Denoising Algorithms: Path to Neural Networks III

Image courtesy Vogel et al. [2018], Gharbi et al. [2019]

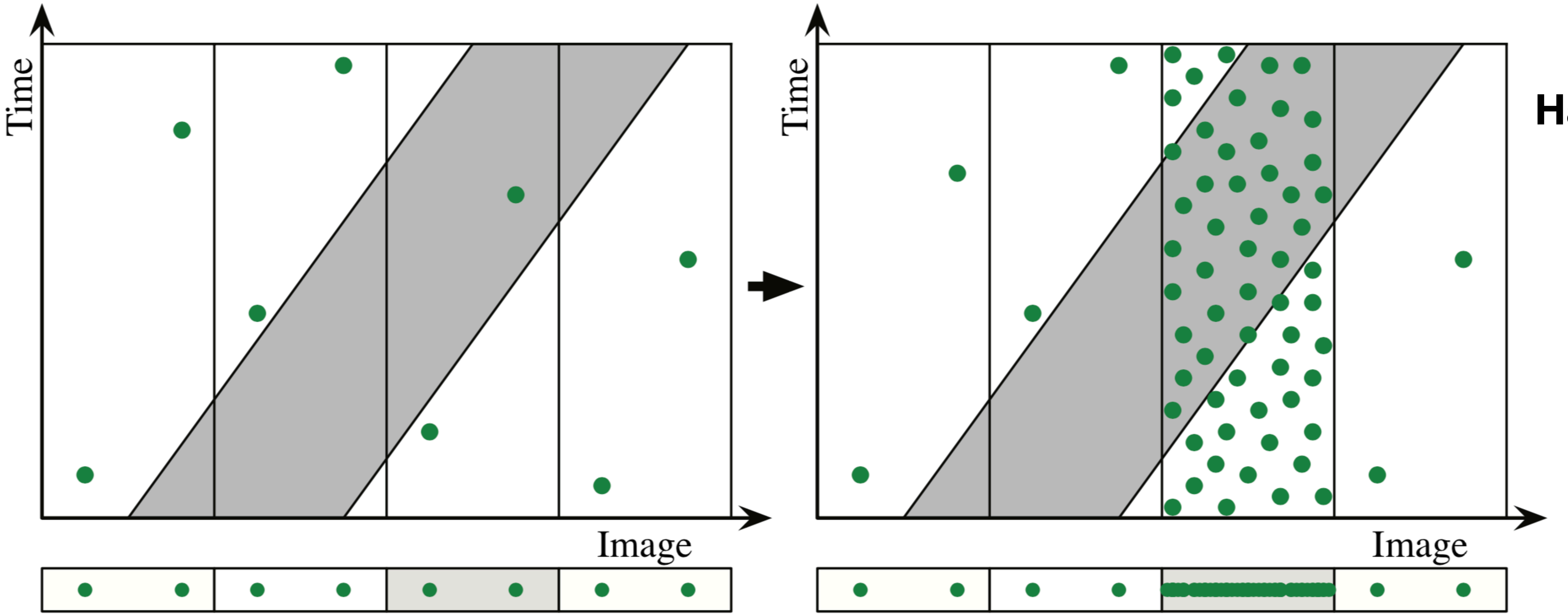
Philipp Slusallek *Karol Myszkowski* ***Gurprit Singh***

Final Exam

20.08.19 from 10:00 to 13:00 in HS 002

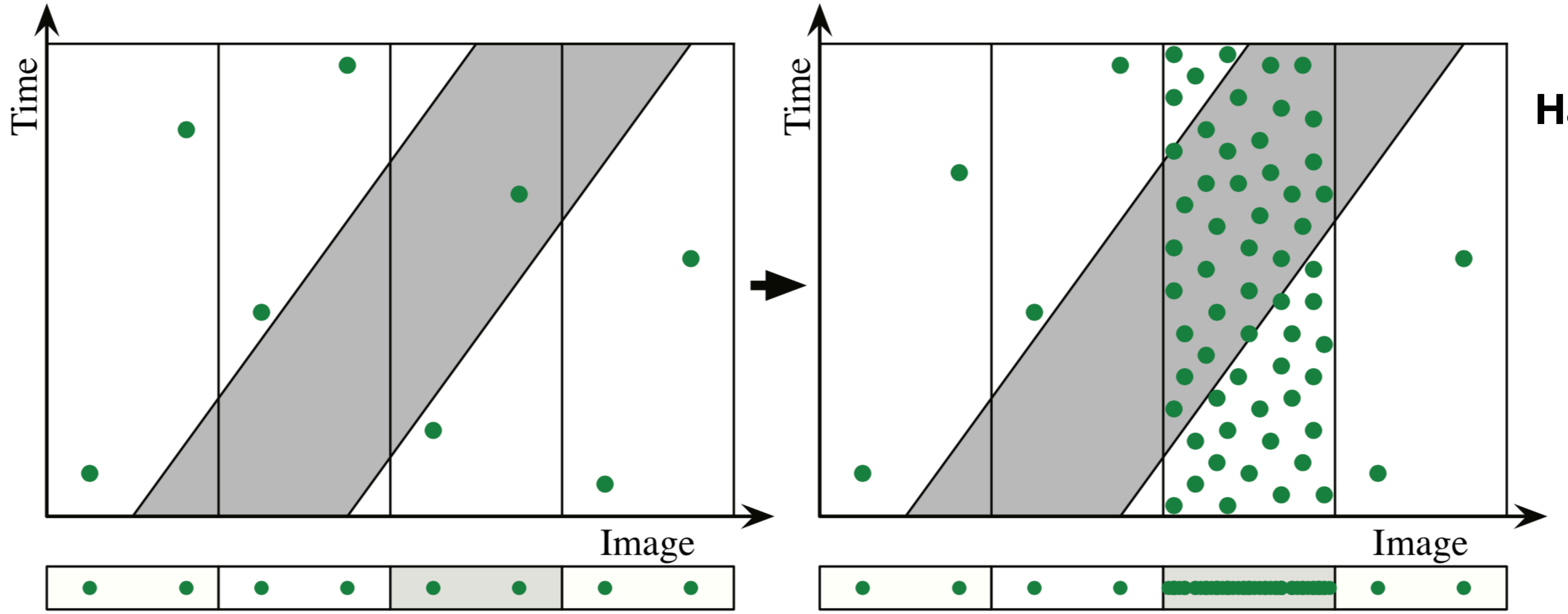
Recap

Image-space Adaptive Sampling



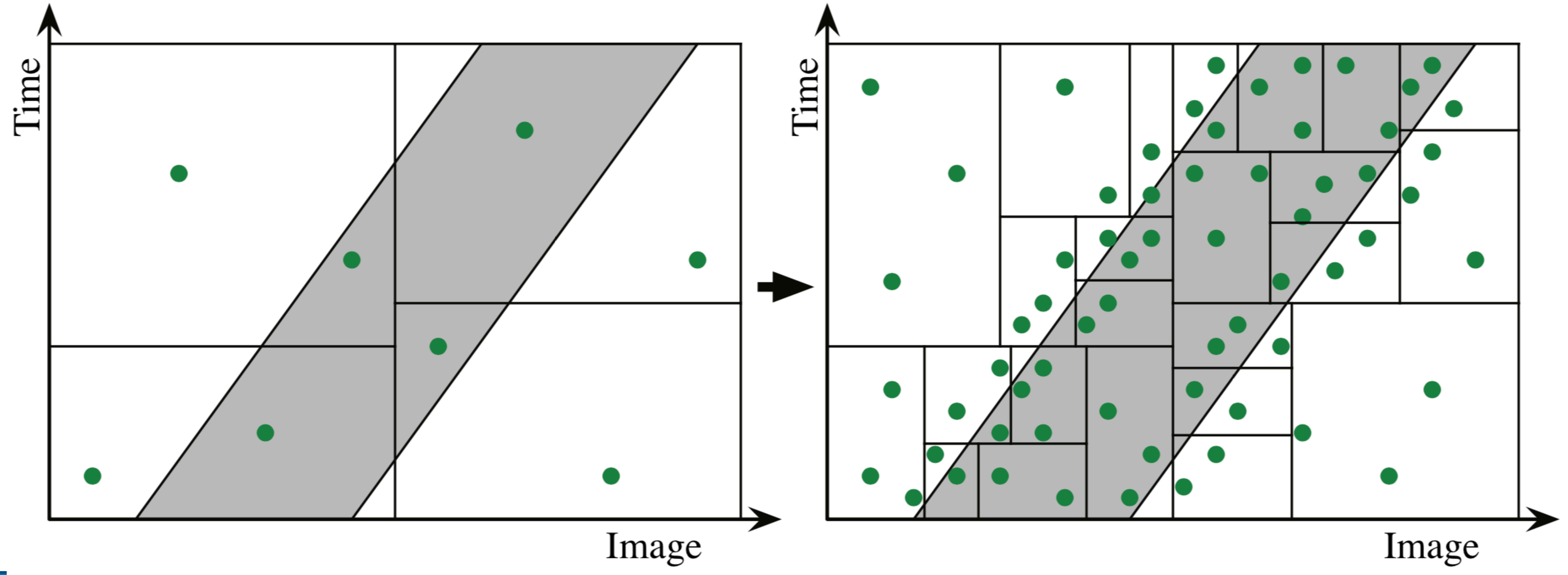
Hachisuka et al. [2008]

Image-space Adaptive Sampling

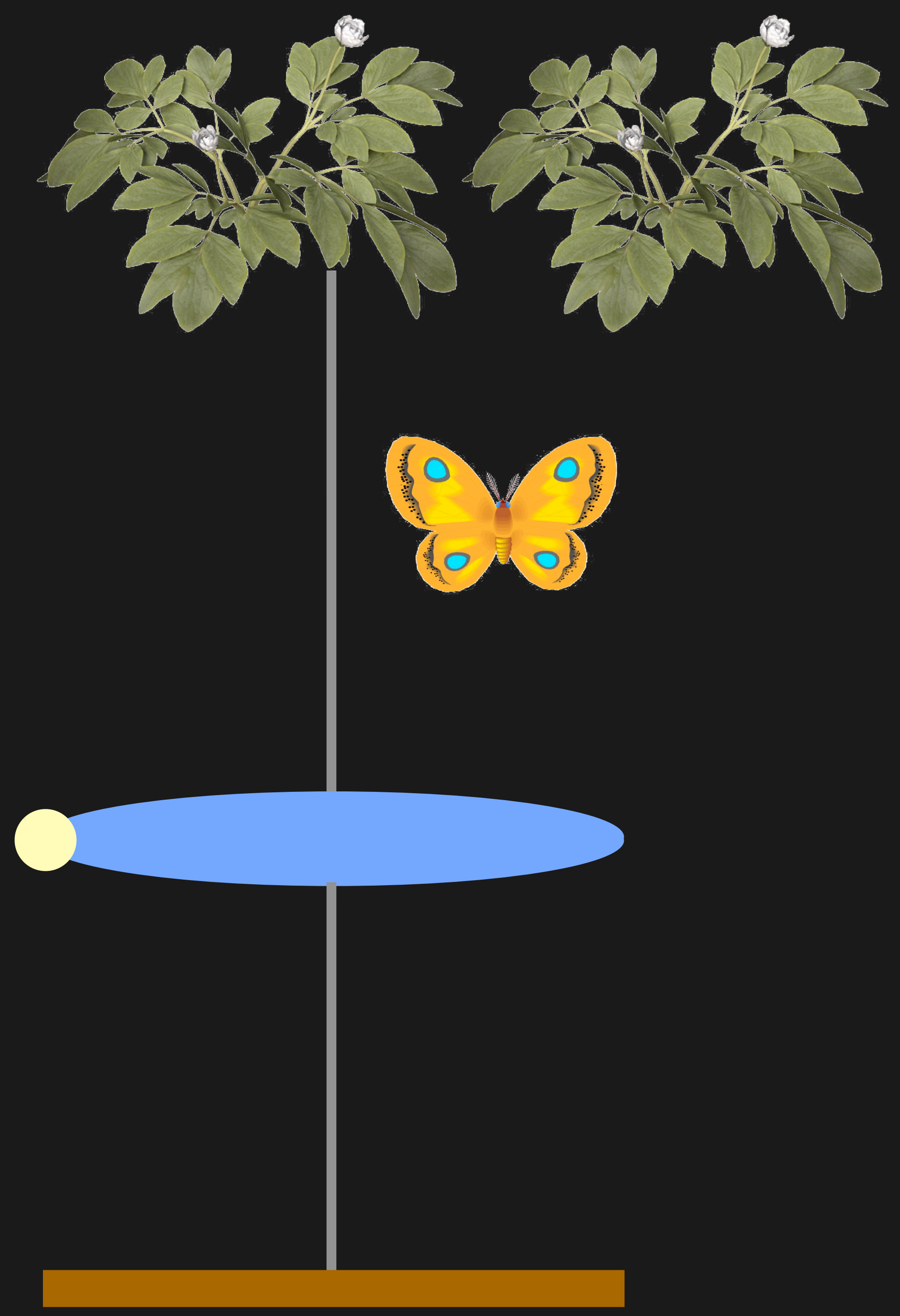


Hachisuka et al. [2008]

Multidimensional Adaptive Sampling



Depth of field

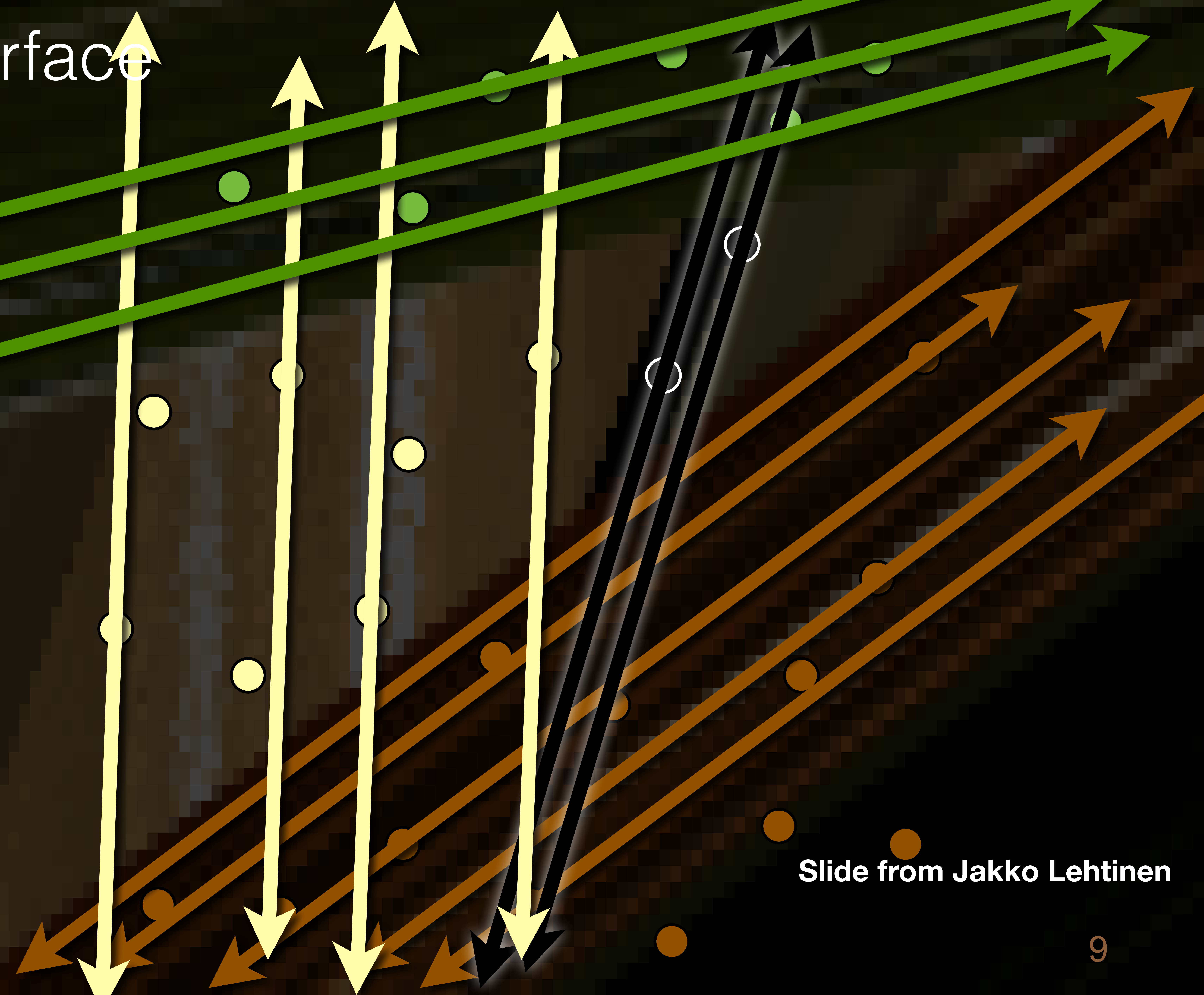


Slide from Jakko Lehtinen

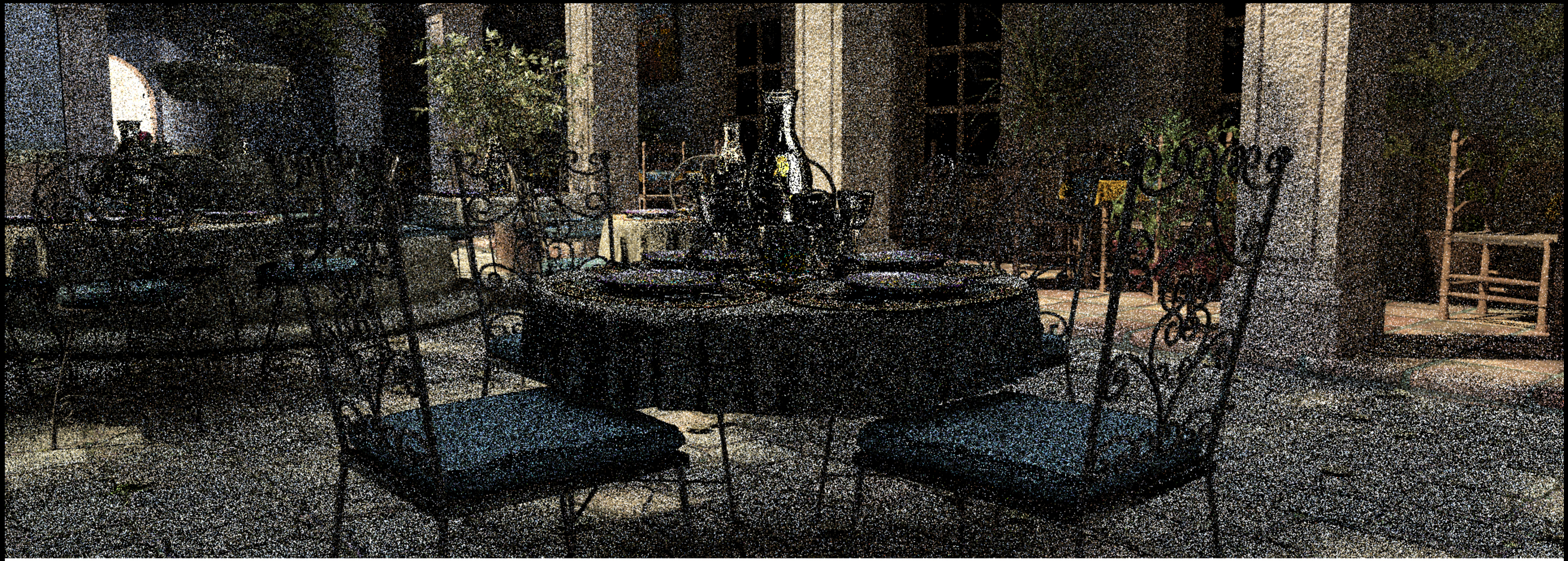

Lens u
Screen x
Slide from Jakko Lehtinen

Visibility: SameSurface

The trajectories of samples originating from a single **apparent surface** never intersect.



Slide from Jakko Lehtinen



input Monte Carlo (8 samples/pixel)

after RPF (8 samples/pixel)

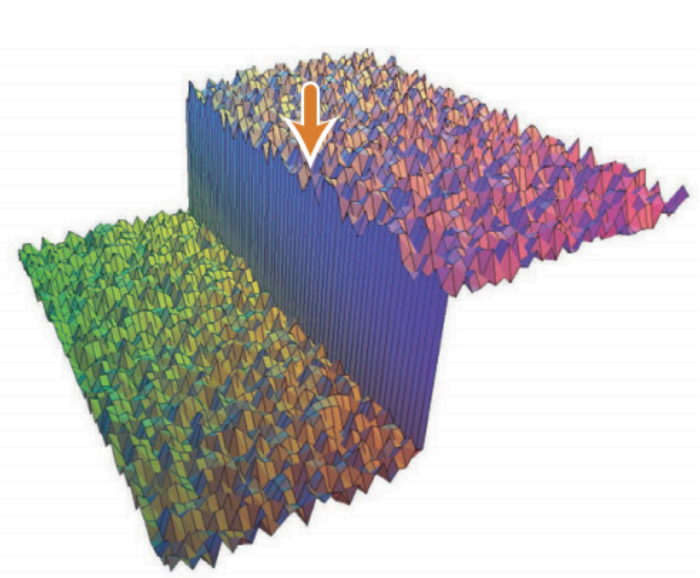
(a) Input MC (8 spp) **(b)** Dependency on (u, v) **(c)** Our approach (RPF)

Bilateral Filtering

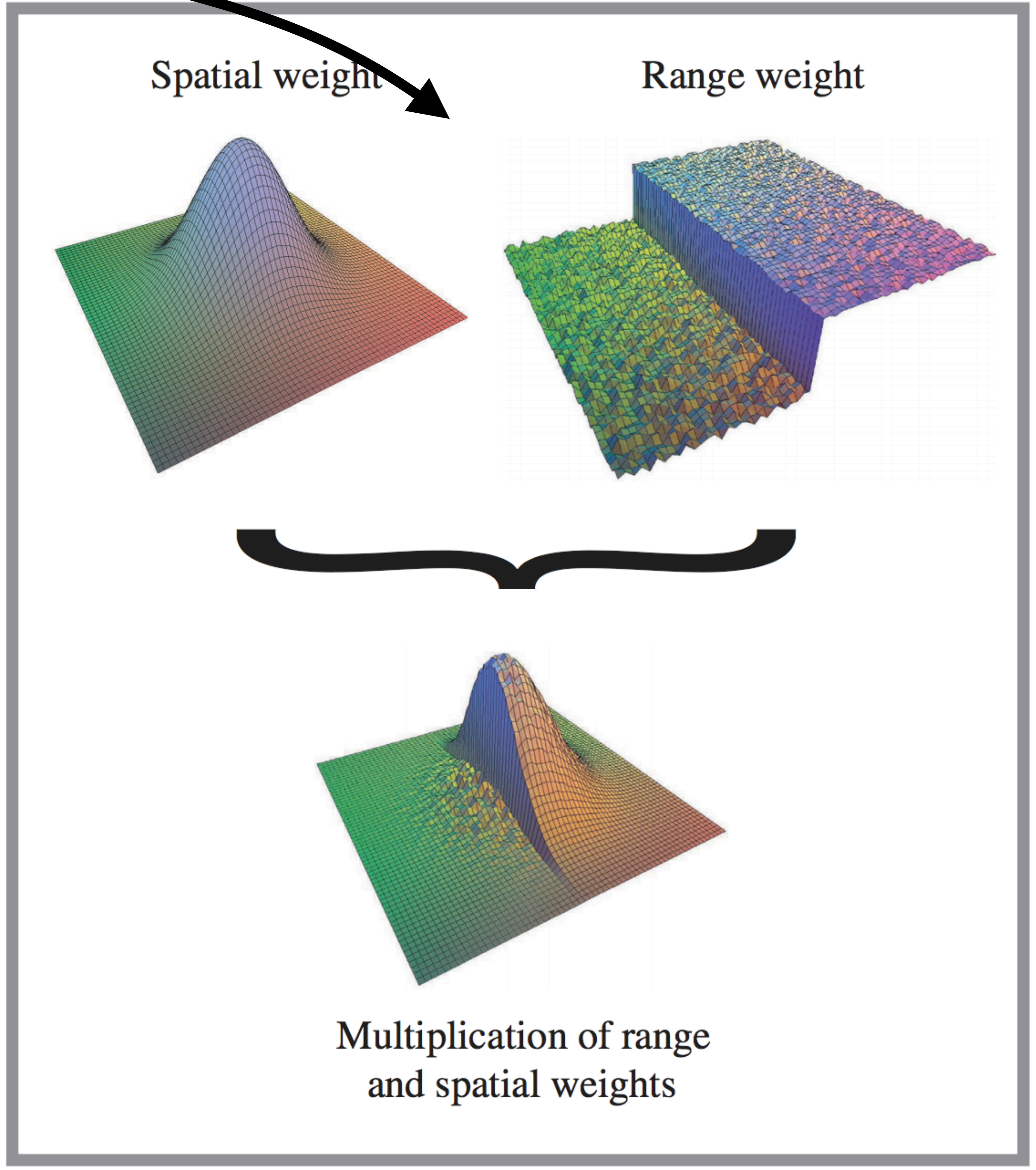
$$BF[I]_{\mathbf{p}} = \frac{1}{W_{\mathbf{p}}} \sum_{\mathbf{q}\in\mathcal{S}} G_{\sigma_s}(\|\mathbf{p}-\mathbf{q}\|)\, G_{\sigma_r}(|I_{\mathbf{p}} - I_{\mathbf{q}}|)\, I_{\mathbf{q}}$$

$$W_{\mathbf{p}} = \sum_{\mathbf{q}\in\mathcal{S}} G_{\sigma_s}(\|\mathbf{p}-\mathbf{q}\|)\, G_{\sigma_r}(|I_{\mathbf{p}} - I_{\mathbf{q}}|)$$

