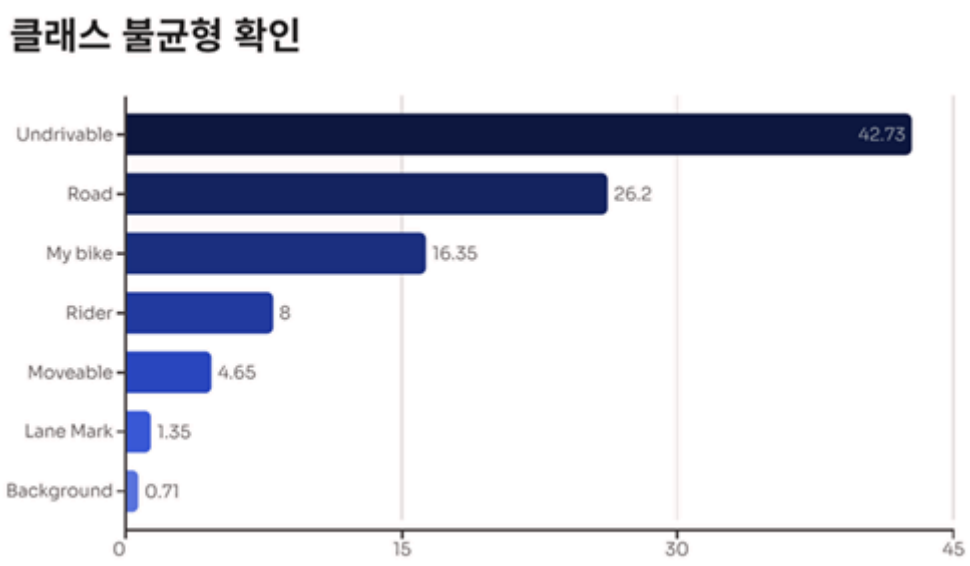
마스크 누락 체크

이미지마다 대응되는 마스크 파일(___save.png)가 있는지 확인.

학습에서 (입력, 정답) 쌍이 깨지면 학습이 바로 실패하거나 왜곡됨

```
mask_set = set(mask_files_png)
for f in original_files:
    expected_mask = f + "___save.png"
    if expected_mask not in mask_set:
        missing_masks.append(f)

bad_files = set(broken_images) |
set(missing_masks)
```

데이터 품질 체크



- Segmentation에서 불균형을 확인할 때는, "픽셀 개수 기준"으로 불균형을 봐야 한다.
- 소수 클래스는 학습 중 무시되기 쉽기 때문에, 불균형을 미리 파악하고 소수 클래스를 위한 적절한 증강 기법을 고려한다.
- 차선(Lane Mark)이 1.35%로 너무 적어서 "조금만 흐려져도 모델이 포기"하기 쉬움.
  → 블러/감마를 적당히 줌으로써 안 보이는/흐릿한 차선을 학습시키기.

클래스 불균형 확인 코드


```
class_pixel_counts = Counter()   # 각 클래스(0, 1, 2...)별 픽셀 개수를 저장할 카운터를 초기화
total_pixels = 0   # 전체 픽셀 수를 저장할 변수
for img_id in img_ids:   # 모든 이미지에 대해 다음을 반복
    mask = create_class_mask(coco, img_id, cat_remap)   # 마스크 만들기
    unique, counts = np.unique(mask, return_counts=True)   # 각 클래스 개수 세기
    for u, c in zip(unique, counts):
        class_pixel_counts[u] += c
    total_pixels += mask.size

# 픽셀별 비율을 재기
class_ratios = {class_names[k]: v / total_pixels for k, v in class_pixel_counts.items() if k <
num_classes}
```

데이터 전처리

512 x 512 리사이징

CNN 백본에 최적화된 크기. 정보 손실과 연산량의 균형을 맞춘 중간 크기다.

