

# 인물모드(Shallow Focus) 구현하고 개선하기

## 1. Shallow Focus 구현 [PASCAL VOC]

- 1-1. 인물 사진에 Shallow Focus 구현
- 1-2. 강아지 사진에 Shallow Focus 구현 (함수로 정리)

## 2. 문제점 찾기

- 2-1. 인물 사진 결과 확인
- 2-2. 강아지 사진 결과 확인

## 3. 개선 후 크로마키 적용

- 3-1. 개선
  - (1) 사용 모델 변경 [Ade20k] : 인물 사진 모델 변경 결정
  - (2) 사진 밝기 조절
- 3-2. 크로마키 적용

```
In [1]: import os
import urllib
import glob
import cv2
import numpy as np
from matplotlib import pyplot as plt
import matplotlib.patches as patches
%config InlineBackend.figure_format = 'retina'
```

```
In [2]: # 운영체제별 한글 폰트 설정
if os.name == 'posix': # Mac 환경 폰트 설정
    plt.rc('font', family='AppleGothic')
elif os.name == 'nt': # Windows 환경 폰트 설정
    plt.rc('font', family='Malgun Gothic')

plt.rc('axes', unicode_minus=False) # 마이너스 폰트 설정
```

```
In [3]: from pixellib.semantic import semantic_segmentation
```

```
In [4]: import warnings
warnings.filterwarnings(action='ignore')
```

## 1. Shallow Focus 구현

### 1-1. 인물 사진에 Shallow Focus 구현

```
In [3]: # 이미지 불러오기
# my_image_path = './files/human_segmentation/images/*'
my_image_path = os.getenv('HOME')+'/aiffel/human_segmentation/images/*'
```

```
image_path_list = glob.glob(my_image_path)

print(image_path_list)

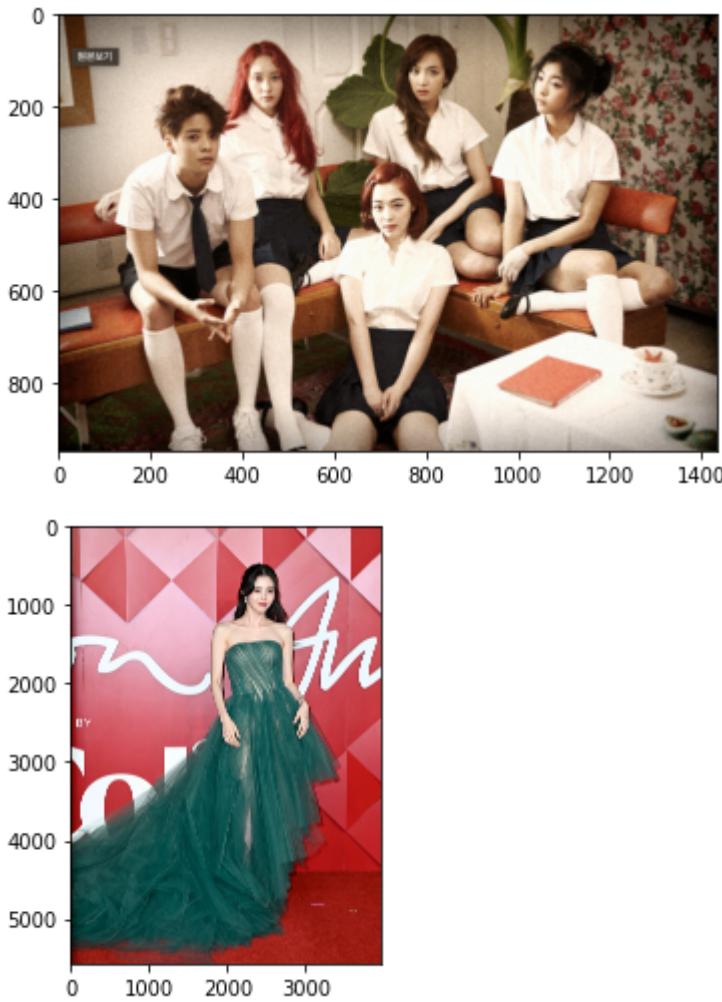
['/aiffel/aiffel/human_segmentation/images/1.jpg', '/aiffel/aiffel/human_segmentation/images/4.jpg', '/aiffel/aiffel/human_segmentation/images/3.png',
 '/aiffel/aiffel/human_segmentation/images/2.jpg']
```

In [4]: # bgr -> rgb 변환 후 image\_list에 저장  
image\_list = []

```
for image in image_path_list:
    img_bgr = cv2.imread(image)
    img_rgb = cv2.cvtColor(img_bgr, cv2.COLOR_BGR2RGB)
    image_list.append(img_bgr)

plt.imshow(img_rgb)
plt.show()
```





```
In [5]: # 모델 불러오기
model_dir = os.getenv('HOME') + '/aiffel/human_segmentation/models'
model_file = os.path.join(model_dir, 'deeplabv3_xception_tf_dim_ordering_tf_
model_url = 'https://github.com/ayoolaolafenwa/PixelLib/releases/download/1.
urlretrieve(model_url, model_file)
model = semantic_segmentation()
model.load_pascalvoc_model(model_file)
```

```
In [6]: # 사진 당 이미지 분류 결과값 저장
segvalues_list = []
output_list = []
for image_path in image_path_list:
    segvalues, output = model.segmentAsPascalvoc(image_path)
    segvalues_list.append(segvalues)
    output_list.append(output)
```

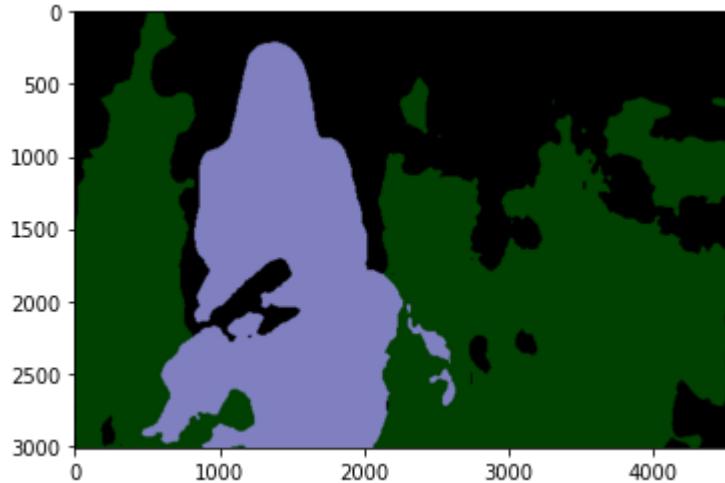
```
In [7]: #pascalvoc 데이터의 라벨종류
LABEL_NAMES = [
    'background', 'aeroplane', 'bicycle', 'bird', 'boat', 'bottle', 'bus',
    'car', 'cat', 'chair', 'cow', 'diningtable', 'dog', 'horse', 'motorbike',
    'person', 'pottedplant', 'sheep', 'sofa', 'train', 'tv'
]
len(LABEL_NAMES)
```

Out[7]: 21

```
In [8]: #segmentAsPascalvoc() 함수를 호출하여 입력된 이미지를 분할한 뒤 나온 결과값 중 output을 n
# 함수를 호출하여 입력된 이미지를 분할한 뒤 나온 결과값 중 배열값을 출력
for i in range(len(image_list)):
    print('===={}===='.format(i))
```

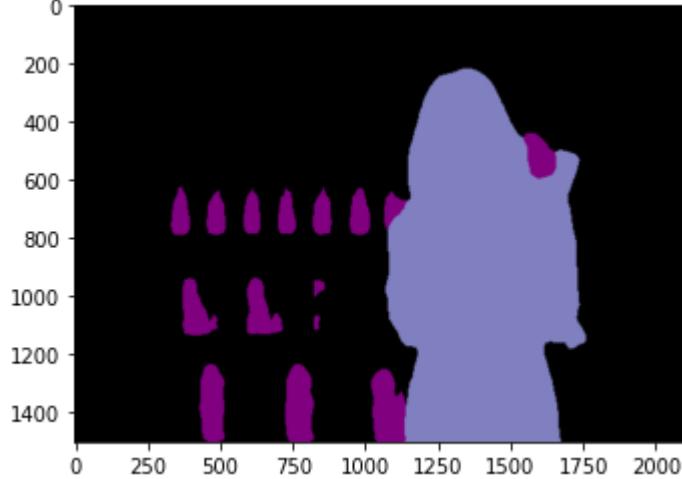
```
plt.imshow(output_list[i])
plt.show()
print(segvalues_list[i])
```

=====0=====



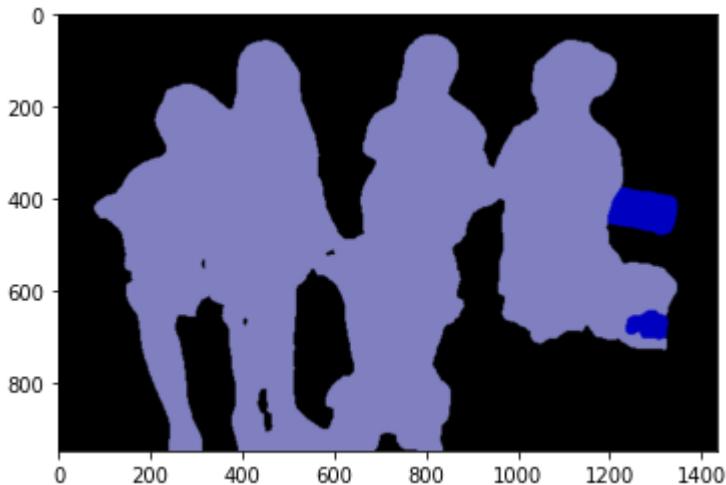
```
{'class_ids': array([ 0, 15, 16]), 'masks': array([[False, False, False,
..., False, False, False],
       [False, False, False, ..., False, False, False],
       [False, False, False, ..., False, False, False],
       ...,
       [False, False, False, ..., False, False, False],
       [False, False, False, ..., False, False, False],
       [False, False, False, ..., False, False, False]])}
```

=====1=====

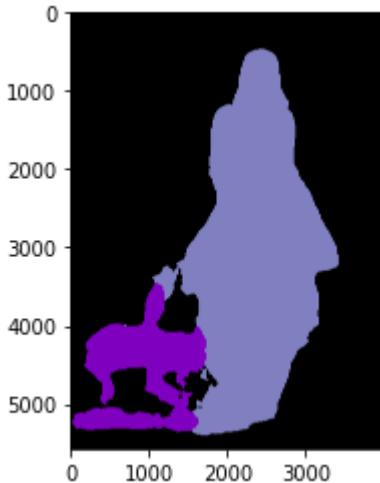


```
{'class_ids': array([ 0,  5, 15]), 'masks': array([[False, False, False,
..., False, False, False],
       [False, False, False, ..., False, False, False],
       [False, False, False, ..., False, False, False],
       ...,
       [False, False, False, ..., False, False, False],
       [False, False, False, ..., False, False, False],
       [False, False, False, ..., False, False, False]])}
```

=====2=====



```
{'class_ids': array([ 0,  9, 15]), 'masks': array([[False, False, False,
..., False, False, False],
[False, False, False, ..., False, False, False],
[False, False, False, ..., False, False, False],
...,
[False, False, False, ..., False, False, False],
[False, False, False, ..., False, False, False],
[False, False, False, ..., False, False, False]])}
=====3=====
```



```
{'class_ids': array([ 0, 13, 15]), 'masks': array([[False, False, False,
..., False, False, False],
[False, False, False, ..., False, False, False],
[False, False, False, ..., False, False, False],
...,
[False, False, False, ..., False, False, False],
[False, False, False, ..., False, False, False],
[False, False, False, ..., False, False, False]])}
```

In [9]: # segvalues에 있는 class\_ids를 담고있는 값을 통해 pacalvoc에 담고있는 라벨을 출력

```
for i in range(len(image_list)):
    print('====={}===='.format(i))
    for class_id in segvalues_list[i]['class_ids']:
        print(LABEL_NAMES[class_id])
```

```
=====0=====
background
person
pottedplant
=====1=====
background
bottle
person
=====2=====
background
chair
person
=====3=====
background
horse
person
```

```
In [10]: # 컬러맵 만들기 (출처 : pixellib)
colormap = np.zeros((256, 3), dtype = int)
ind = np.arange(256, dtype=int)

for shift in reversed(range(8)):
    for channel in range(3):
        colormap[:, channel] |= ((ind >> channel) & 1) << shift
    ind >>= 3

colormap[:20]
```

```
Out[10]: array([[ 0,   0,   0],
       [128,   0,   0],
       [ 0, 128,   0],
       [128, 128,   0],
       [ 0,   0, 128],
       [128,   0, 128],
       [ 0, 128, 128],
       [128, 128, 128],
       [ 64,   0,   0],
       [192,   0,   0],
       [ 64, 128,   0],
       [192, 128,   0],
       [ 64,   0, 128],
       [192,   0, 128],
       [ 64, 128, 128],
       [192, 128, 128],
       [ 0,  64,   0],
       [128,  64,   0],
       [ 0, 192,   0],
       [128, 192,   0]])
```

```
In [11]: colormap[15] #컬러맵 15에 해당하는 배열 출력 (pacalvoc0// LABEL_NAMES 15번째인 사람)
# array([192, 128, 128])
```

```
Out[11]: array([192, 128, 128])
```

```
In [12]: # bgr -> rgb
seg_color = colormap[15][2::-1]
# seg_color = (128,128,192)
```

```
In [13]: seg_color
```

