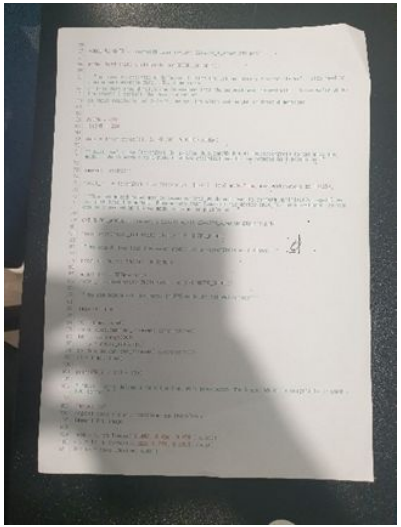
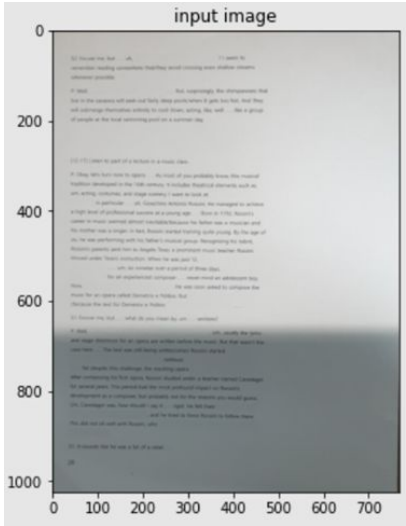
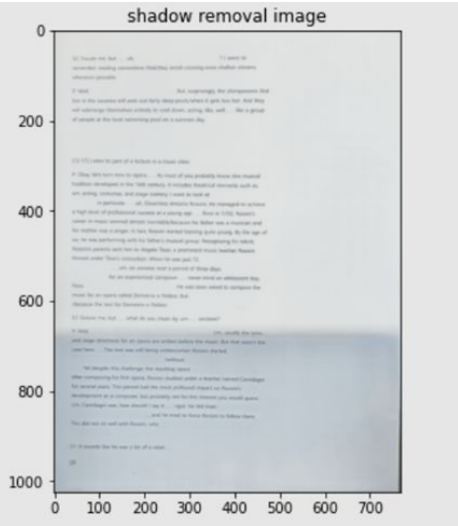
EURON Computer Vision

Team 2

그림자 제거 성능을 탑재한 문서 스캔 알고리즘







목차

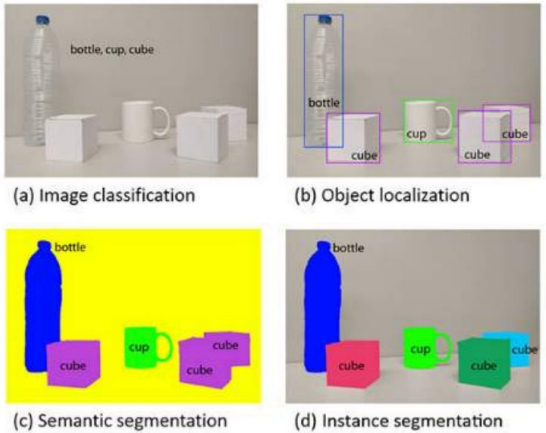
아이디어

스캔본 사진 속 핸드폰 그림자 인식 및 제거 알고리즘

1. 제안 배경
   - 비대면 시험을 치르면서 A4 답안지를 촬영해서 올릴 때 핸드폰 그림자가 생겨 여러 번 스캔을 다시 한 적이 있음.
   - 프린트 된 서류를 사진으로 찍어 pdf로 변환하는 과정에서도 유사한 경험이 많음.

2. 프로젝트 설명
   a. 스마트폰 그림자가 같이 찍힌 그림자 데이터셋 준비
   b. 데이터셋 학습을 통해 주변 색보다 더 어두운 그림자 영역을 detecting하는 AI 모델 개발
   c. detecting된 영역을 Semantic Segmentation으로 영역 지정
   d. 해당 영역의 밝기를 다른 영역 값과 비슷하게 올리는 방식으로 그림자 제거

3. 주요 기술 설명
   a. 위의 데이터셋은 자체적으로 모을 수 있을 것으로 예상함
   b. 위의 b, c에서는 Semantic Segmentation SOTA 기술을 사용할 수 있음
   c. 종이 사진이기 때문에 더 밝은 영역과의 색상 정보 비교가 쉬울 것
      https://jvvp.tistory.com/1030
      위 블로그에 있는 밝기 조절 기능을 detecting된 그림자 영역에만 적용



