

저해상도 조류 이미지 분류

분류왕

Team 분류왕 | 18_강동헌 | 20_강민정 | 21_김연주 | 21_송상현

강동헌, 강민정, 김연주, 송상현이 팀원이고,
우리의 프로젝트 주제는 "새 분류 모델" 만드는 거야.
재치있는 팀명을 지어줘.

재미있고 센스 있는 팀명을 고려해봤어! 몇 가지 아이디어를 줄게.

1. "분류왕" - 최고의 분류 모델을 만들겠다는 포부!
2. "ClassiFi" - Classification(분류) + Fi(팀워크, Finish 의미)
3. "Alrow" - AI + Arrow (정확하게 분류하겠다는 의미)
4. "LabelMates" - 머신러닝에서 'Label'을 다루고, 팀원(Mates)이란 뜻

CONTENTS

01

주제

대회 주제 설명
데이터셋 정의

02

전처리

Augmentation
Detection
Super-Resolution

03

Classification


Model1 - swinV2
Model2 - BEiT2
실험설계

04

결론

분석 결과 및 개선점





01. 주제

01. DAICON 저해상도 조류 이미지 분류 AI 경진대회

입력으로 들어오는 64x64 크기의 저해상도 조류 이미지로부터 종을 분류하는 AI 알고리즘 개발

- Train : 학습용 저해상도 조류 이미지 15,834장
- Test : 평가용 저해상도 조류 이미지 6,786장
- Upscaled Train : 학습용 이미지의 고해상도 버전 (256x256)

저해상도 조류 이미지 분류 AI 경진대회

알고리즘 | 월간 데이콘 | Vision | 분류 | Macro F1 Score

🏆 상금 : 인증서 + 데이스쿨

🕒 2024.04.08 ~ 2024.05.06 09:59 [+ Google Calendar](#)

👤 1,000명 📅 마감



연습

01. DATASET

TRAIN



UPSCALE_TRAIN



TEST



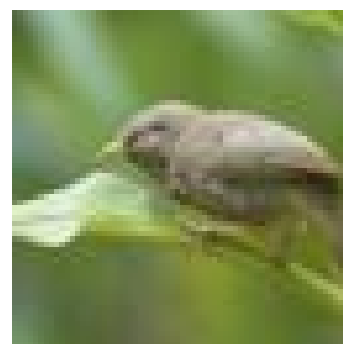
Common Myna



02. 전처리

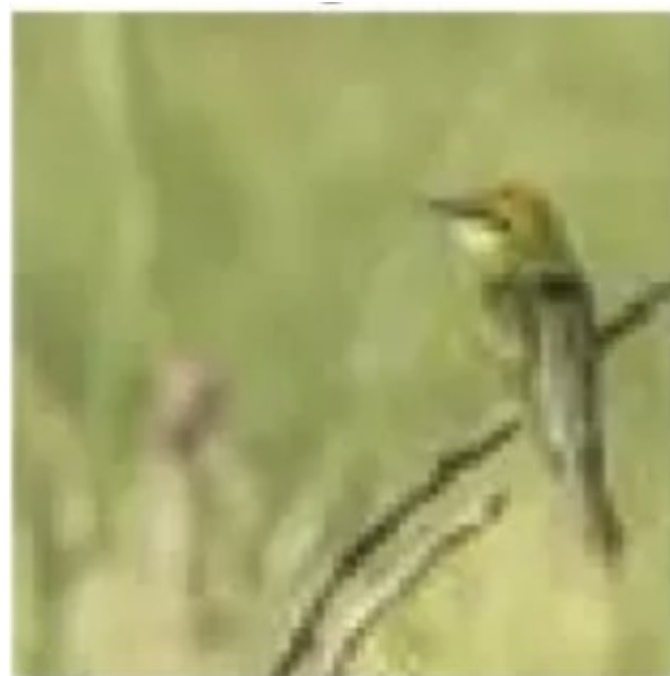
02. DATA AUGMENTATION

- Colorjitter : brightness, contrast, saturation 등 조정
- Resize : 64x64 → 256x256
- Horizontal Flip : 0.5 확률로 좌우 반전

