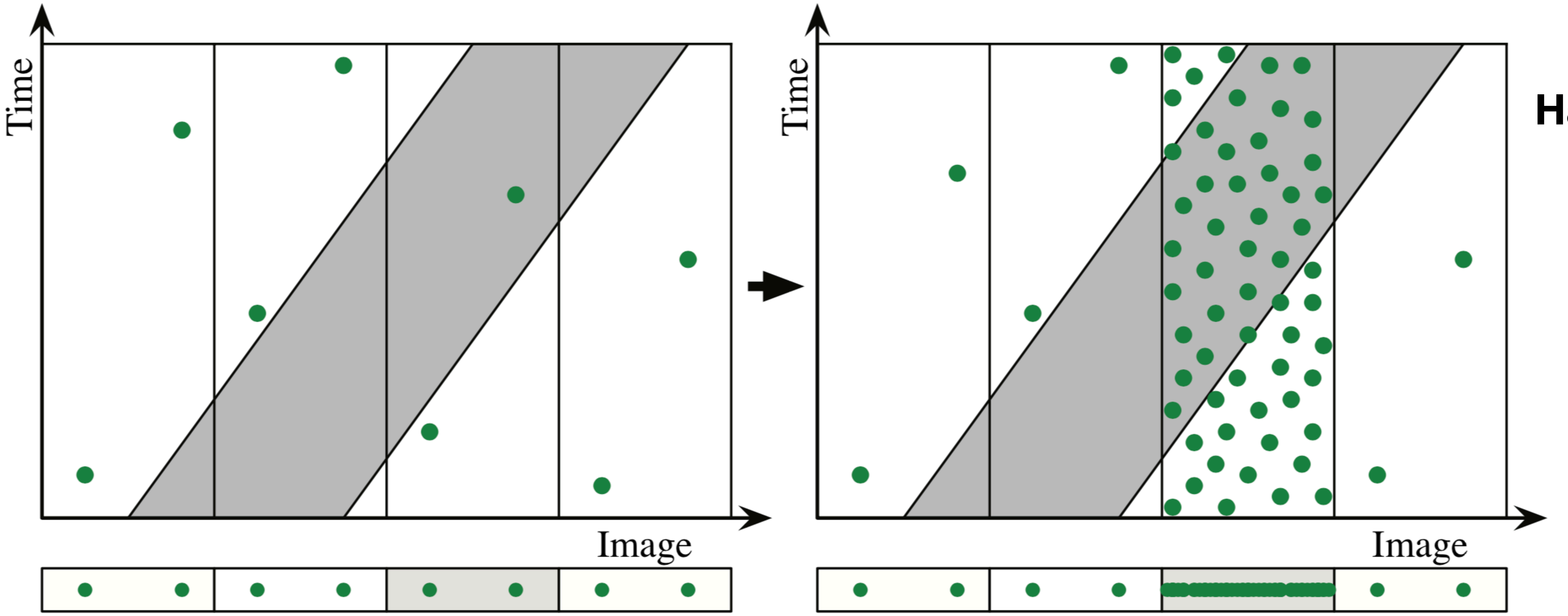
Denoising Algorithms: Path to Neural Networks II

Image courtesy Vogel et al. [2018]

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

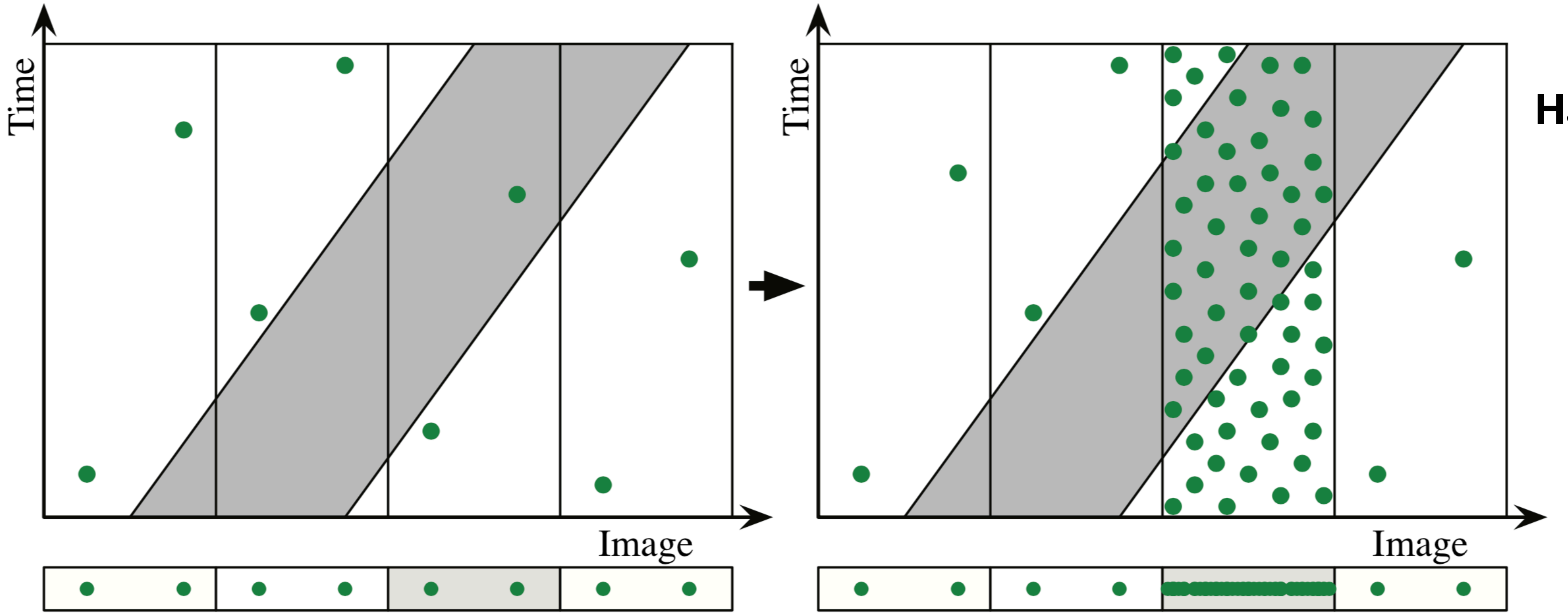
Recap

Image-space Adaptive Sampling



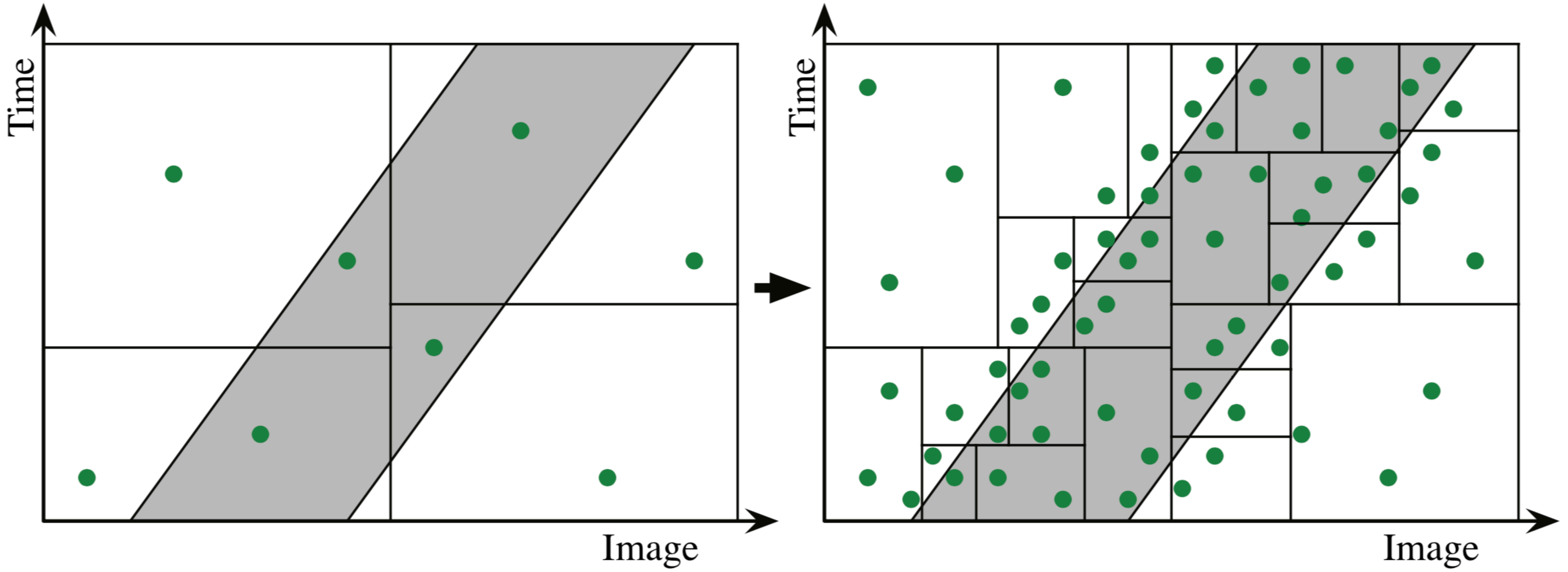
Hachisuka et al. [2008]

