# Final Project

## Encoder-Decoder implementation


**Open Software Project**

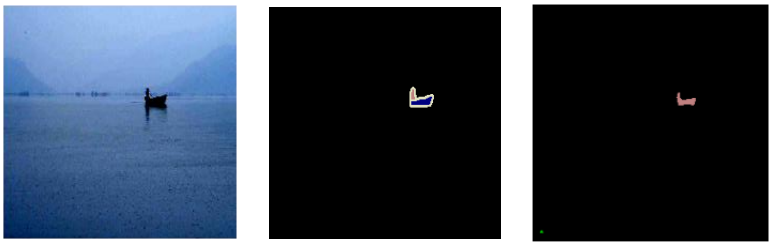
**1829008**

**김민영**

**[Technical Report] : Encorder – Decorder implementation**

**The Subject of Final Project**

본 프로젝트에서는 Pascal VOC 2012 dataset을 사용하여, Original U-Net, U-Net with ResNet Encorder를 구현하여 Image Segment를 한다.
목적은 input data로 label에 가깝게 image segmentation 하는 모델을 만드는 것이다.
코드의 사용방법은 첨부된 Readme.txt를 참고하면 된다.



# 목차

## 1. Dataset

본 프로젝트에서 사용한 데이터셋은 Pascal VOC 2012 dataset이다. 우선, PASCAL VOC Dataset을 http://host.robots.ox.ac.uk/pascal/VOC/voc2012/ 에서 다운받으면 다음과 같은 폴더의 구조를 확인할 수 있다.

<VOC2012>


Annotations : JPEGImages의 original image와 같은 이름의 xml파일들 존재. Object Detection을 위한 정답 데이터
ImageSets : 어떤 이미지 그룹을 test, train, trainval, val로 사용할 것인지, 특정 클래스가 어떤 이미지에 있는지 등에 대한 정보들을 포함하고 있는 폴더
JPEGImages : 이미지 파일들이 모여있는 폴더, Object Detection에서 입력 데이터
SegmentationClass : Semantic segmentation을 학습하기 위한 label 이미지
SegmentationObject : Instance segmentation을 학습하기 위한 label 이미지

<Classes> 해당 데이터의 Class 구성은 다음과 같다.
Person, bird, cat, cow, dog, horse, sheep, aeroplane, bicycle, boat, bus, car, motorbike, train, bottle, chair, dining table, potted plant, sofa, TV/monitor

**<dataset.py>**

해당 데이터를 Load 하는 기능을 하는 코드이다. 이들을 Main 에서 불러와 사용할 것이다.

VOCdataloader 함수: JPEGImage 에서 원본 이미지를 불러오고, SegmentClass 에서 mask(=label)을 불러온다.

