Day 12

Speaker: Dr. Varun Jampani, Google USA

Title: Content Adaptive Convolutional Neural Network.

Content-Adaptive Convolution Neural Networks

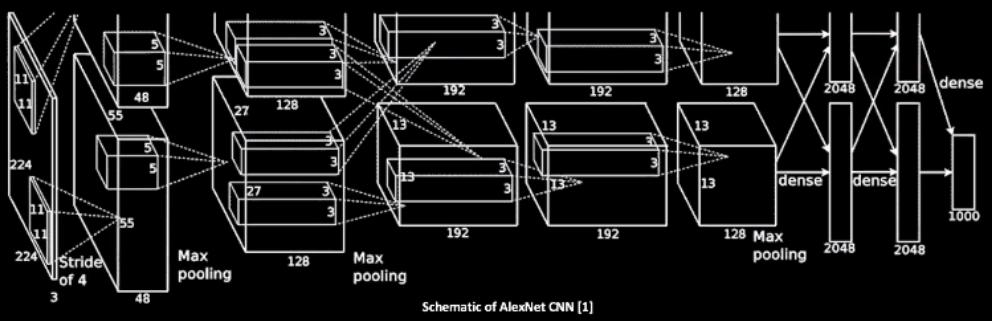
Varun Jampani

Google Research

Summer School on AI, IIIT-Hyderabad, August 2021

Spatial convolution

- *Simplest*, *fastest* and *most used* way of propagating information
- Basic building block of most CNNs



Schematic of AlexNet CNN [1]

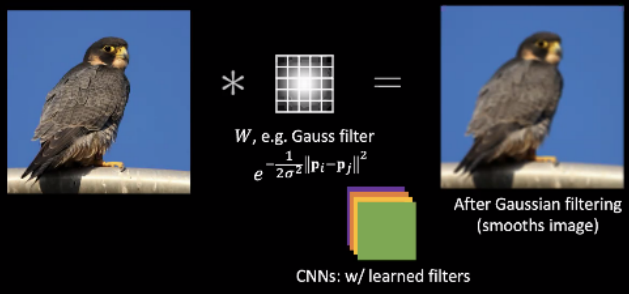
1. Krizhevsky, A., Sutskever, I., & Hinton, G. E. Imagenet classification with deep convolutional neural networks. In *NIPS*, 2012.

Spatial convolution

- Weighted sum over local neighborhoods

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

output feature · filter weights · position offset · input feature

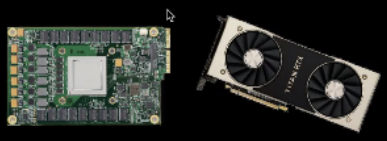


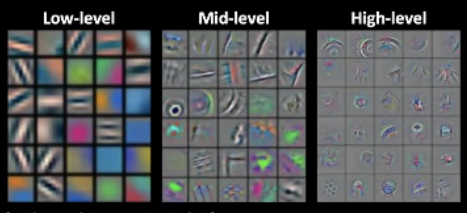
Spatial convolution

- Weighted sum over local neighborhoods

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

- Simple formulation → efficient parallelization
- Spatial sharing → capturing invariance
- Hierarchical features learning

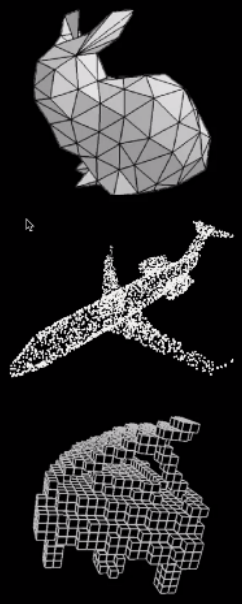




[Zeiler and Fergus, ECCV '13]

Limitations

- **Requires structured inputs**
  - Spatial convolution operates on *regularly structured grids*
  - Data in 3D or higher-dimensional spaces
    - Lack of grid structure
    - Curse of dimensionality



Limitations

- **Requires structured inputs**
- **Content-agnostic filters**
  - Loss gradients are shared across pixel locations
  - Once trained, same filter banks are used across different images

same filters used for all parts in an image.

Bilateral filter

- Generalization of spatial filtering to arbitrary features [1, 2]

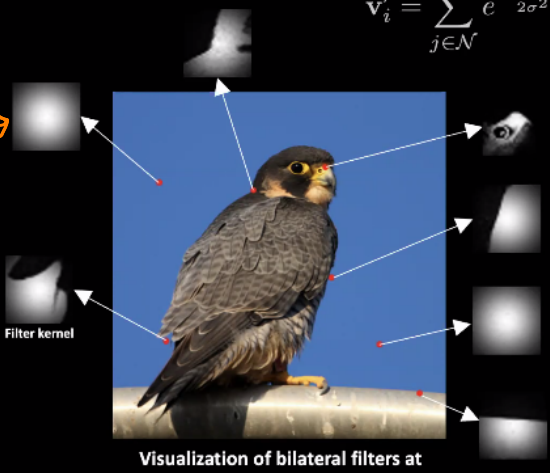
$$\mathbf{v}_i' = \sum_{j \in \mathcal{N}} W(\mathbf{p}_i - \mathbf{p}_j)\mathbf{v}_j \longrightarrow \mathbf{v}_i' = \sum_{j \in \mathcal{N}} W(\mathbf{f}_i - \mathbf{f}_j)\mathbf{v}_j$$

- Features for $\mathbf{f}$ are position and color: $\mathbf{f} = (x, y, r, g, b)$
- Kernel function is Gaussian: $W(\cdot) = e^{-\frac{1}{2}\|\cdot\|^2}$

1. Aurich, V., & Weule, J. Non-linear Gaussian filters performing edge preserving diffusion. In *Mustererkennung*, 1995.
2. Tomasi, C., & Manduchi, R. Bilateral filtering for gray and color images. In *ICCV*, 1998.

Bilateral filter depends on the image content

$$\mathbf{v}_i' = \sum_{j \in \mathcal{N}} e^{-\frac{1}{2\sigma^2}\|\mathbf{f}_i - \mathbf{f}_j\|^2} \mathbf{v}_j$$



Visualization of bilateral filters at different input locations

After bilateral filtering (preserves edges)

Generate filters at each pixel (NOT same filter in whole image)

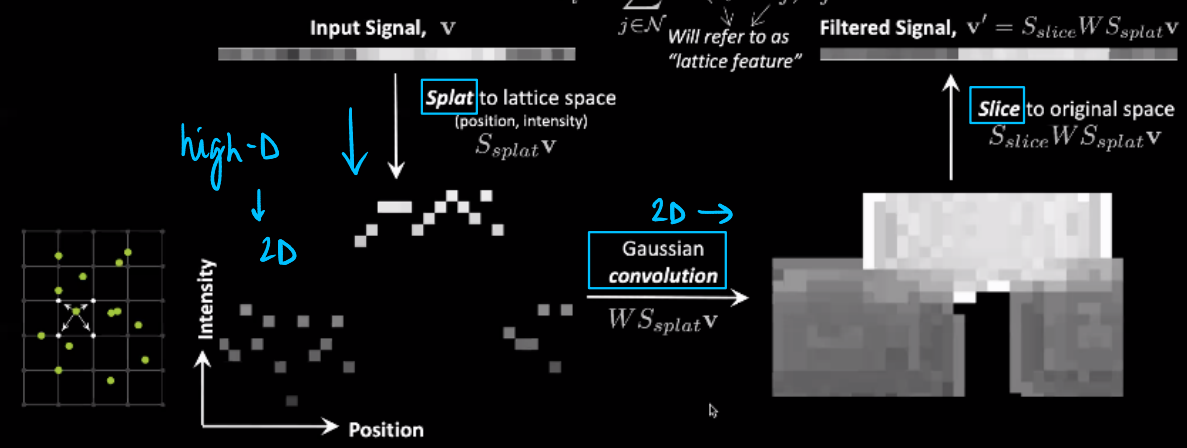
• Bilateral filter Advantage: → Preserves the edges

High-dimensional Gaussian filter

- Bilateral filter as convolution in *high-dimensional feature space* [1]

$$\mathbf{v}'_i = \sum_{j \in \mathcal{N}} W(\mathbf{f}_i - \mathbf{f}_j)\mathbf{v}_j$$

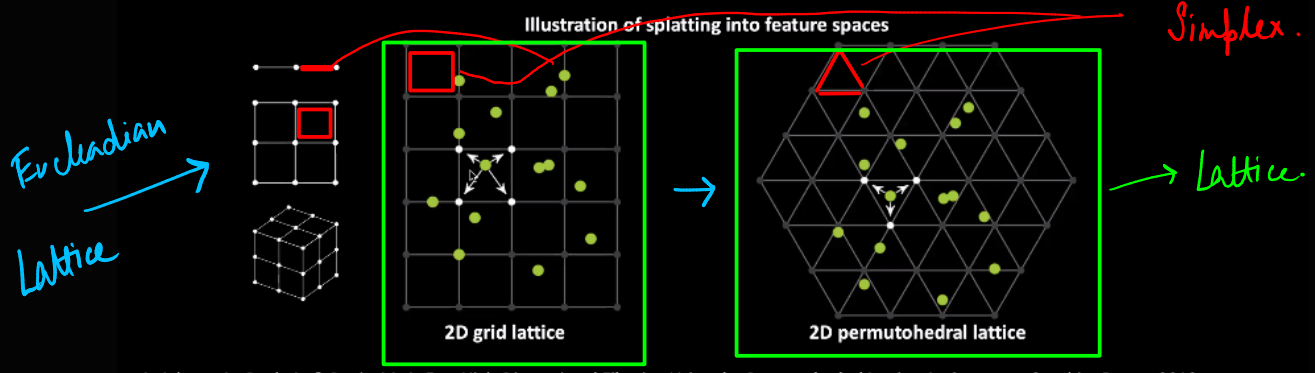
Will refer to as "lattice feature"



1. Paris, S., & Durand, F. A fast approximation of the bilateral filter using a signal processing approach. In *ECCV*, 2006.

