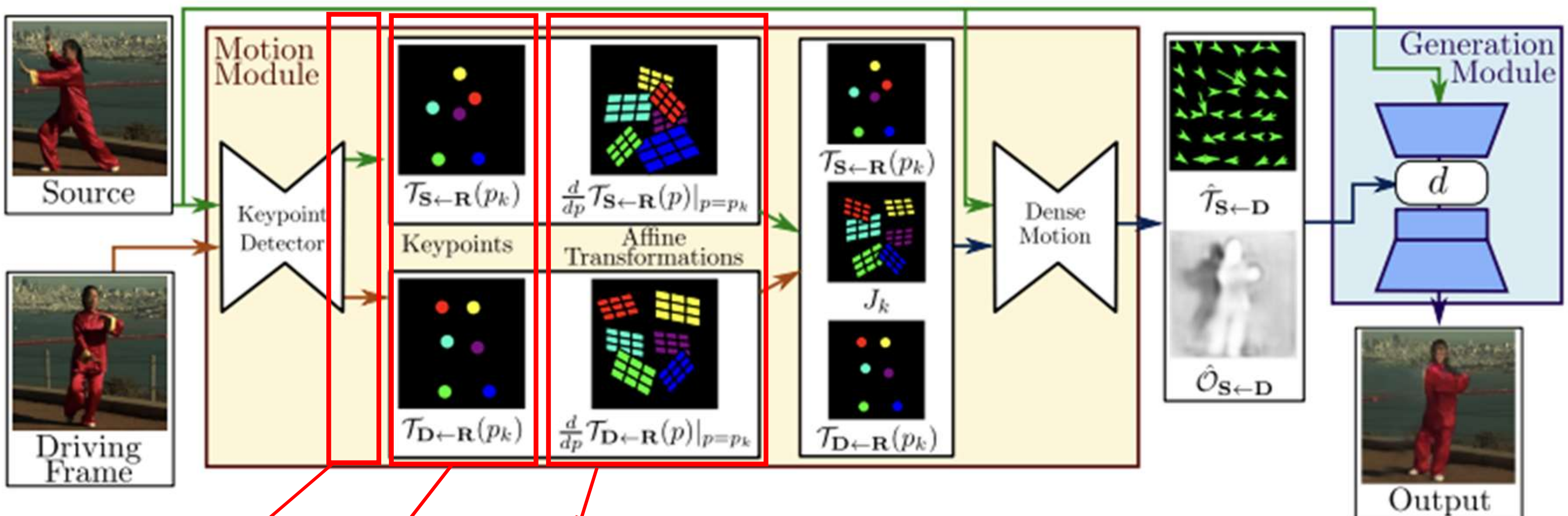
First Order Motion Model (NIPS 2019)

Propose predicting movements by first order Taylor expansion in a neighborhood of key point location