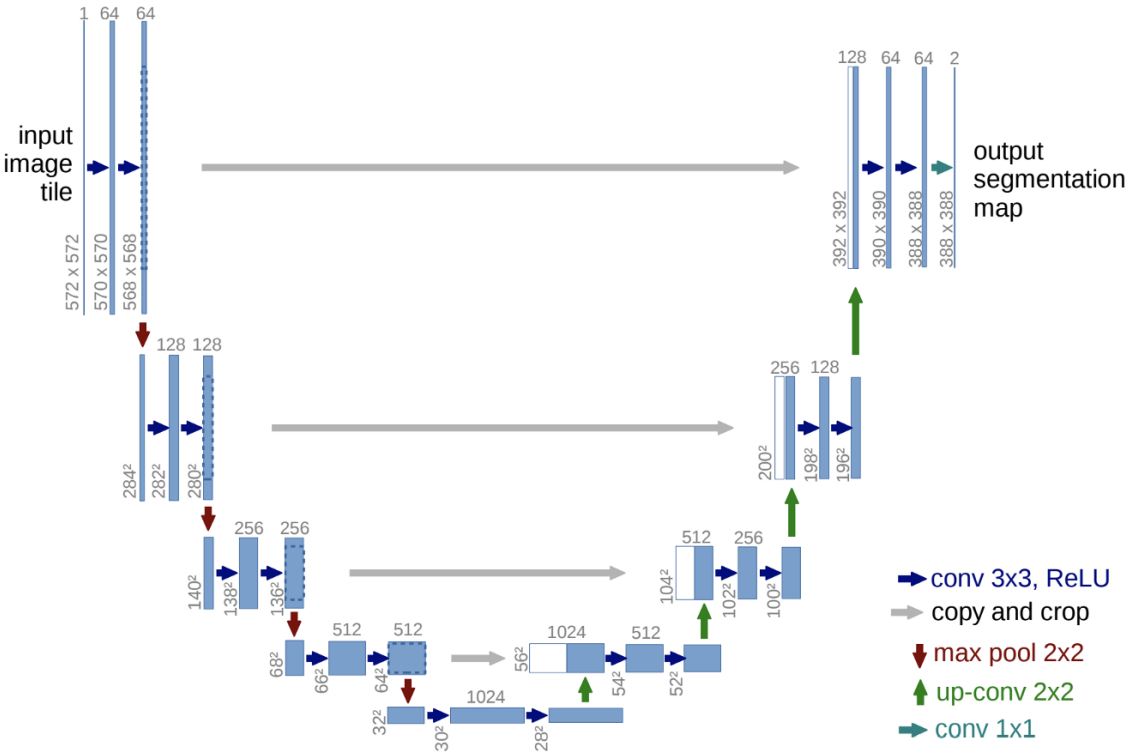
__init__ 함수 : Test data 와 Train data 를 0.2 비율로 나눈다. 이때, 모델을 학습하는데에 많은 시간이 소요되므로, trainLoader 부분에서 50 개의 데이터만을 이용하여 디버깅을 시도해볼 수도 있다.

**(코드생략)**

## 2. Model

### 2-1) Original U-Net model

Unet.py 에서는 U-Net을 구현하는 코드를 구현하였다.




**Olaf Ronneberger, Philipp Fischer, and Thomas Brox**

**"U-Net: Convolutional Networks for Biomedical Image Segmentation."**

우선 conv 함수이다.

```
def conv(in_channels, out_channels):
    return nn.Sequential(
        nn.Conv2d(in_channels, out_channels, 3, padding=1),  # 3은 kernel size
        nn.BatchNorm2d(out_channels),
        nn.ReLU(inplace=True),
        nn.Conv2d(out_channels, out_channels, 3, padding=1),
        nn.BatchNorm2d(out_channels),
        nn.ReLU(inplace=True)
    )
```

Convolution -> ReLU 과정을 두번 거친다. 이때, 중간에 BatchNormalize 를 해줄 수 있다.

다음으로 Unet class 이다.

```
class Unet(nn.Module):
    def __init__(self, in_channels, out_channels):
        super(Unet, self).__init__()

        self.convDown1 = conv(in_channels, 64)
        self.convDown2 = conv(64, 128)
        self.convDown3 = conv(128, 256)
        self.convDown4 = conv(256, 512)
        self.convDown5 = conv(512, 1024)
        self.maxpool = nn.MaxPool2d(2, stride=2)
        self.upsample = nn.Upsample(scale_factor=2, mode='bilinear', align_corner
s=True)
        self.convUp4 = conv(1536, 512)
        self.convUp3 = conv(768, 256)
        self.convUp2 = conv(384, 128)
        self.convUp1 = conv(192, 64)
        self.convUp_fin = nn.Conv2d(64, out_channels, 1)
```

위의 그림에서와 같이, 초기화를 해준다. 다만 여기서 Up sample 할 때에 채널사이즈를 줄이지 않는 방법으로 하기 때문에 convUp 부분에 1024+512, 512+256, 128+64 channel 을 넣어준다.