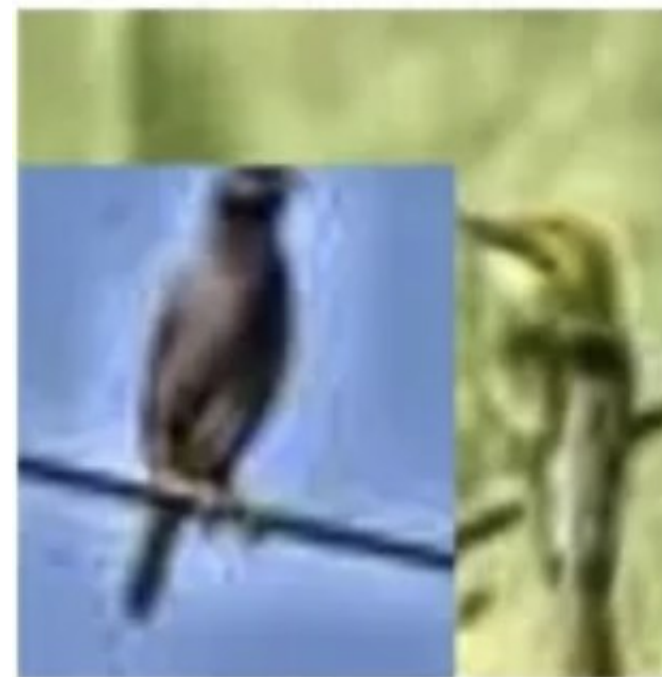


02. DATA AUGMENTATION

- Cutmix : 이미지에서 잘라낸 일부분을 다른 이미지에 삽입
→ 만들어진 이미지 속 특정 class가 차지하는 비율을 새로운 label로 저장



Class A



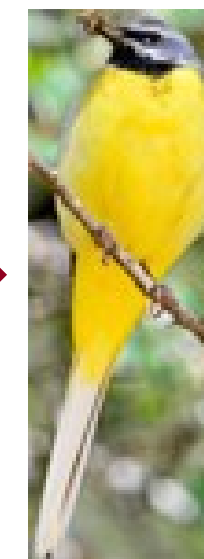
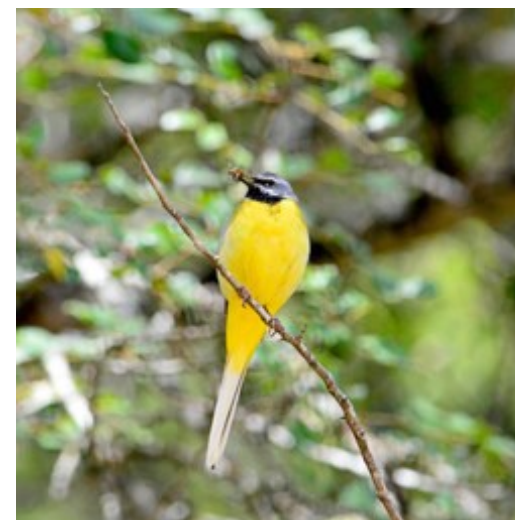
Class 0.6A 0.4B

02. DETECTION

- 문제점 : COCO dataset으로 학습된 yolov8x.pt 모델은 저해상도 사진에서 새 탐지 불가
- 해결방안 : Fine-tuning
 1. 고해상도 사진에서 새 탐지 (box, contour line 두 가지로 나누어 학습)