train셋, val셋 둘 다 적용

```
IMG_SIZE = 512
```

```
train_transform = A.Compose([
    A.Resize(IMG_SIZE, IMG_SIZE),
```

```
val_transform = A.Compose([
    A.Resize(IMG_SIZE, IMG_SIZE),
```

정규화 ImageNet mean/std 사용

DeepLabV3 모델 구조 + ResNet50 백본을 사용할 예정

해당 백본이 이미지 학습할 때, ImageNet mean/std로 정규화 진행. 같은 방식으로 정규화 해주면 익숙한 형태로 받기 때문에 학습이 수월하다.

```
A.Normalize(
    mean=[0.485, 0.456, 0.406],
    std=[0.229, 0.224, 0.225]
)
```

데이터 증강

HorizontalFlip (좌우반전)

이미지+마스크에 적용.

"자동차는 왼쪽에 있어도 자동차고 오른쪽에 있어도 자동차구나"를 배운다.

```
train_transform = A.Compose([
    ...
    A.HorizontalFlip(p=0.5),
```

RandomBrightnessContrast (밝기/대비)

이미지에만 적용

갑자기 화면이 어두워지거나 밝아지는 상황을 만들어서 밝기와는 상관없이 객체를 인식하도록 훈련시킴

```
    A.RandomBrightnessContrast(p=0.3),
```

RandomGamma (감마)

이미지에만 적용

어두운 부분이 더 어둡게, 하이라이트 부분이 더 강조됨

"빛이 강렬해도, 일정 부분이 더 어두워도 오토바이는 오토바이다."라고 구분할 수 있도록 학습

```
A.RandomGamma(gamma_limit=(80, 120), p=0.3),
```

GaussianBlur (빛 번짐)

이미지에만 적용

경계선 흐림 효과

경계가 또렷할 때만 인식하는 게 아니라, 조금 흐릿흐릿해도 클래스를 잘 판단하도록 훈련

```
# 중간 정도의 세기, 적용 빈도 20%
A.GaussianBlur(blur_limit=(3, 7), p=0.2),
```

배치 적용

- batch=4 , 랜덤하게 돌림.
- DeepLabV3-ResNet50은 계산량이 많고, GPU 메모리를 많이 먹는 모델이라 batch size를 크게 잡으면 메모리 터질 확률이 높다.
- 그래서 너무 작게도, 많이도 잡지 않은 batch=4로 설정.

```
BATCH_SIZE = 4
```

```
train_loader = DataLoader(train_dataset, batch_size=BATCH_SIZE, shuffle=True, num_workers=0)
val_loader = DataLoader(val_dataset, batch_size=BATCH_SIZE, shuffle=False, num_workers=0)
```

Part 2

테스트 모델

도로 주행의 안전성을 파악하는데 필요한 기준을 세 가지 관점에서 고려하여 모델 선정
- 정밀도, 효율성 , 실시간성

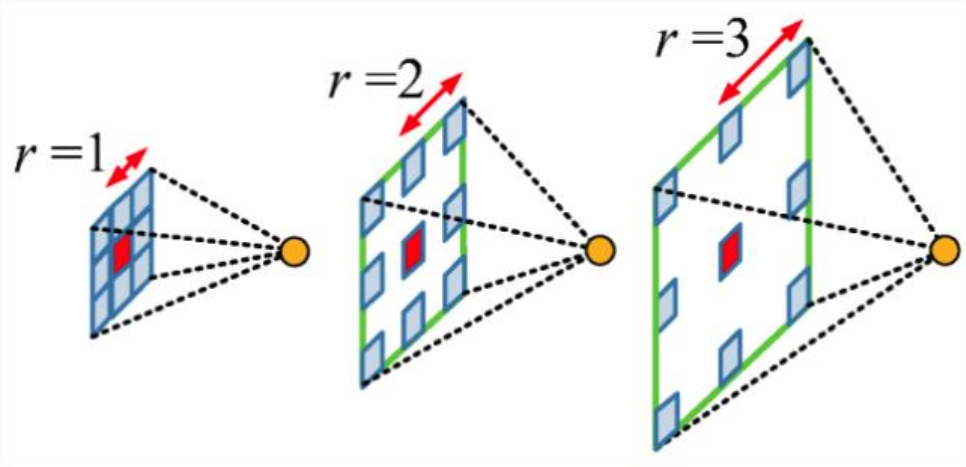
DeepLabV3+	SegFormer	BiSeNet V2
인코더- 디코더 구조를 활용한 정밀성 추구	**트랜스포머 구조. 이미지 전체를 빠른게 파악하는 효율성**	**양방향 구조를 통해 속도와 성능의 밸런스. 실시간성 고려**

