[CVPR19] FSA-Net: Learning Fine-Grained Structure Aggregation for Head Pose Estimation from a Single Image

Speaker : Ji In Kim

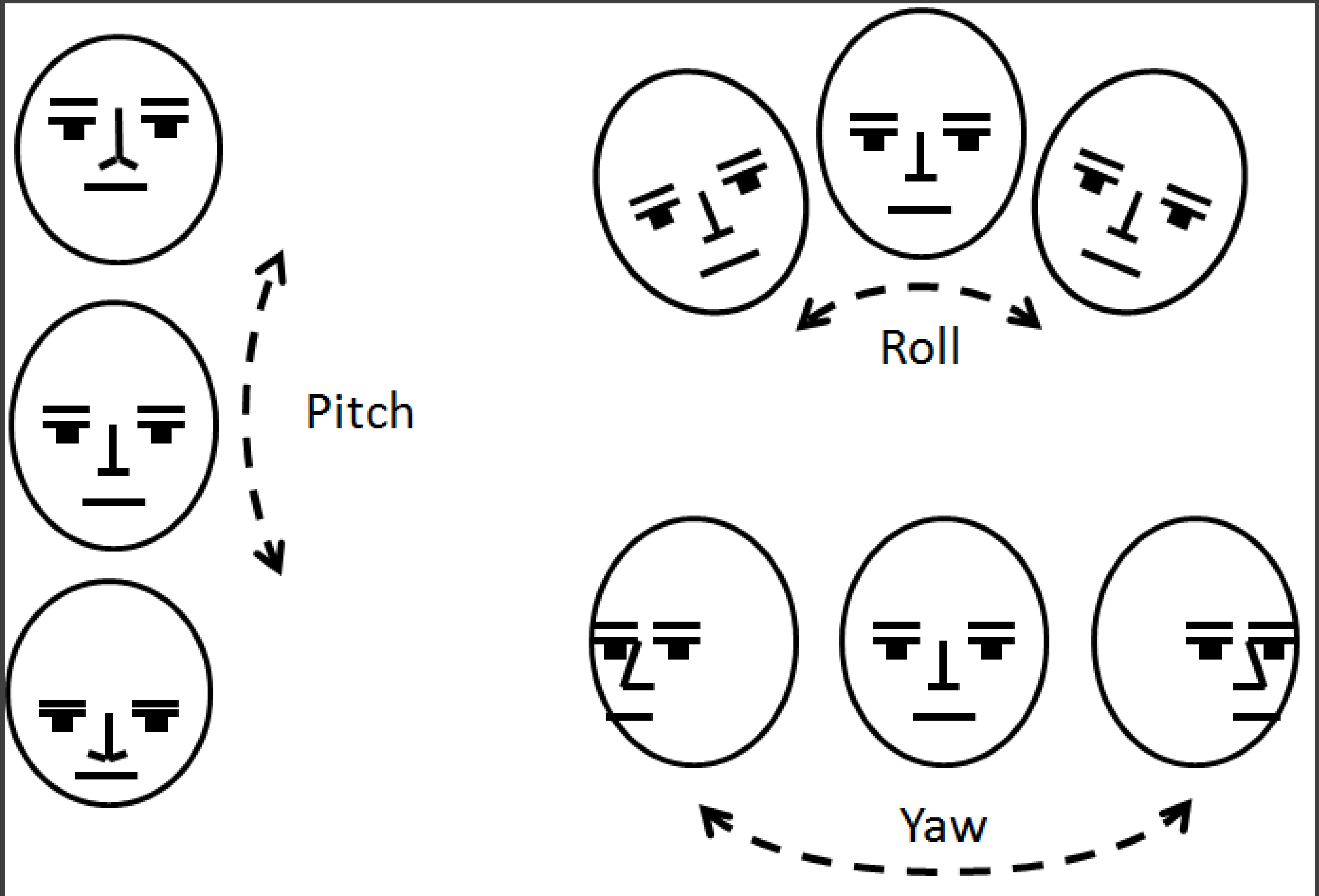
목차

- Introduction
- Related work
- Method
- Estimation

1. Introduction

What is head pose ?

Yaw : Y축
Pitch : X축
Roll : Z 축

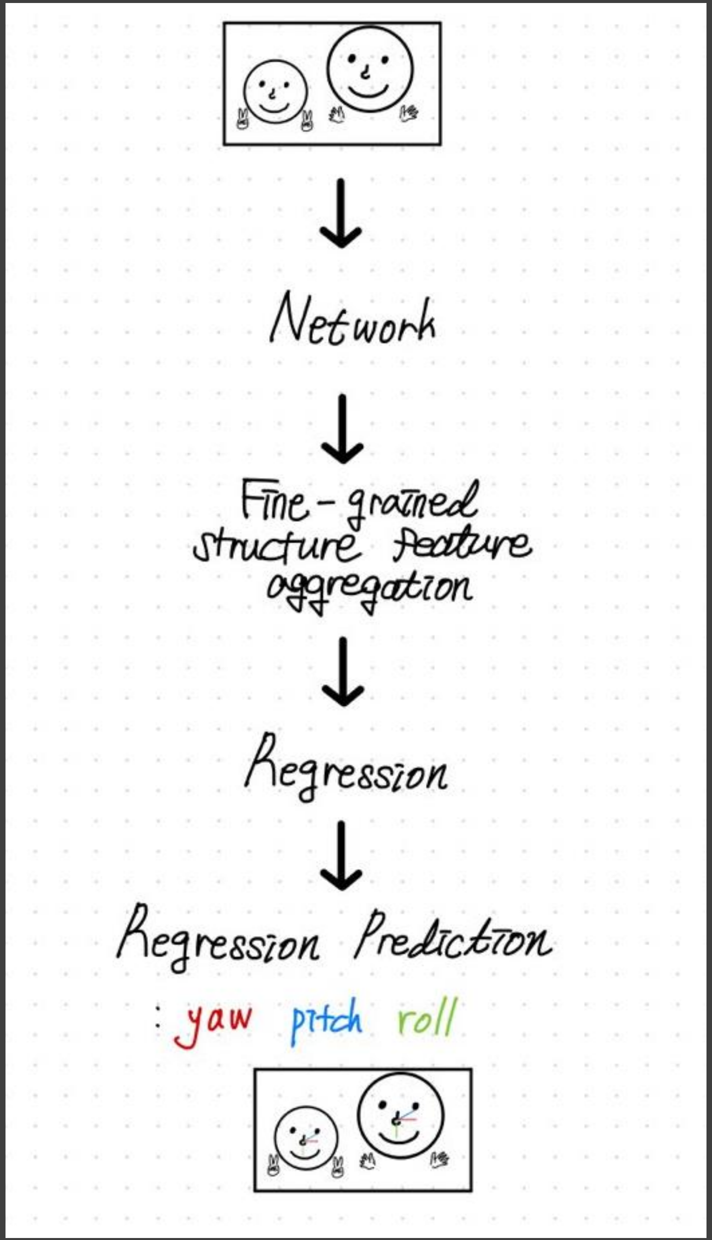


- Head pose estimation problem 이 적용되는 곳
  - driver behavior monitoring
  - human attention monitoring
  - human attention modeling
  - Identity recognition
  - expression recognition

- Head pose estimation problem 에 사용하는 다른 방법들
  - depth images → spatial cameras 가 필요함
  - video sequence → 많은 computation 을 요구하는 recurrent 구조임
  - Facial landmark detection → 많은 computation 과 bigger model 이 필요함

Differentiation with others

FSA-Net



Network
Fine-grained structure feature aggregation
Regression
Regression Prediction
: yaw pitch roll

- Pixel-level features 를 region-level features 로 그룹화 하는 Fine-grained structure mapping 을 찾기 위해 학습함