N-d grid vs. Permutohedral lattice

- Grid cell corner points increase exponentially with dimensions (2^d)
- Permutohedral lattice[1] cells have fewer corner points (linear in dimensions - d+1)

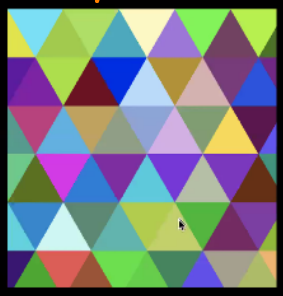


1. Adams, A., Baek, J., & Davis, M. A. Fast High-Dimensional Filtering Using the Permutohedral Lattice. In *Computer Graphics Forum*, 2010.

High-dimensional lattice visualization

Same Colour → Same Simplex

Sample Image



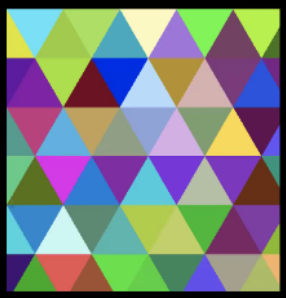
Position features
$\mathbf{f} = (x, y)$

↑ Permutohedral

Pixels falling in the same simplex are shown with the same color

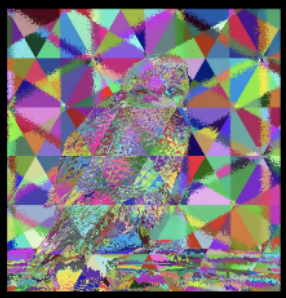
High-dimensional lattice visualization

Sample Image



Position features
$\mathbf{f} = (x, y)$

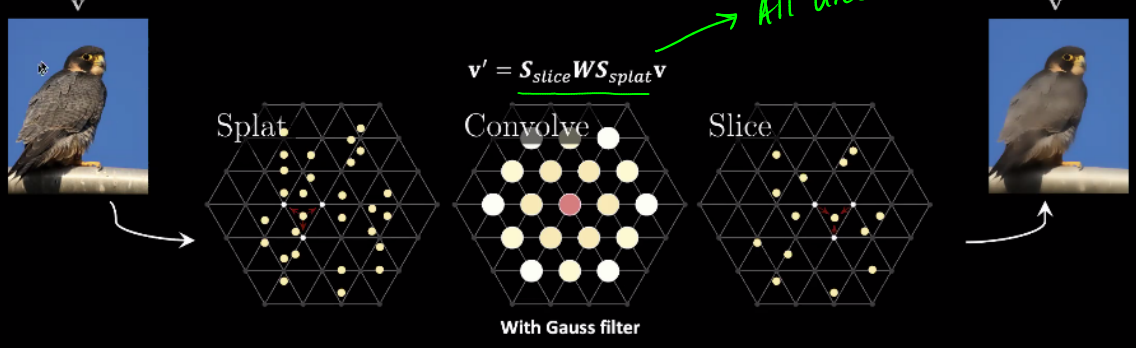
Color features
$\mathbf{f} = (r, g, b)$



Position and Color
$\mathbf{f} = (x, y, r, g, b)$

Pixels falling in the same simplex are shown with the same color

Bilateral filtering with high-dimensional filtering



Learning the filters [CVPR'16, ICLRw'15]

- Parameterize the filter instead of using Gaussian

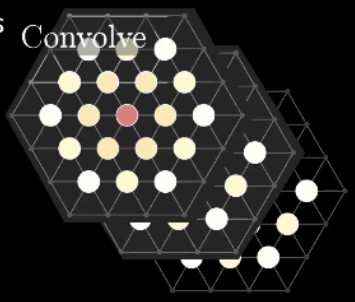
$$\mathbf{v}' = S_{slice} W S_{splat} \mathbf{v}$$

Learn filter weights via back-propagation

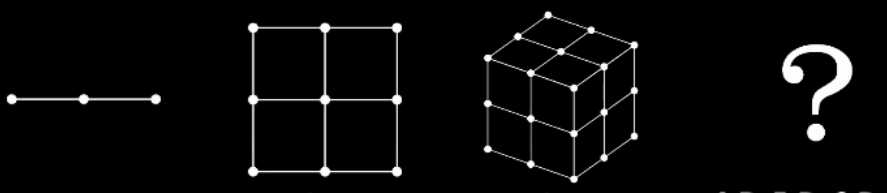
[CVPR'16] Jampani, V., Kiefel, M., & Gehler, P. V. Learning Sparse High Dimensional Filters: Image Filtering, Dense CRFs and Bilateral Neural Networks. *In CVPR,* 2016.
[ICLRw'15] Kiefel, M., Jampani, V., & Gehler, P. V. Permutohedral Lattice CNNs. *In ICLR Workshop,* 2015.

Learning the filters [CVPR'16, ICLRw'15]

- Learning bilateral filters allows the use of stacked filters
- Randomly initialize and learn end-to-end
- **BCL**: Bilateral Convolution Layer



[CVPR'16] Jampani, V., Kiefel, M., & Gehler, P. V. Learning Sparse High Dimensional Filters: Image Filtering, Dense CRFs and Bilateral Neural Networks. *In CVPR,* 2016.
[ICLRw'15] Kiefel, M., Jampani, V., & Gehler, P. V. Permutohedral Lattice CNNs. *In ICLR Workshop,* 2015.

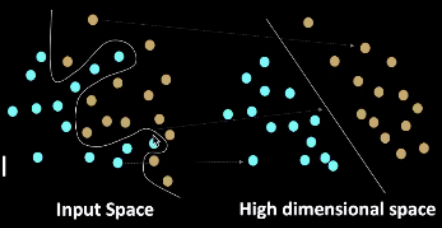


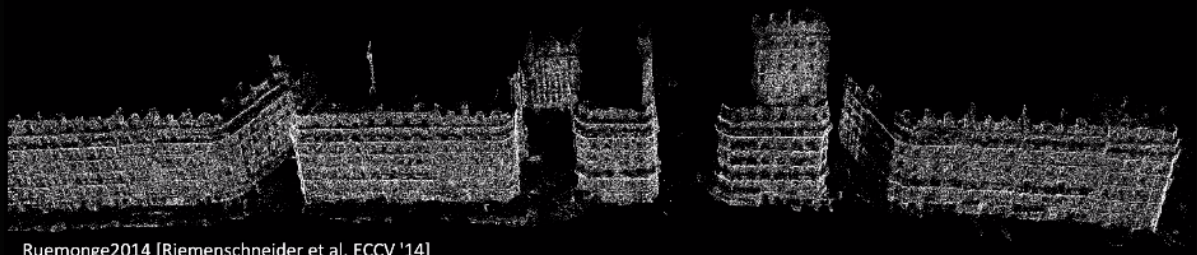
computationally expensive in normal convolutions

Equip neural networks with richer class of
Sparse High-Dimensional Filters

Why sparse and high-dimensional networks?

- Project data to higher dimensions
  - E.g. Bilateral filter
- Data can be inherently sparse and high-dimensional
  - E.g. 3D data, BRDF material properties, video





Ruemonge2014 [Riemenschneider et al. ECCV '14]

Separate lattice and point features

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

"where" lattice feature

point feature "what"