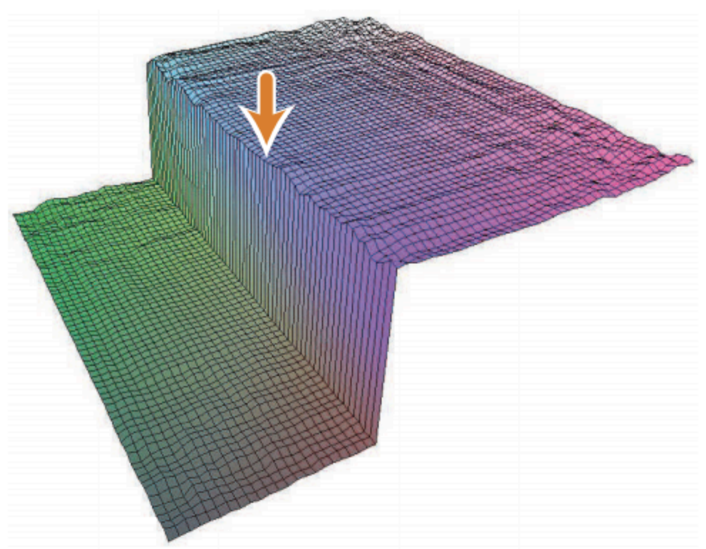
Input

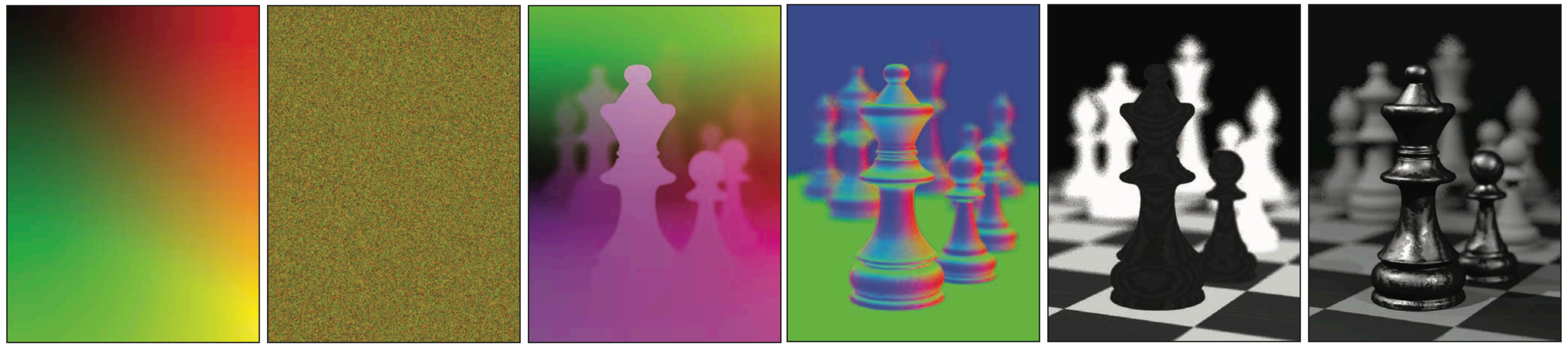


Bilateral filter weights at the central pixel



Result





(a) Screen position **(b)** Random parameters **(c)** World space coords. **(d)** Surface normals **(e)** Texture value **(f)** Sample color

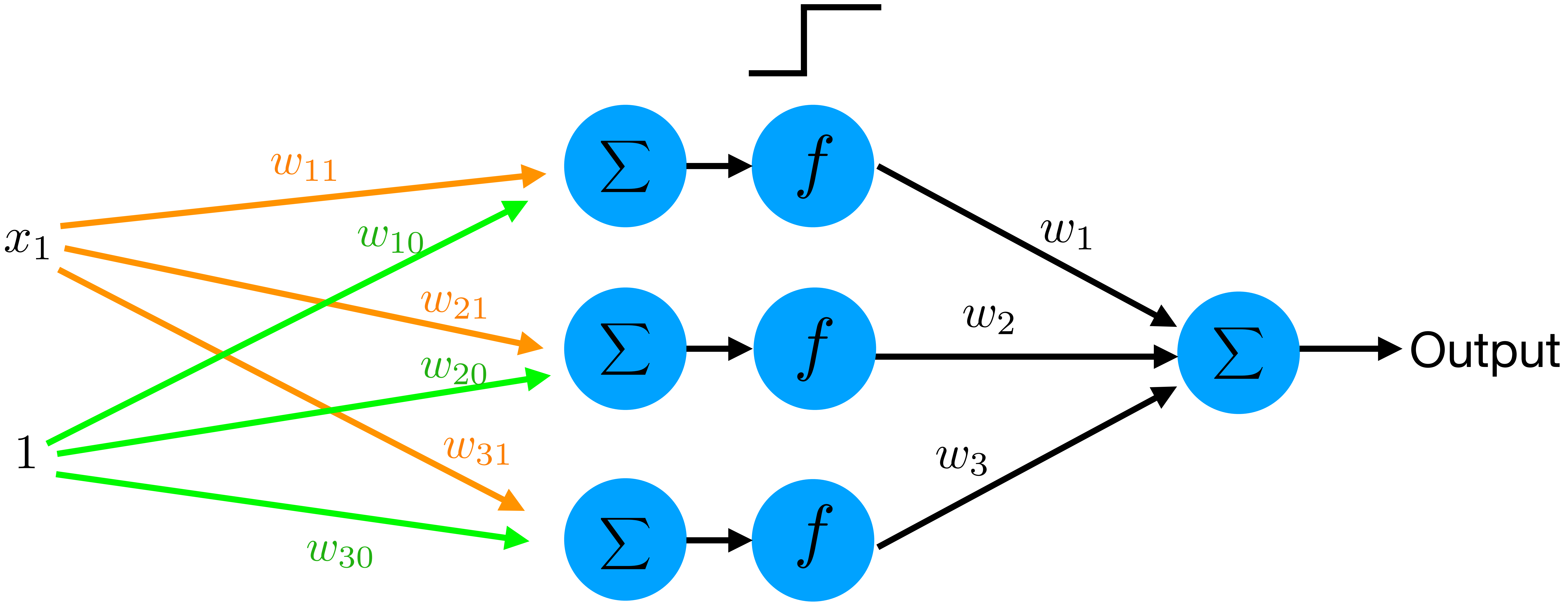
Bilateral Filtering of Features

$$w_{ij} = \exp[-\frac{1}{2\sigma_{\mathbf{p}}^2} \sum_{1 \leq k \leq 2} (\bar{\mathbf{p}}_{i,k} - \bar{\mathbf{p}}_{j,k})^2] \times$$

$$\exp[-\frac{1}{2\sigma_{\mathbf{c}}^2} \sum_{1 \leq k \leq 3} \boxed{\alpha_k} (\bar{\mathbf{c}}_{i,k} - \bar{\mathbf{c}}_{j,k})^2] \times$$

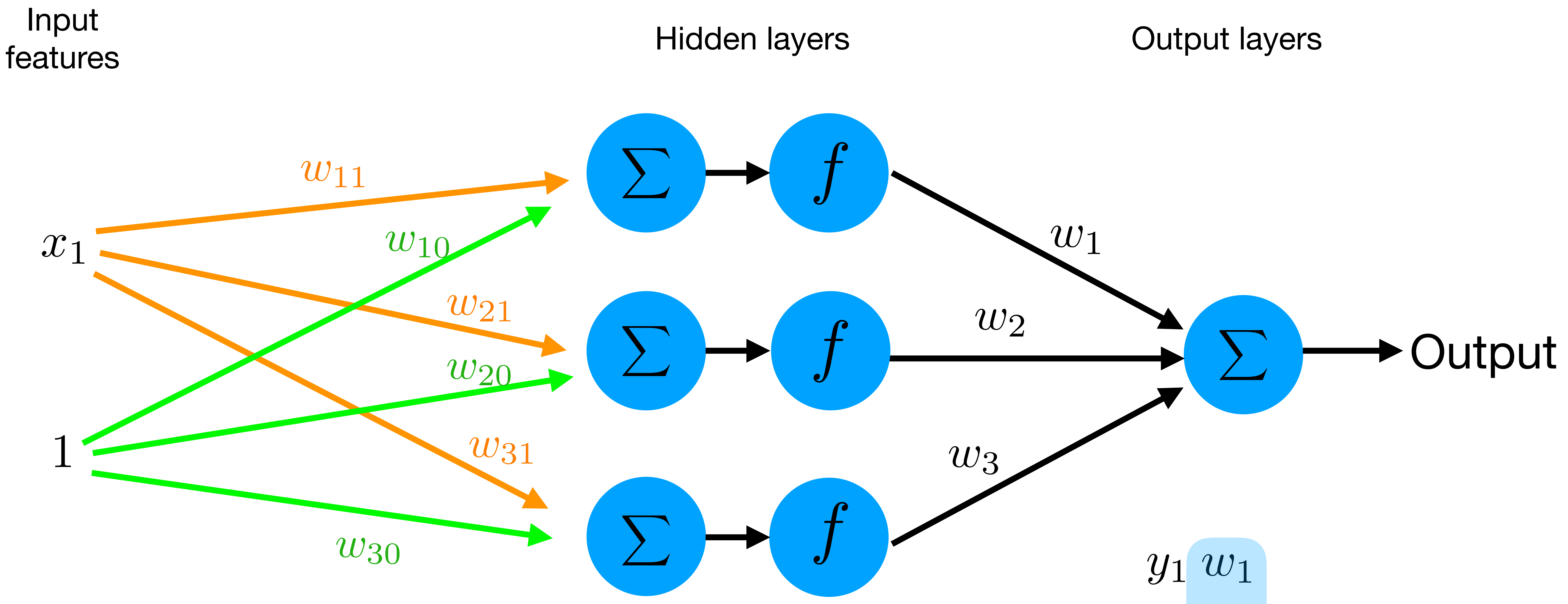
$$\exp[-\frac{1}{2\sigma_{\mathbf{f}}^2} \sum_{1 \leq k \leq m} \boxed{\beta_k} (\bar{\mathbf{f}}_{i,k} - \bar{\mathbf{f}}_{j,k})^2],$$

Multi-layer Perceptron



$$y_1 = f(x_1 w_{11} + w_{10})$$
$$y_2 = f(x_1 w_{21} + w_{20})$$
$$y_3 = f(x_1 w_{31} + w_{30})$$

Multi-layer Perceptron



$$y_1 = f(x_1 w_{11} + w_{10})$$
$$y_2 = f(x_1 w_{21} + w_{20})$$
$$y_3 = f(x_1 w_{31} + w_{30})$$

$y_1 \; w_1$

$y_2 \; w_2$

$y_3 \; w_3$

Filter weights

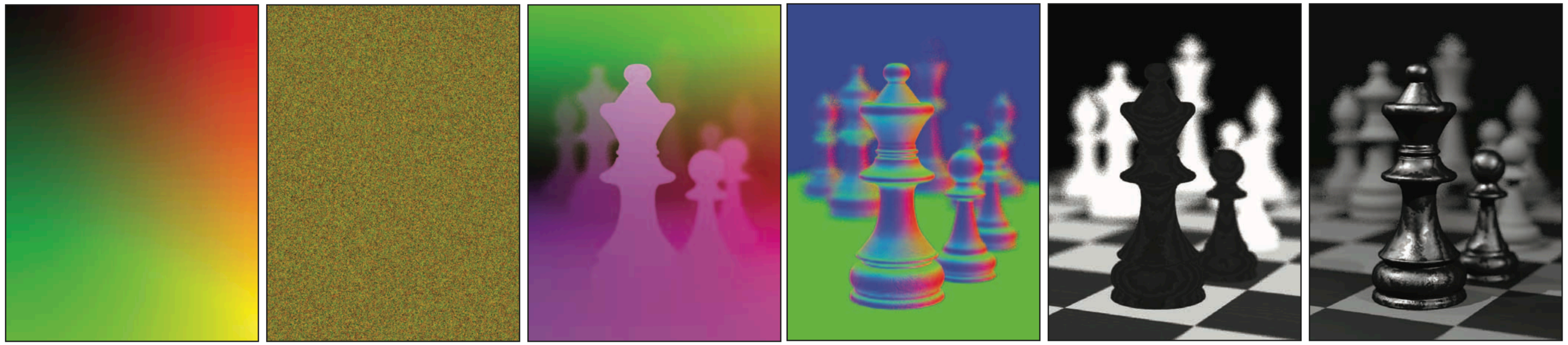
For cross Bilateral filters:

$$d_{i,j} = \exp\left[-\frac{\|\bar{\mathbf{p}}_i - \bar{\mathbf{p}}_j\|^2}{2\alpha_i^2}\right] \times \exp\left[-\frac{D(\bar{\mathbf{c}}_i, \bar{\mathbf{c}}_j)}{2\beta_i^2}\right] \times \prod_{k=1}^{K} \exp\left[-\frac{D_k(\bar{\mathbf{f}}_{i,k}, \bar{\mathbf{f}}_{j,k})}{2\gamma_{k,i}^2}\right],$$

Pixel screen coordinates

Mean sample color value

Scene features



(a) Screen position **(b)** Random parameters **(c)** World space coords. **(d)** Surface normals **(e)** Texture value **(f)** Sample color

Our result with a cross-bilateral filter (4 spp)

Overview on Convolutional Neural Networks (CNNs)

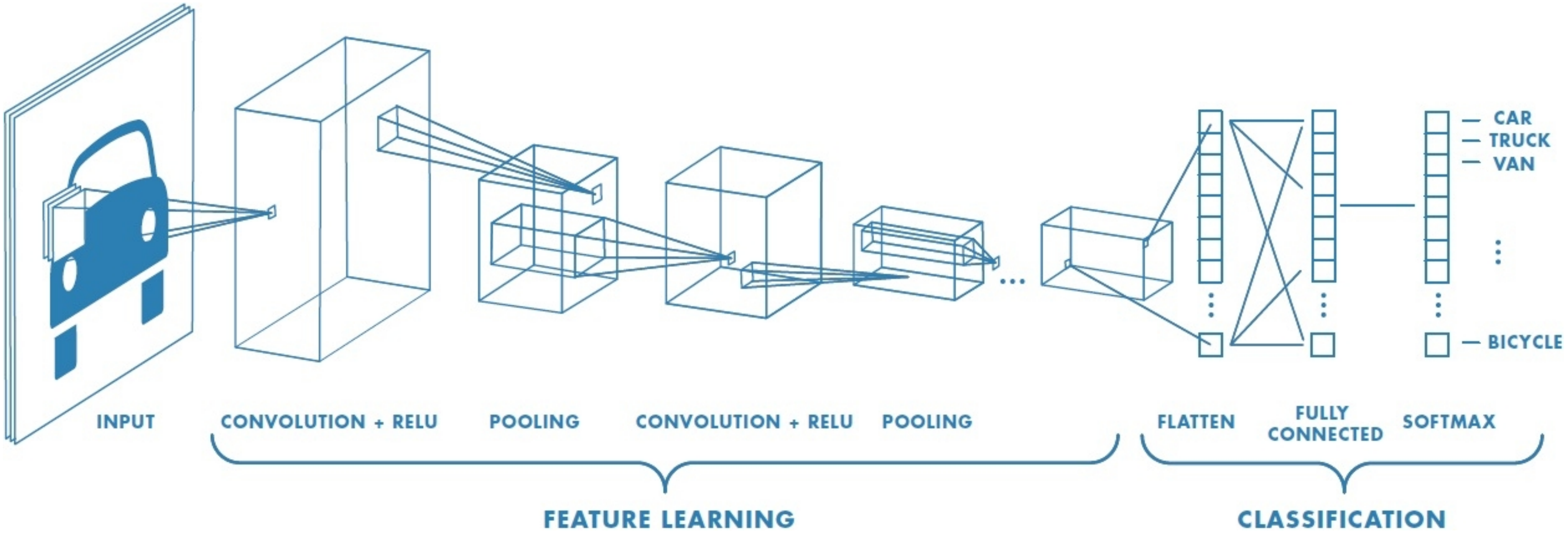
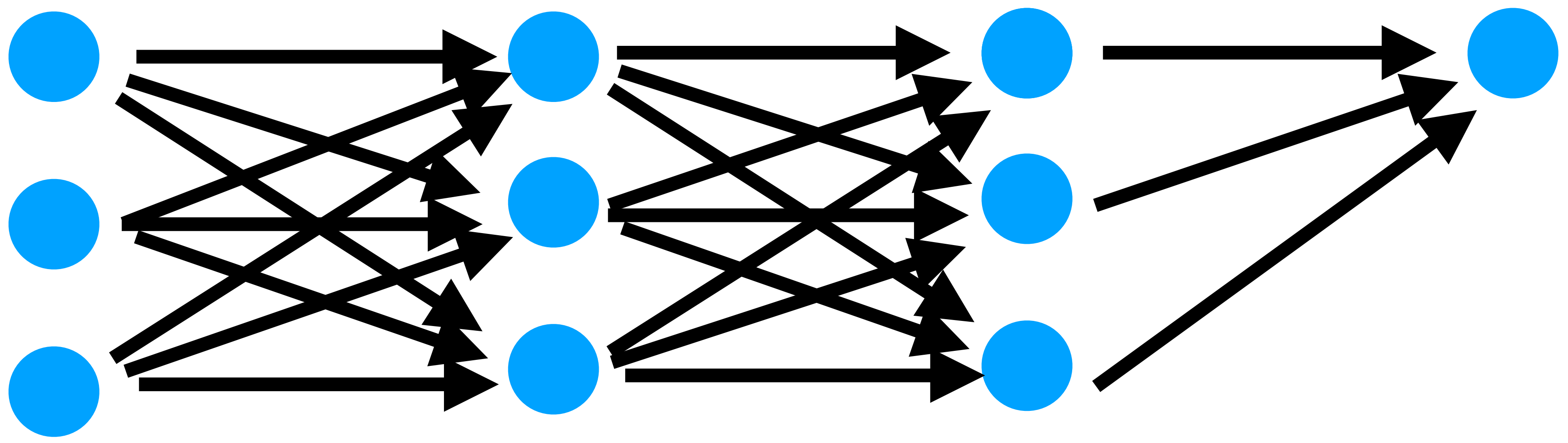
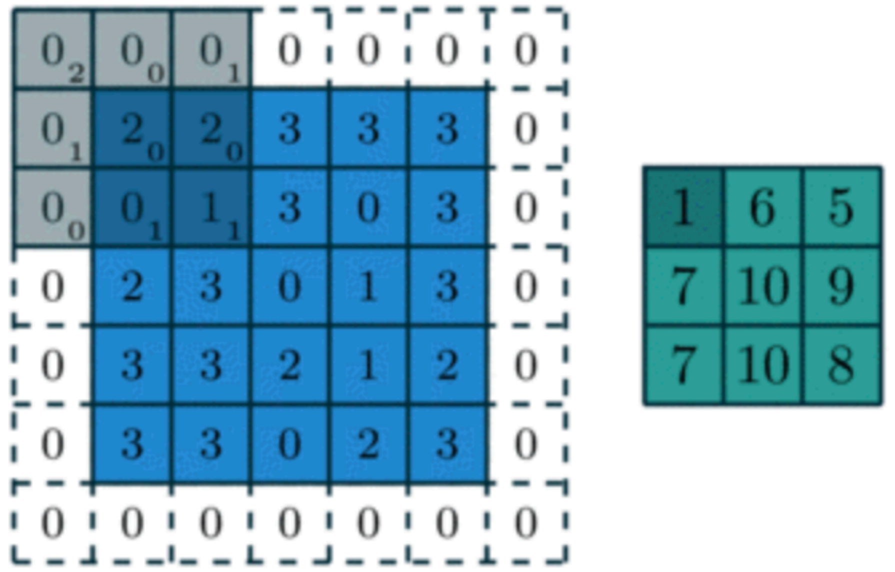
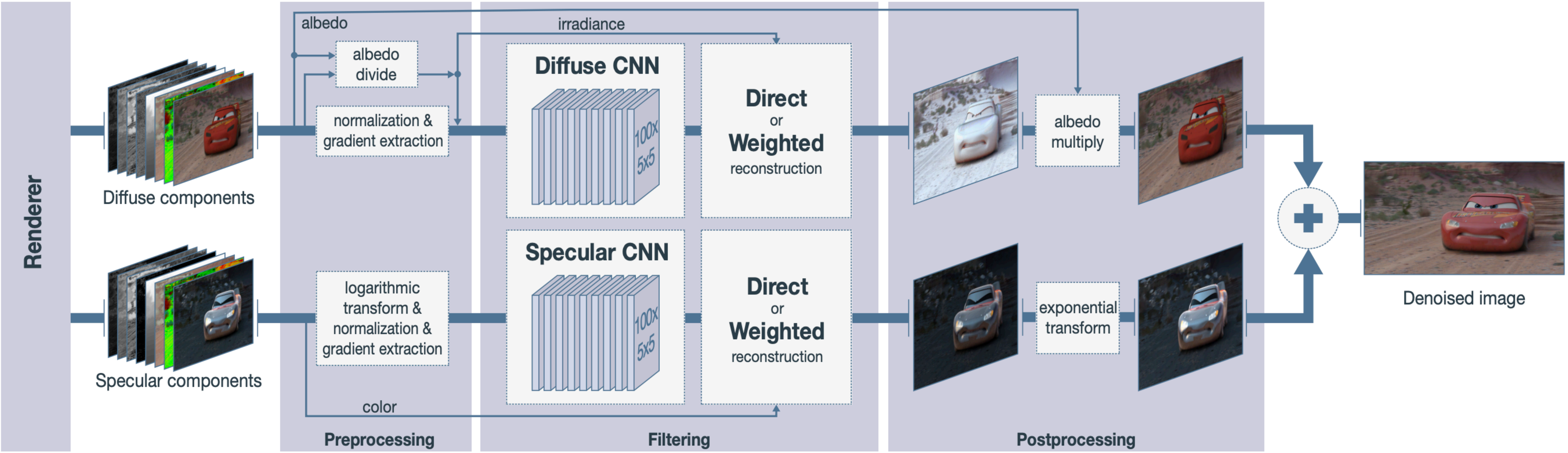


Image Courtesy: Mathworks (online tutorial)

Multi-layer Perceptron vs. CNNs

Multi-layer perceptron	CNNs
	
All nodes are fully connected in all layers	Weights are shared across layers
In theory, should be able to achieve good quality results in small number of layers.	Requires significant number of layers to capture all the features (e.g. Deep CNNs)
Number of weights to be learnt are very high	Relatively small number of weights required

Kernel-Predicting Networks for Denoising Monte-Carlo Renderings



Recurrent AutoEncoder for Interactive Reconstruction

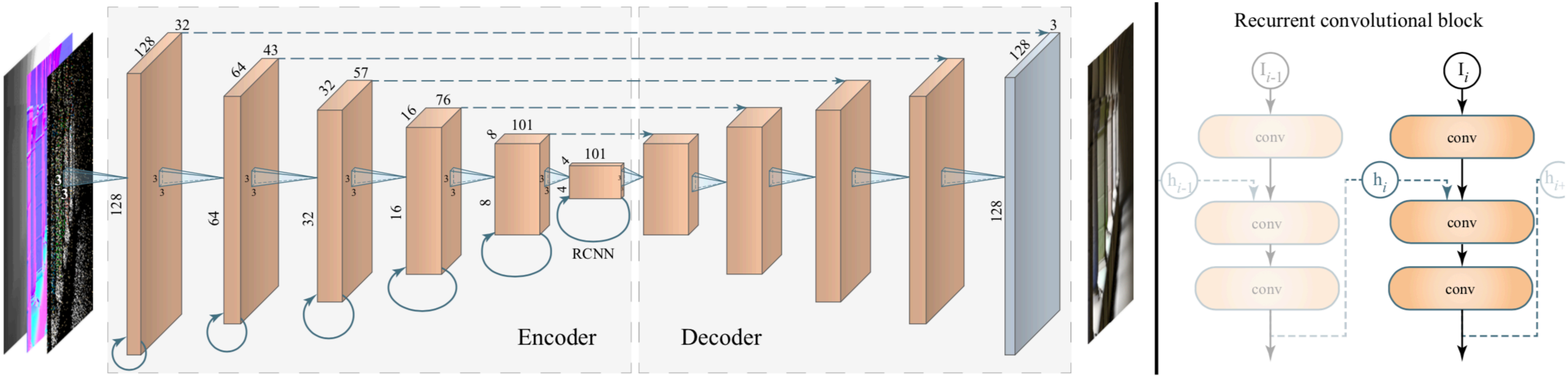
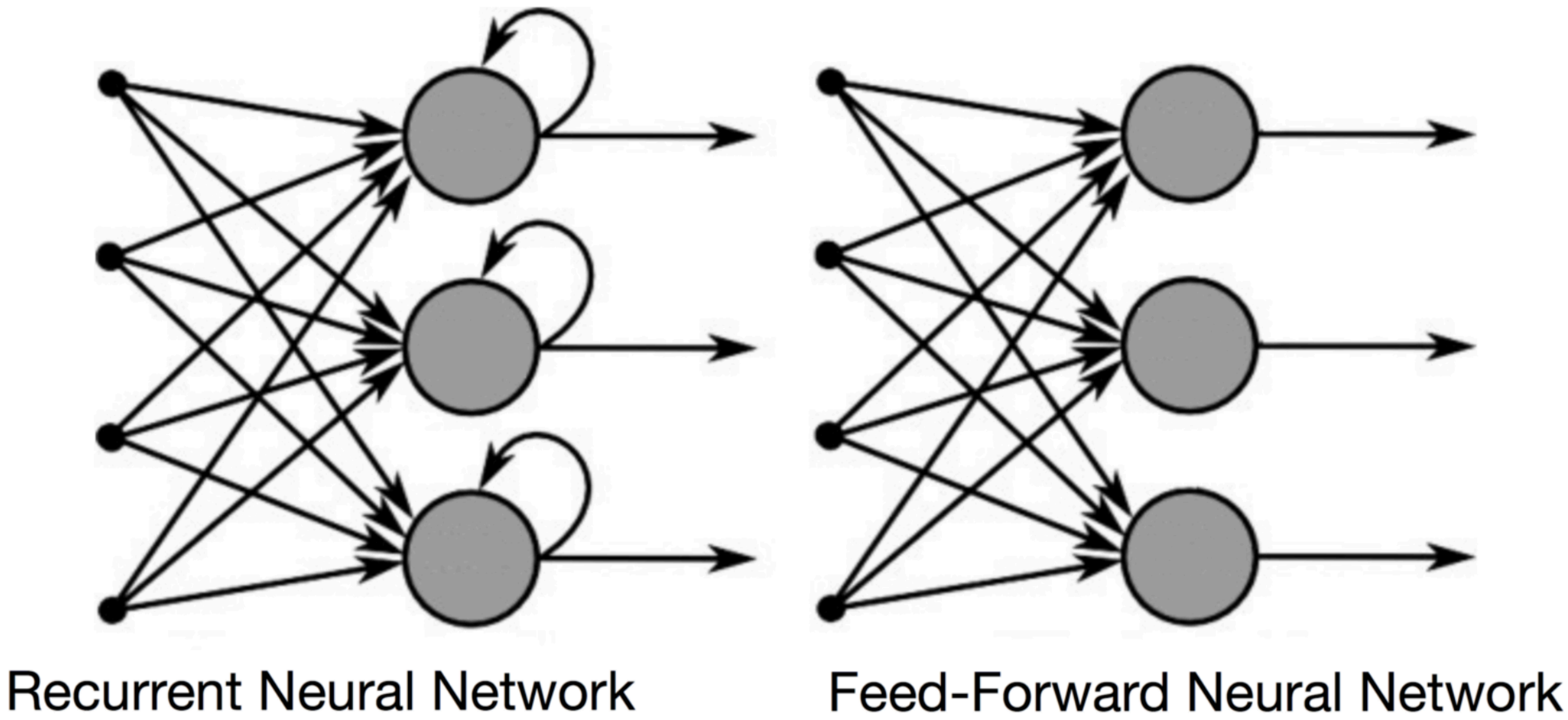


Fig. 2. Architecture of our recurrent autoencoder. The input is 7 scalar values per pixel (noisy RGB, normal vector, depth, roughness). Each encoder stage has a convolution and 2×2 max pooling. A decoder stage applies a 2×2 nearest neighbor upsampling, concatenates the per-pixel feature maps from a skip connection (the spatial resolutions agree), and applies two sets of convolution and pooling. All convolutions have a 3×3-pixel spatial support. On the right we visualize the internal structure of the recurrent RCNN connections. I is the new input and h refers to the hidden, recurrent state that persists between animation frames.