* 방법 2) 만약, Up sample 할 때에 채널사이즈를 줄이는 방법으로 한다면 아래와 같은 수를 넣어주면된다.

```
self.convUp4 = conv(1024, 512)
self.convUp3 = conv(512, 256)
self.convUp2 = conv(256, 128)
self.convUp1 = conv(128, 64)
self.convUp_fin = nn.Conv2d(64, out_channels, 1)
```

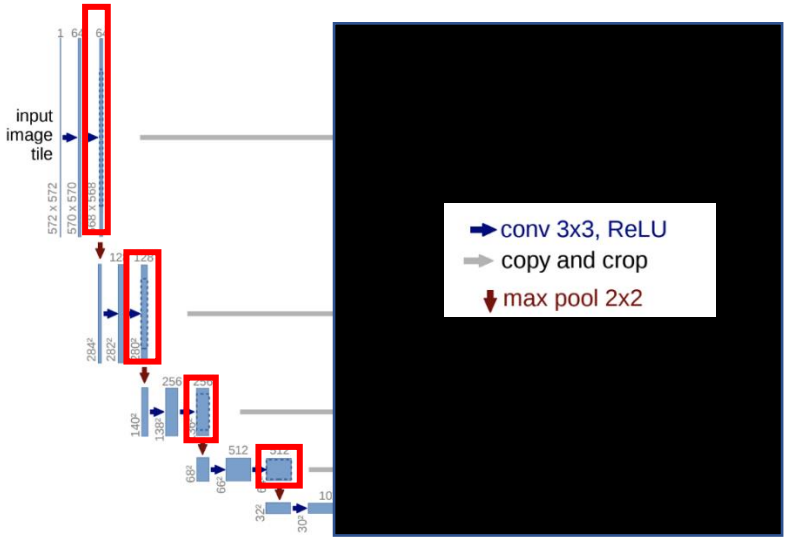
다음으로는 forward 함수이다. Encoder 부분에서는 위에서의 U-Net 그림에서처럼 ConvDown-> maxpooling 과정을 반복한다.

```
    def forward(self, x):
        conv1 = self.convDown1(x)
        x = self.maxpool(conv1)
        conv2 = self.convDown2(x)
        x = self.maxpool(conv2)
        conv3 = self.convDown3(x)
        x = self.maxpool(conv3)
        conv4 = self.convDown4(x)
        x = self.maxpool(conv4)
        conv5 = self.convDown5(x)

        x = self.upsample(conv5)
        x = torch.cat([x, conv4], 1)
        x = self.convUp4(x)
        x = self.upsample(x)
        x = torch.cat([x, conv3], 1)
        x = self.convUp3(x)
        x = self.upsample(x)
        x = torch.cat([x, conv2], 1)
        x = self.convUp2(x)
        x = self.upsample(x)
        x = torch.cat([x, conv1], 1)
        x = self.convUp1(x)
        out = self.convUp_fin(x)

        return out
```