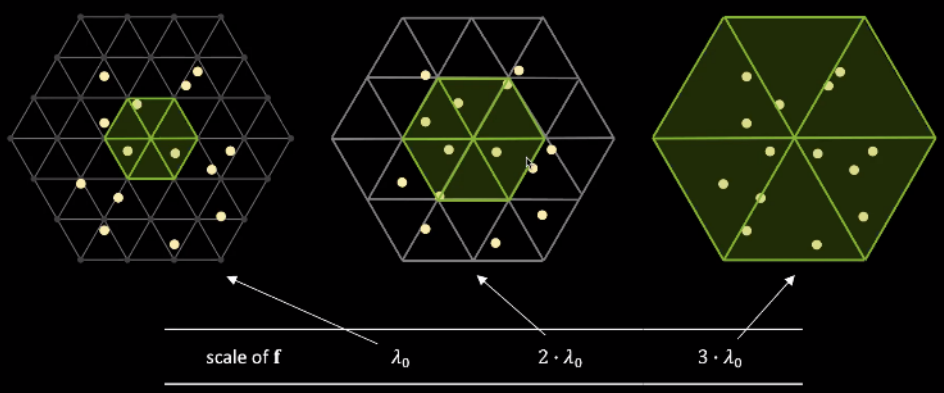
spatial convolution

sparse high-dim. filtering

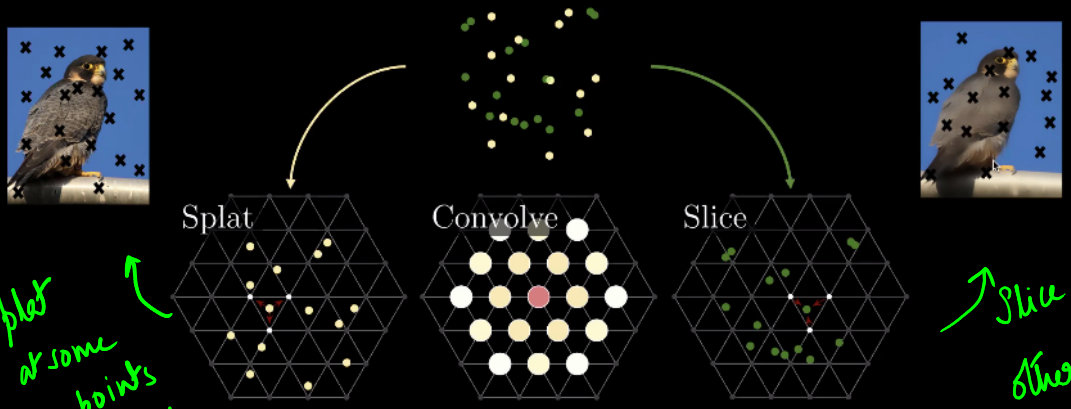
Image	point feature $\mathbf{v}$	(r,g,b)	(r,g,b)	$\mathbf{v}(\dots)$	$\mathbf{v}(\dots)$
	lattice feature $\mathbf{f}$	(x,y)	(x,y,r,g,b)	(x,y)	$(x,y,depth)$
3D Point cloud	point feature $\mathbf{v}$	1	(x,y,z)	(r,g,b)	$\mathbf{v}(\dots)$
	lattice feature $\mathbf{f}$	(x,y,z)	(x,y,z)	(x,y,z,n_x,n_y,n_z)	(x,y,z)

Controlling receptive fields

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



Input and output can be at different points

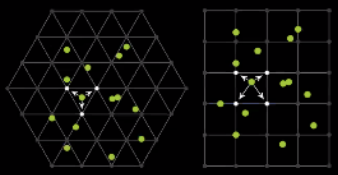


≈ Application
~ Upsampling.

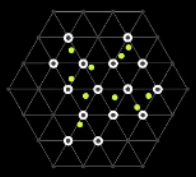
Splat at some points only

Slice into other

Efficient computation



- Permutohedral lattice[1]
- Hash table



[1] Adams *et al.* Fast high-dimensional filtering using the permutohedral lattice. Computer Graphics Forum '10

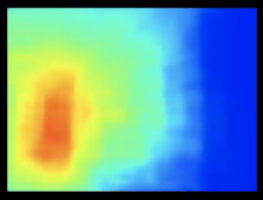
Applications in 2D, video and 3D vision

- BCL & BNNs have wide range of applications
- In this talk:
  - Joint image filtering
  - Video information propagation
  - 3D point cloud segmentation

Joint image filtering

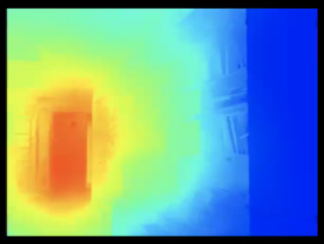
Task: Enhance an input with the help of a given guidance image

Example: Depth upsampling



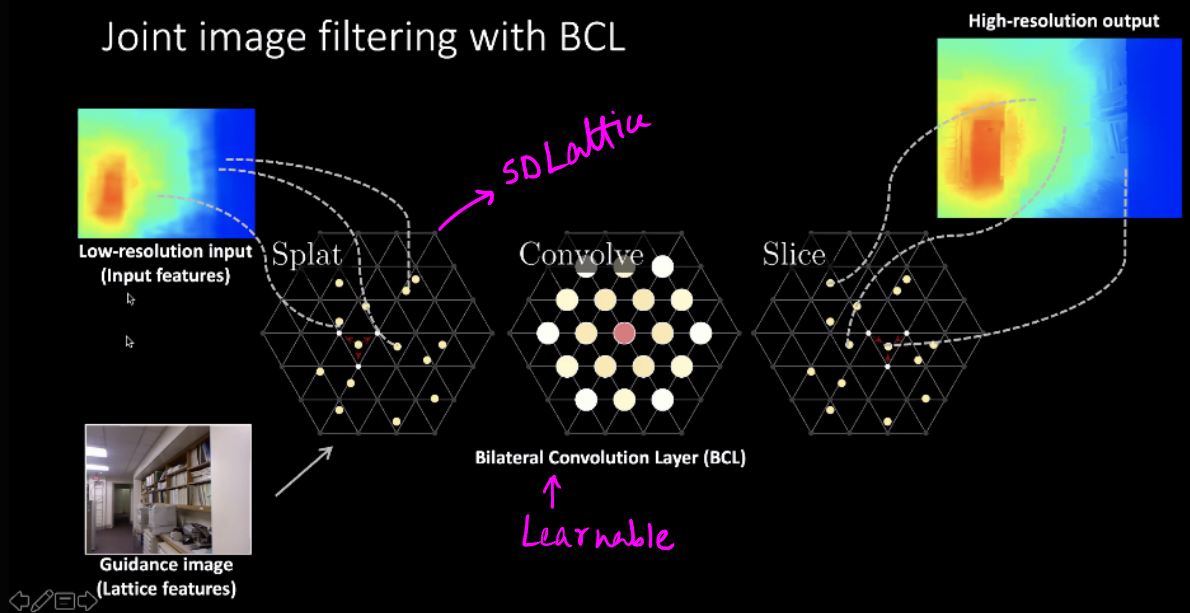
Low-resolution input

Guidance RGB image



After joint image filtering

Joint image filtering with BCL



Depth upsampling results

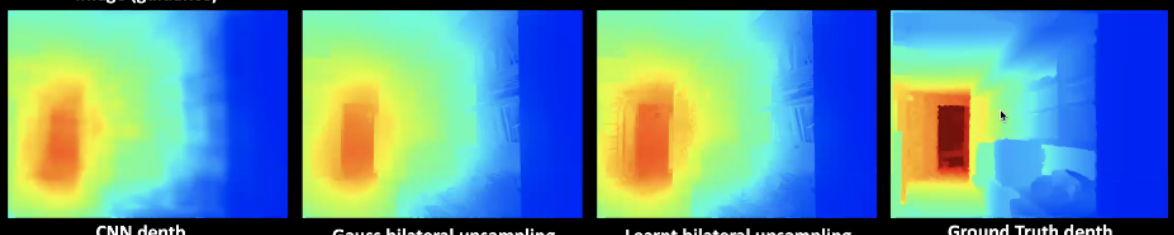
CNN depth estimation results[1] are often low resolution

Image (guidance)

RMSE results on NYU depth dataset [2] (8x upsampling)

	Bicubic	Gauss Bilateral	Learnt Bilateral
Test set	0.753	0.752	**0.748**

→ Loss



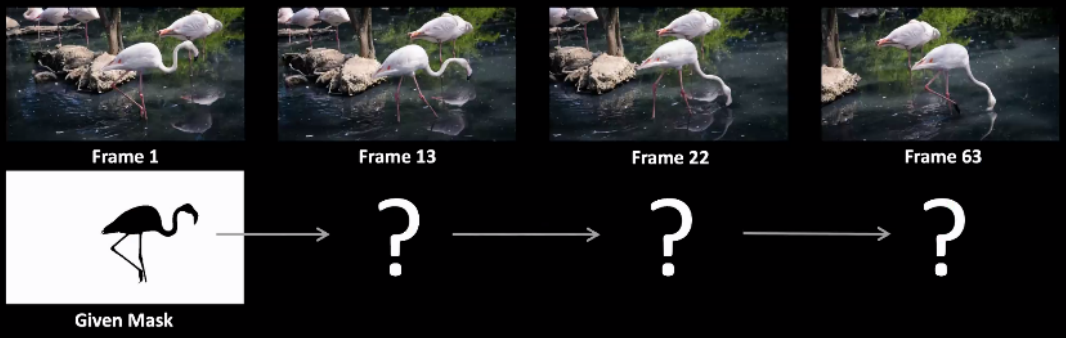
CNN depth (bicubic interpolated) | Gauss bilateral upsampling | Learnt bilateral upsampling | Ground Truth depth

1. Eigen, D., Puhrsch, C., & Fergus, R. Depth map prediction from a single image using a multi-scale deep network. NIPS '14
2. Silberman, N., Hoiem, D., Kohli, P., & Fergus, R. (2012). Indoor segmentation and support inference from RGBD images. ECCV '12

Video propagation

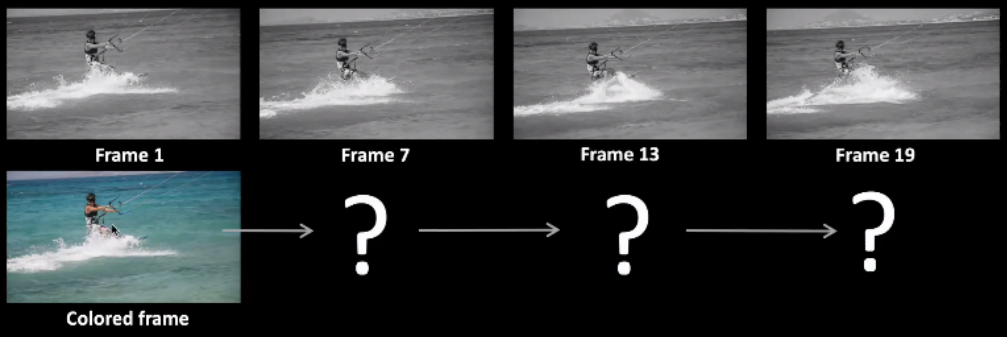
How can we propagate information across video frames?

