Hyper-correlation Squeeze for Few-Shot Segmentation

POSTECH CSE & GSAI arxiv.2021

Fang Zhiyuan

2021.6.18

Prior knowledge

Few shot segmentation

- Defects of Deep-learning-based Segmentation
  - Eager for large amount of samples of target
  - Cannot segment unseen objects
- The aims of Few-shot Segmentation
  - Segment unseen object with only a few references after training

Prior knowledge

Few shot segmentation

- Divide classes C of dataset into seen C_{seen} and unseen C_{unseen}, $C\_seen \cap C\_unseen = \emptyset$
- $D_{train} = \{(S_i, Q_i)\}_{i=1}^{N_{train}}$, $D_{test} = \{(S_i, Q_i)\}_{i=1}^{N_{test}}$
- $S_i = \{(I_{c,k}, M_{c,k}) \mid c \in c_{seen}$ for training $else\ c \in c_{unseen}\}$, $Q_i = \{(I_{c,n}, M_{c,n})\}$
- (S_i, Q_i) called episode is the input to the model during training/testing stage

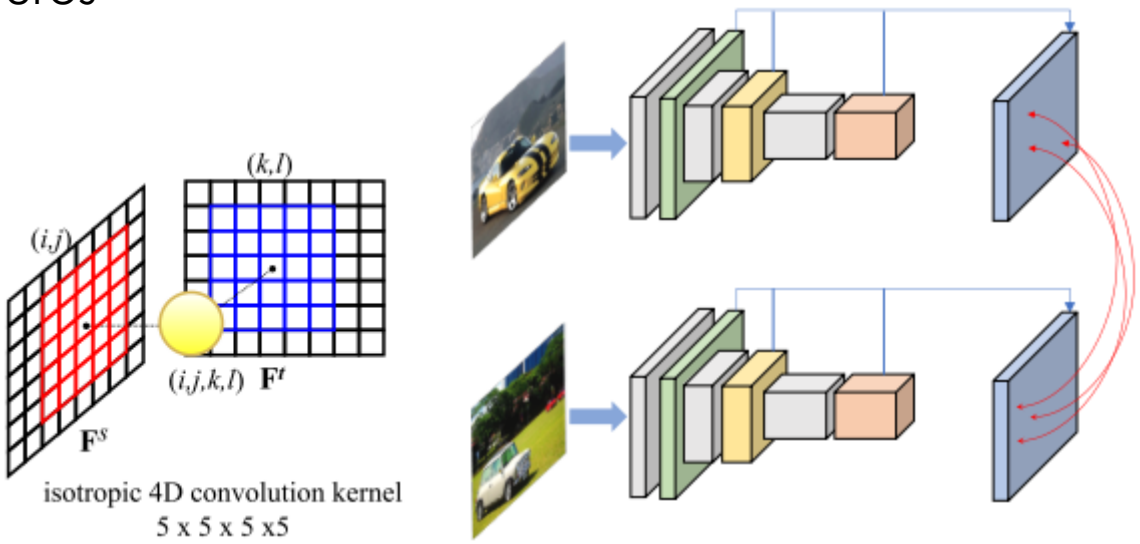
Prior knowledge

Hyper-correlation

- A collection of 4D correlation tensors

Learning visual correspondence

- Find reliable correspondences under challenging degree of variations
- Build correspondences upon convolutional features pretrained on classification task
- Exploiting different levels of convolutional features [1]
- Employ 4D convolutions on dense feature [2]



1. Robust Image Matching By Dynamic Feature Selection – BMVC 2020
2. Correspondence Networks with Adaptive Neighborhood Consensus – CVPR 2020

