**Pusan National University**

**Computer Science and Engineering**

**Technical Report 2023-10**

# 2023 전기 졸업과제 최종보고서

저자 1 : 전기컴퓨터공학부 정보컴퓨터공학전공 201724597 천형주

저자 2 : 정보컴퓨터공학부 202055536 민예진

지도교수 : 이명호 교수님

# 목 차

# 1. 프로젝트 개요

## 1.1. 프로젝트 배경

드라마나 영화를 봤을 때 시대에 맞지 않는 구조물이 발견되는 경우가 있다. 시대적 배경과 맞지 않는 CCTV나 천장형 에어컨이 보인다면 시청자의 몰입감을 떨어뜨리는 요인이 된다.

그렇다면 상황에 맞지 않는 물체를 없앨 수 있다면 몰입감을 떨어뜨리지 않고 실재감을 증대시킬 수 있지 않을까?

증강 현실(AR, Argumented Reality)와 반대로 현실에서 특정 물체는 지우는 기술을 감소 현실(DR, Diminished Reality)라고 한다.

최근 Passthrough 기능으로 현실과 가상세계를 넘나드는 증강 현실 콘텐츠가 개발되면서 DR을 실시간으로 처리해줄 필요가 있다. 예를 들어 증강 현실에서 가상 캐릭터를 띄울 공간에 물체가 있거나 가로막혀 있다면 가상 캐릭터에 대한 몰입감이 떨어질 것이다. 이때 inpainting 기술을 이용하여 가상 캐릭터가 놓일 위치에 몰입감을 방해하는 물체를 실시간으로 제거할 수 있다면 증강 현실 콘텐츠를 즐기는데 큰 도움이 될 것이다.

본 프로젝트는 기존에 존재하는 실시간 image segmentation 모델과 이미지나 동영상에서 특정 부분을 지우고 채워주는 inpainting 모델을 결합하여 실시간으로 객체를 검출하고 지우는 프로그램 구현한다.

## 1.2. 프로젝트 목표

기존에 존재하는 실시간 image segmentation 모델과 이미지 및 동영상에서 특정 영역을 지우고 채워주는 inpainting 모델을 활용하여 증강 현실에서 실재감을 높일 수 실시간 Diminished Reality 프로그램을 구현한다.

## 2. 배경지식

### 2.1. YOLOv8 : You Only Look Once: Unified, Real-Time Object Detection

이미지 세그멘테이션(Image Segmentation)이란 픽셀 별로 미리 정의된 몇 개의 클래스로 분류(classification)하는 문제로 시각적 환경을 완전히 이해하는 데 필요한 핵심적인 컴퓨터 비전 기술 중의 하나이다.[1]

이 프로젝트에서는 YOLO를 사용하는데 기존의 복잡한 파이프라인을 가진 객체 검출 모델의 프로세스를 개선하여 객체 검출 프로세스를 단순화함으로써 빠른 객체 검출을 위해 설계된 프로그램이다.[2]

YOLOv8은 2023년 1월에 발표된 버전으로, YOLOv8을 위해 새로운 repository를 출시하며 dectection, segmentation, classification, pose estimation을 수행하기 위한 프레임워크로 구축되었다.[3]

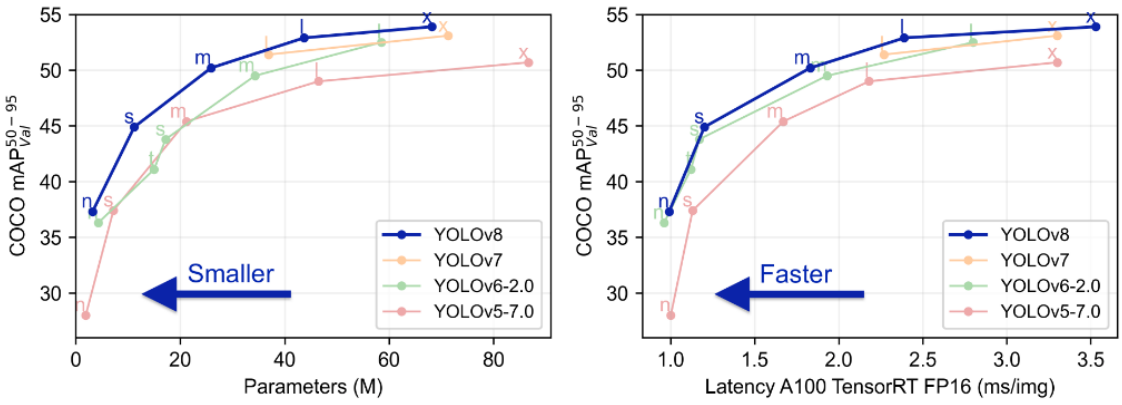



그림 1 YOLOv8 성능 지표


## 2.2. LaMa: Resolution-robust Large Mask Inpainting with Fourier Convolutions

인페인팅(Inpainting)이란 이미지의 특정 부분이 손상되었거나 원하는 부분이 어떤 것에 의해 가려진 상황 속에서 손상되거나 가려진 부분을 복원, 생성하는 개념이다. [4]