Example: Video object segmentation



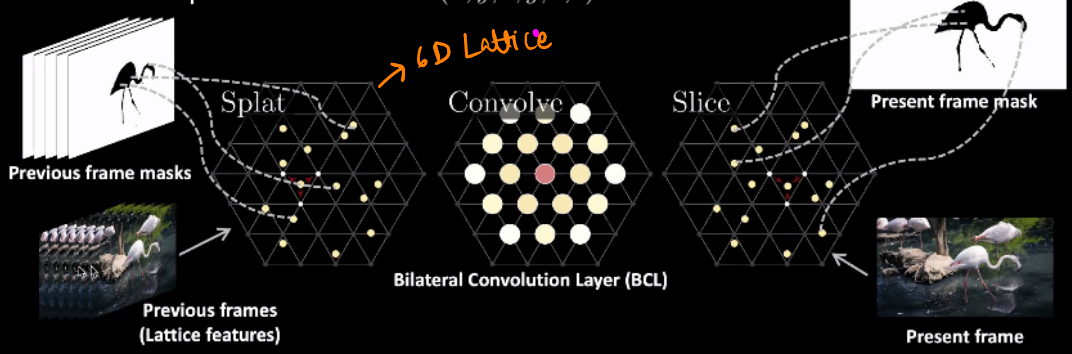
Video propagation

Another example: Color propagation



Video propagation with BCL

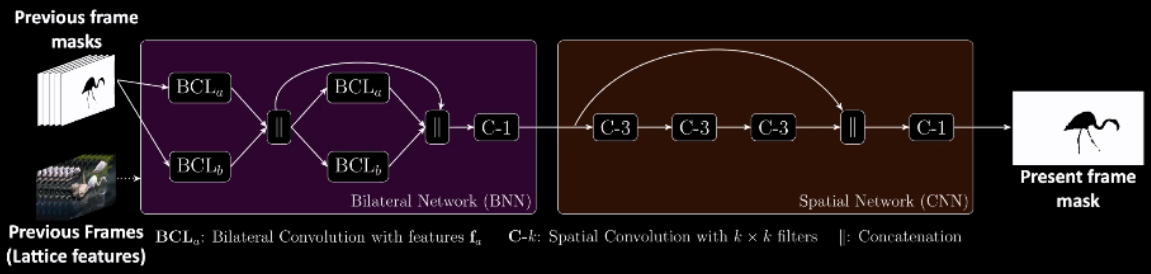
- Splat *previous* frame results, convolve and slice at the *present* frame.
- Lattice feature space with **time**: $\mathrm{f} = (x, y, r, g, b, t)^\top$ time



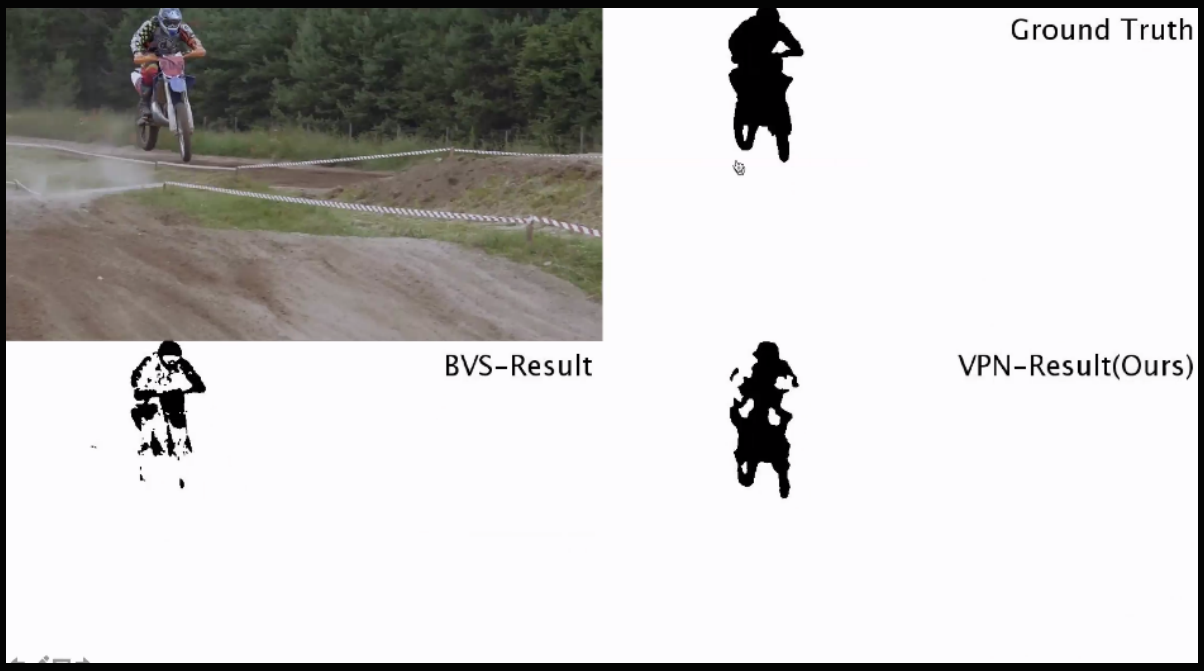
Video Propagation Network (VPN) [CVPR'17]

Two components in VPN

- Bilateral Network (BNN): For propagation from previous frames to present one
- Spatial Network (CNN): For refining features and prediction for the present frame



[CVPR'17] Jampani, V., Gadde, R. and Gehler, P.V., Video Propagation Networks. In *CVPR*, 2017.



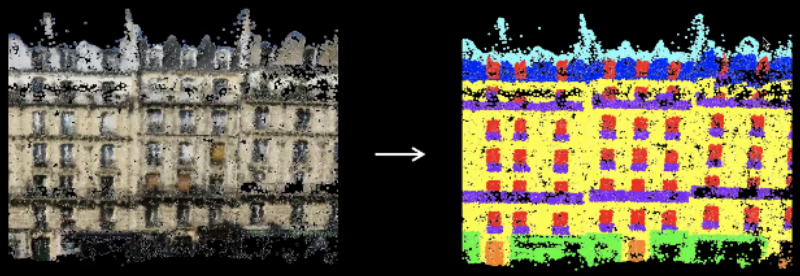
Colour Propagation.

3D point cloud segmentation

Goal: End-to-end trainable networks for analyzing point cloud data

Challenges:

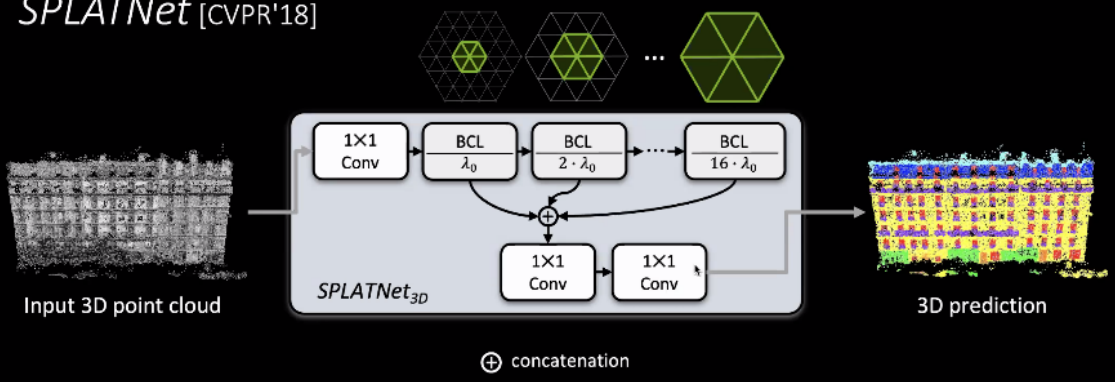
- Point clouds are *unordered* and *sparse*
- No readily defined *connections* between points



Input point cloud

Semantic Segmentation

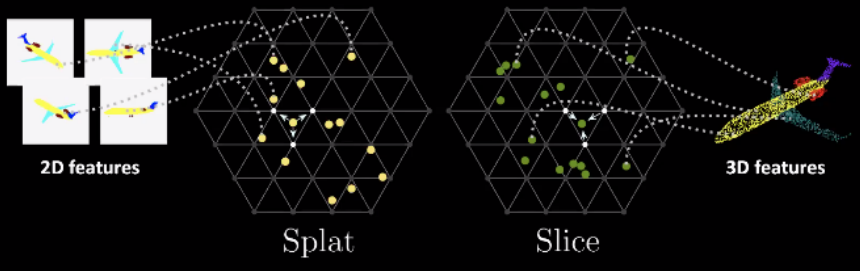
SPLATNet [CVPR'18]



[CVPR'18] Su, H., Jampani, V., Sun, D., Maji, S., Kalogerakis, E., Yang, M.H. and Kautz, J., SPLATNet: Sparse Lattice Networks for Point Cloud Processing. In *CVPR*, 2018.