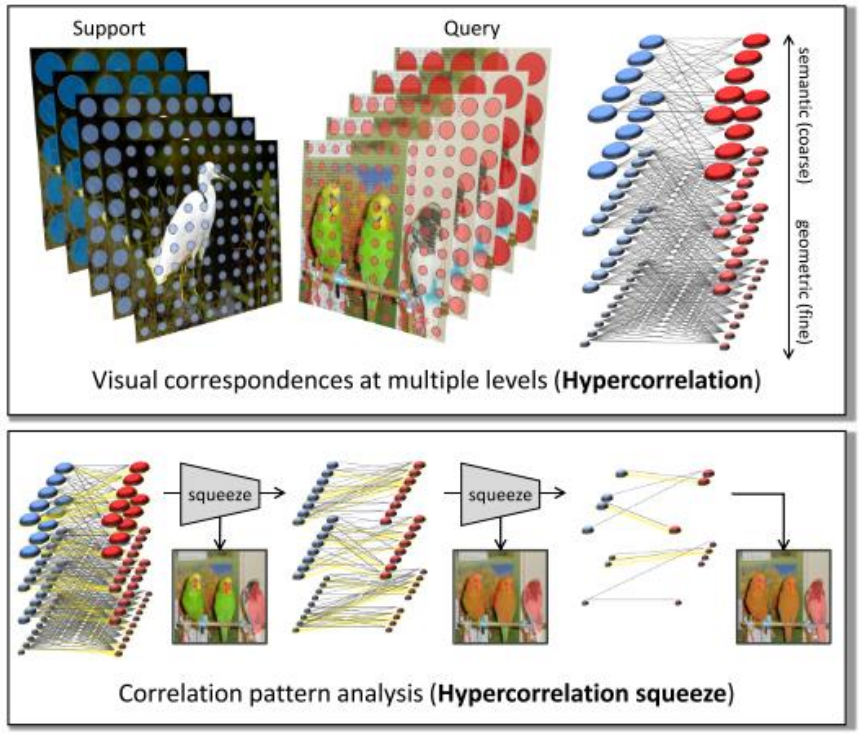
Motivation

Few shot semantic segmentation equals to:

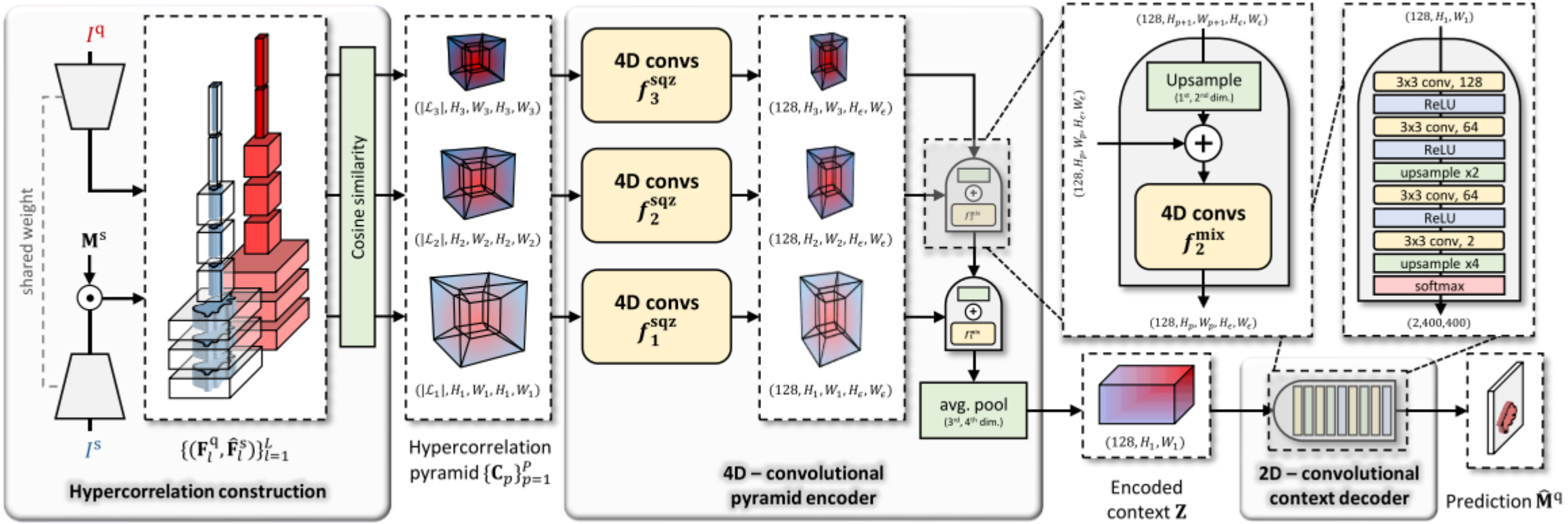
- understand diverse levels of visual cues
- analyze fine-grained correspondence relations between the query and the support images

