조선대학교 CHOSUN UNIVERSITY

DACON

# 이미지 색상화 및 손실 부분 복원 AI 경진대회

조선대학교 인공지능공학과 정지성
고승우
오한결
정지민
섭서호

# Contents.

# 01 / 개요

# 01. 개요

## 대회목적

- 손상된 이미지의 특정 영역을 복구하고
  흑백 이미지에 자연스러운 색상을 입히는 AI 알고리즘 개발

## 활용분야

- 역사적 사진복원
- 영상편집
- 의료 이미지 복구

## 리더보드

평가 산식: SSIM(Structural Similarity Index Measure), 색상 Histogram 유사도

- **S: 전체 SSIM 평균**
- **M: 손실 영역 SSIM 평균**
- **C: 색상 유사도 평균**

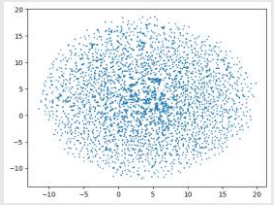
$$\text{Score} = (0.2 \times S) + (0.4 \times M) + (0.4 \times C)$$

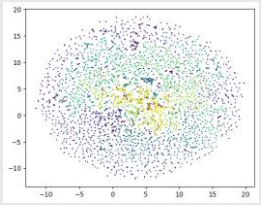
# 02 / EDA

# 02. EDA

## Data composition

- Train_input : 흑백, 일부 손상된 PNG 학습 이미지 (input, 29603장)
- Test_input : 흑백, 일부 손상된 PNG 학습 이미지 (input, 100장)
- Labels : 2456개, Noise : 2162





```
len(set(labels)), sum(labels==-1)
```

```
(2456, 2162)
```

- 손상된 PNG : 흑백화 및 특정 영역 무작위 마스킹 처리

- 원본과 손상된 이미지


# 03 / 솔루션 소개

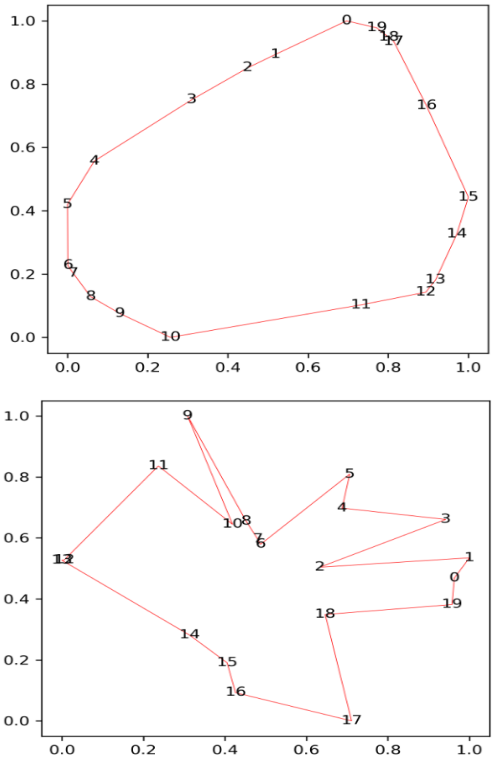
## 이미지 처리

```
pip install polygenerator lightning segmentation-models-pytorch
```

```
    num_points = random.randint(3,20)
    polygon_func = random.choice([
        random_polygon,
        random_star_shaped_polygon,
        random_convex_polygon
    ])
    polygon = polygon_func(num_points=num_points) #scaled 0~1
    polygon = [(round(r*mask_width), round(c*mask_height)) for r,c in polygon]
    polygon_mask = skimage.draw.polygon2mask((mask_width, mask_height), polygon)
    if np.sum(polygon_mask)>(min_polygon_bbox_size//2)**2:
        break
full_image_mask = np.zeros((width, height), dtype=np.uint8)
full_image_mask[bbox_x1:bbox_x2, bbox_y1:bbox_y2] = polygon_mask
```

- random_polygon: 무작위 비제한적 다각형 생성
  random_star_shaped_polygon: 별 모양 다각형 생성
  random_convex_polygon: 볼록 다각형 생성