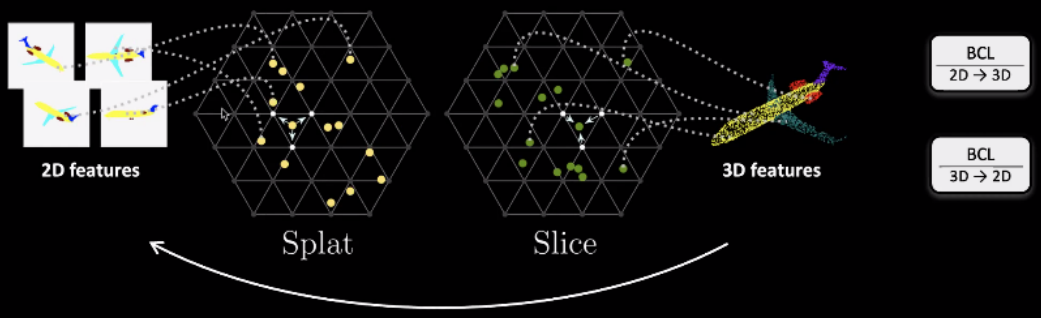
Smooth projection between 2D and 3D

- Each 2D pixel p: $\mathbf{v}(p)$, (x, y, z)
  → Input feature: $\mathbf{v}(p)$
  → Lattice feature: (x, y, z)

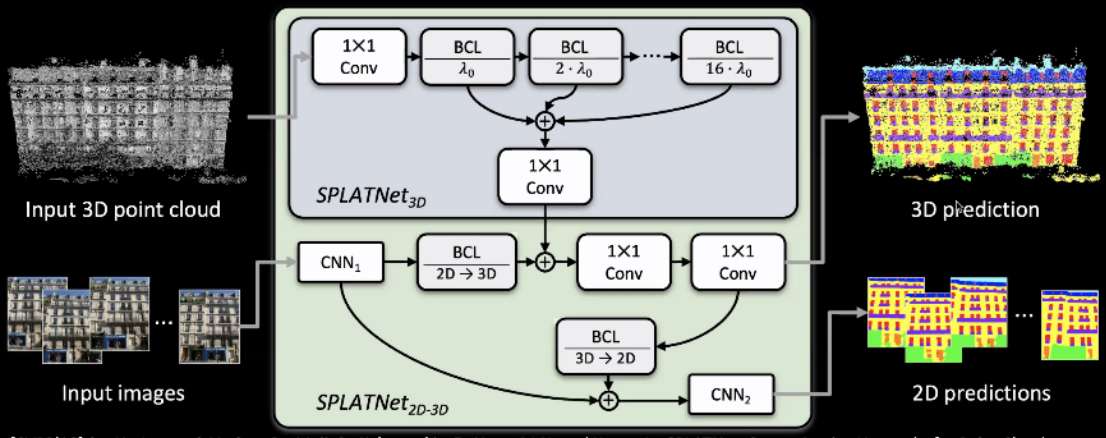


Smooth projection between 2D and 3D

- Each 2D pixel p: $\mathbf{v}(p)$, (x, y, z)
  - → Input feature: $\mathbf{v}(p)$
  - → Lattice feature: (x, y, z)



SPLATNet$_{2D\text{-}3D}$ [CVPR'18]

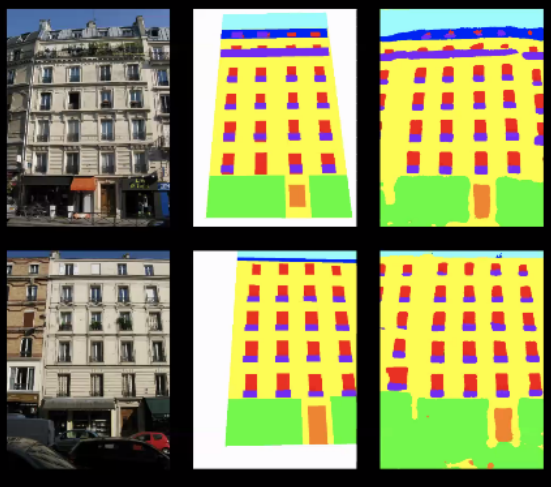


[CVPR'18] Su, H., Jampani, V., Sun, D., Maji, S., Kalogerakis, E., Yang, M.H. and Kautz, J., SPLATNet: Sparse Lattice Networks for Point Cloud Processing. In *CVPR*, 2018.

2D predictions

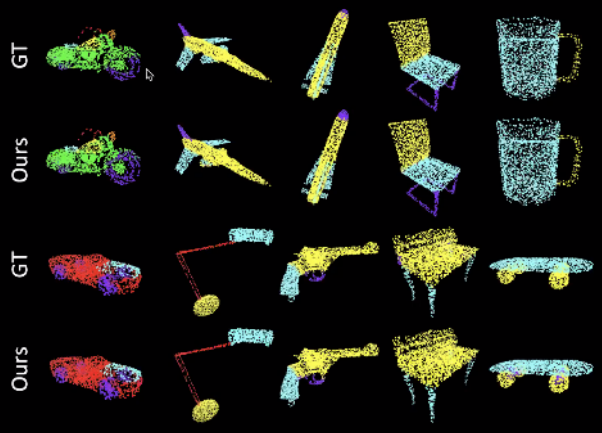
	IoU	runtime (min)
Autocontext$_{2D}$ [1]	60.5	117
Autocontext$_{2D\text{-}3D}$ [1]	62.7	146
2D CNN [2] only	69.3	0.84
SPLATNet$_{2D\text{-}3D}$	70.6	4.34

[1] Gadde *et al.* PAMI '17
[2] Chen *et al.* ICLR '15



Input image | Ground-truth | Our prediction

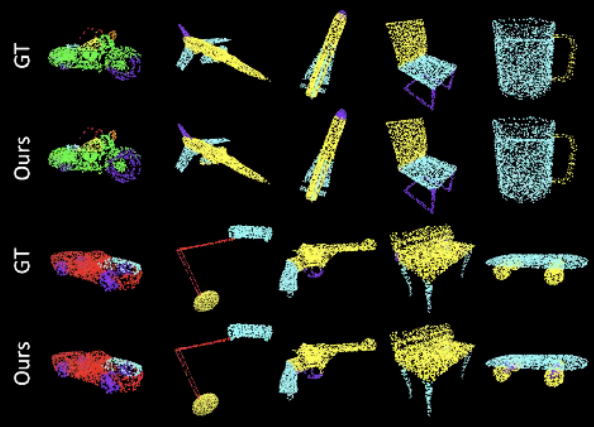
3D object part labeling (ShapeNet [1])



[1] Yi *et al.* SIGGGRAPH Asia '16

3D object part labeling (ShapeNet [1])

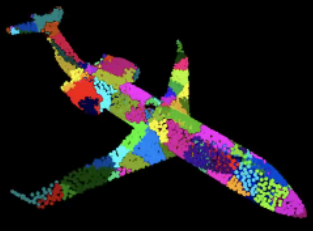
	class avg. IoU	instance avg. IoU
Yi *et al.* [1]	79.0	81.4
3DCNN [2]	74.9	79.4
Kd-network [3]	77.4	82.3
PointNet [2]	80.4	83.7
PointNet++ [4]	81.9	85.1
SyncSpecCNN [5]	82.0	84.7
SPLATNet_{3D}	82.0	84.6
$\text{SPLATNet}_{2D\text{-}3D}$	**83.7**	**85.4**



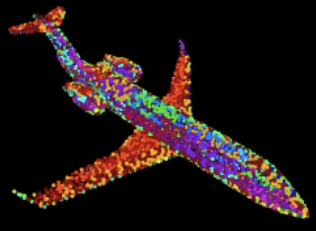
[1] Yi *et al.* SIGGGRAPH Asia '16
[2] Qi *et al.* CVPR '17
[3] Klokov and Lempitsky ICCV '17
[4] Qi *et al.* NIPS '17
[5] Yi *et al.* CVPR '17

Filtering in other lattice spaces

- Adding additional (x, y, z, n_x, n_y, n_z) filtering → +0.2 IoU



(x, y, z)



(n_x, n_y, n_z)

Remarks

$$\mathbf{v}'_i = \sum_{j \in \mathcal{N}} W(\mathbf{p}_i - \mathbf{p}_j)\mathbf{v}_j \quad \longrightarrow \quad \mathbf{v}'_i = \sum_{j \in \mathcal{N}} W(\mathbf{f}_i - \mathbf{f}_j)\mathbf{v}_j$$

- General sparse high-dimensional neural networks
- Advantages
  - Efficient computation with permutohedral lattice and hash tables
  - Flexible specification of lattice features and receptive fields
  - Input and output can be at different points
  - Wide range of applications in 2D, video and 3D vision
- Disadvantages
  - Slower than standard convolution when filtering 2D data
  - Manual specification of lattice features

Content adaptive convolution

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

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

High Dimensional Convolution

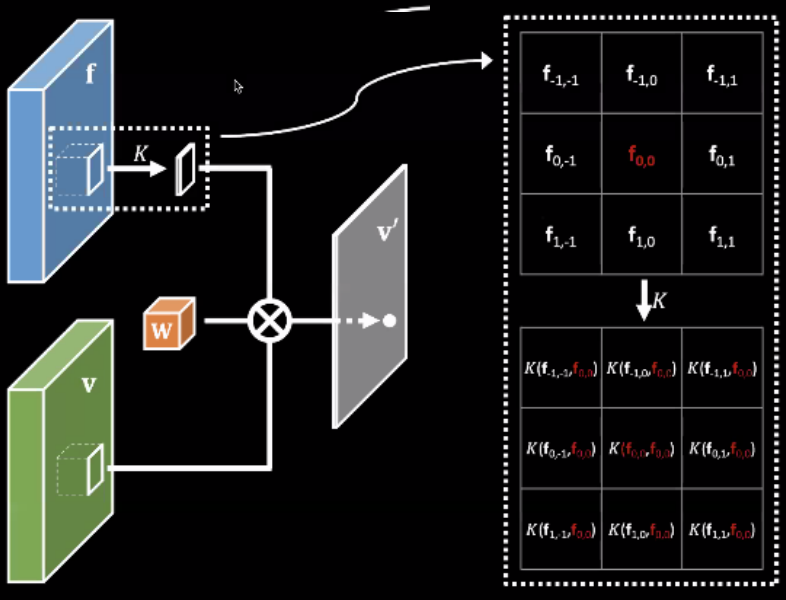
$$\mathbf{v}'_i = \sum_{j \in \Omega(i)} K(\mathbf{f}_i, \mathbf{f}_j)\mathbf{W}[\mathbf{p}_i - \mathbf{p}_j]\mathbf{v}_j$$

