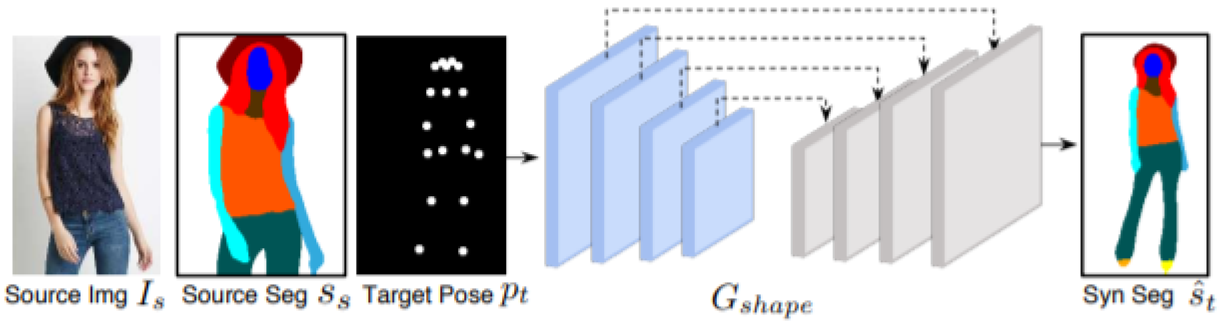
ClothFlow

- despite great improvements have been achieved by these two geometric modeling techniques, they only have limited degrees of freedom

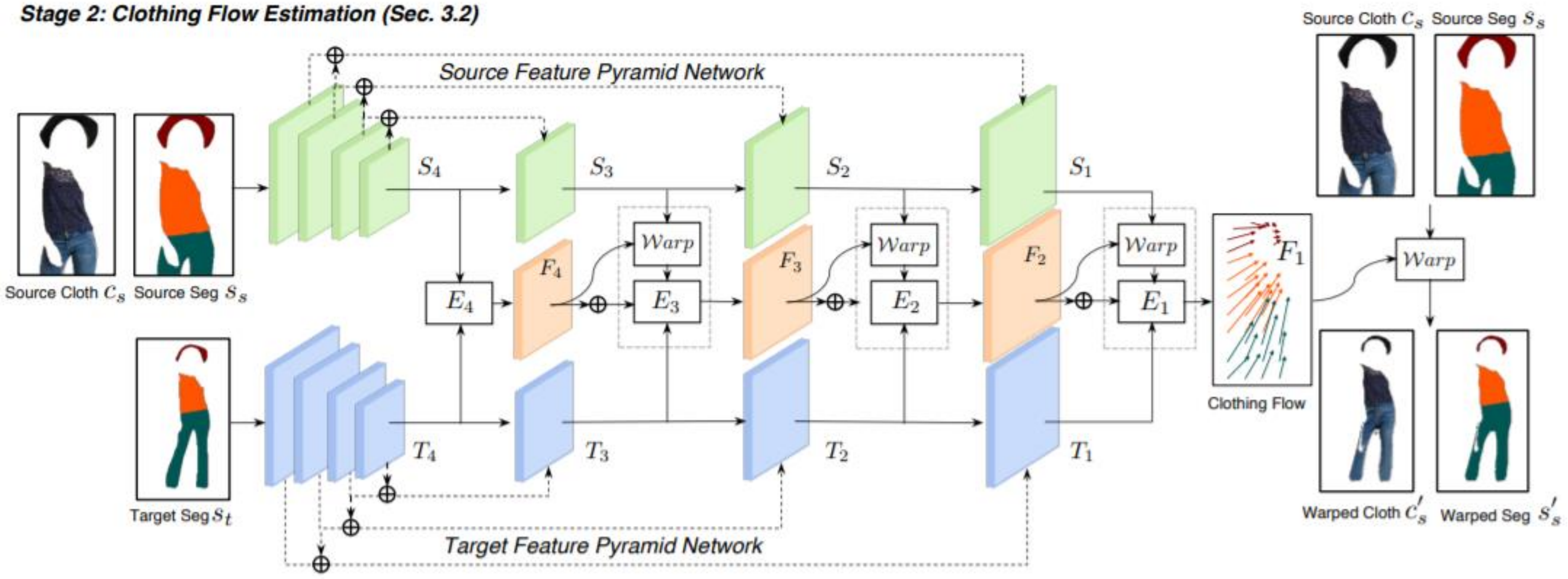
Stage 1: Conditional Layout Generation (Sec. 3.1)