- Regression

Teaser image

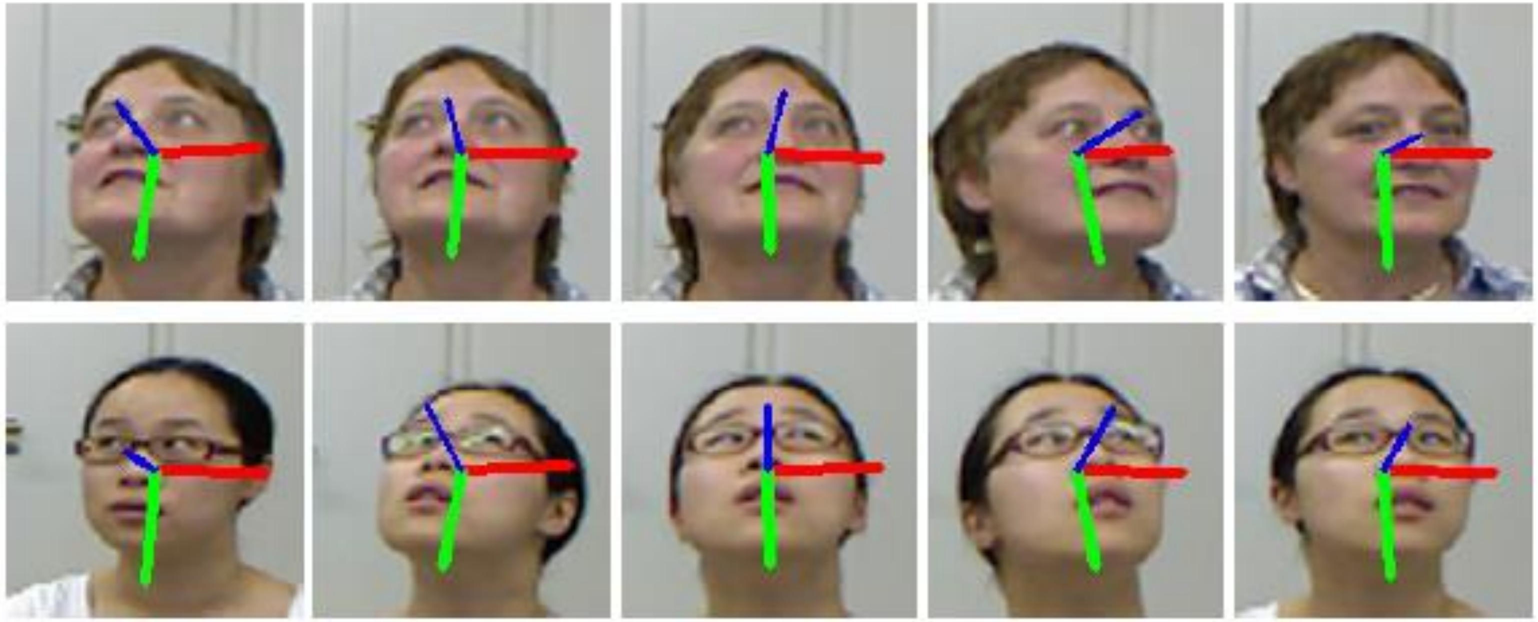


Figure 1. Sample results of pose estimation using the proposed method. Our method only takes as input a single RGB frame. Results for two sequences of head motion are shown. The blue line indicates the direction the subject is facing; the green line for the downward direction while the red one for the side.

2. Related Work

- Landmark-based methods
  - Regression-based methods
  - Model-based methods
  - Deep Learning-based methods
- Methods with different modalities
  - RGB
  - Depth
  - RGB-Time
- Multi-task methods
- Attention

		Year	Conference
Landmark-based methods	Regression-based	2010, 2013, 2014, 2015(2)	IJCV, CVPR(3)
	Model-based	1995, 2004, 2008	IJCV, ECCV
	Deep Learning-based	2013, 2016, 2017	CVPR(2), ICCV
Methods with different modalities	RGB	2017(5)	TPAMI
	Depth	2011, 2015	ICCV
	RGB-Time	2009, 2017, 2018	IJCV, TPAMI, CVPR
Multi-task methods		2012, 2014, 2017(4)	ECCV, TPAMI(2), CVPR
Attention		2017, 2018(4), 2019(2)	ECCV(3), TIP, CVPR(2), NIPS

3. Method

3.1. Problem formulation

- $X = \{ x_n \mid n = 1, \dots, N \}$
  - set of training face images
- y_n
  - pose vector (yaw, pitch, roll)
- Goal
  - $\tilde{y}$ 찾기!

$$J(X) = \frac{1}{N} \sum_{n=1}^{N} \|\tilde{y}_n - y_n\|_1 \,, \qquad (1)$$

3.2. SSR-Net-MD

- SSR-Net 사용 : Age estimating problem.
- Network
  - 회귀 문제를 분류 문제로 만들고, 분류 문제 풀기
  - 연령대에 대한 확률 분포를 출력
  - 확률 분포로 나이 측정
- SSR-Net 은 계층적 분류를 수행하고 $\tilde{y}$을 측정하는 데에 soft stage-wise regression 을 사용한다.

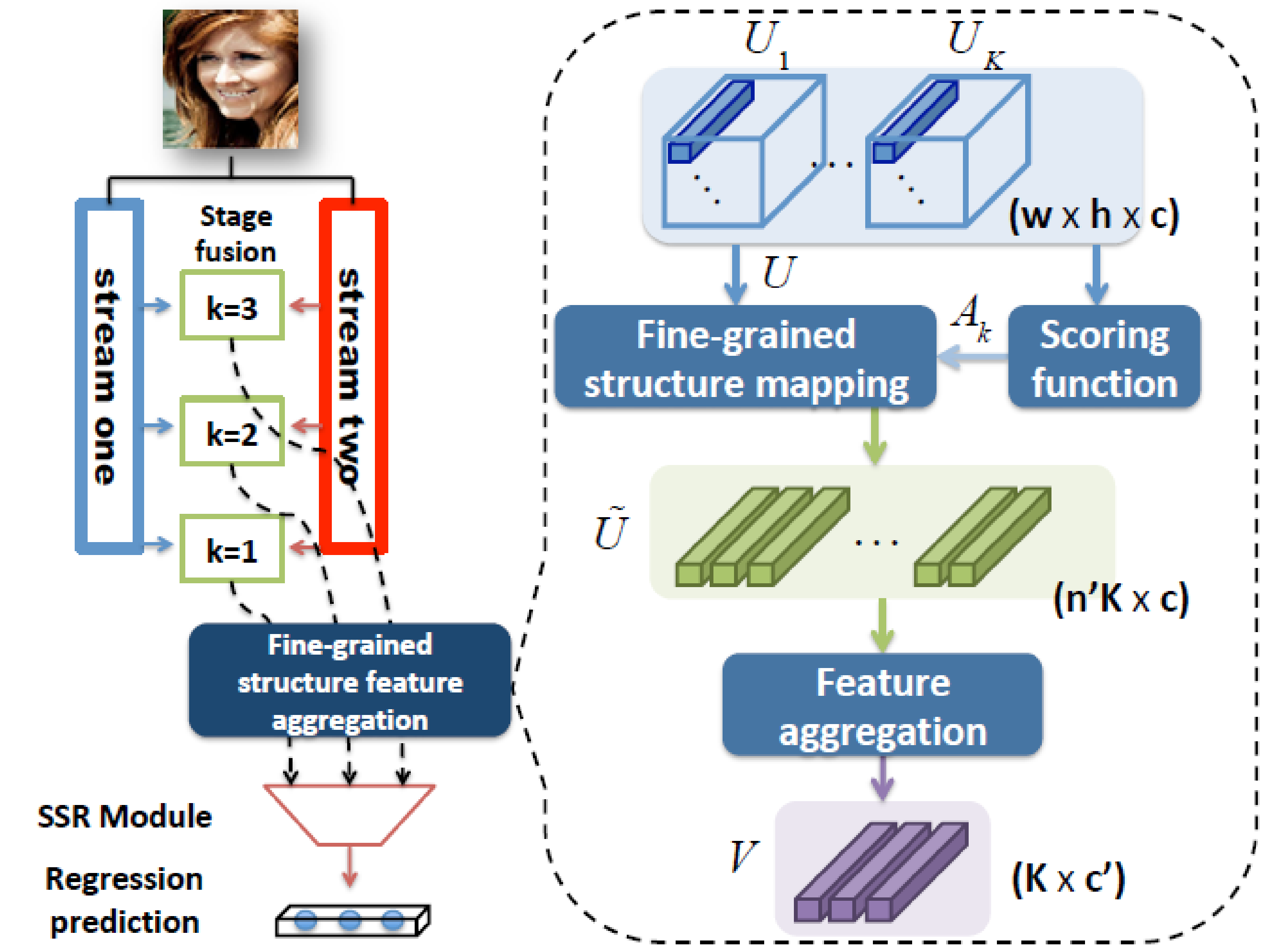
- K : stage 의 수
- $\vec{p}^{(k)}$: k 번째 stage 의 확률 분포
- $\vec{\mu}^{(k)}$ 는 k 번째 stage 에서 age 그룹의 대표 값을 포함하는 벡터
- $\vec{\mu}^{(k)}$ 을 수정
  - $\vec{\eta}^{(k)}$ 는 각 bin 의 center 를 조절
  - $\triangle k$ 는 k 번째 stage 에서 모든 bin 들의 너비를 조절

- Input image 가 들어오면, SSR-Net 은 $\{\vec{p}^{(k)}, \vec{\eta}^{(k)}, \triangle k\}_{k=1}^{K}$ 을 출력으로 내고, 나이를 측정하는 데에 soft stage wise regression 을 사용한다.

