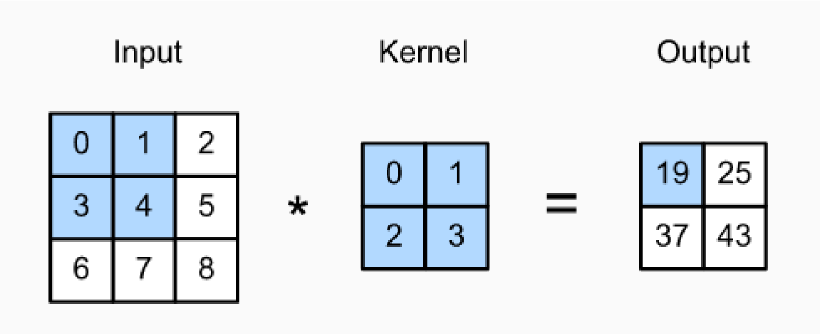
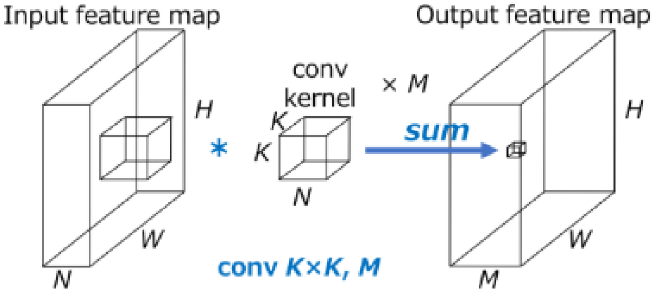
Pixel-Adaptive Convolutional Neural Networks (CVPR 2019)

Hang Su[1], Varun Jampani[2], Deqing Sun[2], Orazio Gallo[2], Erik Learned-Miller[1], and Jan Kautz[2]

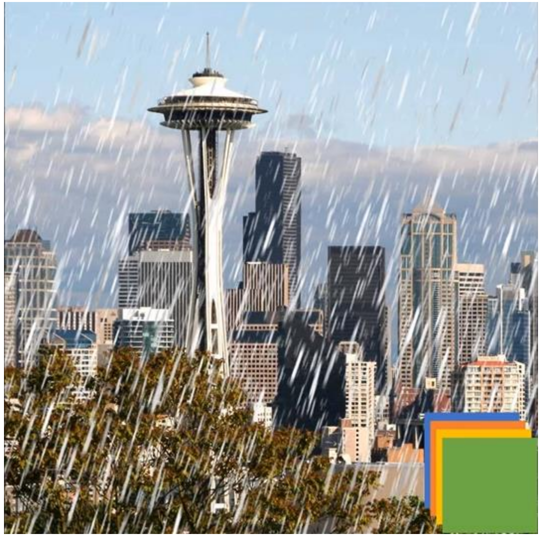
[1]UMass Amherst [2]NVIDIA

standard convolution



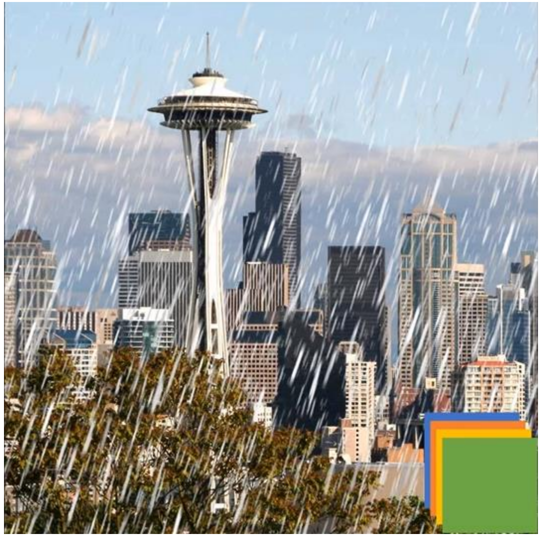


Convolution is spatially-shared



https://www.youtube.com/watch?v=gsQZbHuR64o

Convolution is content-agnostic



Dynamic Filter Networks (NIPS 2016)