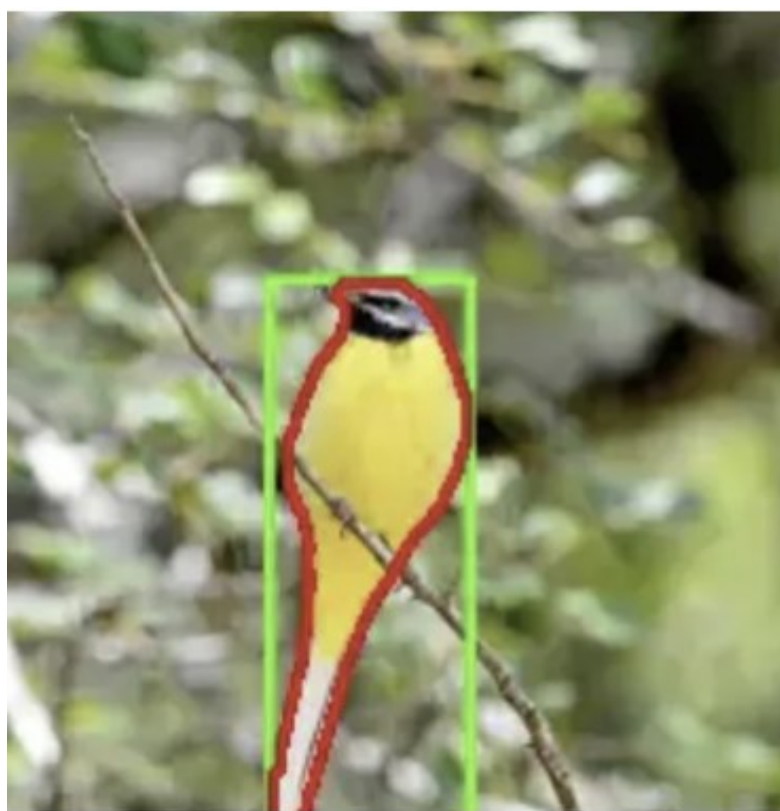
저해상도



고해상도

02. DETECTION

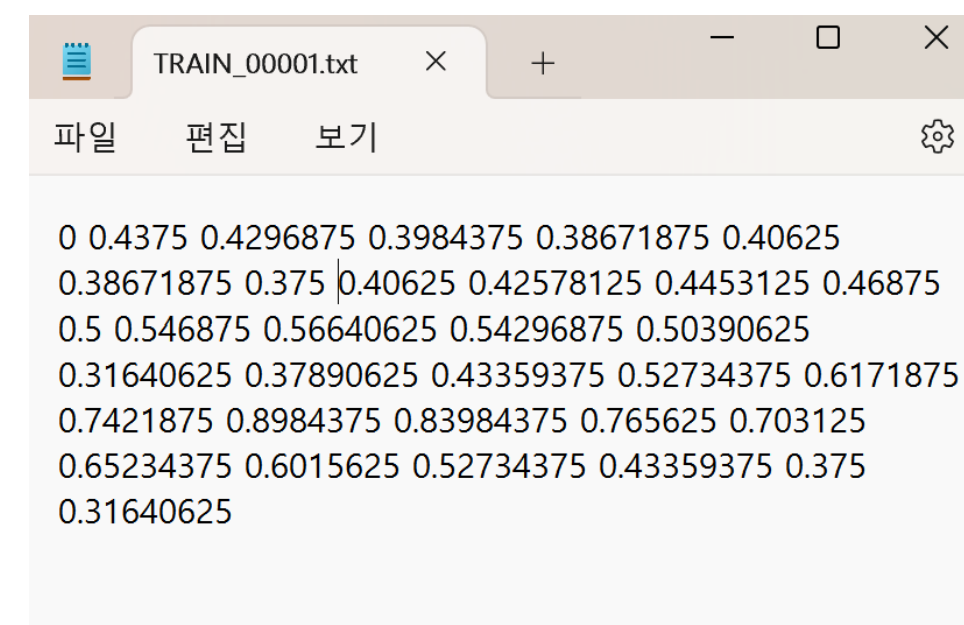
2. 탐지된 박스와 컨투어라인의 좌표를 json 파일로 저장
3. 저해상도 사진에 해당 좌표 적용하여 label 형성 및 YOLO Fine-tuning



detect

```
"id": 2228,  
"image_id": 1581,  
"category_id": 1,  
"bbox": [  
  183,  
  64,  
  72,  
  125  
],  
"segmentation": [  
  199,  
  65,  
  197,  
  67,  
  192,  
  67,  
  191,  
  68,
```