Jacobian of each heatmap

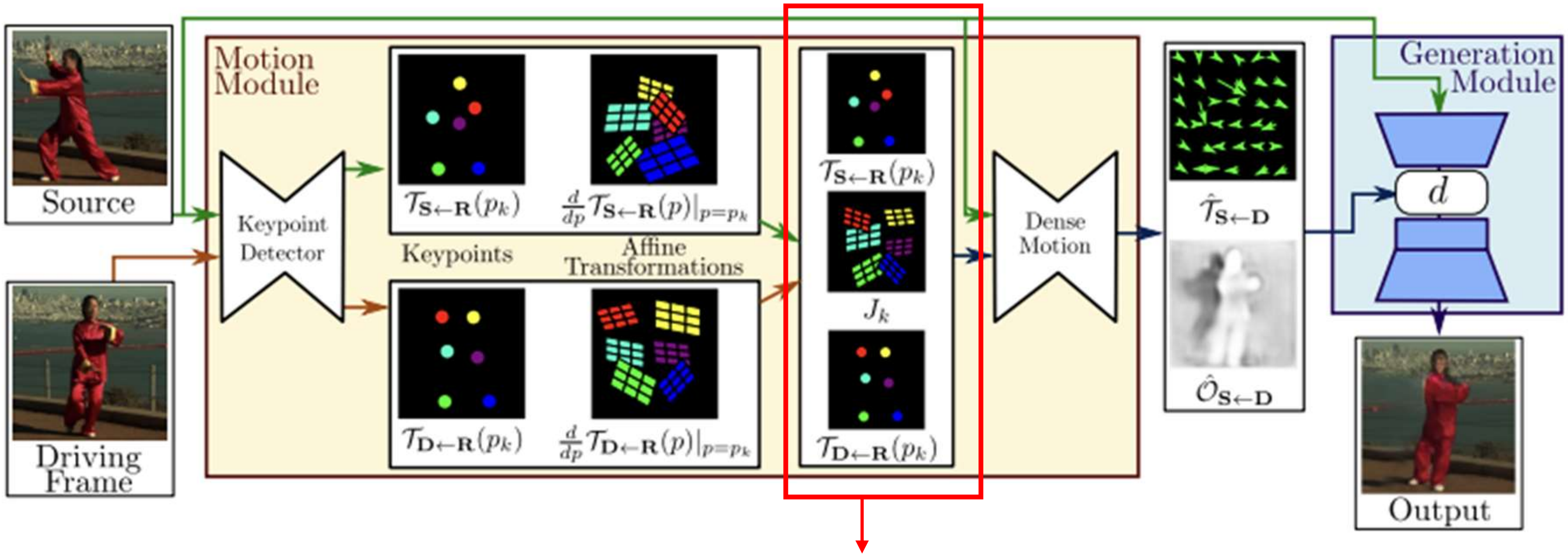
Heatmap of keypoints

Coordinate of center of heatmap

```
jacobian = nn.Conv2d(in_channels=predictor.out_filters, out_channels= 4 * num_jacobian_maps, kernel_size=(7, 7))
jacobian_map = jacobian(feature_map)
jacobian_map = jacobian_map.reshape(final_shape[0], num_jacobian_maps, 4, final_shape[2], final_shape[3])
heatmap = heatmap.unsqueeze(2)

jacobian = heatmap * jacobian_map
jacobian = jacobian.view(final_shape[0], final_shape[1], 4, -1)
jacobian = jacobian.sum(dim=-1)
jacobian = jacobian.view(jacobian.shape[0], jacobian.shape[1], 2, 2)
```

First Order Motion Model (NIPS 2019)

Propose predicting movements by first order Taylor expansion in a neighborhood of key point location



$$\mathcal{T}_{\mathbf{S}\leftarrow\mathbf{D}}(z) \approx \boxed{\mathcal{T}_{\mathbf{S}\leftarrow\mathbf{R}}(p_k)} + \boxed{J_k}(\boxed{z} - \boxed{\mathcal{T}_{\mathbf{D}\leftarrow\mathbf{R}}(p_k)})$$