encoder

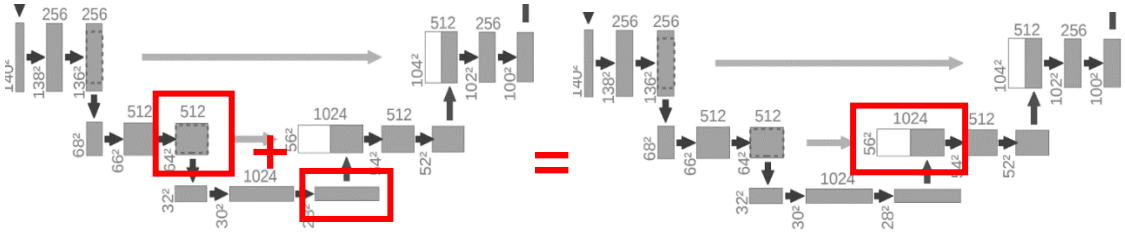
decoder

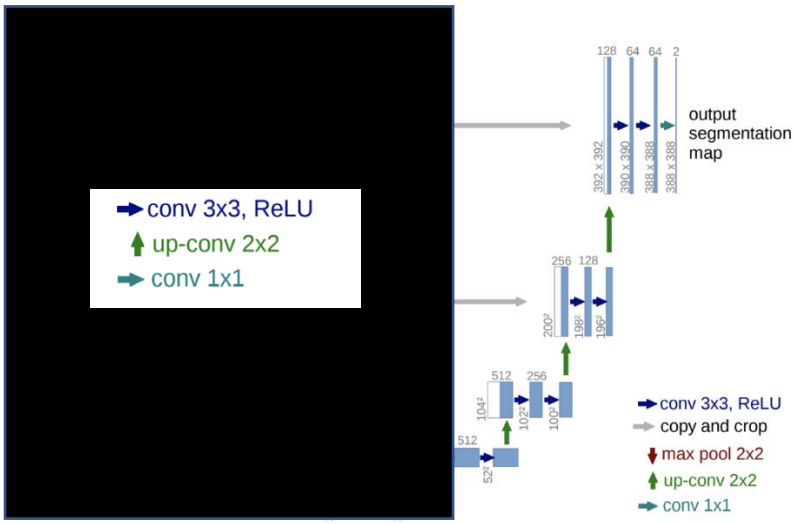
conv 함수를 통해 Convolution -> ReLU 과정을 두번 거치게 되면, conv1 을 얻게된다. 이는 나중에 copy and crop 과정에서 합쳐야하므로 저장해둔다. 이후, max pooling 을 진행하고 이 과정을 conv1 과 마찬가지로 conv2 conv3, conv4 도 반복한다.




-> encoder

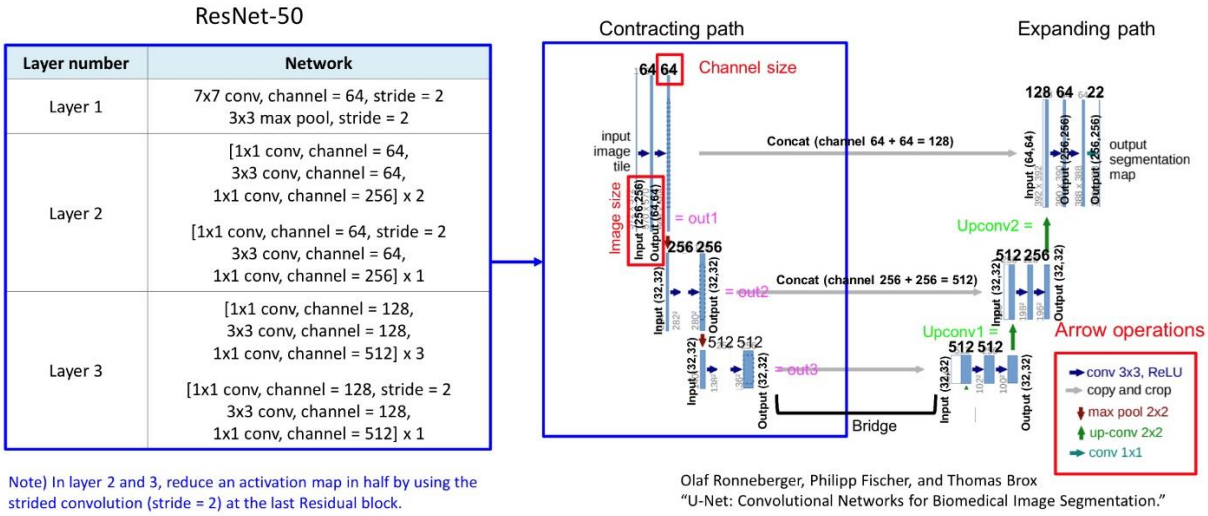
이후 Decoder 부분에서는 먼저 아래 그림과 같은 concat 과정이 필요하다. 앞서 저장해놓은 conv1-4 와 바로 이전의 결과를 합친다. 이후 Convolution -> ReLU 과정을 두번 거치고 다시 up-conv 를 수행한다.





## 2-2) U-Net with ResNet Encoder

두번째로는 Resnet50 을 이용하여 U-net 을 구현하였다. 전체적인 과정은 아래와 같다.