$$\hat{s}_t = G_{layout}(I_s, s_s, p_t)$$

pixel-wise cross entropy loss between s_t and $\hat{s}_t$

clothes are highly deformable with large misalignment

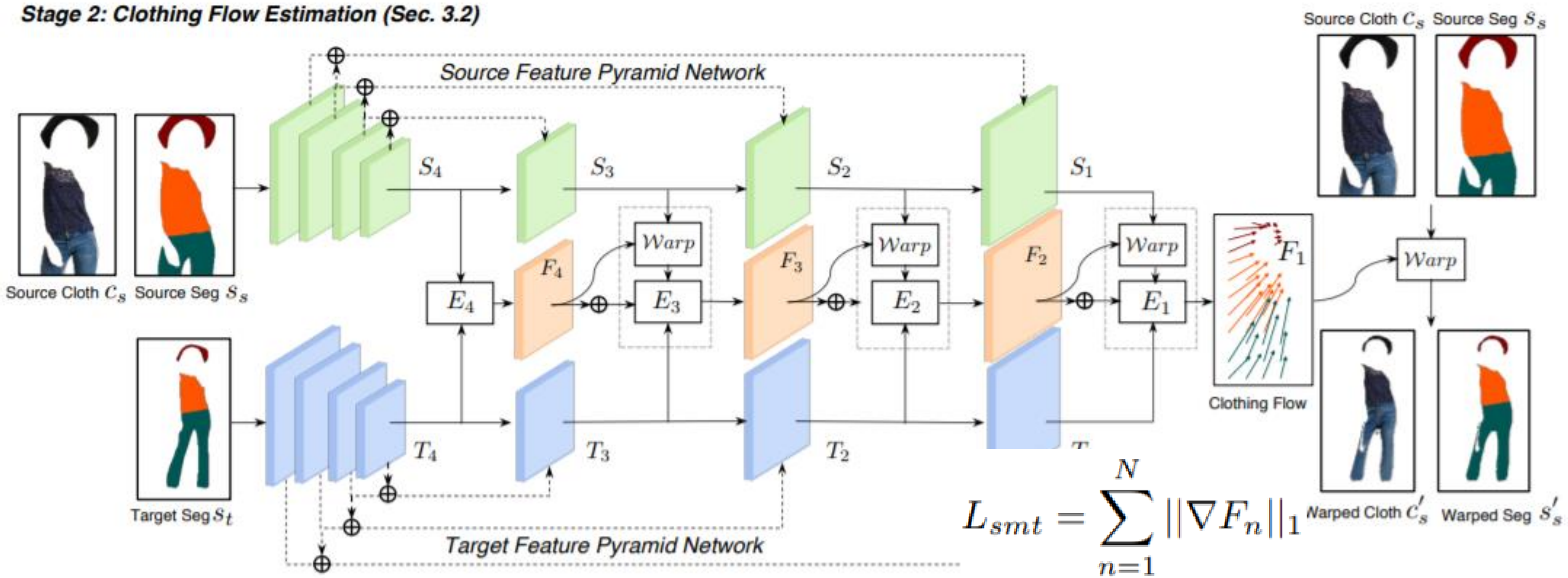


gradually refining the estimation

$2 \times H \times W$

FPNs do not share weights because they encode features from different modalities, which is different from the way to estimate optical flow [38] or object matching [23]. Then

- warping the source features at each pyramid level eases the process of directly modeling large misalignment and significant deformation that usually occur in clothing transfer.



$$L_{smt} = \sum_{n=1}^{N} ||\nabla F_n||_1$$

$$L_{perc}(c_s', c_t) = \sum_{l=0}^{5} \lambda_l ||\phi_l(c_s') - \phi_l(c_t)||_1$$

feature

$$L_{struct}(s_s', s_t) = \sum_i \mathbb{1}(s_{s,i})\mathbb{1}(s_{t,i})||s_{s,i}' - s_{t,i}||_1$$