Pixel-Adaptive Convolution

Pixel-Adaptive Convolution (PAC)

For our experiments, we use

$$K(\mathbf{f}_i, \mathbf{f}_j) = e^{-\frac{1}{2}\|\mathbf{f}_i - \mathbf{f}_j\|^2}$$



$\mathbf{v}$ input features

$\mathbf{v}'$ output features

$\mathbf{p}$ (x, y) coordinates

$\mathbf{W}$ filter weights

$\mathbf{f}$ adapting features

K adapting kernel

[CVPR'19] Su, H., Jampani, V., Sun, D., Gallo, O., Learned-Miller, E. and Kautz, J., Pixel-Adaptive Convolutional Neural Networks. In *CVPR*, 2019.

Pixel-Adaptive Convolution (PAC)

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

2D filters at each pixel location

i^{th} pixel loc

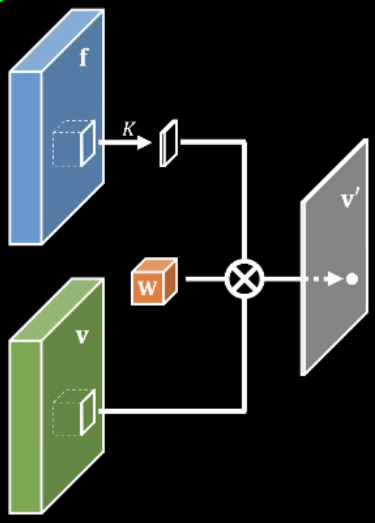
Pixel-adaptive convolution

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

Standard convolution

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

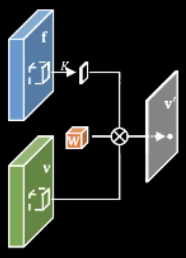
High-dimensional convolution



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

Effective filters are different across images and pixel locations

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



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

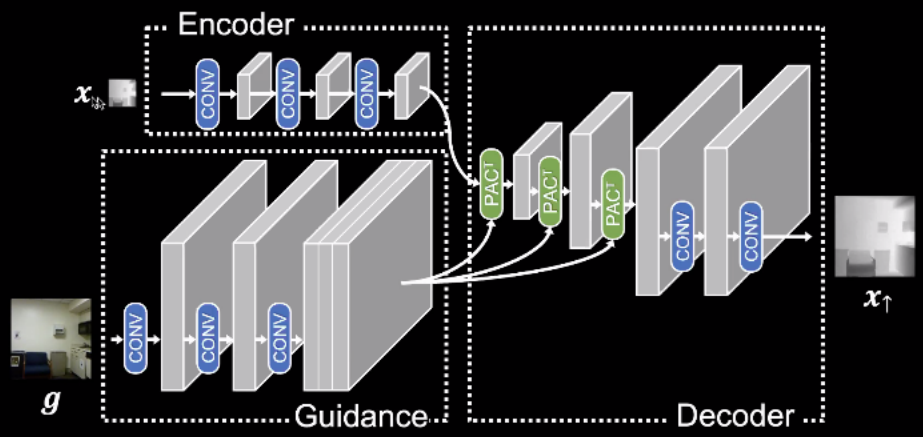
  $\mathbf{W}[p_i - p_j] = \exp(-\frac{1}{2\alpha_2}\|p_i - p_j\|^2)$
- Average pooling $K(\mathbf{f}_i, \mathbf{f}_j) = 1, \mathbf{W}[p_i - p_j] = \frac{1}{F^2}$
- Detail-preserving pooling [1] $K(\mathbf{f}_i, \mathbf{f}_j) = \alpha + (\|\mathbf{f}_i - \mathbf{f}_j\|^2 + \epsilon^2)^\lambda$

PAC generalizes many existing filtering techniques

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

Applications of PAC

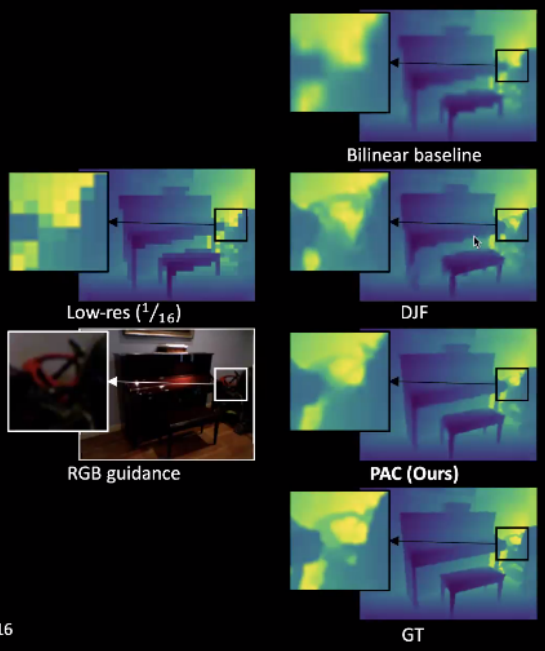
Joint upsampling network with PAC



Joint depth upsampling

Method	4×	8×	16×
Bicubic	8.16	14.22	22.32
MRF	7.84	13.98	22.20
GF [1]	7.32	13.62	22.03
JBU [2]	4.07	8.29	13.35
DMSG [3]	3.78	6.37	11.16
FBS [4]	4.29	8.94	14.59
DJF [5]	3.54	6.20	10.21
DJF+ [6]	3.38	5.86	10.11
PAC	**2.39**	**4.59**	**8.09**

RMSE on NYU-V2 depth maps

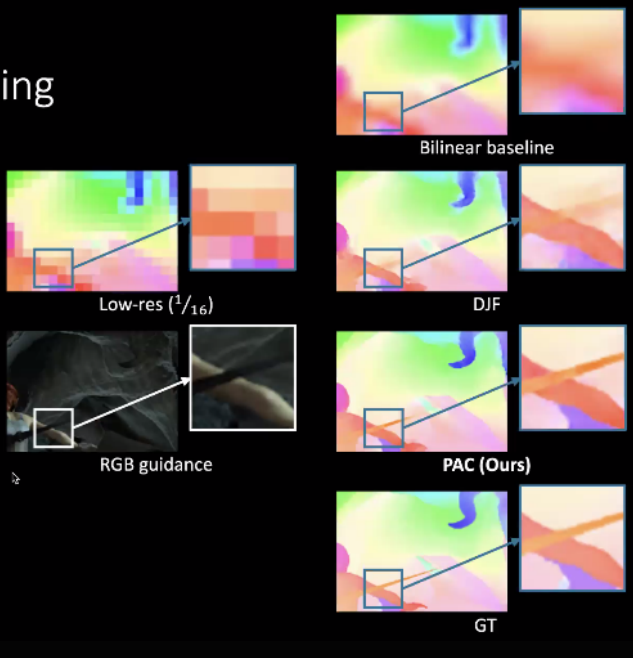


[1] He *et al.* PAMI '13
[2] Kopf *et al.* ToG '07
[3] Hui *et al.* ECCV '16
[4] Barron and Poole. ECCV '16
[5] Li *et al.* ECCV '16
[6] Li *et al.* PAMI '18

Joint optical flow upsampling

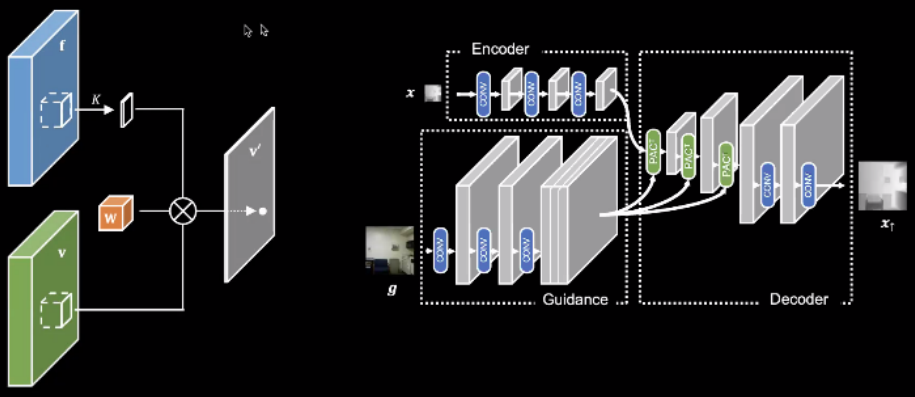
Method	4×	8×	16×
Bicubic	0.465	0.901	1.628
DJF	0.176	0.438	1.043
PAC	**0.105**	**0.256**	**0.592**