먼저, convolution 부분이다. Conv1x1, Cov3x3 함수를 다음과 같이 구현하여 놓는다.

```
def conv1x1(in_channels, out_channels, stride, padding):
    model = nn.Sequential(
        nn.Conv2d(in_channels, out_channels, kernel_size=1, stride=stride, padding=padding),
        nn.BatchNorm2d(out_channels),
        nn.ReLU(inplace=True)
    )
    return model

def conv3x3(in_channels, out_channels, stride, padding):
    model = nn.Sequential(
        nn.Conv2d(in_channels, out_channels, kernel_size=3, stride=stride, padding=padding),
        nn.BatchNorm2d(out_channels),
        nn.ReLU(inplace=True)
    )
    return model
```

이후 ResidualBlock 클래스를 다음과 같이 정의할 수 있다. 이는 앞선 과제의 ResNet50 을 참고하여 구현하였다.

```
class ResidualBlock(nn.Module):
    def __init__(self, in_channels, middle_channels, out_channels, downsample=False):
        super(ResidualBlock, self).__init__()
        self.downsample = downsample

        if self.downsample:
            self.layer = nn.Sequential(
                conv1x1(in_channels,middle_channels,2,0),
                conv3x3(middle_channels,middle_channels,1,1),
                conv1x1(middle_channels,out_channels,1,0)
            )
            self.downsize = conv1x1(in_channels, out_channels, 2, 0)

        else:
            self.layer = nn.Sequential(
                conv1x1(in_channels,middle_channels,1,0),
                conv3x3(middle_channels,middle_channels,1,1),
                conv1x1(middle_channels,out_channels,1,0)
            )
            self.make_equal_channel = conv1x1(in_channels, out_channels, 1, 0)
```

```
    def forward(self, x):
        if self.downsample:
            out = self.layer(x)
            x = self.downsize(x)
            return out + x
        else:
            out = self.layer(x)
            if x.size() is not out.size():
                x = self.make_equal_channel(x)
            return out + x
```

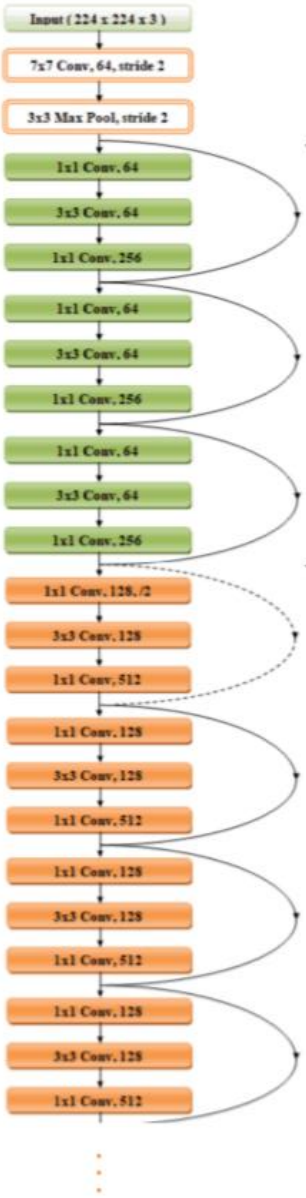
```
def conv(in_channels, out_channels):
    return nn.Sequential(
        nn.Conv2d(in_channels, out_channels, 3, padding=1),  # 3은 kernel size
        nn.BatchNorm2d(out_channels),
        nn.ReLU(inplace=True),  # inplace가 TRUE인 것은 할당 없이 바로 적용하겠다.
        nn.Conv2d(out_channels, out_channels, 3, padding=1),
        nn.BatchNorm2d(out_channels),
        nn.ReLU(inplace=True)
    )
```