Model 1 : DeepLabV3+

다양한 시야를 확보하면서 픽셀 단위로 객체에 대한 분리를 테스트함.

- ASPP (Atrous Spatial Pyramid Pooling) : 서로 다른 dilation rate를 가진 Atrous Convolution을 동시에 여러 개 적용해서 작은 차선부터 큰 건물까지 크기가 다양한 객체를 놓치지 않도록 넓고 다양한 수용 영역을 확보

- ResNet-50을 백본으로 하여 이미지에서 특징을 추출할 때 전이학습을 적용함



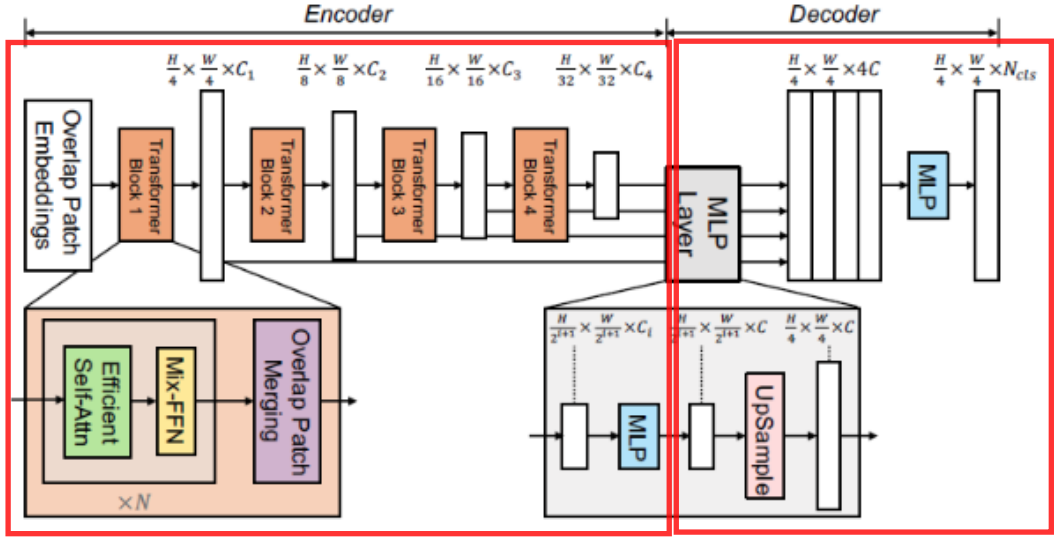
Atrous convolution 예시. 출처

참고문헌

Model 2 : SegFormer

계층적인 Transformer 인코더와 lightweight MLP 디코더를 활용
(* MLP : Multi-Layer Perceptron)

인코더에서 Transformer 구조를 가져, 큰 수용 필드를 가지고, 계층적인 구조로 속도와 표현력이 이점
디코더에서는 별다른 레이어를 추가하지 않고 가볍게 구성