.json

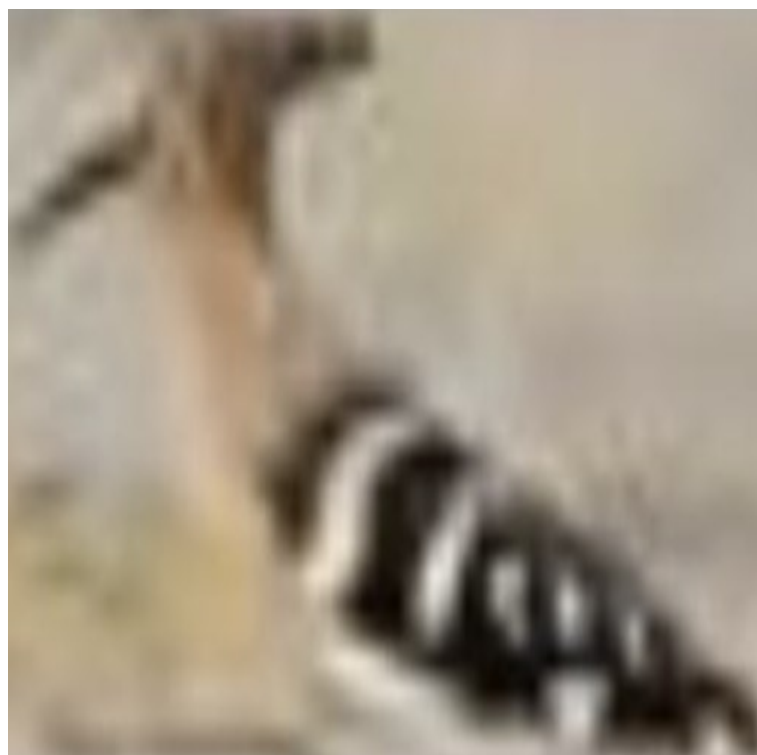


label

02. DETECTION



Non-resize



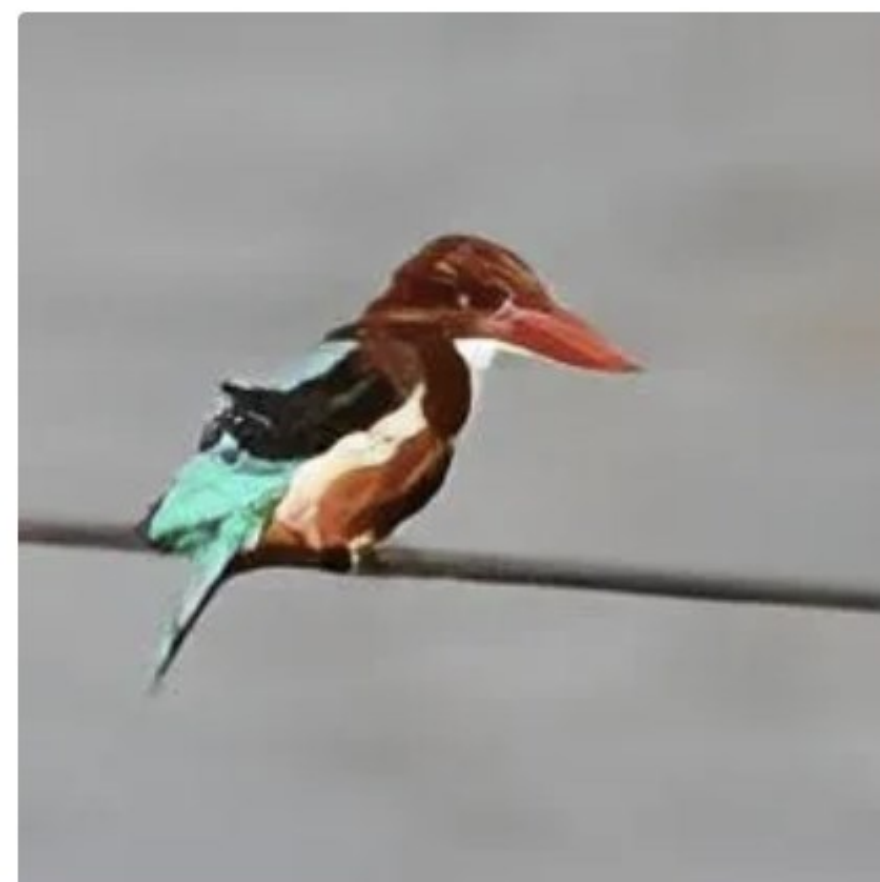
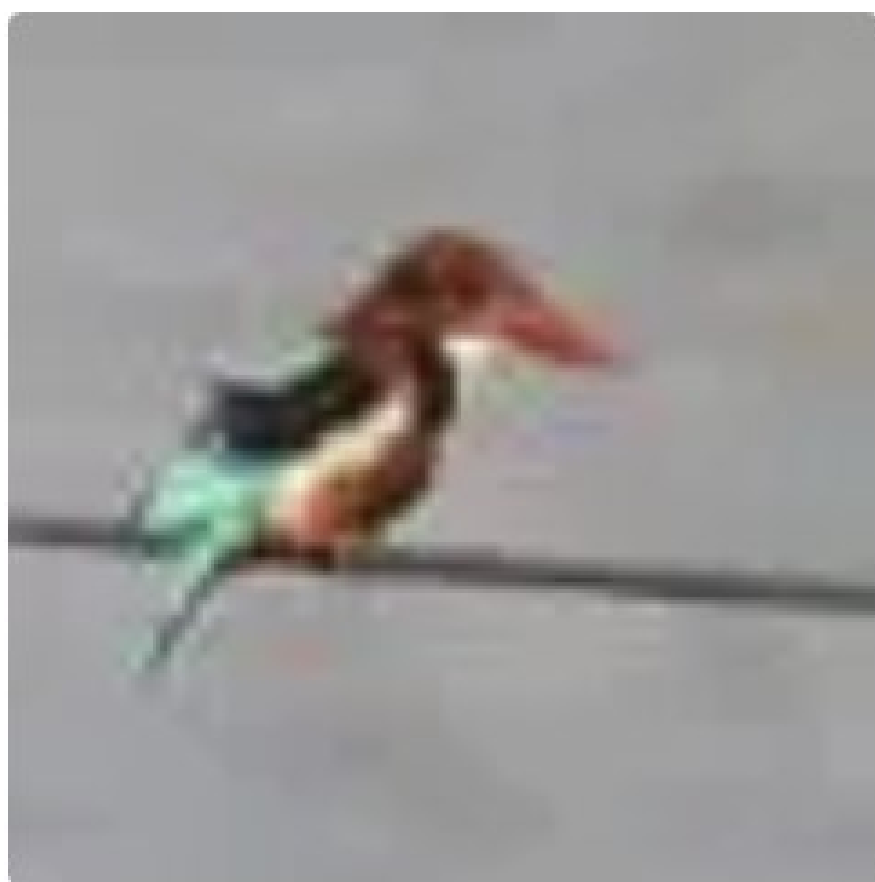
resize