$$\tilde{y} = \sum_{k=1}^{K} \vec{p}^{(k)} \cdot \vec{\mu}^{(k)}, \qquad (2)$$

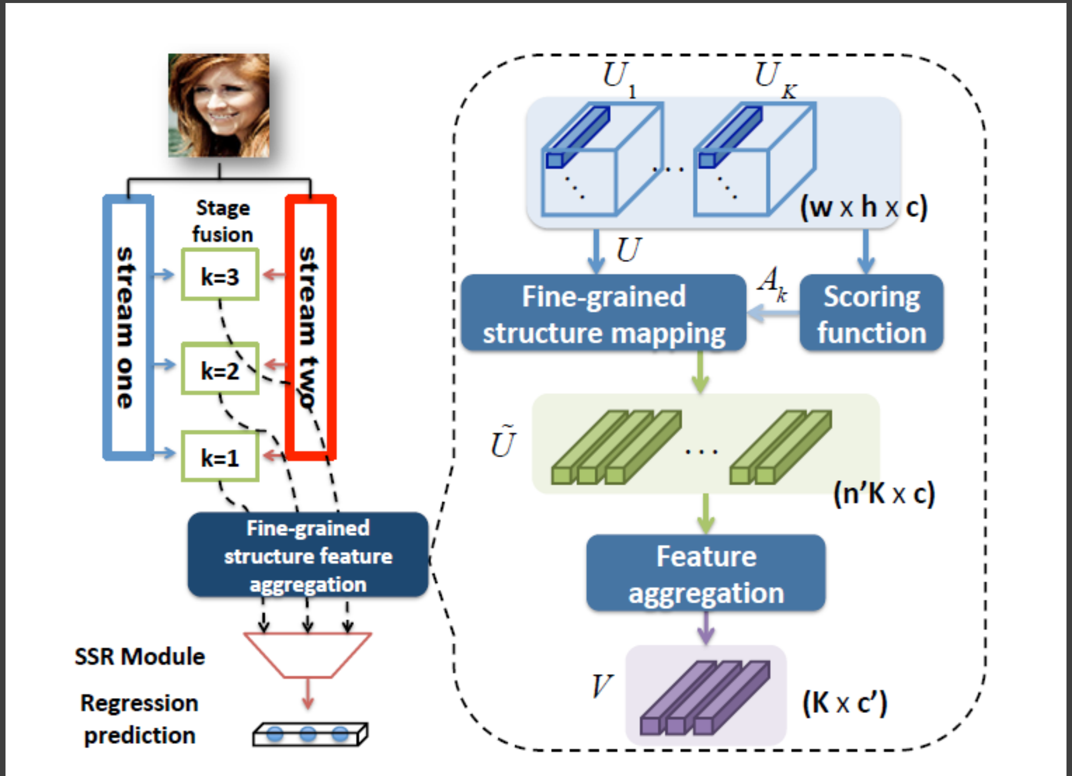
- Regression problem → SSR($\{\vec{p}^{(k)}, \vec{\eta}^{(k)}, \triangle k\}_{k=1}^{K}$) → (2) → 기대 값
- 다차원 회귀에 대한 SSR-Net → SSR-Net-MD

3.3. Overview of FSA-Net

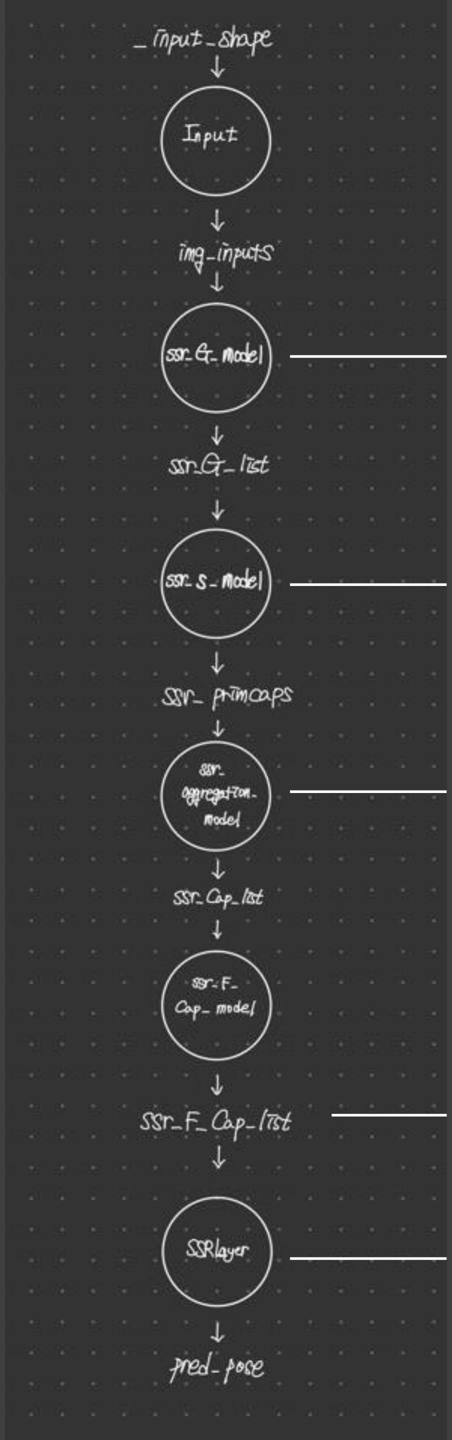


(a) FSA-Net

```python
def __call__(self):
    logging.debug("Creating model...")
    img_inputs = Input(self._input_shape) # _input_shape = (image_size, image_size, 3) --> (64, 64, 3)

    # Build various models
    ssr_G_model = self.ssr_G_model_build(img_inputs) # two streams + stage fusion --> (w, h, c)

    if self.is_noS_model: # using Scoring function
        ssr_S_model = self.ssr_noS_model_build()
    else: # not using Scoring function
        ssr_S_model = self.ssr_S_model_build(num_primcaps=self.num_primcaps,m_dim=self.m_dim)

    # Aggregation
    ssr_aggregation_model = self.ssr_aggregation_model_build((self.num_primcaps,64)) # ssr_aggregation_model_build((7*3, 64)) --> (K, c)

    if self.is_fc_model:
        ssr_F_Cap_model = self.ssr_FC_model_build(self.F_shape,'ssr_F_Cap_model')
    else:
        ssr_F_Cap_model = self.ssr_F_model_build(self.F_shape,'ssr_F_Cap_model') # SSR 함수에 인자로 들어가는 변수인 p(k), eta(k), delta(k) 계산

    # Wire them up
    ssr_G_list = ssr_G_model(img_inputs) # two streams + stage fusion
    ssr_primcaps = ssr_S_model(ssr_G_list) # Attention + Fine grained structure mapping
    ssr_Cap_list = ssr_aggregation_model(ssr_primcaps) # Aggregation
    ssr_F_Cap_list = ssr_F_Cap_model(ssr_Cap_list) # p(k), eta(k), delta(k)

    # pose estimation
    pred_pose = SSRLayer(s1=self.stage_num[0], s2=self.stage_num[1], s3=self.stage_num[2], lambda_d=self.lambda_d, name="pred_pose")(ssr_F_Cap_list)

    return Model(inputs=img_inputs, outputs=pred_pose)
```



(a) FSA-Net



Two streams + stage fusion

Attention + Fine grained structure mapping

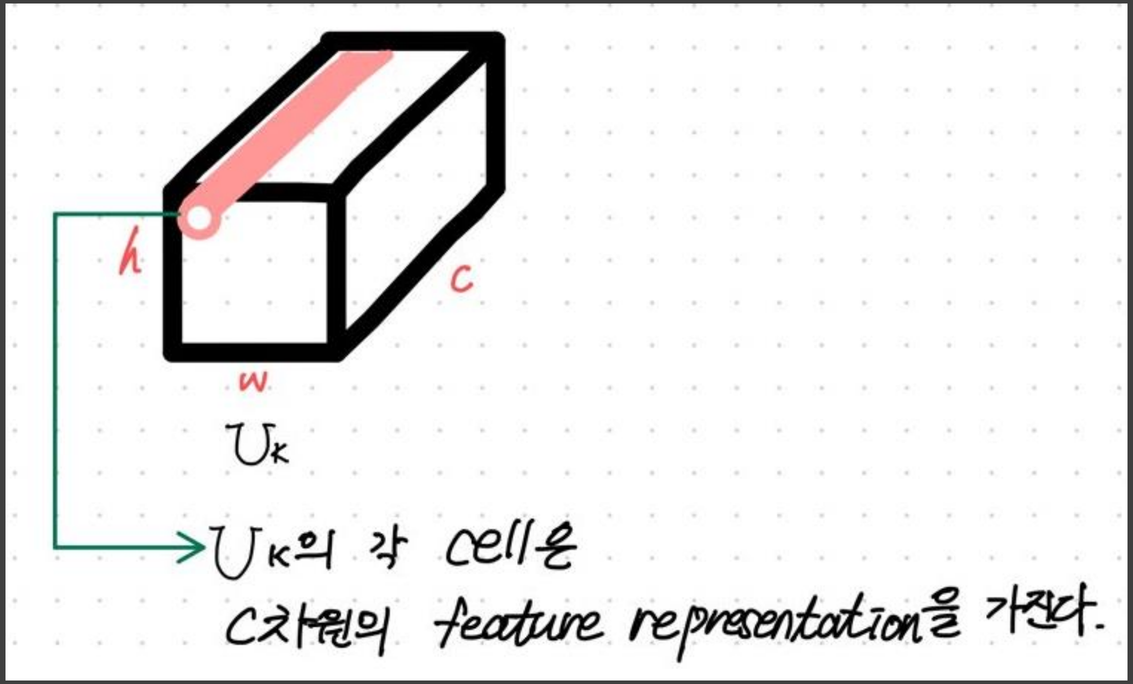
Aggregation

$\vec{p}^{(k)}, \vec{\eta}^{(k)}, \triangle k$

Pose Estimation

- 각 stream 은 각 stage 에서 feature map 을 추출.
- Stage fusion module

1) 2 개의 feature map → element-wise multiplication 으로 합침
2) C 1x1 convolutions 에 적용
3) Average pooling 을 사용 → feature map 의 크기 ➔ w x h
4) w x h x c 크기의 feature map U_k !!!



- Stage fusion module

```
s_layer4 = Conv2D(64,(1,1),activation='tanh')(s_layer4)
x_layer4 = Conv2D(64,(1,1),activation='relu')(x_layer4)

feat_s1_pre = Multiply()([s_layer4,x_layer4])
#-------------------------------------------------------
s_layer3 = Conv2D(64,(1,1),activation='tanh')(s_layer3)
x_layer3 = Conv2D(64,(1,1),activation='relu')(x_layer3)

feat_s2_pre = Multiply()([s_layer3,x_layer3])
#-------------------------------------------------------
s_layer2 = Conv2D(64,(1,1),activation='tanh')(s_layer2)
x_layer2 = Conv2D(64,(1,1),activation='relu')(x_layer2)

feat_s3_pre = Multiply()([s_layer2,x_layer2])
```

```
feat_s3_pre = AveragePooling2D((2,2))(feat_s3_pre) # make sure (8x8x64) feature maps

return Model(inputs=img_inputs,outputs=[feat_s1_pre,feat_s2_pre,feat_s3_pre], name='ssr_G_model')
```

$U_1 . U_2 . U_3$ ($w \times h \times c$)

↓

aggregation module : capsule / NetVLAD
[36] [1]

↓ → more meaningful representation을 얻기 위해.