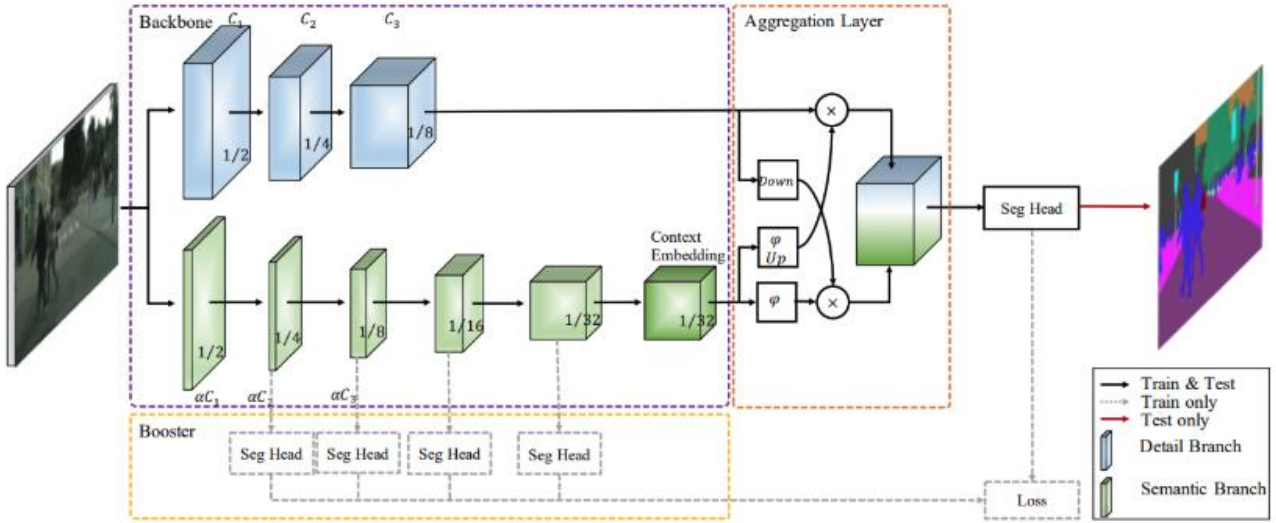
참고문헌

Model 3 : BiSeNet V2

'정확도'와 '속도'라는 상충하는 두 목표를 달성하기 위해 두 개의 서로 다른 경로를 동시에 사용

Detail Branch : 채널 양은 많게 하면서 Layer의 수를 줄인다.낮은 층의 레이어에서 고해상도 정보를 유지

Semantic Branch : 적은 채널 양을 가지며 layer를 깊게 쌓는다. 채널 수를 줄이면서도 넓은 수용 영역을 확보



BiSeNet V2의 전체 구조

참고문헌

평가 지표

세부 객체 인식의 정밀도 측면에서 mIoU, 직관적인 정확도 측정을 위해 Pixel Accuracy를 평가 지표로 삼음

mIoU (Mean Intersection over Union)	Pixel Accuracy
객체의 경계를 얼마나 정확하게 예측했는지 보여주는 지표	이미지의 전체 픽셀 중 정답을 맞힌 비율

평가지표 1 : mIoU

IoU는 합집합에 대한 교집합의 비를 뜻하며, 1에 가까울수록 성능이 좋음
MIoU는 여러 개의 IoU 값들에 대한 평균이다.

→ Lanemark 객체는 화면에서 차지하는 비중이 작아 Pixel Accuracy만으로는 성능 파악이 어려움. mIoU는 이러한 작은 객체들의 인식 정확도를 평가하는데 도움이 됨.

$$\text{IoU} = \frac{Area\ of\ Overlap}{Area\ of\ Union}$$

IoU

참고문헌

평가지표 2 : Pixel Accuracy

전체 이미지 픽셀 중 모델이 정답을 맞힌 비율

데이터셋의 6개 클래스 중 'Road'나 'Undrivable'처럼 큰 면적을 차지하는 클래스를 모델이 얼마나 안정적으로 분류하는지 보여주는 기초적인 척도