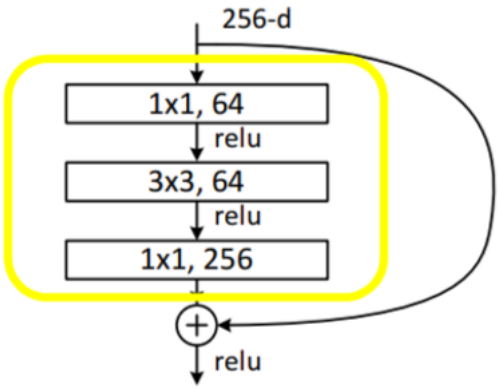
Unet with Resnet50 Encoder 모델은 다음과 같이 정의할 수 있다.

```
class UNetWithResnet50Encoder(nn.Module):
    def __init__(self, n_classes=22):
        super().__init__()
        self.n_classes = n_classes
        self.layer1 = nn.Sequential(
            nn.Conv2d(3, 64, kernel_size=7, stride=2, padding=3), # Code overlaps with prev
ious assignments
            nn.BatchNorm2d(64),
            nn.ReLU(inplace=True)#,
        )
        self.pool = nn.MaxPool2d(3, 2, 1, return_indices=True)

        self.layer2 = nn.Sequential(
            ResidualBlock(64, 64, 256, False),
            ResidualBlock(256, 64, 256, False),
            ResidualBlock(256, 64, 256, True)
        )
        self.layer3 = nn.Sequential(
            ResidualBlock(256,128,512, False),
            ResidualBlock(512,128,512, False),
            ResidualBlock(512,128,512, False),
            ResidualBlock(512,128,512, False),
        )
        self.bridge = conv(512, 512)
        self.UnetConv1 = conv(512, 256)
        self.UpConv1 = nn.Conv2d(512, 256, 3, padding=1)

        self.upconv2_1 = nn.ConvTranspose2d(256, 256, 3, 2, 1)
        self.upconv2_2 = nn.Conv2d(256, 64, 3, padding=1)

        self.unpool = nn.MaxUnpool2d(3, 2, 1)
        self.UnetConv2_1 = nn.ConvTranspose2d(64, 64, 3, 2, 1)
        self.UnetConv2_2 = nn.ConvTranspose2d(128, 128, 3, 2, 1)
        self.UnetConv2_3 = nn.Conv2d(128, 64, 3, padding=1)

        self.UnetConv3 = nn.Conv2d(64, self.n_classes, kernel_size=1, stride=1)
```

```
 def forward(self, x, with_output_feature_map=False): #256

        out1 = self.layer1(x)
        out1, indices = self.pool(out1)
        out2 = self.layer2(out1)
        out3 = self.layer3(out2)
        x = self.bridge(out3) # bridge
        x = self.UpConv1(x)
        x = torch.cat([x,out2],dim=1)
        x = self.UnetConv1(x)
```

```
        x = self.upconv2_1(x, output_size=torch.Size([x.size(0),256,64,64]))
        x = self.upconv2_2(x)
        x = torch.cat([x,out1],dim=1)
        x = self.UnetConv2_2(x, output_size=torch.Size([x.size(0), 128, 128, 128]))
        x = self.UnetConv2_2(x, output_size=torch.Size([x.size(0), 128, 256, 256]))
        x = self.UnetConv2_3(x)
        x = self.UnetConv3(x)
        return x
```




**K. He, X. Zhang, S. Ren, and J. Sun.**
**"Deep residual learning for image recognition." In CVPR, 2016**

## 3. Module.py

우선, 모델을 Train 시키는 과정이다.

```
def train_model(trainloader, model, criterion, optimizer,scheduler, device):
    model.train()
    for i, (inputs, labels) in enumerate(trainloader):
        from datetime import datetime
        inputs = inputs.to(device)
        labels = labels.to(device=device, dtype=torch.int64)
        criterion = criterion.cuda()

        #Get the output out of model, and Get the loss
        outputs = model(inputs)
        loss = criterion(outputs, labels)

        #Optimizer & backpropagation
        optimizer.zero_grad()
        loss.backward()
        optimizer.step()

        #train_loss += loss.item()
```