Image-space Adaptive Sampling

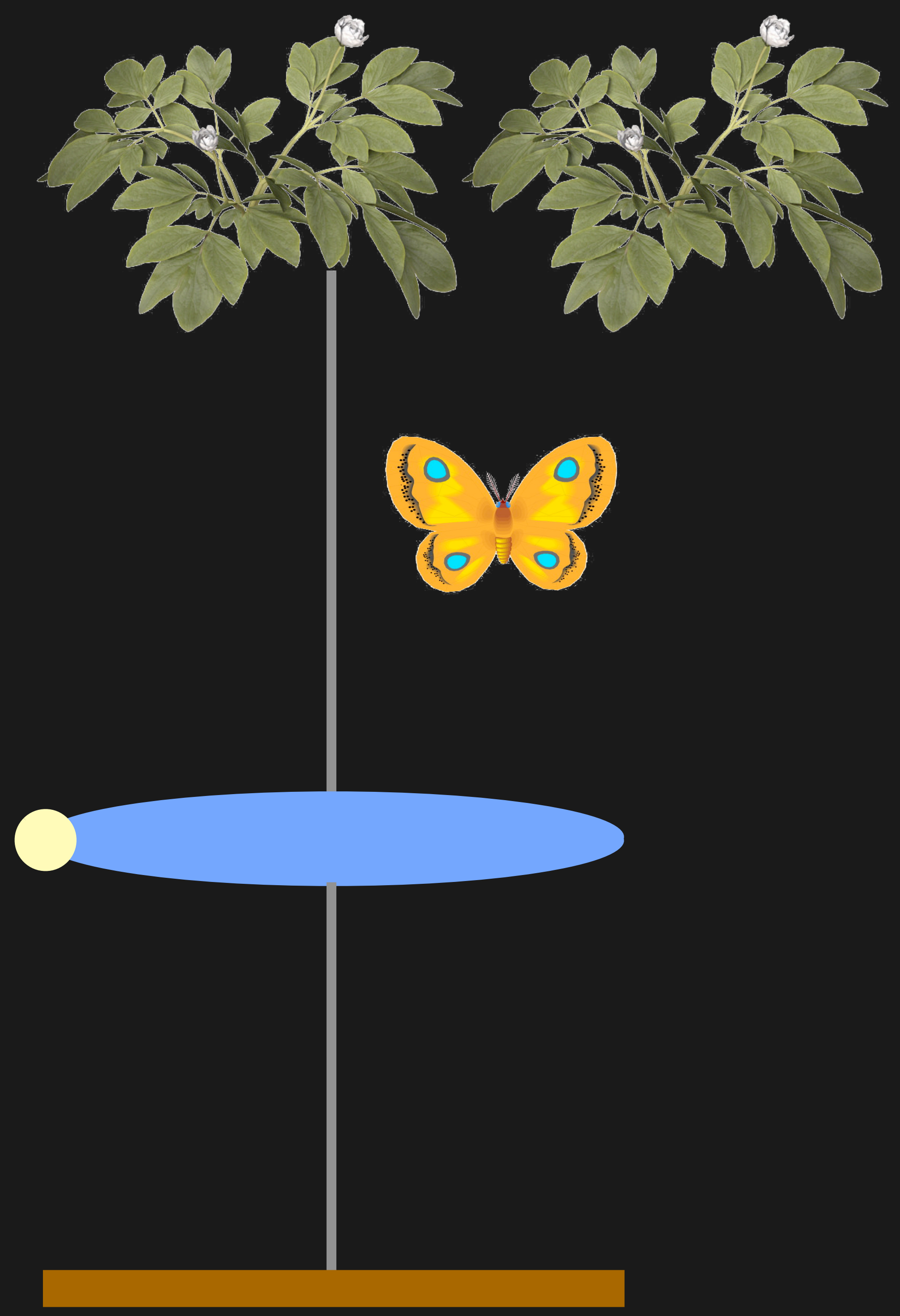


Hachisuka et al. [2008]

Multidimensional Adaptive Sampling



Depth of field



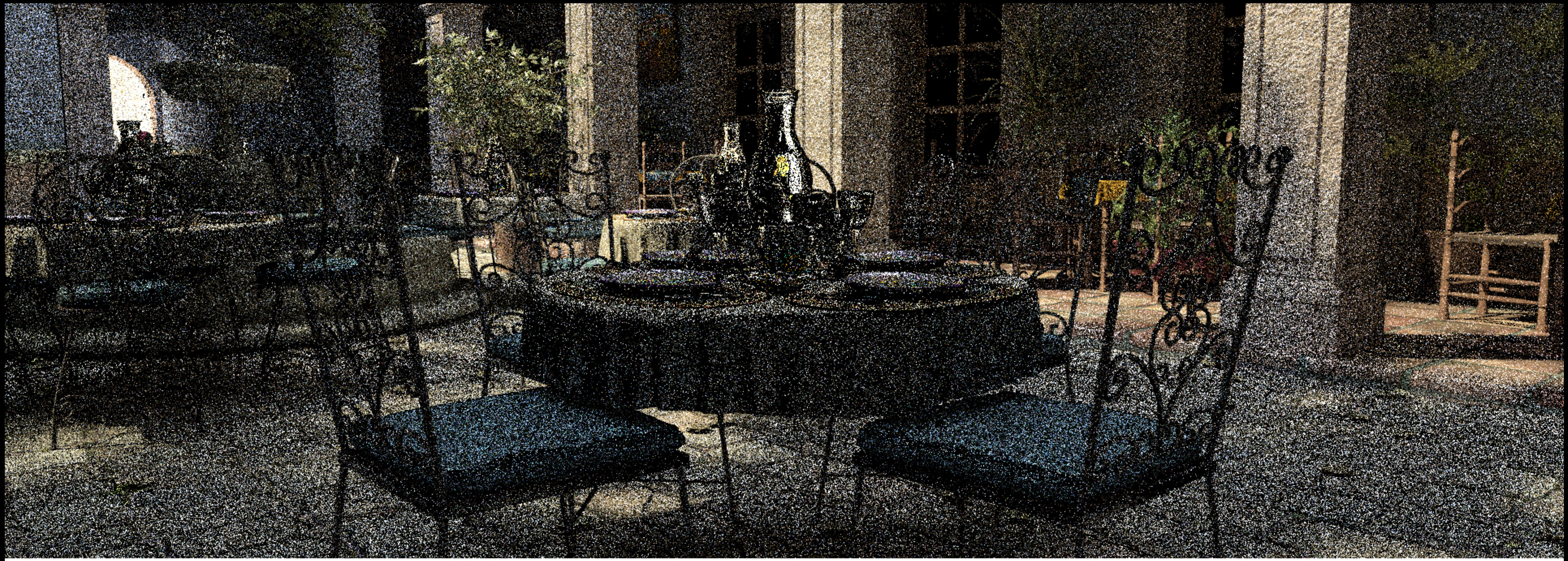
Slide from Jakko Lehtinen


Lens u
Slide from Jakko Lehtinen
Screen x

Visibility: SameSurface

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

Slide from Jakko Lehtinen


input Monte Carlo (8 samples/pixel)

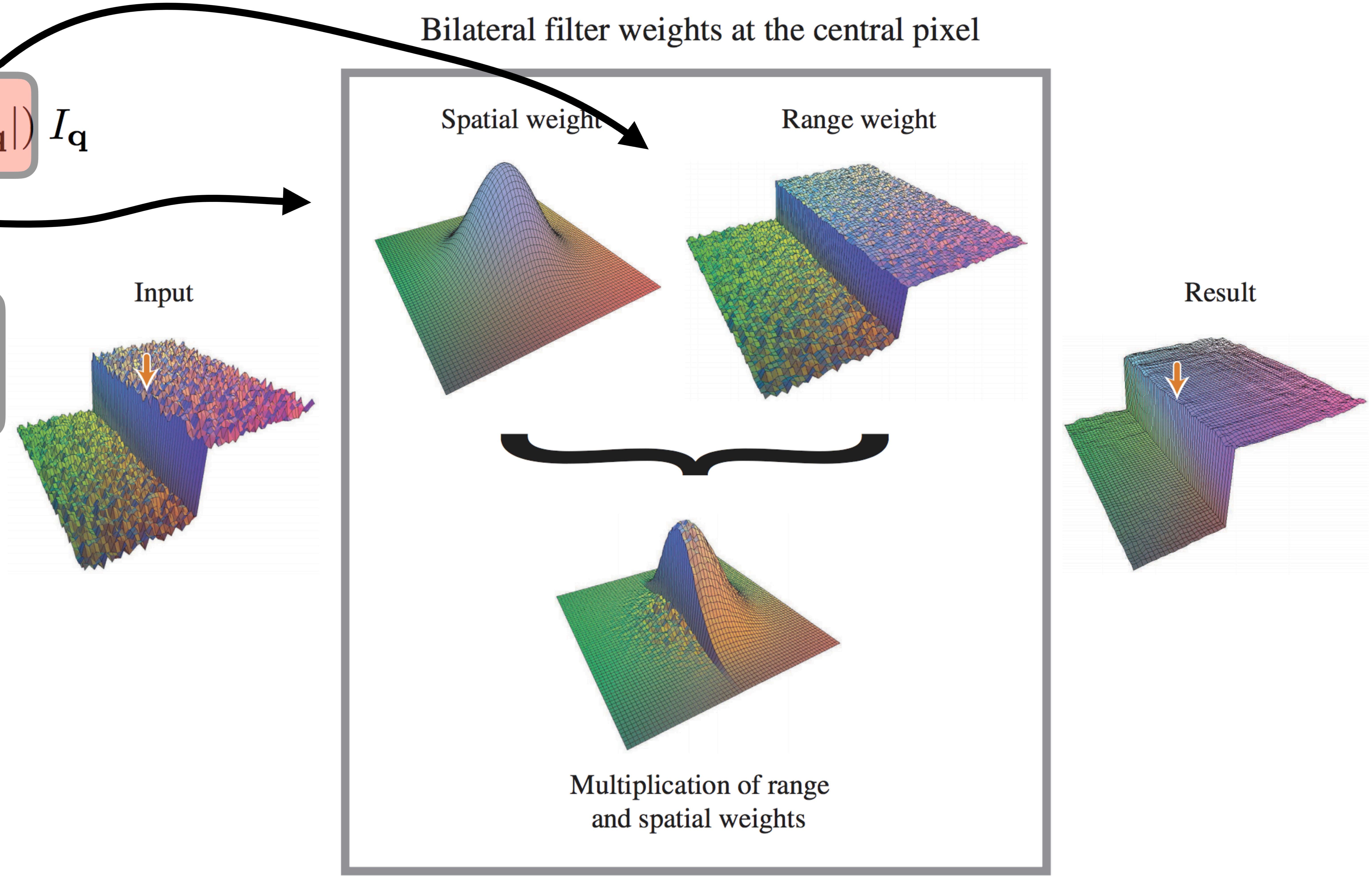
after RPF (8 samples/pixel)