비율 유지하며 resize

02. SUPER-RESOLUTION

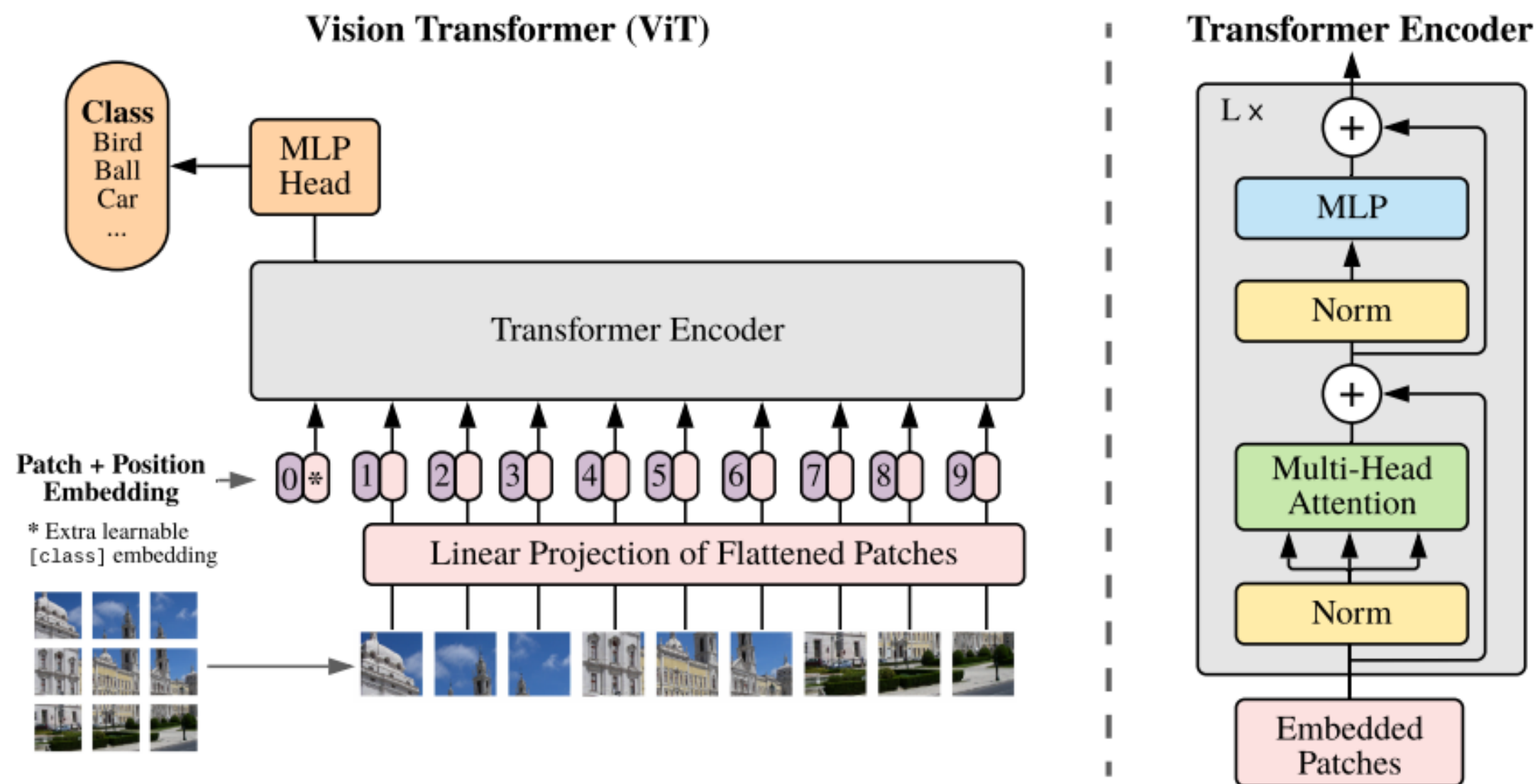
- 사전 학습된 Real-ESRGAN 모델을 이용한 test data upscaling