Accuracy_check, accuracy_check_for_batch, get_loss_train 함수에서는 출력할 정확도와 Loss 를 계산한다.

```
def accuracy_check(label, pred):
    ims = [label, pred]
    np_ims = []
    for item in ims:
        item = np.array(item)
        np_ims.append(item)
    compare = np.equal(np_ims[0], np_ims[1])
    accuracy = np.sum(compare)
    return accuracy / len(np_ims[0].flatten())

def accuracy_check_for_batch(labels, preds, batch_size):
    total_acc = 0
    for i in range(batch_size):
        total_acc += accuracy_check(labels[i], preds[i])
    return total_acc/batch_size
```

```
def get_loss_train(model, trainloader, criterion, device):

    model.eval()
    total_acc = 0
    total_loss = 0
    for batch, (inputs, labels) in enumerate(trainloader):
        with torch.no_grad():
            inputs = inputs.to(device)
            labels = labels.to(device = device, dtype = torch.int64)
            inputs = inputs.float()

            outputs = model(inputs)
            loss = criterion(outputs,labels)

            outputs = np.transpose(outputs.cpu(), (0,2,3,1))
            preds = torch.argmax(outputs, dim=3).float()
            acc = accuracy_check_for_batch(labels.cpu(), preds.cpu(), inputs.size()[0])
            total_acc += acc
            total_loss += loss.cpu().item()
    return total_acc/(batch+1), total_loss/(batch+1)
```

Val_model 함수에서는 image, predicted result 의 segmentation mask 를 cls_invert[] 를 이용하여 r,g,b 로 변환하고 이 결과물을 저장한다.

```
def val_model(model, valloader, criterion, device, dir):

    cls_invert = {0: (0, 0, 0), 1: (128, 0, 0), 2: (0, 128, 0),
                  3: (128, 128, 0), 4: (0, 0, 128), 5: (128, 0, 128),
                  6: (0, 128, 128), 7: (128, 128, 128), 8: (64, 0, 0),
                  9: (192, 0, 0), 10: (64, 128, 0), 11: (192, 128, 0),
                  12: (64, 0, 128), 13: (192, 0, 128), 14: (64, 128, 128),
                  15: (192, 128, 128), 16: (0, 64, 0), 17: (128, 64, 0),
                  18: (0, 192, 0), 19: (128, 192, 0), 20: (0, 64, 128),
                  21: (224, 224, 192)}
    total_val_loss = 0
    total_val_acc = 0
    n=0
    for batch, (inputs, labels) in enumerate(valloader):
        with torch.no_grad():
            inputs = inputs.to(device)
            labels = labels.to(device=device, dtype=torch.int64)

            outputs = model(inputs)
            loss = criterion(outputs,labels)

            outputs = np.transpose(outputs.cpu(), (0, 2, 3, 1))
            preds = torch.argmax(outputs, dim=3).float()

            acc = accuracy_check_for_batch(labels.cpu(), preds.cpu(), inputs.size()[0])

            total_val_acc += acc
            total_val_loss += loss.cpu().item()

            for i in range(preds.shape[0]):
                temp = preds[i].cpu().data.numpy()
                temp_l = labels[i].cpu().data.numpy()
                temp_rgb = np.zeros((temp.shape[0], temp.shape[1], 3))
                temp_label = np.zeros((temp.shape[0], temp.shape[1], 3))

                for j in range(temp.shape[0]):
                    for k in range(temp.shape[1]):
                        temp_rgb[k,j,0] = cls_invert[temp[k,j]][0]
                        temp_rgb[k,j,1] = cls_invert[temp[k,j]][1]
                        temp_rgb[k,j,2] = cls_invert[temp[k,j]][2]

                        temp_label[k,j,0] = cls_invert[temp_l[k,j]][0]
                        temp_label[k,j,1] = cls_invert[temp_l[k,j]][1]
                        temp_label[k,j,2] = cls_invert[temp_l[k,j]][2]


                img = inputs[i].cpu()
                img = np.transpose(img, (2, 1, 0))

                img_print = Image.fromarray(np.uint8(temp_label))
                mask_print = Image.fromarray(np.uint8(temp_rgb))

                img_print.save(dir + str(n) + 'label' + '.png')
                mask_print.save(dir + str(n) + 'result' + '.png')

                n += 1

    return total_val_acc/(batch+1), total_val_loss/(batch+1)
```