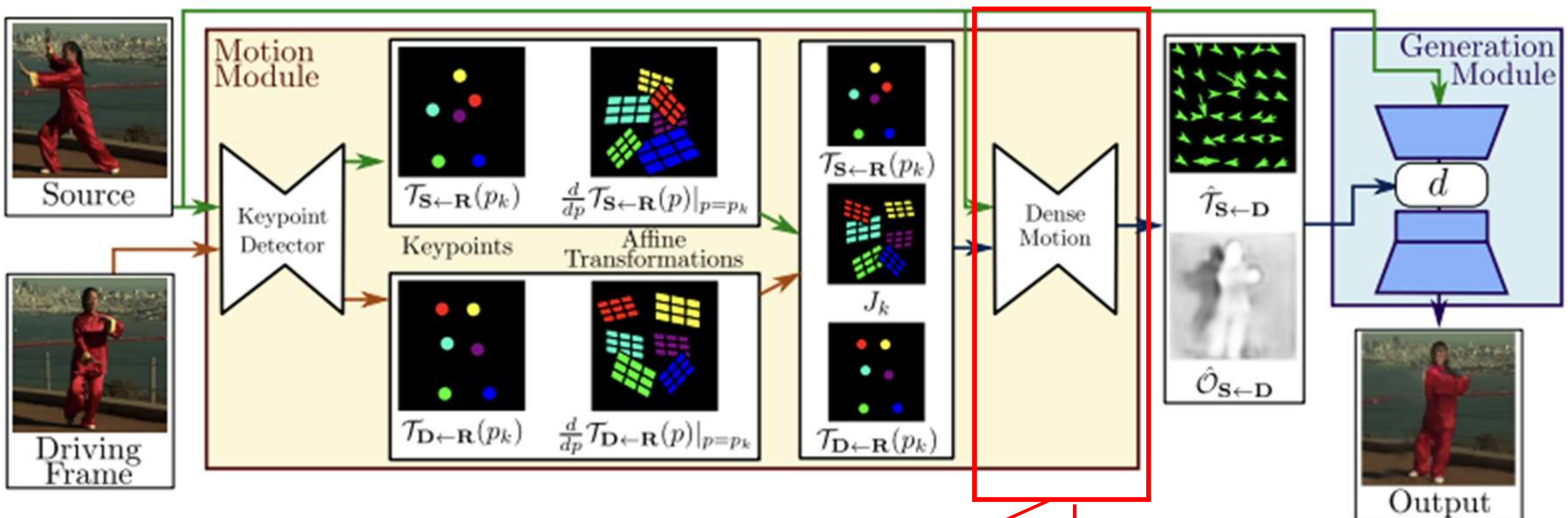
```
coordinate_grid = identity_grid - kp_driving['value'].view(bs, self.num_kp, 1, 1, 2)
if 'jacobian' in kp_driving:
    jacobian = torch.matmul(kp_source['jacobian'], torch.inverse(kp_driving['jacobian']))
    jacobian = jacobian.unsqueeze(-3).unsqueeze(-3)
    jacobian = jacobian.repeat(1, 1, h, w, 1, 1)
    coordinate_grid = torch.matmul(jacobian, coordinate_grid.unsqueeze(-1))
    coordinate_grid = coordinate_grid.squeeze(-1)

driving_to_source = coordinate_grid + kp_source['value'].view(bs, self.num_kp, 1, 1, 2)
```

$$\mathcal{T}_{\mathbf{S}\leftarrow\mathbf{D}} = \mathcal{T}_{\mathbf{S}\leftarrow\mathbf{R}} \circ \mathcal{T}_{\mathbf{R}\leftarrow\mathbf{D}} = \mathcal{T}_{\mathbf{S}\leftarrow\mathbf{R}} \circ \mathcal{T}_{\mathbf{D}\leftarrow\mathbf{R}}^{-1}$$

$$J_k = \left(\frac{d}{dp}\mathcal{T}_{\mathbf{S}\leftarrow\mathbf{R}}(p)\Big|_{p=p_k}\right)\left(\frac{d}{dp}\mathcal{T}_{\mathbf{D}\leftarrow\mathbf{R}}(p)\Big|_{p=p_k}\right)^{-1}$$

First Order Motion Model (NIPS 2019)

Propose predicting movements by first order Taylor expansion in a neighborhood of key point location



Get heatmaps of each keypoints from coordinate

$$\mathbf{H}_k(z) = exp\left(\frac{(\mathcal{T}_{\mathbf{D}\leftarrow\mathbf{R}}(p_k) - z)^2}{\sigma}\right) - exp\left(\frac{(\mathcal{T}_{\mathbf{S}\leftarrow\mathbf{R}}(p_k) - z)^2}{\sigma}\right)$$

Make k+1 frames from source image with each keypoints
(Assumption: an object is composed of K parts and each part is moved according to the First Order Taylor estimation)