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

Bilateral Filtering

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

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



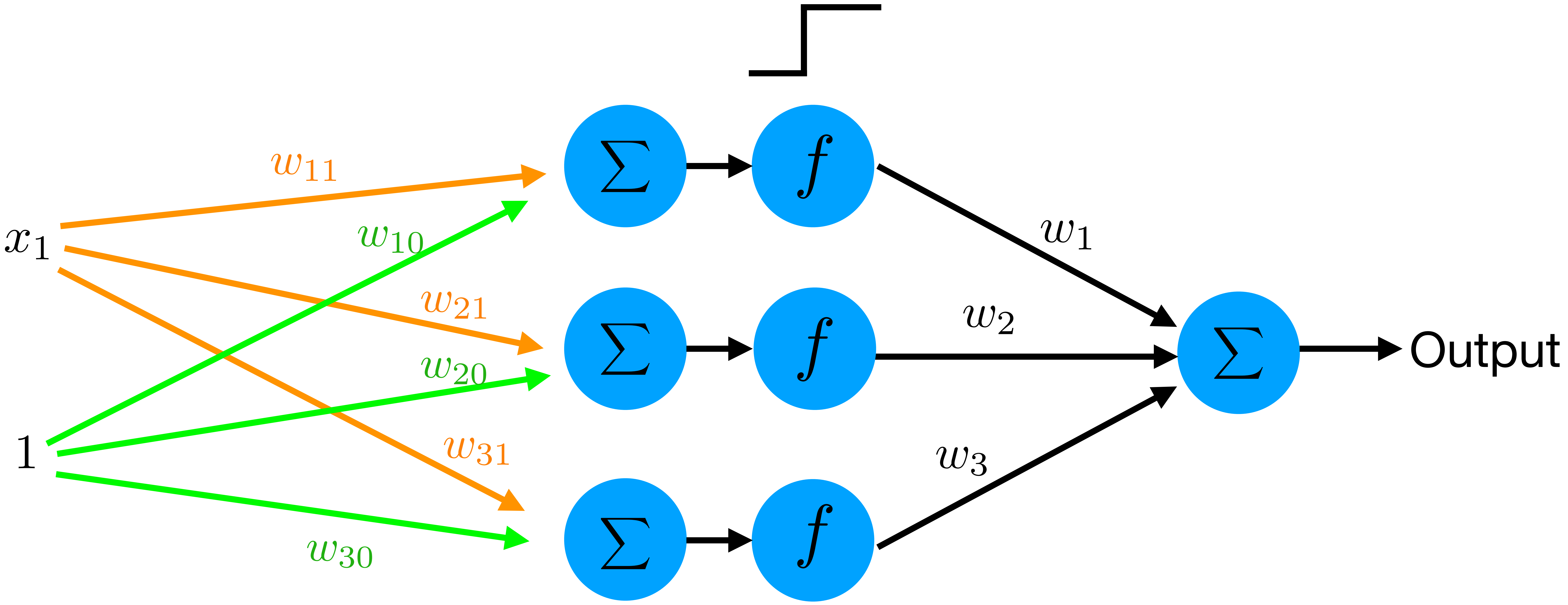
Bilateral Filtering of Features

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

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

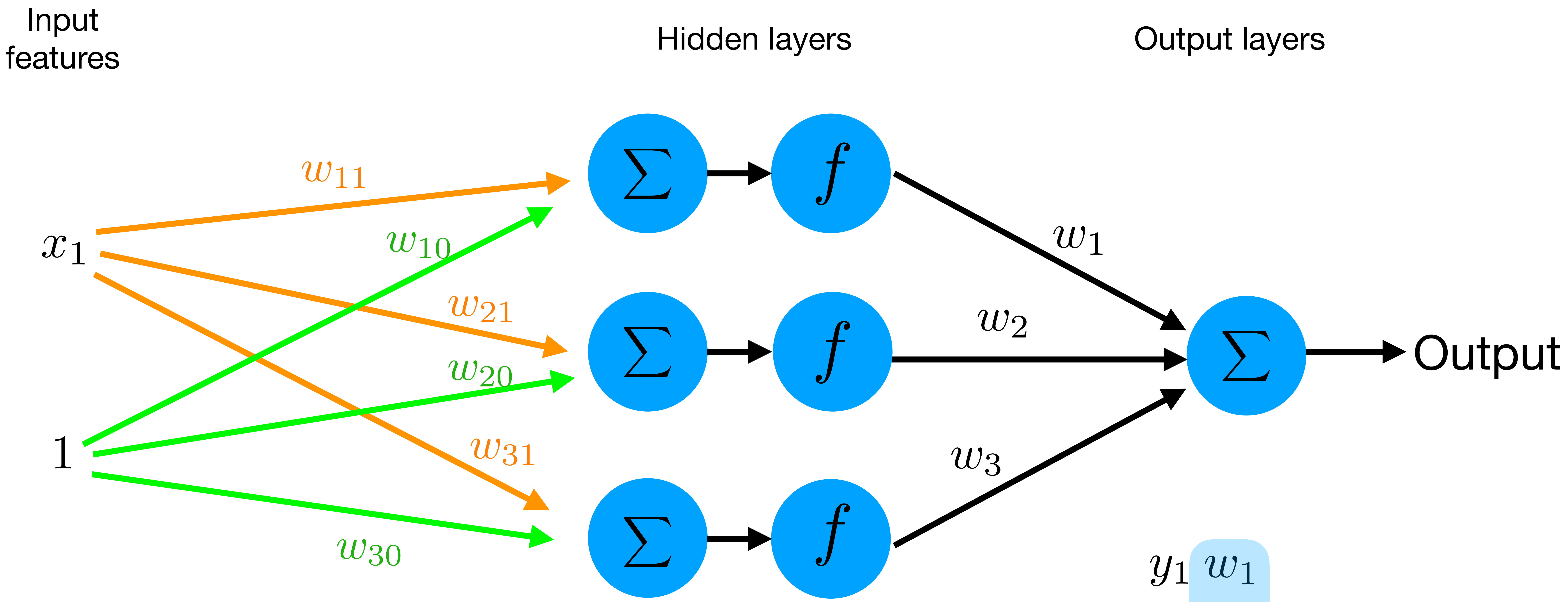
$$\exp[-\frac{1}{2\sigma_{\mathbf{f}}^2} \sum_{1 \leq k \leq m} \boxed{\beta_k} (\bar{\mathbf{f}}_{i,k} - \bar{\mathbf{f}}_{j,k})^2],$$

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

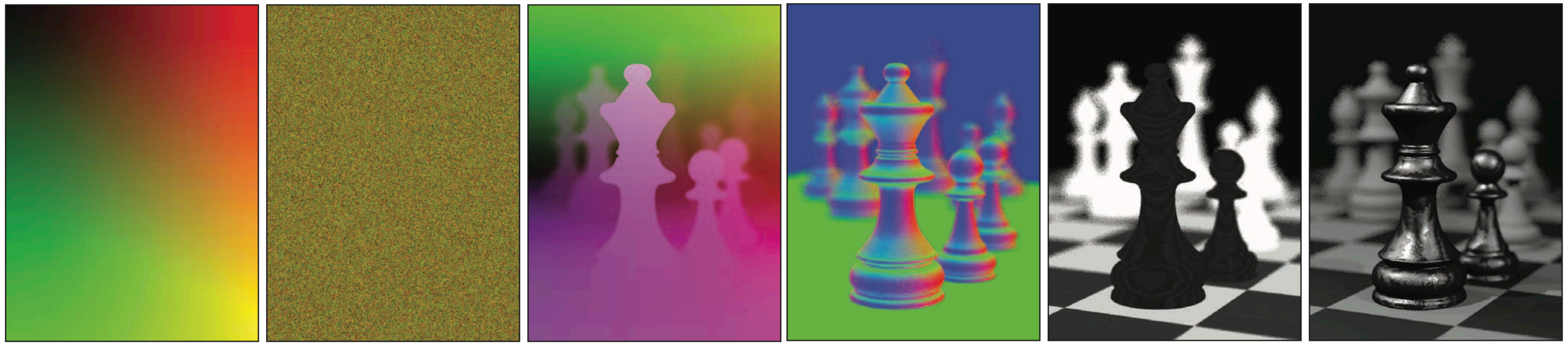
For cross Bilateral filters:

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

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)

Basics of Neural Networks

Each network has a forward pass and a backward (back-propagation) pass.

All components of the network must be differentiable.

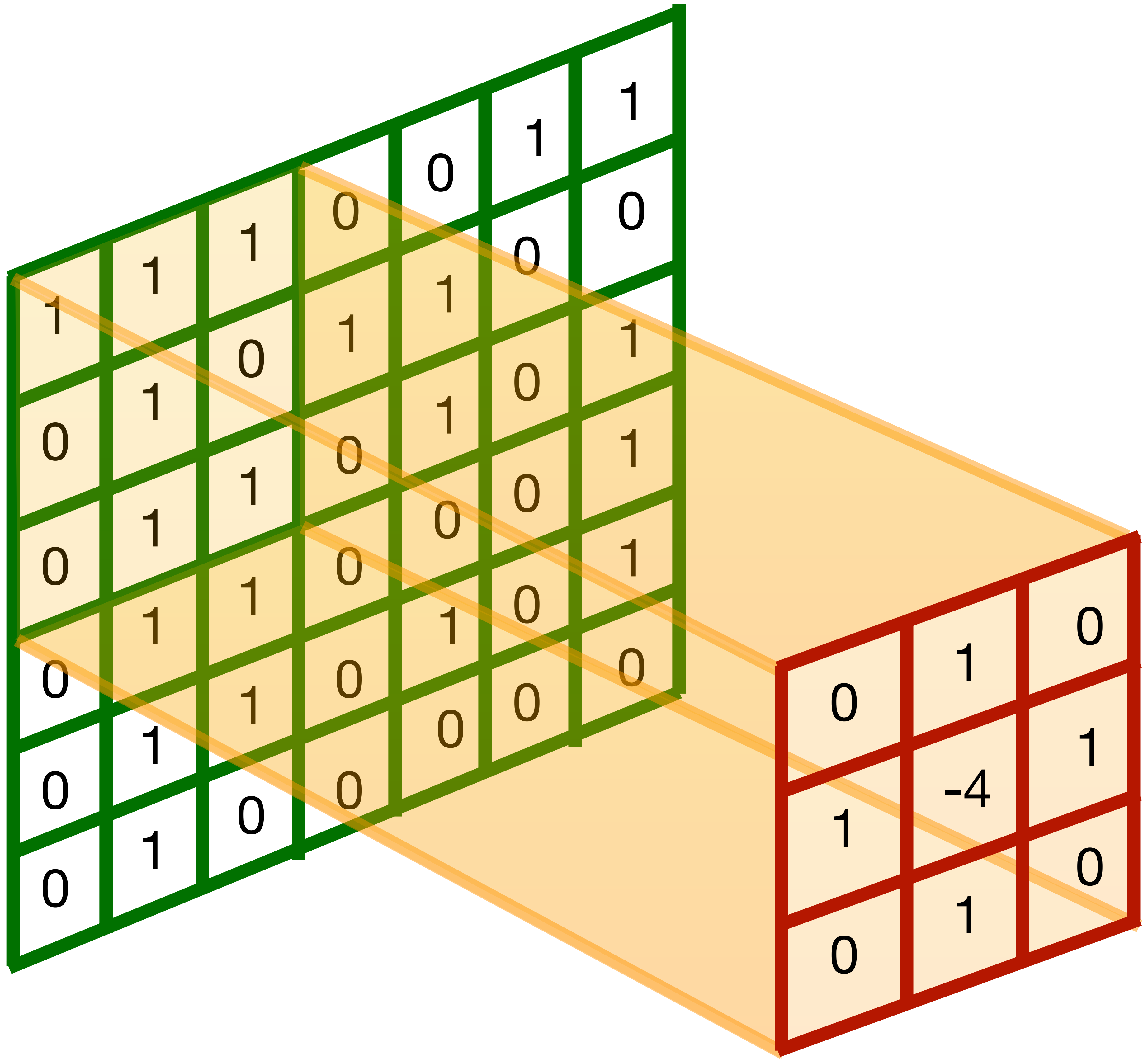
Differentiability is essential for back-propagation of error.