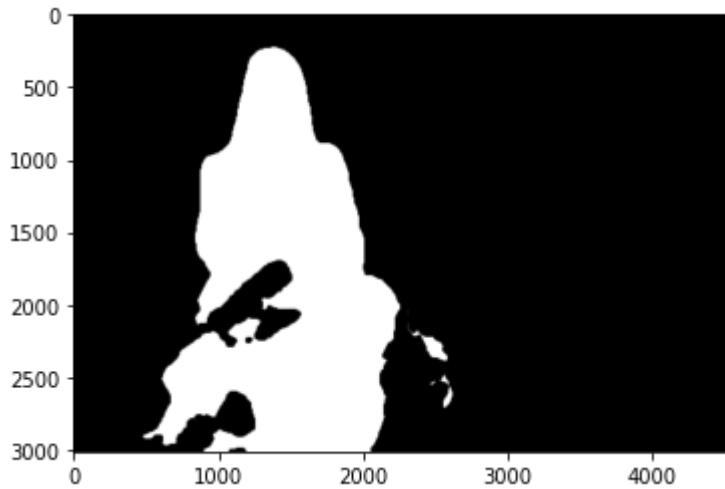
```
Out[13]: array([128, 128, 192])
```

```
In [14]: # seg_color로 이루어진 마스크 만들기
# output의 픽셀 별로 색상이 seg_color와 같다면 1(True), 다르다면 0(False)이 됩니다
# seg_color 값이 person을 값이므로 사람이 있는 위치를 제외하고는 gray로 출력
# cmap 값을 변경하면 다른 색상으로 확인이 가능함
seg_map_list = []
```

```
for i in range(len(image_list)):
    output = output_list[i]
    seg_map = np.all(output==seg_color, axis=-1)
    seg_map_list.append(seg_map)
    print('====={}===='.format(i))
    print(seg_map.shape)
    plt.imshow(seg_map, cmap='gray')
    plt.show()
```

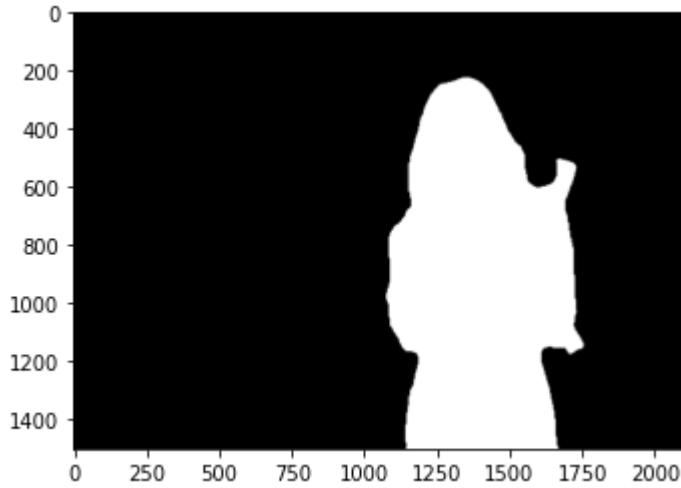
=====0=====

(3012, 4518)



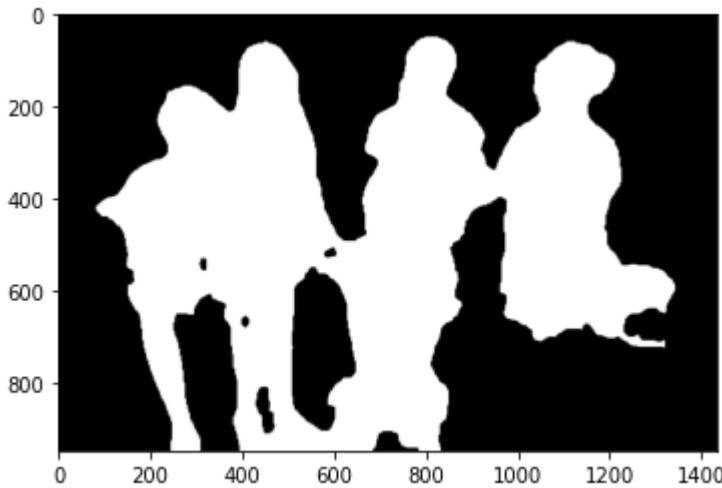
=====1=====

(1500, 2100)



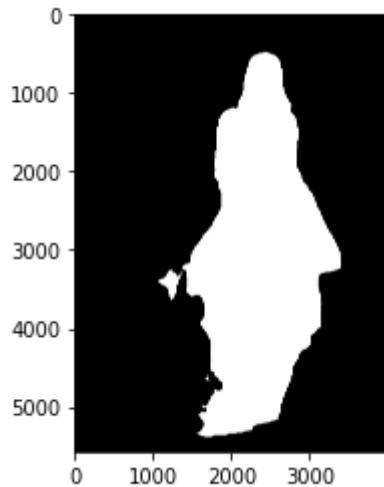
=====2=====

(948, 1434)



=====3=====

(5570, 3966)



```
In [15]: # 원본이미지를 img_show에 할당한뒤 이미지 사람이 있는 위치와 배경을 분리해서 표현한 color_mask
img_show_list = image_list.copy()

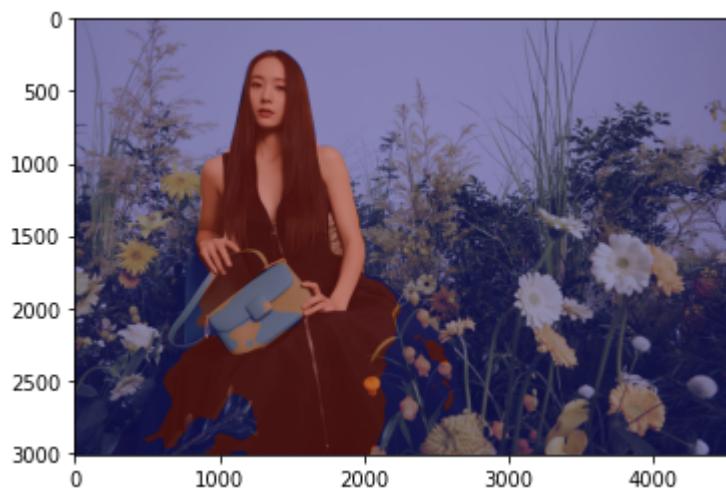
img_mask_list = []

for i in range(len(image_list)):
    seg_map = seg_map_list[i]
    img_show = img_show_list[i]

    # True과 False인 값을 각각 255과 0으로 바꿔줍니다
    img_mask = seg_map.astype(np.uint8) * 255
    img_mask_list.append(img_mask)
    # 255와 0을 적당한 색상으로 바꿔봅니다
    color_mask = cv2.applyColorMap(img_mask, cv2.COLORMAP_JET)
    # 원본 이미지와 마스크를 적당히 합쳐봅니다
    # 0.6과 0.4는 두 이미지를 섞는 비율입니다.
    img_show = cv2.addWeighted(img_show, 0.6, color_mask, 0.4, 0.0)

    print('====={}====='.format(i))
    plt.imshow(cv2.cvtColor(img_show, cv2.COLOR_BGR2RGB))
    plt.show()
```

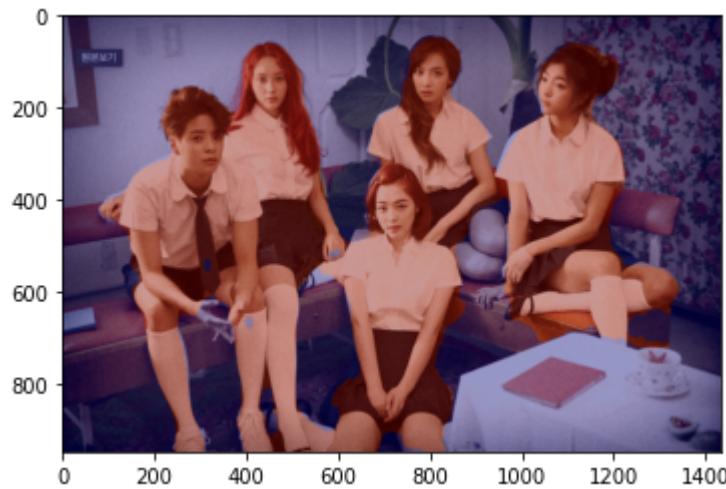
=====0=====



=====1=====



=====2=====

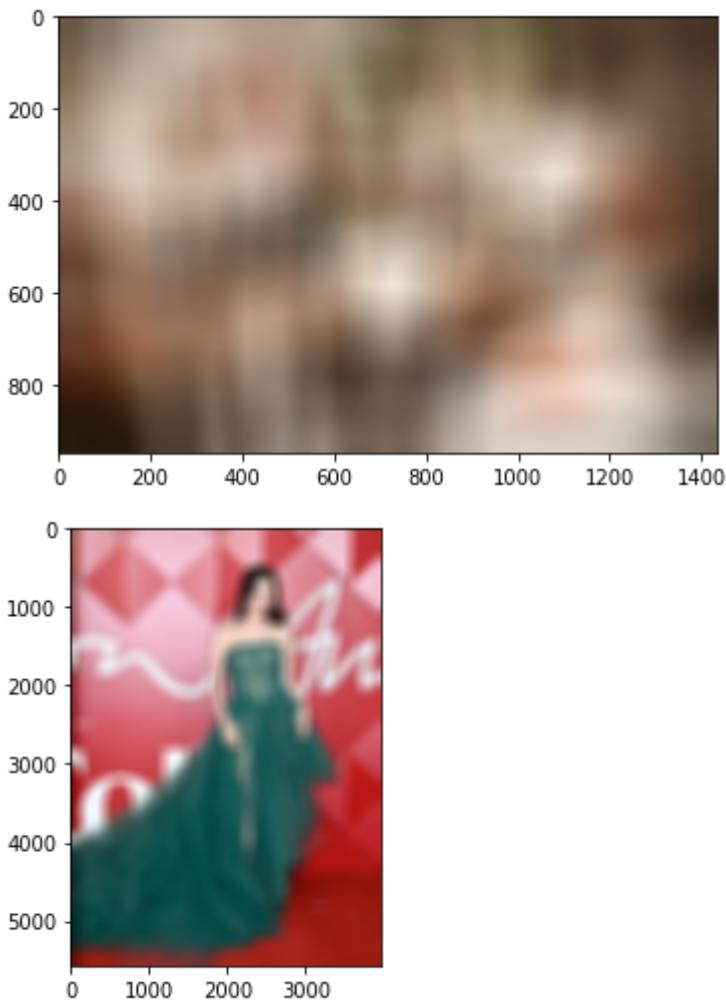


=====3=====



```
In [16]: # 원본 사진 블러처리 후 저장  
img_orig.blur_list = []  
  
for i in range(len(image_list)):  
    img_orig = image_list[i]  
  
    img_orig.blur = cv2.blur(img_orig, (200,200)) # (200,200) : blurring kernel  
    img_orig.blur_list.append(img_orig.blur)  
  
    plt.imshow(cv2.cvtColor(img_orig.blur, cv2.COLOR_BGR2RGB))  
    plt.show()
```



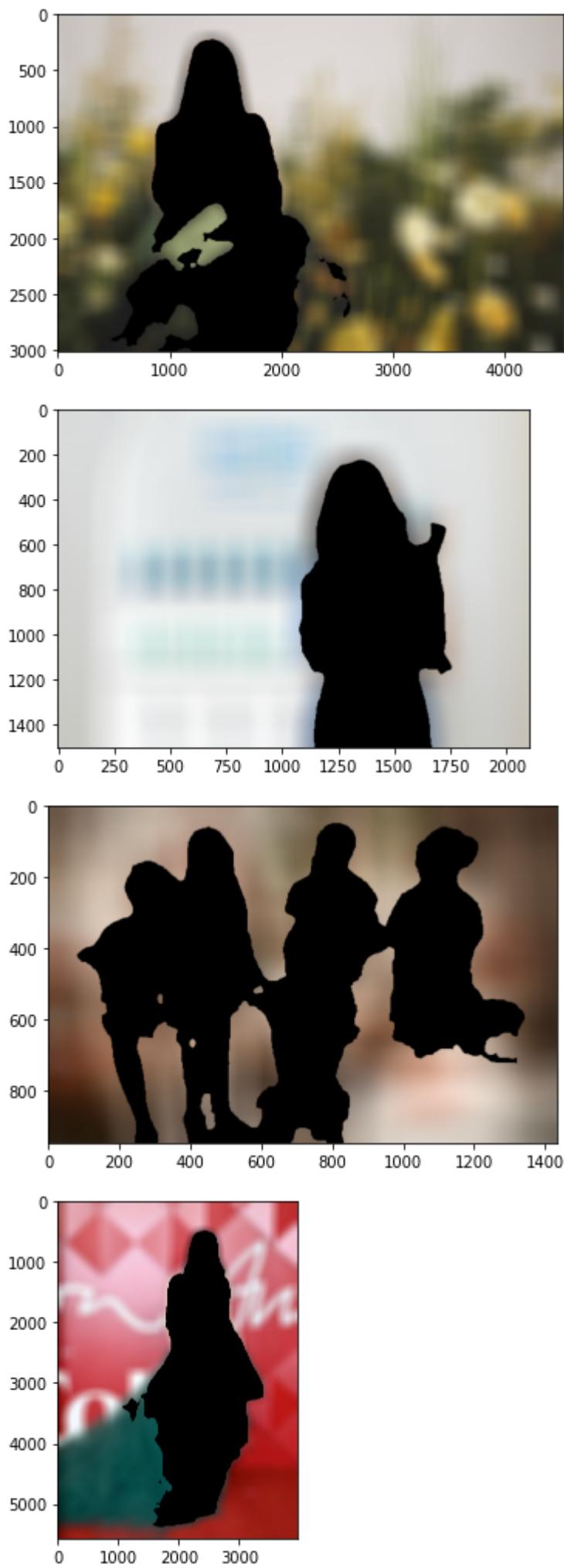


```
In [17]: # 배경 추출
img_bg_blur_list = []
img_mask_color_list = []

for i in range(len(image_list)):
    img_mask = img_mask_list[i]
    img_orig.blur = img_orig.blur_list[i]

    img_mask_color = cv2.cvtColor(img_mask, cv2.COLOR_GRAY2BGR)
    img_mask_color_list.append(img_mask_color)
    # cv2.bitwise_not(): 이미지가 반전 -> 배경: 255, 사람: 0 으로 만들
    img_bg_mask = cv2.bitwise_not(img_mask_color)
    # cv2.bitwise_and()을 사용하면 배경만 있는 영상을 얻을 수 있습니다.
    # 0과 어떤 수를 bitwise_and 연산을 해도 0이 되기 때문에
    # 사람이 0인 경우에는 사람이 있던 모든 픽셀이 0이 됩니다. 결국 사람이 사라지고 배경만 남아요.
    img_bg.blur = cv2.bitwise_and(img_orig.blur, img_bg_mask)
    img_bg.blur_list.append(img_bg.blur)

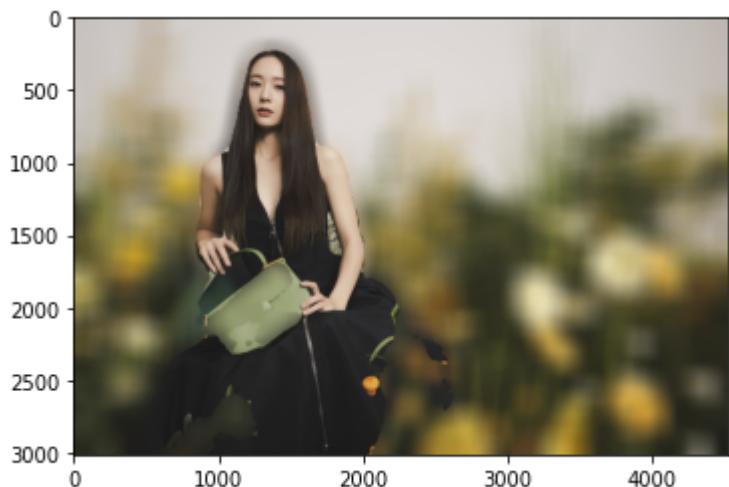
plt.imshow(cv2.cvtColor(img_bg.blur, cv2.COLOR_BGR2RGB))
plt.show()
```



```
In [18]: # 블러 배경과 사람 합치기
# np.where(조건, 참일때, 거짓일때)
# 세그멘테이션 마스크가 255인 부분만 원본 이미지 값을 가지고 오고
# 아닌 영역은 블러된 이미지 값을 사용합니다.
for i in range(len(image_list)):
    img_orig = image_list[i]
    img_bg_blur = img_bg_blur_list[i]
    img_mask_color = img_mask_color_list[i]

    img_concat = np.where(img_mask_color==255, img_orig, img_bg_blur)
    img_concat = cv2.cvtColor(img_concat, cv2.COLOR_BGR2RGB)

    plt.imshow(img_concat)
    plt.show()
    save_path = os.getenv('HOME')+'/aiffel/human_segmentation/' # 확인
    save_folder = 'image_results'
    cv2.imwrite('{}/{}/result_{}.jpg'.format(save_path, save_folder, i+1), img)
    print('{}번째 사진 변환 및 저장 완료'.format(i+1))
```