(a) Image classification (b) Object localization
(c) Semantic segmentation (d) Instance segmentation

진행 과정

- OpenCV를 활용해서 스캔 알고리즘 구성
  - Edge Detection -> Finding the Contours -> Perspective Transformation
  - 그림자제거: OpenCV의 dilate, medianBlur, absdiff, normalize, merge를 활용
  - 입력 이미지의 4개의 꼭짓점을 인식하면 테두리를 읽고 문서를 스캔함

  →이미지를 스캔하고 그림자를 없애기 위해서 그림자 제거 알고리즘을 찾기로 결정

- 스마트폰 그림자가 같이 찍힌 그림자 데이터셋 준비 -> 구글링으로 찾을 수 있는 데이터셋이 존재하지 않음.
  - 구글링으로 'smartphone shadow', '스마트폰 그림자', 'shadow on paper' 등으로 검색했을 때 사용 가능한 사진 데이터가 없었음.
  - 이미 pretrain된 모델을 찾아보았으나 문서 위에 스마트폰 그림자가 찍힌 데이터를 이용한 모델은 찾을 수 없었음.
  - -> 직접 데이터셋을 생성하거나 pretrain된 모델을 찾기 어려웠기 때문에 이미 있는 그림자 제거 알고리즘을 조사하기로 함.

진행 과정

- waterfilling algorithm을 찾아 이를 적용하는 것을 시도
  - 공개된 github 코드를 활용한 예시가 없고 github 코드도 활용 방식을 찾을 수 없었음.
  - OpenCV 문서 스캔 코드를 찾을 수 있었기에 이를 구현하게 되었음.

- waterfilling algorithm 논문에 참고된 다른 논문들의 그림자 제거 알고리즘을 찾아 적용해보고자 함.
  - 오래된 논문의 경우 github에 코드가 공개되어있지 않았음.
  - 최근 논문의 경우에도 논문에 알고리즘에 쓰인 수식은 설명되어있으나 코드가 공개되어있지 않았음.
    -> 기존의 그림자 제거 성능을 가진 딥러닝 모델을 사용하기로 결정

진행 과정

- 기존의 프로젝트 목표인 사진 속 그림자 제거를 주제로 하는 다른 딥러닝 모델을 찾게 됨.
  - https://github.com/GuoLanqing/Awesome-Shadow-Removal
    해당 깃허브에 공개된 논문들 중 코드가 공개되어있는 모델 적용을 시도
  - https://github.com/IsHYuhi/BEDSR-Net_A_Deep_Shadow_Removal_Network_from_a_Single_Document_Image
    이 모델의 경우 demo 파일이 있어 쉽게 test image를 여러가지 넣어볼 수 있었음.
  - https://github.com/vinthony/ghost-free-shadow-removal
    이 경우도 위와 동일

진행 과정

- BEDSR-Net을 이용했을 때 결과물과 그 한계점
  - edge를 자른 이미지를 입력해도 읽지 못하는 경우가 발생.
  - 알고리즘이 그림자 영역을 어떻게 판단하느냐에 따라 그림자 제거 성능이 달라짐
  - 성능향상이 눈에 띄게 좋아졌다고 말하기 어려움.

- 성능을 높일 수 있는 아이디어
  -> original input을 넣었을 때 나온 output을 다시 ghost-free-shadow-removal 알고리즘의 input으로 넣어 그림자가 얼마나 더 제거되는지 확인
  - 한번씩만 알고리즘 거쳐나온 결과물보다 훨씬 좋은 성능을 보여줌
  - 이미지의 화질이 떨어져 글씨가 선명하지 않음