Proposed methods

- multi-level feature correlation
- efficient 4D convolutions

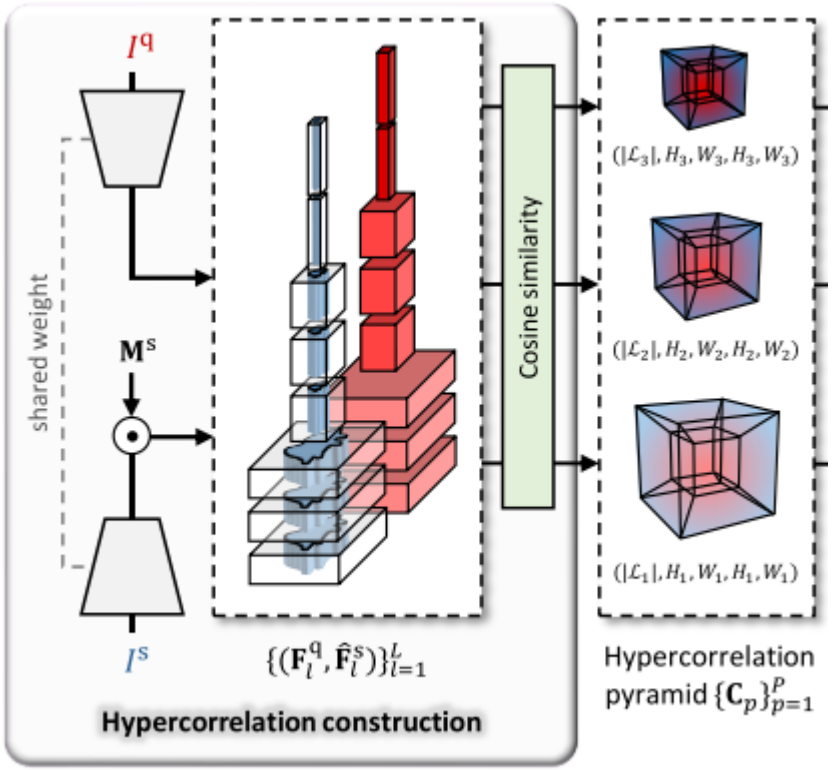


Method


shared weight
Cosine similarity
Hypercorrelation construction
Hypercorrelation pyramid
4D convs
4D convs
4D convs
4D – convolutional pyramid encoder
avg. pool
Upsample
4D convs
Encoded context
2D – convolutional context decoder
3x3 conv, 128
ReLU
3x3 conv, 64
ReLU
upsample x2
3x3 conv, 64
ReLU
3x3 conv, 2
upsample x4
softmax
(2,400,400)
Prediction