- 생성된 다각형은 이미지에 마스크로 적용



# 03. 솔루션 소개

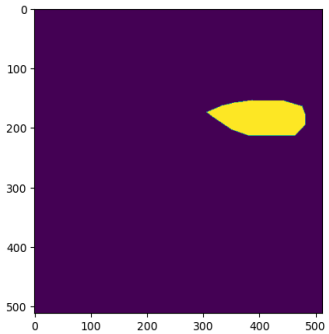
## 이미지 처리

```
image_gray = image.convert('L')
image_gray_array = np.array(image_gray)  # Convert to numpy
random_color = random.randint(0, 255)  # Random grayscale
image_gray_array[full_image_mask == 1] = random_color
image_gray_masked = Image.fromarray(image_gray_array)
```

```
bbox_x1 = random.randint(0, width-min_polygon_bbox_size)
bbox_y1 = random.randint(0, height-min_polygon_bbox_size)
bbox_x2 = random.randint(bbox_x1, width)  # Ensure width > 10
bbox_y2 = random.randint(bbox_y1, height)  # Ensure height > 10
if (bbox_x2-bbox_x1)<min_polygon_bbox_size or (bbox_y2-bbox_y1)<min_polygon_bbox_size:
    continue

mask_bbox = [bbox_x1, bbox_y1, bbox_x2, bbox_y2]
mask_width = bbox_x2-bbox_x1
mask_height = bbox_y2-bbox_y1
```

```
def normalize_image(self, image):
    return (np.array(image)/255-self.mean)/self.std
```




- Grayscale 변환(계산 복잡도, 채널 통일) : PIL 라이브러리의 convert('L') 메서드를 사용하여 컬러 이미지를 흑백으로 변환
  RGB 이미지를 흑백으로 변환할 때는 각 색상 채널에 가중치를 적용하여 계산
  $Y = 0.299R + 0.587G + 0.114B$

- 정규화와 표준화: 이미지 픽셀값을 0~1 범위로 정규화 (/255)평균(self.mean)을 빼고 표준편차(self.std)로 나누어 표준화

- 크기 처리: 랜덤한 위치에서 bounding box 좌표 생성
  min_polygon_bbox_size를 고려하여 최소 크기 보장
  이미지 내에서 유효한 영역만 선택되도록 범위 제한

# 03. 솔루션 소개

Train (L1, L2) => 데이터에 노이즈가 많거나 에러가 큰 데이터가 거의 없는 상황 모두에 대비하기 위해 L1 + MSE(L2) loss combine 사용

```
def training_step(self, batch, batch_idx):
    masks, images_gray_masked, images_gray, images_gt = batch['masks'], batch['images_gray_masked'], batch['images_gray'], batch['images_gt']
    images_gray_restored, images_restored = self(images_gray_masked)

    loss_pixel_gray = F.l1_loss(images_gray, images_gray_restored, reduction='mean') * 0.5 + F.mse_loss(images_gray, images_gray_restored, reduction='mean') * 0.5
    loss_pixel = F.l1_loss(images_gt, images_restored, reduction='mean') * 0.5 + F.mse_loss(images_gt, images_restored, reduction='mean') * 0.5
    loss = loss_pixel_gray * 0.5 + loss_pixel * 0.5

    self.log("train_loss", loss, on_step=True, on_epoch=False)
    self.log("train_loss_pixel_gray", loss_pixel_gray, on_step=True, on_epoch=False)
    self.log("train_loss_pixel", loss_pixel, on_step=True, on_epoch=False)
    return loss
```

: 회색 복원 이미지(images_gray_restored)와 원본 회색 이미지(images_gray)를 유사하게 학습

: 최종 복원된 이미지(images_restored)와 원본 이미지(images_gt)를 유사하게 학습

**L1 loss** $L = \sum_{i=1}^{n} |y_i - f(x_i)|$

- 노이즈, outlier(에러가 큰 데이터 포인트)로 손실 값 급증가 방지
- 실제값 - 예측값 차이값에 절댓값(오차합 최소화)
- 절댓값 기반, 노이즈에 강함
- Outlier에 둔감
- 기울기가 일정 -> 안정적인 학습

**MSE loss(L2 loss)** $L = \sum_{i=1}^{n} (y_i - f(x_i))^2$

- 섬세한 복원 품질 향상
- 실제값 - 예측값 차이값 제곱합
- F.mse_loss는 PyTorch의 Functional API 제공 함수
- 제곱값 기반, 오차 강조
- Outlier에 민감