K c´-d features

$U_1 . U_2 . U_3$ ($w \times h \times c$)

↓

spatial grouping of features
(fine-grained structure mapping module)

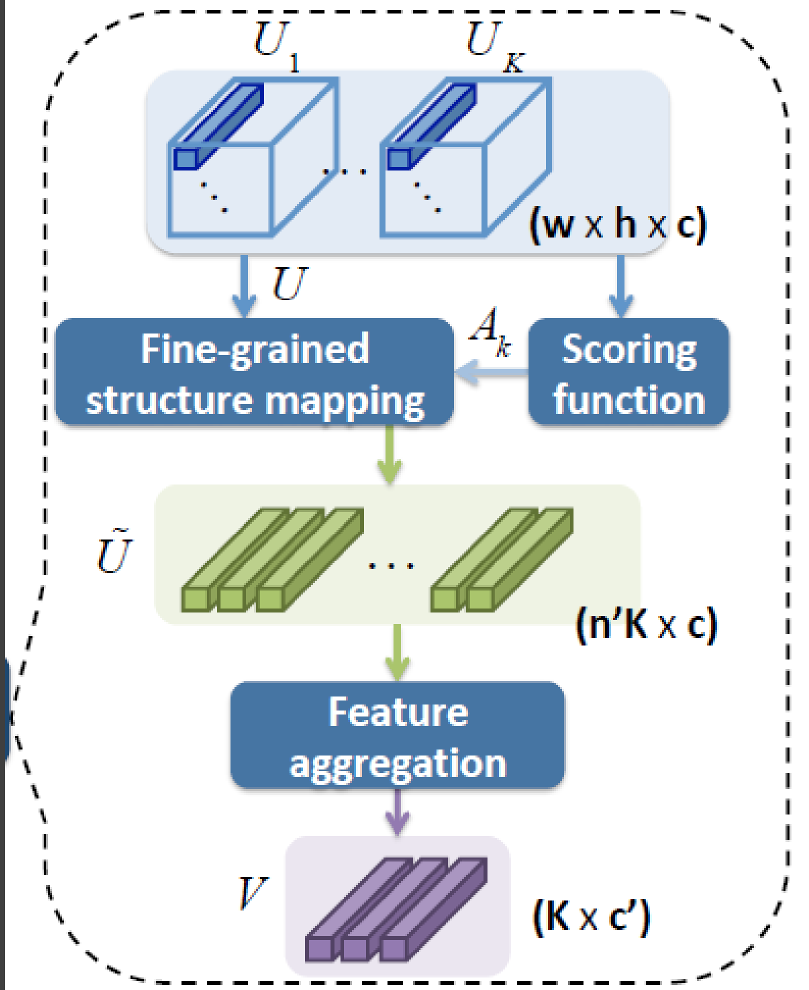
→ spatial information을 많이 포함하기 위해.

↓

aggregation module

↓

K c´-d features



3.4. Scoring function

- Scoring function $\Phi(u)$
  - Pixel-level feature 인 $u = (u_1, \ldots, u_c)$ 가 주어지면 중요도를 측정하는 함수

- Scoring function 의 결과로 attention map A_K (=중요도)를 얻을 수 있다.

$$A_k(i,j) = \Phi(U_k(i,j)).$$

- Scoring function 의 세가지 option

(1) 1 x 1 convolution $i.e., \Phi(u) = \sigma(w \cdot u),$

- σ : sigmoid function
- w : learnable convolutional kernel

(2) Variance $\Phi(u) = \sum_{i=1}^{c} (u_i - \mu)^2$ $\mu = \frac{1}{c} \sum_{i=1}^{c} u_i.$

(3) Uniform $\Phi(u) = 1.$

- $\widetilde{U} = U$
- Fine-grained structure mapping 이 수행되지 않는다.

(1) 1 x 1 convolution

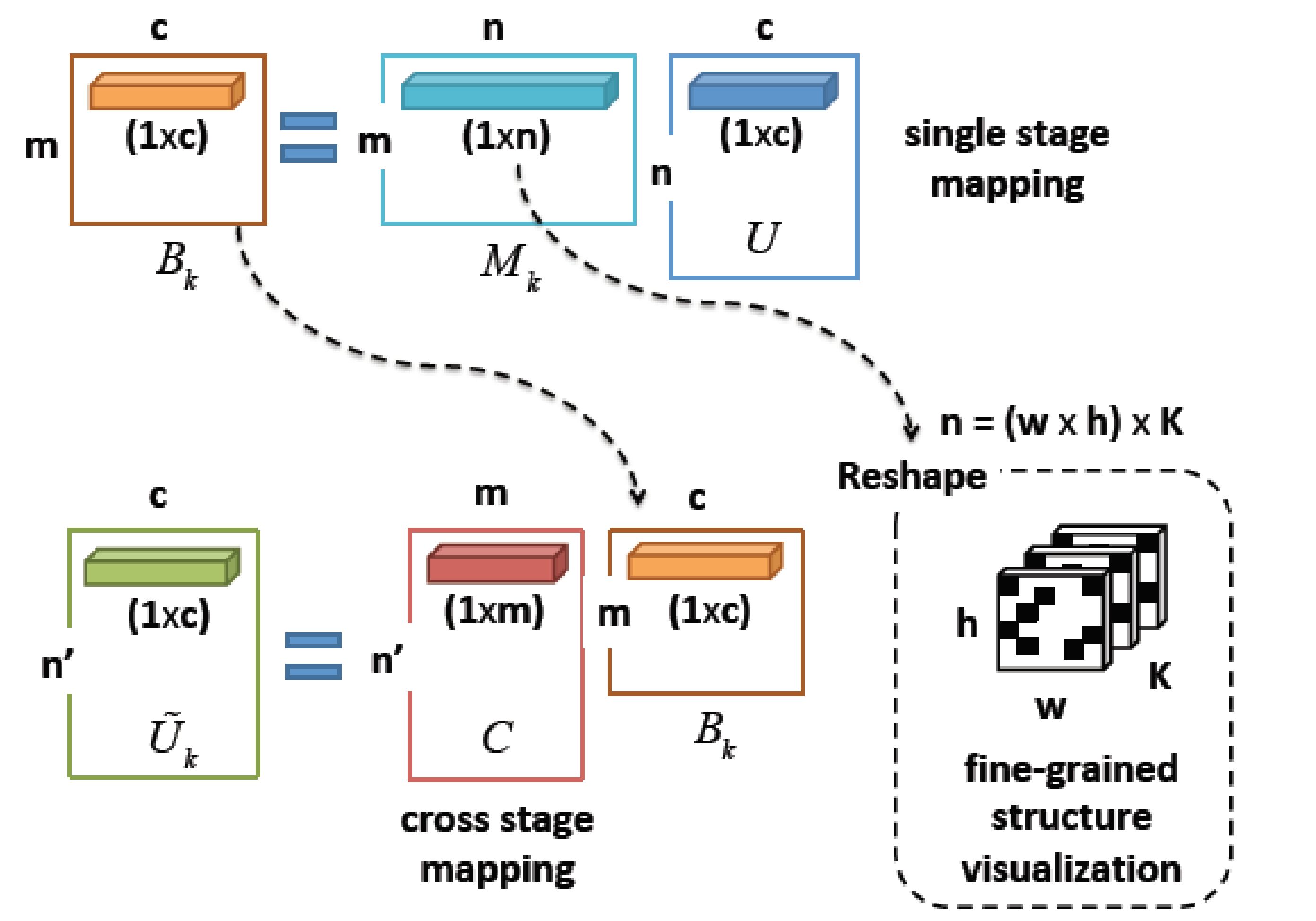
```
else:
    feat_preS = Conv2D(1,(1,1),padding='same',activation='sigmoid')(input_preS)
```

(2) Variance

```
if self.is_varS_model:
    feat_preS = MomentsLayer()(input_preS)
```