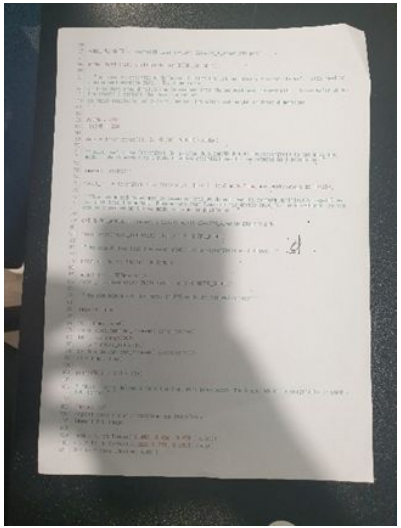
알고리즘 설명

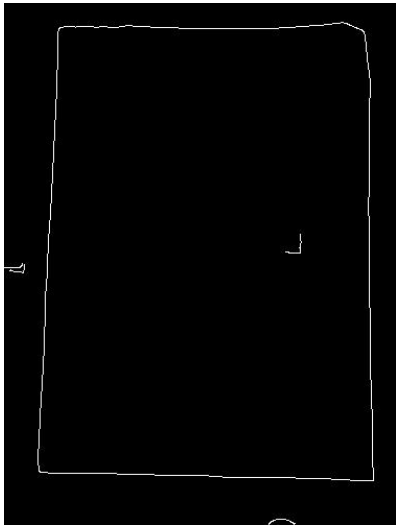
- 1. 스캔 알고리즘

  <STEP 1> Edge Detection

  - cv2.cvtColor를 통해 이미지를 흑백으로 변환
  - cv2.GaussianBlur를 통해 고주파 노이즈 제거
  - cv2.Canny를 이용해서 edge detection

```
# convert the image to grayscale, blur it, and find edges in the image
gray = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)
# GaussianBlur를 통해 고주파 노이즈 제거
gray = cv2.GaussianBlur(gray, (5, 5), 0)
# Canny edge detection
edged = cv2.Canny(gray, 75, 200)
```





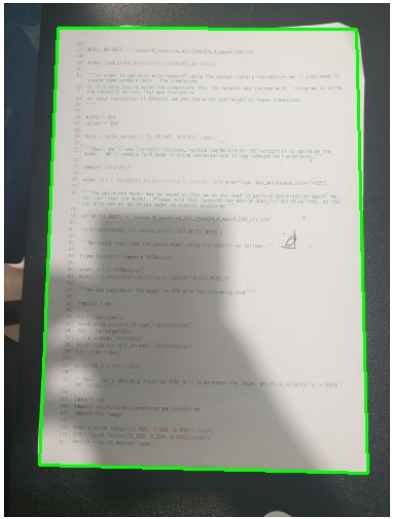
알고리즘 설명

- 1. 스캔 알고리즘

  <STEP 2> Finding the Contours

  - cv2.findContours를 사용하여 <STEP 1>을 통해 얻은 edged image에서 contour 추출
  - cv2.approxPolyDP를 통해 입력 이미지의 4개의 꼭짓점을 인식하고 테두리를 형성하여 이미지를 재구성

```
# find the contours in the edged image, keeping only the largest ones
# and initialize the screen contour
cnts = cv2.findContours(edged.copy(), cv2.RETR_LIST, cv2.CHAIN_APPROX_SIMPLE)
cnts = imutils.grab_contours(cnts)
cnts = sorted(cnts, key=cv2.contourArea, reverse=True)[:5]

# loop over the contours
for c in cnts:
    # approximate the contour
    peri = cv2.arcLength(c, True)
    approx = cv2.approxPolyDP(c, 0.02 * peri, True)

    # if our approximatedd contour has four points,
    # then we can assume that we have found our screen
    if len(approx) == 4:
        screenCnt = approx
        break

# show the contour (outline) of the piece of paper
cv2.drawContours(image, [screenCnt], -1, (0, 255, 0), 2)
```



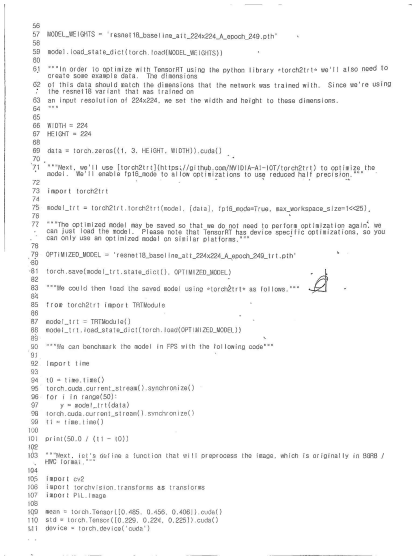
알고리즘 설명

- 1. 스캔 알고리즘

  <STEP 3> Perspective Transformation

  - top-down view로 이미지 변형
  - threshold_local의 gaussian 계산을 통해 이미지를 grayscale로 변경하고 black-size 주변 영역을 이진화함 -> 그림자를 최대한 제거하기 위한 용도

```
# apply the four point transform to obtain a top-down view of the original image
warped = transform.four_point_transform(orig, screenCnt.reshape(4, 2) * ratio)

# convert the warped image to grayscale, then threshold it to give it that 'black and white' paper effect
warped = cv2.cvtColor(warped, cv2.COLOR_BGR2GRAY)
T = threshold_local(warped, 11, offset=10, method="gaussian")
warped = (warped > T).astype("uint8") * 255
```