Introduction to CNNs

Kernel Predicting Denoising

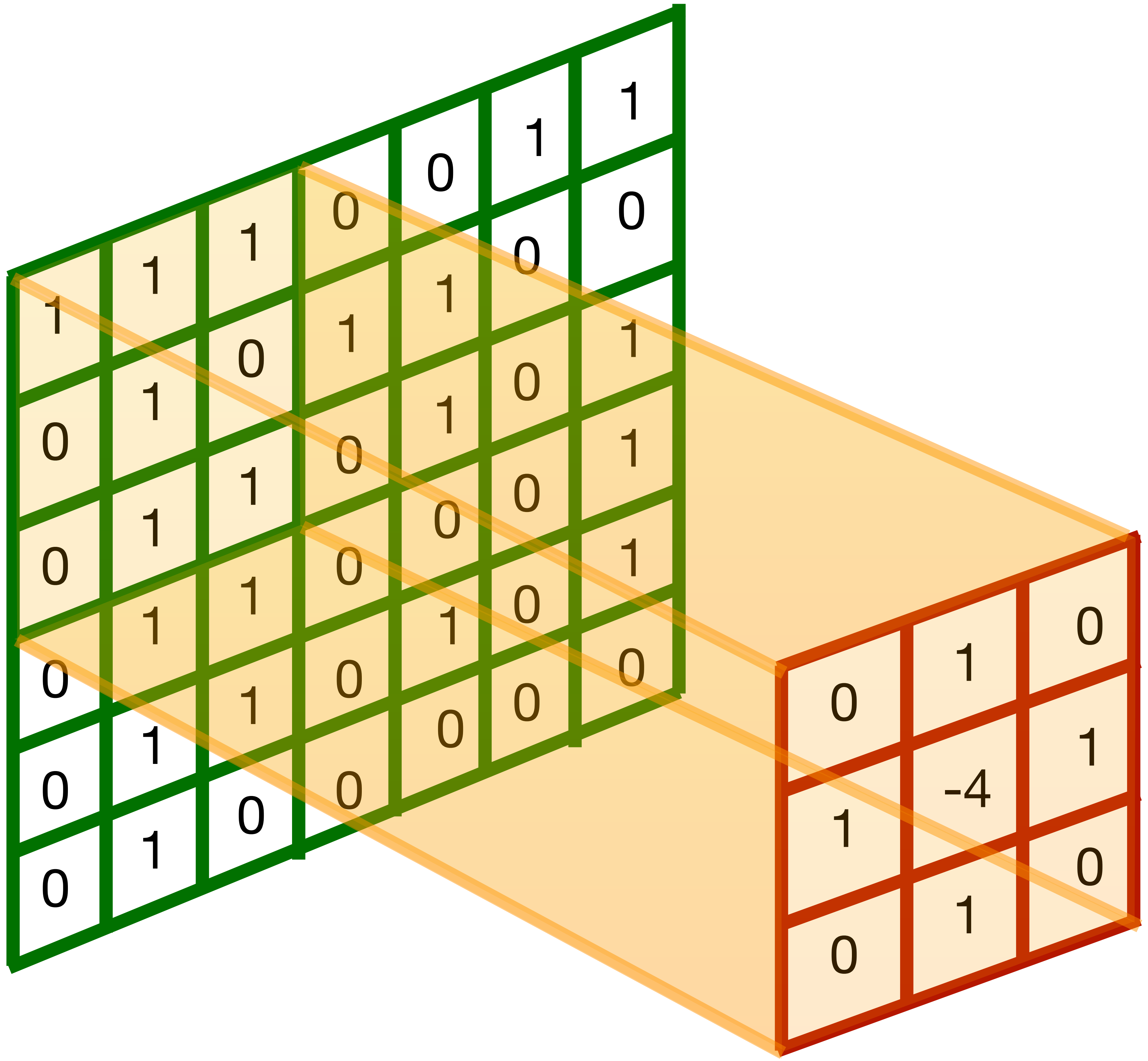
Sample-based MC Denoising

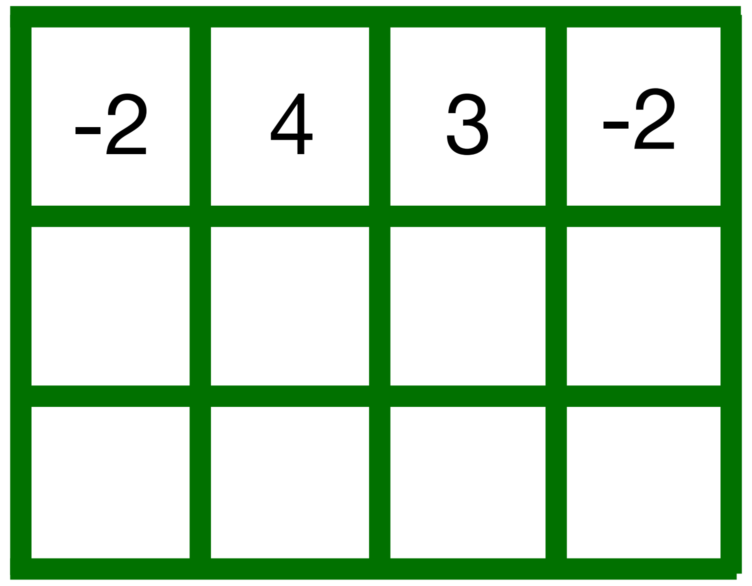
Convolution



No zero padding

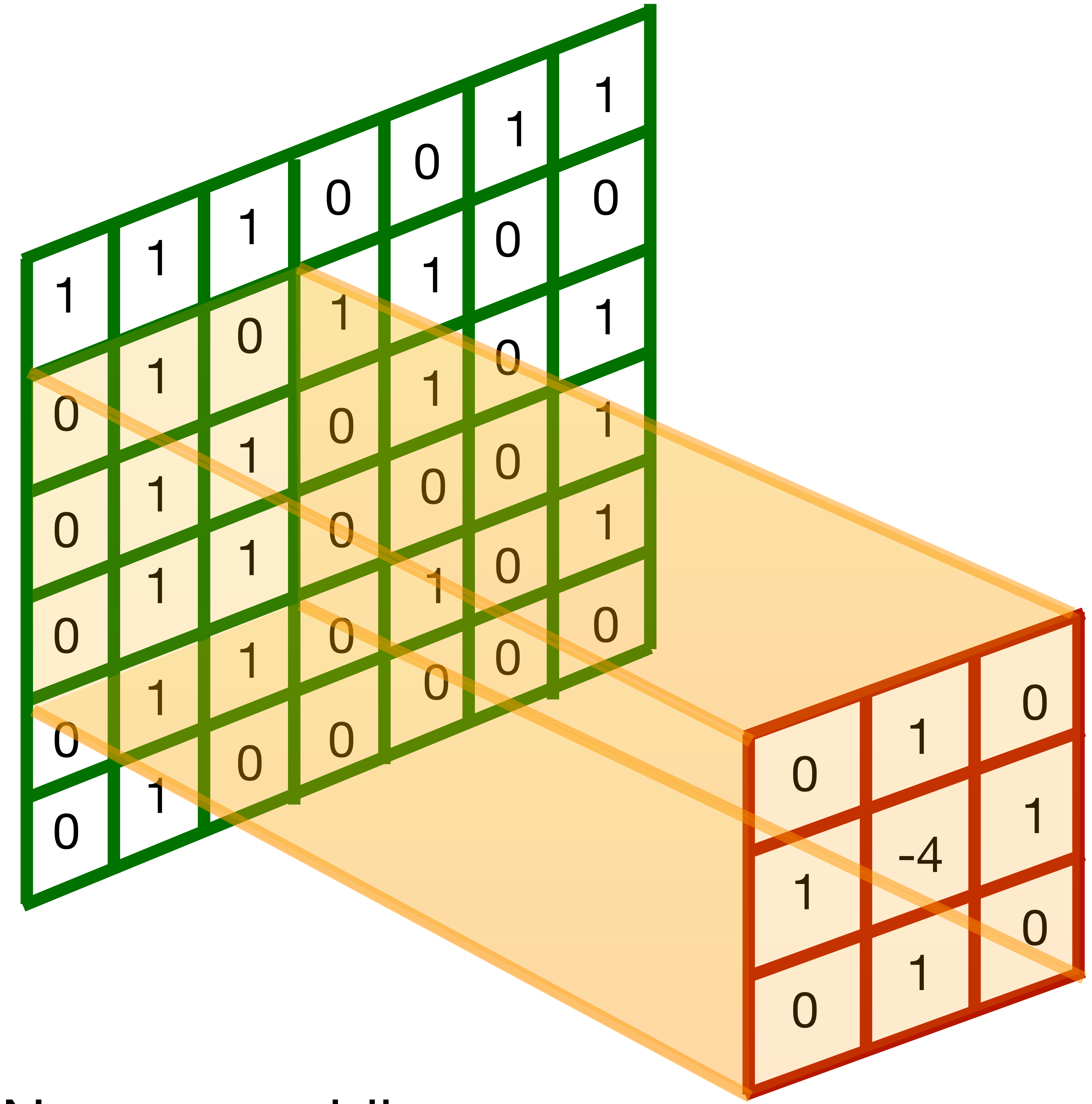
Stride-1 Convolution





No zero padding

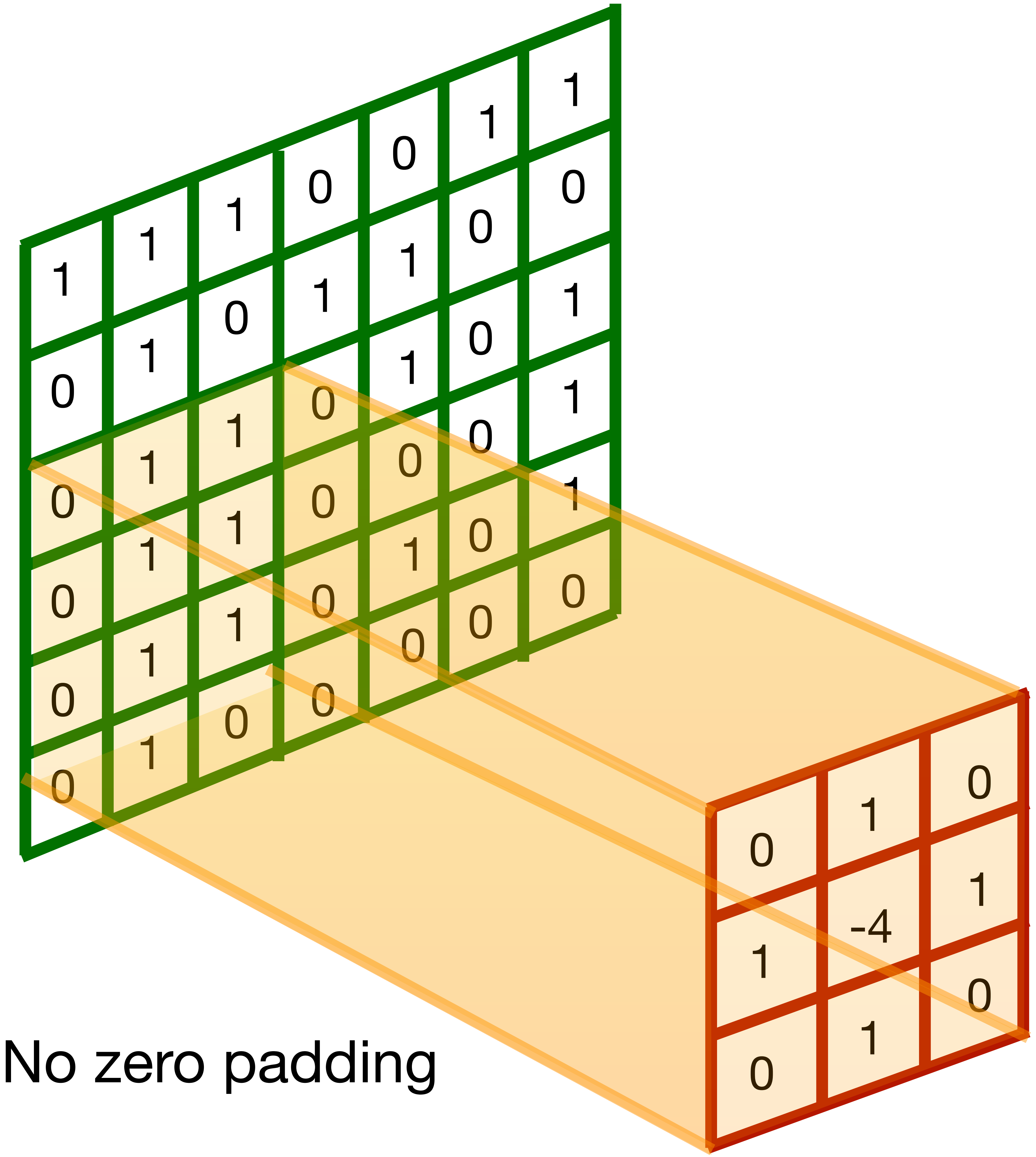
Stride-1 Convolution



-2	4	3	-2
-1	-2	3	-3

No zero padding

Stride-1 Convolution

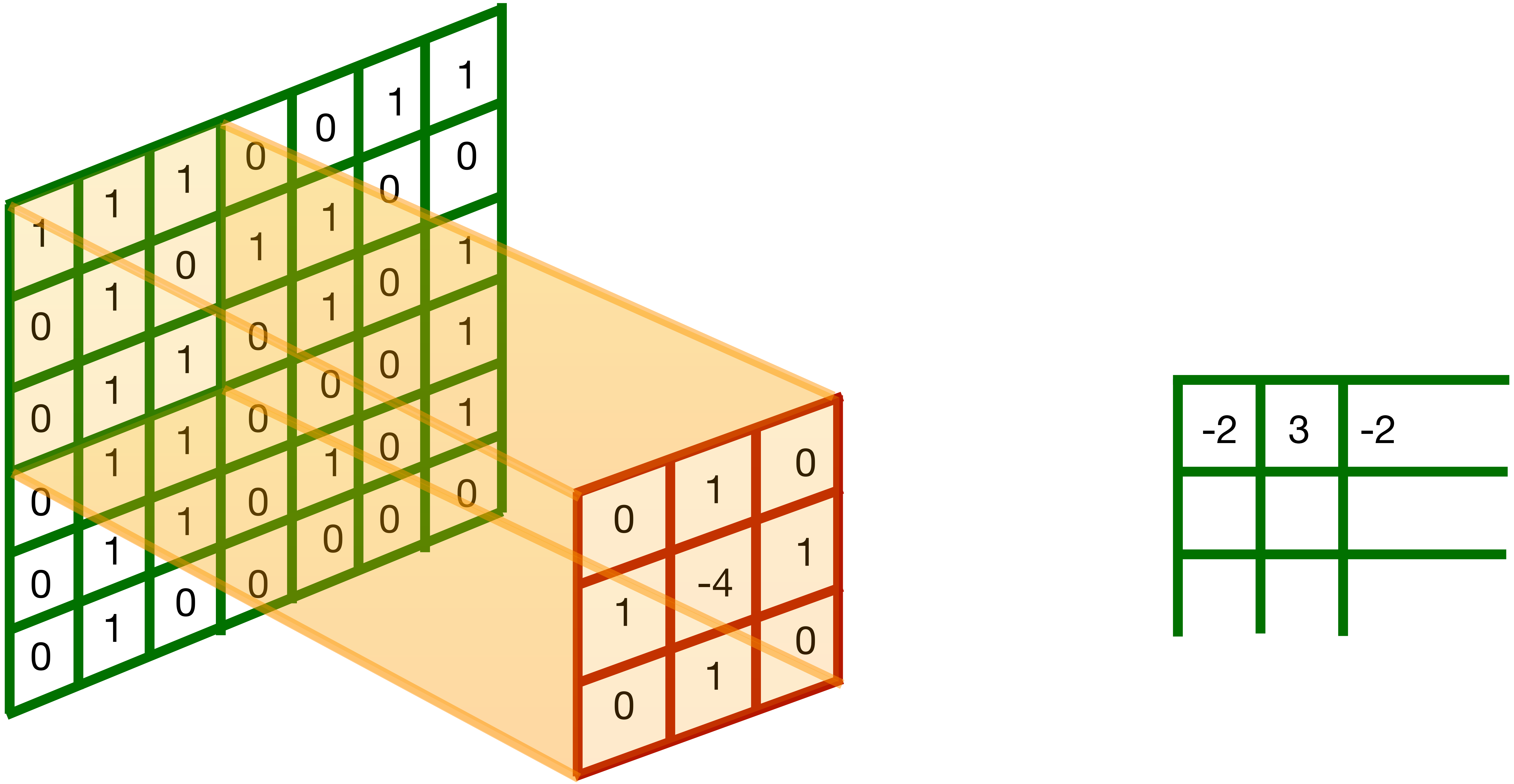


No zero padding

-2	4	3	-2
-1	-2	3	-3
-1	-2	1	1

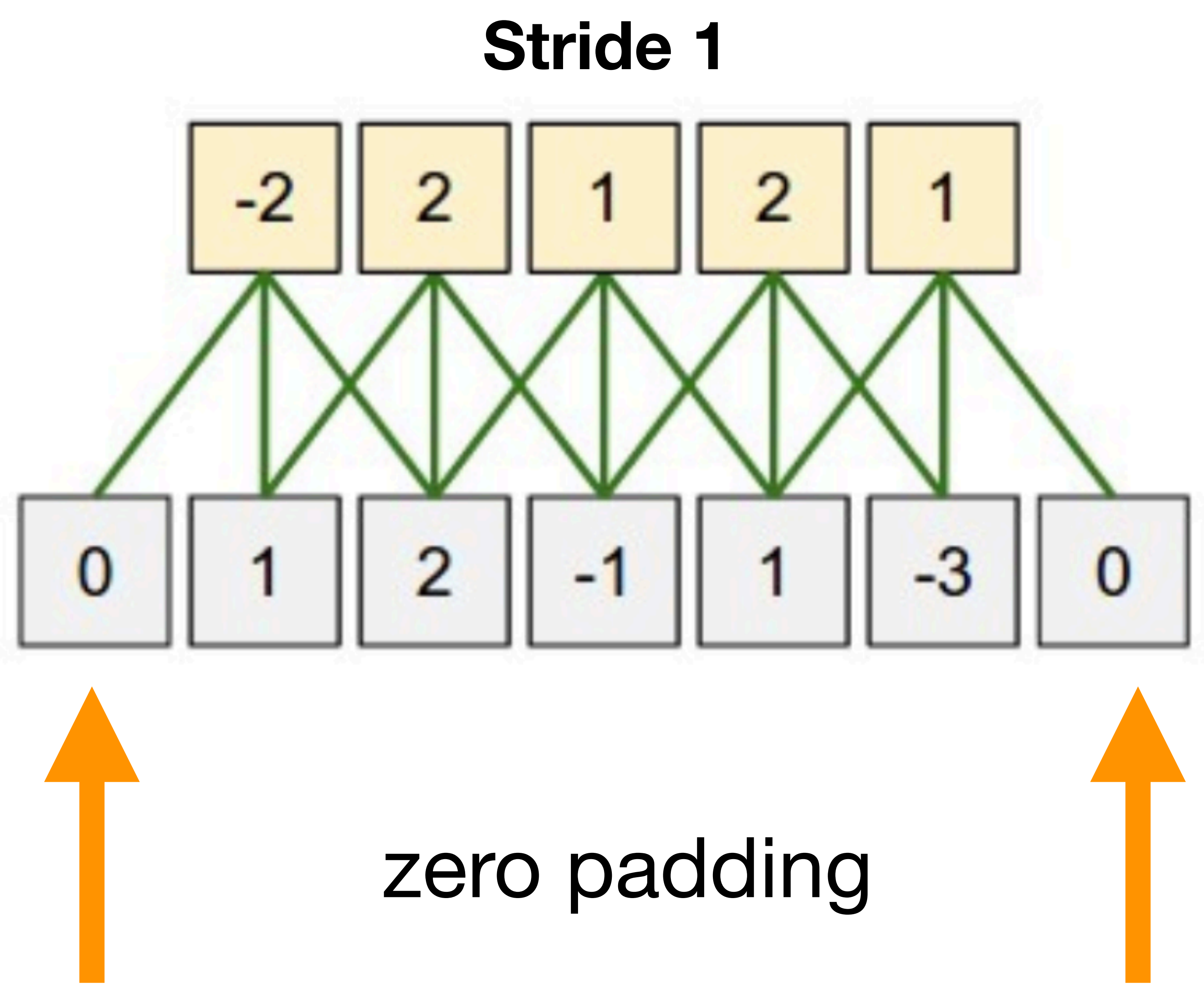
0

Stride-2 Convolution



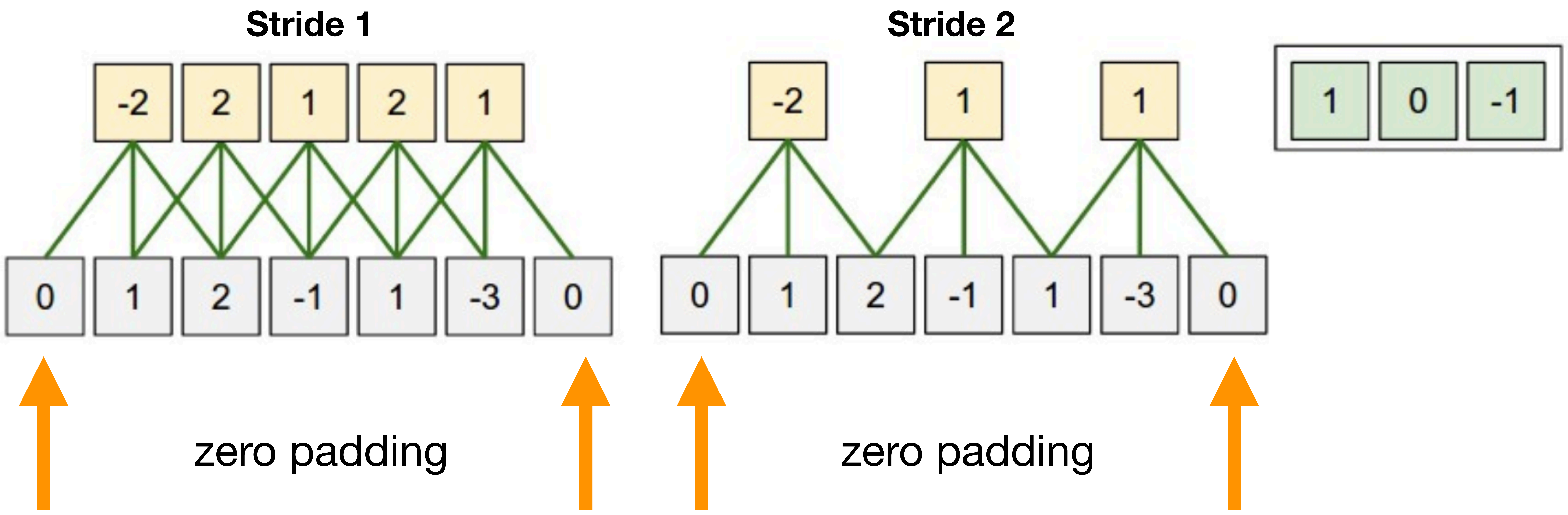
Zero Padding and Strides

1D image to illustrate the strides and zero padding

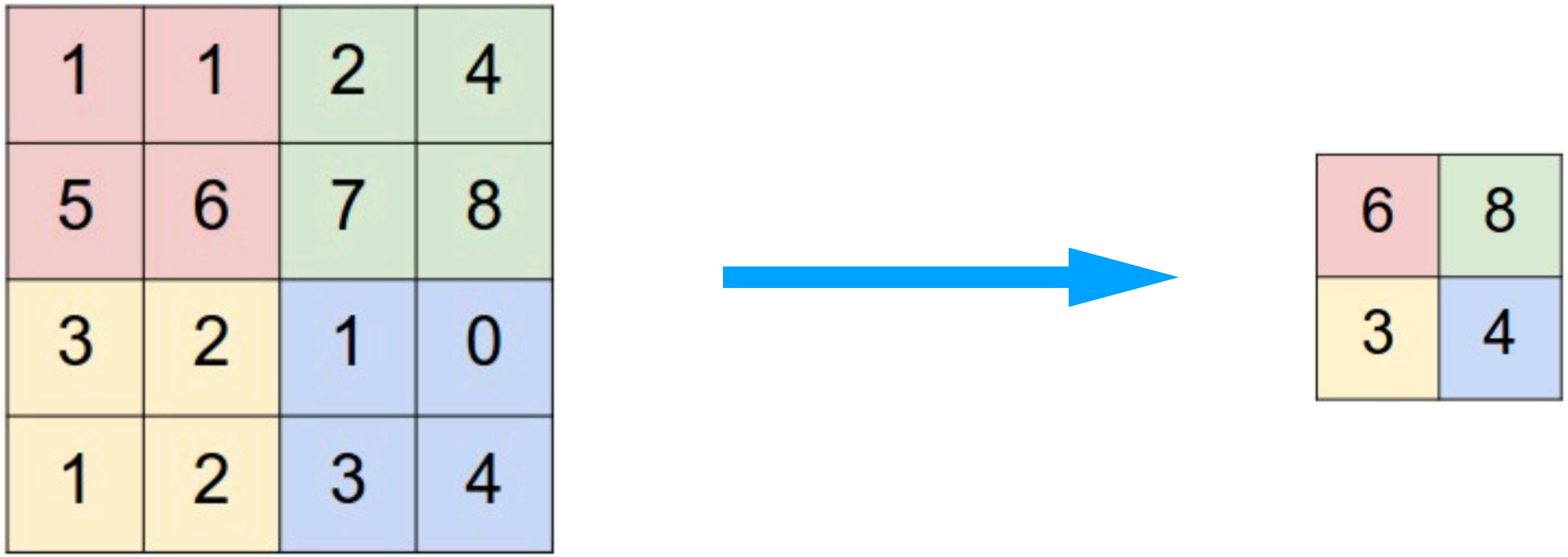


Strides

1D image to illustrate the strides and zero padding



Max Pooling / Down Sampling



Overview on Convolutional Neural Networks

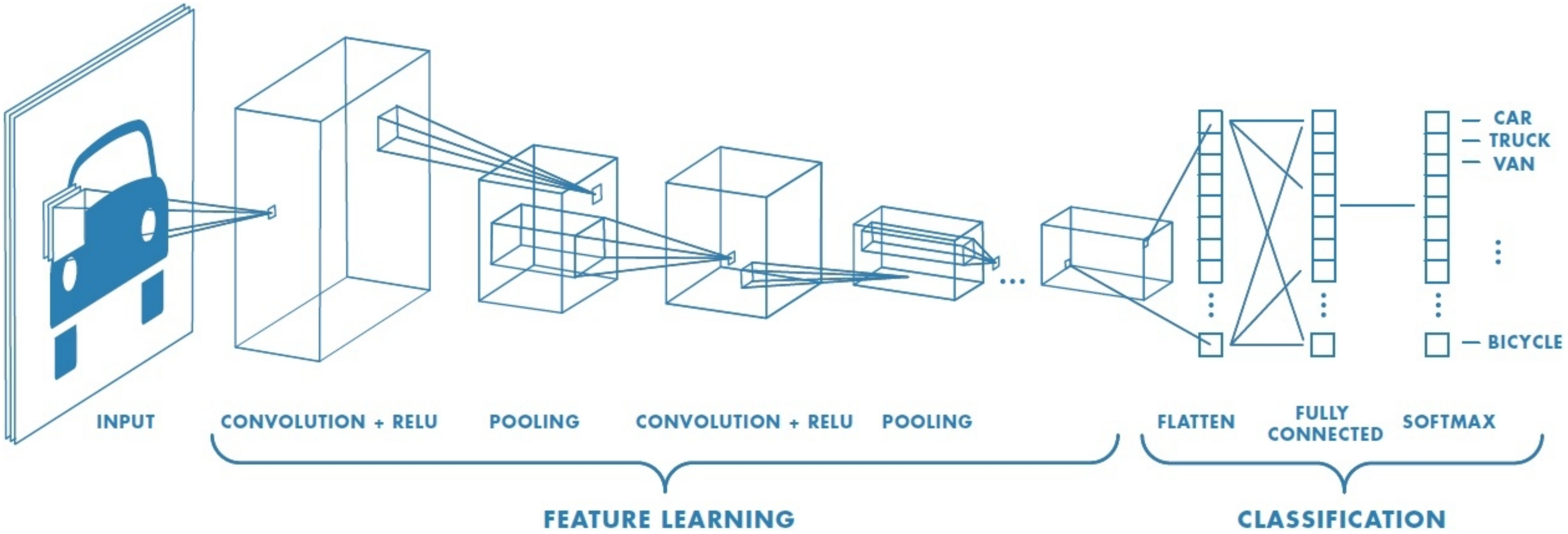
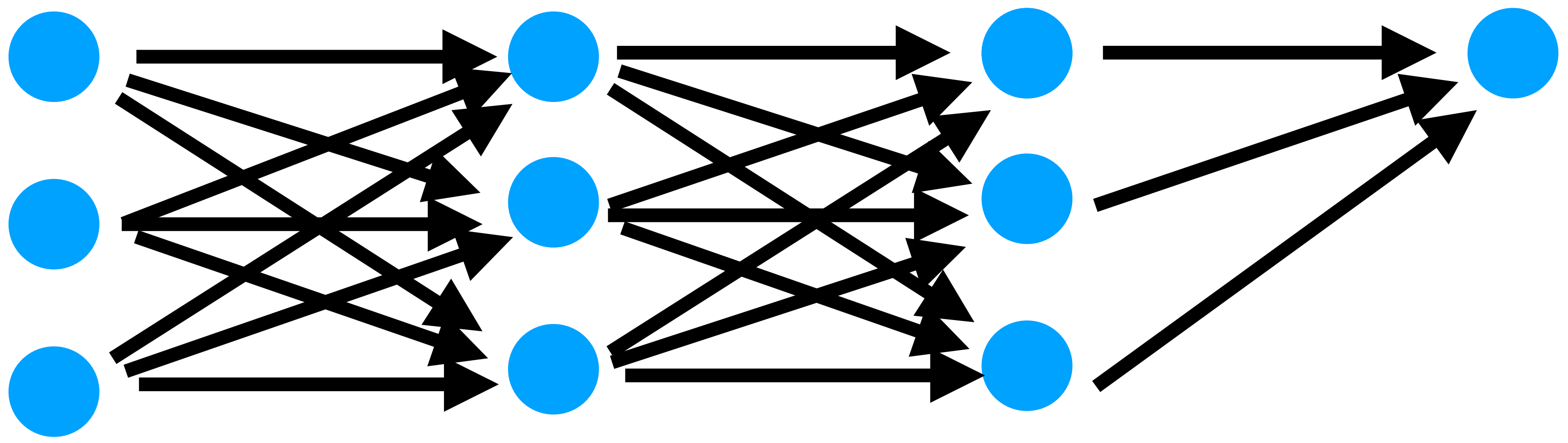
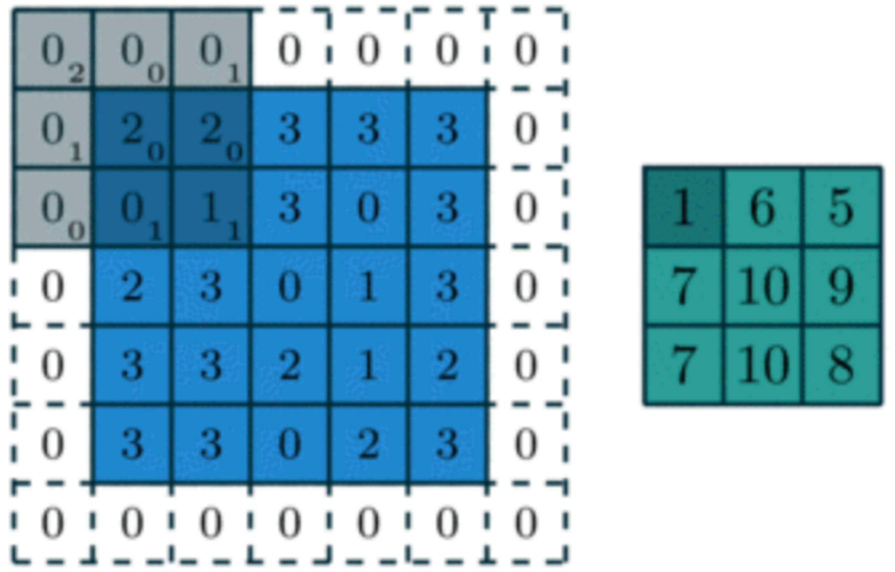


Image Courtesy: Mathworks (online tutorial)

Multi-layer Perceptron vs. CNNs

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

Introduction to CNNs