(3) Uniform

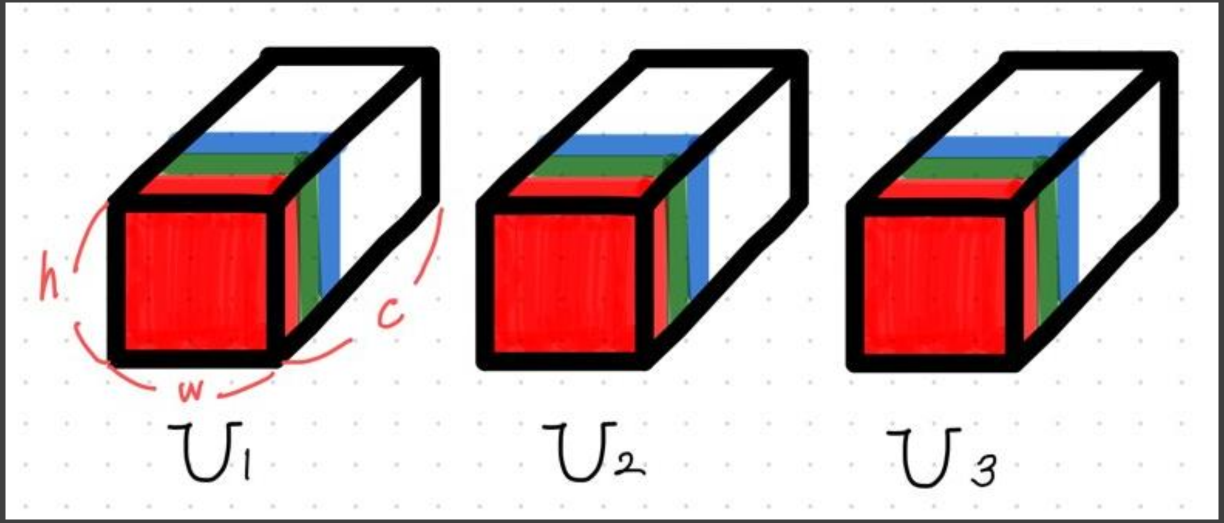
3.5. Fine-grained structure mapping

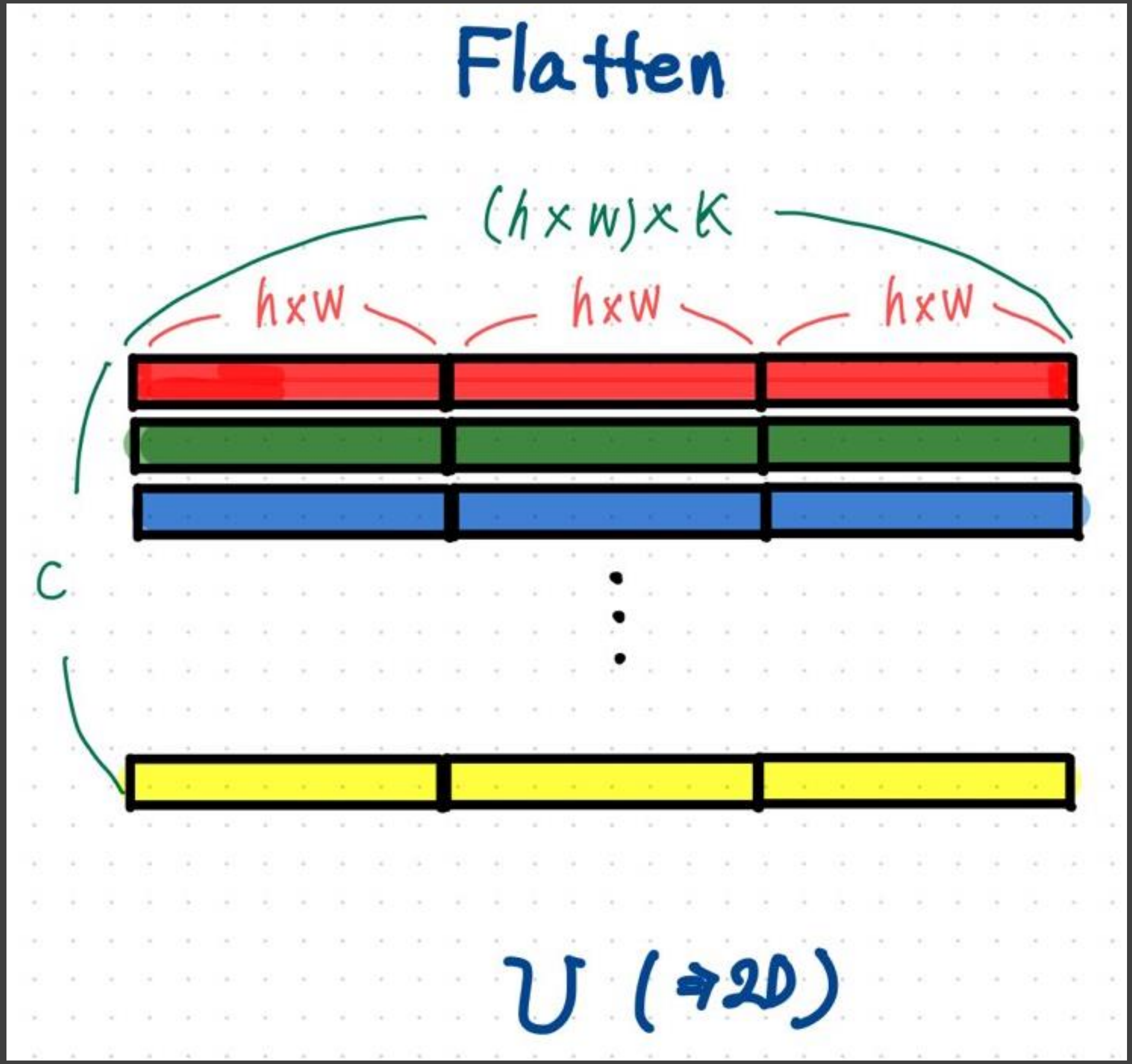


(b) Fine-grained structure mapping

1) 모든 feature map 인 U_k 들을 하나의 matrix U (2D) 로 만든다.

U 에서, n= w x h x K, $U \in R^{n \times c}$





2) K 번째 stage 에서 $\widetilde{U}_k = S_k U$ 를 사용해서 U 에 있는 feature 들을 n’ 개의 대표 feature 들인 $\widetilde{U}_k$ 로 만든다. $S_k \in \mathbb{R}^{n' \times n}$ and $\tilde{U}_k \in \mathbb{R}^{n' \times c}$.

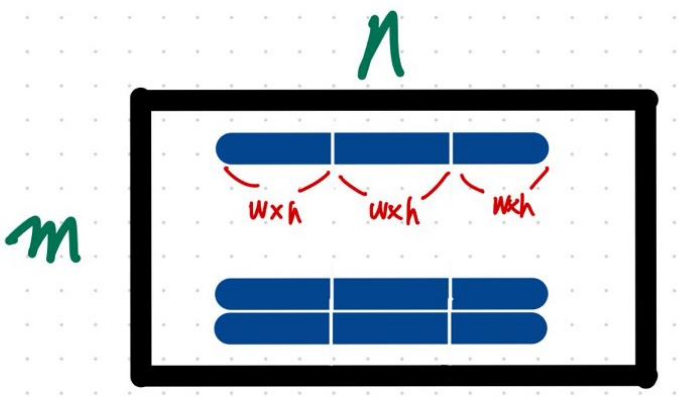
$$S_k = CM_k,$$

$C \in \mathbb{R}^{n' \times m}$, $M_k \in \mathbb{R}^{m \times n}$ and m is a parameter.

$$M_k = \sigma(f_M(A_k)),$$
$$C = \sigma(f_C(A)),$$

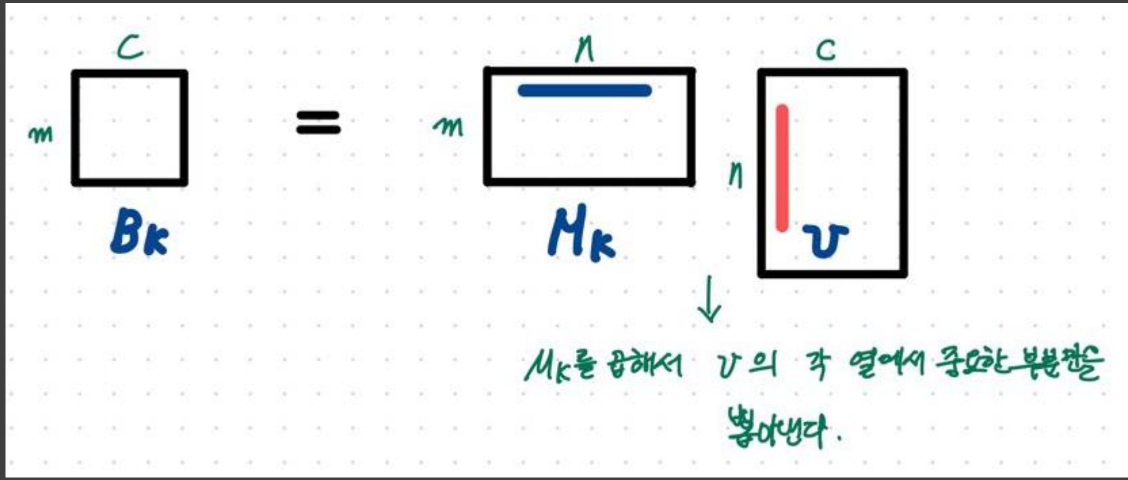
- f_M 과 f_C 는 fully-connected layers 를 정의하는 두 개의 다른 함수이다.
- f_M 과 f_C 는 FSA-Net 에서 training data 로부터 학습을 통해 발견된다.