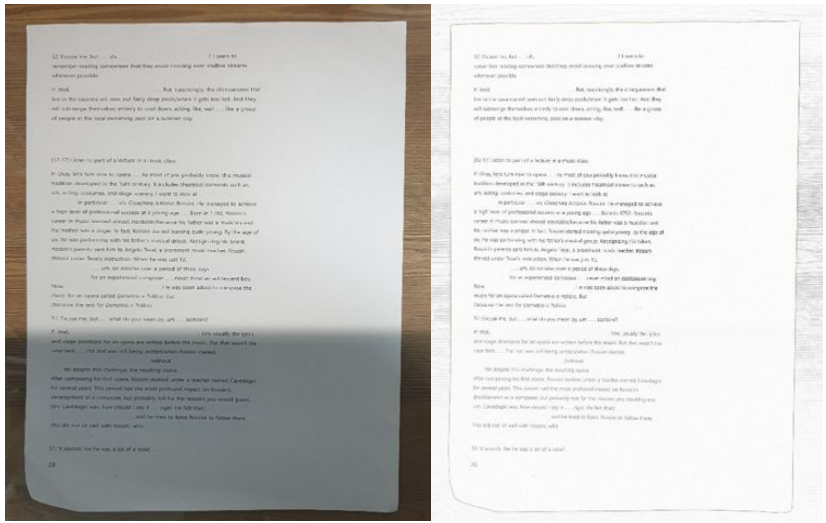
알고리즘 설명

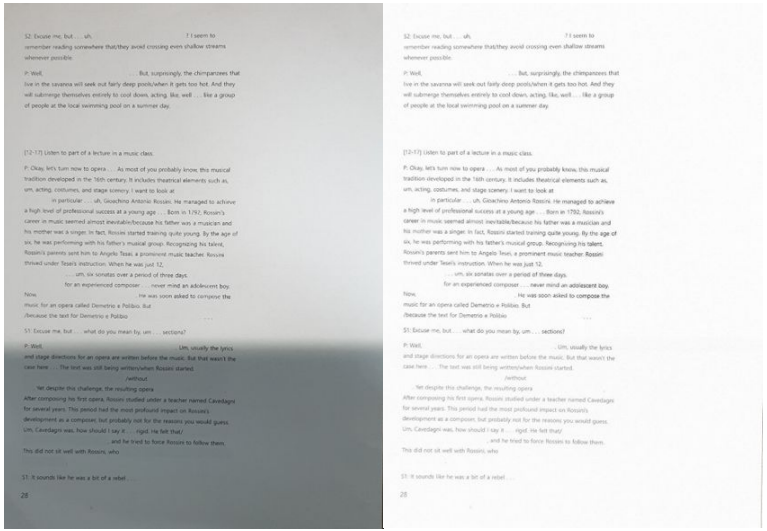
- 2. OpenCV 활용 그림자 제거 알고리즘
  - OpenCV의 dilate, medianBlur, absdiff, normalize, merge를 활용
  - dilate 팽창연산을 통해 영역을 확장시켜 이미지를 인식함
  - medianBlur,absdiff, normalize, merge 를 통해 이미지의 잡음 제거를 하고 그 과정에서 픽셀값에 영향을 주게 됨. (그림자 제거)

```
import cv2
import numpy as np

img = cv2.imread('shadows.png', -1)

rgb_planes = cv2.split(img)

result_planes = []
result_norm_planes = []
for plane in rgb_planes:
    dilated_img = cv2.dilate(plane, np.ones((7,7), np.uint8))
    bg_img = cv2.medianBlur(dilated_img, 21)
    diff_img = 255 - cv2.absdiff(plane, bg_img)
    norm_img = cv2.normalize(diff_img,None, alpha=0, beta=255, norm_type=cv2.NORM_MINMAX,
    result_planes.append(diff_img)
    result_norm_planes.append(norm_img)

result = cv2.merge(result_planes)
result_norm = cv2.merge(result_norm_planes)

cv2.imwrite('shadows_out.png', result)
cv2.imwrite('shadows_out_norm.png', result_norm)
```