레이블이 둘 다
있을 때만

$$F_N = E_N([S_N, T_N]),$$

flow conv concat

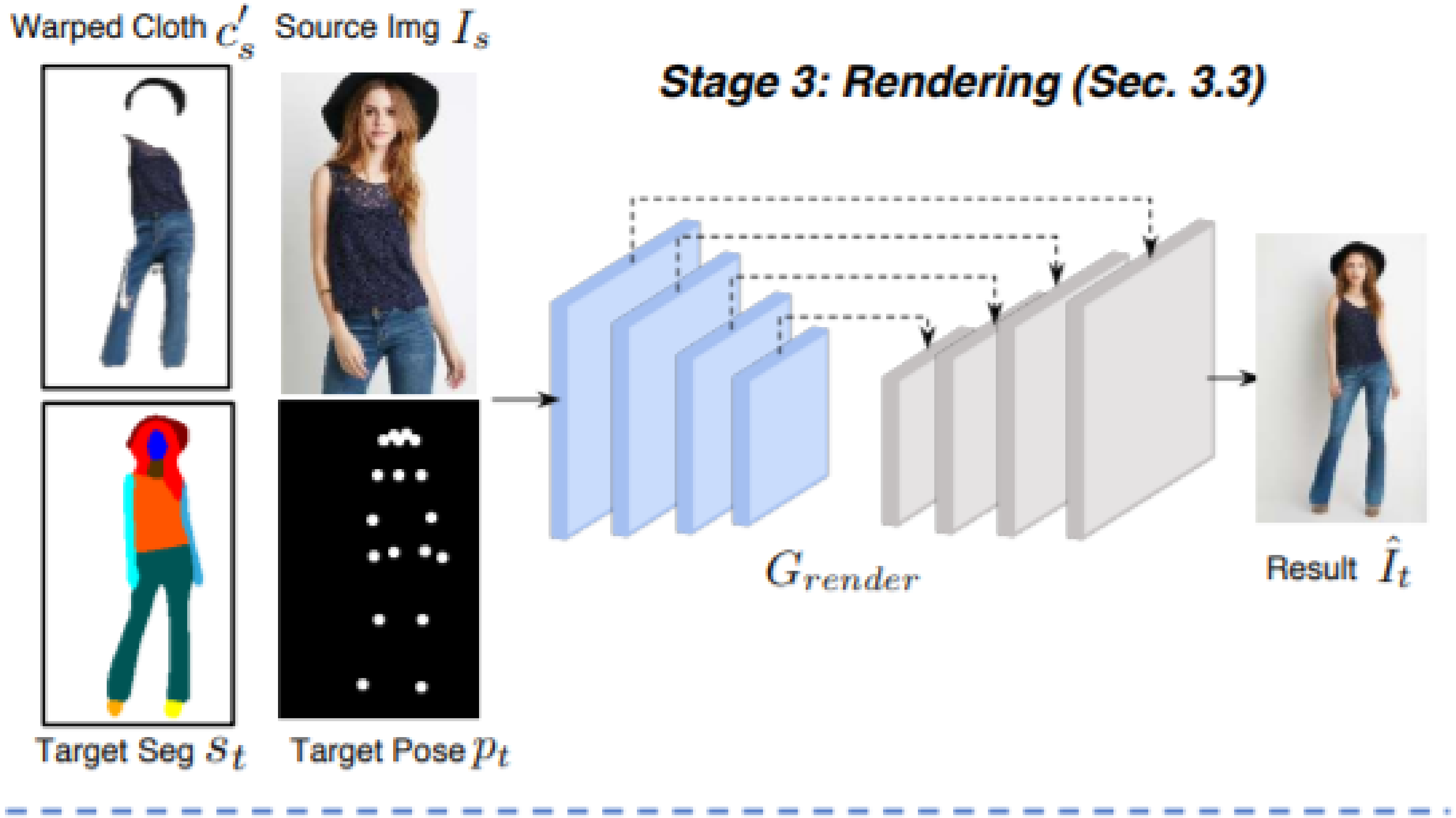
$$F_{n-1} = \mathcal{U}(F_n) + E_{n-1}([\mathcal{W}(S_{n-1}, \mathcal{U}(F_n)), T_{n-1}])$$