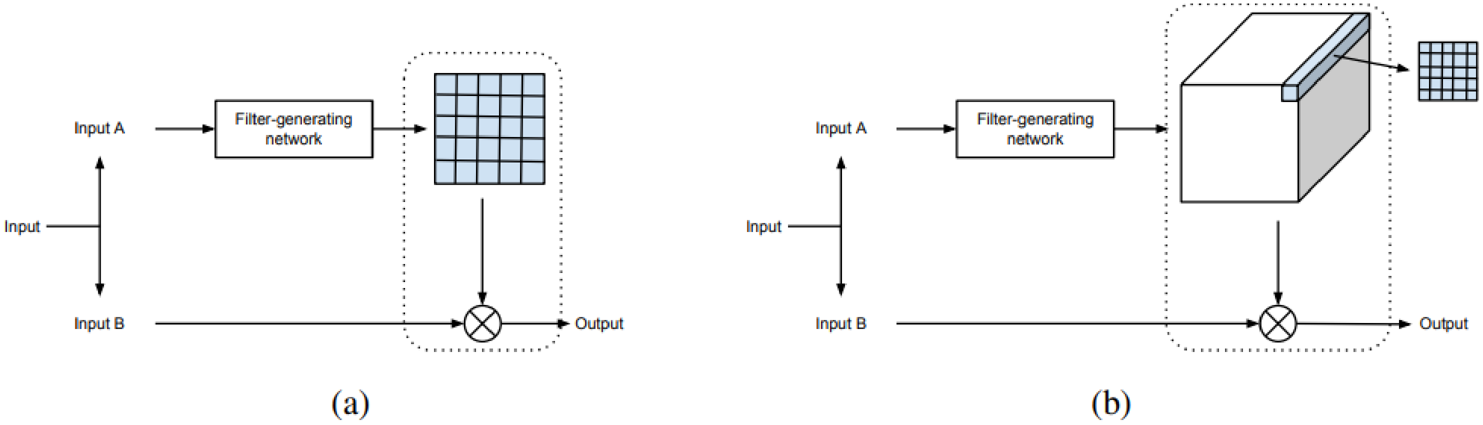


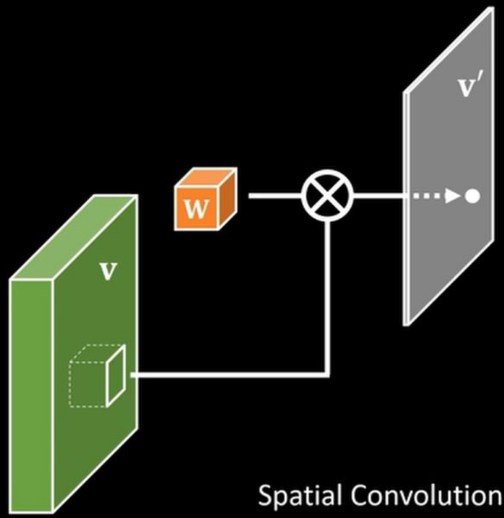
Figure 2: *Left:* Dynamic convolution: the filter-generating network produces a single filter that is applied convolutionally on I_B. *Right:* Dynamic local filtering: each location is filtered with a location-specific dynamically generated filter.