## 4. Main.py

우선 transforms 를 정의하고, 앞서 dataset.py 에서 구현한 함수들을 통해 train dataset 과 val dataset 을 불러온다.

```
transforms = transforms.Compose([
    transforms.ToPILImage(),
    transforms.Resize([resize_size,resize_size], PIL.NEAREST),
    transforms.ToTensor(),
    transforms.Normalize((0.485, 0.456, 0.406), (0.229, 0.224, 0.225))
])

trainset = Loader(data_dir, flag ='train', resize = resize_size, transforms = tra
nsforms)
valset = Loader(data_dir, flag = 'val', resize = resize_size, transforms = transf
orms)

trainLoader = DataLoader(trainset, batch_size = 16, shuffle=True)
validLoader = DataLoader(valset, batch_size = 16, shuffle=True)
```

이후 앞서 Unet.py 에서 구현한 Unet 모델을 불러오고 Loss function, optimizer 를 정의한다. 여기서 사용한 Loss 는 CrossEntropyLoss 이고, optimizer 는 Adam 이다. 또한 모델파라미터를 저장하여준다.

```
model = Unet(3,22)

# Loss Function
criterion = nn.CrossEntropyLoss()

# Optimizer
optimizer = torch.optim.Adam(model.parameters(),lr=learning_rate)
scheduler = StepLR(optimizer, step_size=4, gamma=0.1)

# parameters
epochs = 40

device = torch.device('cuda:0' if torch.cuda.is_available() else 'cpu')
model = model.to(device)

PATH = '/content/drive/My Drive/openSWproject/trained_model/UNet_trained_model.pt
h'
checkpoint = torch.load(PATH,map_location=torch.device('cuda:0'))
```

다음으로 Train 부분이다. History 에 예측한 결과를 저장한다.

```
for epoch in range(epochs):
    train_model(trainLoader, model, criterion, optimizer, scheduler, device)
    train_acc, train_loss = get_loss_train(model, trainLoader, criterion, device)

    predict_save_folder = predict_save_dir + 'epoch' + str(epoch) + '/'
    createFolder(predict_save_folder)
    val_acc, val_loss = val_model(model, validLoader, criterion, device, predict_
save_folder)

    history['train_loss'].append(train_loss)
    history['train_acc'].append(train_acc)
    history['val_loss'].append(val_loss)
    history['val_acc'].append(val_acc)

    if epoch % 4 == 0:
        savepath2 = savepath1 + str(epoch) + ".pth"
        torch.save(model.state_dict(),savepath2)
```

## 5. 결과

**- Unet**

Train 50 개, Test 10 개의 데이터를 사용한 결과)


**- ResNet** (epoch-4)

Train 50 개, Test 10 개의 데이터를 사용한 결과)

```
trainset
valset
tainLoader
valLoader
Training
train_model
epoch 1 train loss :  2.165231019258499 train acc :  0.6713337898254395
epoch 1 val loss :  1.4114373922348022 val acc :  0.8029693603515625
train_model
epoch 2 train loss :  1.597087800502777 train acc :  0.711843729019165
epoch 2 val loss :  1.3182076215744019 val acc :  0.8029693603515625
train_model
epoch 3 train loss :  2.502751648426056 train acc :  0.6765291690826416
epoch 3 val loss :  1.6299495697021484 val acc :  0.8029693603515625
train_model
epoch 4 train loss :  2.110670268535614 train acc :  0.6068897247314453
epoch 4 val loss :  1.9500147104263306 val acc :  0.8029693603515625
```

**결과물 일부)**

**Original Image** | **Label Image** | **Result image**