upsampling warping

$$L_{roi\_perc}(c_s', c_t, s_s', s_t) = \sum_{l=0}^{5} \lambda_l \sum_i \mathbb{1}(s_{s,i})\mathbb{1}(s_{t,i})||\phi_l(s_{s,i}' \odot c_s') - \phi_l(s_{t,i} \odot c_t)||_1$$

masking

- However, only minimizing Lperc(c ′ s , ct) may produce inaccurate warping when different clothing items have similar visual patterns,



$$L_{render} = L_{perc} + L_{style}.$$

$$L_{style} = \sum_{l=1}^{5} \gamma_l ||\mathcal{G}_l(\hat{I}_t) - \mathcal{G}_l(I_t)||_1.$$

- ClothFlow estimates the clothing flow with the feature extracted on the whole image and does not struggle to model long-range correspondence or partial observability.

- Also, they usually require to obtain a computationally expensive cost volume, but ClothFlow achieves satisfactory performance with one conv layer Ei to predict the flow at each pyramid level.

- At the core of ClothFlow is a cascaded appearance flow estimation network with a two-stream architecture to progressively warp the source image features and refine the flow prediction.

SonderFlowEstimator
(SonderVITON.py)

self.netFlow

FlowEstimator
(networks.py)

$$F_N = E_N([S_N, T_N]),$$
$$F_{n-1} = \mathcal{U}(F_n) + E_{n-1}([\mathcal{W}(S_{n-1}, \mathcal{U}(F_n)), T_{n-1}])$$

self.loss_G_perc

VGGPerceptualLoss
(networks.py)

$$L_{perc}(c'_s, c_t) = \sum_{l=0}^{5} \lambda_l ||\phi_l(c'_s) - \phi_l(c_t)||_1$$

self.loss_G_struct

l1_loss(self.warped_mask, self.parse_cloth)

$$L_{struct}(s'_s, s_t) = \sum_i \mathbb{1}(s_{s,i})\mathbb{1}(s_{t,i})||s'_{s,i} - s_{t,i}||_1$$

self.loss_TV

tv_loss
(utils.py)

$$L_{smt} = \sum_{n=1}^{N} ||\nabla F_n||_1$$

FlowEstimator
(networks.py)

```python
def forward(self, c_s, s_s, s_t):
    """[Forward pass of flow estimation network]

    Arguments:
        c_s {[torch Tensor]} -- [Source clothing item]
        s_s {[torch Tensor]} -- [Source segmentation]
        s_t {[torch Tensor]} -- [Target segmentation]

    Returns:
        [type] -- [description]
    """

    source_input = torch.cat([c_s, s_s], dim=1)
    s1, s2, s3, s4, s5 = self.SourceFPN(source_input)
    t1, t2, t3, t4, t5 = self.TargetFPN(s_t)

    f5 = self.e5(torch.cat([s5, t5], dim=1))
    f4 = self.upsample(f5) + self.e4(torch.cat([self.warp(s4, self.upsample(f5)), t4], dim=1))

    f3 = self.upsample(f4) + self.e3(torch.cat([self.warp(s3, self.upsample(f4)), t3], dim=1))

    f2 = self.upsample(f3) + self.e2(torch.cat([self.warp(s2, self.upsample(f3)), t2], dim=1))

    f1 = self.upsample(f2) + self.e1(torch.cat([self.warp(s1, self.upsample(f2)), t1], dim=1))

    # Warped clothing item
    c_s_prime = self.warp(c_s, self.upsample(f1))
    s_s_prime = self.warp(s_s, self.upsample(f1))

    return f5, f4, f3, f2, f1, c_s_prime, s_s_prime
```