수행 결과

- OpenCV의 dilate, medianBlur, absdiff, normalize, merge를 활용한 shadow removal 알고리즘 사용 결과

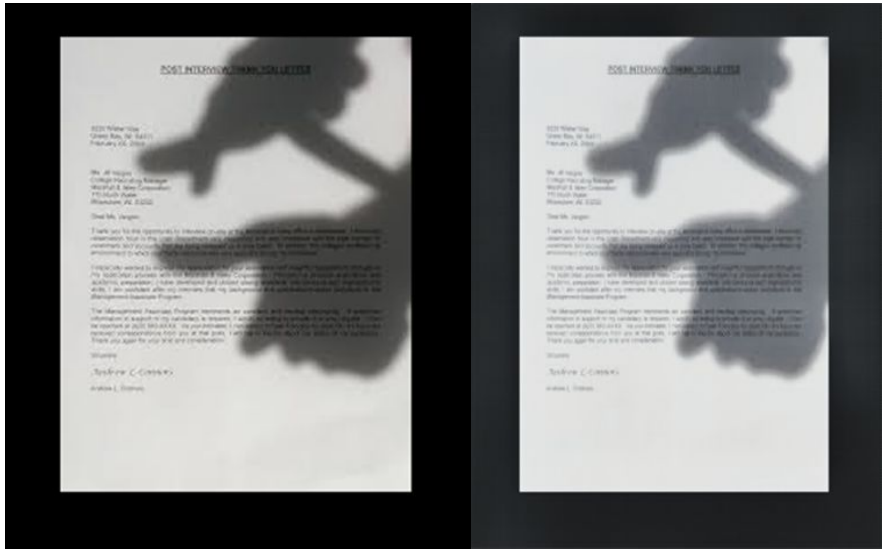




background 유무, 즉 edge의 유무와 상관없이 그림자 제거가 가능하지만
본래 이미지가 손상되는 한계점을 발견할 수 있었다.

수행 결과

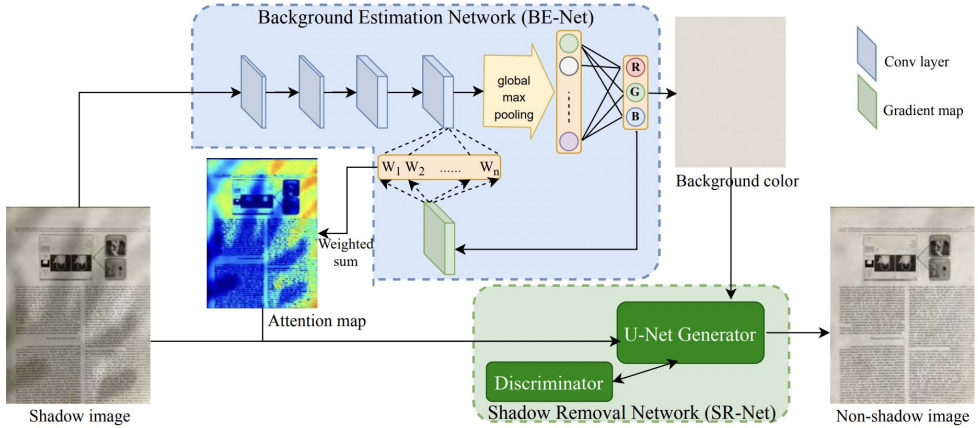
- OpenCV의 dilate, medianBlur, absdiff, normalize, merge를 활용한 shadow removal 알고리즘 사용 결과