이 프로젝트에서는 3가지 모델을 사용하는데 첫 번째 모델이 [5]Large effective receptive field를 확보하여 기존에 겪던 Inpainting의 한계를 개선한 LaMa이다. 이 모델의 특징은 (i) Fast Fourier Convolution (FFCs)를 이용한 architecture, (ii) High receptive field perceptual loss, (iii) large training mask이다.

### (i) Fast Fourier Convolution (FFCs)

고정된 scale로 연산하는 바닐라 합성곱(vanilla convolution)의 한계를 개선하고자 사용한 연산이다. Fast Fourier Transform(FFT) 사용한 Spectral transformer, Fourier Unit(FU), Local Fourier Unit(LFU)로 구성되어 있다. Spectral transformer를 통해 spectral domain에서 non-local한 연산을 진행하고 FU를 통해 이미지 전체에 대한 정보를 다루고 LFU를 통해 semi-local한 정보를 다룬다.

### (ii) High receptive field perceptual loss

LaMa는 전체 구조(global structure)를 이해하는 것을 목표로 하기 때문에 high receptive field를 기반으로 한 HRF pretrained model을 사용하여 perceptual loss를 계산한다.

### (iii) Large training mask

이미지의 50%를 넘지 않지만 기존 training mask보다 큰 mask로 학습하였다

# deepfillv2-pytorch

A PyTorch reimplementation of the paper **Free-Form Image Inpainting with Gated Convolution** (DeepFillv2) (https://arxiv.org/abs/1806.03589) based on the original TensorFlow implementation.

Example images (raw | masked | inpainted):

## 2.3. DeepFillv2: Free-Form Image Inpainting with Gated Convolution[6]

유저가 그린 free-form mask에 동작하는 Inpainting 모델을 목적으로 설계되었다. 이 프로젝트에서는 딥러닝 프레임워크의 통일을 위해 Pytorch로 재구현된 모델을 사용하였다. 이 모델의 특징은 (i) Gated Convolution, (ii) SN-PatchGAN, Patch-based GAN loss이다.

### (i) Gated Convolution

모든 input pixel을 유효한 정보로 보고 다룬 vanilla convolution의 단점을 해결하고, 유효 픽셀에만 적용되도록 제한하는 partial convolution을 일반화하여 학습 가능한 dynamic feature selection mechanism을 가진 방법이다. 각각 다른 weight에 대한 convolution 연산을 적용한 게이팅(Gating)과 Feature를 각 행렬의 원소끼리만 곱하여 (element-wise product) valid feature일 확률이 높은 Feature에 가중치를 두어 연산한다.

### (ii) SN-PatchGAN

임의의 위치와 모양을 가진 다수의 구멍을 자연스럽게 채우는 Free-form Image Inpainting을 위해 고안되었다. 피드 포워딩(Feed forwarding) 후 strided convolution의 결과인 feature map에 GAN을 적용한 것을 기반으로 Generator와 Discriminator을 구성하였다.

MAT: Mask-Aware Transformer for Large Hole Image Inpainting (CVPR 2022 Best Paper Finalist, Oral)