$\mathbf{v}$ input features

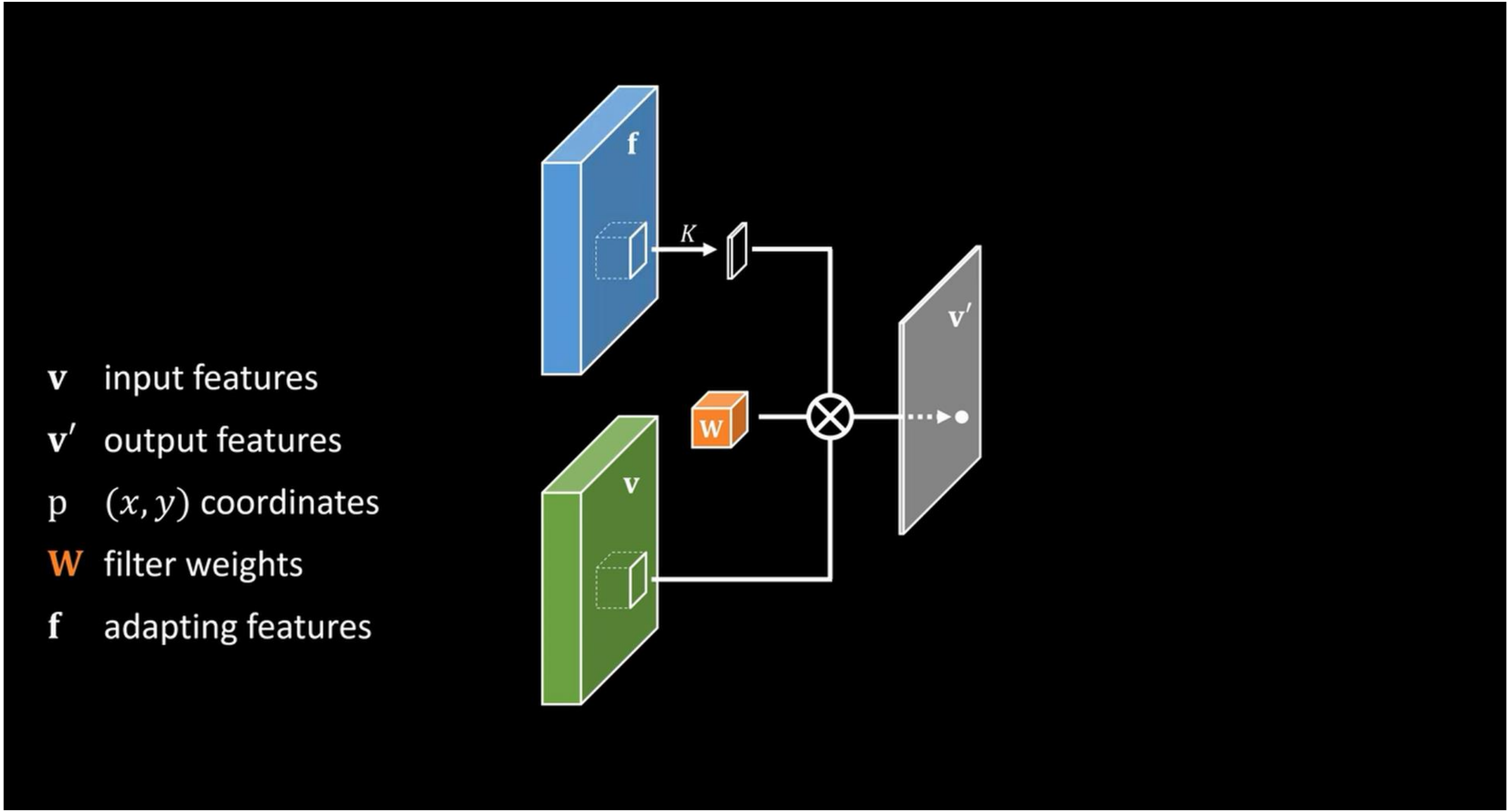
$\mathbf{v}'$ output features

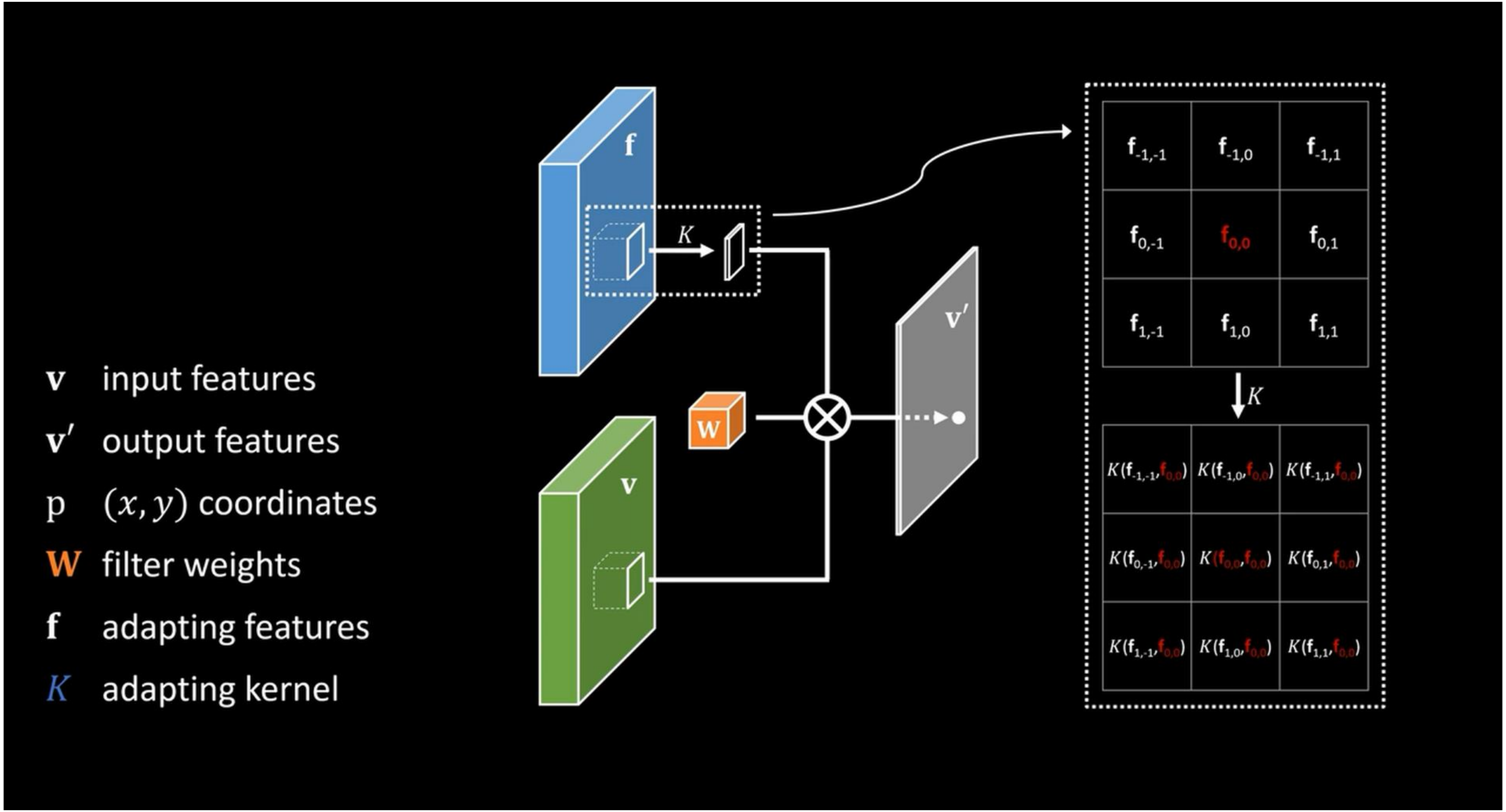
p (x, y) coordinates

$\mathbf{W}$ filter weights



Spatial Convolution $\mathbf{v}_i' = \sum_{j \in \Omega(i)} \mathbf{W}[\mathrm{p}_i - \mathrm{p}_j]\mathbf{v}_j + \mathbf{b}$


v input features
v′ output features
p (x, y) coordinates