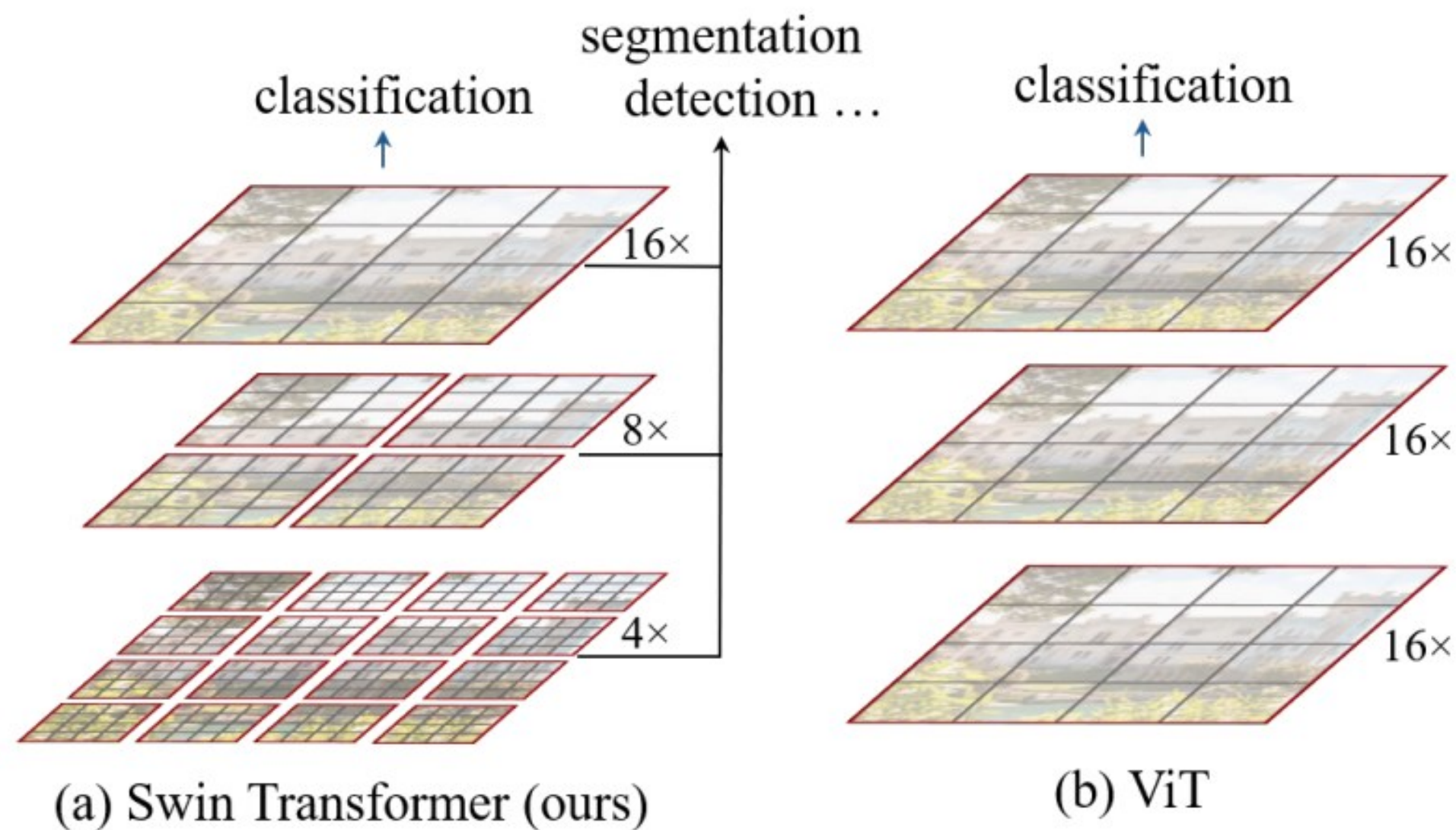
03. Classification

03. ViT



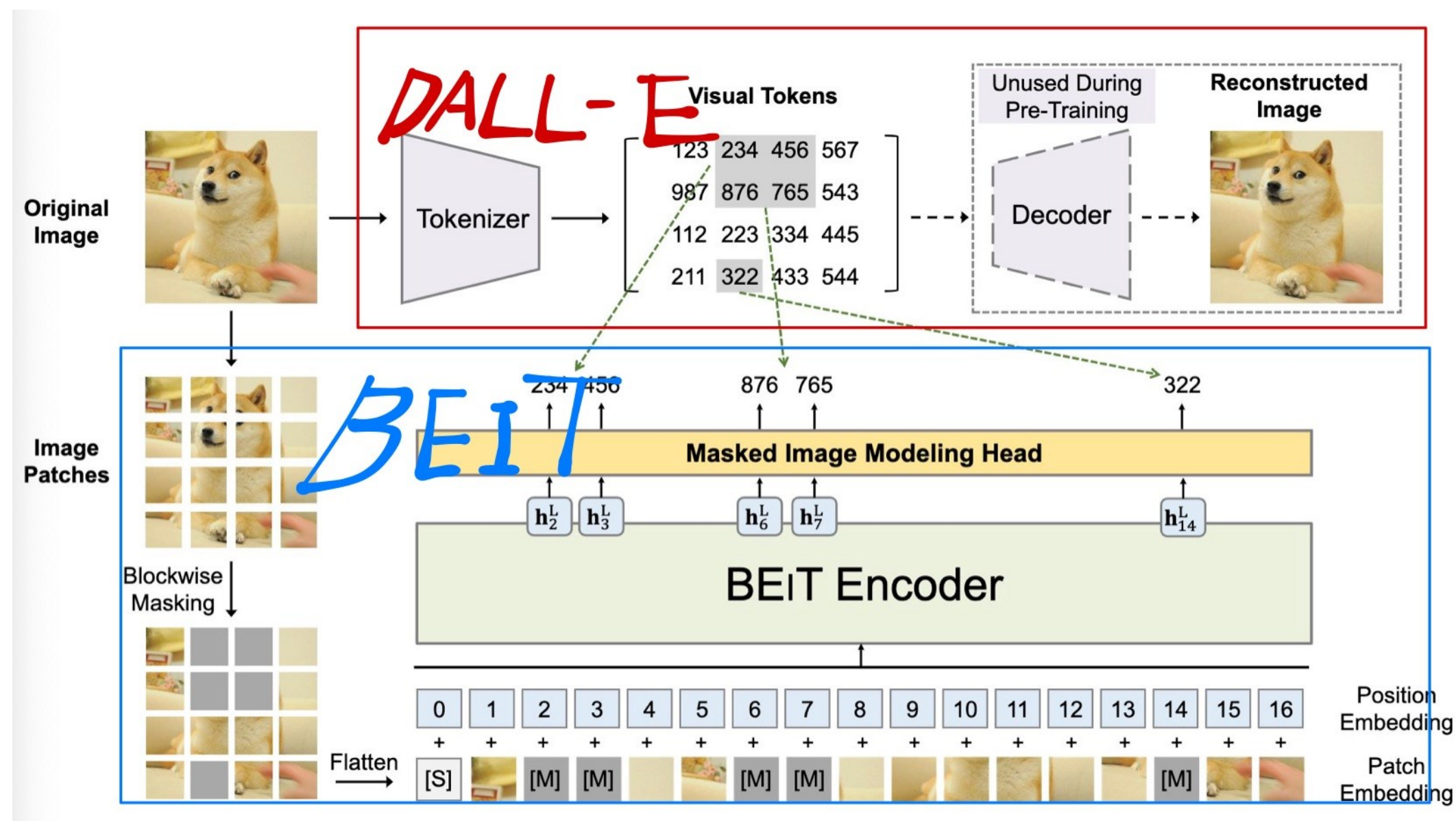
- 패치 분할 및 임베딩 + [CLS] 토큰 + 포지션 임베딩 = 입력 Embedding 생성
- 트랜스포머 인코더 블록 내에서 토큰 간 Self-Attention 및 FFN으로 학습 진행

03. Model1 - SwinV2



- 로컬 윈도우 내에서만 어텐션을 계산, Window Shift로 전역 정보 파악, 연산량 ↓
- 단계 별로 패치 병합(Downsampling)을 하여 전역적 특징 학습(CNN-like)

03. Model2 - BEiT



- DALL-E의 dVAE 모델로 패치 단위 토큰나이징 (픽셀이 아닌 Semantic하게 파악)
- 일부 패치를 Masking하고 [M] 패치의 이산화 된 토큰을 예측하도록 ViT가 훈련됨

03. 실험설계 - SwinV2

- 사용 모델 : microsoft/swinv2-base-patch4-window16-256ImageNet-21k 사전학습 후 ImageNet-1k(1000 클래스)로 파인 튜닝된 모델

```
class SwinClassifier(pl.LightningModule):
    def __init__(self, num_classes, model_name=model_name, learning_rate=CFG['LEARNING_RATE']):
        super().__init__()
        self.save_hyperparameters()
        self.learning_rate = learning_rate
        self.backbone = AutoModel.from_pretrained(model_name)
        self.latent_dim = self.backbone.num_features
        self.classifier = nn.Linear(self.latent_dim, num_classes)

        self.validation_step_outputs = []

    def forward(self, pixel_values):
        outputs = self.backbone(pixel_values=pixel_values)
        pooled_output = outputs.last_hidden_state[:, 0]
        logits = self.classifier(pooled_output)
        return logits