### 2.4. MAT: Mask-Aware Transformer for Large Hole Image Inpainting[7]

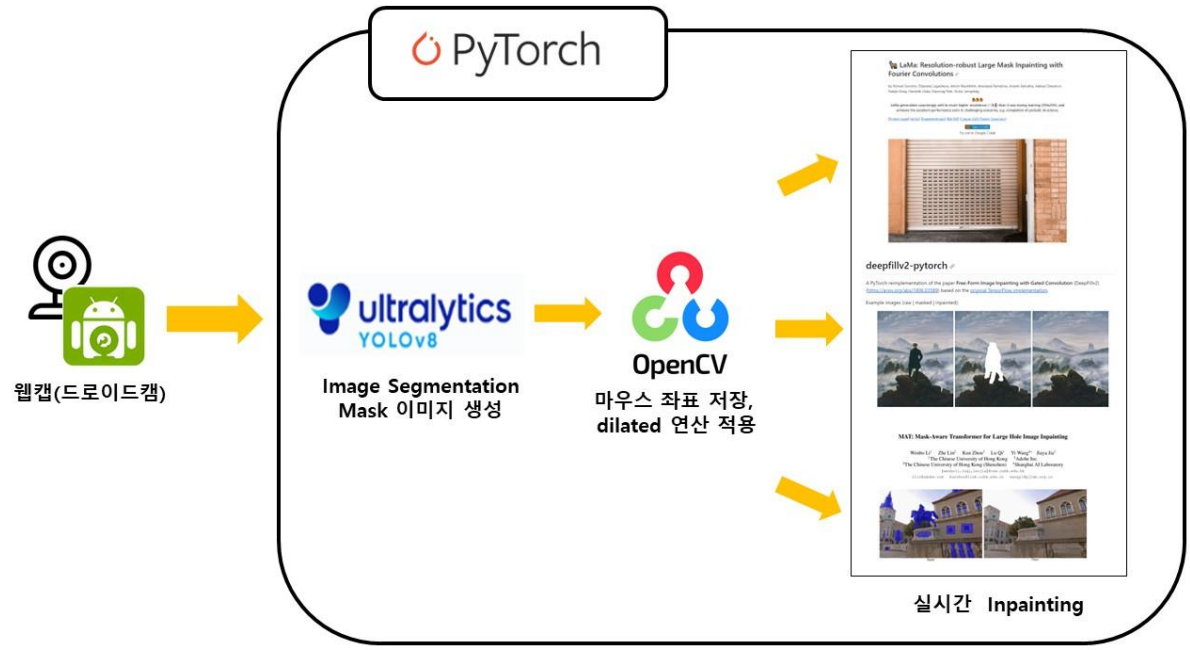
저해상도에 주로 사용하는 어텐션(attention) 기법 혹은 transformer을 개선하여 고해상도 사진에도 효과적인 성능을 내고자 하였다. 특징은 layer normalization을 제거하고 잔차 학습(residual learning) 대신 fusion learning을 사용한 adjusted transformer block과 Inpainting 결과를 자연스럽게 하기 위해 이미지를 Editing 할 수 있는 style manipulation module이다.

## 3. 프로젝트 내용

### 3.1. 기존 제약 사항에 대한 수정 사항

- **실험기기 및 환경 변경 :** VR 기기에서 AR 환경을 조성하기 위해 passthrough 기능을 구현하였지만 passthrough 결과물에서 컬러 이미지와 이미지의 테두리를 그리는 흑백 이미지가 일치하지 않았고, passthrough 기능을 실행할 때 미러링 기능을 사용할 수 없어 실험 기기를 VR 기기에서 웹캠으로 변경하였다.

### 3.2. 프로젝트 설계




1) 웹캠(드로이드캠)의 프레임 이미지를 YOLOv8을 이용해 Image Segmentation

2) Image Segmentation한 이미지에서 마우스 클릭을 통해 객체를 선택, 선택한 객체의 마스크 이미지 생성

3) 생성한 마스크 이미지를 Dilate 연산을 적용해 마스크 영역을 확장

4) 웹캠 프레임과 선택한 객체의 마스크 이미지를 이용해 실시간으로 inpainting을 실행

### 3.2.1. Image Segmentation

1) 객체 검출을 위해 YOLOv8에서 segmentation을 위한 yolov8n-seg.pt 모델을 사용한다. 모델에 적용할 소스로 웹캠(DroidCam)을 사용한다.

```
# yolo Init
model = YOLO("yolov8n-seg.pt")
names = model.names
results = model(source="http://172.21.215.175:4747/video", stream=True, verbose=False)     # verbose : console 출력 여부
```

2) 객체 선택을 위한 마우스 클릭 시 좌표 저장을 위한 class 구현하였다.

```
# 마우스 좌표 클래스
class MouseGesture:
    def __init__(self) -> None:
        self.is_plotted = False
        self.label = -1
        self.cx, self.cy = -1, -1

    def on_mouse(self, event, x, y, flags, param):
        if event == cv2.EVENT_LBUTTONDOWN:
            self.cx, self.cy = x, y
            print("왼쪽 버튼 눌림 \t좌표 : x : {} y : {}".format(x,y) )
            self.is_plotted = False
```

3) CLI에서 show argument 입력 시 동작하는 show 메소드에 마우스 클릭 이벤트를 추가하여 OpenCV의 VideoCapture 화면에 마우스 클릭이 인식되도록 수정함.