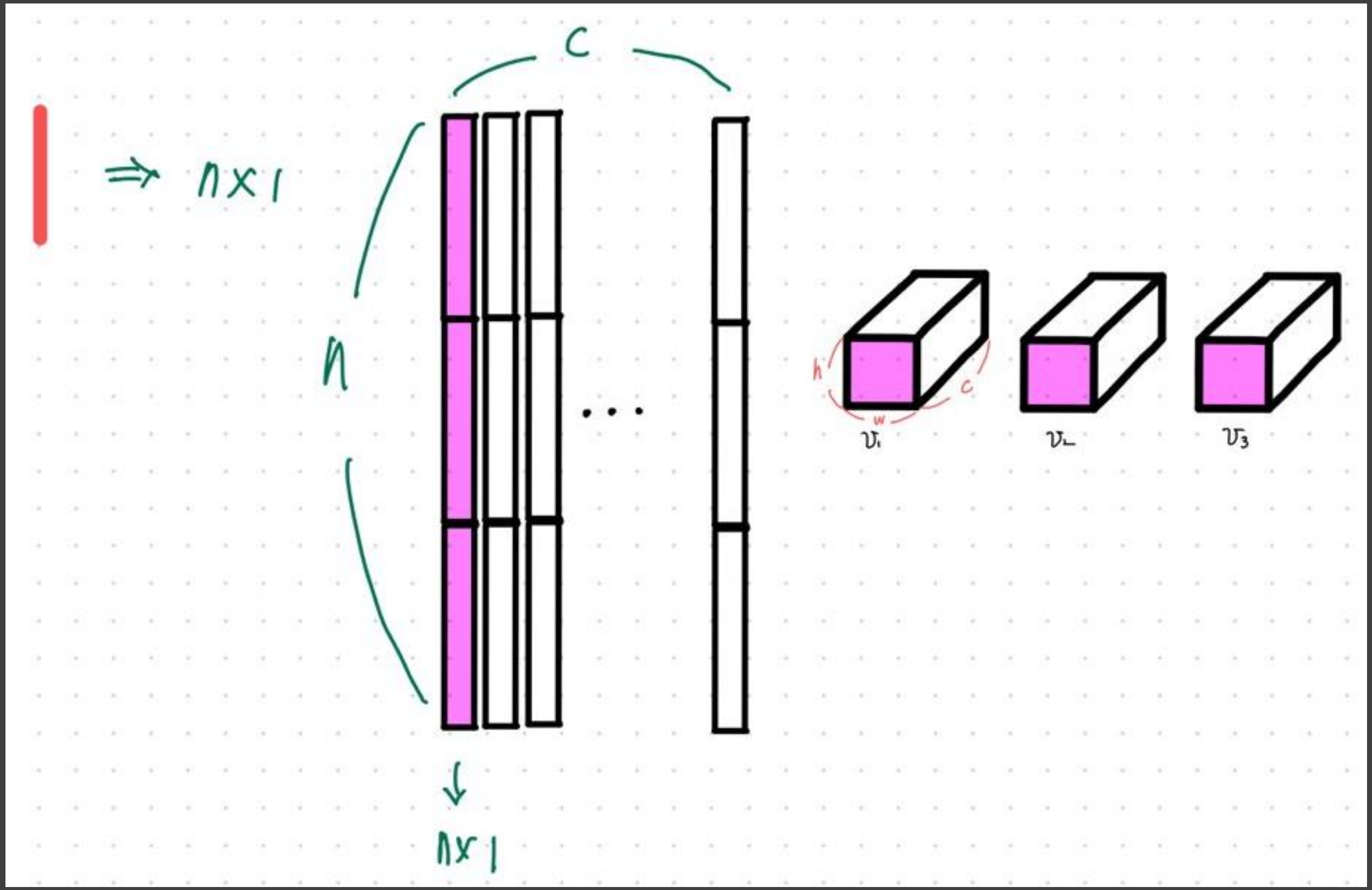
- M_k 의 각 행을 w x h 크기의 K 개의 map 으로 만들 수 있다.



- 각각의 map 은 어떻게 pixel-level features 가 representative feature 에 기여하는지 나타낸다.
- 그러므로 M_k 의 각 행은 fine-grained structure 이다.

- 마지막 set 인 $\widetilde{U} = [\,\widetilde{U}_1, \widetilde{U}_2, \ldots \widetilde{U}_k]$ $\tilde{U} \in \mathbb{R}^{(n' \cdot K) \times c}$.


C
m
=
m
n
c
n
B_K
M_K
v
M_K를 곱해서 v의 각 열에서 중요한 부분만을 뽑아낸다.


⇒ n x 1
C
n
n x 1
h
w
c
v_1
v_2
v_3

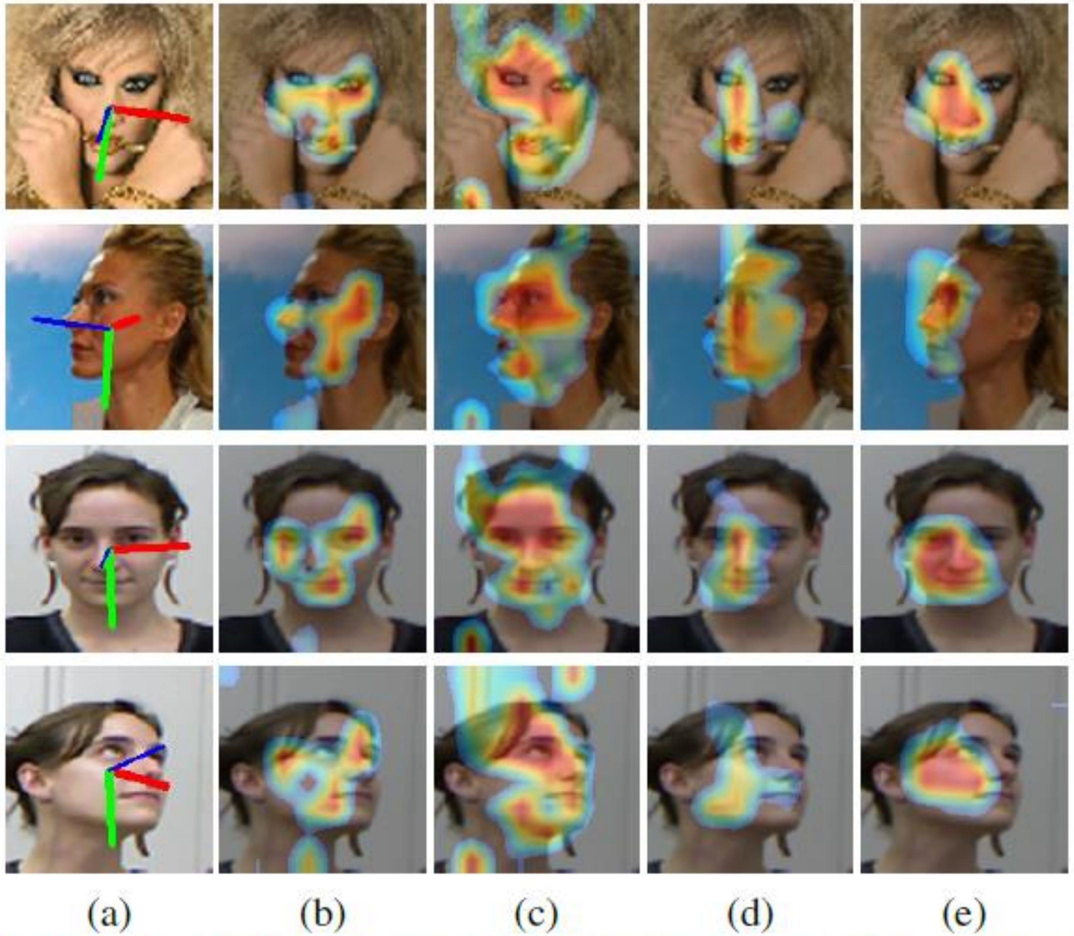
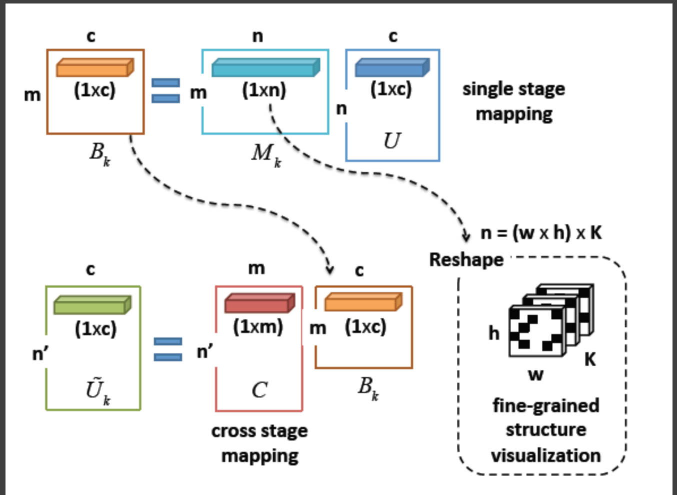
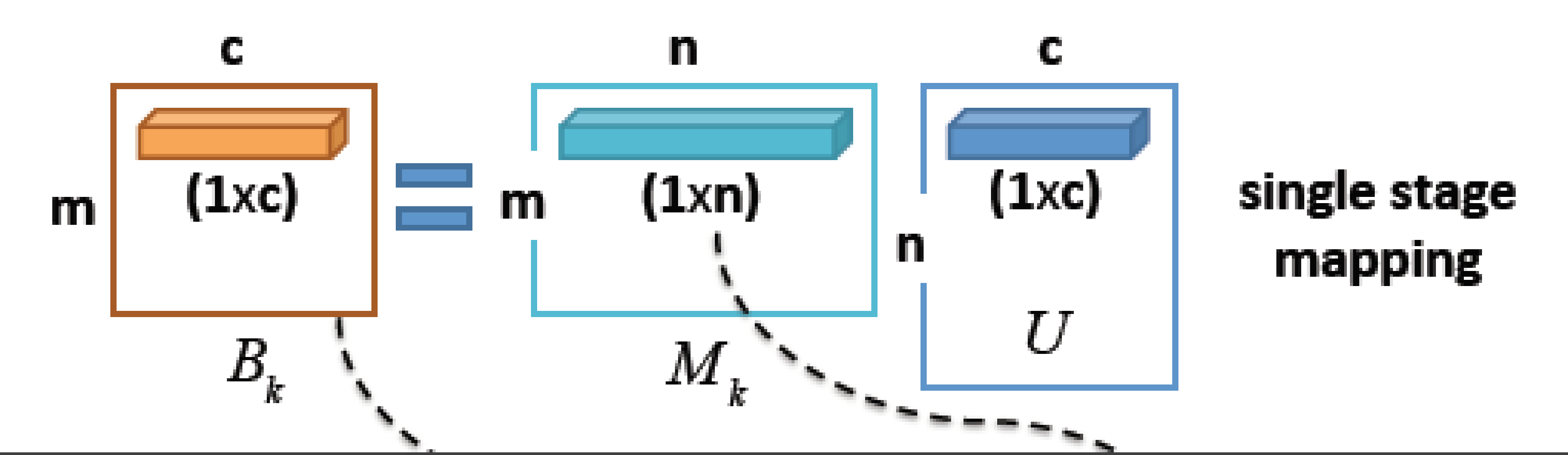


Figure 5. Visualizations of the discovered fine-grained spatial structures. The model is the FSA-Caps (1×1) trained on the 300W-LP dataset. The first column shows the estimated head poses. The other four columns display four spatial structures by heatmaps which visualize the folded versions of some rows of M_k discovered by the model. They show how pixels are aggregated for a specific representative feature. The examples of the first two rows are from the AFLW2000 dataset, and those of the last two rows come from the BIWI dataset.