1번째 사진 변환 및 저장 완료



2번째 사진 변환 및 저장 완료



3번째 사진 변환 및 저장 완료



4번째 사진 변환 및 저장 완료

## 1-2. 강아지 사진에 Shallow Focus 구현

- 강아지 사진은 함수를 만들어서 구현

pascalvoc 데이터의 라벨종류 (LABEL\_NAMES[dog]==12)

- LABEL\_NAMES
  - 'background',
  - 'aeroplane', 'bicycle', 'bird', 'boat', 'bottle',
  - 'bus', 'car', 'cat', 'chair', 'cow',
  - 'diningtable', 'dog', 'horse', 'motorbike', 'person',
  - 'pottedplant', 'sheep', 'sofa', 'train', 'tv'

```
In [30]: # 컬러맵 만들기 (출처 : pixellib)
colormap = np.zeros((256, 3), dtype = int)
ind = np.arange(256, dtype=int)

for shift in reversed(range(8)):
    for channel in range(3):
        colormap[:, channel] |= ((ind >> channel) & 1) << shift
    ind >>= 3
```

```
In [31]: # (12:dog) bgr -> rgb
dog_seg_color = colormap[12][2::-1]
dog_seg_color
```

```
Out[31]: array([128, 0, 64])
```

```
In [21]: # 이미지 불러오기
# my_image_path = './files/human_segmentation/dog_images/*'
dog_image_path = os.getenv('HOME')+'/aiffel/human_segmentation/dog_images/*'
dog_image_path_list = glob.glob(dog_image_path)
dog_image_path_list
```

```
Out[21]: ['/aiffel/aiffel/human_segmentation/dog_images/3.jpeg',
 '/aiffel/aiffel/human_segmentation/dog_images/1.jpg',
 '/aiffel/aiffel/human_segmentation/dog_images/2.jpg']
```

```
In [55]: # image_list에 저장
def save_image_list(image_path_list):
    image_list = []

    for image in image_path_list:
        img_bgr = cv2.imread(image)
        img_rgb = cv2.cvtColor(img_bgr, cv2.COLOR_BGR2RGB)
        image_list.append(img_bgr)

        plt.imshow(img_rgb)
        plt.show()

    return image_list

# 사진 당 이미지 분류 결과값 저장
def save_model_output(image_path_list, image_list):
    # 모델 불러오기
    model_dir = os.getenv('HOME')+'/aiffel/human_segmentation/models'
    model_file = os.path.join(model_dir, 'deeplabv3_xception_tf_dim_ordering_tf_keras.h5')
    model_url = 'https://github.com/ayoolaolafenwa/PixelLib/releases/download/1.1.0/deeplabv3_xception_tf_dim_ordering_tf_keras.h5'
    urllib.request.urlretrieve(model_url, model_file)
    model = semantic_segmentation()
    model.load_pascalvoc_model(model_file)

    segvalues_list = []
    output_list = []

    for image_path in image_path_list:
        segvalues, output = model.segmentAsPascalvoc(image_path)
        segvalues_list.append(segvalues)
        output_list.append(output)

    # pascalvoc 데이터의 라벨종류
    LABEL_NAMES = [
        'background', 'aeroplane', 'bicycle', 'bird', 'boat', 'bottle', 'bus',
        'car', 'cat', 'chair', 'cow', 'diningtable', 'dog', 'horse', 'motorbike',
        'person', 'pottedplant', 'sheep', 'sofa', 'train', 'tv'
    ]
    # segvalues에 있는 class_ids를 담고있는 값을 통해 pascalvoc에 담고있는 라벨을 출력
    print('-----LABEL_NAMES-----')
    for i in range(len(image_list)):
        print('====={}===='.format(i))
        for class_id in segvalues_list[i]['class_ids']:
            print(LABEL_NAMES[class_id])
    print('-----')
```

```

    return segvalues_list, output_list

# set_color로 이루어진 마스크 생성
def make_seg_mask(image_list, output_list, seg_color):
    img_mask_list = []
    # img_show_list = image_list.copy()

    for i in range(len(image_list)):
        output = output_list[i]
        seg_map = np.all(output==seg_color, axis=-1)
        # True과 False인 값을 각각 255과 0으로 바꿔줍니다
        img_mask = seg_map.astype(np.uint8) * 255
        img_mask_list.append(img_mask)

    return img_mask_list

# 원본 사진 블러 처리 -> 배경 추출 -> 블러 배경과 타겟 합치기
def concat_w_blur(image_list, img_mask_list, save_folder=None):
    # 원본 사진 블러 처리 후 저장
    img_orig.blur_list = []

    for i in range(len(image_list)):
        img_orig = image_list[i]

        img_orig.blur = cv2.blur(img_orig, (200,200)) # (200,200) : blurring
        img_orig.blur_list.append(img_orig.blur)

    # 배경 추출
    img_bg.blur_list = []
    img_mask_color_list = []

    for i in range(len(image_list)):
        img_mask = img_mask_list[i]
        img_orig.blur = img_orig.blur_list[i]

        img_mask_color = cv2.cvtColor(img_mask, cv2.COLOR_GRAY2BGR)
        img_mask_color_list.append(img_mask_color)
        # cv2.bitwise_not(): 이미지가 반전 -> 배경 : 255, 사람: 0 으로 만들
        img_bg.mask = cv2.bitwise_not(img_mask_color)
        # cv2.bitwise_and()을 사용하면 배경만 있는 영상을 얻을 수 있습니다.
        # 0과 어떤 수를 bitwise_and 연산을 해도 0이 되기 때문에
        # 사람이 0인 경우에는 사람이 있던 모든 픽셀이 0이 됩니다. 결국 사람이 사라지고 배경만 남
        img_bg.blur = cv2.bitwise_and(img_orig.blur, img_bg.mask)
        img_bg.blur_list.append(img_bg.blur)

    # 블러 배경과 타겟 합치기

    for i in range(len(image_list)):
        img_orig = image_list[i]
        img_bg.blur = img_bg.blur_list[i]
        img_mask_color = img_mask_color_list[i]

        img_concat = np.where(img_mask_color==255, img_orig, img_bg.blur)
        img_concat = cv2.cvtColor(img_concat, cv2.COLOR_BGR2RGB)

    # 저장
    plt.figure(figsize=(20, 20))
    plt.imshow(img_concat)
    plt.show()
    # save_path = os.getenv('HOME')+'/aiffel/human_segmentation/' # 저장
    # cv2.imwrite('{}_{}.jpg'.format(save_path, save_folder, i+1)
    print('{})번째 사진 변환 및 저장 완료'.format(i+1))

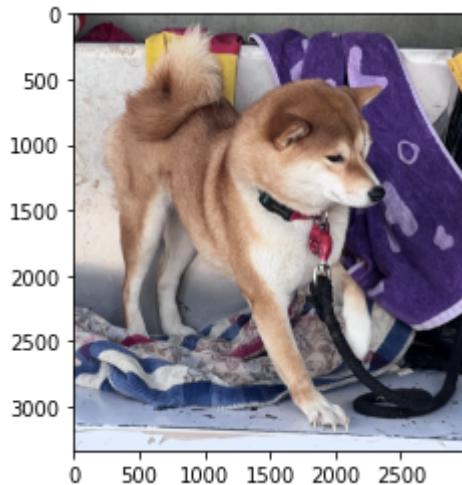
```

```
print('='*70)
print('필터 적용 및 저장 완료')

return None

# 전체 실행
def make_bg_blur(image_path_list, seg_color, save_folder):
    def_image_list = save_image_list(image_path_list)
    def_segvalues_list, def_output_list = save_model_output(image_path_list,
    def_img_mask_list = make_seg_mask(def_image_list, def_output_list, seg_c
concat_w.blur(def_image_list, def_img_mask_list, save_folder)
```

```
In [23]: save_folder = 'dog_results'
dog_image_gb_blur_ = make_bg_blur(dog_image_path_list, dog_seg_color)
dog_image_gb_blur_
```





-----LABEL NAMES-----

====0====

background

dog

====1====

background

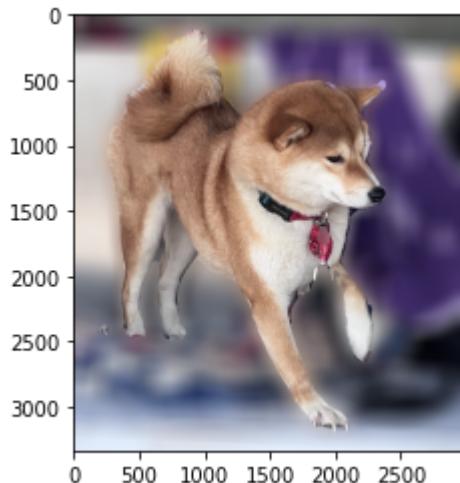
dog

====2====

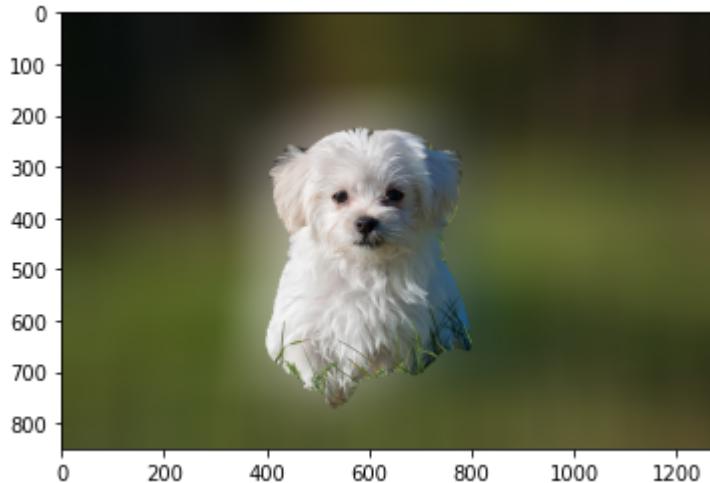
background

dog

person



1번째 사진 변환 및 저장 완료



2번째 사진 변환 및 저장 완료



3번째 사진 변환 및 저장 완료

=====

필터 적용 및 저장 완료

---

## 2. 문제점 찾기

- 표시한 이미지들을 jupyter notebook에 포함
- 해결 방안 제시
  - DeepLab 모델의 Semantic Segmentation이 만들어 낸 Mask 영역에 어떻게 적용되어 문제점을 보완하게 되는지의 메커니즘이 포함된 솔루션

### 2-1. 인물 사진 결과 확인

```
In [6]: # 이미지 불러오기
my_image_path = './files/human_segmentation/image_results/*'
# my_image_path = os.getenv('HOME')+'/aiffel/human_segmentation/image_result'
image_path_list = glob.glob(my_image_path)

# bgr -> rgb 변환 후 image_list에 저장
image_list = []

for image in image_path_list:
    img_bgr = cv2.imread(image)
    img_rgb = cv2.cvtColor(img_bgr, cv2.COLOR_BGR2RGB)
    image_list.append(img_rgb)
```

```
In [70]: plt.clf()
fig = plt.figure(figsize=(20, 20))
ax = fig.add_subplot(1, 1, 1)

# 사각형 그리기
ax.add_patch(patches.Rectangle((1800, 1600), 500, 800, edgecolor = 'red', fill=False))
ax.add_patch(patches.Rectangle((1050, 3200), 300, 500, edgecolor = 'red', fill=False))

# 텍스트 삽입하기
ax.text(1800, 2500, '팔과 허리 사이 배경 인식 안됨', fontsize=20)
ax.text(1050, 3800, '드레스 블러처리 안됨', fontsize=20)
ax.text(3000, 1000, '드레소히 진짜 공주님 그 잡채', fontsize=10)

ax.imshow(image_list[0])
plt.show()
```

&lt;Figure size 640x480 with 0 Axes&gt;