4) 기존 YOLO predict class에 맞게 구성된 plot 메소드를 mask와 box 데이터를 바로 받아 사용할 수 있도록 추출 후 수정하였다.

```
# 화면 출력 코드
def show(img):
    global mouse_info

    cv2.imshow("1", img)
    cv2.setMouseCallback("1", mouse_info.on_mouse, param=img)
    cv2.waitKey(1)
    # cv2.waitKey(500 if batch[3].startswith('image') else 1)  # 1 millisecond

# 마스크, 박스 이미지 출력
def plot(origin, boxes, masks):
    annotator = Annotator(deepcopy(origin))

    # 마스크 출력
    if (masks is not None and len(masks) != 0):
        # mask = mask.unsqueeze(0)
        im_gpu = torch.as_tensor(origin, dtype=torch.float16, device=masks.data.device).permute(
            2, 0, 1).flip(0).contiguous() / 255
        idx = boxes.cls if boxes else range(len(masks))

        annotator.masks(masks.data, colors=[colors(x, True) for x in idx], im_gpu=im_gpu)

    # 박스 출력
    if (boxes is not None and len(boxes) != 0):
        for box in reversed(boxes):
            c, id = int(box.cls), None if box.id is None else int(box.id.item())
            name = ('' if id is None else f'id:{id} ') + names[c]
            annotator.box_label(box.xyxy.squeeze(), name, color=colors(c, True))
    return annotator.result()
```