W filter weights
f adapting features
f
K
W
v
v′

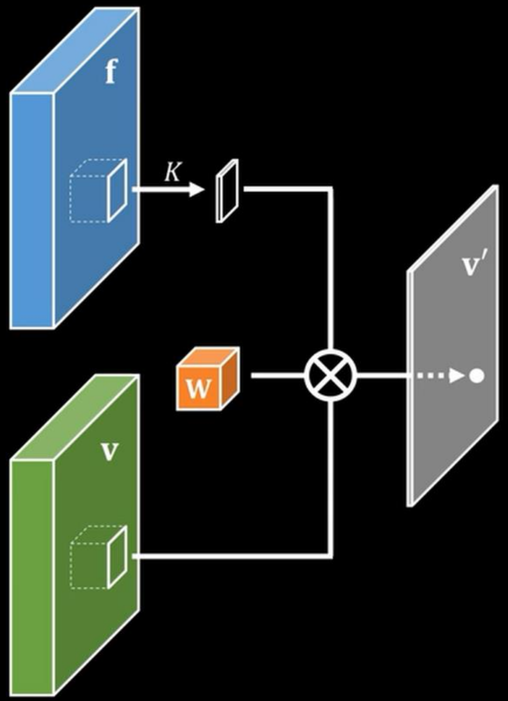

v input features
v′ output features
p (x, y) coordinates
W filter weights
f adapting features
K adapting kernel
f
K
v
v′
W
f-1,-1 f-1,0 f-1,1
f0,-1 f0,0 f0,1
f1,-1 f1,0 f1,1
K
K(f-1,-1,f0,0) K(f-1,0,f0,0) K(f-1,1,f0,0)
K(f0,-1,f0,0) K(f0,0,f0,0) K(f0,1,f0,0)
K(f1,-1,f0,0) K(f1,0,f0,0) K(f1,1,f0,0)