Method

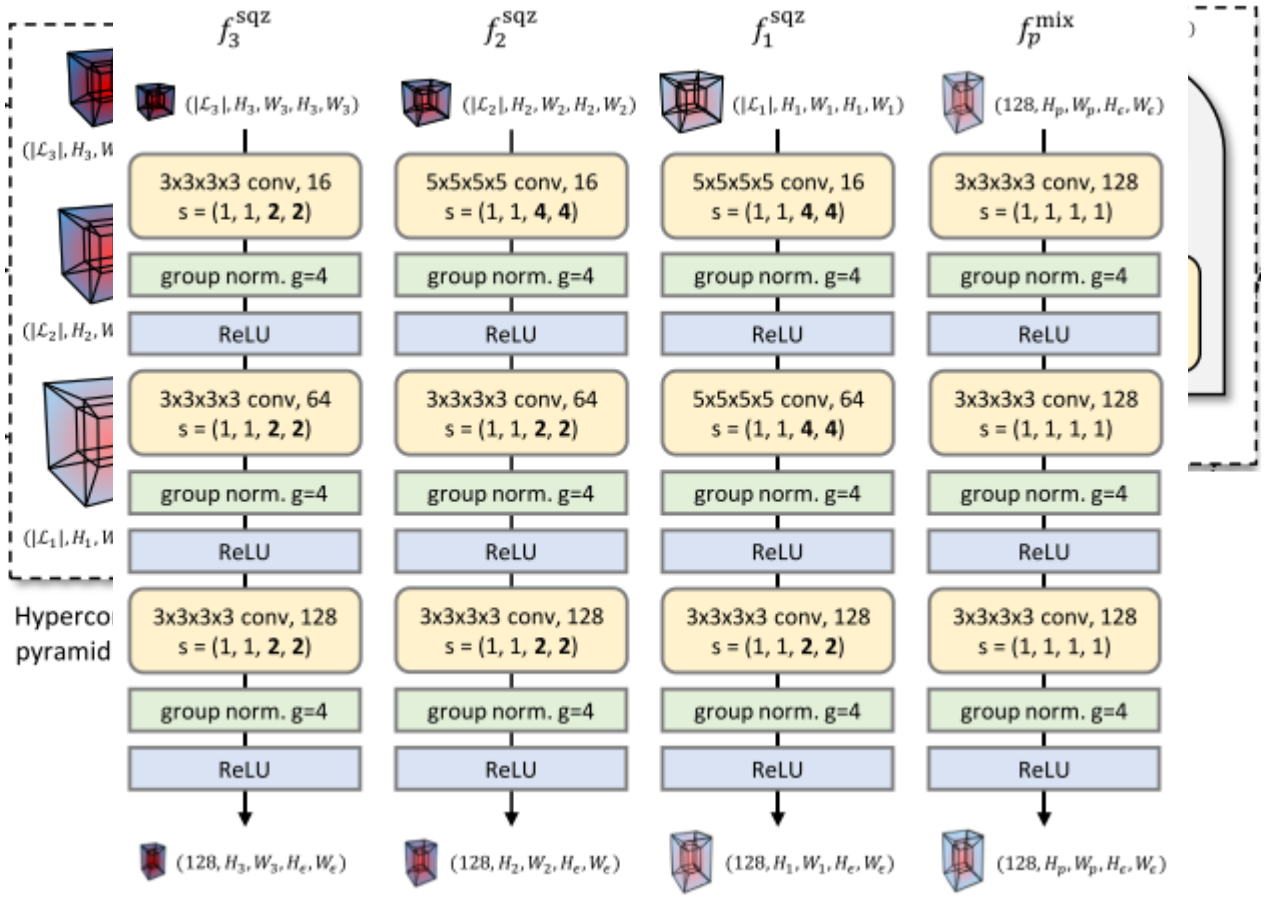
- Capture multi-level semantic and geometric patterns
- Query and support images $I^q, I^s \in \mathbb{R}^{3\times H\times W}$
- L pairs of intermediate feature maps $\{(F_l^q, F_l^s)\}_{l=1}^{L}$
- Mask each F_l^s with support mask $M^s \in \{0,1\}^{H\times W}$:
  - $\hat{F}_l^s = F_l^s \odot \zeta(M^s)$
  - $\odot$ - Hadamard product
  - ζ - bilinearly interpolates
- Form a 4D correlation tensor $\hat{C}_l \in \mathbb{R}^{H_l\times W_l\times H_l\times W_l}$:
  - $\hat{C}_l(x^q, x^s) = ReLU\left(\frac{F_l^q(x^q)\cdot \hat{F}_l^s(x^s)}{\|F_l^q(x^q)\|\|\hat{F}_l^s(x^s)\|}\right)$



- Collect 4D tensors having the same spatial sizes $\{\hat{C}_l\}_{l\in\mathcal{L}_p}$ at some pyramidal layer p
- Form a hyper-correlation $C_p \in \mathbb{R}^{|\mathcal{L}_p|\times H_p\times W_p\times H_p\times W_p}$ by concatenating $\{\hat{C}_l\}_{l\in\mathcal{L}_p}$ along channel.