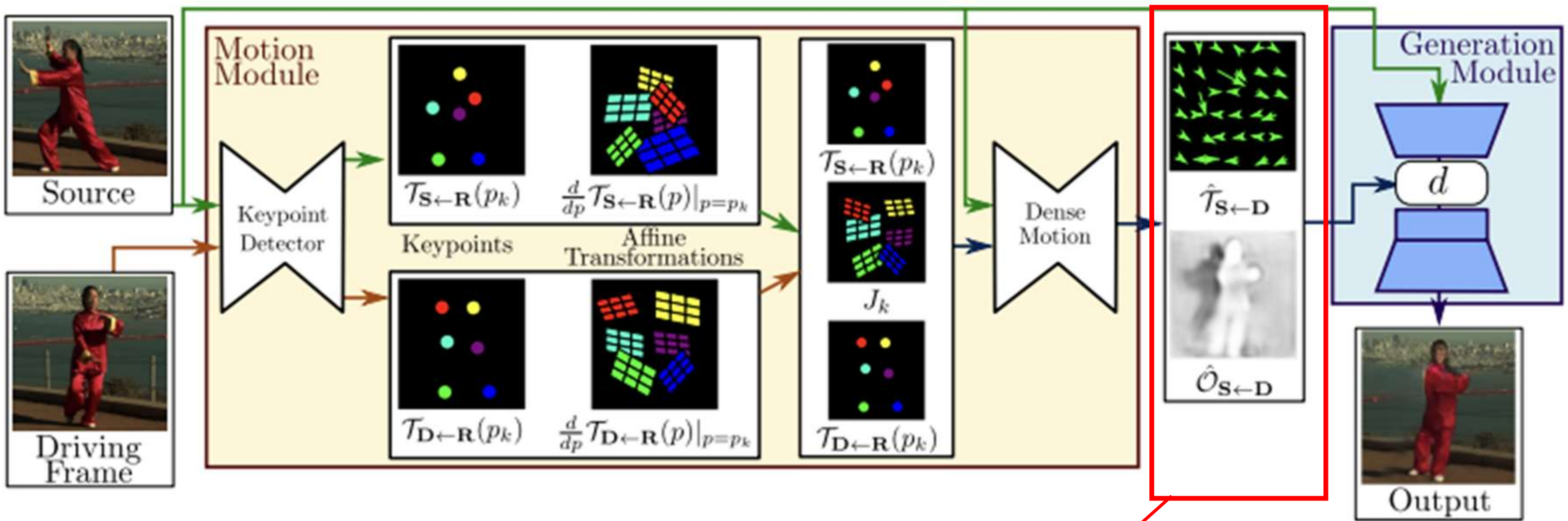
$$\hat{\mathcal{T}}_{\mathbf{S}\leftarrow\mathbf{D}}(z) = \mathbf{M}_0 z + \sum_{k=1}^{K} \mathbf{M}_k \left(\mathcal{T}_{\mathbf{S}\leftarrow\mathbf{R}}(p_k) + J_k(z - \mathcal{T}_{\mathbf{D}\leftarrow\mathbf{R}}(p_k))\right)$$

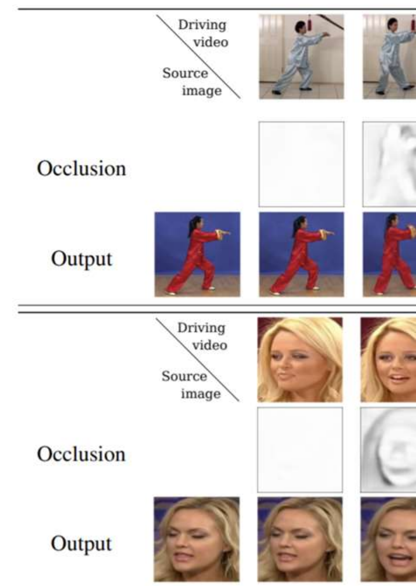
$\mathcal{T}_{\mathbf{S}\leftarrow\mathbf{D}}(z)$

$$\hat{\mathcal{T}}_{\mathbf{S}\leftarrow\mathbf{D}}(z) \xrightarrow{\text{Hourglass}} \hat{\mathcal{O}}_{\mathbf{S}\leftarrow\mathbf{D}}$$

First Order Motion Model (NIPS 2019)

Propose predicting movements by first order Taylor expansion in a neighborhood of key point location





Get feature map of source image after convolution layer.