Kernel-Predicting Denoising

Kernel-Predicting Networks for Denoising Monte-Carlo Renderings

Bako et al. [2017]

Limitations of MLP based Denoiser

Kernel was pre-selected to be joint bilateral filter

- Unable to explicitly capture all details
- lacked flexibility to handle wide range of MC noise in production scenes

Fixed

- can cause unstable weights causing bright ringing and color artifacts

Too many parameters to optimize

Requirements

The function must be flexible to capture complex relationship between input data and reference colors over wide range of scenarios.

Choice of loss function is crucial. Should capture perceptual aspects of the scene.

To avoid overfitting, large dataset required

Using a Vanilla CNN

Denoising a raw, noisy color buffer causes overblurring

- difficulty in distinguishing scene details and MC noise

High dynamic range

- can cause unstable weights causing bright ringing and color artifacts

Vanilla CNN

Ours | Input (32 spp) | Vanilla CNN | Ours | Ref. (1K spp)

UNIVERSITÄT DES SAARLANDES

Denoising Model

$$\widehat{\theta}_p = \underset{\theta}{\operatorname{argmin}}\, \ell(\bar{\mathbf{c}}_p, g(\mathbf{X}_p; \theta))$$

Denoised function with parameters

Reference image

$$\widehat{\mathbf{c}}_p = g(\mathbf{X}_p; \widehat{\theta}_p)$$

Denoised value

$$\ell(\bar{\mathbf{c}}, \widehat{\mathbf{c}})$$

Loss function

Computational Model

$$\widehat{\boldsymbol{\theta}}_p = \underset{\boldsymbol{\theta}}{\operatorname{argmin}} \sum_{q \in \mathcal{N}(p)} \left(\mathbf{c}_q - \boldsymbol{\theta}^\top \phi(\mathbf{x}_q) \right)^2 \omega(\mathbf{x}_p, \mathbf{x}_q)$$

Neighborhood

$$\widehat{\mathbf{c}}_p = g(\mathbf{X}_p; \widehat{\boldsymbol{\theta}}_p)$$

Denoised value

$$\phi : \mathbb{R}^{3+D} \to \mathbb{R}^M$$

$\omega(\mathbf{x}_p, \mathbf{x}_q)$ Kernel weights

$$\widehat{\mathbf{c}}_p = \widehat{\boldsymbol{\theta}}_p^\top \phi(\mathbf{x}_p)$$

Final denoised value

Direct Prediction Network

Direct prediction convolution network: outputs denoised image

$$\widehat{\mathbf{c}}_p = g_{\text{direct}}(\mathbf{X}_p; \boldsymbol{\theta}) = \mathbf{z}_p^L$$

Direct Prediction Network

Direct prediction convolution network: outputs denoised image

$$\widehat{\mathbf{c}}_p = g_{\text{direct}}(\mathbf{X}_p; \boldsymbol{\theta}) = \mathbf{z}_p^L$$