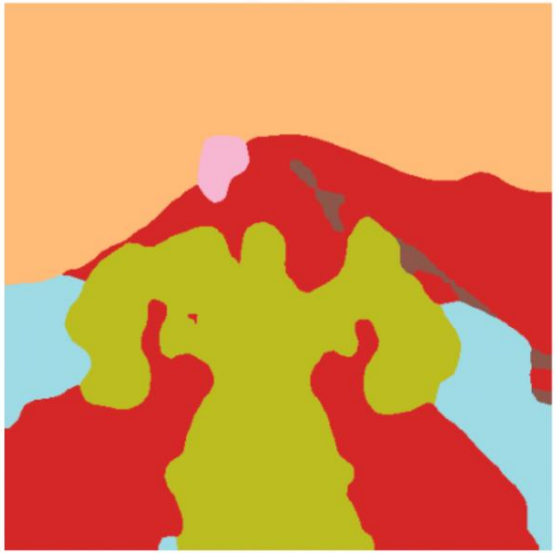
$$PixelAccuracy = \frac{TP+TN}{TP+TN+FP+FN}$$

Part 3

CAM – 1st Image

Original (H,W)

Pred mask (512x512)



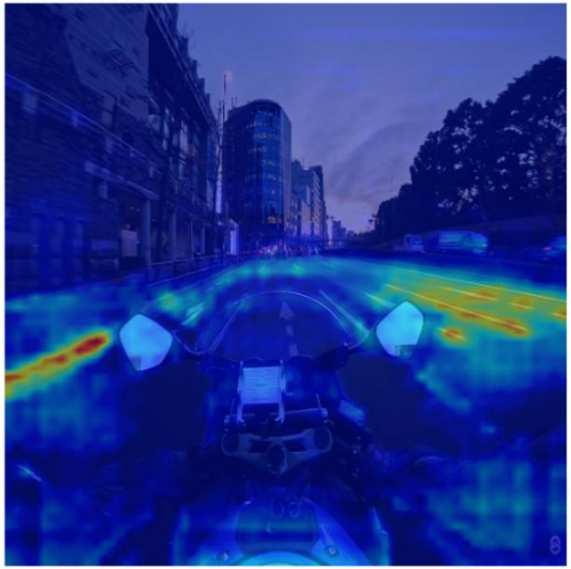
Grad-CAM on 'Lane Mark' (512x512)

CAM – 2nd Image

Original (H,W)

Pred mask (512x512)

Grad-CAM on 'Lane Mark' (512x512)

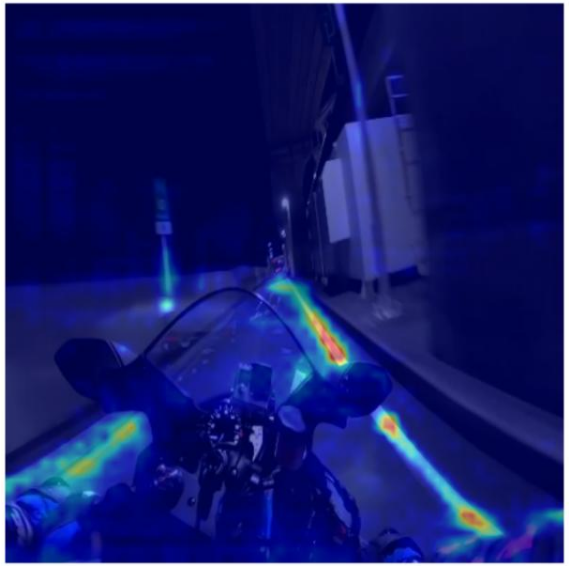


CAM – 3rd Image

Original (H,W)

Pred mask (512x512)

Grad-CAM on 'Lane Mark' (512x512)