In [69]:

```
plt.clf()
fig = plt.figure(figsize=(20, 20))
ax = fig.add_subplot(1, 1, 1)

# 사각형 그리기
ax.add_patch(patches.Rectangle((1700, 1300), 200, 500, edgecolor = 'red', f
ax.add_patch(patches.Rectangle((1000, 1800), 700, 600, edgecolor = 'red', f
ax.add_patch(patches.Rectangle((800, 2500), 500, 400, edgecolor = 'red', fi
ax.add_patch(patches.Rectangle((2000, 1800), 800, 1000, edgecolor = 'red', f

# 텍스트 삽입하기
```

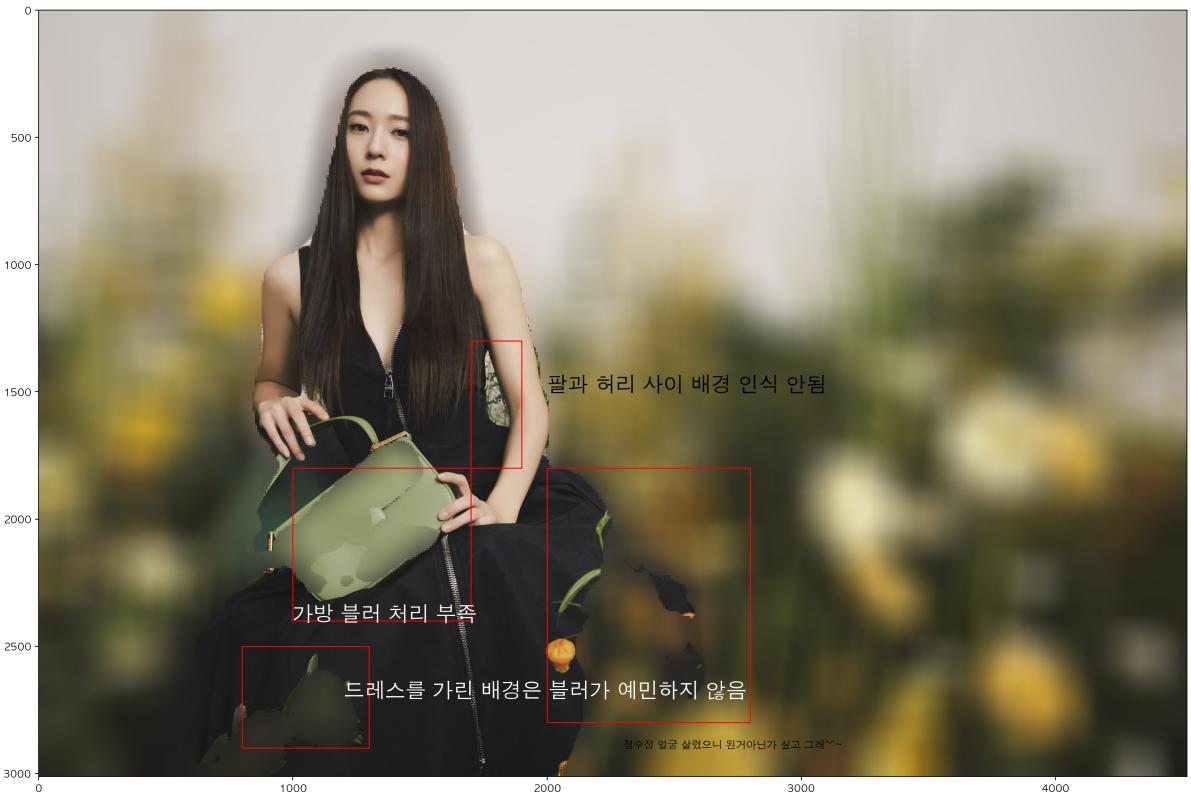
```

ax.text(2000, 1500, '팔과 허리 사이 배경 인식 안됨', fontsize=20)
ax.text(1000, 2400, '가방 블러 처리 부족', color='white', fontsize=20)
ax.text(1200, 2700, '드레스를 가린 배경은 블러가 예민하지 않음', color='white', fontsi
ax.text(2300, 2900, '정수점 얼굴 살렸으니 된거아닌가 싶고 그래^^~', fontsize=10)

ax.imshow(image_list[1])
plt.show()

```

&lt;Figure size 640x480 with 0 Axes&gt;



In [66]:

```

plt.clf()
fig = plt.figure(figsize=(20, 20))
ax = fig.add_subplot(1, 1, 1)

# 사각형 그리기
ax.add_patch(patches.Rectangle((1500, 430), 300, 150, edgecolor = 'red', fi

# 텍스트 삽입하기
ax.text(1600, 400, '손이 과하게 블러 됨', fontsize=20)
ax.text(1600, 420, '소하짱이랑 아이컨택하느라 그런거같지요 아마두요 ㅋㅋ', fontsize=10)

ax.imshow(image_list[2])
plt.show()

```

&lt;Figure size 640x480 with 0 Axes&gt;



```
In [62]: plt.clf()
fig = plt.figure(figsize=(20, 20))
ax = fig.add_subplot(1, 1, 1)

ax.text(900, 900, '사람이 여려명으로 많아지면 다음과 같이 인식오류가 많아짐', fontsize=15)
ax.text(900, 930, '이 정도면..에프엑스한테 반해서 정신 못차리는거다', fontsize=10)

ax.imshow(image_list[3])
plt.show()
```

<Figure size 640x480 with 0 Axes>



## 2-2. 강아지 사진 결과 확인

```
In [35]: # 이미지 불러오기
dog_image_path = './files/human_segmentation/dog_results/*'
# dog_image_path = os.getenv('HOME') + '/aiffel/human_segmentation/dog_results'
dog_image_path_list = glob.glob(dog_image_path)

# bgr -> rgb 변환 후 image_list에 저장
dog_image_list = []

for image in dog_image_path_list:
    img_bgr = cv2.imread(image)
    img_rgb = cv2.cvtColor(img_bgr, cv2.COLOR_BGR2RGB)
    dog_image_list.append(img_rgb)
```

```
In [71]: plt.clf()
fig = plt.figure(figsize=(20, 20))
ax = fig.add_subplot(1, 1, 1)

# 사각형 그리기
ax.add_patch(patches.Rectangle((190, 2300), 100, 200, edgecolor = 'red', fill=False))
ax.add_patch(patches.Rectangle((400, 2400), 600, 200, edgecolor = 'red', fill=False))
ax.add_patch(patches.Rectangle((2200, 2500), 200, 400, edgecolor = 'red', fill=False))

# 텍스트 삽입하기
ax.text(300, 3000, '다리 부분이 애매하게 블러가 되었지만\n이게 요놈 다리인걸요\n잘생긴 얼굴은')

ax.imshow(dog_image_list[0])
plt.show()
```

<Figure size 640x480 with 0 Axes>



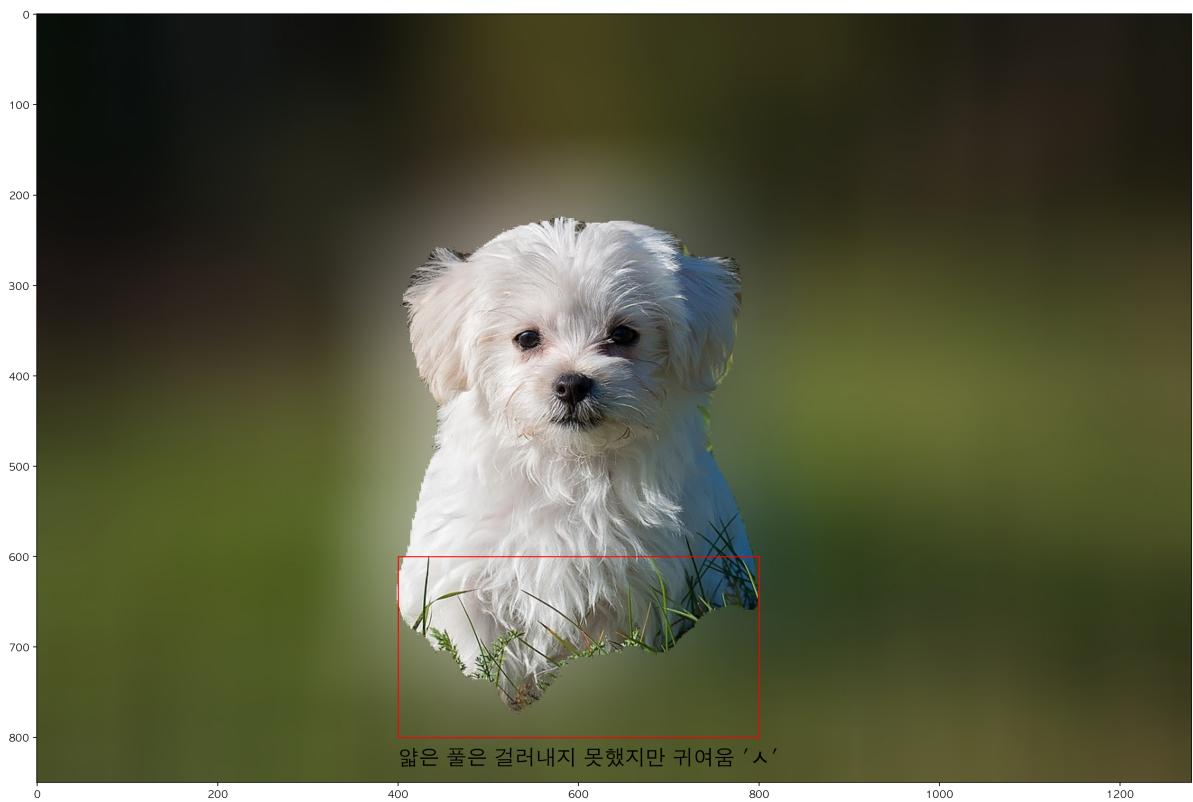
```
In [60]: plt.clf()
fig = plt.figure(figsize=(20, 20))
ax = fig.add_subplot(1, 1, 1)

# 사각형 그리기
ax.add_patch(patches.Rectangle((400, 600), 400, 200, edgecolor = 'red', fill = False))

# 텍스트 삽입하기
ax.text(400, 830, "얇은 풀은 걸러내지 못했지만 귀여움 'ㅅ'", fontsize=20)

ax.imshow(dog_image_list[1])
plt.show()
```

<Figure size 640x480 with 0 Axes>



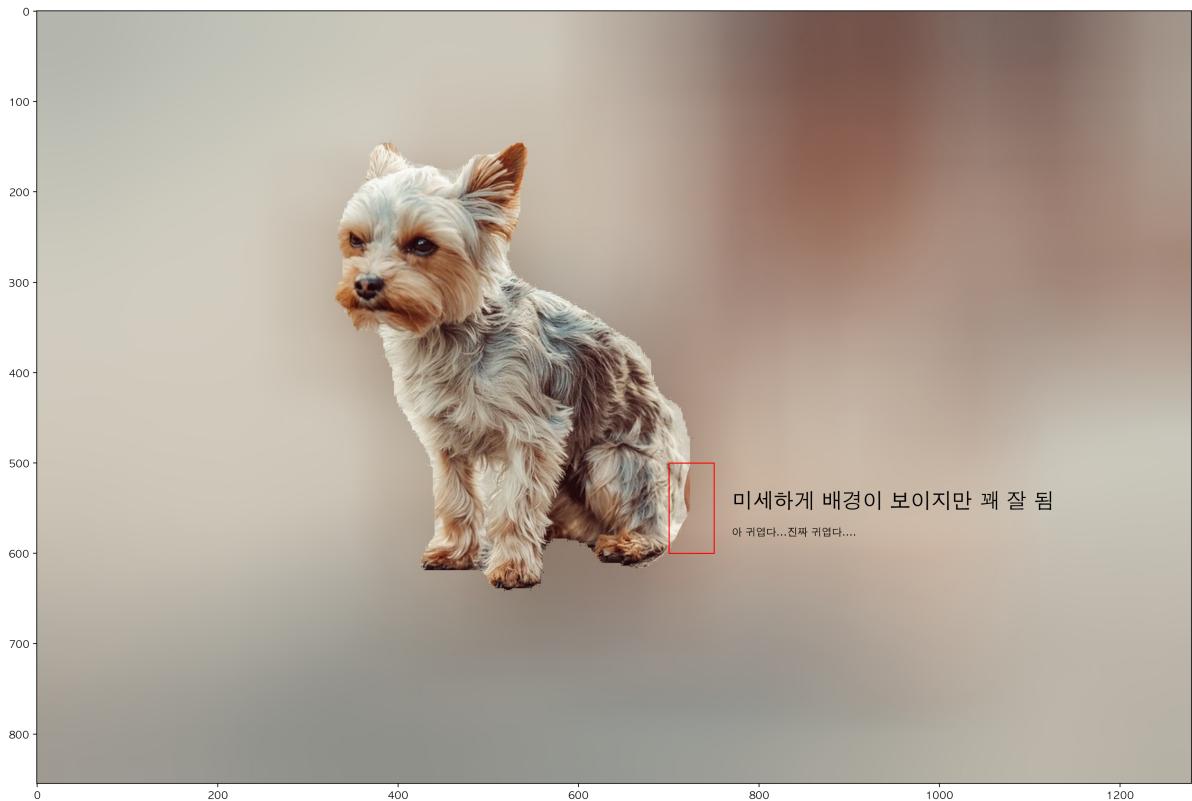
```
In [61]: plt.clf()
fig = plt.figure(figsize=(20, 20))
ax = fig.add_subplot(1, 1, 1)

# 사각형 그리기
ax.add_patch(patches.Rectangle((700, 500), 50, 100, edgecolor = 'red', fill=False))

# 텍스트 삽입하기
ax.text(770, 550, '미세하게 배경이 보이지만 꽤 잘 됨', fontsize=20)
ax.text(770, 580, '아 귀엽다...진짜 귀엽다....', fontsize=10)

ax.imshow(dog_image_list[2])
plt.show()

<Figure size 640x480 with 0 Axes>
```