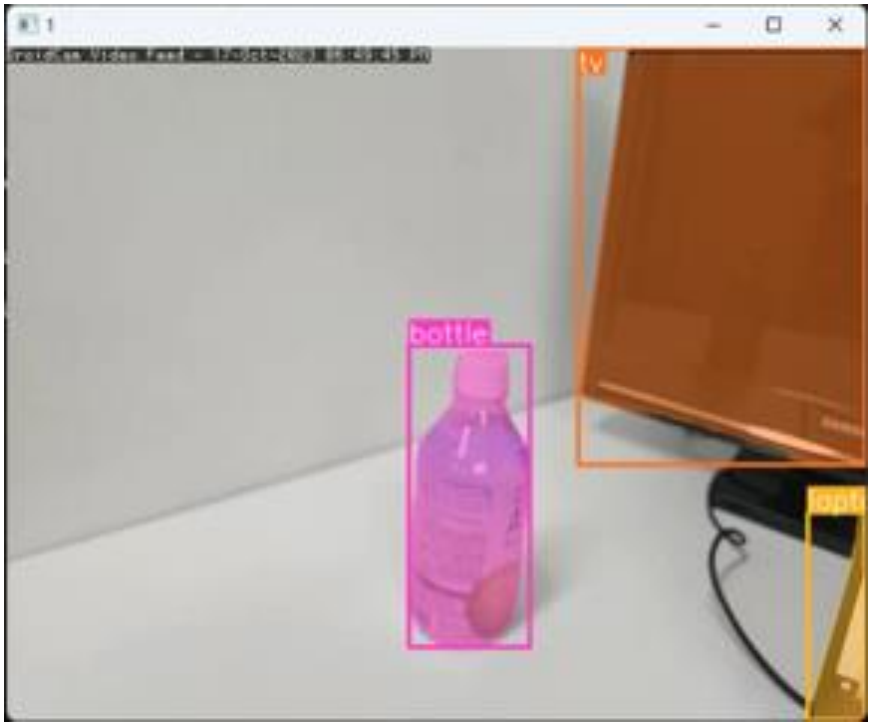



그림 2 YOLOv8 적용

5) 처음 클릭 했을 때 마스크 이미지에 클릭한 좌표가 포함되어 있을 때 해당 마스크 이지를 가지고 있는 객체의 클래스 라벨을 저장한다. 저장된 라벨을 이용하여 해당 객체를 계속 추적(tracking)하고, 객체의 마스크 이미지를 실시간으로 받아온다.

```
# 마스크 이미지 출력
else:
    # 마우스 좌표가 바뀐 경우
    if mouse_info.is_plotted == False:
        for i, e in enumerate(masks.data):
            # mask_tensor = e
            mask = e.cpu().numpy()
            label = int(boxes[i].cls)

            # 마스크 데이터에 마우스 좌표가 포함되어 있는지 확인
            if mask[mouse_info.cy][mouse_info.cx] == 1:
                cur_mask = mask
                mouse_info.label = label
                mouse_info.is_plotted = True
                is_tracking = True
                break

        # 마스크 데이터가 없으면 원본 이미지 출력
        if (is_tracking == False):
            mouse_info.label = -1

    # 좌표가 바뀌지 않은 경우
    else:
        # 클릭한 좌표 밖으로 대상이 벗어났을 때
        # 라벨이 있는 경우 기존 라벨을 이용해서 마스크 트래킹
        if mouse_info.label != -1:
            for i, box in enumerate(boxes):
                if label == int(box.cls):
                    # mask_tensor = masks.data[i]
                    cur_mask = masks.data[i].cpu().numpy()
                    is_tracking = True
                    break

    # 출력할 마스크가 없는 경우 원본 이미지 출력
    if (is_tracking == False):
        # 클릭한 곳에 마스킹할 대상이 없는 경우를 위해 변수 세팅
        mouse_info.is_plotted = True
        cur_mask = []

        temp_img = plot(r.orig_img, boxes, masks)
        show(temp_img)
```

### 3.2.2. Inpainting

1) 마우스 클릭으로 객체가 선택되었을 때 마스크 이미지가 있다면 Inpainting 실행한다. 현재 inpainting에 사용할 수 있는 모델은 LaMa, Deepfillv2, MAT가 있다.

```
# 출력할 마스크가 있는 경우 inpainting 실행
else:
    # 마스크 영역 넓힘 -> iterations 횟수 올리면 테두리가 넓어짐
    temp_mask = cv2.dilate(cur_mask, None, iterations=3)

    # inpainting 실행
    #lama
    # result_img = lama_predict(model, predict_config, device, r.orig_img, cur_mask)
    result_img = lama_predict(model, predict_config, device, r.orig_img, temp_mask)
    result_img = cv2.cvtColor(result_img, cv2.COLOR_BGR2RGB)
    show(result_img)

    #deepfillv2
    # result_img = deepfillv2(device, r.orig_img, temp_mask)
    # show(result_img)

    #MAT
    #result_img = MAT(device, r.orig_img, temp_mask)
    #show(result_img)

    y_pred = torch.Tensor(result_img).reshape(3, 640, -1).unsqueeze(0)
    acc = metric(y_pred, y_true).item()
    print("{} ==> {}".format(repr(metric), acc))
```

2) 기존 Inpainting 모델로 predict할 때 이미지 파일을 입력 값으로 받았지만 Inpaiting 처리 속도 향상을 위해 numpy 배열을 입력 값으로 받을 수 있게 구현하였다.

**LaMa**

```
# ================================================================================
# lama predict를 위한 코드
class InpaintingDataset(Dataset):
    def __init__(self, datadir, name, origin, mask, img_suffix='.jpg'):
        self.datadir = datadir
        self.mask_filenames = {name}
        self.img_filenames = {f'{name}_mask.png'}
        self.origin = origin
        self.mask = mask

    def __len__(self):
        return len(self.mask_filenames)

    def __getitem__(self, i):
        image = load_image(self.origin, mode='RGB')
        mask = load_image(self.mask, mode='L')
        result = dict(image=image, mask=mask[None, ...])

        return result

def make_dataset(origin, mask, out_size=512, transform_variant='default', **kwargs):
    if transform_variant is not None:
        transform = get_transforms(transform_variant, out_size)

    dataset = InpaintingDataset('', 1, origin, mask)

    return dataset

def load_image(img, mode='RGB', return_orig=False):
    if img.all() == None : return None

    if img.ndim == 3:
        img = np.transpose(img, (2, 0, 1))
    out_img = img.astype('float32') / 255
    if return_orig:
        return out_img, img
    else:
        return out_img
```

기존 LaMa repository에서 predict하는 부분과 LaMa에서 사용하는 데이터셋 클래스인 InpaintingDataset과 numpy 배열을 가공하는 load_image, 데이터셋을 구성하는 make_dataset 메서드 부분을 따로 추출하였다.