Recurrent Neural Networks vs. Simple Feed-Forward NN

Source link



Loss Functions

Spatial Loss to emphasize more the dark regions

$$L_s = \frac{1}{N}\sum_{i}^{N}|P_i - T_i|$$

Temporal loss

$$L_t = \frac{1}{N}\sum_{i}^{N}\left(\left|\frac{\partial P_i}{\partial t} - \frac{\partial T_i}{\partial t}\right|\right)$$

High frequency error norm loss for stable edges

$$L_g = \frac{1}{N}\sum_{i}^{N}|\nabla P_i - \nabla T_i|$$

Final Loss is a weighted averaged of above losses

$$L = w_s L_s + w_g L_g + w_t L_t$$

Pixel-space Kernel Predicting Denoising

#Learnable Parameters?

How to compute "learnable" parameters?

Sample-based MC Denoising

How to compute "learnable" parameters?

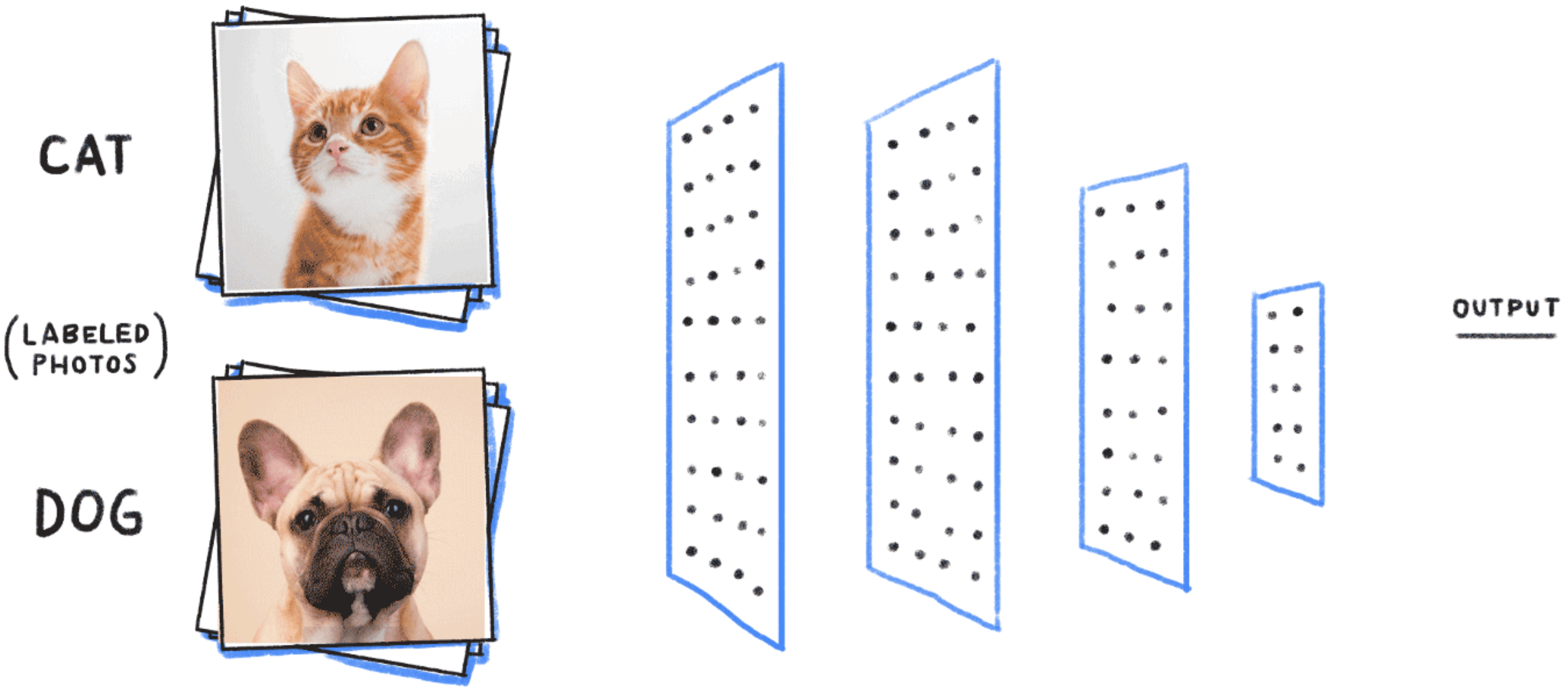


Image Source: Google

How to compute "learnable" parameters?

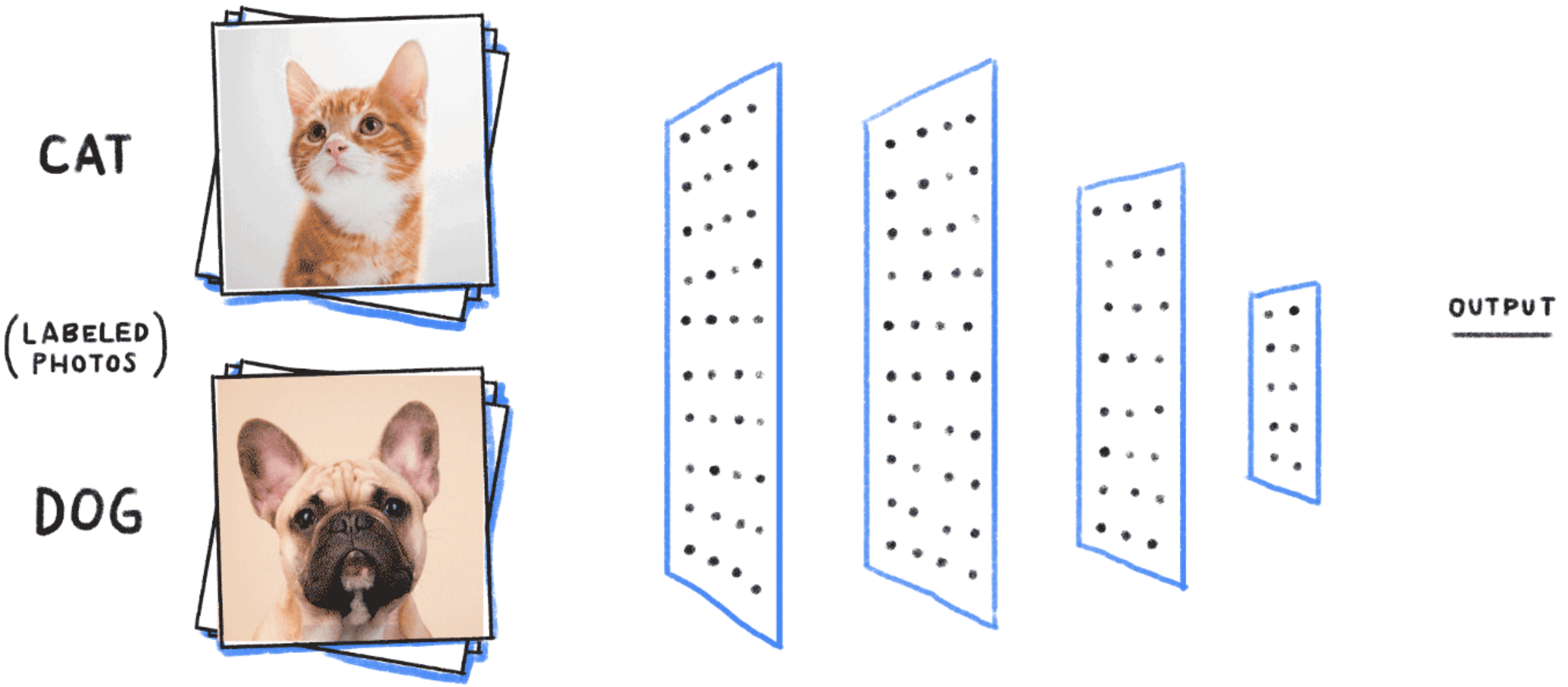


Image Source: Google

Feed-Forward Neural Network

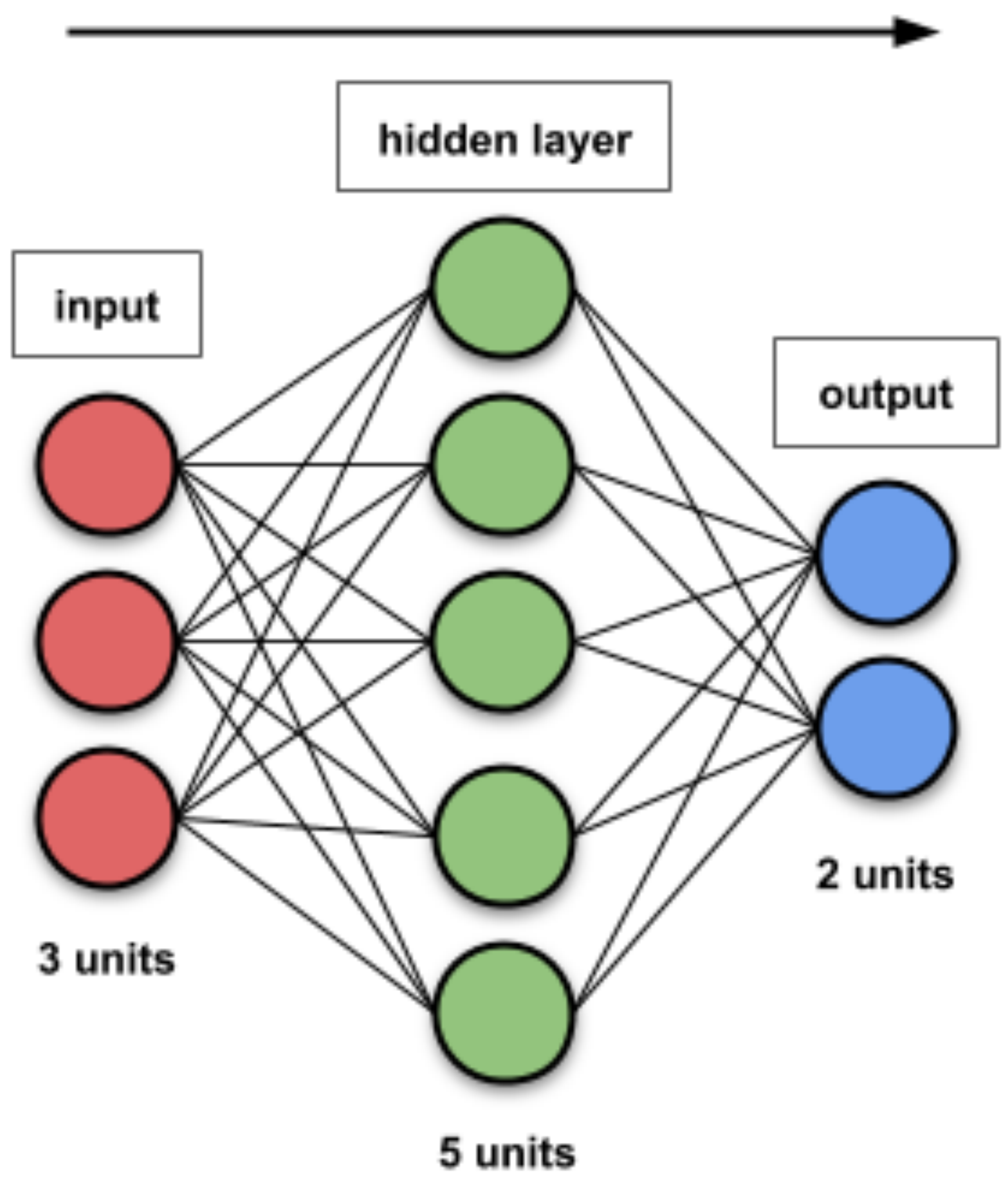
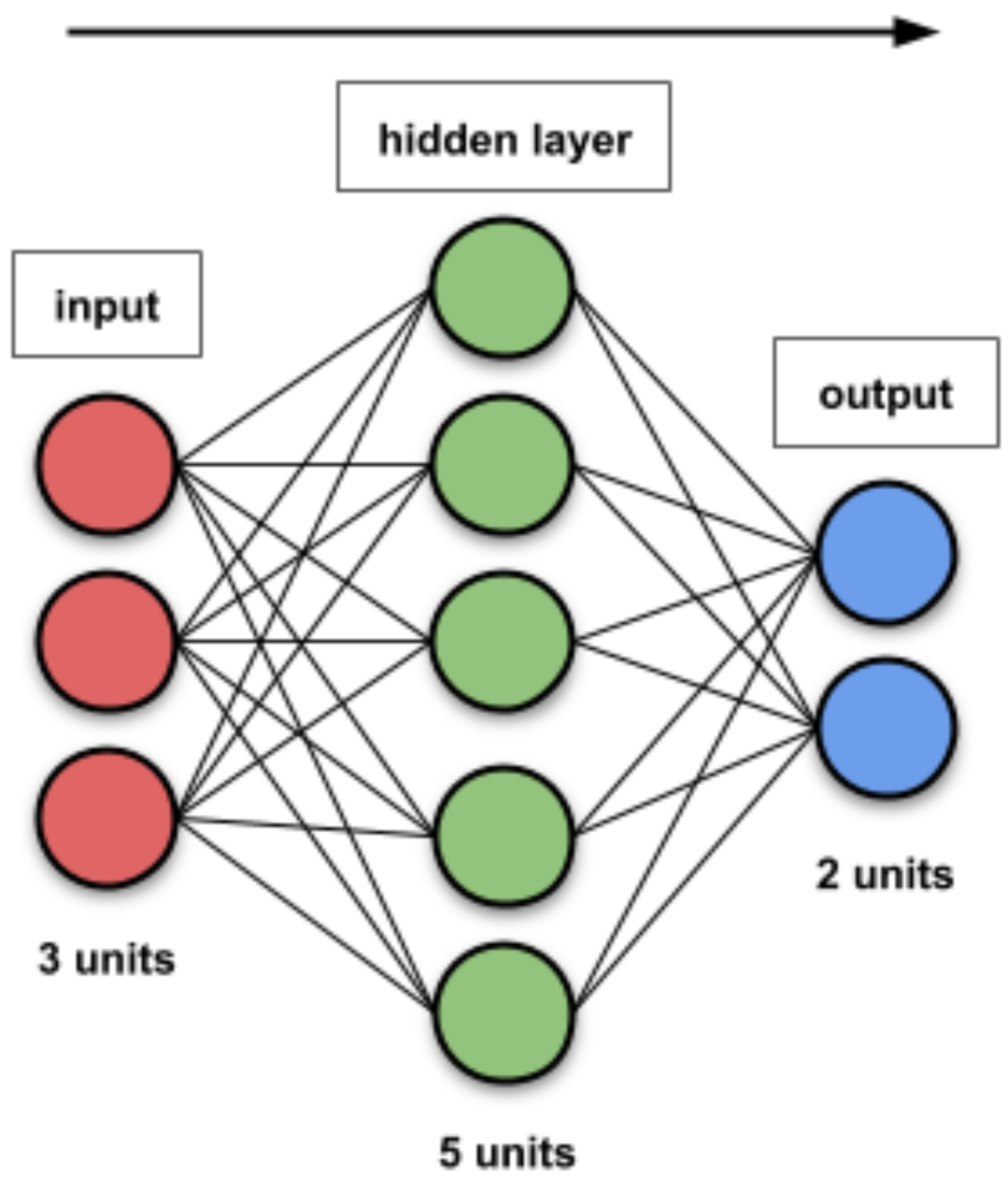


Image Source: towards-data-science

Feed-Forward Neural Network



(3 x 5) + (5 x 2) + (5 + 2) = 17 parameters

weights biases

Image Source: towards-data-science

Feed-Forward Neural Network

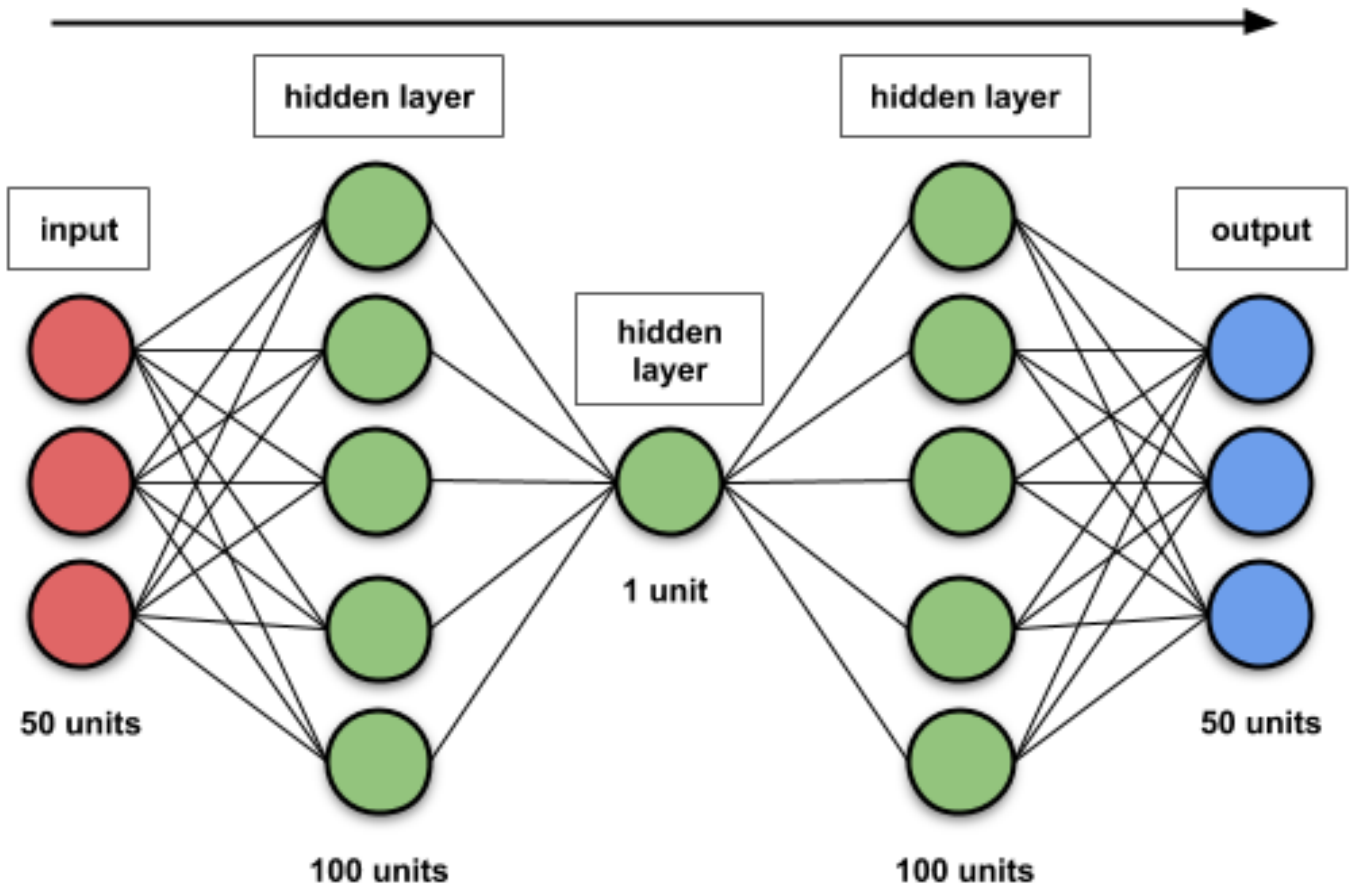


Image Source: towards-data-science

Overview on Convolutional Neural Networks (CNNs)

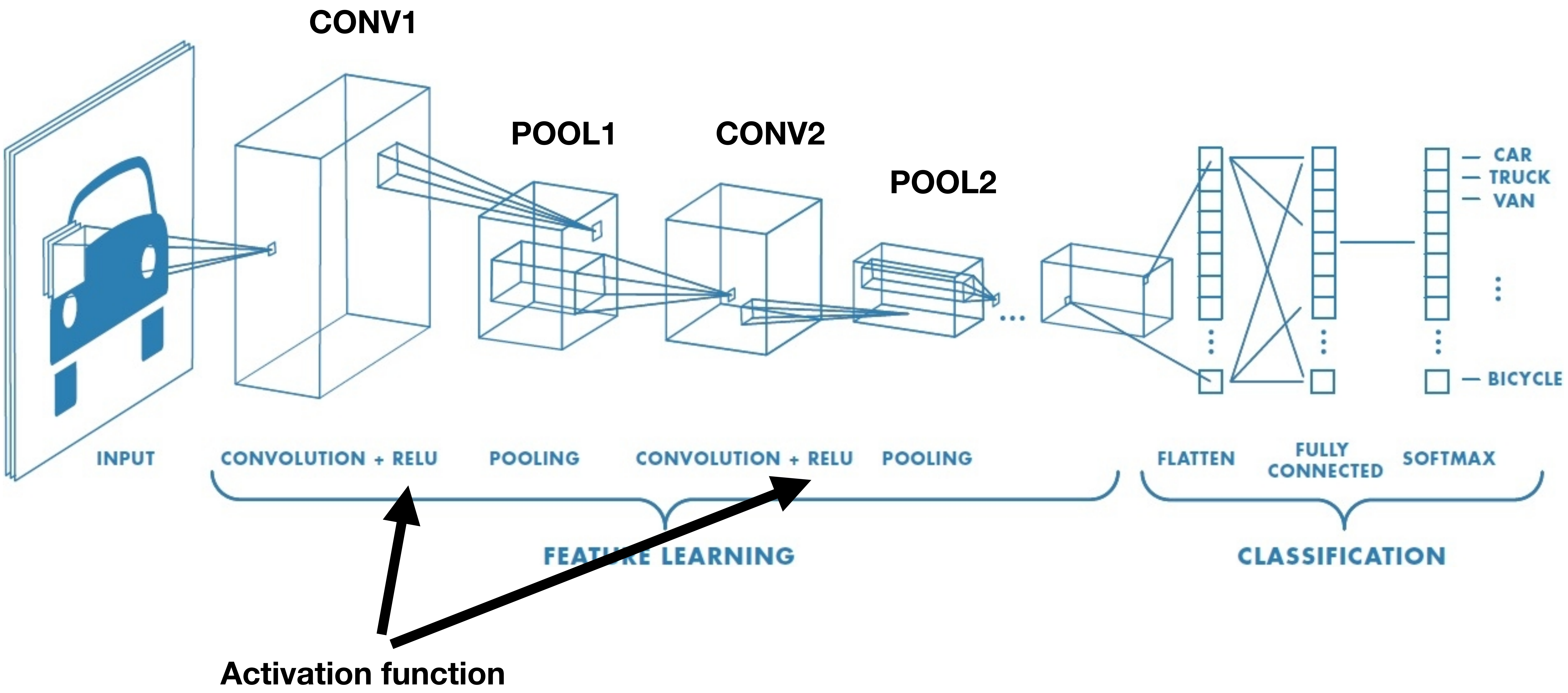
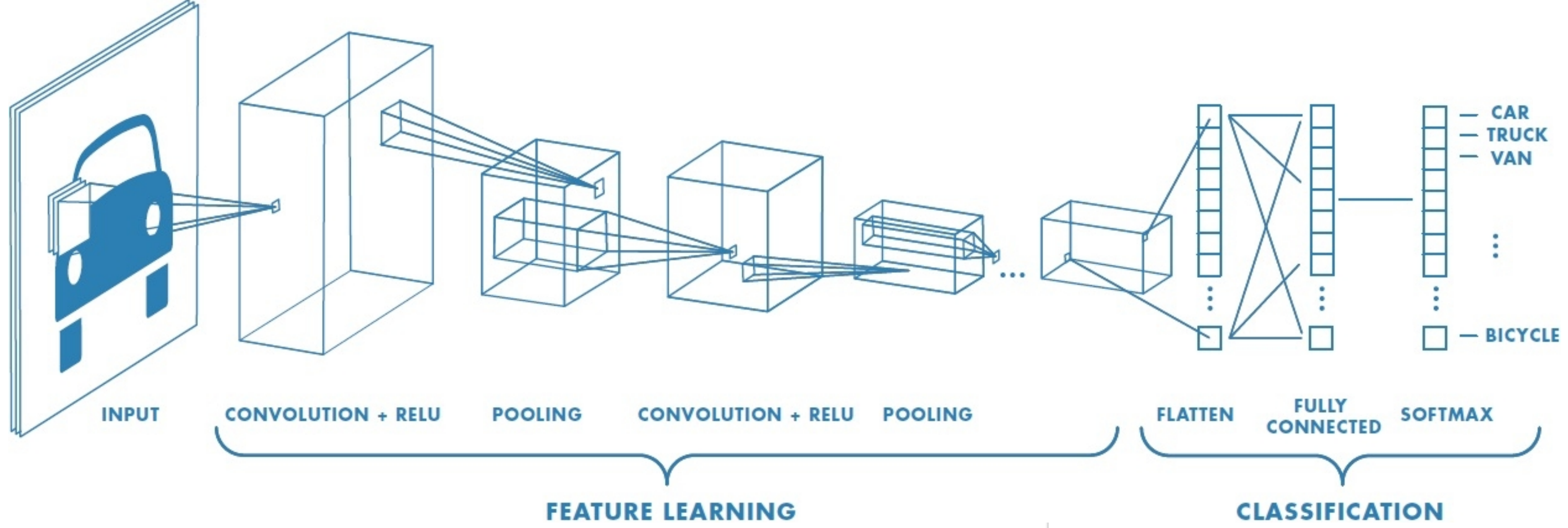