Making B_k

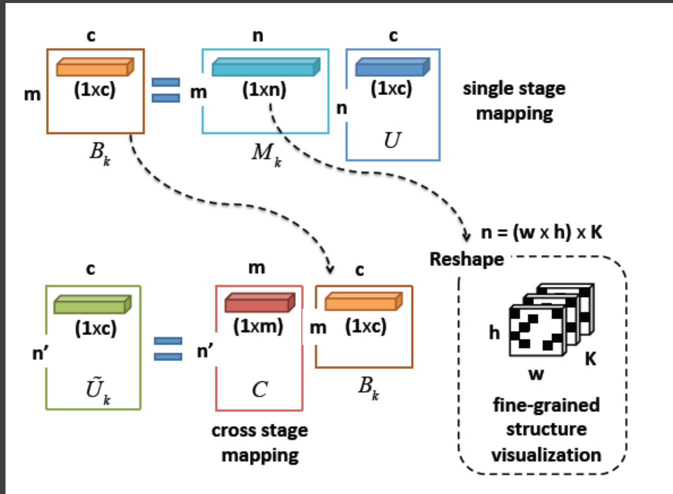


(b) Fine-grained structure mapping

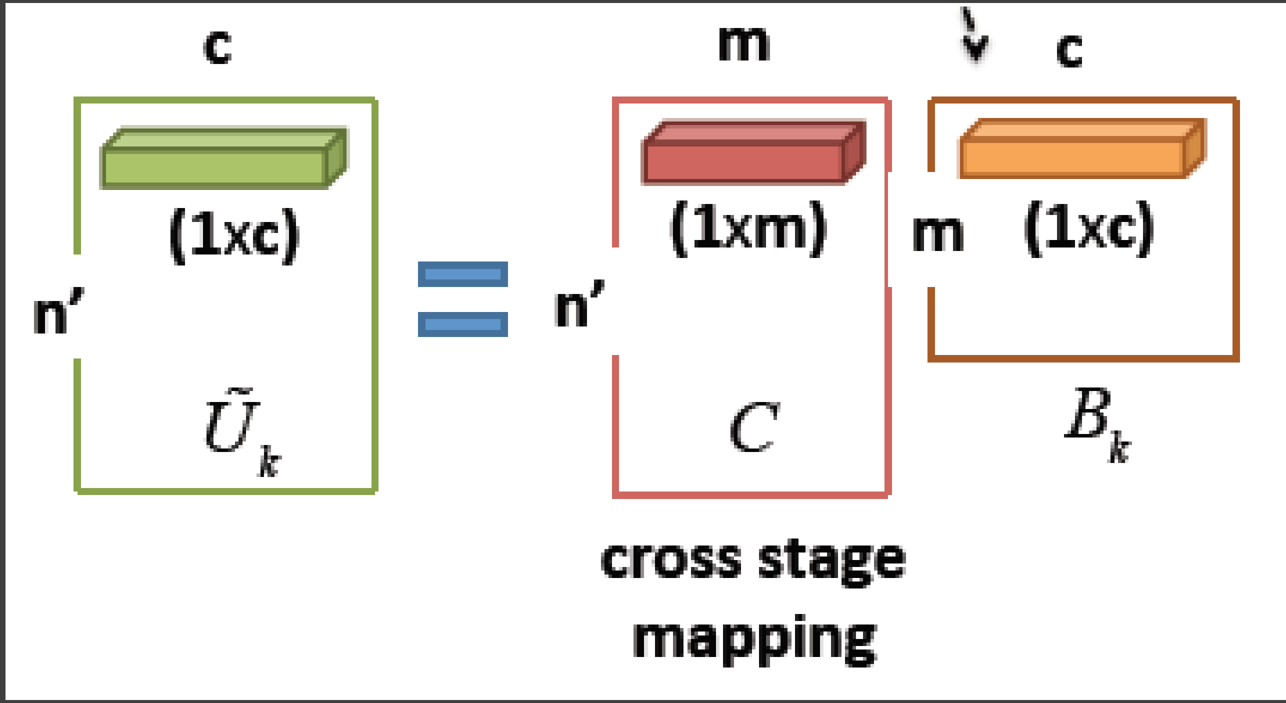


```python
def ssr_feat_S_model_build(self, m_dim): # Scoring function 사용 시 옵션 선택 (conv 또는 variable 중 택 1)
    input_preS = Input((self.map_xy_size,self.map_xy_size,64)) # (8, 8, 64) --> (w, h, c)

    if self.is_varS_model: # option 으로 variable 사용
        feat_preS = MomentsLayer()(input_preS)
    else: # option 으로 conv 사용
        feat_preS = Conv2D(1,(1,1),padding='same',activation='sigmoid')(input_preS)

    # Scoring function 적용한 결과값이 feat_preS = A(k)
    feat_preS = Reshape((-1,))(feat_preS) # Reshape((-1,)) : 행의 갯수를 알아서 잘 설정해줌

    # feat_preS로 SR_matrix 만들기
    SR_matrix = Dense(m_dim*(self.map_xy_size*self.map_xy_size*3),activation='sigmoid')(feat_preS) # Dense(m*w*h*3) = Dense(m*w*h*K) = Dense(m*n)
    SR_matrix = Reshape((m_dim,(self.map_xy_size*self.map_xy_size*3)))(SR_matrix) # Reshape((m, w*h)) == (5, 64) == (m, c) == B(k)

    return Model(inputs=input_preS,outputs=[SR_matrix,feat_preS],name='feat_S_model')
```

Making $\tilde{U}_k$



(b) Fine-grained structure mapping



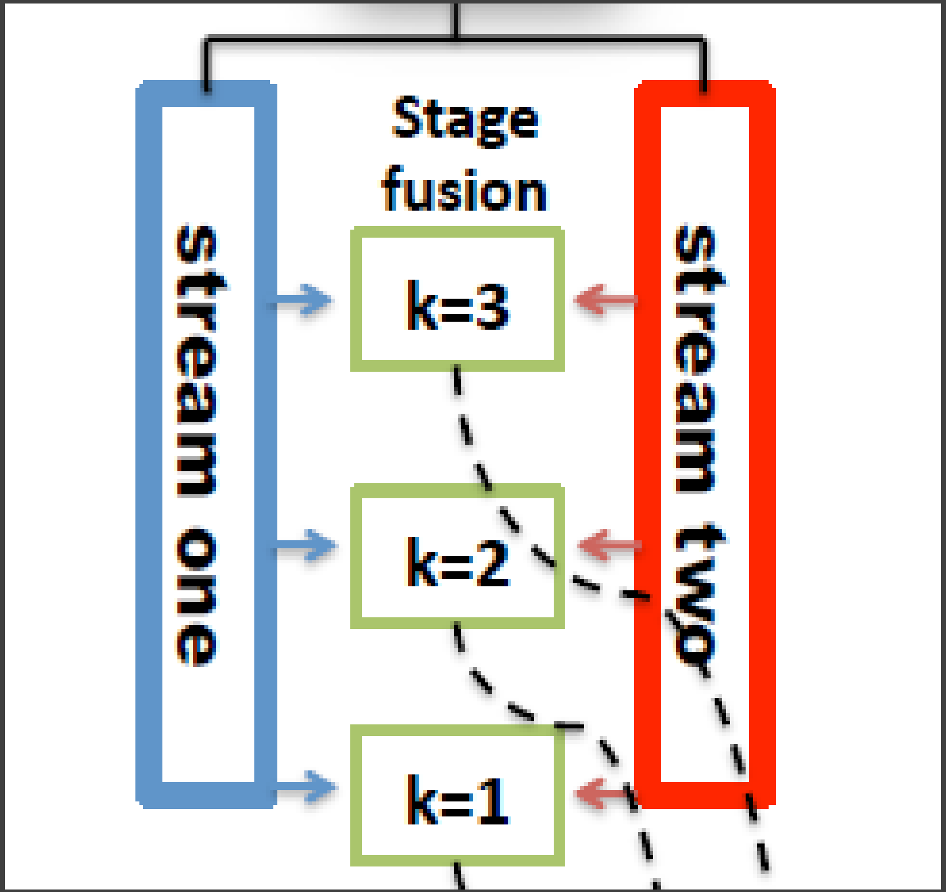
```python
def ssr_S_model_build(self, num_primcaps, m_dim): # Scoring function 사용 함
    input_s1_preS = Input((self.map_xy_size,self.map_xy_size,64)) # (8, 8, 64) --> (w, h, c)
    input_s2_preS = Input((self.map_xy_size,self.map_xy_size,64)) # (8, 8, 64)
    input_s3_preS = Input((self.map_xy_size,self.map_xy_size,64)) # (8, 8, 64)

    # Scoring function 선택 시 옵션 선택 (conv 또는 variable 중 택 1)
    feat_S_model = self.ssr_feat_S_model_build(m_dim) # ssr_feat_S_model_build(m=5)

    SR_matrix_s1,feat_s1_preS = feat_S_model(input_s1_preS) # Scoring function 적용 --> SR_matrix_s1 : (m, c)
    SR_matrix_s2,feat_s2_preS = feat_S_model(input_s2_preS) # Scoring function 적용 --> SR_matrix_s2 : (m, c)
    SR_matrix_s3,feat_s3_preS = feat_S_model(input_s3_preS) # Scoring function 적용 --> SR_matrix_s3 : (m, c)

    feat_pre_concat = Concatenate()([feat_s1_preS,feat_s2_preS,feat_s3_preS]) # feat_pre_concat : A (= A1 + A2 + A3)

    # S(k)의 요소인 C 만들기 --> SL_matrix 는 C 이다.
    SL_matrix = Dense(int(num_primcaps/3)*m_dim,activation='sigmoid')(feat_pre_concat) # Dense((7*3/3)*3,5) --> Dense(21,5)
    SL_matrix = Reshape((int(num_primcaps/3),m_dim))(SL_matrix) # (n',m) == C

    # C * B(k) = tilda U(k)
    S_matrix_s1 = MatrixMultiplyLayer(name="S_matrix_s1")([SL_matrix,SR_matrix_s1]) # C * SR_matrix_s1(B(k)) --> (n', m) * (m, c) = (n', c)
    S_matrix_s2 = MatrixMultiplyLayer(name='S_matrix_s2')([SL_matrix,SR_matrix_s2]) # C * SR_matrix_s2
    S_matrix_s3 = MatrixMultiplyLayer(name='S_matrix_s3')([SL_matrix,SR_matrix_s3]) # C * SR_matrix_s3
```