**Deepfillv2**

```
def deepfillv2(device, org_img, mask_img):
    generator_state_dict = torch.load(DEEFILLIV2_PATH)['G']

    if 'stage1.conv1.conv.weight' in generator_state_dict.keys():
        from model.networks import Generator
    else:
        from model.networks_tf import Generator

     # set up network
    generator = Generator(cnum_in=5, cnum=48, return_flow=False).to(device)

    generator_state_dict = torch.load(DEEFILLIV2_PATH)['G']
    generator.load_state_dict(generator_state_dict, strict=True)

    # prepare input
    image = T.ToTensor()(org_img)
    mask = T.ToTensor()(mask_img)

    _, h, w = image.shape
    grid = 8

    image = image[:3, :h//grid*grid, :w//grid*grid].unsqueeze(0)
    mask = mask[0:1, :h//grid*grid, :w//grid*grid].unsqueeze(0)

    print(f"Shape of image: {image.shape}")

    image = (image*2 - 1.).to(device)  # map image values to [-1, 1] range
    mask = (mask > 0.5).to(dtype=torch.float32,
                           device=device)  # 1.: masked 0.: unmasked

    image_masked = image * (1.-mask)  # mask image

    ones_x = torch.ones_like(image_masked)[:, 0:1, :, :]
    x = torch.cat([image_masked, ones_x, ones_x*mask],
                  dim=1)  # concatenate channels

    with torch.inference_mode():
        _, x_stage2 = generator(x, mask)

    # complete image
    image_inpainted = image * (1.-mask) + x_stage2 * mask

    # save inpainted image
    img_out = ((image_inpainted[0].permute(1, 2, 0) + 1)*127.5)
    img_out = img_out.to(device='cpu', dtype=torch.uint8)

    return img_out.numpy()
```

기존 이미지 이름으로 파일에 접근하여 predict 후 이미지를 저장하는 방식에서 원본 이미지와 마스크 이미지의 numpy 배열을 매개변수로 받아 predict 후 inpainting 결과 numpy 배열을 반환하도록 수정하였다.

**MAT**

```
def MAT(device, org_img, mask_img):
    resolution = 512
    truncation_psi = 1
    noise_mode = 'const'

    network_pkl = '../MAT/pretrained/Places.pkl'
    with dnnlib.util.open_url(network_pkl) as f:
        G_saved = legacy.load_network_pkl(f)['G_ema'].to(device).eval().requires_grad_(False) # type: ignore
    net_res = 512 if resolution > 512 else resolution
    G = Generator(z_dim=512, c_dim=0, w_dim=512, img_resolution=net_res, img_channels=3).to(device).eval().requires_grad_(False)
    copy_params_and_buffers(G_saved, G, require_all=True)
    label = torch.zeros([1, G.c_dim], device=device)

    mask2 = 1 - mask_img

    img_512 = cv2.resize(org_img, (512,512), interpolation=cv2.INTER_LINEAR)
    mask_512 = cv2.resize(mask2, (512,512), interpolation=cv2.INTER_LINEAR)

    img = (torch.from_numpy(img_512).float().to(device) / 127.5 - 1).unsqueeze(0)
    img = img.permute(0, 3, 1, 2)
    mask = torch.from_numpy(mask_512).float().to(device).unsqueeze(0).unsqueeze(0)


    with torch.no_grad():
        z = torch.from_numpy(np.random.randn(1, G.z_dim)).to(device)
        output = G(img, mask, z, label, truncation_psi=truncation_psi, noise_mode=noise_mode)
        output = (output.permute(0, 2, 3, 1) * 127.5 + 127.5).round().clamp(0, 255).to(torch.uint8)
        output = output[0].cpu().numpy()
        #output = cv2.cvtColor(output, cv2.COLOR_RGB2BGR)

    return output
```

기존 mask 이미지를 불러와 GRAYSCALE로 바꾼 후 predict 하는 방식에서 매개변수에 담긴 0~1 범위의 mask numpy 배열을 반전하고 512x512 크기의 이미지로 resize하여 inpainting 결과 numpy 배열을 반환하도록 수정하였다.

# 4. 결과 및 개선 사항

## 4.1. 결과

| LaMa | Deepfillv2 | MAT |
|---|---|---|
|  |  |  |

LaMa와 Deepfillv2는 선택한 객체를 없애고 자연스럽게 채운 모습을 볼 수 있다. MAT는 선택한 객체를 주변 사물 정보를 이용하여 채운 모습을 볼 수 있다. 하지만 객체의 edge가 두드러지고 객체의 모양이 남아있는 것을 볼 수 있다.

| | LaMa | Deepfillv2 | MAT |
|---|---|---|---|
| PSNR | 27.586 | 25.378 | 22.120 |
| SSIM | 0.931 | 0.908 | 0.890 |
| LPIPS | 0.045 | 0.054 | 0.065 |

화질 손실량(PSNR, SSIM), 특징점 유사도(LPIPS)를 비교했을 때 LaMa 모델을 사용한 결과가 모두 좋은 성능을 보였다. MAT는 PSNR, SSIM는 LaMa와 Deepfillv2와 비슷하게 나와 화질 손상 정도는 나머지 모델과 비슷하나 LPIPS에서 약 2.5배의 차이를 보여 사람 눈에 그럴듯한 이미지를 출력하는데 부족함을 보였다.