model = SwinClassifier(num_classes=num_classes)

# Trainer 생성 (accelerator="auto"로 GPU 사용 가능 시 자동 선택)
trainer = pl.Trainer(
    max_epochs=CFG['EPOCHS'],
    accelerator="auto",
    devices="auto",
    precision=16,
    accumulate_grad_batches=2,
    callbacks=[checkpoint_callback, early_stop_callback]
)
```

- Batch_size : 16, epochs : 20, learning_rate : 0.00005
- accumulate_grad_batches=2로 설정하여 메모리 효율적인 배치 증가 효과

03. 실험설계 - BEiT

- 사용 모델 : microsoft/beit-base-patch16-224-pt22k-ft22k
- ImageNet-21k 데이터셋으로 자가 지도학습(마스킹 복원) 후 분류 파인 튜닝된 모델

```
def configure_optimizers(self):
    optimizer = torch.optim.AdamW(
        self.parameters(),
        lr=self.learning_rate,
        weight_decay=1e-2
    )

    # ReduceLROnPlateau 스케줄러만 사용
    plateau_scheduler = torch.optim.lr_scheduler.ReduceLROnPlateau(
        optimizer,
        mode='min',
        factor=0.7,      # 개선 없을 경우 학습률 70%로 감소
        patience=2,      # 2 에폭 동안 개선 없으면 감소
        min_lr=1e-7,     # 학습률이 너무 낮아지지 않도록 최소치 설정
        verbose=True,
    )

    # 조기종료
    from pytorch_lightning.callbacks import EarlyStopping

    early_stop_callback = EarlyStopping(
        monitor="val_f1",      # f1 score 모니터링
        min_delta=0.001,      # 개선으로 간주될 최소 변화량
        patience=10,
        verbose=True,
        mode="max"
    )
```