# 03. 솔루션 소개

## Optimizer

```
def configure_optimizers(self):
    return torch.optim.AdamW(self.parameters(), lr=1e-4)
```

optimizer: AdamW(Adam with Weight Decay)
Adam의 변형, 가중치 감쇠 적용
L2 정규화를 통한 모델 가중치 ↓ -> 모델 복잡성 제어, 오버피팅 완화

self.parameter(): 모델의 모든 학습 가능한 파라미터 반환
lr=1e-4: 학습률, 0.0001로 설정 -> 안정적이고 효율적인 작은 값

## Early stopping

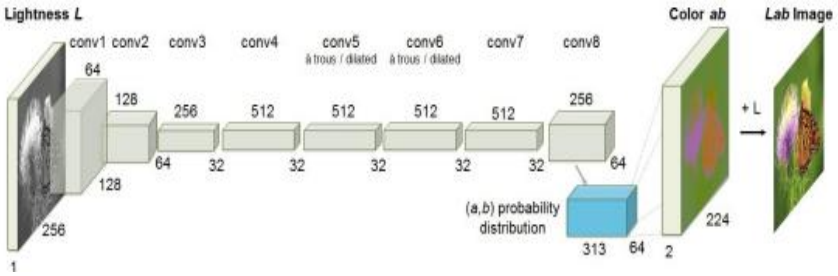
```
earlystopping_callback = EarlyStopping(monitor="val_score", mode="max", patience=3)
```

Early stopping: 학습 중 검증 성능 일정 기간 동안 향상 X -> 학습 조기 종료 = 과적합 방지
monitor="val_score": 모니터링 할 값, earlystopping의 기준
mode="max": val_score 최대화로 학습
patience=3: 기다리는 기간, 3번 연속epoch동안 개선 X -> 학습 종료

# 04 / 시행착오

# 04. 시행착오

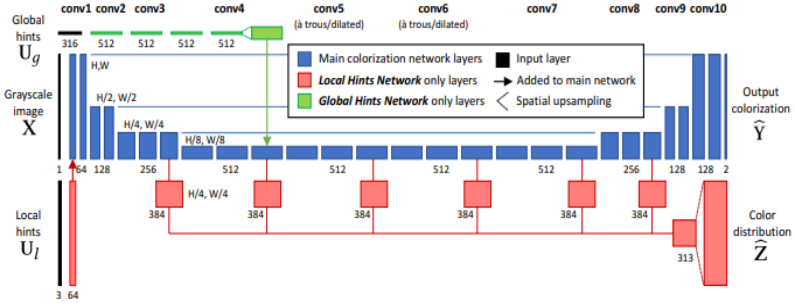
◆ **Colorization모델 탐색 : eccv16, siggraph17**




- 흑백 이미지를 8개의 컨볼루션 레이어로 통과
- conv8에서 313개 색상 분포(ab값) 예측
- 원본 L채널과 예측된 ab채널 결합해 컬러 이미지 생성

<제출 결과>

0.3929052742




- 메인 컬러화 네트워크(파란색)와 로컬 힌트 네트워크(빨간색)로 구성된 두 가지 병렬 구조
- 그레이스케일 이미지와 글로벌/로컬 힌트를 입력으로 받아 10개의 컨볼루션 레이어 통과
- 두 네트워크의 특징을 결합하여 최종적으로 313차원의 색상 분포(Z) 예측