## 4.2. 개선 사항

1) Dilate 연산

| Dilate 연산 전 | Dilate 연산 후 |
| --- | --- |
|  |  |

YOLOv8의 segmentation mask를 그대로 사용하면 inpainting으로 채운 부분에서 마스크의 edge가 드러나게 결과가 나온 것을 볼 수 있었다. 마스크 영역을 확장하면 inpainting quality 향상에 도움이 될 수 있다는 아이디어를 얻어 필터 내부의 가장 높은 값으로 변환(or)하는 팽창 연산인 **dilate** 연산을 통해 마스크 영역을 확장하면서 마스크의 edge가 두드러지는 현상을 완화시켰다. 이를 Deepfillv2와 MAT에도 적용했을 때도 마찬가지로 edge가 덜 뚜렷하게 보이는 것을 확인할 수 있었다.

2) 개체 선택 방법

기존에는 YOLOv8이 인식한 객체 중 선택한 개체의 클래스 번호를 저장하여 클래스 번호와 일치하는 개체만을 지우도록 구현하였다. 이와 같이 구현하면 서로 다른 개체지만 같은 클래스로 묶이는 개체는 구분이 불가능하다. 이를 해결하기 위해 YOLOv8에서 Image Segmentation할 때 클래스로 분류 전 감지된 객체들에 부여되는 index로 구분하도록 구현하였다.

# 5. 결론 및 향후 연구 방향

## 5.1. 결론

실시간 image segmentation 모델인 YOLOv8과 이미지 및 동영상 inpainting 모델(LaMa, deepfillv2, MAT)을 활용하여 실시간 개체 inpainting 모델을 구축할 수 있었다. 모델들을 비교했을 때 LaMa가 가장 높은 정확도와 성능을 보여주었다.

마스크 이미지의 영역을 넓힘으로써 inpainting 결과를 좀 더 자연스럽게 할 수 있었고, image segmentation 시 감지된 개체들에게 부여되는 index를 통해 개체를 선택할 수 있게 구현하여 프로그램의 성능을 높힐 수 있었다.

## 5.2. 향후 연구 방향

마스크 이미지의 영역을 넓힘으로써 inpainting의 결과물이 자연스러워졌지만 경계선이 모호해지는 대신 blur 현상이 나타나 quality 향상을 위해 다른 방법을 모색할 필요가 있다. 객체 분류 방식으로 labeling 전 index를 사용하는 방법도 확실한 개체 구분을 위해 index에만 의존하는 방식에서 개선책이 필요하다. 또한 YOLO는 객체의 그림자 부분까지는 인식하지 않아 객체를 지우더라도 그림자 때문에 실재감을 증대하는데 어려움이 있다. Image segmentation 과정에서 그림자 영역까지 mask를 만드는 방법을 고안하면 실재감을 증대하는데 더욱 효과적일 것으로 기대한다.

## 6. 멘토의견서에 대한 반영 및 답변

개발하고자 하는 주제가 증대인지 유지인지 불분명한 부분에서 증대를 방향으로 잡아 Inpainting 결과물의 퀄리티를 높이기 위해 노력하였다. Dilate 연산을 적용하면 마스크 영역을 넓혀 결과물의 퀄리티를 높일 수 있었다.

차후 연구 방향은 객체를 구별하는 방법에 대해 연구하고, 학습되지 않은 물체를 구별할 수 있게 구현하려고 한다. 또 Inpainting한 결과에 AR 기술을 접목하여 실재감 증대를 목표로 하려고 한다.

## 7. 시연계획

프로그램의 속도를 높이기 위해 GPU사용을 교내 자연대 연구실험동에 있는 XRLab의 컴퓨터에 원격으로 연결하고 프로그램을 실행시킬 계획이다. 프로젝트가 실시간으로 Inpainting하는 결과물을 보일 필요가 있기 때문에 드로이드캠(DroidCam)을 이용하여 6공학관에서 실시간으로 영상을 송출하여 결과를 확인할 수 있도록 계획하였다.