### 3. 개선 후 크로마키 적용

#### 3-1. 개선

##### (1) 사용 모델 변경(Ade20k)

```
In [41]: # image_list에 저장
def save_image_list(image_path_list):
    image_list = []

    for image in image_path_list:
        img_bgr = cv2.imread(image)
        img_rgb = cv2.cvtColor(img_bgr, cv2.COLOR_BGR2RGB)
        image_list.append(img_bgr)

    #plt.imshow(img_rgb)
    #plt.show()

    return image_list

# 사진 당 이미지 분류 결과값 저장
# model 변경
def save_model_output(image_path_list, image_list):
    # 모델 불러오기
    model_dir = os.getenv('HOME') + '/aiffel/human_segmentation/models'
    model_file = os.path.join(model_dir, 'deeplabv3_xception65_ade20k.h5')
    model_url = 'https://github.com/ayoolaolafenwa/PixelLib/releases/download'
    urllib.request.urlretrieve(model_url, model_file)
    model = semantic_segmentation()
    model.load_ade20k_model(model_file)

    segvalues_list = []
    output_list = []
```

```

for image_path in image_path_list:
    segvalues, output = model.segmentAsAde20k(image_path)
    segvalues_list.append(segvalues)
    output_list.append(output)

#     # Ade20k 데이터의 라벨 (모델 테스트로 돌려보면서 확인함)
#     # person : 13
#     # animal : 127

    # segvalues에 있는 class_names을 통해 Ade20k에 담겨있는 라벨을 출력
    print('-----LABEL NAMES-----')
    for i in range(len(image_list)):
        print('====={}===='.format(i))
        print(segvalues_list[i]['class_names'])
    print('-----')

return segvalues_list, output_list

# set_color로 이루어진 마스크 생성
def make_seg_mask(image_list, output_list, segvalues_list, label_num):
    img_mask_list = []
    # img_show_list = image_list.copy()

    for i in range(len(image_list)):
        output = output_list[i]
        segvalues = segvalues_list[i]
        seg_color = segvalues['class_colors'][segvalues['class_ids'].index(1)]
        seg_color = seg_color[2::-1]      # bgr -> rgb
        seg_map = np.all(output==seg_color, axis=-1)
        # True과 False인 값을 각각 255과 0으로 바꿔줍니다
        img_mask = seg_map.astype(np.uint8) * 255
        img_mask_list.append(img_mask)

    return img_mask_list

# 원본 사진 블러 처리 -> 배경 추출 -> 블러 배경과 타겟 합치기
def concat_w_blur(image_list, img_mask_list, save_folder=None):
    # 원본 사진 블러 처리 후 저장
    img_orig.blur_list = []

    for i in range(len(image_list)):
        img_orig = image_list[i]

        img_orig.blur = cv2.blur(img_orig, (200,200)) # (200,200) : blurring
        img_orig.blur_list.append(img_orig.blur)

    # 배경 추출
    img_bg.blur_list = []
    img_mask_color_list = []

    for i in range(len(image_list)):
        img_mask = img_mask_list[i]
        img_orig.blur = img_orig.blur_list[i]

        img_mask_color = cv2.cvtColor(img_mask, cv2.COLOR_GRAY2BGR)
        img_mask_color_list.append(img_mask_color)
        # cv2.bitwise_not(): 이미지가 반전 -> 배경 : 255, 사람: 0 으로 만듦
        img_bg.mask = cv2.bitwise_not(img_mask_color)
        # cv2.bitwise_and()을 사용하면 배경만 있는 영상을 얻을 수 있습니다.
        # 0과 어떤 수를 bitwise_and 연산을 해도 0이 되기 때문에

```

```

# 사람이 0인 경우에는 사람이 있던 모든 픽셀이 0이 됩니다. 결국 사람이 사라지고 배경만 남습니다.
img_bg_blur = cv2.bitwise_and(img_orig Blur, img_bg_mask)
img_bg_blur_list.append(img_bg_blur)

# 블러 배경과 타겟 합치기

for i in range(len(image_list)):
    img_orig = image_list[i]
    img_bg_blur = img_bg_blur_list[i]
    img_mask_color = img_mask_color_list[i]

    img_concat = np.where(img_mask_color==255, img_orig, img_bg_blur)
    img_concat = cv2.cvtColor(img_concat, cv2.COLOR_BGR2RGB)

    # 저장
    plt.figure(figsize=(20,20))
    plt.imshow(img_concat)
    plt.show()
    # save_path = os.getenv('HOME')+'/aiffel/human_segmentation/' # 저장 경로
    # cv2.imwrite('{}_{}/result_{}.jpg'.format(save_path, save_folder, i+1), img_concat)
    print('{}번째 사진 변환 및 저장 완료'.format(i+1))

print('*'*70)
print('필터 적용 및 저장 완료')

return None

# 전체 실행
def make_bg_blur(image_path_list, label_num, save_folder=None):
    def_image_list = save_image_list(image_path_list)
    def_segvalues_list, def_output_list = save_model_output(image_path_list, label_num)
    def_img_mask_list = make_seg_mask(def_image_list, def_output_list, def_segvalues_list)
    concat_w_blur(def_image_list, def_img_mask_list, save_folder)

```

In [6]:

```

# 이미지 불러오기
# image_path = './files/human_segmentation/images/*'
image_path = os.getenv('HOME')+'/aiffel/human_segmentation/images/*'
image_path_list = glob.glob(image_path)
image_path_list

```

Out[6]:

```

['/aiffel/aiffel/human_segmentation/images/1.jpg',
 '/aiffel/aiffel/human_segmentation/images/4.jpg',
 '/aiffel/aiffel/human_segmentation/images/3.png',
 '/aiffel/aiffel/human_segmentation/images/2.jpg']

```

자꾸 커널이 죽어서 함수 하나씩 진행

In [7]:

```

person_label_num = 13
person_image_list = save_image_list(image_path_list)

```

In [8]:

```

person_segvalues_list, person_output_list = save_model_output(image_path_list, person_label_num)

```

```
-----LABEL NAMES-----  
====0====  
['flower', 'person', 'wall', 'plant', 'sky', 'bottle', 'tree', 'bag', 'bal  
l']  
====1====  
['wall', 'person', 'bottle', 'signboard', 'mirror', 'shelf', 'box']  
====2====  
['person', 'wall', 'table', 'floor', 'seat', 'bench', 'chair', 'plant', 'tra  
y', 'signboard', 'ball', 'stage']  
====3====  
['wall', 'person', 'apparel', 'floor', 'curtain', 'chair', 'bed', 'table',  
'mountain']  
-----
```

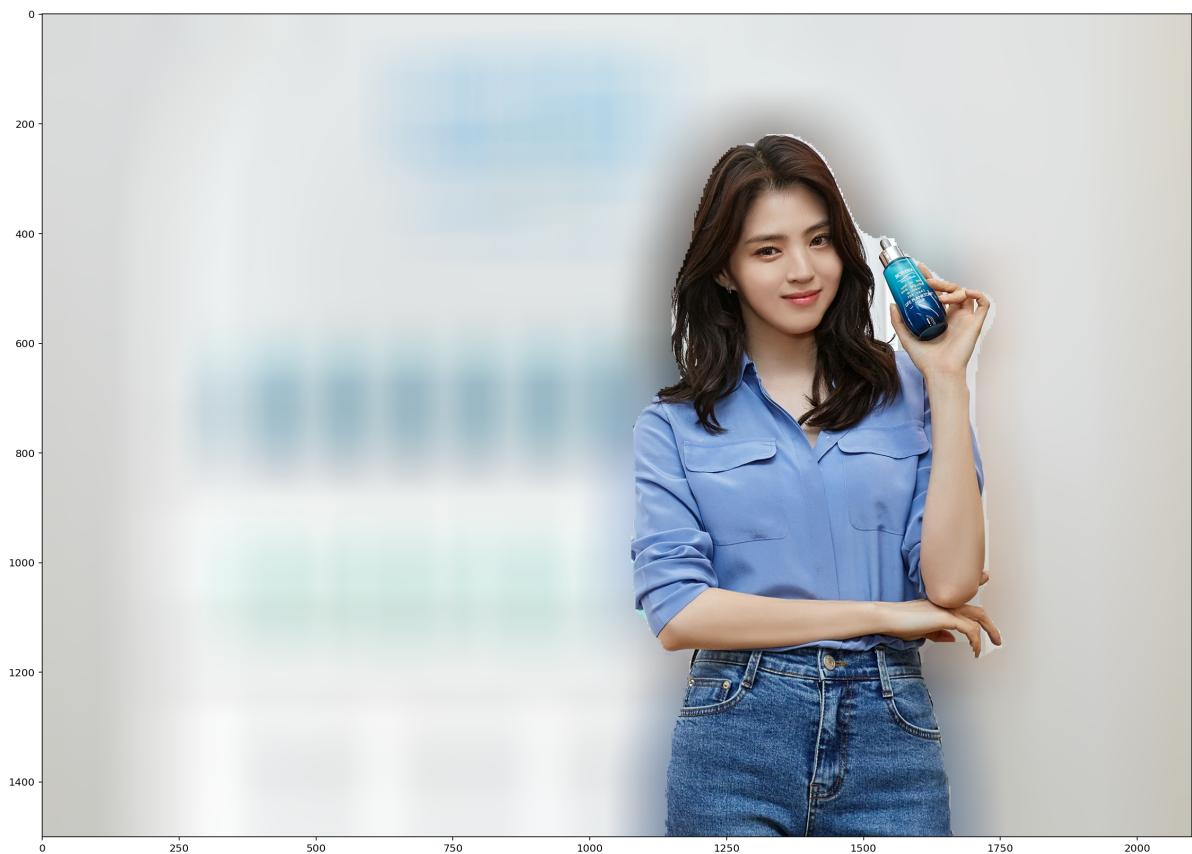
```
In [9]: person_img_mask_list = make_seg_mask(person_image_list, person_output_list,
```

```
In [10]: concat_w_blur(person_image_list, person_img_mask_list)
```

findfont: Font family ['AppleGothic'] not found. Falling back to DejaVu Sans.



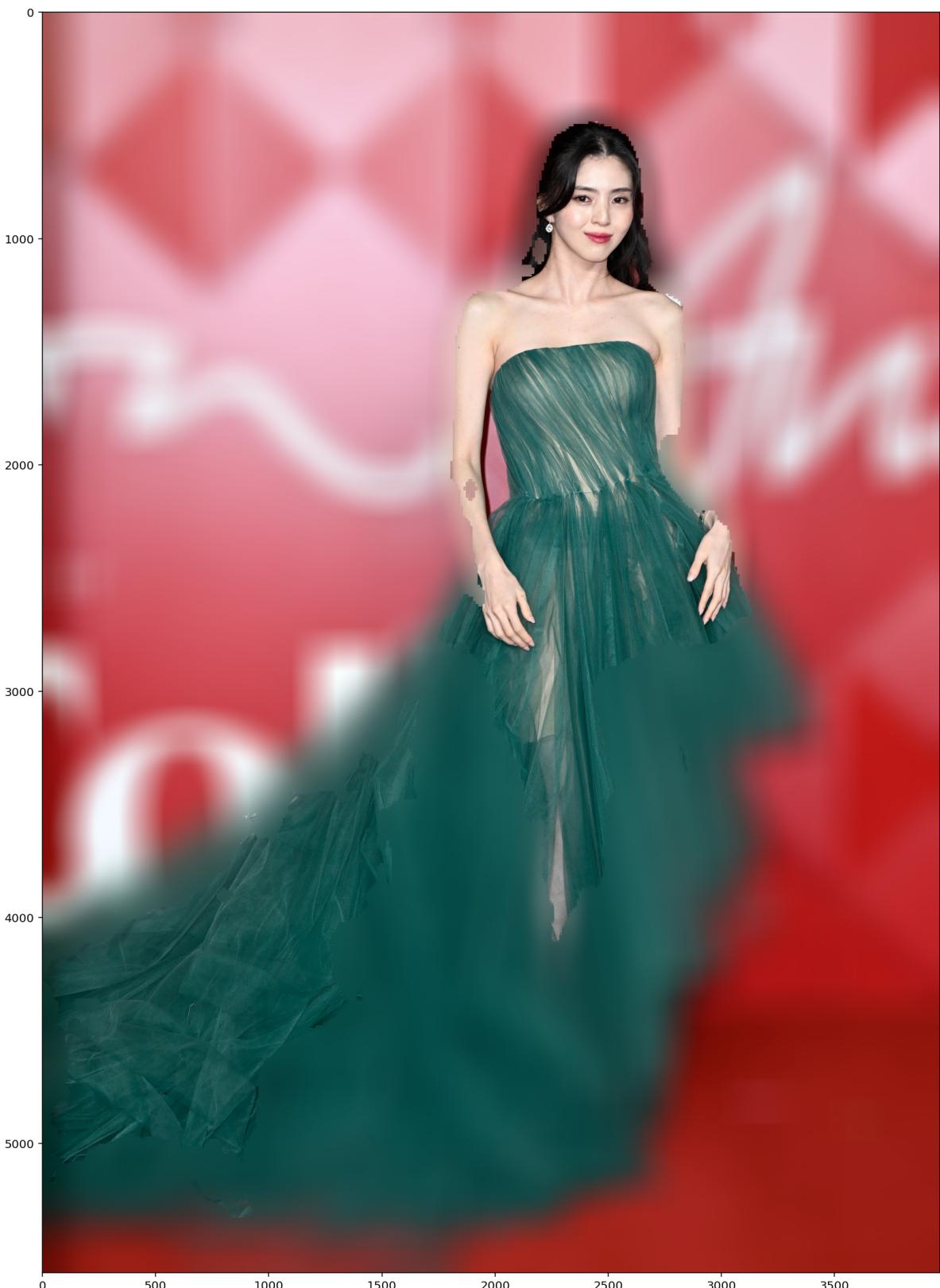
1번째 사진 변환 및 저장 완료



2번째 사진 변환 및 저장 완료



3번째 사진 변환 및 저장 완료



4번째 사진 변환 및 저장 완료

=====

필터 적용 및 저장 완료

```
In [12]: # 강아지 이미지 불러오기
# dog_image_path = './files/human_segmentation/dog_images/*'
dog_image_path = os.getenv('HOME')+ '/aiffel/human_segmentation/dog_images/*'
dog_image_path_list = glob.glob(dog_image_path)
dog_image_path_list
```