<제출 결과>

0.4813265431

# 04. 시행착오

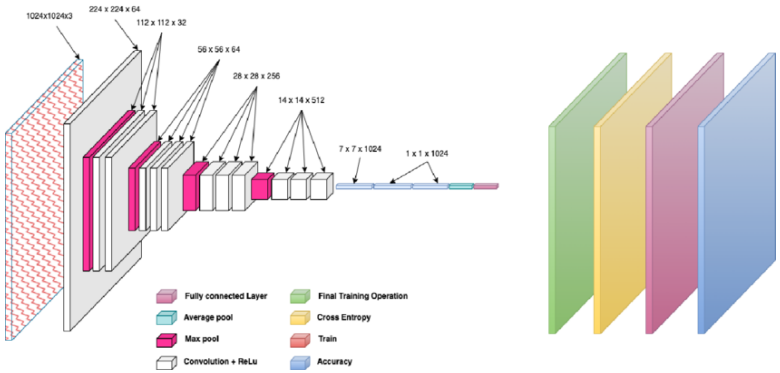
## ◆ MobileNet

### ▪ Code

```
model_1 = smp.Unet(
    encoder_name="mobilenet_v2",
    encoder_weights="imagenet",
    in_channels=1,
    classes=1,
    decoder_use_batchnorm=True,
    decoder_attention_type="scse"
)

model_2 = smp.Unet(
    encoder_name="efficientnet-b0",
    encoder_weights="imagenet",
    in_channels=1,
    classes=3,
    decoder_use_batchnorm=True,
    decoder_attention_type="scse"
)
```

### ▪ MobileNet Architecture




- EfficentNet-B0는 경량화되고 효율적인 인코더로, 높은 성능-자원 효율성을 제공
- 적은 계산량으로도 고품질의 특징 추출.
- 주로 메모리가 제한적인 환경에서 이진 세그멘테이션에 적합.

Val Score : 40 Epoch 0.5872

Public Score: 0.49569

# 04. 시행착오

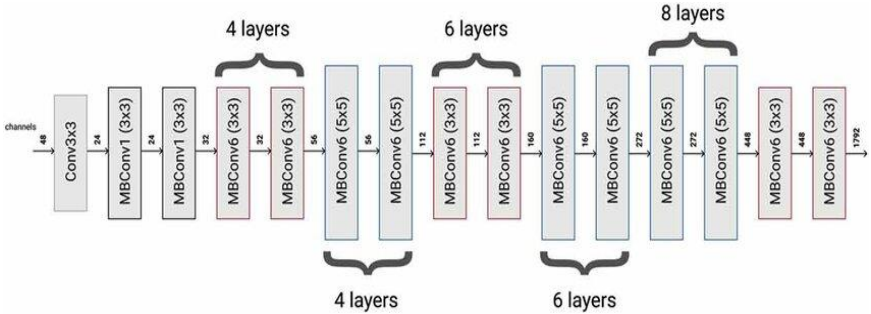
◆ **Efficientnet-b4**(Resnext50_32x4d)

- **Code**

```
# Define models
model_1 = smp.Unet(
    encoder_name="efficientnet-b4",  # EfficientNet for gray mask restoration
    encoder_weights="imagenet",
    in_channels=1,  # Input channel for grayscale
    classes=1,  # Output single-channel grayscale
)

model_2 = smp.Unet(
    encoder_name="resnext50_32x8d",  # ResNeXt for RGB restoration
    encoder_weights="imagenet",
    in_channels=1,  # Input channel for grayscale restoration
    classes=3,  # Output RGB channels
)
```

- **Efficientnet-b4 Architecture**




- 상대적으로 적은 파라미터와 계산량으로 높은 성능을 제공하는 **Efficientnet-b4**를 사용

- **ResNet**을 개선한 이미지 분류 모델로 계산량 증가를 최소화,
  다양한 특성 을 학습하는 **Bottleneck** 구조인 **ResNext50_32x4d** 모델 사용

Val Score: 36 Epoch 0.6176

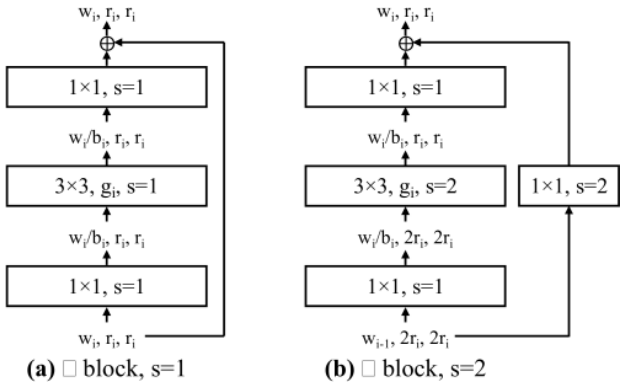
# 04. 시행착오

## ◆ Regnetx

```python
# Gray mask restoration
model_1 = smp.Unet(
    encoder_name="timm-regnetx-008",
    encoder_weights="imagenet",  # pretrained=False 대신 사용
    in_channels=1,
    classes=1,
)

# Color restoration
model_2 = smp.Unet(
    encoder_name="timm-regnetx-008",
    encoder_weights="imagenet",  # pretrained=False 대신 사용
    in_channels=1,
    classes=3,
)

lit_ir_model = LitIRModel(model_1=model_1, model_2=model_2)
```