Image Courtesy: Mathworks (online tutorial)

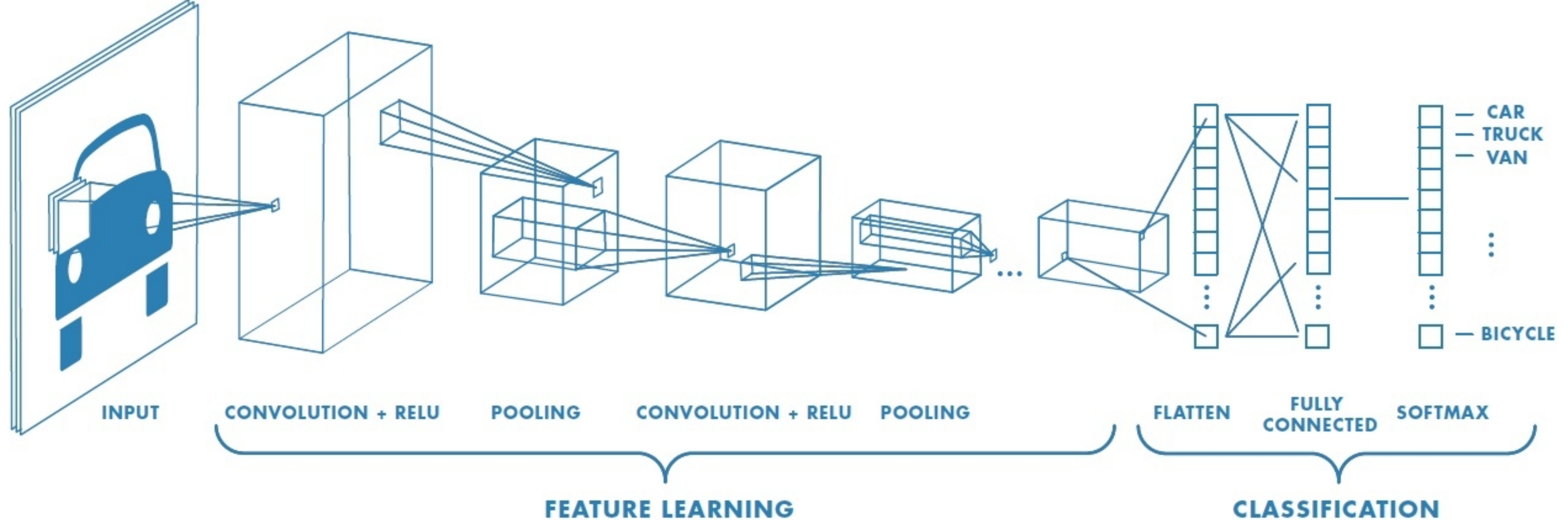


Neural network example

	Activation shape	Activation Size	# parameters
Input:	(32,32,3)		
CONV1 (f=5, s=1)	(28,28,8)		
POOL1	(14,14,8)		
CONV2 (f=5, s=1)	(10,10,16)		
POOL2	(5,5,16)		
FC3	(120,1)		
FC4	(84,1)		
Softmax	(10,1)		

https://www.coursera.org/learn/convolutional-neural-networks/lecture/uRYL1/cnn-example

Andrew Ng

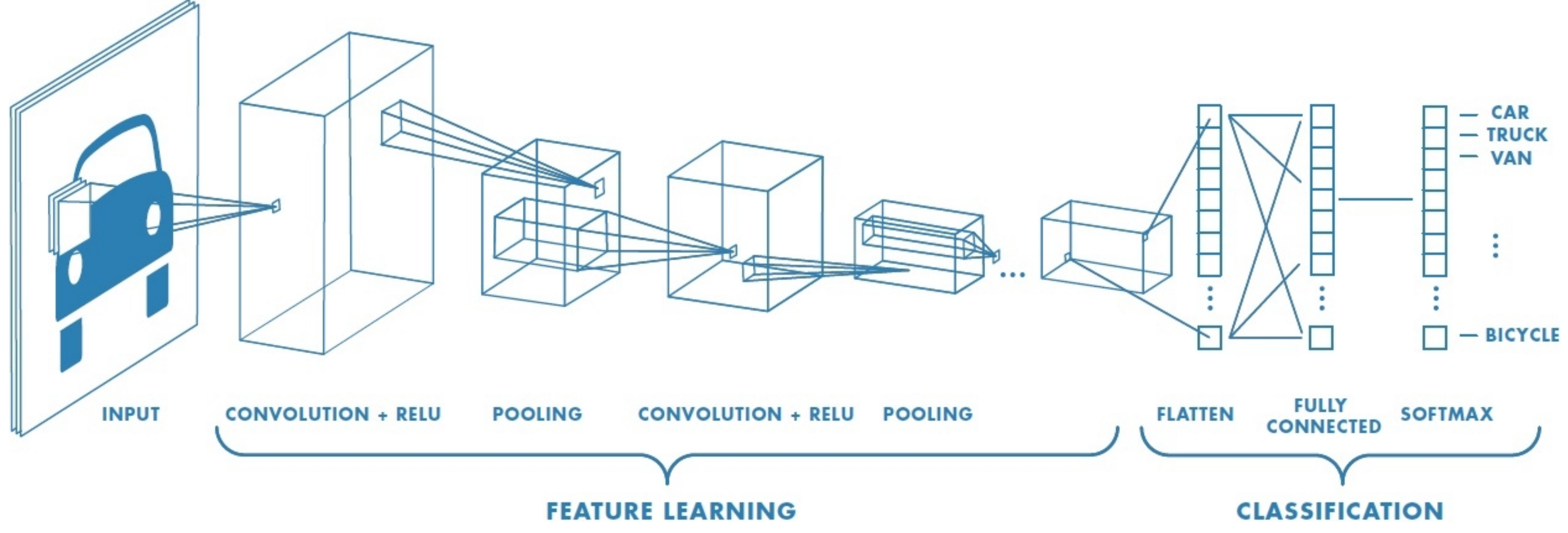


	Activation size	# parameters
Input layer:		
Conv1 layer:		
Pool1 layer:		
Conv2 layer:		
Pool2 layer:		
FC3 layer:		
FC4 layer:		
Softmax layer:		

Neural network example

	Activation shape	Activation Size	# parameters
Input:	(32,32,3)		
CONV1 (f=5, s=1)	(28,28,8)		
POOL1	(14,14,8)		
CONV2 (f=5, s=1)	(10,10,16)		
POOL2	(5,5,16)		
FC3	(120,1)		
FC4	(84,1)		
Softmax	(10,1)		

Andrew Ng

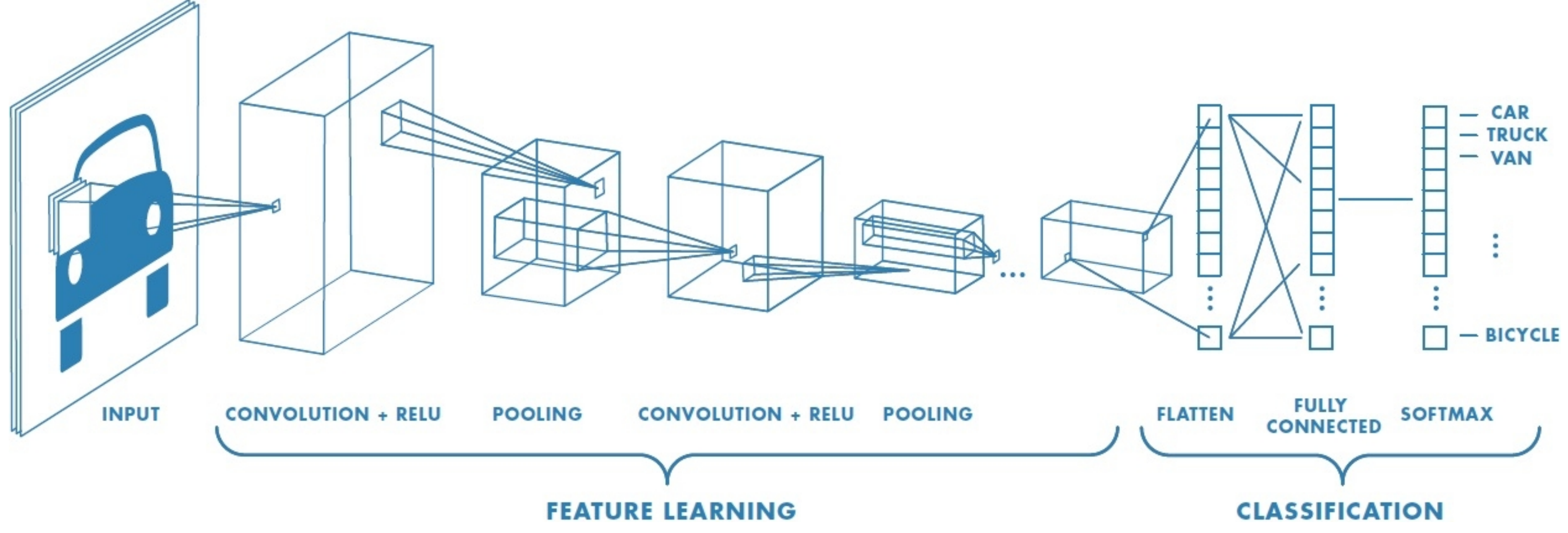


Neural network example

	Activation shape	Activation Size	# parameters
Input:	(32,32,3)		
CONV1 (f=5, s=1)	(28,28,8)		
POOL1	(14,14,8)		
CONV2 (f=5, s=1)	(10,10,16)		
POOL2	(5,5,16)		
FC3	(120,1)		
FC4	(84,1)		
Softmax	(10,1)		

Andrew Ng

	Activation size	# parameters
Input layer:	32*32*3 = 3072	
Conv1 layer:		
Pool1 layer:		
Conv2 layer:		
Pool2 layer:		
FC3 layer:		
FC4 layer:		
Softmax layer:		

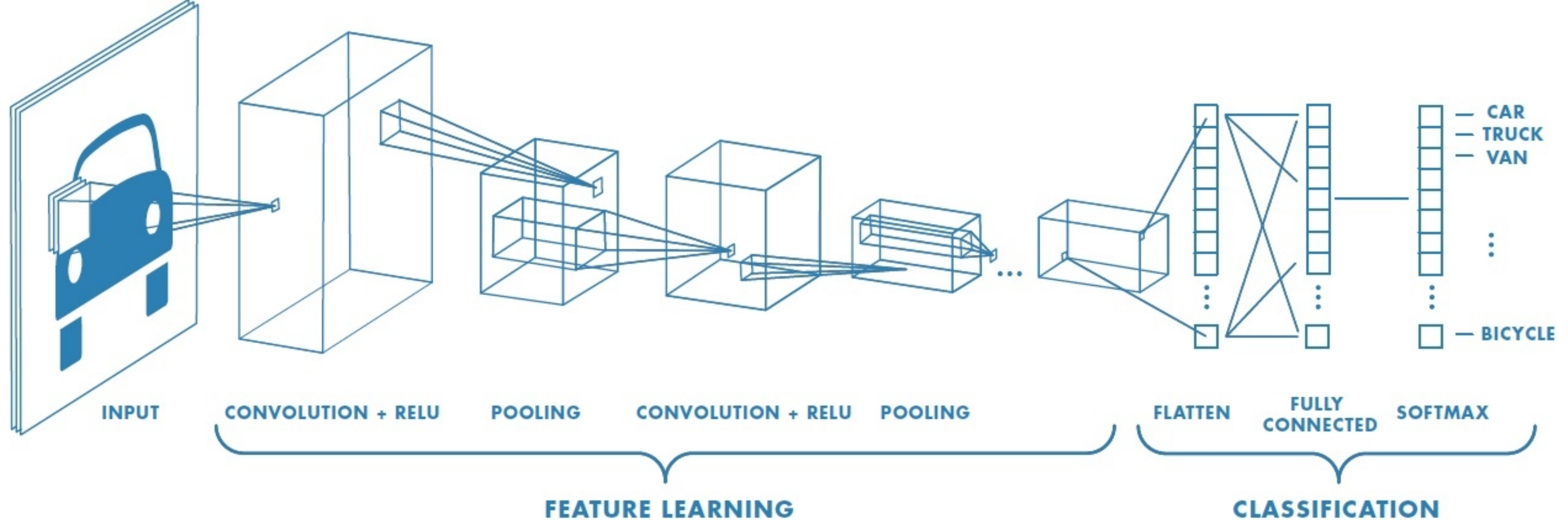


Neural network example

	Activation shape	Activation Size	# parameters
Input:	(32,32,3)		
CONV1 (f=5, s=1)	(28,28,8)		
POOL1	(14,14,8)		
CONV2 (f=5, s=1)	(10,10,16)		
POOL2	(5,5,16)		
FC3	(120,1)		
FC4	(84,1)		
Softmax	(10,1)		

Andrew Ng

	Activation size	# parameters
Input layer:	32*32*3 = 3072	
Conv1 layer:	28*28*8 = 6272	
Pool1 layer:	14*14*8 = 1568	
Conv2 layer:	10*10*16 = 1600	
Pool2 layer:	5*5*16 = 400	
FC3 layer:	120*1 = 120	
FC4 layer:	84*1 = 84	
Softmax layer:	10*1 = 10	

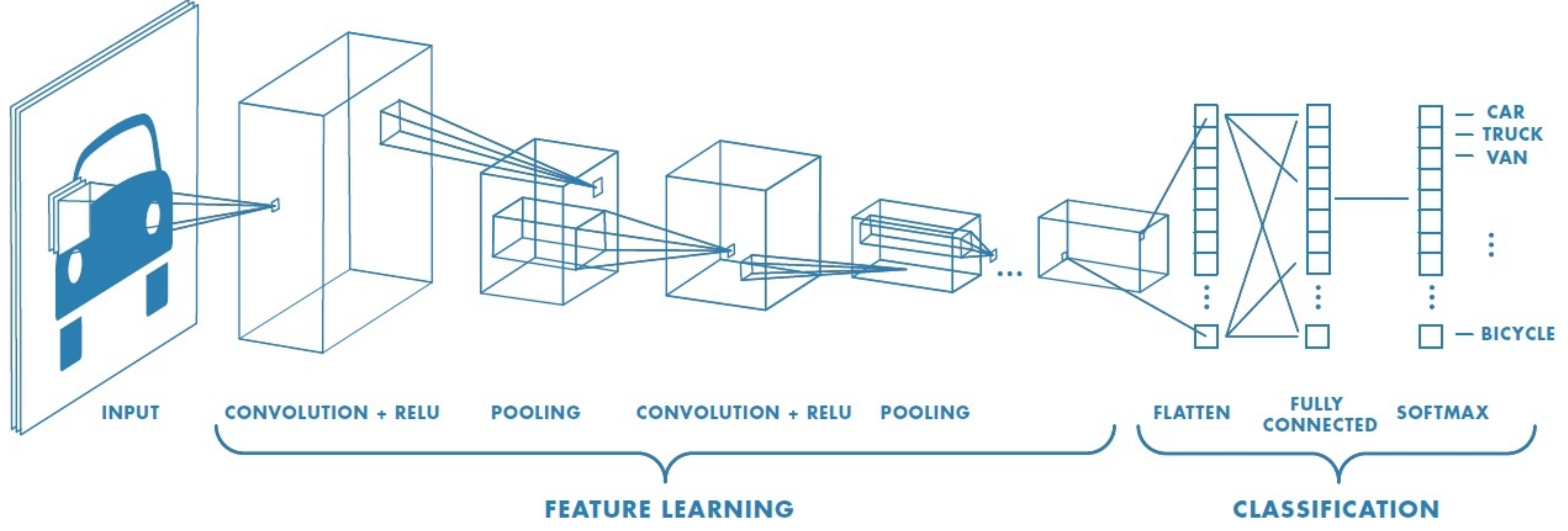


Neural network example

	Activation shape	Activation Size	# parameters
Input:	(32,32,3)		
CONV1 (f=5, s=1)	(28,28,8)		
POOL1	(14,14,8)		
CONV2 (f=5, s=1)	(10,10,16)		
POOL2	(5,5,16)		
FC3	(120,1)		
FC4	(84,1)		
Softmax	(10,1)		

Andrew Ng

	Activation size	# parameters
Input layer:	32*32*3 = 3072	0
Conv1 layer:	28*28*8 = 6272	
Pool1 layer:	14*14*8 = 1568	
Conv2 layer:	10*10*16 = 1600	
Pool2 layer:	5*5*16 = 400	
FC3 layer:	120*1 = 120	
FC4 layer:	84*1 = 84	
Softmax layer:	10*1 = 10	

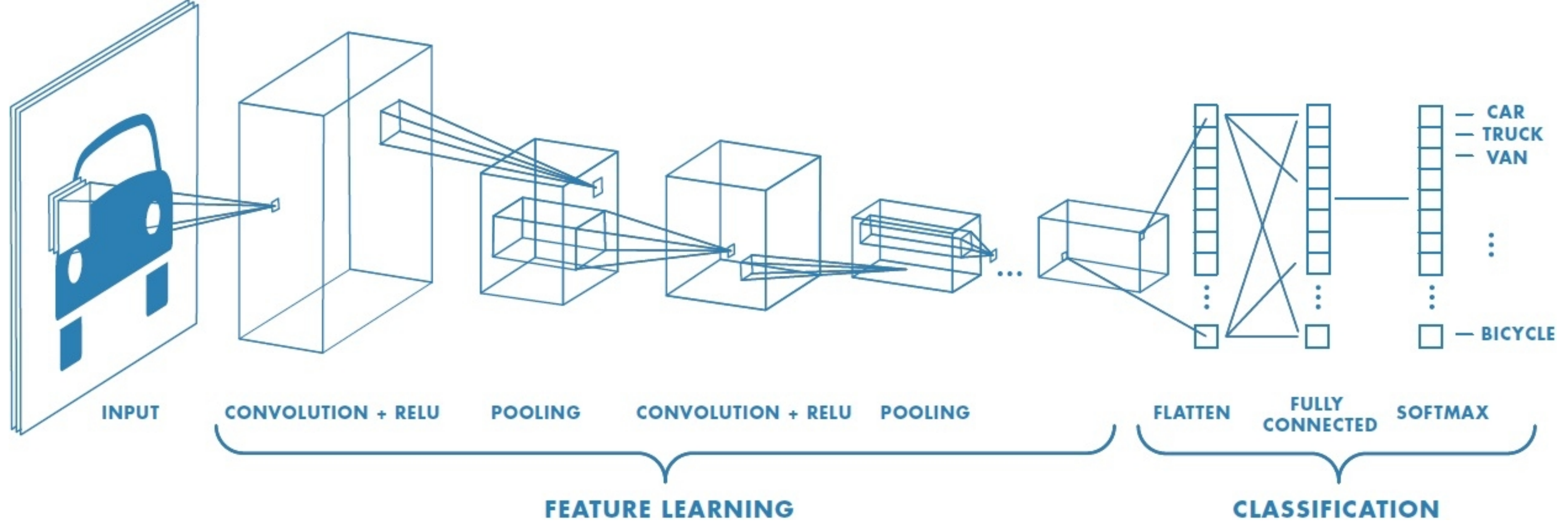


Neural network example

	Activation shape	Activation Size	# parameters
Input:	(32,32,3)		
CONV1 (f=5, s=1)	(28,28,8)		
POOL1	(14,14,8)		
CONV2 (f=5, s=1)	(10,10,16)		
POOL2	(5,5,16)		
FC3	(120,1)		
FC4	(84,1)		
Softmax	(10,1)		

Andrew Ng

	Activation size	# parameters
Input layer:	32*32*3 = 3072	0
Conv1 layer:	28*28*8 = 6272	(fwidth*fheight+1)*numfilters
Pool1 layer:	14*14*8 = 1568	
Conv2 layer:	10*10*16 = 1600	
Pool2 layer:	5*5*16 = 400	
FC3 layer:	120*1 = 120	
FC4 layer:	84*1 = 84	
Softmax layer:	10*1 = 10	

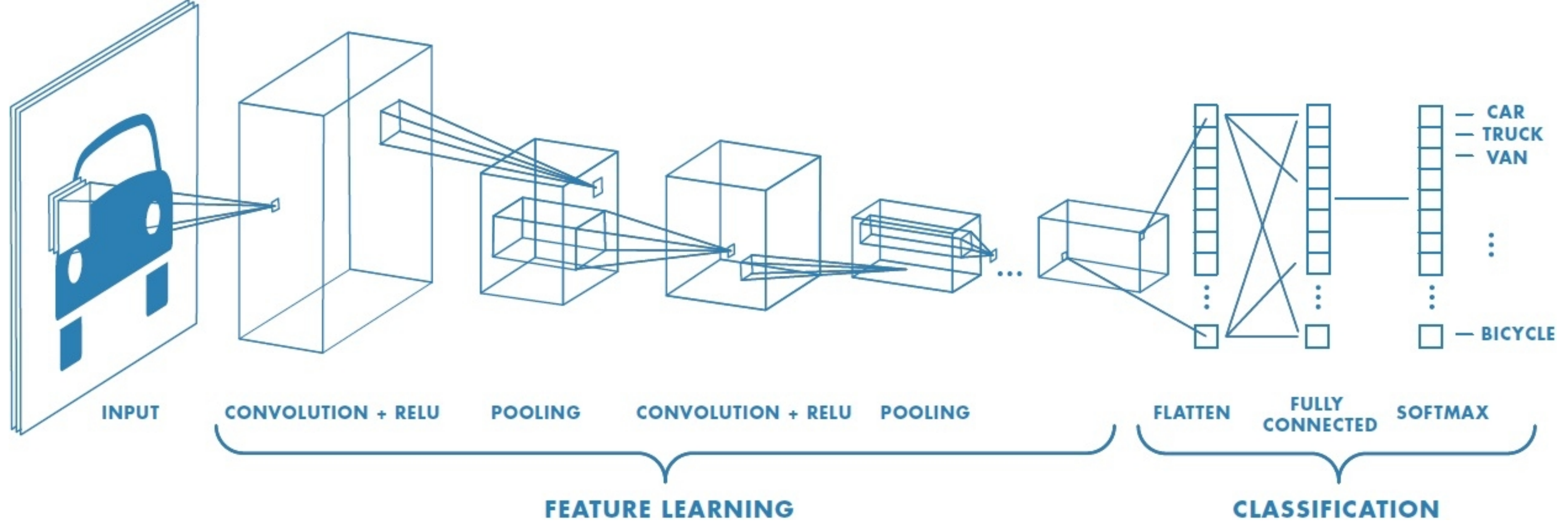


Neural network example

	Activation shape	Activation Size	# parameters
Input:	(32,32,3)		
CONV1 (f=5, s=1)	(28,28,8)		
POOL1	(14,14,8)		
CONV2 (f=5, s=1)	(10,10,16)		
POOL2	(5,5,16)		
FC3	(120,1)		
FC4	(84,1)		
Softmax	(10,1)		

Andrew Ng

	Activation size	# parameters
Input layer:	32*32*3 = 3072	0
Conv1 layer:	28*28*8 = 6272	(5*5+1)*8 = 208
Pool1 layer:	14*14*8 = 1568	0
Conv2 layer:	10*10*16 = 1600	(5*5+1)*16 = 416
Pool2 layer:	5*5*16 = 400	0
FC3 layer:	120*1 = 120	currentLayer*PrevLayer+1
FC4 layer:	84*1 = 84	
Softmax layer:	10*1 = 10	

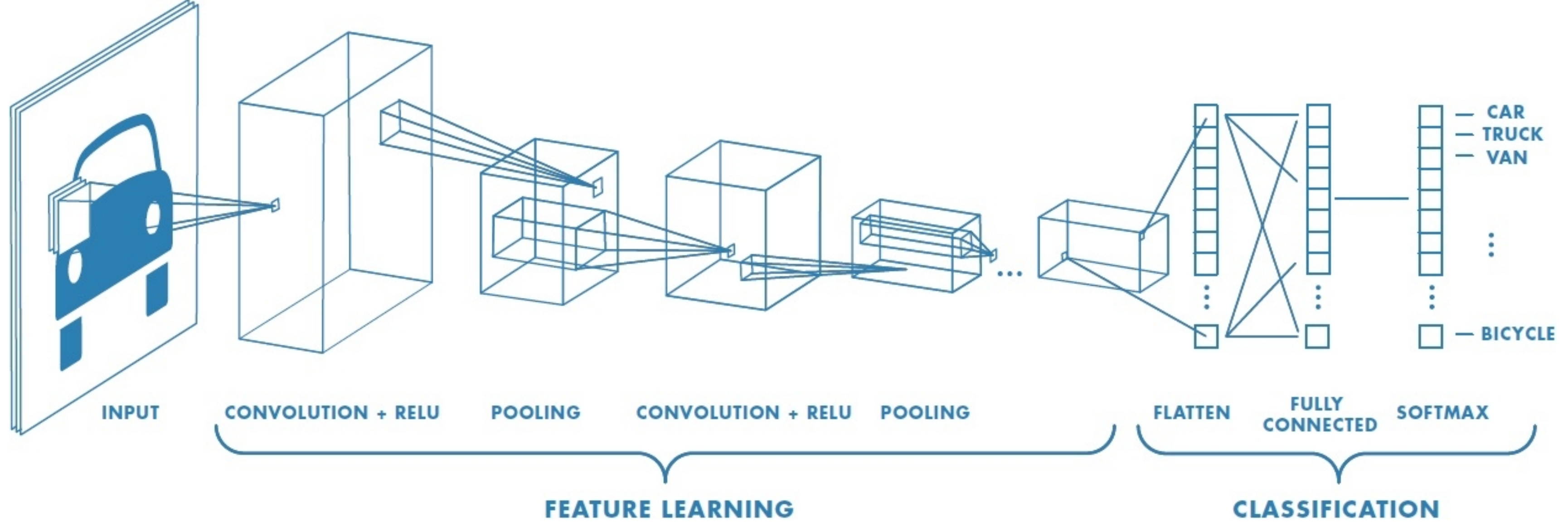


Neural network example

	Activation shape	Activation Size	# parameters
Input:	(32,32,3)		
CONV1 (f=5, s=1)	(28,28,8)		
POOL1	(14,14,8)		
CONV2 (f=5, s=1)	(10,10,16)		
POOL2	(5,5,16)		
FC3	(120,1)		
FC4	(84,1)		
Softmax	(10,1)		

Andrew Ng

	Activation size	# parameters
Input layer:	32*32*3 = 3072	0
Conv1 layer:	28*28*8 = 6272	(5*5+1)*8 = 208
Pool1 layer:	14*14*8 = 1568	0
Conv2 layer:	10*10*16 = 1600	(5*5+1)*16 = 416
Pool2 layer:	5*5*16 = 400	0
FC3 layer:	120*1 = 120	currentLayer*PrevLayer+1
FC4 layer:	84*1 = 84	
Softmax layer:	10*1 = 10	



Neural network example

	Activation shape	Activation Size	# parameters
Input:	(32,32,3)		
CONV1 (f=5, s=1)	(28,28,8)		
POOL1	(14,14,8)		
CONV2 (f=5, s=1)	(10,10,16)		
POOL2	(5,5,16)		
FC3	(120,1)		
FC4	(84,1)		
Softmax	(10,1)		

Andrew Ng

	Activation size	# parameters
Input layer:	32*32*3 = 3072	0
Conv1 layer:	28*28*8 = 6272	(5*5+1)*8 = 208
Pool1 layer:	14*14*8 = 1568	0
Conv2 layer:	10*10*16 = 1600	(5*5+1)*16 = 416
Pool2 layer:	5*5*16 = 400	0
FC3 layer:	120*1 = 120	120*400+1 = 48001
FC4 layer:	84*1 = 84	120*84+1 = 10081
Softmax layer:	10*1 = 10	10*84+1 = 841

Pixel-space
Kernel Predicting
Denoising

#Learnable
Parameters?