Then warp according to $\hat{\mathcal{T}}_{\mathbf{S}\leftarrow\mathbf{D}}(z) \longrightarrow f_w(\boldsymbol{\xi}, \hat{\mathcal{T}}_{\mathbf{S}\leftarrow\mathbf{D}})$

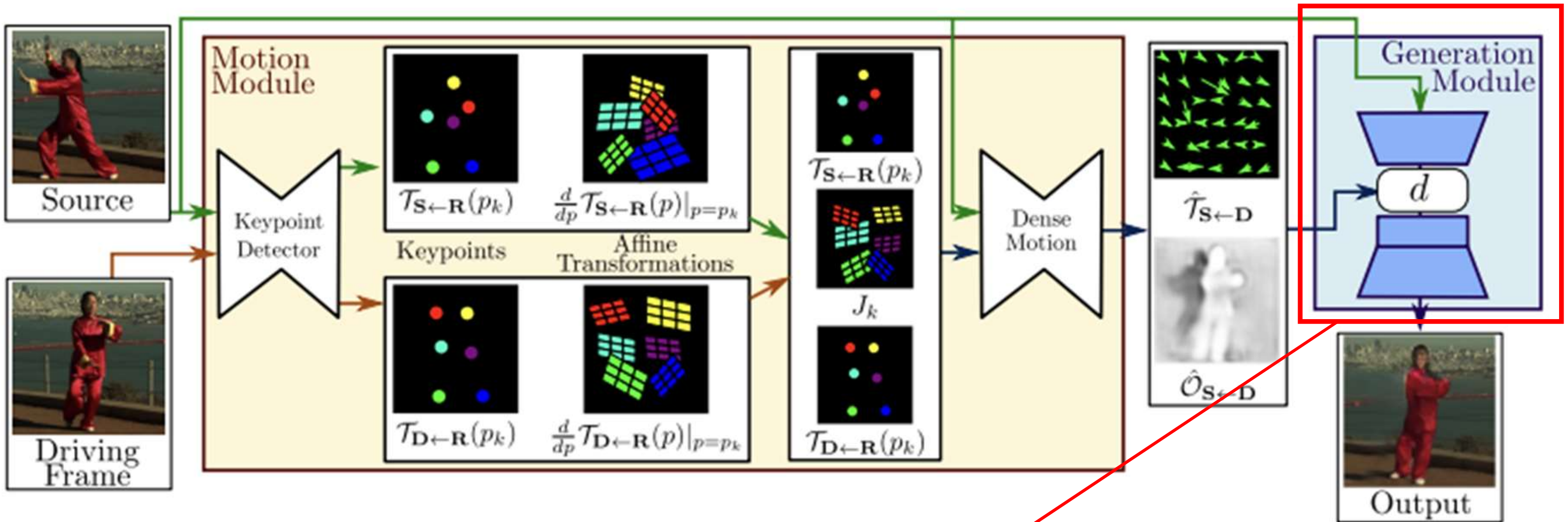
Transformed feature map is:

$\boldsymbol{\xi}' = \hat{\mathcal{O}}_{\mathbf{S}\leftarrow\mathbf{D}} \odot f_w(\boldsymbol{\xi}, \hat{\mathcal{T}}_{\mathbf{S}\leftarrow\mathbf{D}})$ $\boxed{\odot \text{ denotes the Hadamard product}}$

선형대수학에서, **아다마르 곱**(영어: Hadamard product)은 같은 크기의 두 행렬의 각 성분을 곱하는 연산이다. 즉, 일반 행렬곱은 $m \times n$과 $n \times p$의 꼴의 두 행렬을 곱하지만, 아다마르 곱은 $m \times n$과 $m \times n$의 꼴의 두 행렬을 곱한다. 덧셈에 대하여 분배 법칙을 따른다. 기호는 ◯.

First Order Motion Model (NIPS 2019)

Propose predicting movements by first order Taylor expansion in a neighborhood of key point location



Decode $\xi' = \hat{\mathcal{O}}_{\mathbf{S}\leftarrow\mathbf{D}} \odot f_w(\xi, \hat{\mathcal{T}}_{\mathbf{S}\leftarrow\mathbf{D}})$

Calculate perceptual loss

Calculate equivariance loss
- Keypoints from driving image and generated image
- Jacobian from driving image and generated image

Calculate discriminator loss
- Loss from image pyramid of generated image and driving image

$$L_{rec}(\hat{\mathbf{D}}, \mathbf{D}) = \sum_{i=1}^{I} \left| N_i(\hat{\mathbf{D}}) - N_i(\mathbf{D}) \right|$$