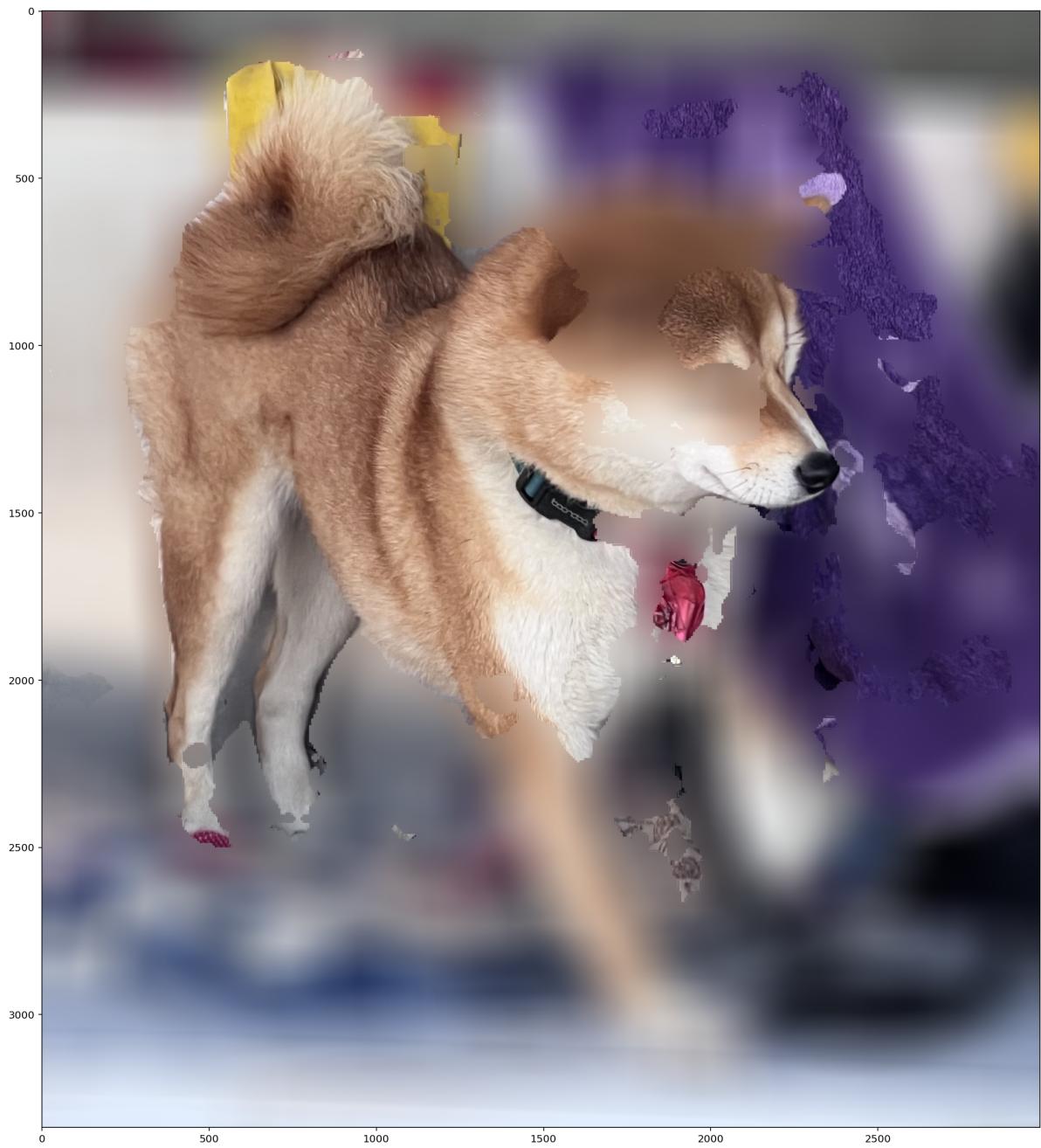
```
Out[12]: ['/aiffel/aiffel/human_segmentation/dog_images/3.jpeg',
          '/aiffel/aiffel/human_segmentation/dog_images/1.jpg',
          '/aiffel/aiffel/human_segmentation/dog_images/2.jpg']
```

```
In [13]: animal_label_num = 127
dog_image_list = save_image_list(dog_image_path_list)

In [14]: dog_segvalues_list, dog_output_list = save_model_output(dog_image_path_list,
-----LABEL_NAMES-----
====0====
['animal', 'person', 'wall', 'bag', 'table', 'floor', 'towel', 'door', 'bed',
'apparel', 'curtain', 'earth', 'fence', 'pillow', 'basket', 'flower', 'car',
'chair', 'food', 'cushion', 'barrel', 'box', 'tree', 'tent', 'blanket',
'hovel', 'water', 'sky', 'ceiling', 'bench', 'plant', 'clock', 'windowpane',
'bottle', 'building']
====1====
['grass', 'sky', 'tree', 'person', 'plant', 'mountain', 'door', 'curtain',
'animal', 'waterfall', 'flower', 'water', 'blanket', 'towel']
====2====
['floor', 'person', 'curtain', 'sky', 'animal', 'wall', 'water', 'earth', 'sculpture',
'base', 'towel', 'shower', 'grass', 'rug', 'plaything', 'flag']
-----
```

```
In [15]: dog_img_mask_list = make_seg_mask(dog_image_list, dog_output_list, dog_segva
```

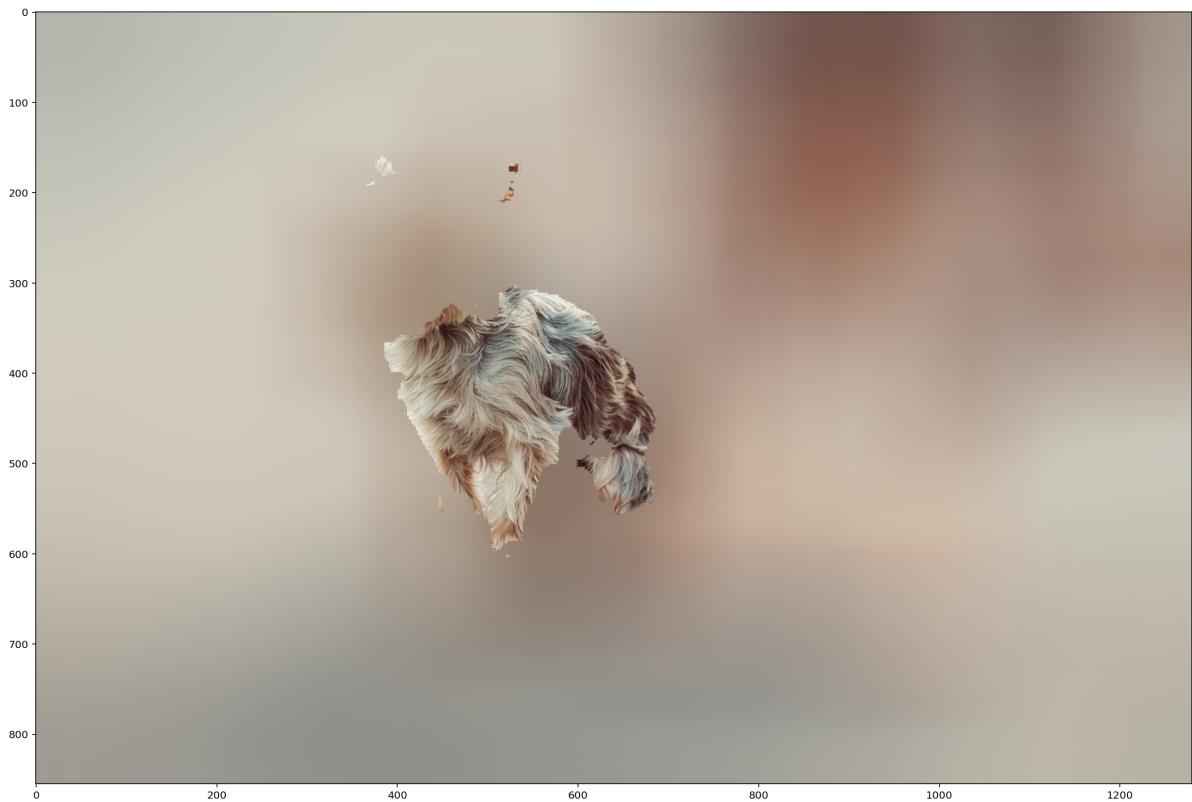
```
In [16]: concat_w.blur(dog_image_list, dog_img_mask_list)
```



1번째 사진 변환 및 저장 완료



2번째 사진 변환 및 저장 완료



3번째 사진 변환 및 저장 완료

---

=====  
필터 적용 및 저장 완료

## (2) 사진 밝기 조절

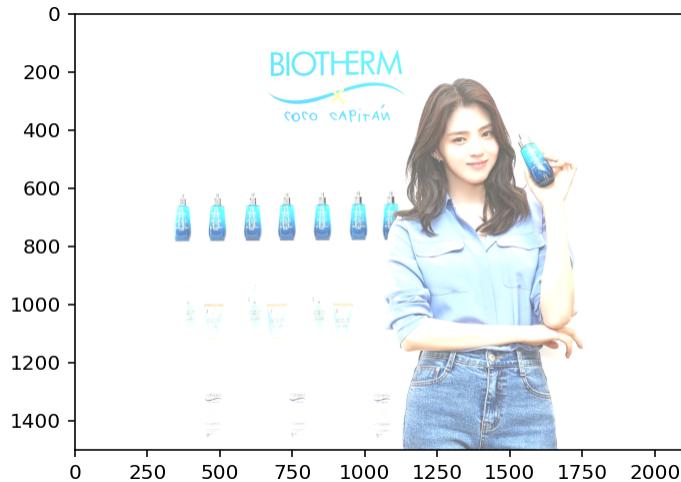
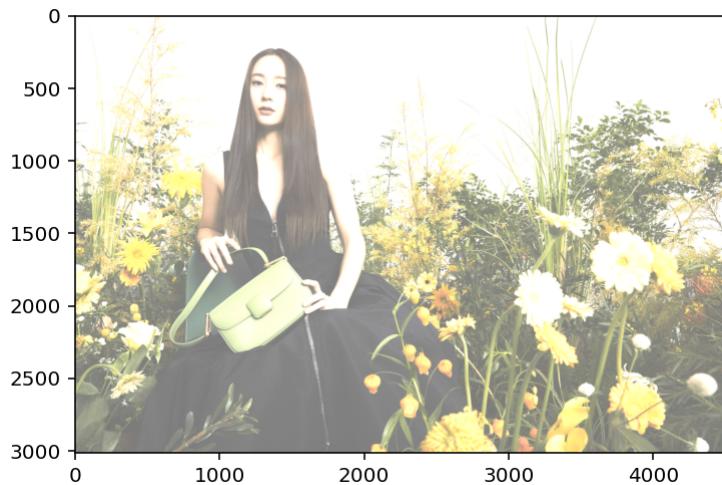
```
In [18]: # 사람 이미지 불러오기(단체사진 제외) (model:Ade20k)
# image_path = './files/human_segmentation/images/*'
image_path = os.getenv('HOME') + '/aiffel/human_segmentation/images/*'
image_path_list = glob.glob(image_path)
image_path_list
```

```
Out[18]: ['/aiffel/aiffel/human_segmentation/images/1.jpg',
 '/aiffel/aiffel/human_segmentation/images/4.jpg',
 '/aiffel/aiffel/human_segmentation/images/2.jpg']
```

```
In [26]: # 밝기 조절 후 image_list에 저장
person_image_list = []
```

```
for image in image_path_list:
    img_bgr = cv2.imread(image)
    img_bgr = cv2.add(img_bgr, (100, 100, 100, 0))
    img_rgb = cv2.cvtColor(img_bgr, cv2.COLOR_BGR2RGB)
    person_image_list.append(img_bgr)

plt.imshow(img_rgb)
plt.show()
```



```
In [27]: person_label_num = 13
person_segvalues_list, person_output_list = save_model_output(image_path_list)

-----LABEL NAMES-----
====0====
['flower', 'person', 'wall', 'plant', 'sky', 'bottle', 'tree', 'bag', 'bal
l']
====1====
['wall', 'person', 'bottle', 'signboard', 'mirror', 'shelf', 'box']
====2====
['wall', 'person', 'apparel', 'floor', 'curtain', 'chair', 'bed', 'table',
'mountain']
-----
```

```
In [28]: person_img_mask_list = make_seg_mask(person_image_list, person_output_list,
```

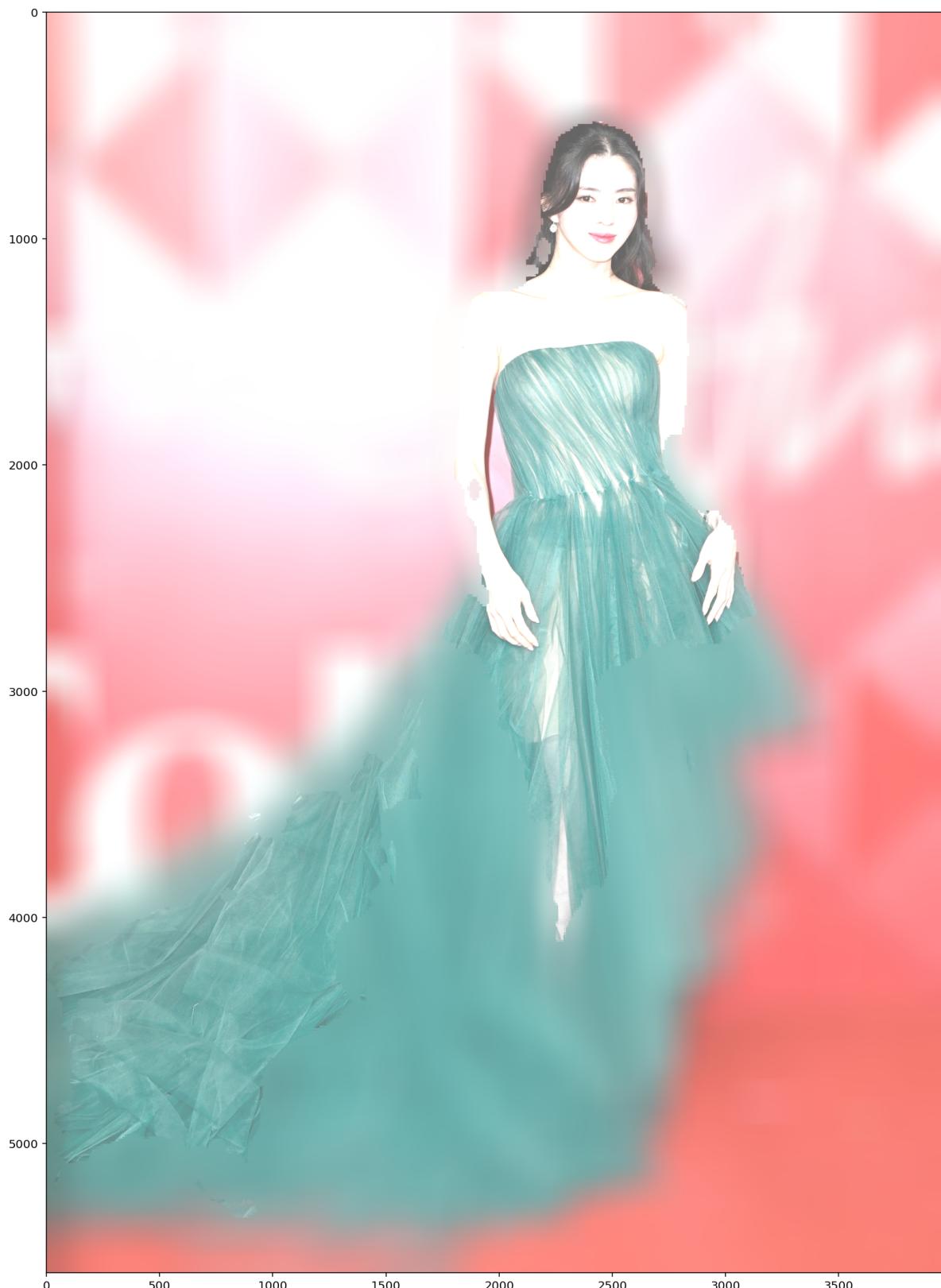
```
In [29]: concat_w_blur(person_image_list, person_img_mask_list)
```