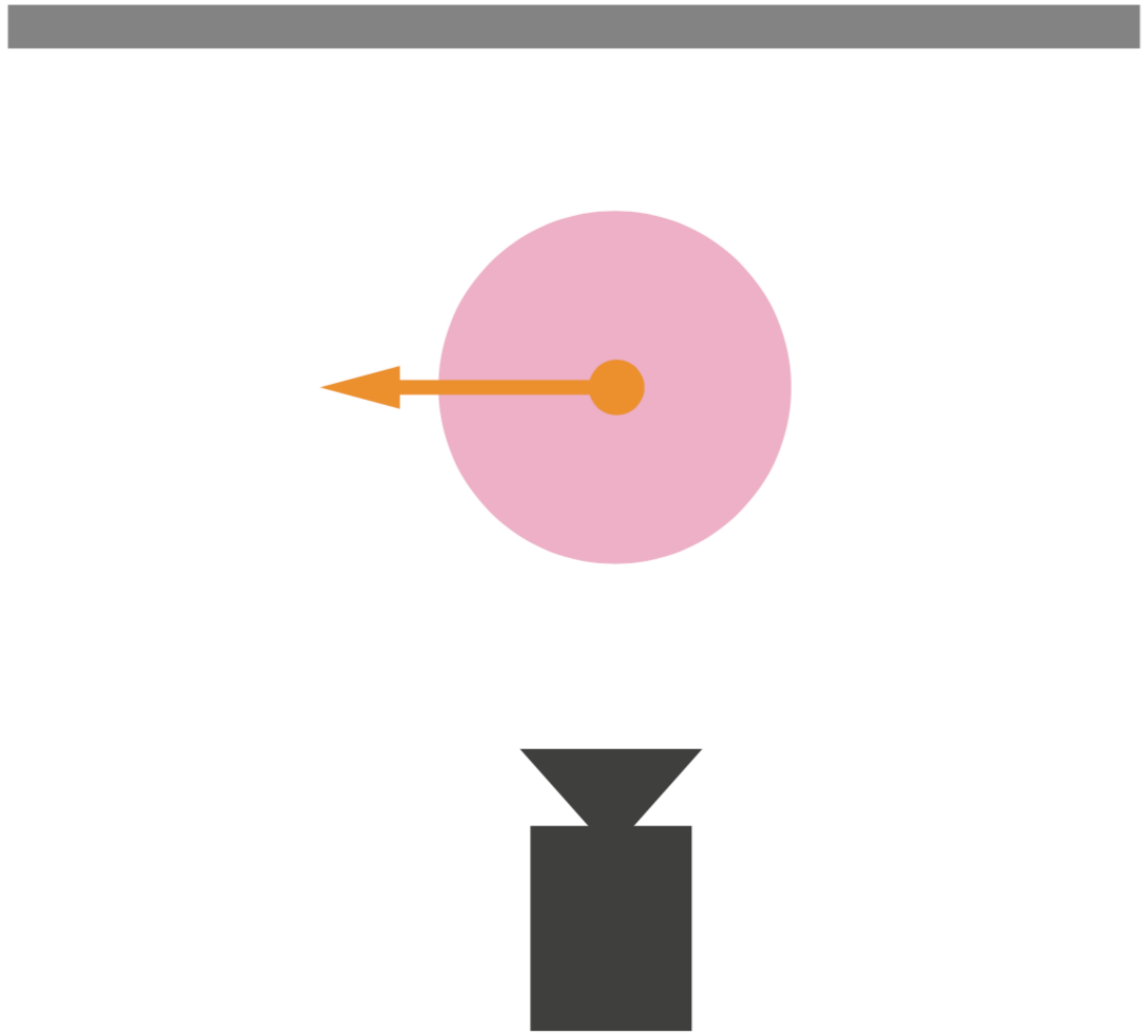
Sample-based
MC Denoising

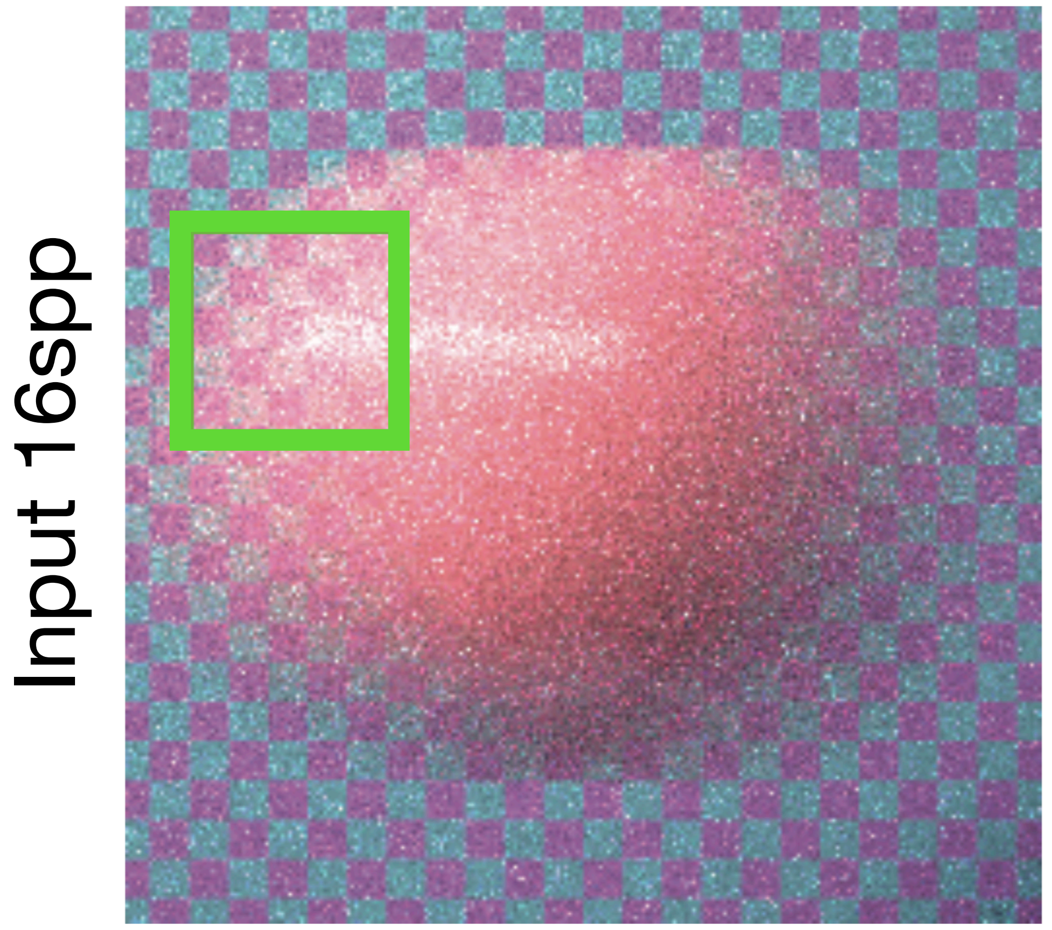
Sample-based Denoising Network

Michael Gharbi, Tzu-Mao Li, Miika Aittala, Jakko Lehtinen, Fredo Durand

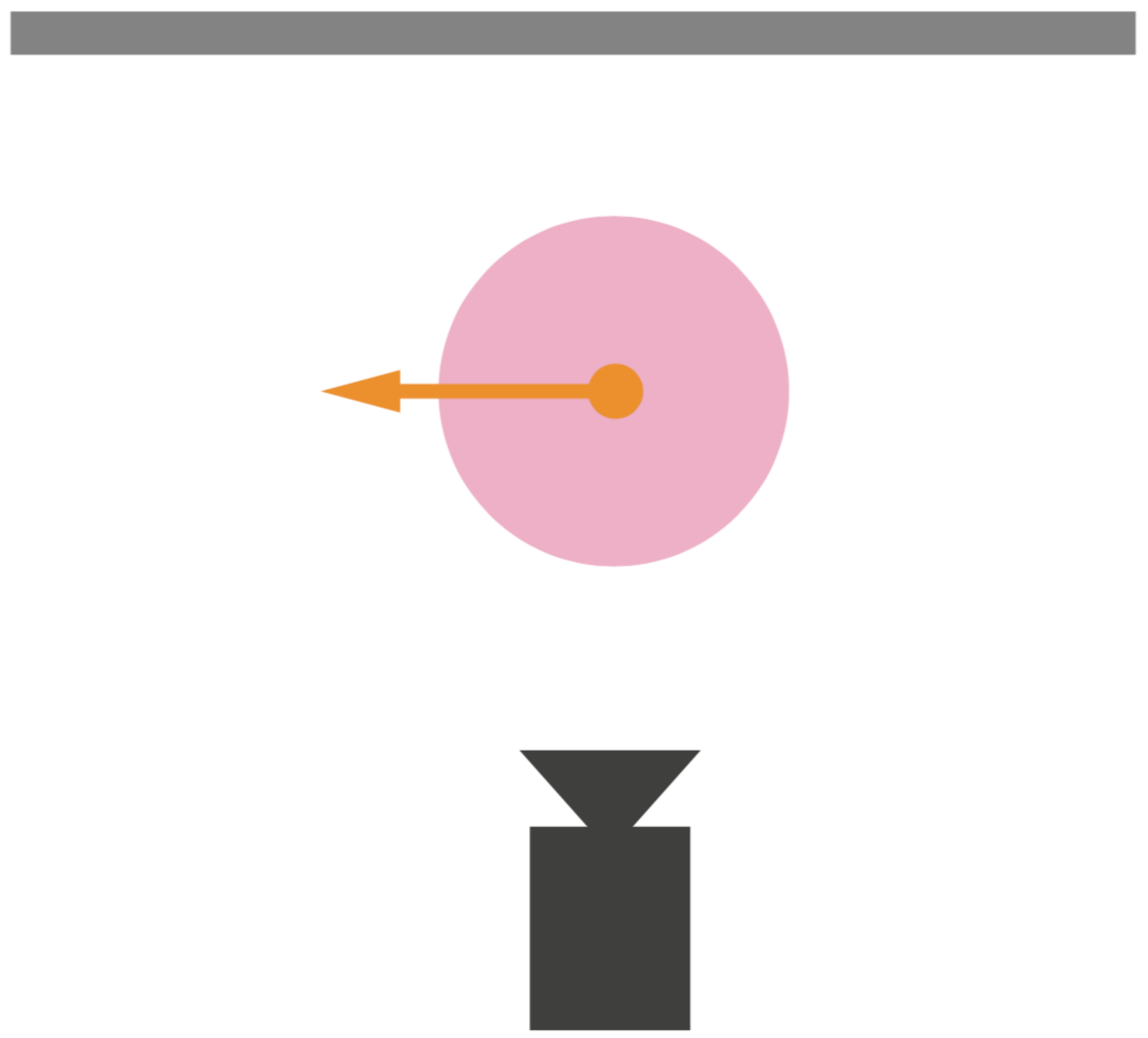
SIGGRAPH 2019

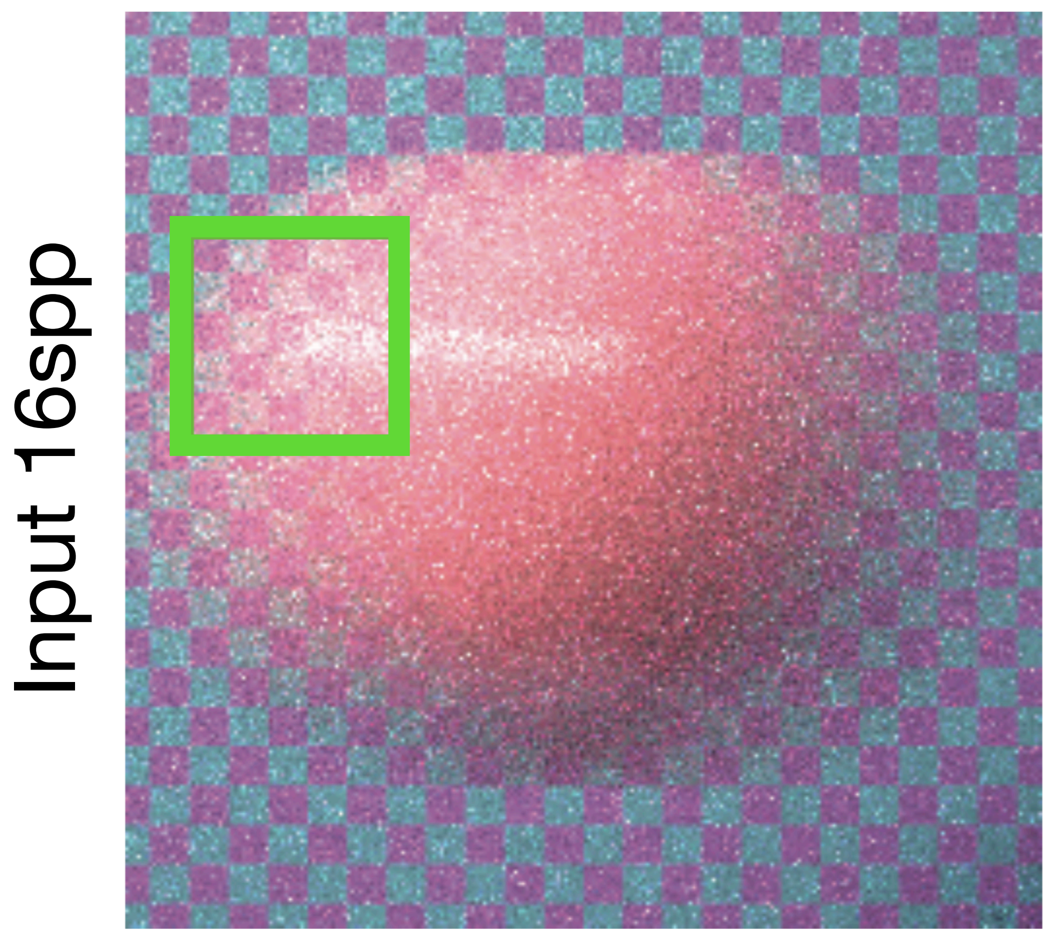
Multimodal distribution of sample features

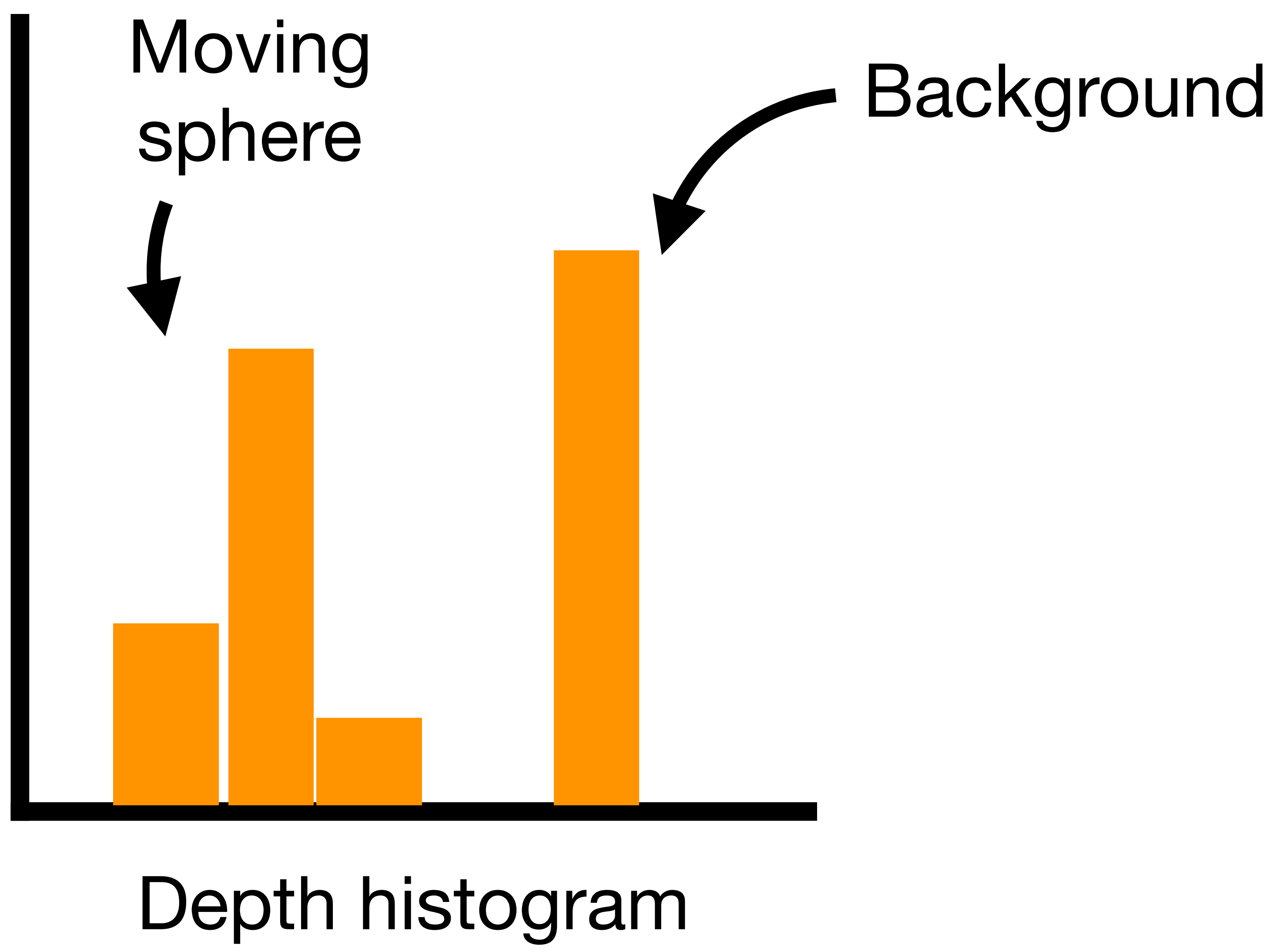




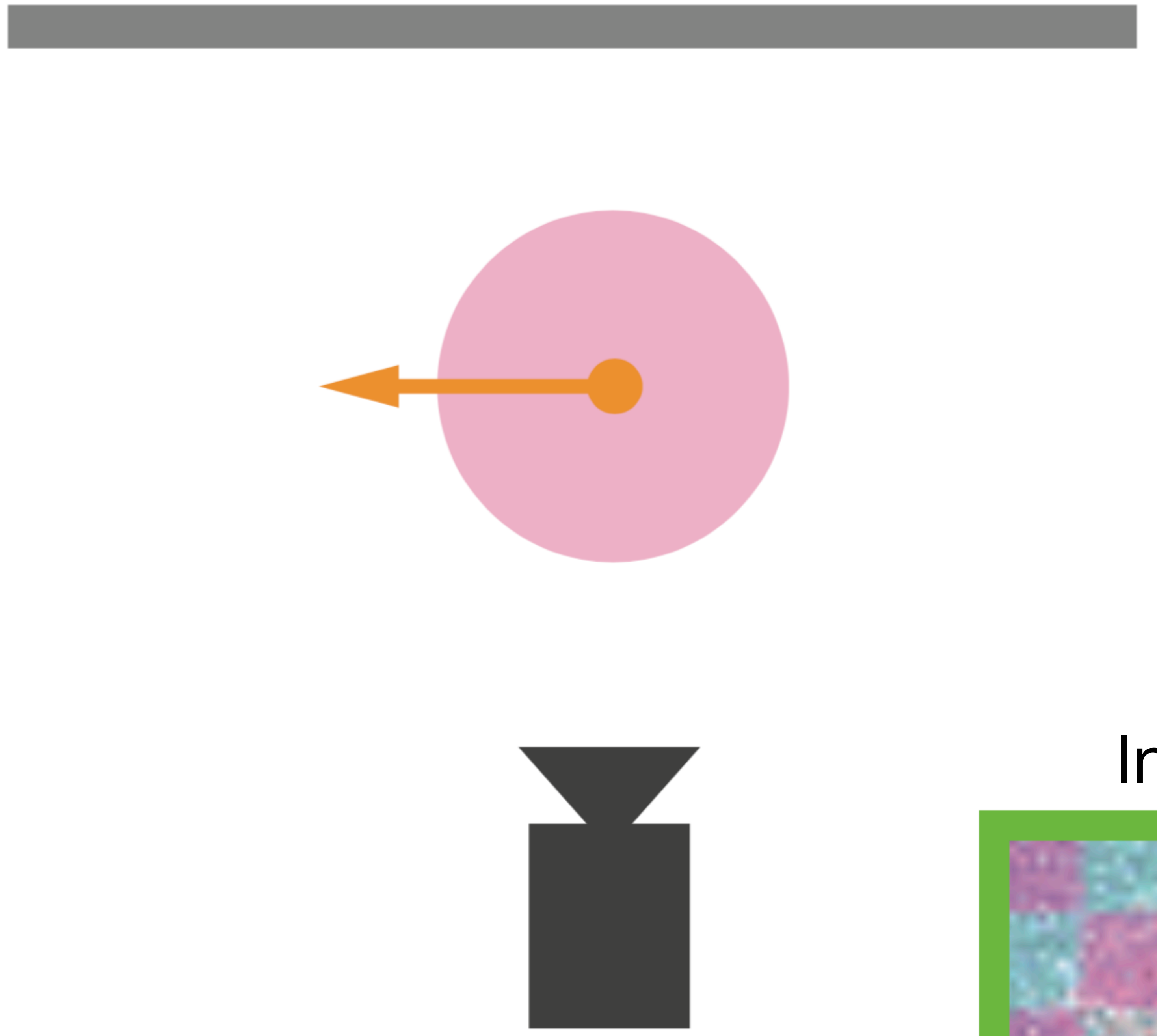
Multimodal distribution of sample features