- Batch_size : 32, epochs : 100 (val_F1 score 계산 후 early stop), optimizer : AdamW
- learning_rate : 0.00005, ReduceLROnPlateau 스케줄러 이용하여 2 에폭 동안 F1_score 개선이 없을 시 학습률 0.7배로 감소



04. 결론

04. 분석결과

1. SwinV2

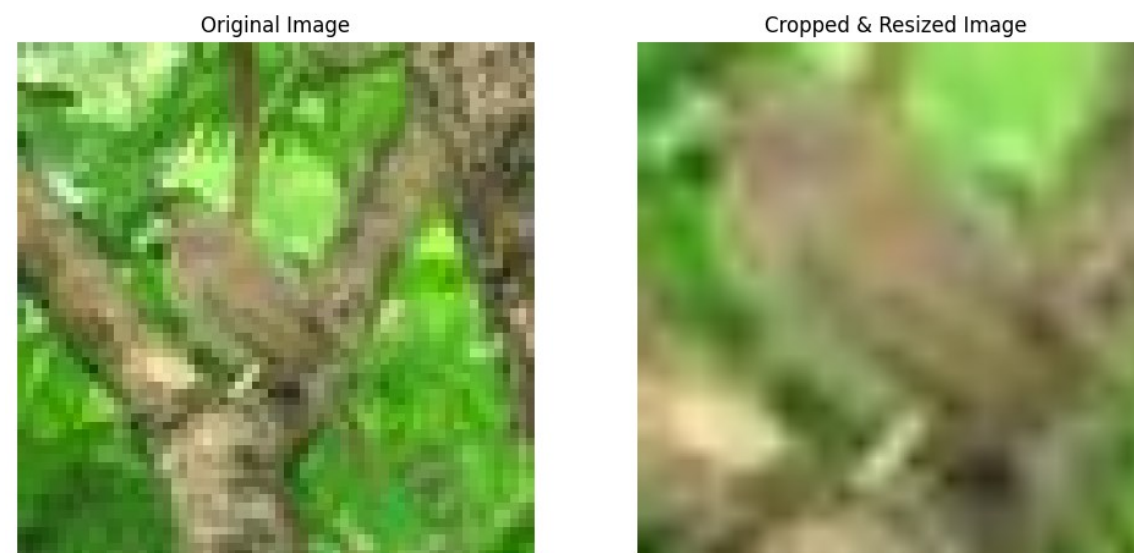
A : SwinV2 (base)	B : SwinV2 (upscale + Cutmix)	C : SwinV2 (테스트 이미지 super resolution)	
0.939, 0.946	0.957, 0.964	0.923, 0.926	
D : SwinV2 (detection - contour line)	E : SwinV2 (detection – padding)	F : SwinV2 (detection – 비율 유지 후 resizing)	G : SwinV2 (detection – 강제 resizing)
0.928, 0.931	0.922, 0.924	0.939, 0.940	0.920, 0.926

2. BeiT

A : BeiT (base)	B : BeiT (Cutmix)	C : BeiT (upscale + learning rate 조절 + cutmix)	
0.668, 0.681	0.753, 0.748	0.954, 0.962	
D : BeiT (detection – contour line)	E : BeiT (detection – padding)	F : BeiT (detection – 비율 유지 후 resizing)	G : BeiT (detection – 강제 resizing)
0.910, 0.910	0.902, 0.903	0.928, 0.926 (base) 0.888, 0.891 (upscale + cutmix)	0.911, 0.906

04. 개선점

- BeiT 모델보다 Swin 모델의 기본적인 성능이 훨씬 나아보였으나, 데이터의 증량 (upscaled 데이터 추가), learning rate 수정 등 변경 사항으로 충분히 성능을 끌어올릴 수 있었음
- 이미지 전처리 시, 객체 탐지를 진행한 뒤 해당 영역만 crop 하여 학습시킨 것보다 그냥 이미지 원본 자체를 학습시키는 게 성능이 더 좋았음
- Albumentations 라이브러리의 resize 방식으로 detail 손실 과함 ~ 객체 탐지 이후 upscale 필요성
- 컴퓨팅 리소스의 한계로 모델의 복잡도와 다양성을 늘리는 것보다, 전처리 과정에 초점을 두고 여러 방법들을 테스트해보았지만 성능이 만족스럽지 않았음 !





Thank You