1번째 사진 변환 및 저장 완료



2번째 사진 변환 및 저장 완료



3번째 사진 변환 및 저장 완료

=====

필터 적용 및 저장 완료

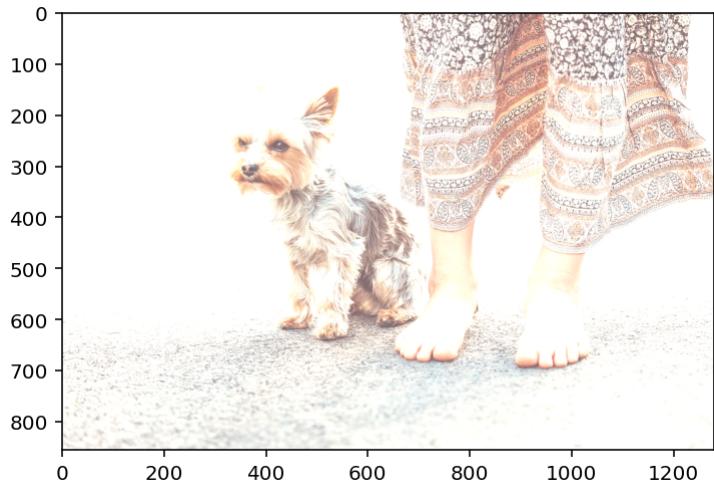
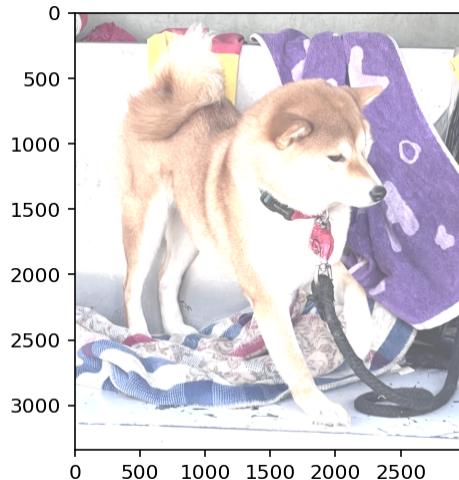
```
In [33]: # 강아지 이미지 불러오기(model:PASCAL VOC)
# dog_image_path = './files/human_segmentation/dog_images/*'
dog_image_path = os.getenv('HOME')+ '/aiffel/human_segmentation/dog_images/*'
dog_image_path_list = glob.glob(dog_image_path)
dog_image_path_list
```

```
Out[33]: ['/aiffel/aiffel/human_segmentation/dog_images/3.jpeg',
          '/aiffel/aiffel/human_segmentation/dog_images/1.jpg',
          '/aiffel/aiffel/human_segmentation/dog_images/2.jpg']
```

```
In [35]: # 밝기 조절 후 image_list에 저장
dog_image_list = []

for image in dog_image_path_list:
    img_bgr = cv2.imread(image)
    img_bgr = cv2.add(img_bgr, (100, 100, 100, 0))
    img_rgb = cv2.cvtColor(img_bgr, cv2.COLOR_BGR2RGB)
    dog_image_list.append(img_bgr)

plt.imshow(img_rgb)
plt.show()
```



```
In [36]: dog_segvalues_list, dog_output_list = save_model_output(dog_image_path_list,
```

```
WARNING:tensorflow:5 out of the last 14 calls to <function Model.make_predict_function.<locals>.predict_function at 0x7fc606e88f70> triggered tf.function retracing. Tracing is expensive and the excessive number of tracings could be due to (1) creating @tf.function repeatedly in a loop, (2) passing tensors with different shapes, (3) passing Python objects instead of tensors. For (1), please define your @tf.function outside of the loop. For (2), @tf.function has experimental_relax_shapes=True option that relaxes argument shapes that can avoid unnecessary retracing. For (3), please refer to https://www.tensorflow.org/guide/function#controlling_retracing and https://www.tensorflow.org/api_docs/python/tf/function for more details.
```

```
-----LABEL NAMES-----
```

```
====0====
```

```
background
```

```
dog
```

```
====1====
```

```
background
```

```
dog
```

```
====2====
```

```
background
```

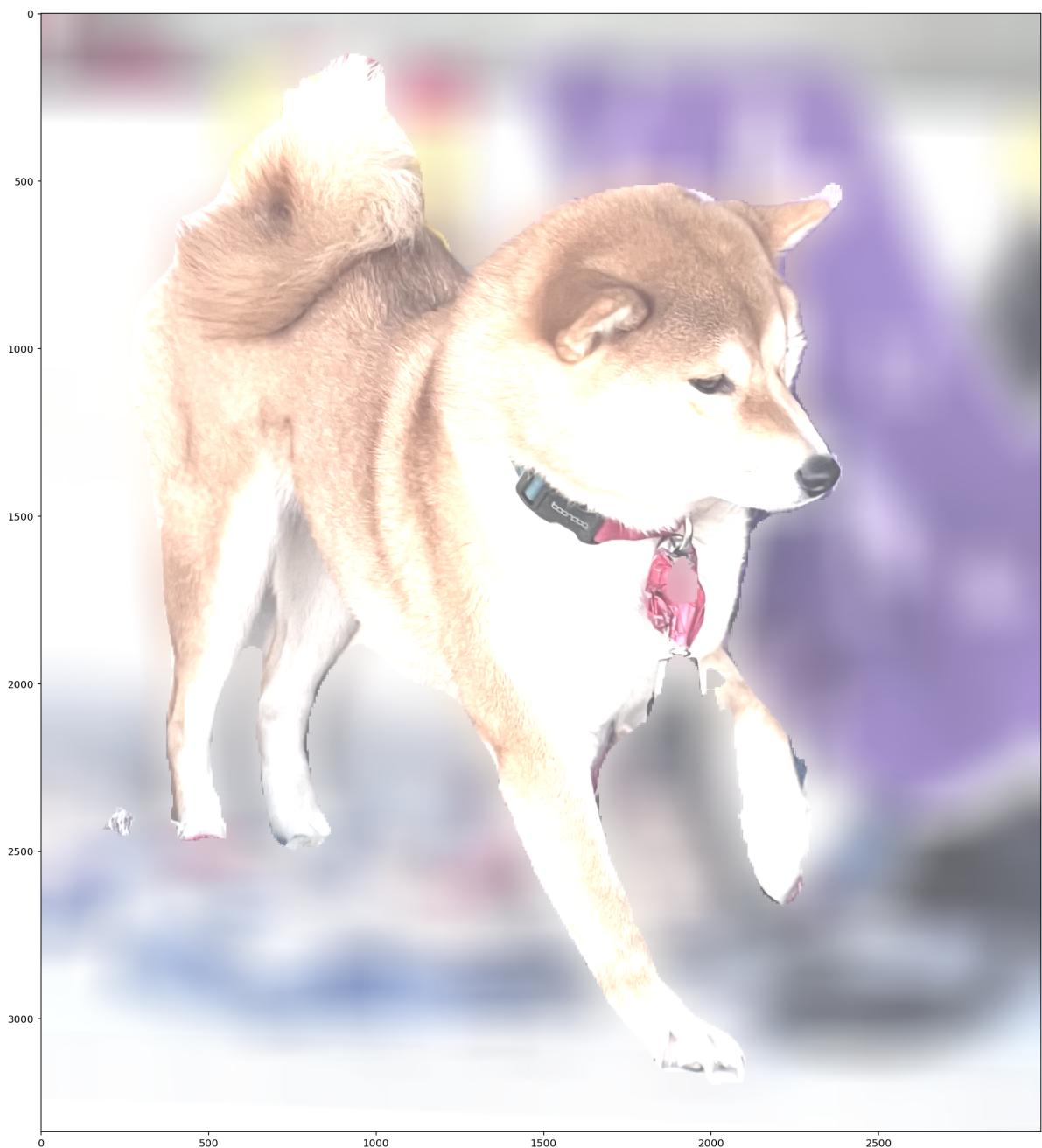
```
dog
```

```
person
```

```
-----
```

```
In [37]: dog_img_mask_list = make_seg_mask(dog_image_list, dog_output_list, dog_seg_c
```

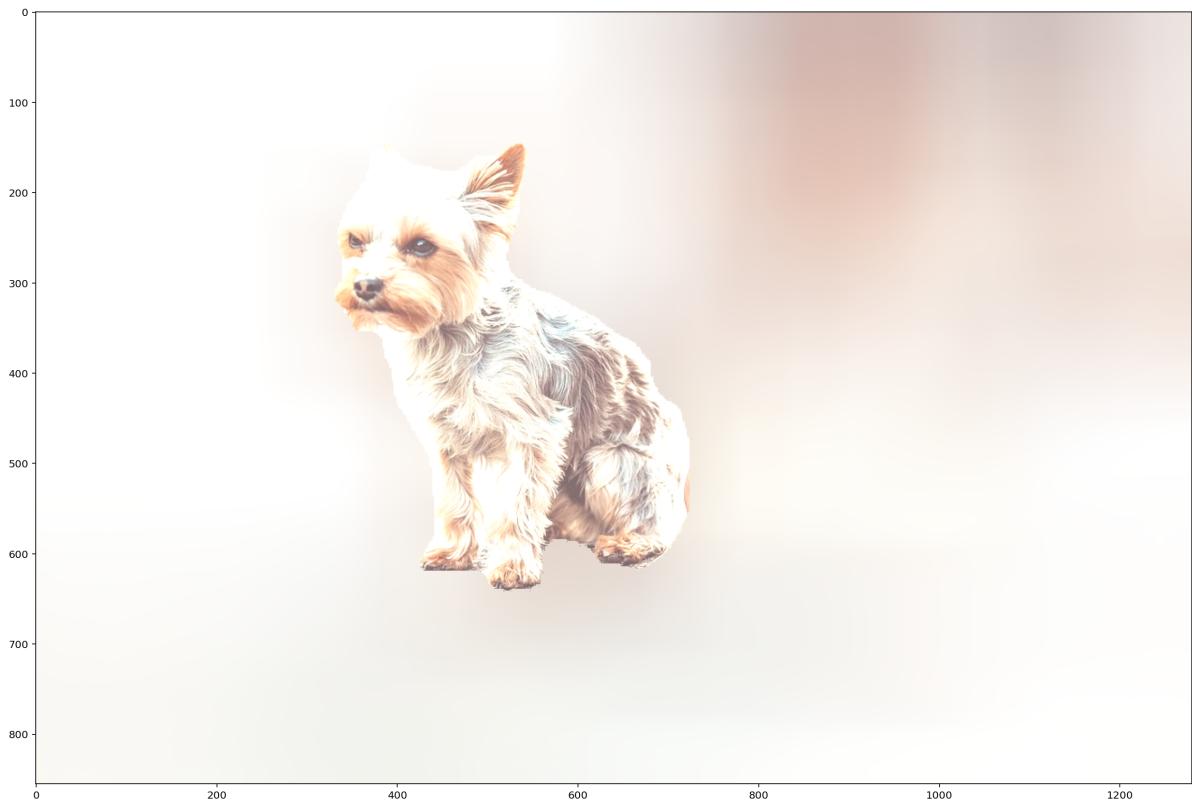
```
In [40]: concat_w_blur(dog_image_list, dog_img_mask_list)
```



1번째 사진 변환 및 저장 완료



2번째 사진 변환 및 저장 완료



3번째 사진 변환 및 저장 완료

---

 필터 적용 및 저장 완료

### 3-2. 크로마키 적용

```
In [53]: # 원본 사진 블러 처리 -> 배경 추출 -> 블러 배경과 타겟 합치기
def chromakey(background, image_list, img_mask_list):

    # 배경 추출
    img_mask_color_list = []
    img_bg_list = []
```

```

image_resize_list = []

for i in range(len(image_list)):
    img_orig = image_list[i]
    img_mask = img_mask_list[i]
    # 사이즈 맞추기
    image_resize = cv2.resize(background, (img_orig.shape[1], img_orig.shape[0]))
    image_resize_list.append(image_resize)

    img_mask_color = cv2.cvtColor(img_mask, cv2.COLOR_GRAY2BGR)
    img_mask_color_list.append(img_mask_color)
    # cv2.bitwise_not(): 이미지가 반전 -> 배경 : 255, 사람: 0 으로 만들기
    img_bg_mask = cv2.bitwise_not(img_mask_color)
    # cv2.bitwise_and()을 사용하면 배경만 있는 영상을 얻을 수 있습니다.
    # 0과 어떤 수를 bitwise_and 연산을 해도 0이 되기 때문에
    # 사람이 0인 경우에는 사람이 있던 모든 픽셀이 0이 됩니다. 결국 사람이 사라지고 배경만 남게 됩니다.
    img_bg = cv2.bitwise_and(image_resize, img_bg_mask)
    img_bg_list.append(img_bg)

    # 블러 배경과 타겟 합치기

for i in range(len(image_list)):
    img_orig = image_list[i]
    img_bg = img_bg_list[i]
    img_mask_color = img_mask_color_list[i]

    img_concat = np.where(img_mask_color==255, img_orig, img_bg)
    img_concat = cv2.cvtColor(img_concat, cv2.COLOR_BGR2RGB)

plt.figure(figsize=(20,20))
plt.imshow(img_concat)
plt.show()

return None