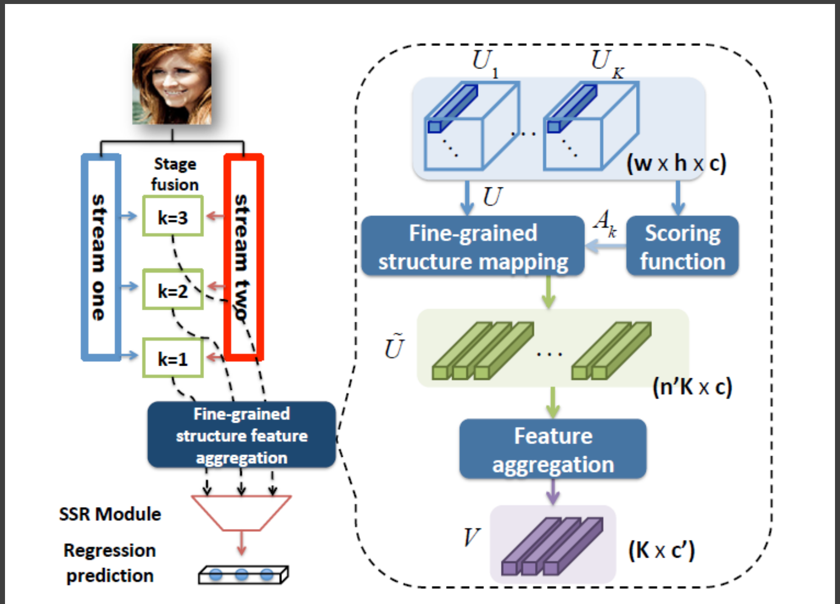
```python
# Very important!!! Without this training won't converge.
# norm_S_s1 = Lambda(lambda x: K.tile(K.sum(x,axis=-1,keepdims=True),(1,1,64)))(S_matrix_s1)
norm_S_s1 = MatrixNormLayer(tile_count=64)(S_matrix_s1)
norm_S_s2 = MatrixNormLayer(tile_count=64)(S_matrix_s2)
norm_S_s3 = MatrixNormLayer(tile_count=64)(S_matrix_s3)

feat_s1_pre = Reshape((self.map_xy_size*self.map_xy_size,64))(input_s1_preS) # (8*8, 64)
feat_s2_pre = Reshape((self.map_xy_size*self.map_xy_size,64))(input_s2_preS) # (8*8, 64)
feat_s3_pre = Reshape((self.map_xy_size*self.map_xy_size,64))(input_s3_preS) # (8*8, 64)

feat_pre_concat = Concatenate(axis=1)([feat_s1_pre, feat_s2_pre, feat_s3_pre]) # (8*8, 64*3)

# Warining: don't use keras's 'K.dot'. It is very weird when high dimension is used.
# https://github.com/keras-team/keras/issues/9779
# Make sure 'tf.matmul' is used
# primcaps = Lambda(lambda x: tf.matmul(x[0],x[1])/x[2])([S_matrix,feat_pre_concat, norm_S])
# B(k) * (c, n)
primcaps_s1 = PrimCapsLayer()([S_matrix_s1,feat_pre_concat, norm_S_s1]) # (S_matrix_s1 (matmul) feat_pre_concat)/norm_S_s1 1
primcaps_s2 = PrimCapsLayer()([S_matrix_s2,feat_pre_concat, norm_S_s2])
primcaps_s3 = PrimCapsLayer()([S_matrix_s3,feat_pre_concat, norm_S_s3])

primcaps = Concatenate(axis=1)([primcaps_s1,primcaps_s2,primcaps_s3])

return Model(inputs=[input_s1_preS, input_s2_preS, input_s3_preS],outputs=primcaps, name='ssr_S_model')
```

Aggregation

```
# Aggregation
def ssr_aggregation_model_build(self, shape_primcaps):
    input_primcaps = Input(shape_primcaps)
    capsule = CapsuleLayer(self.num_capsule, self.dim_capsule, routings=self.routings, name='caps')(input_primcaps)

    feat_s1_div, feat_s2_div, feat_s3_div = AggregatedFeatureExtractionLayer(num_capsule=self.num_capsule)(capsule)

    feat_s1_div = Reshape((-1,))(feat_s1_div) # capsule[:,0:1,:]
    feat_s2_div = Reshape((-1,))(feat_s2_div) # capsule[:,1:2,:]
    feat_s3_div = Reshape((-1,))(feat_s3_div) # capsule[:,2:3,:]

    return Model(inputs=input_primcaps,outputs=[feat_s1_div,feat_s2_div,feat_s3_div], name='ssr_Cap_model')
```

3.6. Details of the architecture





(a) FSA-Net

3.6. Details of the architecture

- FSA-Net 의 두 개의 stream 은 B_R, B_T 를 사용함.

$$\mathrm{B_R(c)} \equiv \{\mathrm{SepConv2D}(3\times3,\ \mathrm{c})\text{-BN-ReLU}\},$$
$$\mathrm{B_T(c)} \equiv \{\mathrm{SepConv2D}(3\times3,\ \mathrm{c})\text{-BN-Tanh}\},$$

- Structure of First stream
  - {B_R (16)-AvgPool(22)-B_R(32)-B_R(32)-AvgPool(22)} – {B_R(64)-B_R(64)-AvgPool(22)}- {B_R(128)-B_R(128)}

- Structure of Second stream
  - {B_T(16)-MaxPool(22)-B_T(32)-B_T(32)-MaxPool(22)}- {B_T(64)-B_T(64)-MaxPool(22)} – {B_T(128)-B_T(128)}

- K=3, Feature map 에서 w=8, h=8, c=64, Fine-grained structure 에서 m=5, n'=7

- Structure of First stream

```
x = self._convBlock(img_inputs, num_filters=16, activation='relu')
x_layer1 = AveragePooling2D((2,2))(x)
x = self._convBlock(x_layer1, num_filters=32, activation='relu')
x = self._convBlock(x, num_filters=32, activation='relu')
x_layer2 = AveragePooling2D((2,2))(x)
x = self._convBlock(x_layer2, num_filters=64, activation='relu')
x = self._convBlock(x, num_filters=64, activation='relu')
x_layer3 = AveragePooling2D((2,2))(x)
x = self._convBlock(x_layer3, num_filters=128, activation='relu')
x_layer4 = self._convBlock(x, num_filters=128, activation='relu')
```

- Structure of Second stream

```
s = self._convBlock(img_inputs, num_filters=16, activation='tanh')
s_layer1 = MaxPooling2D((2,2))(s)
s = self._convBlock(s_layer1, num_filters=32, activation='tanh')
s = self._convBlock(s, num_filters=32, activation='tanh')
s_layer2 = MaxPooling2D((2,2))(s)
s = self._convBlock(s_layer2, num_filters=64, activation='tanh')
s = self._convBlock(s, num_filters=64, activation='tanh')
s_layer3 = MaxPooling2D((2,2))(s)
s = self._convBlock(s_layer3, num_filters=128, activation='tanh')
s_layer4 = self._convBlock(s, num_filters=128, activation='tanh')
```

Q & A

Thank You~