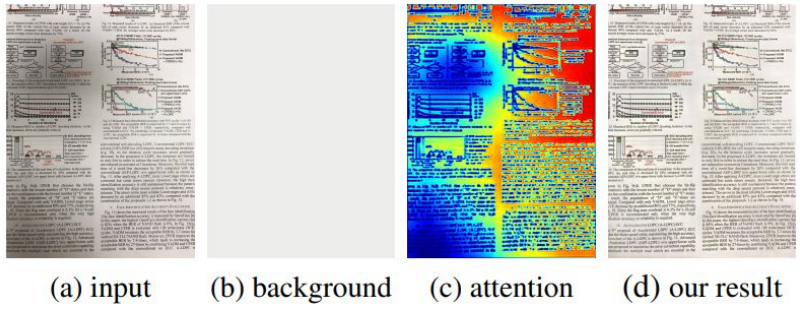


그림자가 짙을수록 그림자 제거 성능이 좋지 않았다

알고리즘 설명

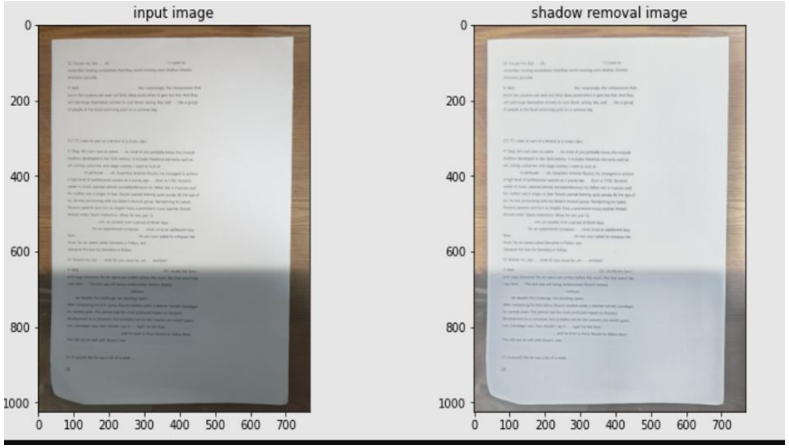
- 3. BEDSRnet 그림자 제거 알고리즘
  - the first deep network specifically designed for document image shadow removal
  - 2개의 sub-networks로 구성
    - BE-Net : 문서의 background color 예측
    - SR-Net : 그림자 제거