Pixel Adaptive Convolution (PAC)

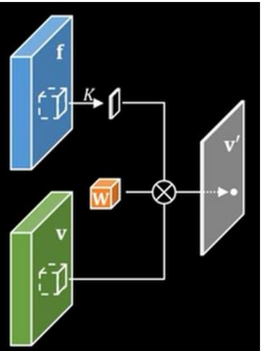
$$\mathbf{v}'_i = \sum_{j \in \Omega(i)} K(\mathbf{f}_i, \mathbf{f}_j)\mathbf{W}[\mathrm{p}_i - \mathrm{p}_j]\mathbf{v}_j + \mathbf{b}$$

Spatial Convolution

$$\mathbf{v}'_i = \sum_{j \in \Omega(i)} \mathbf{W}[\mathrm{p}_i - \mathrm{p}_j]\mathbf{v}_j + \mathbf{b}$$



$$\mathbf{v}'_i = \sum_{j \in \Omega(i)} K(\mathbf{f}_i, \mathbf{f}_j) \mathbf{W}[\mathrm{p}_i - \mathrm{p}_j] \mathbf{v}_j + \mathbf{b}$$

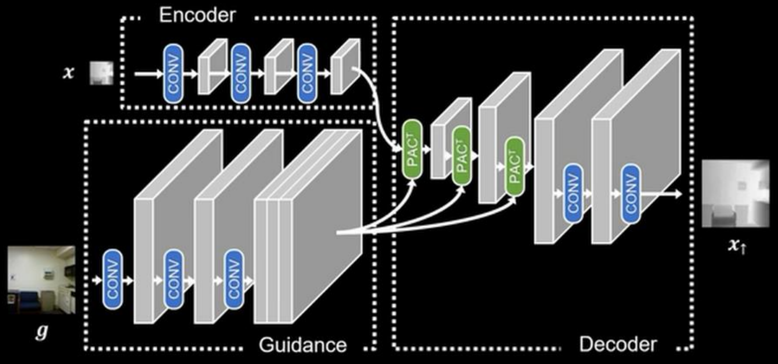


- Spatial convolution $K(\mathbf{f}_i, \mathbf{f}_j) = 1$
- Bilateral filtering $\mathbf{f} = (r, g, b), K(\mathbf{f}_i, \mathbf{f}_j) = \exp(-\frac{1}{2\alpha_1} \|\mathbf{f}_i - \mathbf{f}_j\|^2)$