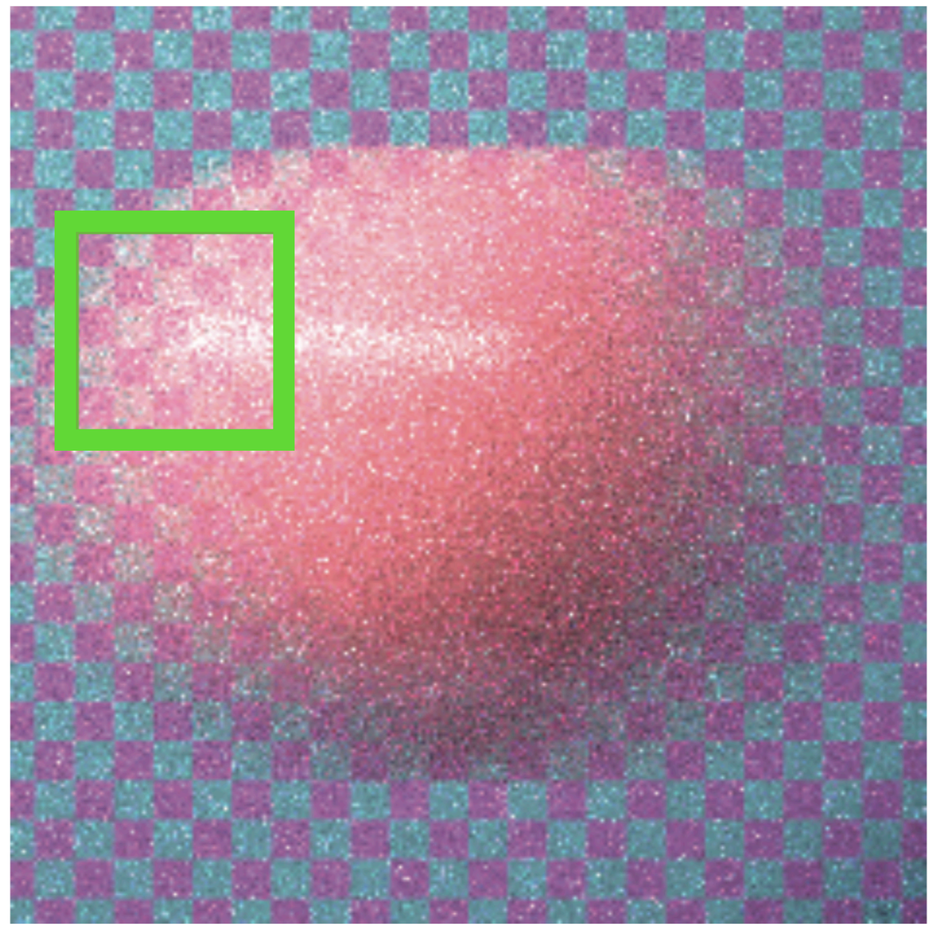


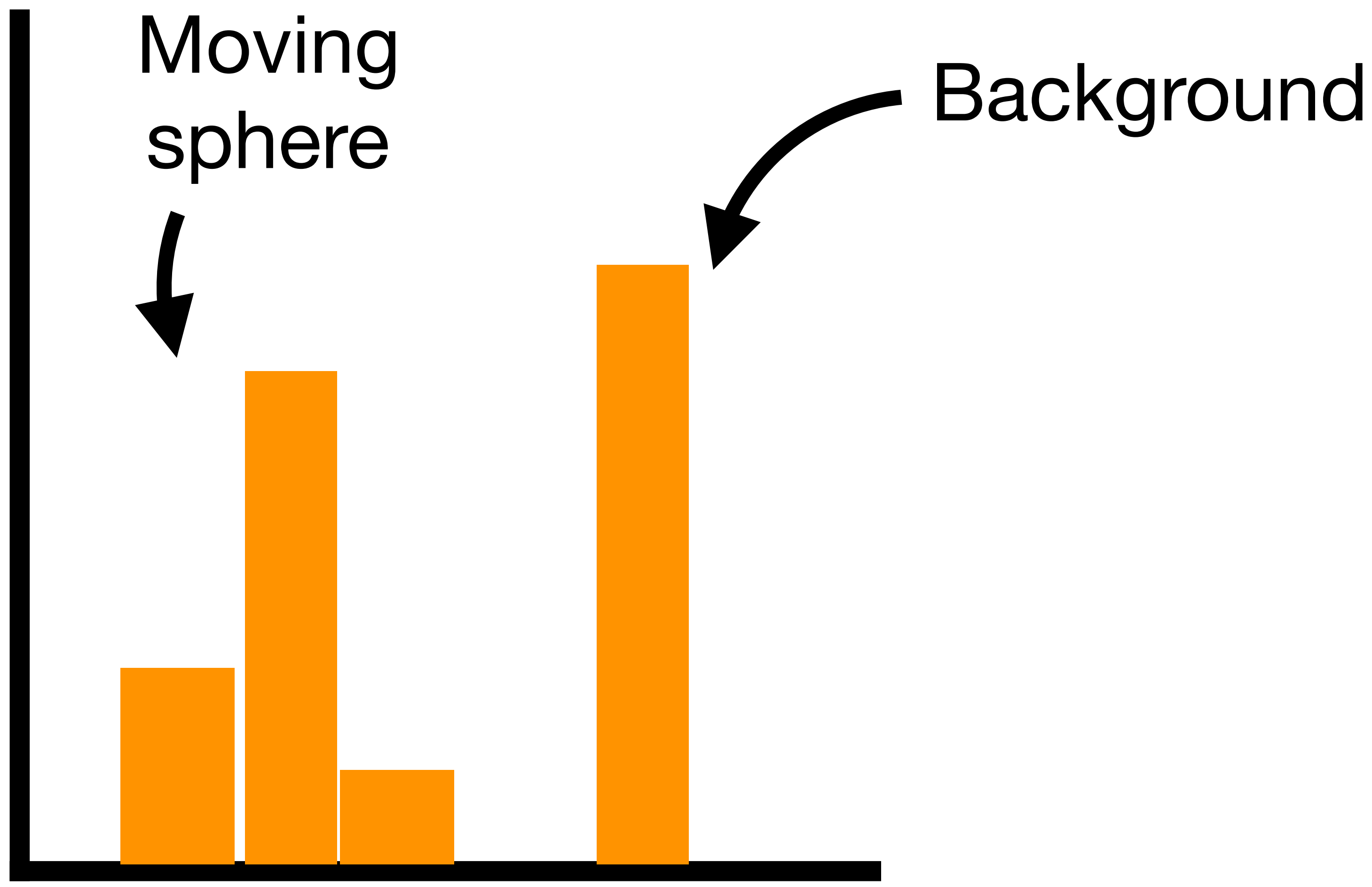


Multimodal distribution of sample features



Input 16spp

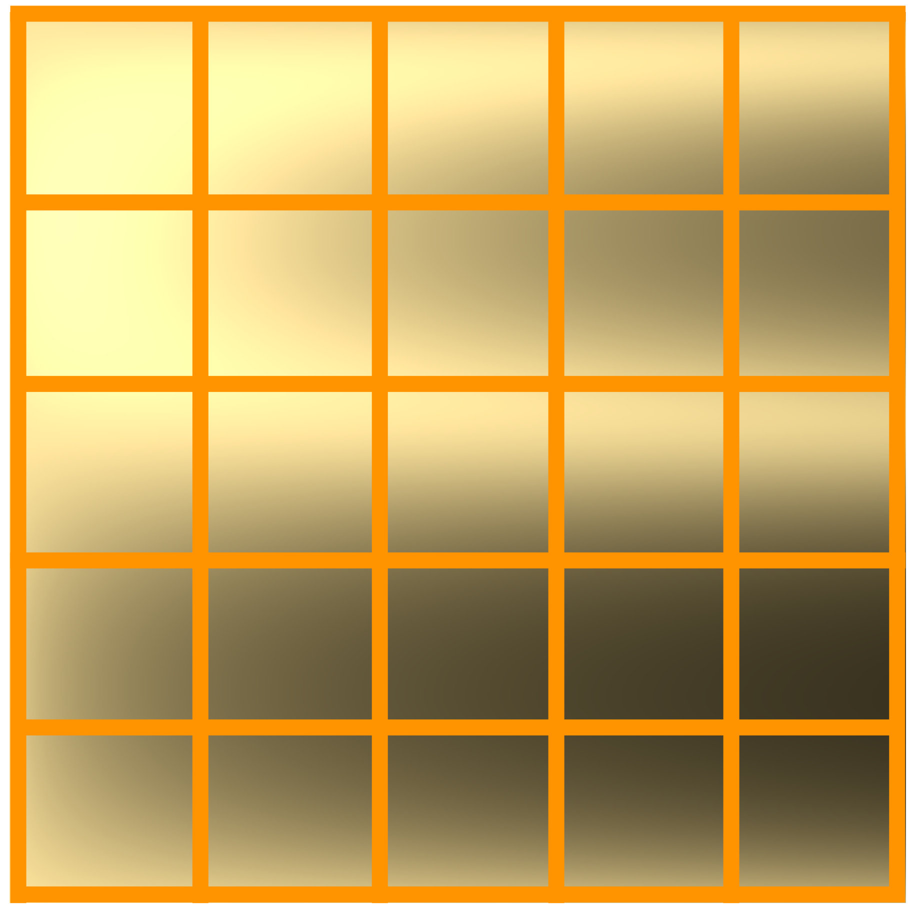




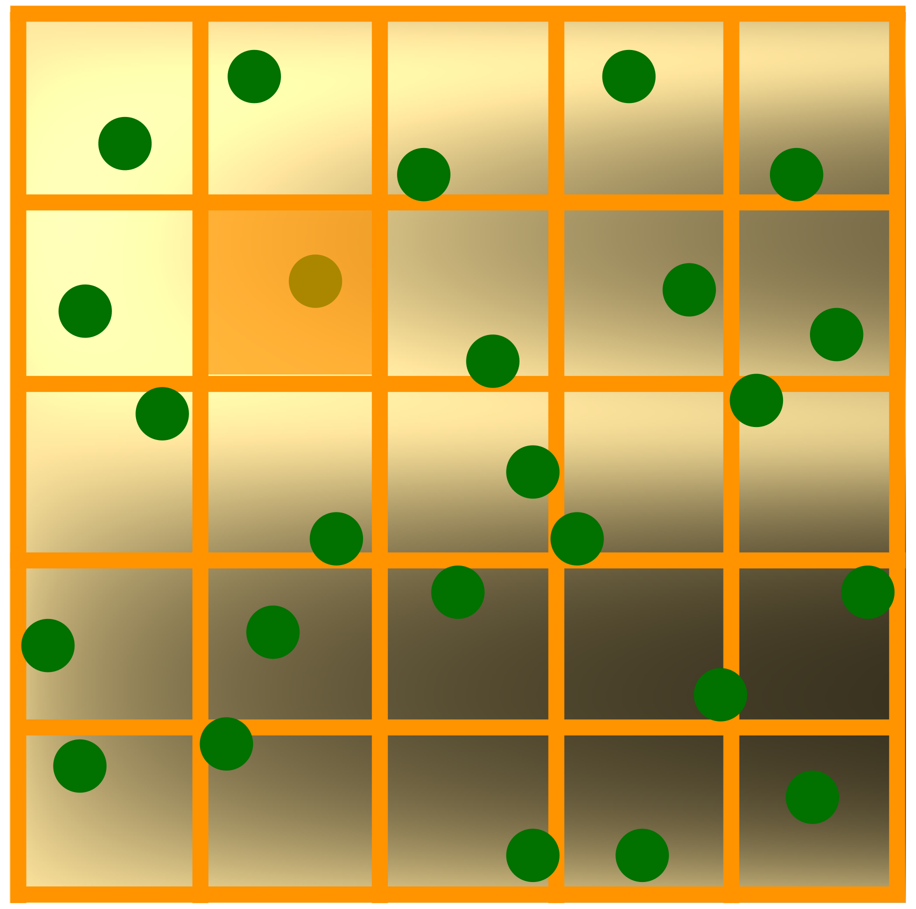
Depth histogram

Inset 16spp | Reference | Sen [2012] | Proposed

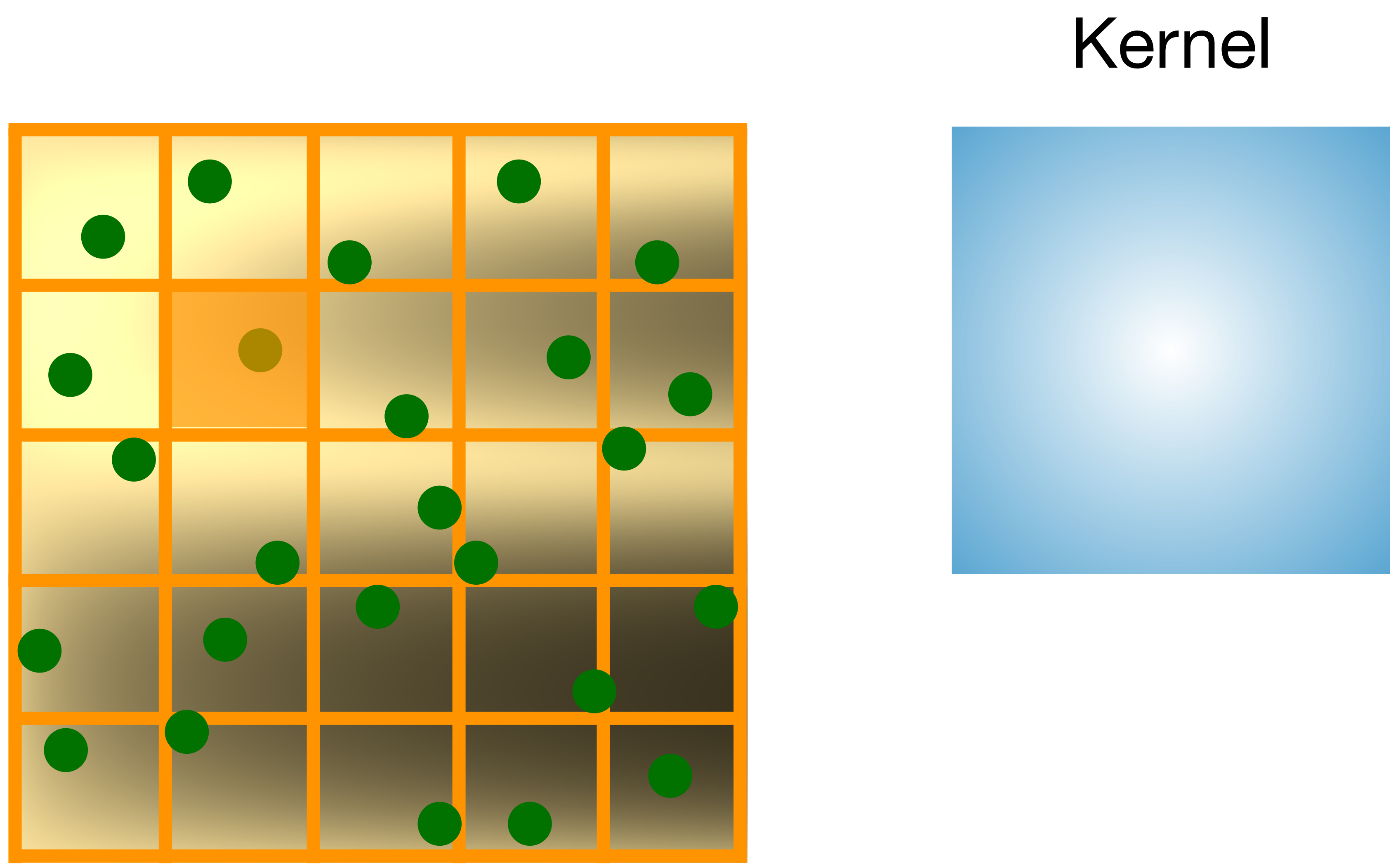
Reconstruction: Kernel Gather



Reconstruction: Kernel Gather

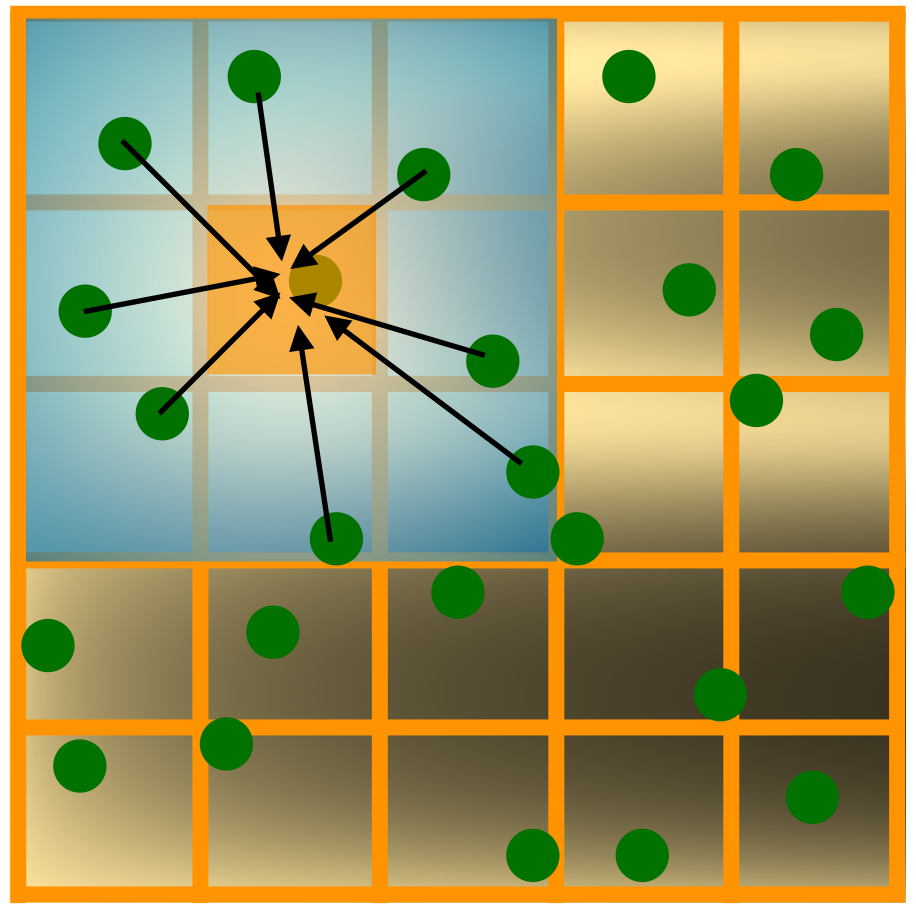


Reconstruction: Kernel Gather



Reconstruction: Kernel Gather

Kernel gather

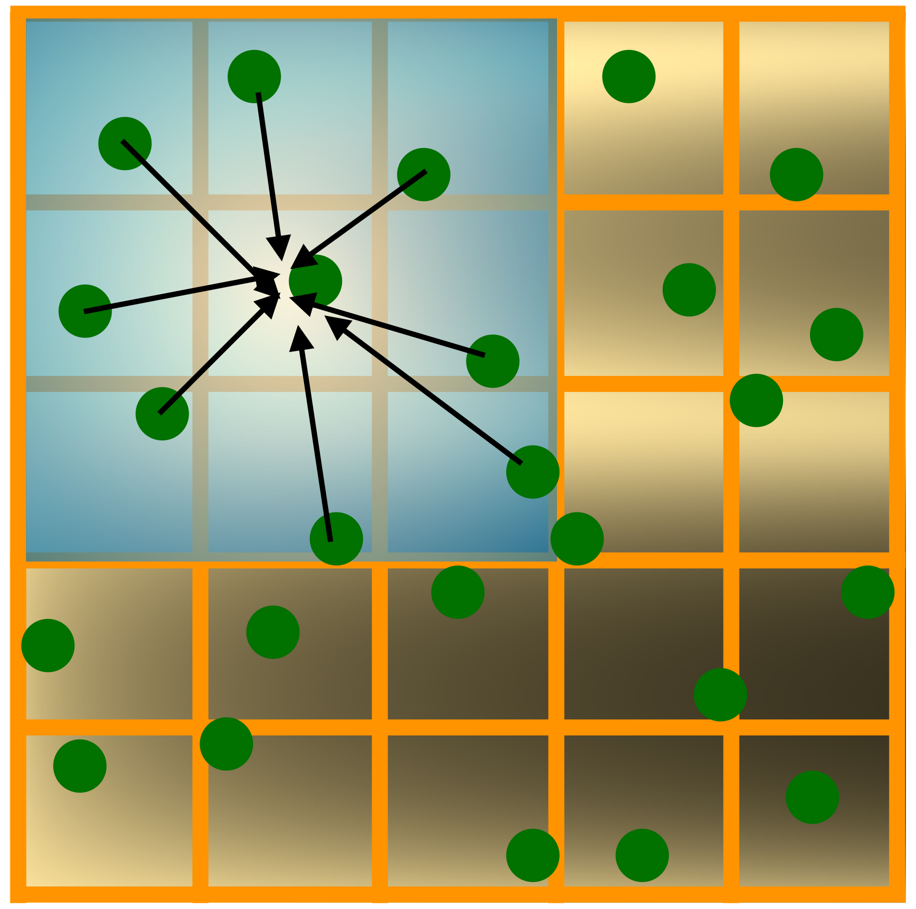


2D example

How should nearby samples influence me?

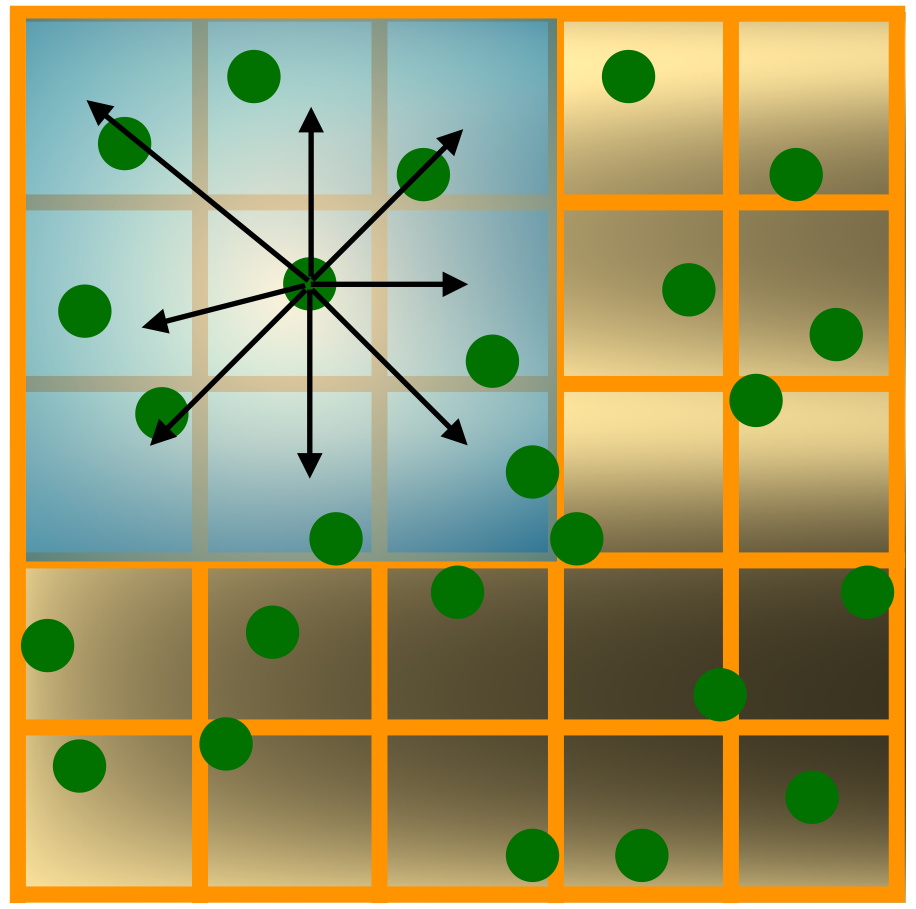
Reconstruction: Kernel Splatting

Kernel gather



2D example

Kernel Splatting

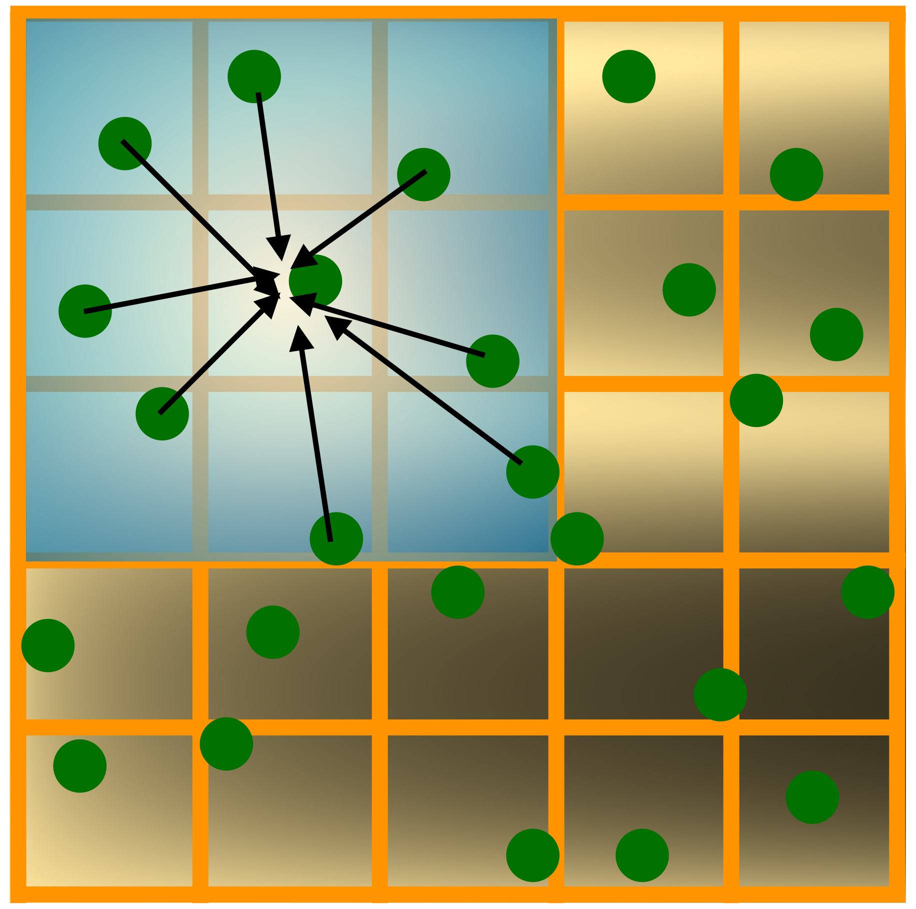


2D example

How should nearby samples influence me?