CAM – 3rd Image

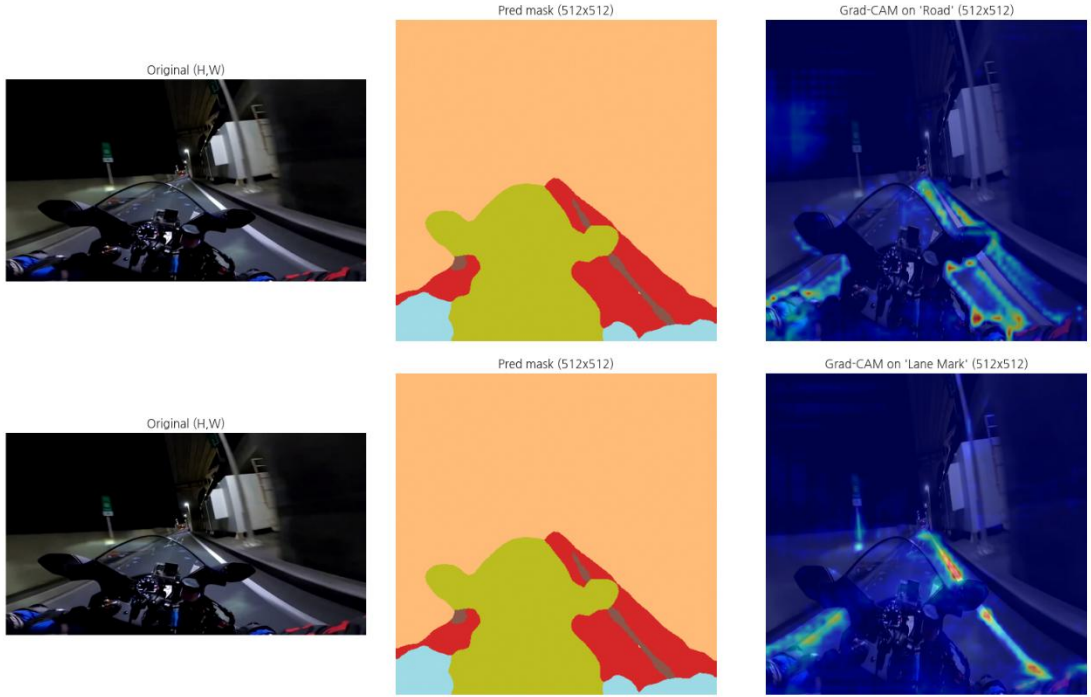
In [27]:

```
# ============================================================
# 3-7. CAM 실행 함수 + 예시 (Lane Mark 등)
# ============================================================

import numpy as np
import os
name_to_idx = {name: i for i, name in enumerate(class_names)}
from PIL import Image
import matplotlib.pyplot as plt
from pytorch_grad_cam import GradCAM
from pytorch_grad_cam.utils.image import show_cam_on_image

def run_segmentation_cam(
    img_np_rgb: np.ndarray,
    target_class_name: str = "Lane Mark",
):
    """
    img_np_rgb: (H,W,3) RGB uint8
    target_class_name: class_names에 존재하는 이름 (예: 'Lane Mark')
    """
    assert target_class_name in class_names, f"{target_class_name} not in class_names"

    # 1) 입력을 모델 입력 크기(512x512)로 변환
    t = val_transform(image=img_np_rgb)
    img_tensor = t["image"].float().to(device)  # (3,512,512)

    # 2) 예측 마스크 생성 (CAM 타깃 마스크로 사용)
    with torch.no_grad():
        logits = cam_model(img_tensor.unsqueeze(0))  # (1,C,512,512)
        pred = logits.argmax(1).squeeze(0).detach().cpu().numpy()  # (512,512)

    class_idx = name_to_idx[target_class_name]
    target_mask = (pred == class_idx).astype(np.float32)  # (512,512) 0/1

    # 타깃 클래스가 아예 없으면 CAM이 의미 없어서 종료
    if target_mask.sum() == 0:
        print(f"⚠️ pred에 '{target_class_name}' 픽셀이 0이라 CAM 스킵")
        return
    # 3) GradCAM
    cam = GradCAM(model=cam_model, target_layers=target_layers)

    targets = [SemanticSegmentationTarget(class_idx, target_mask)]
    grayscale_cam = cam(
        input_tensor=img_tensor.unsqueeze(0),
        targets=targets
    )[0]  # (512,512) 0~1

    # 4) 시각화: 0~1 범위 RGB로 맞춘 다음 overlay
    img_512 = img_tensor.detach().cpu().permute(1,2,0).numpy()
    img_vis = (img_512 - img_512.min()) / (img_512.max() - img_512.min() + 1e-10)

    cam_overlay = show_cam_on_image(img_vis, grayscale_cam, use_rgb=True)

    # 5) 출력
    plt.figure(figsize=(16,5))
    plt.subplot(1,3,1); plt.imshow(img_np_rgb); plt.title("Original (H,W)"); plt.axis("off")
    plt.subplot(1,3,2); plt.imshow(pred, cmap="tab20", vmin=0, vmax=num_classes-1); plt.title("Pred mask (51
    plt.subplot(1,3,3); plt.imshow(cam_overlay); plt.title(f"Grad-CAM on '{target_class_name}' (512x512)");
    plt.tight_layout()
    plt.show()

# ========================
# 예시: val에서 3장 뽑아서 Lane Mark CAM 보기
# ========================
np.random.seed(42)
picked_ids = []
for _ in range(min(80, len(val_ids))):
    img_id = int(np.random.choice(list(val_ids), 1)[0])
    info_list = coco.loadImgs([img_id])
    if not info_list:
        continue
    info = info_list[0]
    img_path = str(IMAGE_DIR) + "/" + info["file_name"]
    if os.path.exists(img_path):
        picked_ids.append(img_id)
    if len(picked_ids) >= 3:
        break

print("picked_ids:", picked_ids)

for img_id in picked_ids:
    info = coco.loadImgs([img_id])[0]
    img_path = str(IMAGE_DIR) + "/" + info["file_name"]
    img_rgb = np.array(Image.open(img_path).convert("RGB"))
    print(f"\n=== {info['file_name']} ===")
    run_segmentation_cam(img_rgb, target_class_name="Lane Mark")
```