# 8. 개발 일정 및 역할 분담

## 8.1. 개발 일정

<table>
<tr><td colspan="2">5월</td><td colspan="4">6월</td><td colspan="4">7월</td><td colspan="5">8월</td><td colspan="2">9월</td></tr>
<tr><td>4주</td><td>5주</td><td>1주</td><td>2주</td><td>3주</td><td>4주</td><td>1주</td><td>2주</td><td>3주</td><td>4주</td><td>1주</td><td>2주</td><td>3주</td><td>4주</td><td>5주</td><td>1주</td><td>2주</td></tr>
<tr><td colspan="2">착수<br>보고서</td><td colspan="2"></td><td colspan="2"></td><td colspan="2"></td><td colspan="2"></td><td colspan="2"></td><td colspan="2"></td><td colspan="2"></td><td colspan="2"></td><td></td></tr>
<tr><td colspan="4">유니티 파이썬 연동</td><td colspan="2"></td><td colspan="2"></td><td colspan="2"></td><td colspan="2"></td><td colspan="2"></td><td colspan="2"></td><td colspan="2"></td><td></td></tr>
<tr><td colspan="2"></td><td colspan="2"></td><td colspan="4">화면상 객체 선택과<br>Image Segmentation 구현</td><td colspan="4"></td><td colspan="2"></td><td colspan="2"></td><td colspan="2"></td><td></td></tr>
<tr><td colspan="2"></td><td colspan="2"></td><td colspan="2"></td><td colspan="4"></td><td colspan="4">Inpainting 적용 및 실험</td><td colspan="2"></td><td></td></tr>
<tr><td colspan="2"></td><td colspan="2"></td><td colspan="2"></td><td colspan="4"></td><td colspan="4"></td><td colspan="4">최종 발표<br>보고서 준비</td></tr>
</table>

## 8.2. 역할 분담

| | |
|---|---|
| 천형주 | Yolo v8과 OpenCV를 이용하여 객체를 직접 선택해<br>Image Segmentation하는 코드 구현<br>Deepfillv2, MAT 모델 적용 |
| 민예진 | 마우스 클릭 로직 구현<br>Inpainting 모델 서칭 및 LaMa 모델 적용 |
| 공통 | 포스터, 최종보고서, 시연 보고서 작성 및 발표 |

## 9. 참고 문헌

논문 내용에 직접 관련이 있는 문헌에 대해서는 관련이 있는 본문 중에 참고문헌 번호를 쓰고 그 문헌을 참고문헌란에 인용 순서대로 기술한다. 참고문헌은 영문으로만 표기하며 학술지의 경우에는 저자, 제목, 학술지명, 권, 호, 쪽수, 발행년도의 순으로, 단행본은 저자, 도서명, 발행소, 발행년도의 순으로 기술한다.


[1] Park, Sang-Yong ; Heo, Yong-Seok. Semantic Segmentation Technology Trend Analysis using Deep Learning, The proceedings of KIEE, v.67, no.7, pp.18, 2018. (in Korean)

[2] Redmon, J., Divvala, S., Girshick, R., & Farhadi, A. "You only look once: Unified, real-time object detection, *Computer Vision and Pattern Recognition (CVPR)* , pp. 779-788, 2016.

[3] qtly_u. (2023, March 8). [YOLO] Yolo Version - from Yolo v1 to Yolo v8 (as of 23.03). Available: https://velog.io/@qtly_u/n4ptcz54 (downloaded 2023, October. 17)

[4] Omar Elharrouss, Noor Almaadeed, Somaya Al-Maadeed, Younes Akbari. Image inpainting: A review, Neural Processing Letters, 51, 2020.

[5] Suvorov, R., Logacheva, E., Mashikhin, A., Remizova, A., Ashukha, A., Silvestrov, A., Naejin Kong, Harshith Goka, Kiwoong Park, Lempitsky, V., "Resolution-robust large mask inpainting with fourier convolutions," *IEEE/CVF Winter Conference on Applications Computer Vision (WACV)*, pp. 2149-2159, 2022.

[6] Yu, J., Lin, Z., Yang, J., Shen, X., Lu, X., Huang, T. S., "Free-form image inpainting with gated convolution," *IEEE/CVF International Conference on Computer Vision (ICCV)*, pp. 4471-4480, 2019.

[7] Li, W., Lin, Z., Zhou, K., Qi, L., Wang, Y., Jia, J., "Mat: Mask-aware transformer for large hole image inpainting,", *IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 10758-10768, 2022.