```

In [44]: background\_path = os.getenv('HOME') + '/aiffel/human\_segmentation/background.jpg'  
background = cv2.imread(background\_path)

In [45]: # 사람 이미지 불러오기(단체사진 제외) (model:Ade20k)  
# image\_path = './files/human\_segmentation/images/\*'  
image\_path = os.getenv('HOME') + '/aiffel/human\_segmentation/images/\*'  
image\_path\_list = glob.glob(image\_path)  
image\_path\_list

Out[45]: ['/aiffel/aiffel/human\_segmentation/images/1.jpg',  
'/aiffel/aiffel/human\_segmentation/images/4.jpg',  
'/aiffel/aiffel/human\_segmentation/images/2.jpg']

In [46]: person\_label\_num = 13  
person\_image\_list = save\_image\_list(image\_path\_list)

In [47]: person\_segvalues\_list, person\_output\_list = save\_model\_output(image\_path\_list)

```
WARNING:tensorflow:5 out of the last 13 calls to <function Model.make_predict_function.<locals>.predict_function at 0x7fc607790b80> triggered tf.function retracing. Tracing is expensive and the excessive number of tracings could be due to (1) creating @tf.function repeatedly in a loop, (2) passing tensors with different shapes, (3) passing Python objects instead of tensors. For (1), please define your @tf.function outside of the loop. For (2), @tf.function has experimental_relax_shapes=True option that relaxes argument shapes that can avoid unnecessary retracing. For (3), please refer to https://www.tensorflow.org/guide/function#controlling_retracing and https://www.tensorflow.org/api_docs/python/tf/function for more details.
```

```
-----LABEL NAMES-----
```

```
====0=====
```

```
['flower', 'person', 'wall', 'plant', 'sky', 'bottle', 'tree', 'bag', 'bal
```

```
l']
```

```
====1=====
```

```
['wall', 'person', 'bottle', 'signboard', 'mirror', 'shelf', 'box']
```

```
====2=====
```

```
['wall', 'person', 'apparel', 'floor', 'curtain', 'chair', 'bed', 'table',
```

```
'mountain']
```

```
-----
```

```
In [48]: person_img_mask_list = make_seg_mask(person_image_list, person_output_list,
```

```
In [54]: chromakey(background, person_image_list, person_img_mask_list)
```



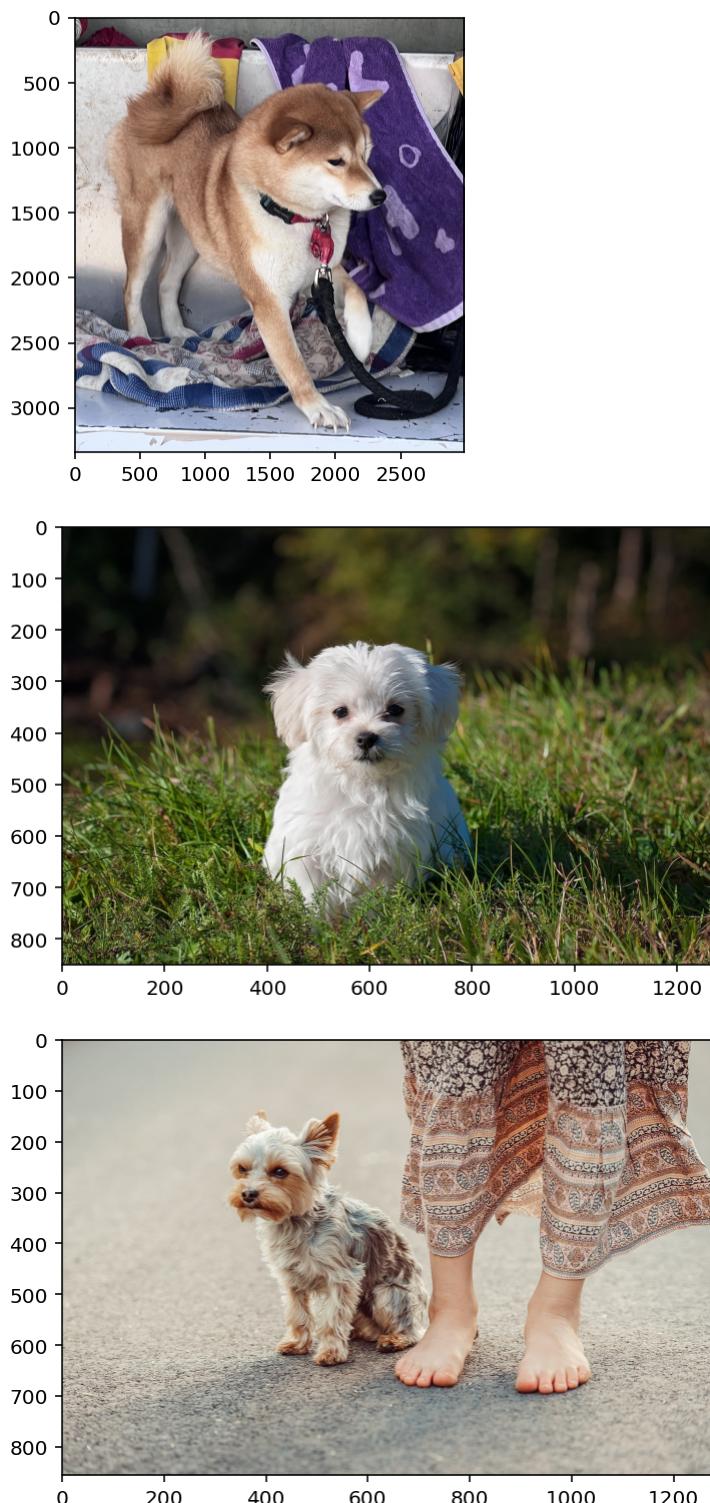




```
In [56]: # 강아지 이미지 불러오기(model:PASCAL VOC)
# dog_image_path = './files/human_segmentation/dog_images/*'
dog_image_path = os.getenv('HOME') + '/aiffel/human_segmentation/dog_images/*'
dog_image_path_list = glob.glob(dog_image_path)
dog_image_path_list
```

```
Out[56]: ['/aiffel/aiffel/human_segmentation/dog_images/3.jpeg',
          '/aiffel/aiffel/human_segmentation/dog_images/1.jpg',
          '/aiffel/aiffel/human_segmentation/dog_images/2.jpg']
```

```
In [57]: dog_image_list = save_image_list(dog_image_path_list)
```

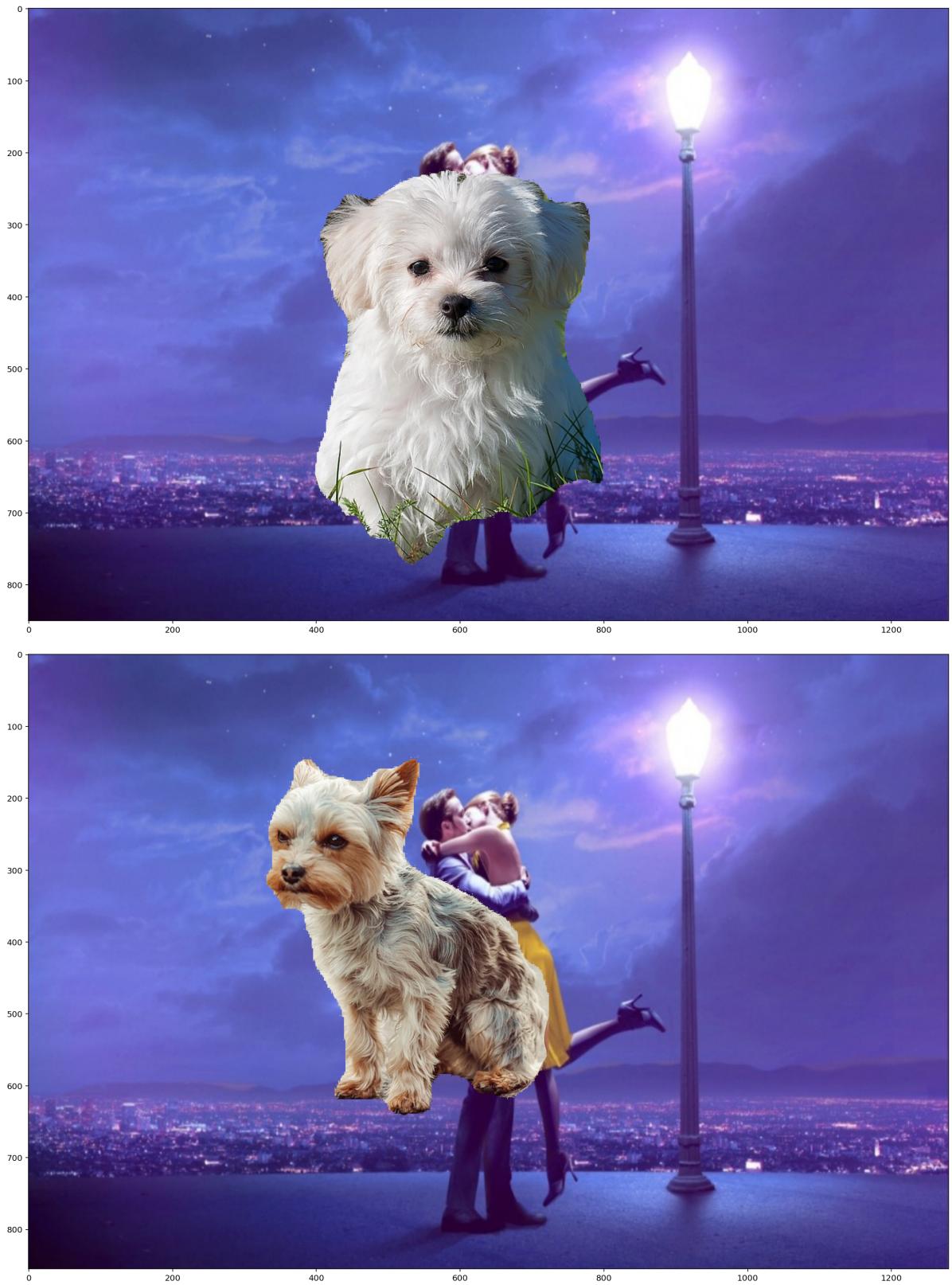


```
In [58]: dog_segvalues_list, dog_output_list = save_model_output(dog_image_path_list,  
-----LABEL_NAMES-----  
====0====  
background  
dog  
====1====  
background  
dog  
====2====  
background  
dog  
person  
-----
```

```
In [59]: dog_img_mask_list = make_seg_mask(dog_image_list, dog_output_list, dog_seg_c
```

```
In [60]: chromakey(background, dog_image_list, dog_img_mask_list)
```





---

## 고찰

### (1) 사용한 모델을 바꿔봄

```
semantic_segmentation.load_pascalvoc_model(20개의 분류) ->  
semantic_segmentation.load_ade20k_model(150개 분류)
```

<https://paperswithcode.com/dataset/pascal-voc><https://paperswithcode.com/dataset/ade20k>

## 인물

- 1, 2번 사진 : 자연스럽게 자르기 위해 사람이 아닌 사물을 포함하는 결과를 볼 수 있음
- 3, 4번 사진 : 더 상세하게 분류하려다 보니 결과가 더 안좋아 보임

## 강아지

- 어이고야 개판임... 이전 모델이 훨씬 나음

## 데이터 셋에 따라 다르다!!

**-> 인물1,2,4번 : ade20k 모델/ 강아지 : pascalvoc 모델 사용해서 이 후 과정 진행**

---

## (2) Shllow Focus == 심도가 얕다

- 카메라의 이미지 센서가 클 수록
- 이미지를 zoom 할수록
- 조리개가 많이 개방될 수록 (F값이 작을 수록). 빛을 많이 받을 수록
- 카메라와 피사체간의 거리가 가까울 때
- 피사체와 배경의 거리가 멀 수록

## 인물

- 바지와 같이 다리의 형태가 드러나는 경우 인식이 잘 되는 편인듯 하지만 드레스의 경우 '사람'의 형태와 달라서 다른 모델을 활용해 '사람+ドレス'를 인식하게 해야 할 것 같음
  - 가방, 물건 등으로 실루엣이 가려질 때, 신체 일부까지 불러 됨
  - 사람이 여러명일 때 오류가 심함
  - 팔과 허리 사이의 공간이 배경으로 인식되지 못함
- 피사체가 카메라와는 가깝고 배경과는 멀어야하지만 예시의 사진들의 인물이 배경과 가까워서 문제가 있었던 걸로 보인다  
 -> 이미 찍혀 있는 사진에 대해서 해결 할 수 있는 방법
  1. 밝기 조절 : 큰 차이 없음
  2. 상체 위주로 조절 : 현재 결과물에서도 상체는 잘 분류된 것을 볼 수 있음

## 강아지

- 사람보다 인식이 잘 되는 편 인듯
  - 다리 부분이 상대적으로 인식률이 떨어짐
1. 밝기 조절 : 큰 차이 없음
-

### (3)

1. 변수 설정 유의할 필요가 있어보인다. 반복문을 설정하면서 자주 하는 실수였다 > 규칙을 정하는 것이 좋을 듯!
  - 여러개 : -s
  - list : -list
2. 데이터 셋에 따라 좋은 결과를 내는 모델이 다르기 때문에 최대한 많이 알고 끊임 없이 시도해 보아야 하는 것 같다 :)
3. 과정을 정리하기 위해 위아래로 오간 부분이 있다. 큰 그림을 볼 수 있도록 정리하자 'ㅅ' ~