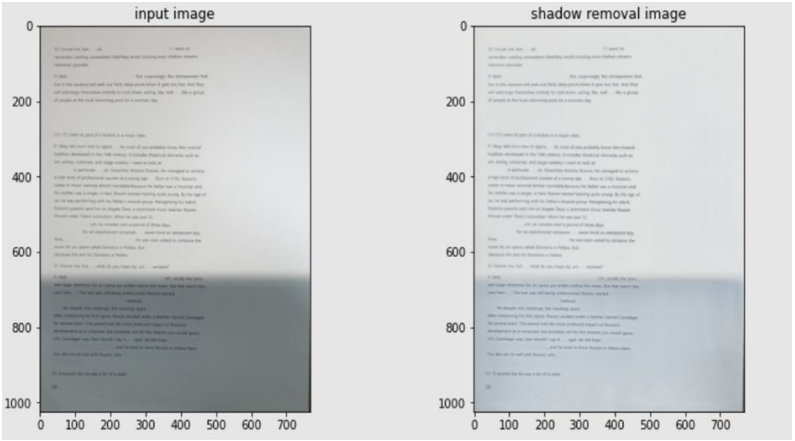




수행 결과

- BEDSR-Net 모델 테스트 결과

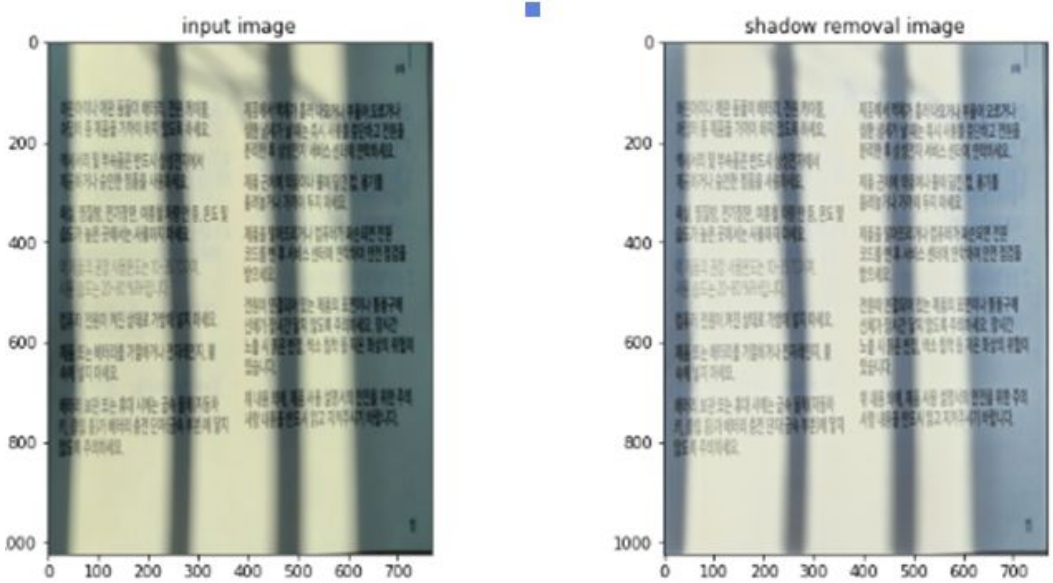




background 유무, 즉 edge의 유무와 상관없이 그림자 제거가 가능하지만
openCV알고리즘과 같은 좋은 성능을 기대하기가 어려웠다.

수행 결과

- BEDSR-Net 모델 테스트 결과



줄무늬로 그림자가 지는 경우, 이미지와 그림자를 인식하지 못해 이미지가 전체적으로 흐려지는 결과물을 얻게된다.
전체적으로 흐려지는 현상이 있고 문서의 색상이 무채색 계열이 아닌 경우 오차가 생기는 것으로 보인다.

알고리즘 설명

- ## 4. Ghost-free-shadownet 그림자 제거 알고리즘
  - Dual Hierarchical Aggregation Network and Shadow Matting GAN 을 통한 shadow generation 방식으로 제거
  - VGG19에서 image를 input으로 받고 여러 층의 Conv layer를 지나게 됨. 각 Conv layer block은 LeaklyReLU를 따르고 encoding에 down-sampling가 없어서 이미지의 세부정보를 잘 보존할 수 있음
  - 그림자를 인식하고 낮은 수준에도 그림자 이미지의 세부정보를 잘 보존하기 위해 dual hierarchical aggregations 방식으로 알고리즘을 설계함.

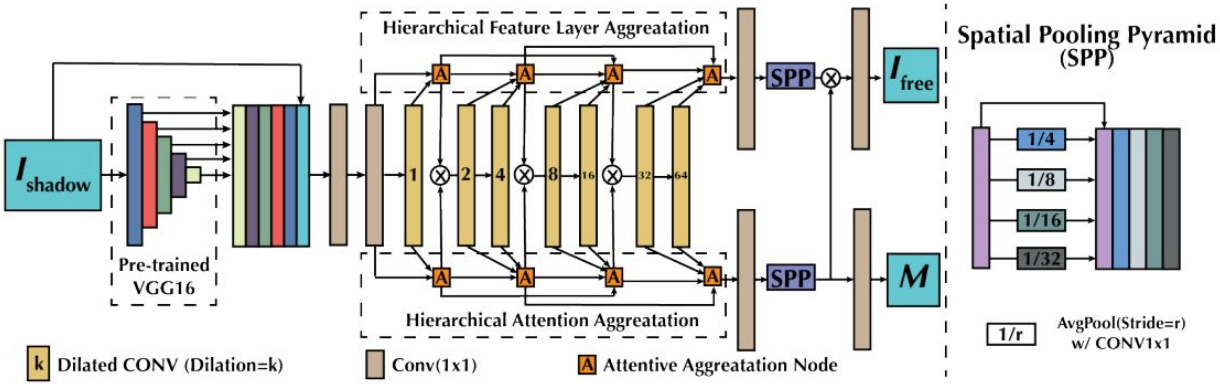
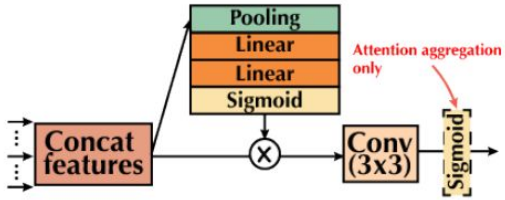