End-Point-Error (EPE) on Sintel



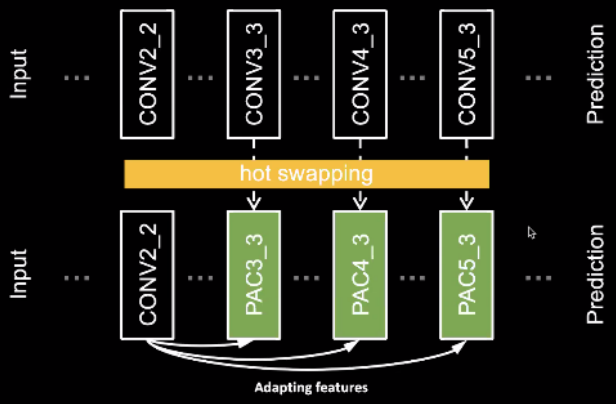
Layer “hot-swapping” with PAC

- Two ways to get adapting features: (1) dedicated branch



Layer “hot-swapping” with PAC

- Two ways to get adapting features: (1) dedicated branch (2) re-using features

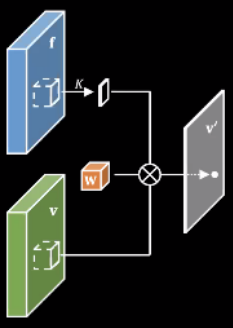


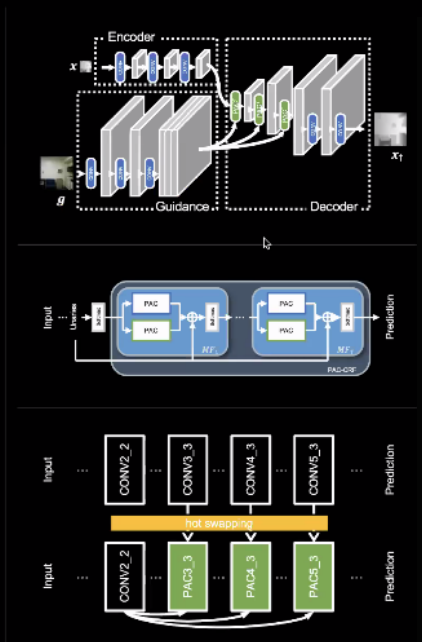
Method	mIoU	Runtime
FCN	67.20	39ms
PAC-FCN	69.18	41ms
PAC-FCN + PAC-CRF	**71.34**	118ms

Evaluated on Pascal VOC test images

Remarks: Pixel-adaptive convolution

- Pixel-Adaptive Convolution:
  - Content-adaptive
  - Generalizes several existing filtering techniques
- Three use case examples:
  - Joint upsampling networks
  - Efficient CRF inference
  - Network layer hot-swapping





Content adaptive convolution

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

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

High Dimensional Convolution

$$\mathbf{v}_i' = \sum_{j \in \Omega(i)} \mathbf{W_i}[p_i - p_j]\mathbf{v}_j + \mathbf{b}$$

Direct Filter Prediction [1,2,3]

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

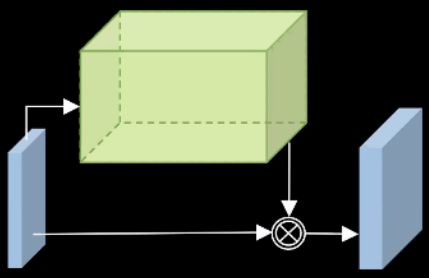
Pixel-Adaptive Convolution

[1] Brabandere et al. Dynamic filter networks. NIPS '16
[2] Bako et al. Kernel-Predicting Convolutional Networks for Denoising Monte Carlo Renderings. TOG '17
[3] Wu et al. Dynamic Sampling Convolutional Neural Networks. ECCV '18

Content-adaptive convolution

- Kernel-prediction networks / Dynamic filter networks [1,2,3]

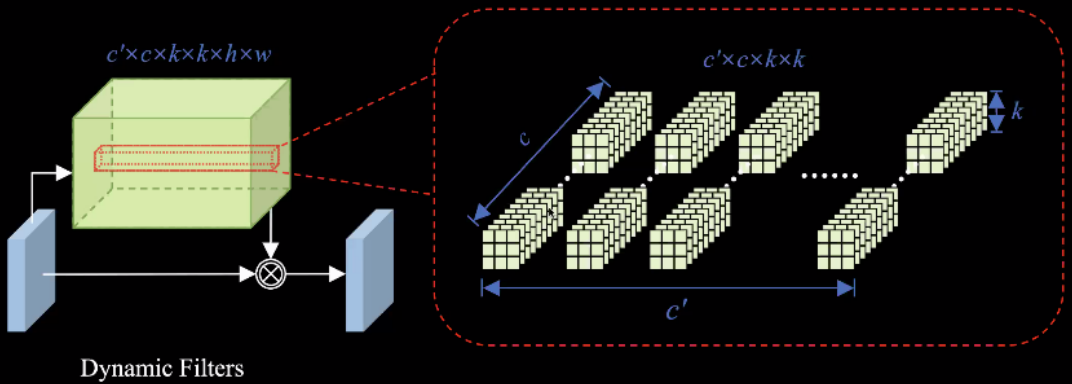
$$\mathbf{v}_i' = \sum_{j \in \Omega(i)} \mathbf{W_i}[p_i - p_j]\mathbf{v}_j + \mathbf{b}$$



[1] Brabandere et al. Dynamic filter networks. NIPS '16
[2] Bako et al. Kernel-Predicting Convolutional Networks for Denoising Monte Carlo Renderings. TOG '17
[3] Wu et al. Dynamic Sampling Convolutional Neural Networks. ECCV '18

Dynamic filter networks

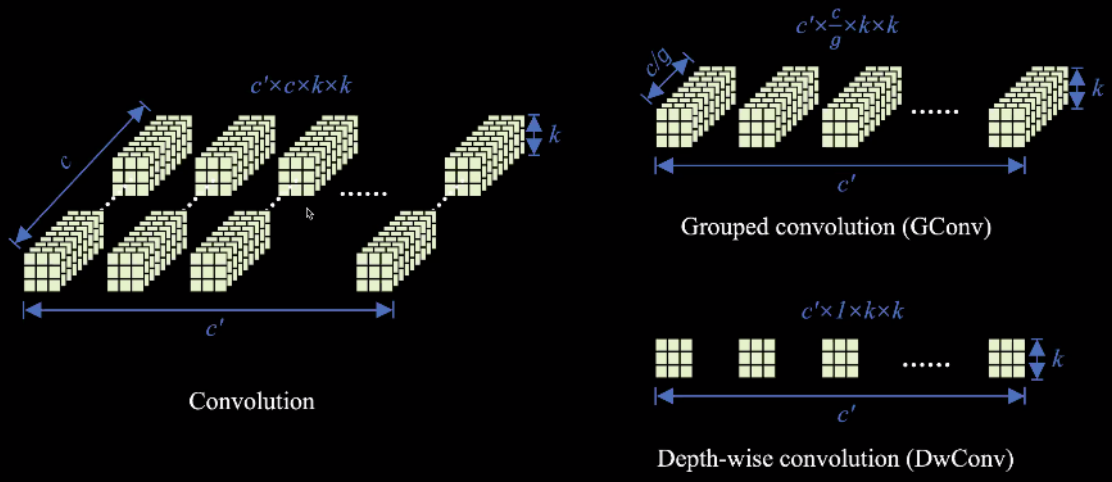
(Huge no. of parameters – Drawback)



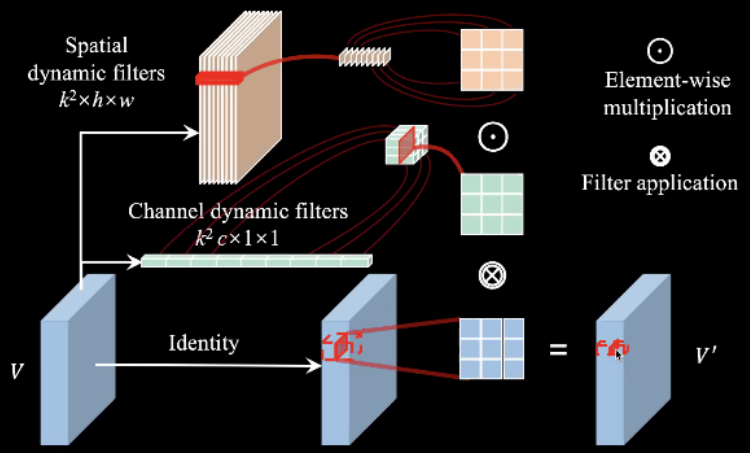
[1] Brabandere et al. Dynamic filter networks. NIPS '16
[2] Bako et al. Kernel-Predicting Convolutional Networks for Denoising Monte Carlo Renderings. TOG '17
[3] Wu et al. Dynamic Sampling Convolutional Neural Networks. ECCV '18

Reduce parameters and computation

≈ loss of performance



Decoupled Dynamic Filters [CVPR'21]



[CVPR'21] Zhou, J., Jampani, V., Pi, Z., Liu, Q. and Yang, M-H., Decoupled Dynamic Filter Networks. In *CVPR*, 2021.