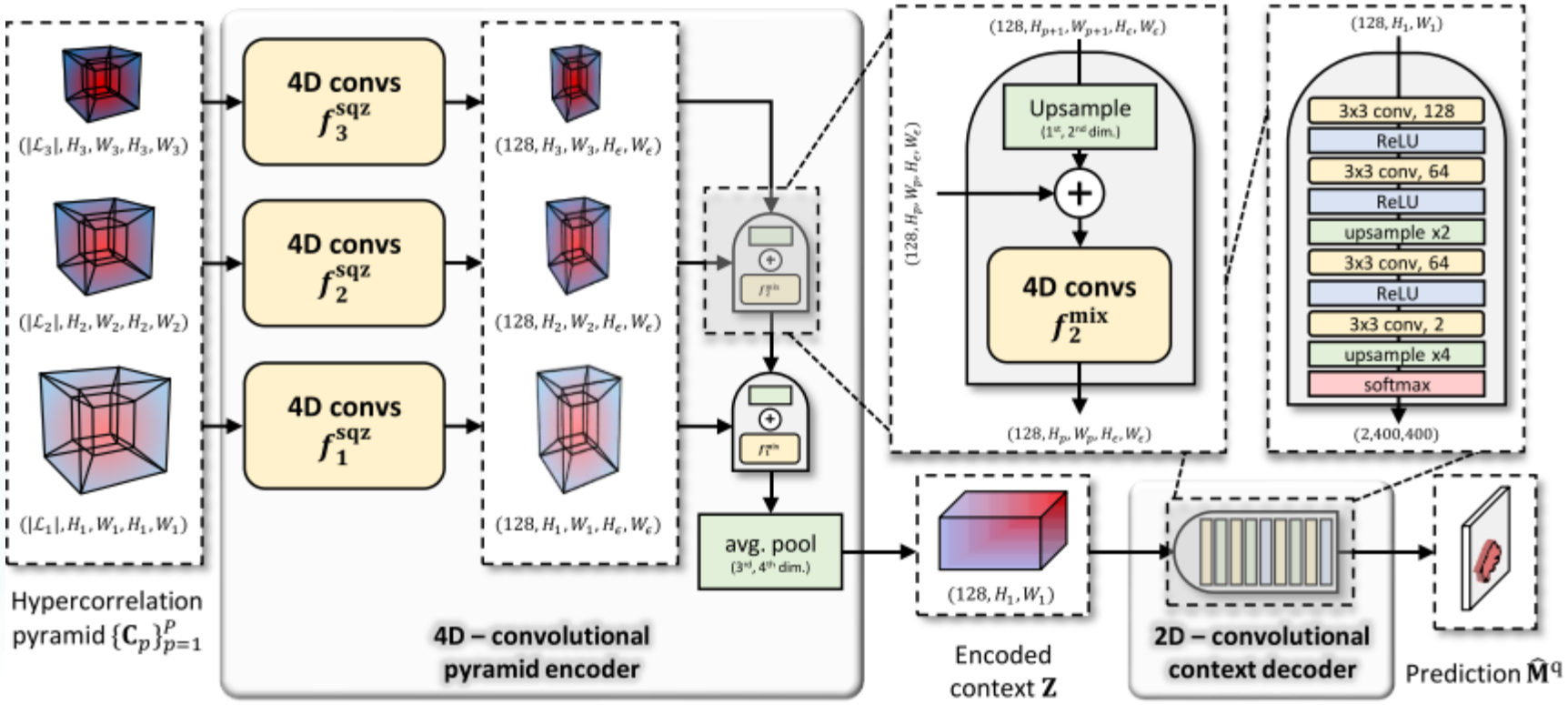
Method

- Squeeze $C = \{C_P\}_{p=1}^{P}$ into $Z \in \mathbb{R}^{128 \times H_1 \times W_1}$
- Two blocks: f_p^{sqz} and f_p^{mix}
- Each block consists of
  - multi-channel 4D convolution
  - Group normalization
  - ReLU
- f_p^{sqz} squeeze last two (support) spatial dimensions
- f_p^{mix} has a FPN-like fuse structure
  - Up-sample first two spatial (query) dimensions
  - element-wise addition
- Average pooling the out of f_1^{mix} on last two (support) spatial dimensions