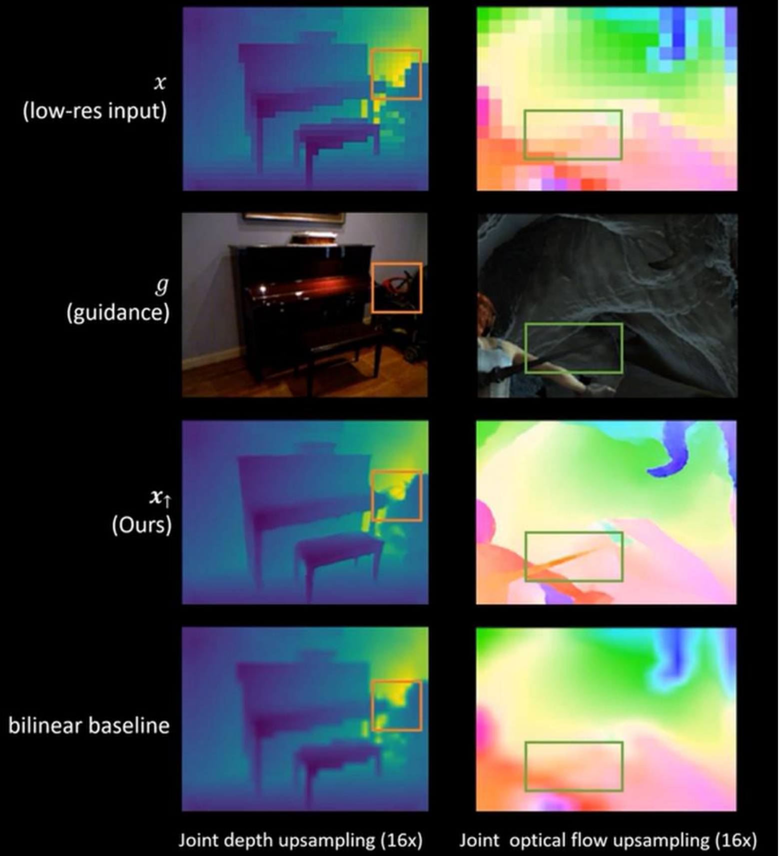
  $\mathbf{W}[\mathrm{p}_i - \mathrm{p}_j] = \exp(-\frac{1}{2\alpha_2} \|\mathrm{p}_i - \mathrm{p}_j\|^2)$
- Average pooling $K(\mathbf{f}_i, \mathbf{f}_j) = 1, \mathbf{W}[\mathrm{p}_i - \mathrm{p}_j] = \frac{1}{F^2}$
- Detail-preserving pooling [1] $K(\mathbf{f}_i, \mathbf{f}_j) = \alpha + (\|\mathbf{f}_i - \mathbf{f}_j\|^2 + \epsilon^2)^\lambda$

PAC generalizes many existing filtering techniques

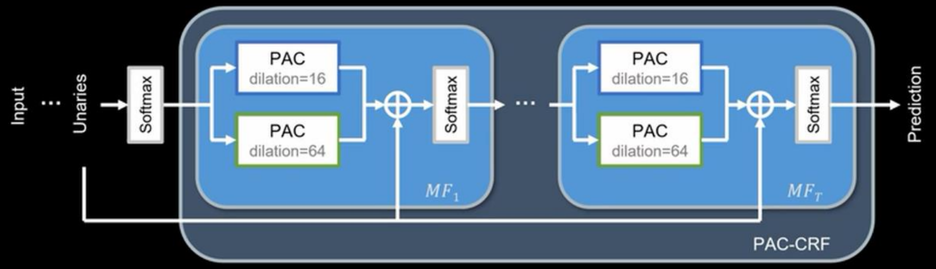
[1] F. Saeedan, et al. Detail Preserving Pooling in Deep Networks. CVPR '18.