Decoupled Dynamic Filter (DDF)

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

$$\mathbf{v}'_{(r,i)} = \sum_{j \in \Omega(i)} D_i^{sp}[\mathbf{p}_i - \mathbf{p}_j]\, D_r^{ch}[\mathbf{p}_i - \mathbf{p}_j]\, \mathbf{v}_{(r,j)}$$

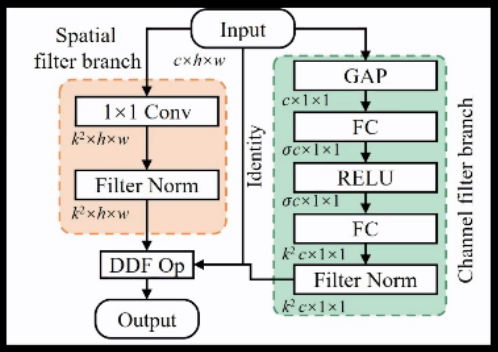
Decoupled Dynamic Filtering

[CVPR'21] Zhou, J., Jampani, V., Pi, Z., Liu, Q. and Yang, M-H., Decoupled Dynamic Filter Networks. In *CVPR*, 2021.

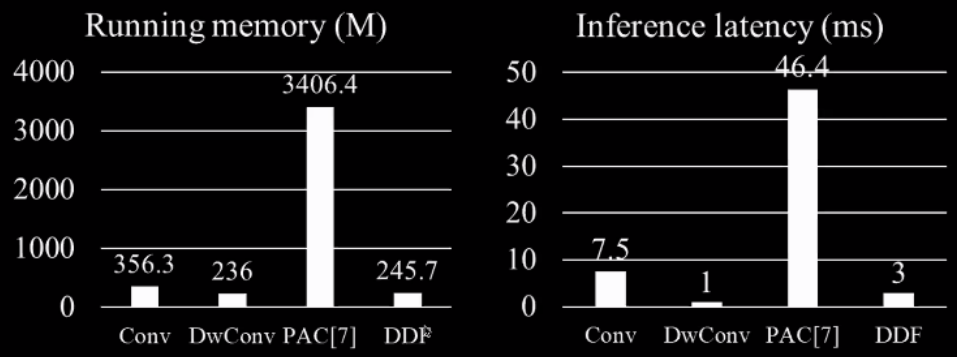
DDF module

$$\mathbf{v}'_{(r,i)} = \sum_{j \in \Omega(i)} D_i^{sp}[\mathbf{p}_i - \mathbf{p}_j]\, D_r^{ch}[\mathbf{p}_i - \mathbf{p}_j]\, \mathbf{v}_{(r,j)}$$

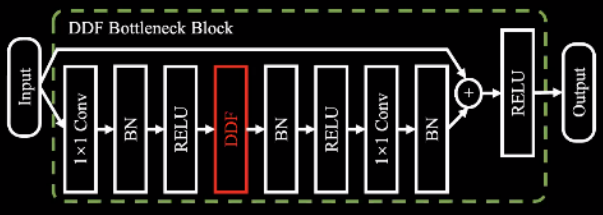
Decoupled Dynamic Filtering



DDF is both memory and computationally efficient



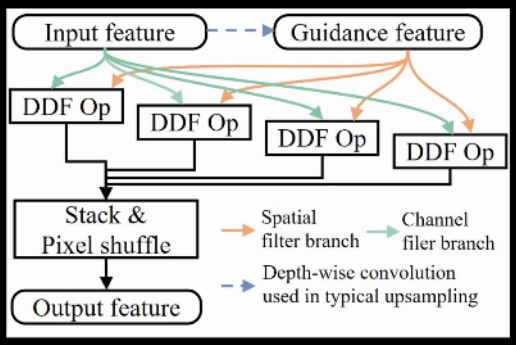
DDF networks for image classification



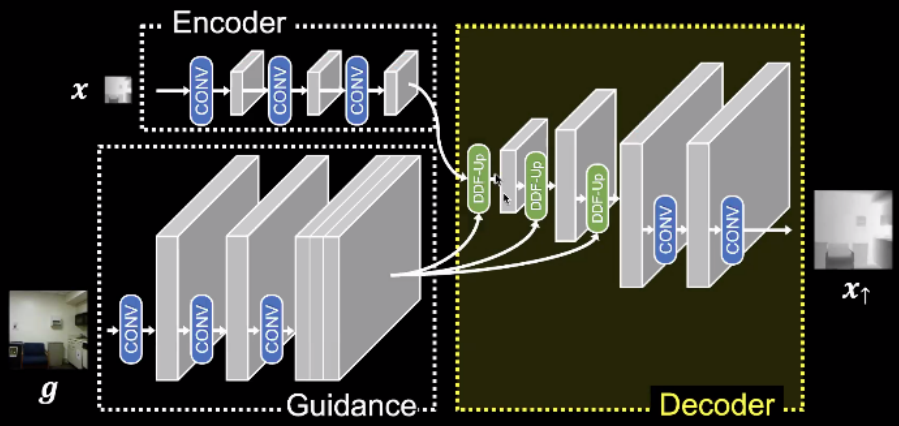
Comparison against related filters

Arch	ConvType	Params	FLOPs	Top-1 Acc
R18	*Base Model*	11.7M	1.8B	69.6
	Adaptive	11.1M	–	70.2
	DyNet	16.6M	0.6B	69.0
	DDF	**7.7M**	**0.4B**	**70.6**
R50	*Base Model*	25.6M	4.1B	77.2
	DyNet	–	**1.1B**	76.3
	CondConv	104.8M	4.2B	78.6
	DwCondConv	**14.5M**	2.3B	78.3
	DDF	**16.8M**	**2.3B**	**79.1**
R101	*Base Model*	44.5M	7.8B	78.9
	DDF	**28.1M**	**4.1B**	**80.2**

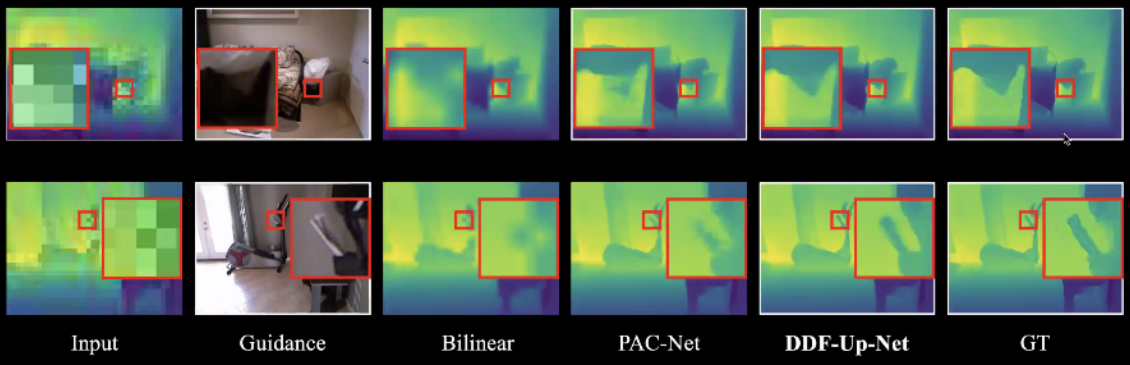
DDF for joint image filtering



Joint upsampling network with DDF-Up



Joint depth upsampling with DDF-Up



Joint depth upsampling with DDF-Up

RMSE results on the NYU Depth V2 dataset

Methods	4×	8×	16×
Bicubic	8.16	14.22	22.32
JBU[1]	4.07	8.29	13.35
DJF[2]	3.54	6.20	10.21
DJF+[3]	3.38	5.86	10.11
PAC	2.39	4.59	8.09
DDF-Up	**2.16**	**4.44**	**7.72**

[1] Johannes Kopf, et al. *Joint bilateral upsampling*. ACM Transactions on Graphics, 2007.
[2] Yijun Li, et al. *Deep joint image filtering*. CVPR, 2016.
[3] Yijun Li, et al. *Joint image filtering with deep convolutional networks*. PAMI, 2019.

DDF: Remarks

- Decoupled dynamic filtering
  - Learn to predict decoupled spatial and channel dynamic filters

$$\mathbf{v}_i' = \sum_{j \in \Omega(i)} \mathbf{W}[\mathbf{p}_i - \mathbf{p}_j]\mathbf{v}_j \longrightarrow \mathbf{v}'_{(r,i)} = \sum_{j \in \Omega(i)} D_i^{sp}[\mathbf{p}_i - \mathbf{p}_j]\, D_r^{ch}[\mathbf{p}_i - \mathbf{p}_j]\, \mathbf{v}_{(r,j)}$$

- Several favorable properties
  - Content-adaptive
  - Readily replace convolution operation (faster and better performance)
  - Performance even better than existing content-adaptive filters
  - Small memory footprint

Summary and Future outlook

Sparse High Dimensional Convolutions

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

- Can filter sparse and high-dimensional data.
- Helps in long-range information propagation.
- Manual specification of feature spaces.
- Future outlook: Learn lattice features via backpropagation.

Pixel Adaptive Convolutions

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

- Learns guidance features.
- Fast and can hot-swap standard convolutions.
- Works only on dense 2D data.
- Future outlook: Other applications and filtering sparse and high-dimensional data.

Decoupled Dynamic Filters

$$\mathbf{v}'_{(r,i)} = \sum_{j \in \Omega(i)} D_i^{sp}[\mathbf{p}_i - \mathbf{p}_j]\, D_r^{ch}[\mathbf{p}_i - \mathbf{p}_j]\, \mathbf{v}_{(r,j)}$$

- Learns decoupled spatial and channel dynamic filters.
- Faster, lighter-weight and better performance than convolutions and also existing content adaptive filters.
- Works only on dense 2D data.
- Future outlook: Other applications and filtering sparse and high-dimensional data.

Thank you

Comments and suggestions are most welcome!

Varun Jampani
https://varunjampani.github.io/