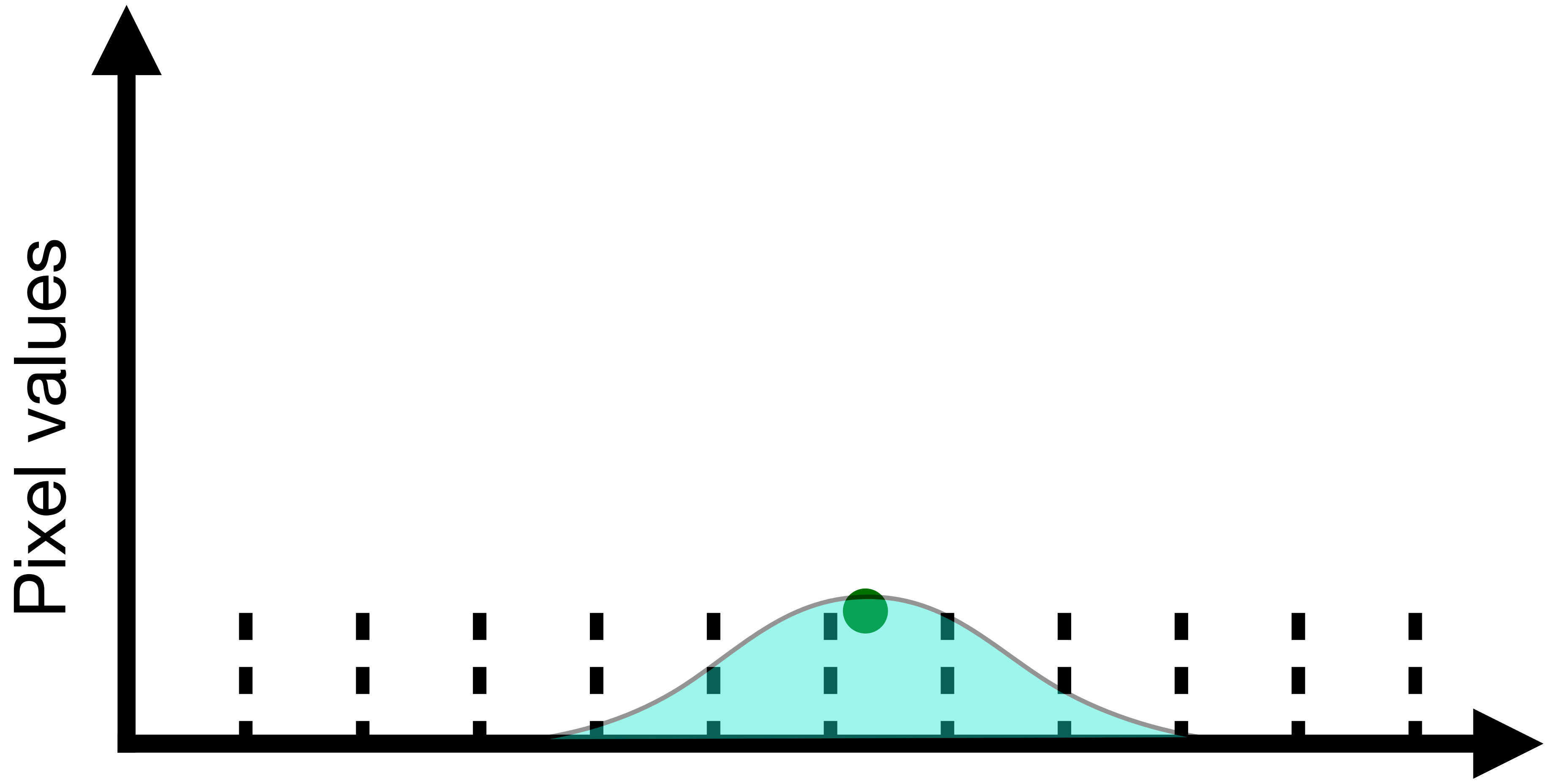
Reconstruction: Kernel Splatting

Kernel gather



2D example

Kernel Splatting

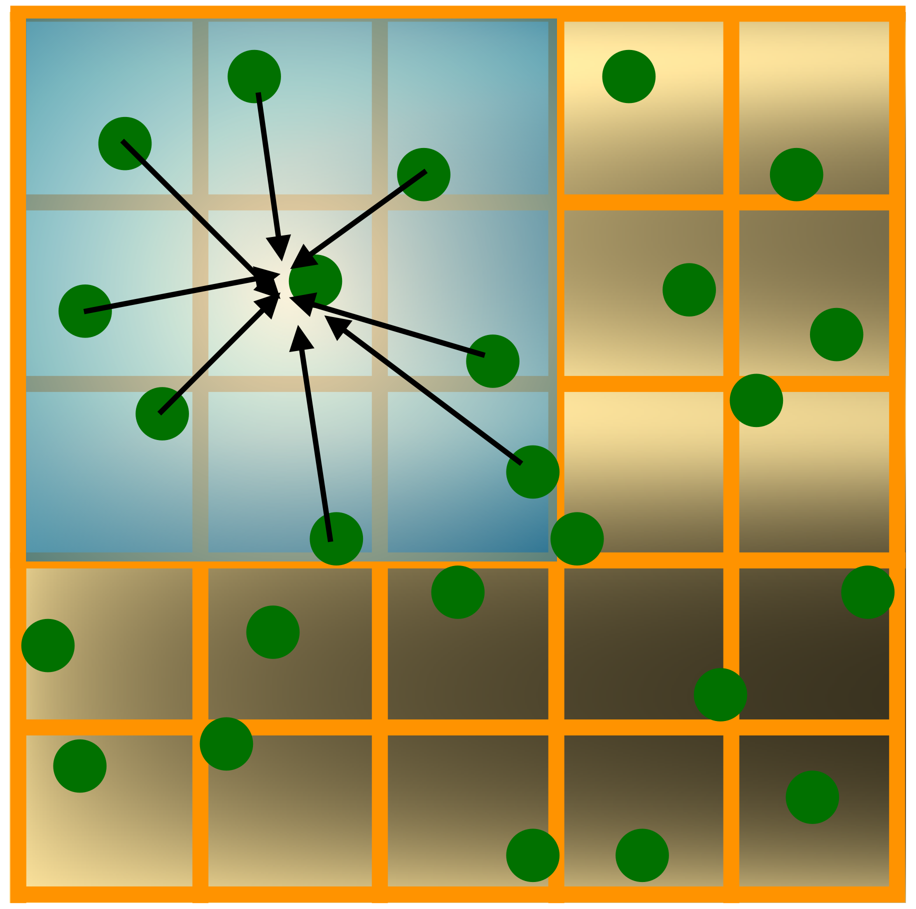


1D example

How should nearby samples influence me?

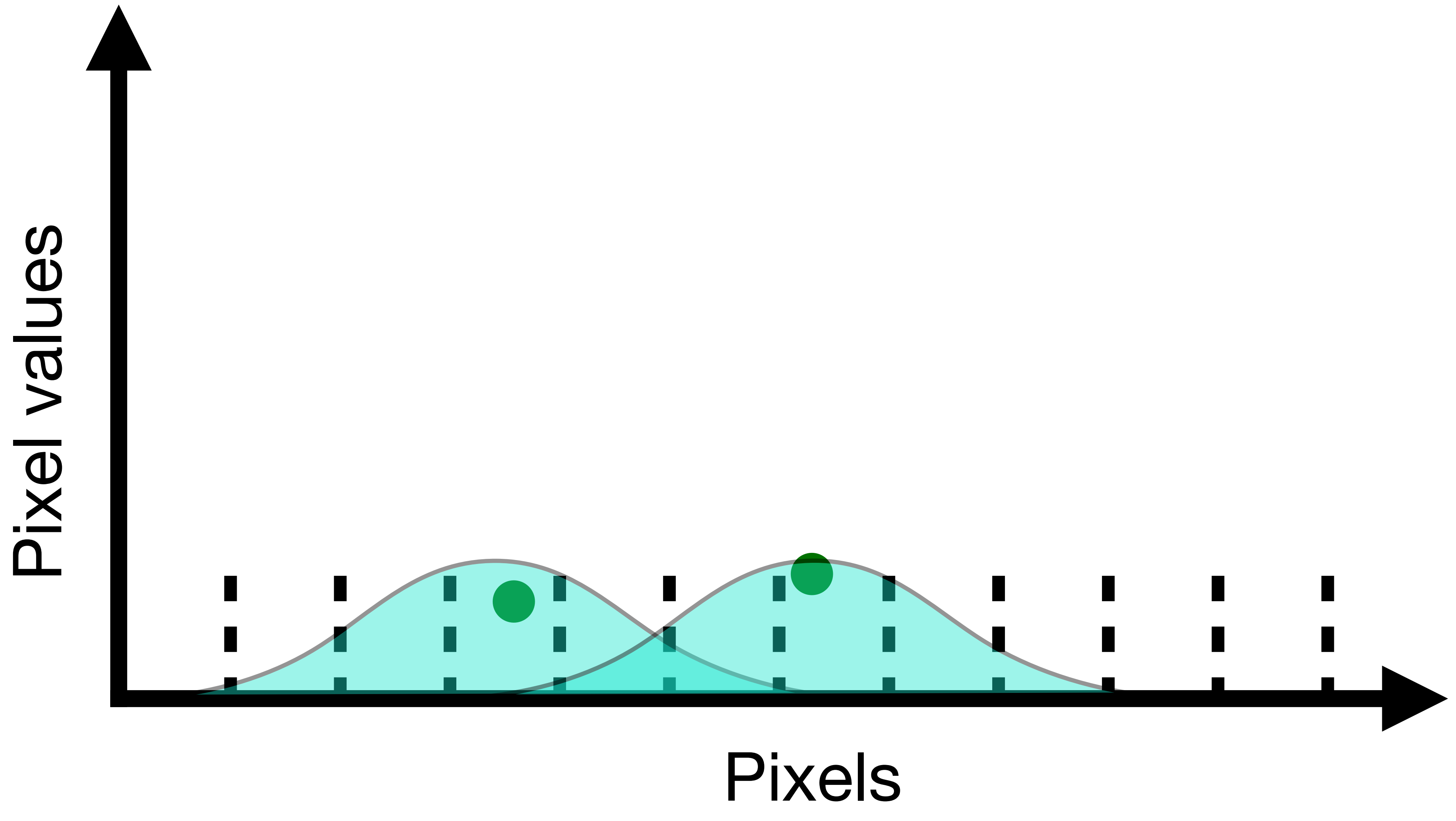
Reconstruction: Kernel Splatting

Kernel gather



2D example

Kernel Splatting

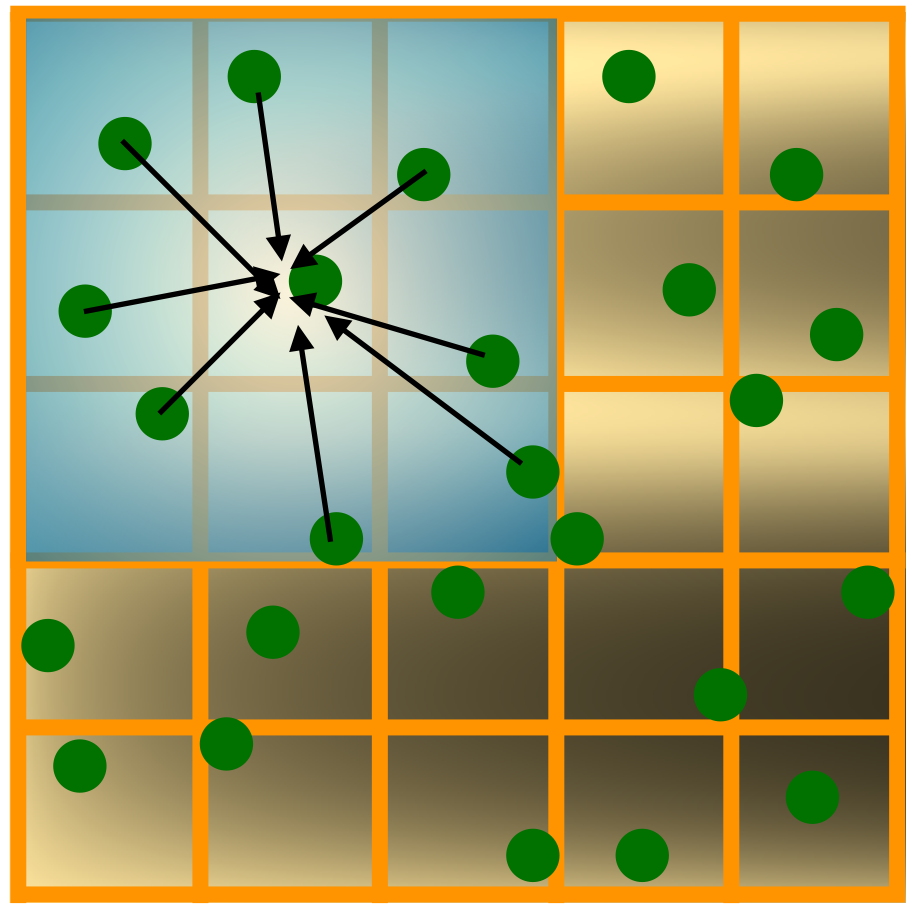


1D example

How should nearby samples influence me?

Reconstruction: Kernel Splatting

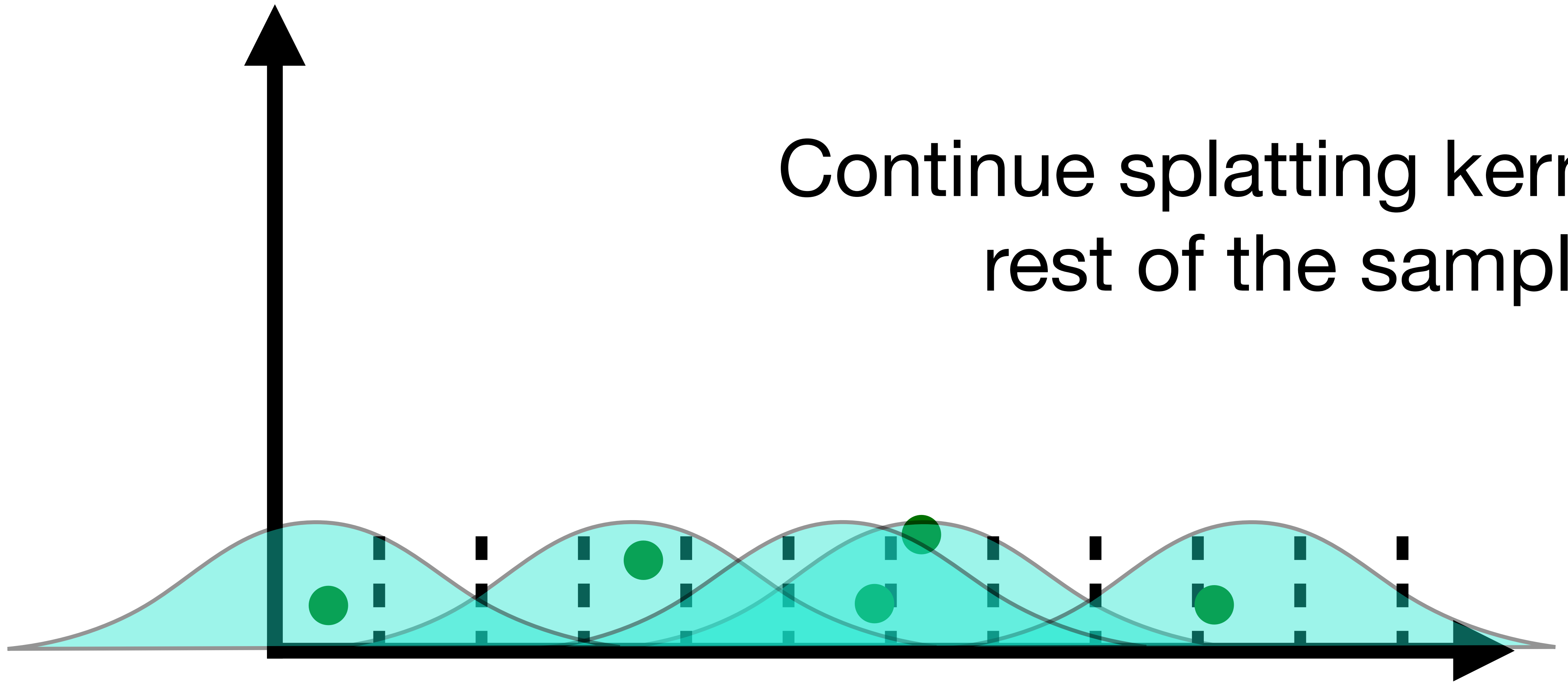
Kernel gather



2D example

How should nearby samples influence me?

Kernel Splatting

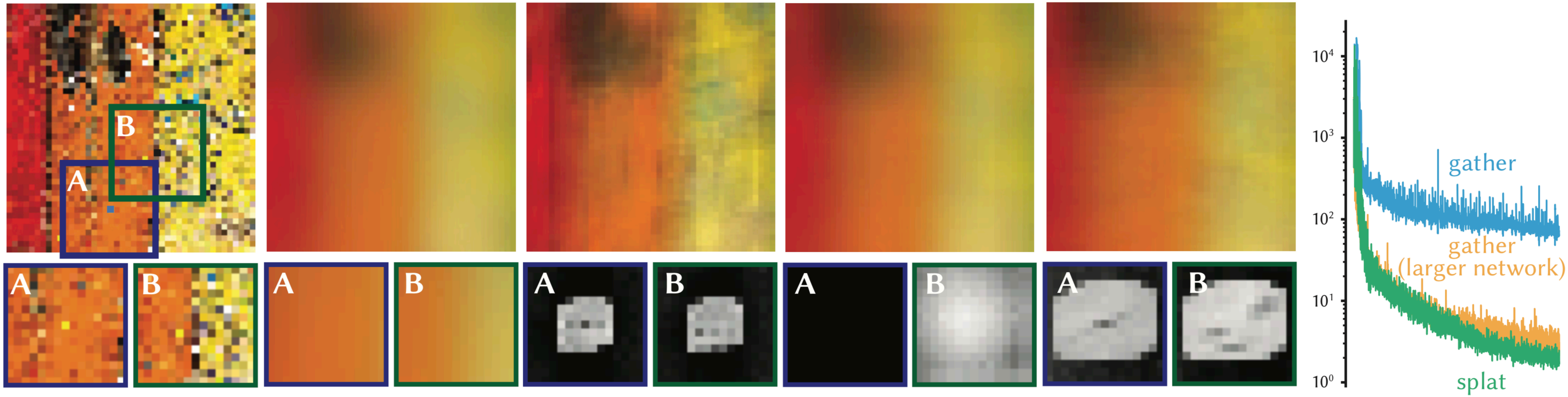


Continue splatting kernels for the rest of the samples....

How do I contribute to nearby pixels, given all the samples around me?

Network: Kernel Gather vs Splatting

intput reference gather splat gather (larger network)



gather kernels splat kernels gather kernels (large capacity network)

Permutation Invariance

Permutation Invariance

A model that produces the same output regardless of the order of elements in the input vector

Permutation Invariance: Example

 *  =

Permutation Invariance: Example

*

=

Permutation Invariance: Example

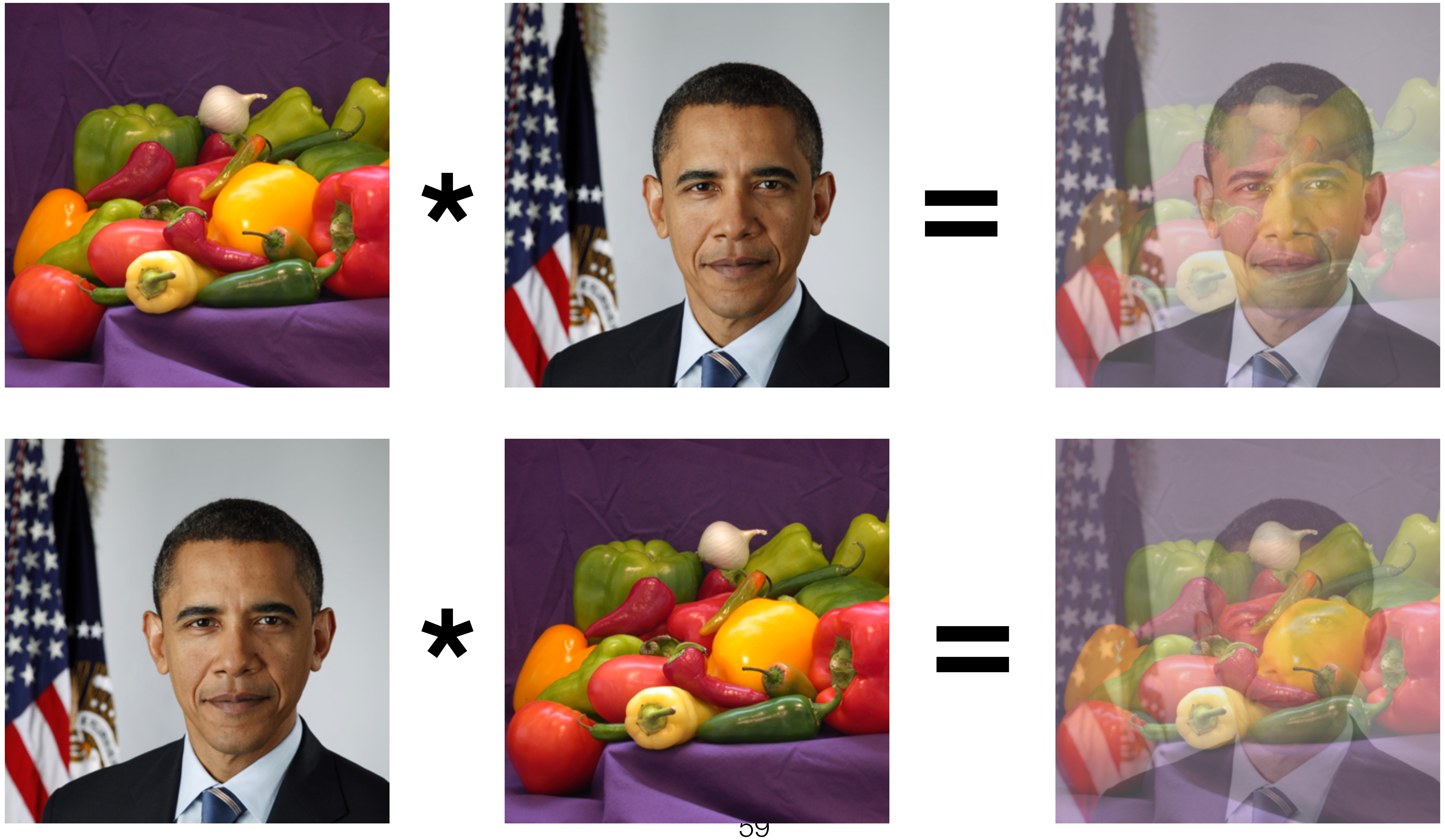
Not Permutation Invariance

 *  = 

 *  =

Permutation Invariance: Example

Not Permutation Invariance

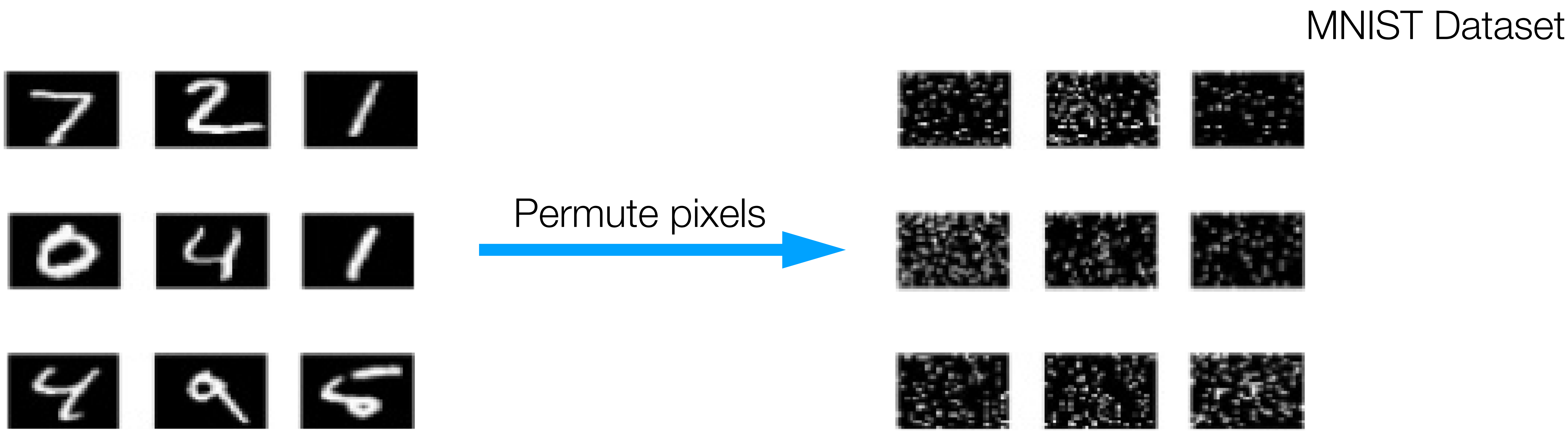


Permutation Invariance: Architectures

- A standard feedforward neural net such as multilayer perceptron (MLP) is insensitive to order of elements in input vector - so it is inherently permutation insensitive

- However, both a Convnet and RNNs for instance make full use of input ordering - they are permutation sensitive.