Issues:

The constrained nature and complexity of the problem makes optimization difficult.

The magnitude and variance of stochastic gradients computed during training can be large, which slows convergence of training loss.

Kernel Prediction Network

Kernel prediction convolution network: outputs learned kernel weights

$$w_{pq} = \frac{\exp([\mathbf{z}_p^L]_q)}{\sum_{q' \in \mathcal{N}(p)} \exp([\mathbf{z}_p^L]_{q'})}$$

$$0 \leq w_{pq} \leq 1$$

Softmax activation to enforce weights within range

Denoised color values:

$$\widehat{\mathbf{c}}_p = g_{\text{weighted}}(\mathbf{X}_p; \boldsymbol{\theta}) = \sum_{q \in \mathcal{N}(p)} \mathbf{c}_q w_{pq}$$

Kernel Prediction Network

$$w_{pq} = \frac{\exp([\mathbf{z}_p^L]_q)}{\sum_{q' \in \mathcal{N}(p)} \exp([\mathbf{z}_p^L]_{q'})}$$

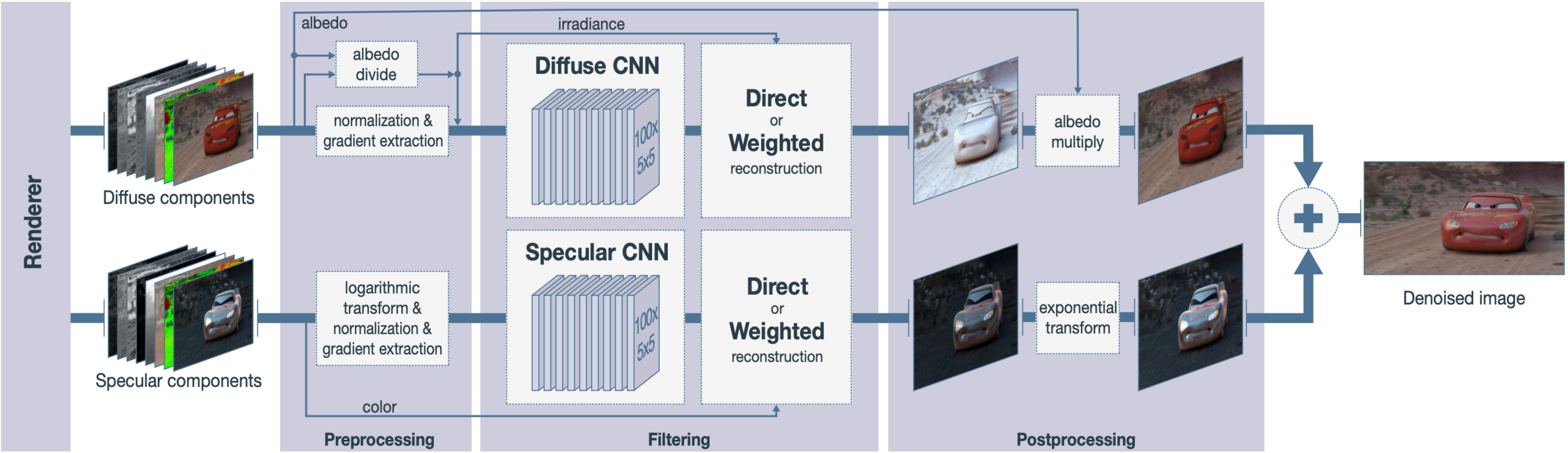
$$0 \leq w_{pq} \leq 1$$

$$\widehat{\mathbf{c}}_p = g_{\text{weighted}}(\mathbf{X}_p; \boldsymbol{\theta}) = \sum_{q \in \mathcal{N}(p)} \mathbf{c}_q w_{pq}$$

Final color estimate always lies within the convex hull of the respective neighborhood (avoid color shifts).