Method

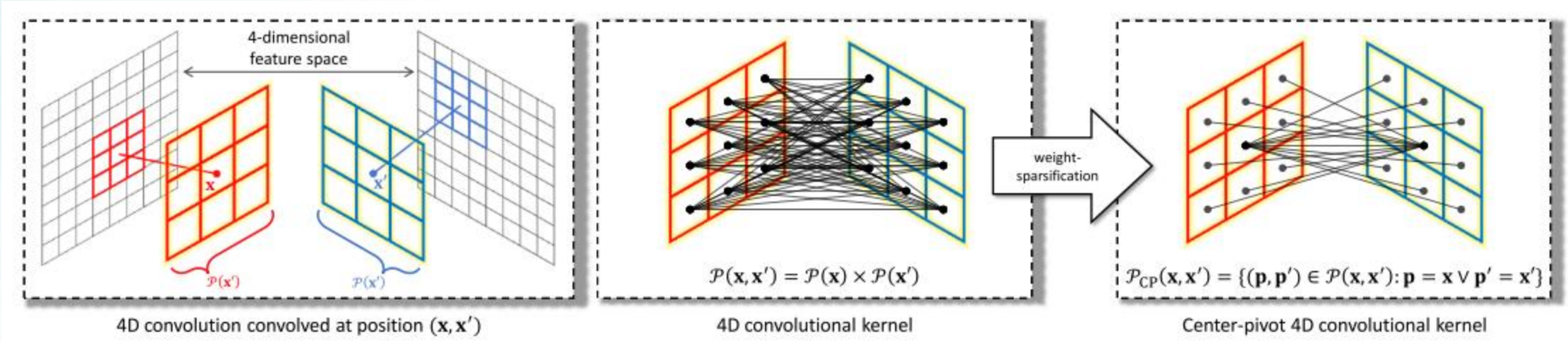
- 2D-convolutional context decoder
  - 2D convolutions
  - ReLU
  - Upsampling layers
  - Softmax



Method

Center-pivot 4D convolution

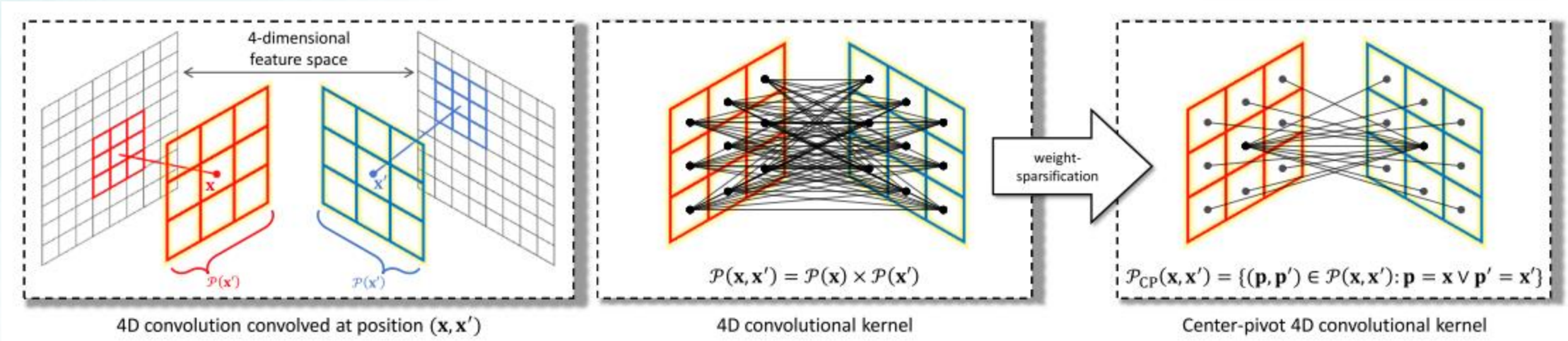
- 4D convolution and its limitation:
  - quadratic complexity is too high
  - over-parameterization of the high-dimensional kernel



Method

Center-pivot 4D convolution

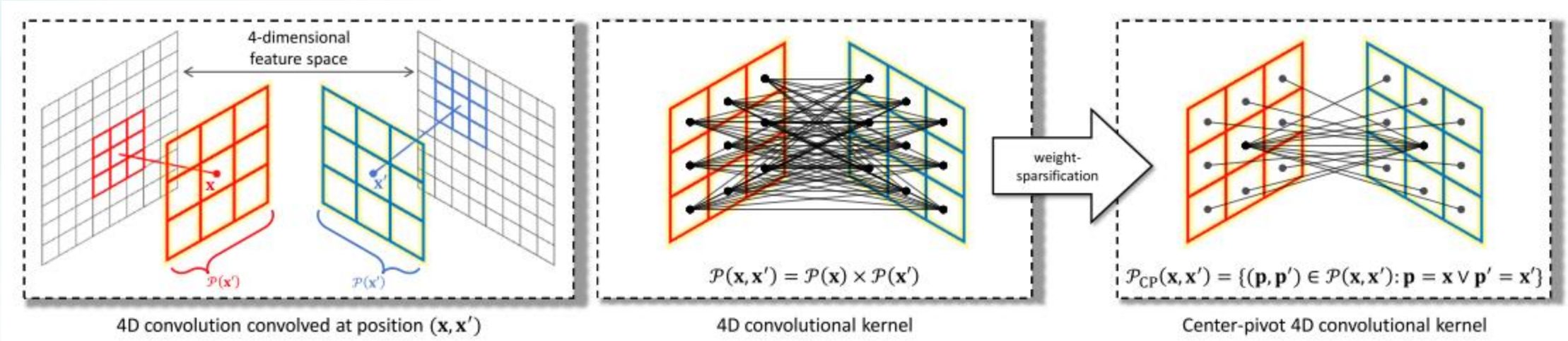
- Goal: design a lightweight and efficient 4D kernel
- Main idea: disregard a large number of activations
- Given 4D position (x, x'), define two respective sets:
  - $P_c(x, x') = \{(p, p') \in P(x, x') : p = x\}$
  - $P_{c'}(x, x') = \{(p, p') \in P(x, x') : p' = x'\}$