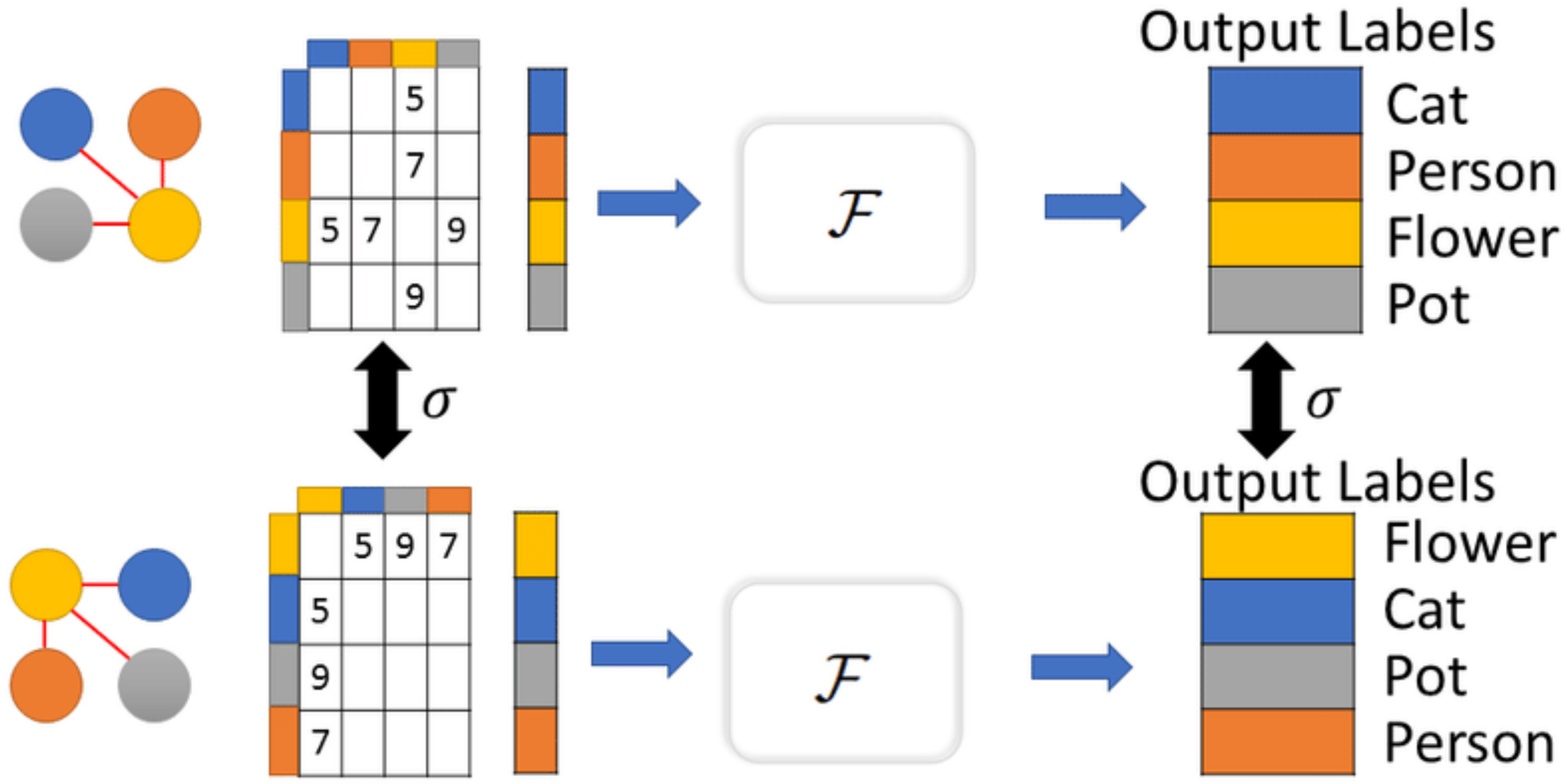
Permutation Invariance: Example



Permuting pixels makes it difficult for humans to understand the images.

However, permutation invariant networks like MLP can detect digits irrespective of the order of pixels

Permutation Invariance: Example

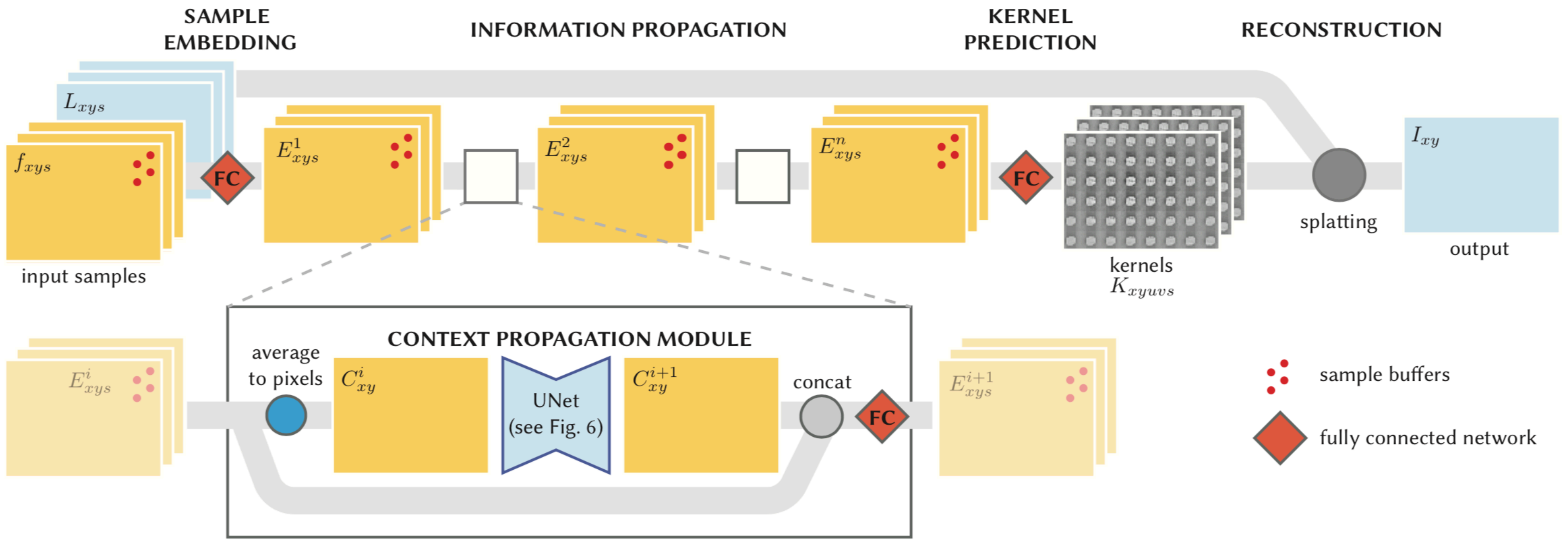


A graph labeling function F is graph permutation invariant (GPI) if permuting the names of nodes maintains the output. Herzig et al.[2018]

Permutation Invariance

- In MLPs, since each component is connected to each other, the order does not matter
- In structured convolutions, the order matters and therefore, it is not permutation invariant.

Proposed Network Architecture



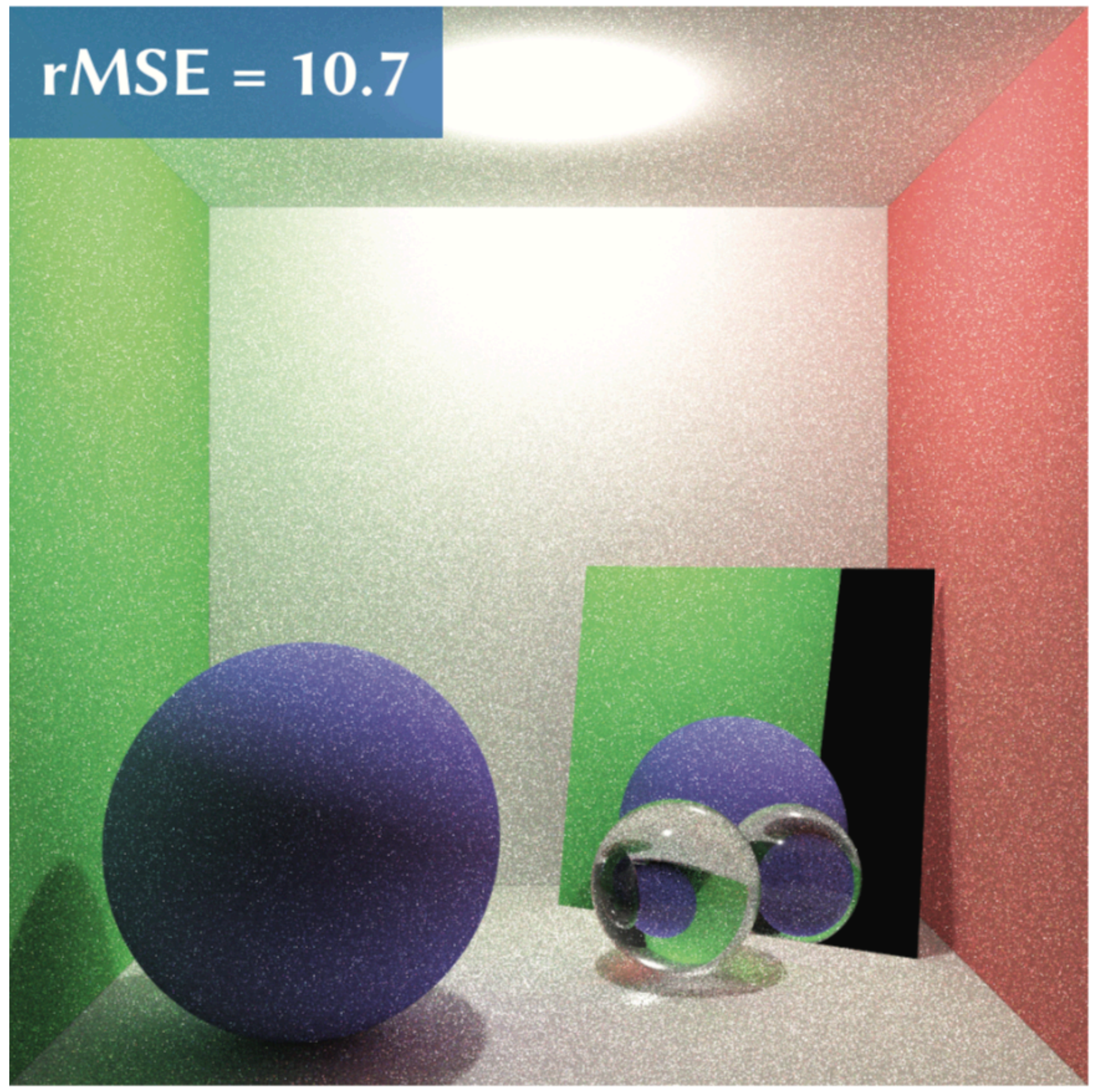
Dataset and Training Procedure

Procedurely generated dataset: 300,000 renderings with 128x128 resolution

Also generated input buffer (4, 32 spp), but this time also maintained auxiliary features

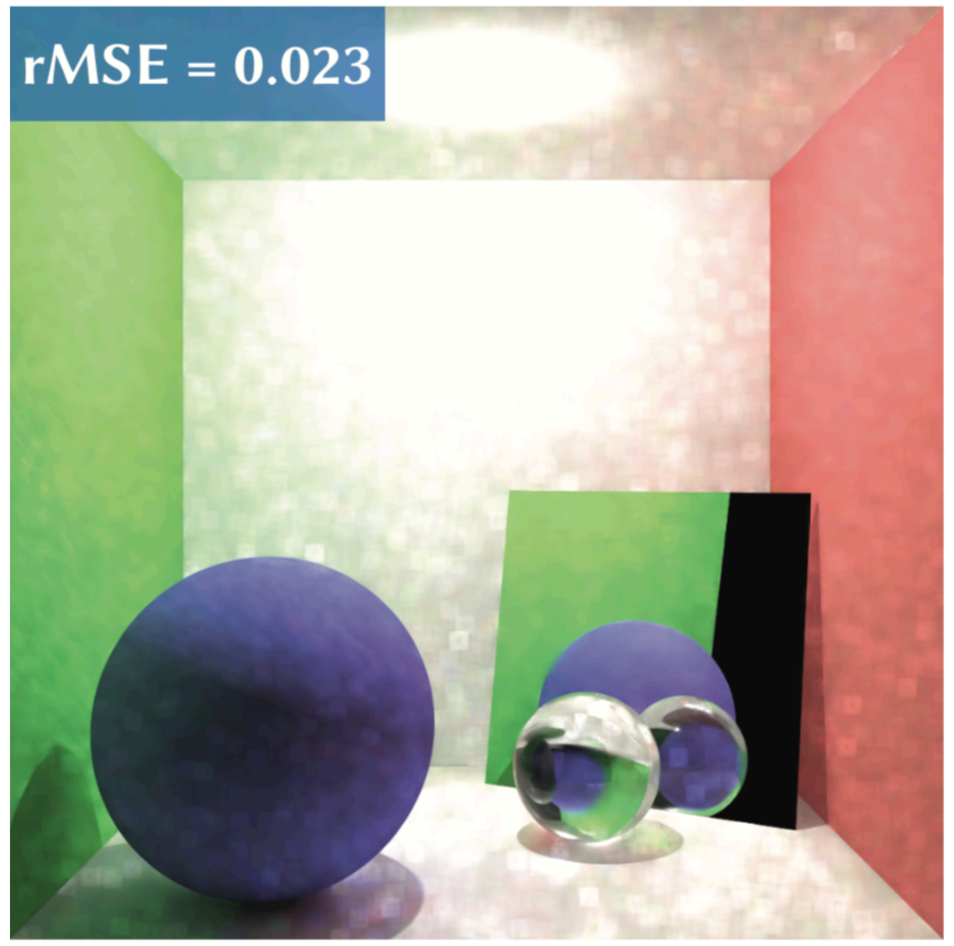
Reference was generated for 4096 samples

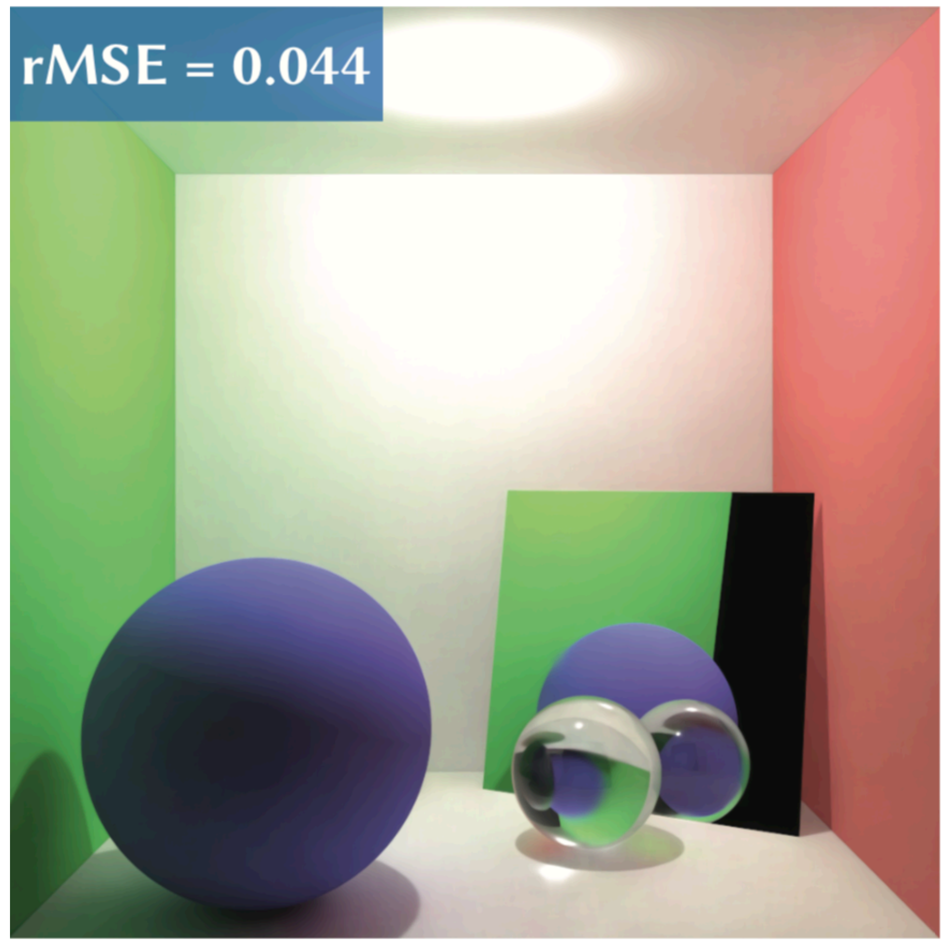
Splat vs Gather



Input

per sample gather

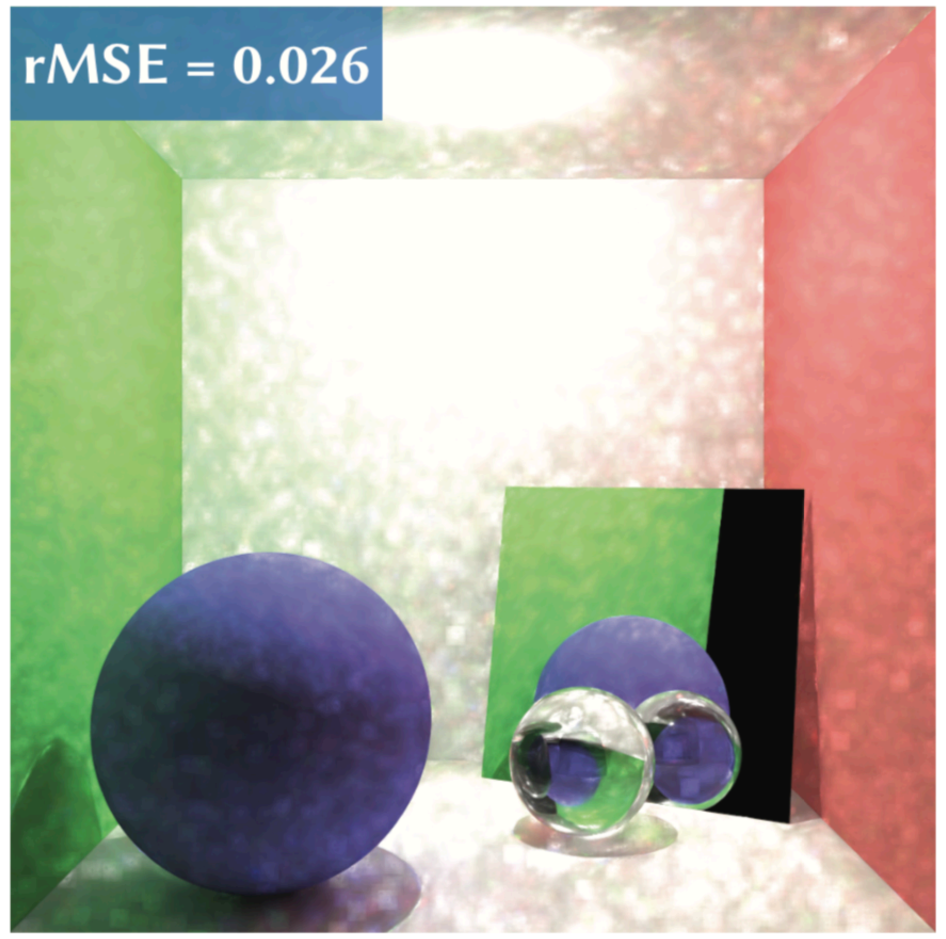


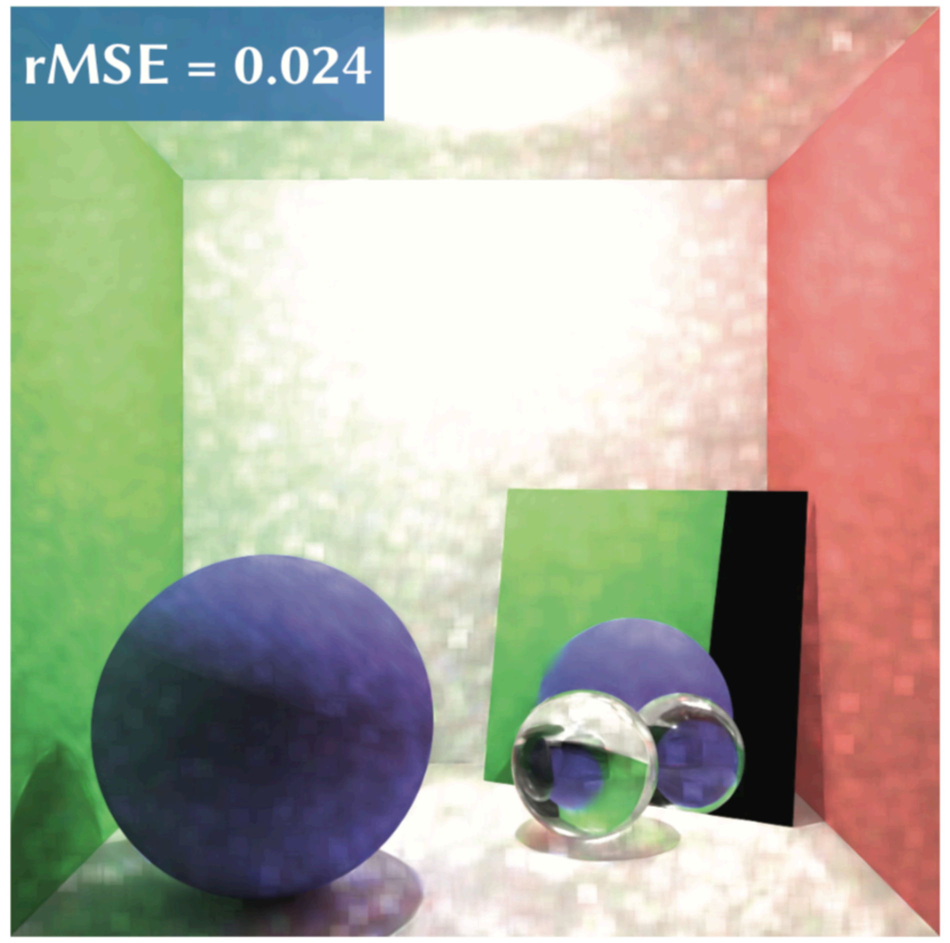


per sample splat

Reference

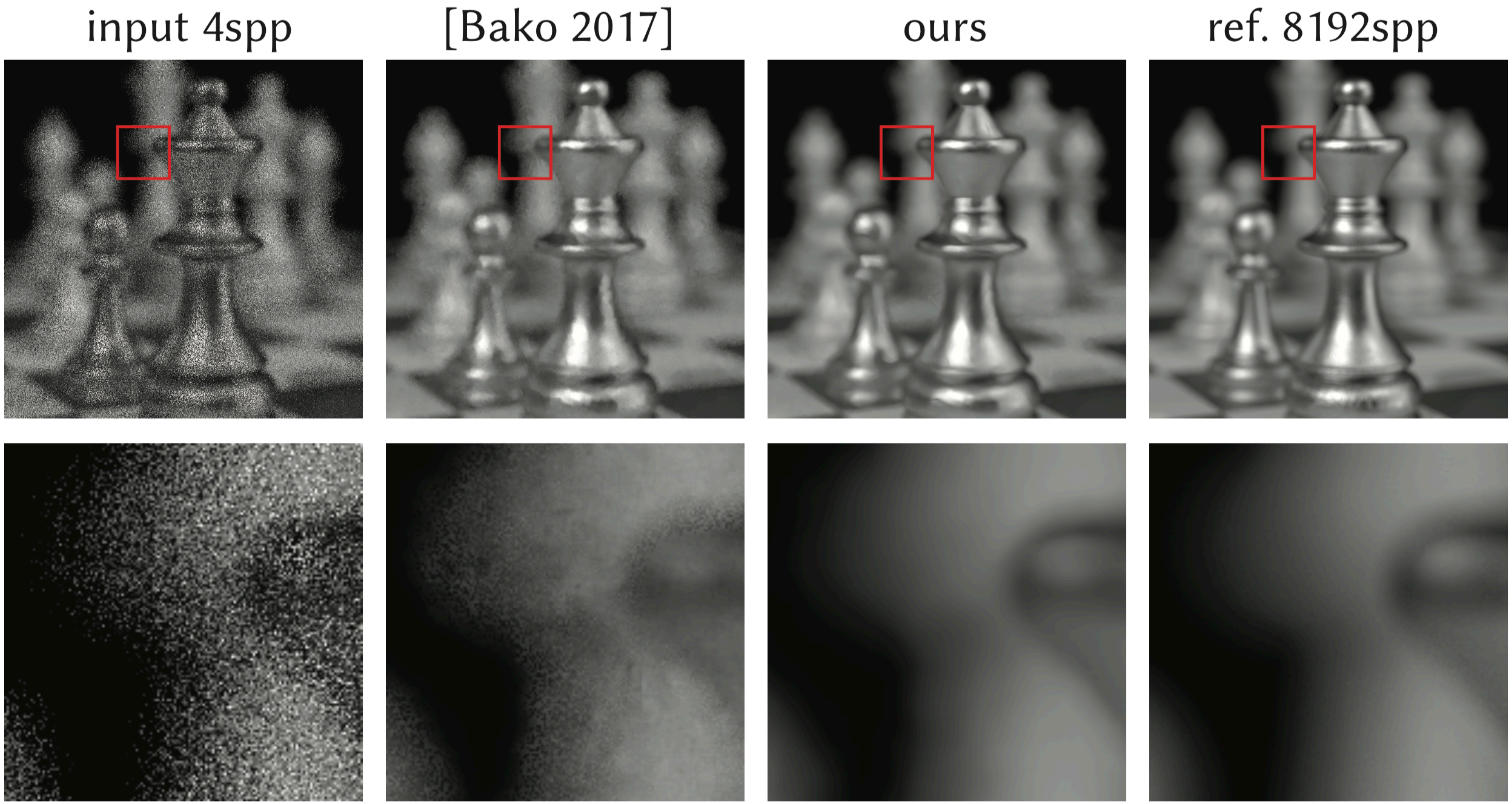
per pixel gather





per pixel splat

Results



Network Architecture Comparisons

Final Exam

20.08.19 from 10:00 to 13:00 in HS 002

References

A frequency analysis of light transport, Durand et al. SIGGRAPH 2005

Frequency Analysis and Sheared Reconstruction for Rendering Motion Blur, Egan et al. SIGGRAPH 2009

Temporal Light Field Reconstruction for Rendering Distribution Effects, Lehtinen et al. SIGGRAPH 2011

On Filtering the Noise from the Random Parameters in Monte Carlo Rendering, Sen and Darabi 2012

A Machine Learning Approach for Filtering Monte Carlo Noise, Kalantari et al. SIGGRAPH 2015

Interactive Reconstruction of Monte Carlo Image Sequences using a Recurrent Denoising Autoencoder, Chaitanya et al. SIGGRAPH 2017

Kernel-Predicting Convolutional Networks for Denoising Monte Carlo Renderings, Bako et al. SIGGRAPH 2017

Sample-based Monte Carlo Denoising using a Kernel-Splatting Network, Gharbi et al. SIGGRAPH 2019