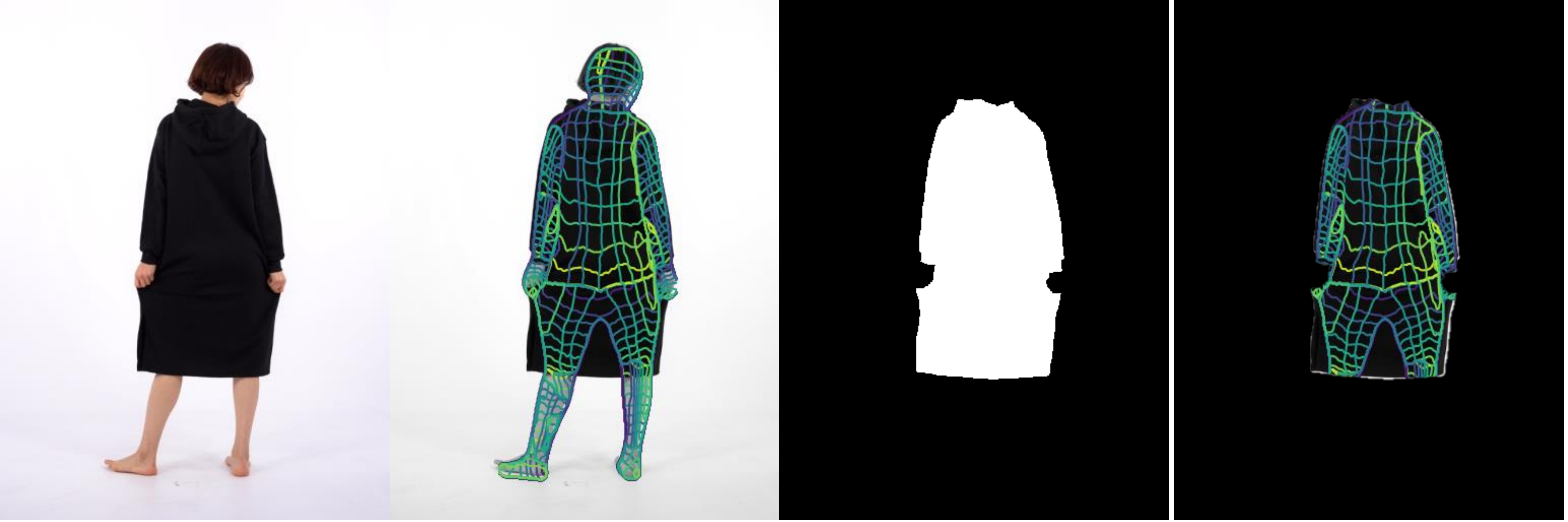
VGGPerceptualLoss
(networks.py)

```python
# Source: https://github.com/NVIDIA/pix2pixHD
class VGGPerceptualLoss(nn.Module):
    def __init__(self):
        super(VGGPerceptualLoss, self).__init__()
        self.vgg = Vgg19().cuda().eval()
        self.criterion = nn.L1Loss()
        self.weights = [1.0 / 32, 1.0 / 16, 1.0 / 8, 1.0 / 4, 1.0]

    def forward(self, x, y):
        x_vgg, y_vgg = self.vgg(x), self.vgg(y)
        loss = 0
        for i in range(len(x_vgg)):
            loss += self.weights[i] * self.criterion(x_vgg[i], y_vgg[i].detach())
        return loss
```

tv_loss
(utils.py)

```python
def tv_loss(img, tv_weight):
    """
    Compute total variation loss.
    Inputs:
    - img: PyTorch Variable of shape (1, 3, H, W) holding an input image.
    - tv_weight: Scalar giving the weight w_t to use for the TV loss.
    Returns:
    - loss: PyTorch Variable holding a scalar giving the total variation loss
      for img weighted by tv_weight.
    """
    w_variance = torch.sum(torch.pow(img[:, :, :, :-1] - img[:, :, :, 1:], 2))
    h_variance = torch.sum(torch.pow(img[:, :, :-1, :] - img[:, :, 1:, :], 2))
    loss = tv_weight * (h_variance + w_variance)
    return loss
```



마스크 내부 iuv맵이 존재하는 픽셀에 대응하는 이미지 픽셀 좌표를 맵핑