Method

Center-pivot 4D convolution

- Origin: $(c * k)(x, x') = \sum_{(p,p')\in P(x,x')} c(p, p')k(p - x, p' - x')$
- Improve: $(c * k_{CP})(x, x') = \sum_{p'\in P(x')} c(x, p')k_c^{2D}(p' - x') + \sum_{p\in P(x)} c(p.x')k_{c'}^{2D}(p - x)$



Experiments

Backbone network	Methods	1-shot 5^0	5^1	5^2	5^3	mean	FB-IoU	5-shot 5^0	5^1	5^2	5^3	mean	FB-IoU	# learnable params
VGG16 [61]	OSLSM [58]	33.6	55.3	40.9	33.5	40.8	61.3	35.9	58.1	42.7	39.1	43.9	61.5	276.7M
	co-FCN [51]	36.7	50.6	44.9	32.4	41.1	60.1	37.5	50.0	44.1	33.9	41.4	60.2	34.2M
	AMP-2 [60]	41.9	50.2	46.7	34.7	43.4	61.9	40.3	55.3	49.9	40.1	46.4	62.1	15.8M
	PANet [72]	42.3	58.0	51.1	41.2	48.1	66.5	51.8	64.6	59.8	46.5	55.7	70.7	14.7M
	PFENet [67]	56.9	**68.2**	54.4	52.4	58.0	72.0	59.0	**69.1**	54.8	52.9	59.0	72.3	10.4M
	HSNet (ours)	**59.6**	65.7	**59.6**	**54.0**	**59.7**	**73.4**	**64.9**	69.0	**64.1**	**58.6**	**64.1**	**76.6**	**2.6M**
ResNet50 [17]	PANet [72]	44.0	57.5	50.8	44.0	49.1	-	55.3	67.2	61.3	53.2	59.3	-	23.5M
	PGNet [82]	56.0	66.9	50.6	50.4	56.0	69.9	57.7	68.7	52.9	54.6	58.5	70.5	17.2M
	PPNet [35]	48.6	60.6	55.7	46.5	52.8	69.2	58.9	68.3	66.8	58.0	63.0	75.8	31.5M
	PFENet [67]	61.7	69.5	55.4	56.3	60.8	73.3	63.1	70.7	55.8	57.9	61.9	73.9	10.8M
	RePRI [4]	59.8	68.3	**62.1**	48.5	59.7	-	64.6	71.4	**71.1**	59.3	66.6	-	-
	HSNet (ours)	**64.3**	**70.7**	60.3	**60.5**	**64.0**	**76.7**	**70.3**	**73.2**	67.4	**67.1**	**69.5**	**80.6**	**2.6M**
ResNet101 [17]	FWB [43]	51.3	64.5	56.7	52.2	56.2	-	54.8	67.4	62.2	55.3	59.9	-	43.0M
	PPNet [35]	52.7	62.8	57.4	47.7	55.2	70.9	60.3	70.0	**69.4**	60.7	65.1	77.5	50.5M
	DAN [71]	54.7	68.6	57.8	51.6	58.2	71.9	57.9	69.0	60.1	54.9	60.5	72.3	-
	PFENet [67]	60.5	69.4	54.4	55.9	60.1	72.9	62.8	70.4	54.9	57.6	61.4	73.5	10.8M
	RePRI [4]	59.6	68.6	**62.2**	47.2	59.4	-	66.2	71.4	67.0	57.7	65.6	-	-
	HSNet (ours)	**67.3**	**72.3**	62.0	**63.1**	**66.2**	**77.6**	**71.8**	**74.4**	67.0	**68.3**	**70.4**	**80.6**	**2.6M**
	HSNet† (ours)	66.2	69.5	53.9	56.2	61.5	72.5	68.9	71.9	56.3	57.9	63.7	73.8	**2.6M**