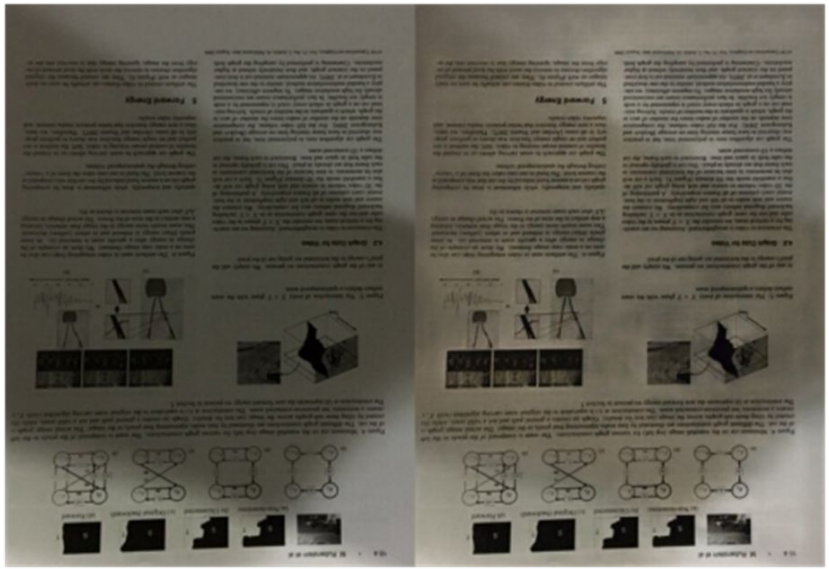
Figure 2: The network structure of the proposed Dual Hierarchical Aggregation Network.



The structure of Attentive aggregation Node

수행 결과

- Ghost-Free Shadow removal 테스트 결과

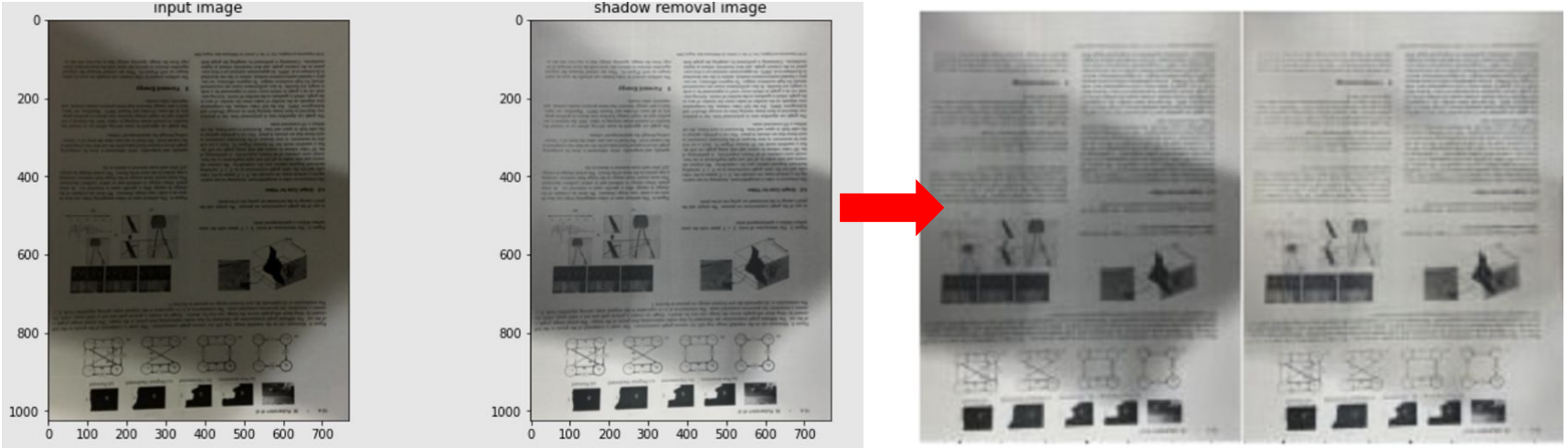


그림자 영역을 인식하여 제거하지만 완전히 그림자를 제거할 수 없었다.
또한 그림자 영역을 잘못 파악하면 그림자 일부분만 제거되는 한계점이 있었다.

수행 결과

- BEDSR-Net + Ghost-Free Shadow removal 테스트 결과

BEDSR-Net의 output을 ghost-free shadow removal의 input으로 넣은 결과. 즉 2번 그림자 제거 알고리즘을 동작할 때의 결과물



그림자 영역을 2번 지워 한 번 shadow removal 모델을 돌리는 것에 비해 좋은 성능을 보였다.
다만, 그럼에도 완벽하게 그림자가 지워지지는 않는다는 한계가 있었다.