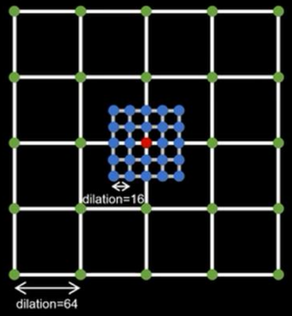


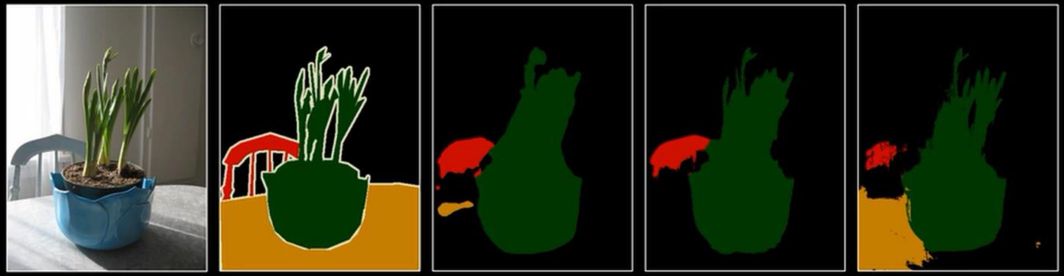
Joint upsampling network with PAC



PAC for efficient CRF inference



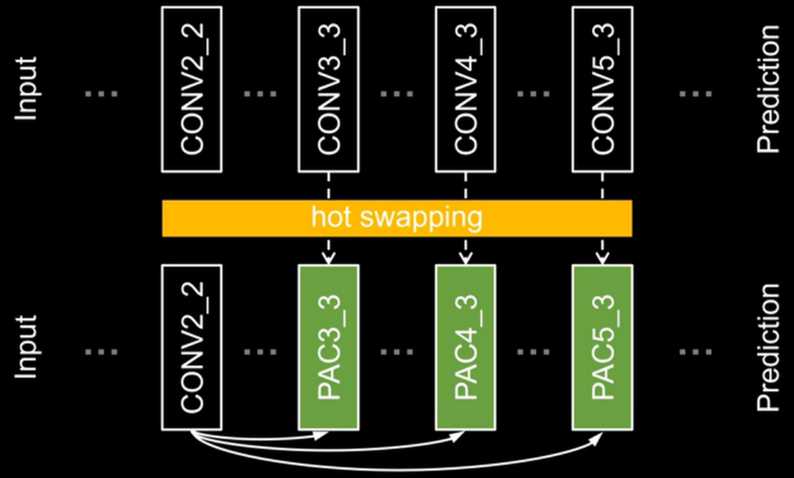




Input GT Unary Full-CRF [1] Ours

[1] Philipp Krähenbühl and Vladlen Koltun. Efficient Inference in Fully Connected CRFs with Gaussian Edge Potentials. NIPS '11.

Layer "hot-swapping" with PAC

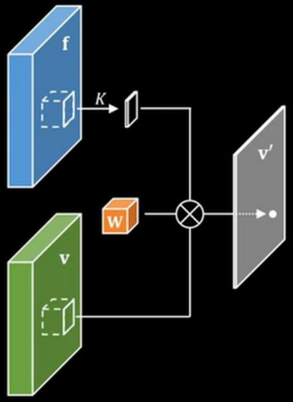


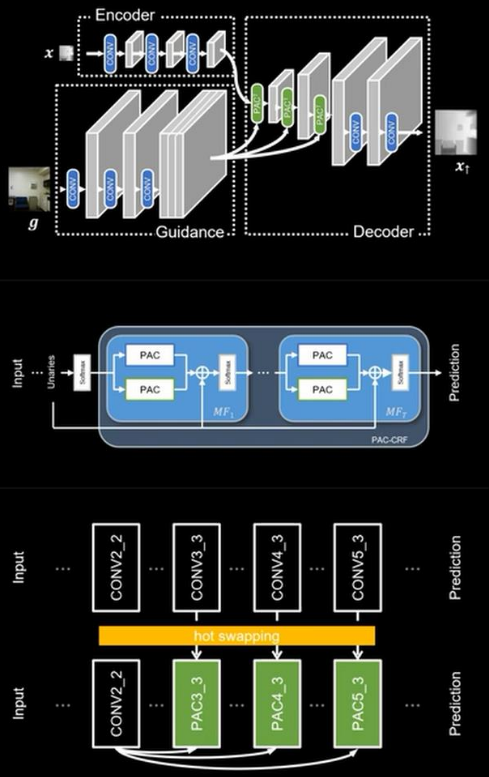
Method	CRF	mIoU	Runtime
FCN		67.20	39ms
FCN	✓	69.82	117ms
PAC-FCN		69.18	41ms
PAC-FCN	✓	71.34	118ms

*evaluated on Pascal VOC 2012 test set

Summary

- Pixel Adaptive Convolution
  Sec.3:
  - Content-adaptive
  - Generalizes several existing filtering techniques
- Three use cases:
  - Joint upsampling networks
    Sec. 4
  - Efficient CRF inference
    Sec. 5
  - Network layer hot-swapping
    Sec. 6





```
in_ch, g_ch = 16, 8                          # channel sizes of input and guidance
stride, f, b, h, w = 5, 2, 64, 64            # stride, filter size, batch size, input height and width
input = torch.rand(b, in_ch, h, w)
guide = torch.rand(b, g_ch, h, w)            # guidance feature

pool = nn.AvgPool2d(f, stride)
out_pool = pool(input)                       # standard spatial convolution

pacpool = PacPool2d(f, stride)
out_pac = pacpool(input, guide)              # PAC
out_pac = pacpool(input, None, guide_k)      # alternative interface
                                             # guide_k is pre-computed 'K'
                                             # of shape [b, g_ch, f, f, h, w]. packernel2d can be
                                             # used for its creation.
```