Table 1: Performance on PASCAL-5^i [58] in mIoU and FB-IoU. Some results are from [4, 35, 67, 71, 76]. Superscript † denotes our model ***without*** support feature masking (Eqn. 1). Numbers in bold indicate the best performance and underlined ones are the second best.

Experiments

Backbone network	Methods	1-shot 20^0	20^1	20^2	20^3	mean	FB-IoU	5-shot 20^0	20^1	20^2	20^3	mean	FB-IoU
ResNet50 [17]	PPNet [35]	28.1	30.8	29.5	27.7	29.0	-	39.0	40.8	37.1	37.3	38.5	-
	PMM [76]	29.3	34.8	27.1	27.3	29.6	-	33.0	40.6	30.3	33.3	34.3	-
	RPMM [76]	29.5	36.8	28.9	27.0	30.6	-	33.8	42.0	33.0	33.3	35.5	-
	PFENet [67]	**36.5**	38.6	34.5	33.8	35.8	-	36.5	43.3	37.8	38.4	39.0	-
	RePRI [4]	32.0	38.7	32.7	33.1	34.1	-	39.3	45.4	39.7	41.8	41.6	-
	HSNet (ours)	36.3	**43.1**	**38.7**	**38.7**	**39.2**	**68.2**	**43.3**	**51.3**	**48.2**	**45.0**	**46.9**	**70.7**
ResNet101 [17]	FWB [43]	17.0	18.0	21.0	28.9	21.2	-	19.1	21.5	23.9	30.1	23.7	-
	DAN [71]	-	-	-	-	24.4	62.3	-	-	-	-	29.6	63.9
	PFENet [67]	36.8	41.8	38.7	36.7	38.5	63.0	40.4	46.8	43.2	40.5	42.7	65.8
	HSNet (ours)	**37.2**	**44.1**	**42.4**	**41.3**	**41.2**	**69.1**	**45.9**	**53.0**	**51.8**	**47.1**	**49.5**	**72.4**

Table 2: Performance on COCO-20^i [43] in mIoU and FB-IoU. The results of other methods are from [4, 35, 67, 71, 76].

Backbone network	Methods	mIoU 1-shot	5-shot
VGG16 [61]	OSLSM [58]	70.3	73.0
	GNet [52]	71.9	74.3
	FSS [31]	73.5	80.1
	DoG-LSTM [2]	80.8	83.4
	HSNet (ours)	**82.3**	**85.8**
ResNet50 [17]	HSNet (ours)	**85.5**	**87.8**
ResNet101 [17]	DAN [71]	85.2	88.1
	HSNet (ours)	**86.5**	**88.5**

Table 3: Mean IoU comparison on FSS-1000 [31]. Some results are from [2, 71].

Experiments

Kernel type	1-shot 5^0	5^1	5^2	5^3	mean	5-shot 5^0	5^1	5^2	5^3	mean	# learnable params	time (*ms*)	memory footprint (*GB*)	FLOPs (*G*)
Original 4D kernel [55]	64.5	71.4	62.3	61.7	64.9	70.8	**74.8**	**67.4**	67.5	70.1	11.3M	512.17	4.12	702.35
Separable 4D kernel [77]	66.1	72.0	**63.2**	62.6	65.9	71.2	74.1	67.2	68.1	70.2	4.4M	28.48	1.50	28.40
Center-pivot 4D kernel (ours)	**67.3**	**72.3**	62.0	**63.1**	**66.2**	**71.8**	74.4	67.0	**68.3**	**70.4**	**2.6M**	**25.51**	**1.39**	**20.56**

Table 4: Comparison between three different 4D conv kernels in model size, per-episode inference time, memory consumption and FLOPs. For fair comparison, the inference times of all the models are measured on a machine with an Intel i7-7820X and an NVIDIA Titan-XP.