CAM – 3가지 Image에 대한 요약

세 가지 결과를 종합하면, 모델은 전반적으로 차선 구조를 중심으로 판단하지만, 상황에 따라 도로 중앙이나 전경 영역에도 반응이 나타났습니다.

CAM – Road vs Lane Mark 비교



In [24]:
```
run_segmentation_cam(img_rgb, target_class_name="Road")
run_segmentation_cam(img_rgb, target_class_name="Lane Mark")
```

CAM – 전체 결과 해석

- Road와 Lane Mark CAM을 비교했을 때, 두 클래스의 heatmap 패턴이 명확히 구분되는 경향을 보였습니다.

- Road는 도로 면적과 경계 중심으로 넓게 반응했고, Lane Mark는 차선 라인을 따라 더 집중적인 반응을 나타냈습니다.

- 이는 모델이 단순히 밝은 영역을 기준으로 판단하는 것이 아니라, 클래스별 특징적인 구조를 학습했음을 의미합니다.

- 다만 일부 프레임에서는 오토바이 전경이나 도로 중앙에도 반응이 나타났는데, 이는 고정된 라이더 시점 데이터에서 발생할 수 있는 편향으로 해석됩니다.

➢ 결론적으로 CAM 분석을 통해 모델이 실제 위험 요소인 차선을 기준으로 판단하고 있는지 검증할 수 있었으며, 동시에 개선이 필요한 bias 가능성도 함께 확인할 수 있었습니다.

CAM – 결론

CAM 분석을 통해 모델이 실제 차선 구조를 근거로 판단하고 있음을 확인했으며,

동시에 라이더 시점 데이터에서 발생할 수 있는 bias 가능성도 함께 확인할 수 있었습니다.

감 사 합 니 다