- timm-regnetx-008
- PyTorch의 timm (PyTorch Image Models) 라이브러리에서 제공하는 RegNetX 모델
- RegNetX는 Facebook AI에서 개발한 네트워크 아키텍처
- ReLU 함수 사용, resnet와 같은 Bottleneck Block 구조
- 008: 중간 크기의 regnetx

Val Score : 22 Epoch 0.5512

Public : 0.4906216176

# 05 / 최적화

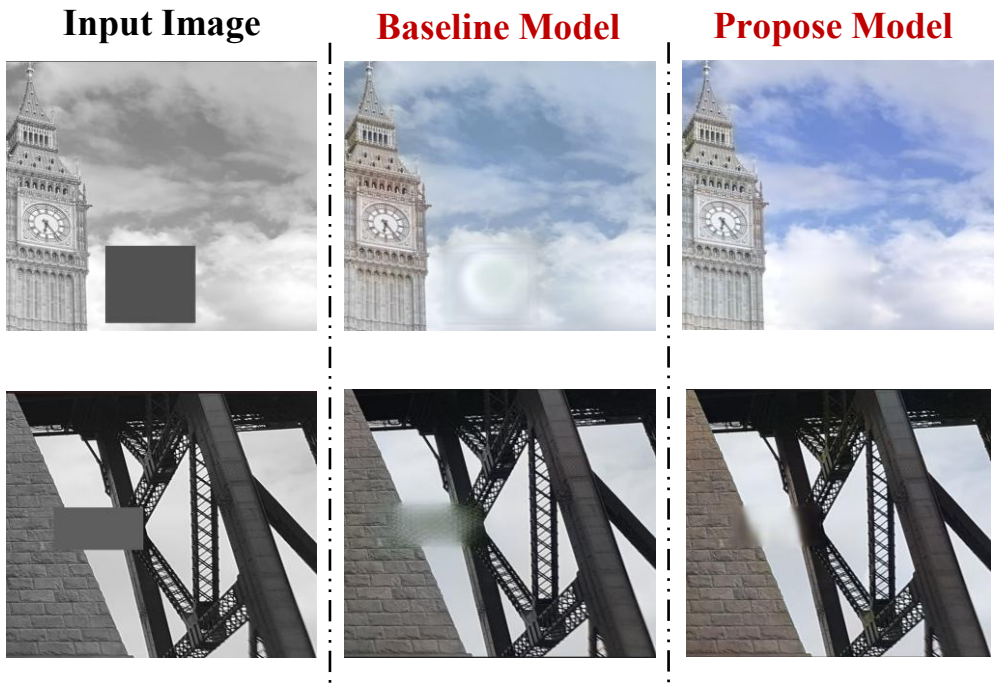
# 05. 최적화

## 이미지 복원

- **U-Net**(EfficientNet-B0)
- **U-Net**(MobileNet-V2)
- **MAnet**(EfficientNet-B1)
  (Multi-scale Attention Network)

## 이미지 색상화

- **U-Net**(EfficientNet-B1)
- **DeepLabV3**(EfficientNet-B2)
- **FPN**(ResNet50)
  (Feature Pyramid Network)




Val Score : 40 Epoch 0.6761

Public Score: 0.57615

# 06 / 역할분담

# 06. 역할분담

| | |
|---|---|
| 정지성 | • 프로젝트 관리 및 최종 솔루션 개발<br>• 복원과 색상화 모델의 통합 및 최적화 수행 |
| 고승우 | • 색상화 모델 구현 및 초기 개발 환경 설정<br>• 모델 학습과 테스트 환경 구축 및 결과 분석 |
| 오한결 | • 데이터 특성과 손상 패턴에 대한 EDA 수행<br>• 색상화 모델 탐색 및 데이터 전처리 기법 탐색 |
| 정지민 | • 최신 이미지 복원 모델 탐색 및 구현<br>• 모델 학습 스케줄러 설계와 하이퍼 파라미터 튜닝 |
| 섭서호 | • 이미지 복원 및 색상화 논문 리뷰 |

# Thank you.