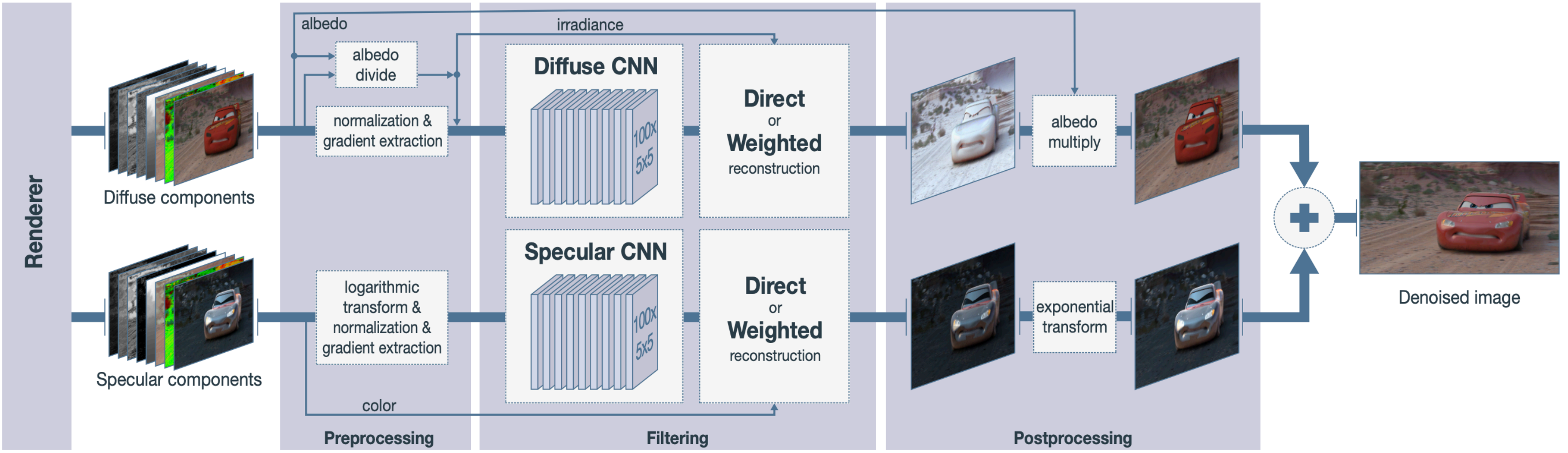
Ensures well-behaved gradients of the error w.r.t the kernel weights

Proposed Architecture



Diffuse/Specular components

Each component is denoised separately



Diffuse components are well-behaved and typically has small ranges

- albedo is factored out to allow large range kernels $\tilde{\mathbf{c}}_{\text{diffuse}} = \mathbf{c}_{\text{diffuse}} \oslash (\mathbf{f}_{\text{albedo}} + \epsilon)$

Specular components are challenging due to high dynamic ranges: uses logarithmic transform

$$\tilde{\mathbf{c}}_{\text{specular}} = \log(1 + \mathbf{c}_{\text{specular}})$$


Training Dataset: 600 frames

Training

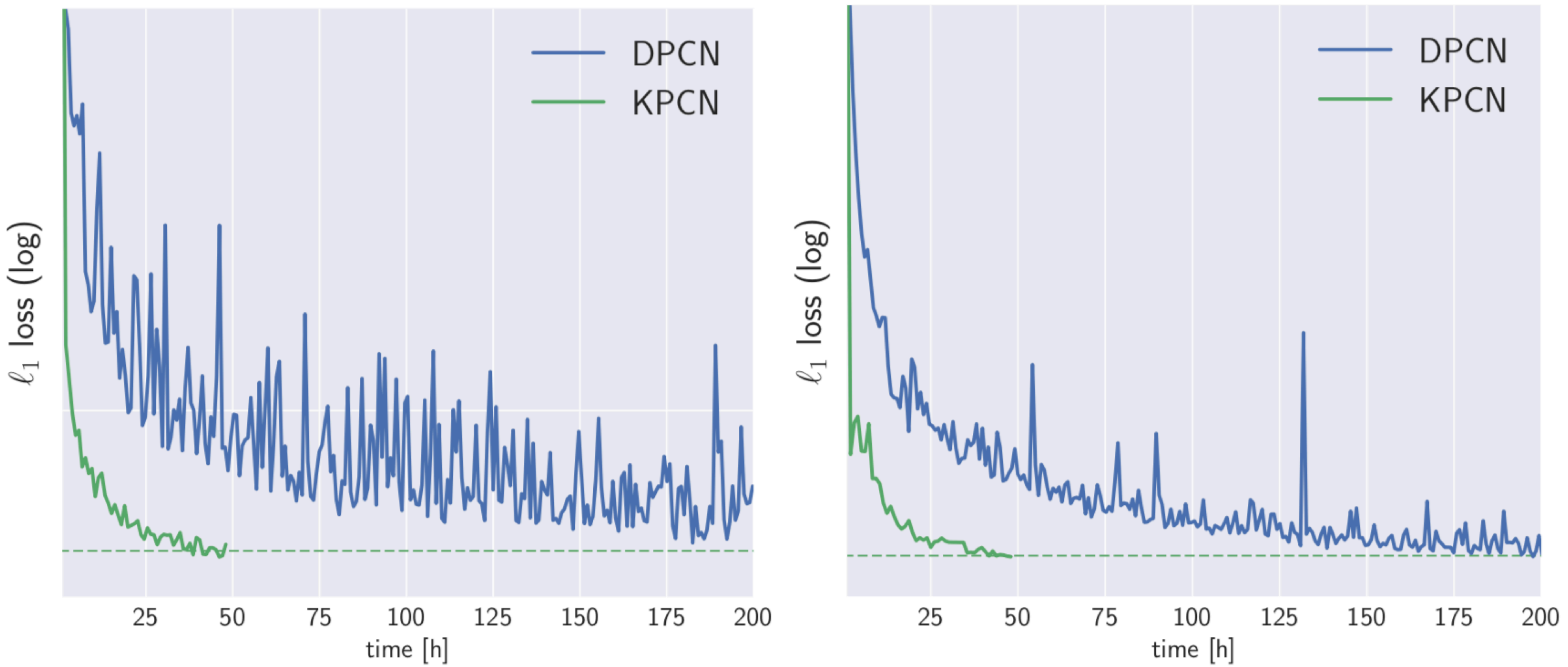
8-hidden layers used with 100 kernels of 5x5 in each layer for each network

For KPCN (kernel-predicting network), output kernel size used = 21

Weights for 128 app and 32 spp networks were initialized using Xavier method

Diffuse and specular components were independently trained with L1 loss metric

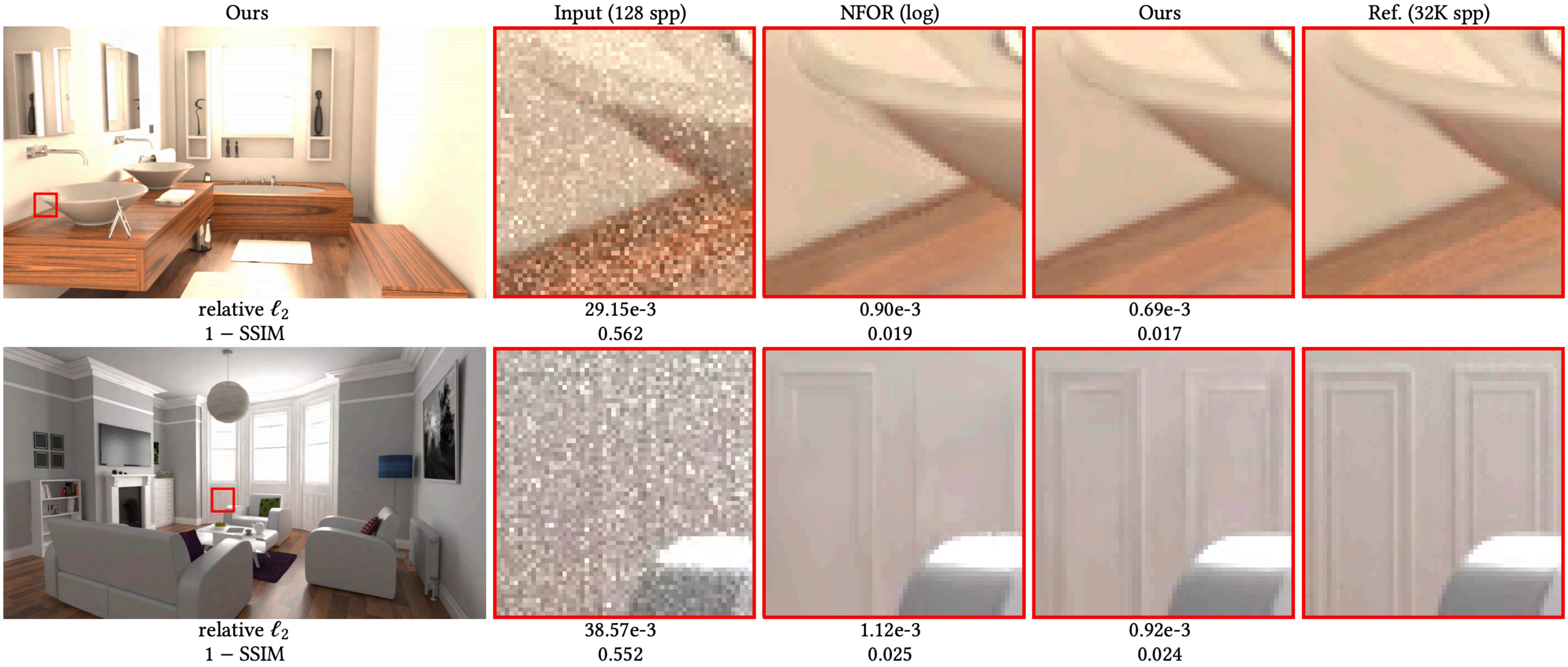
Learning rate of DPCN vs. KPCN



(a) Diffuse (b) Specular

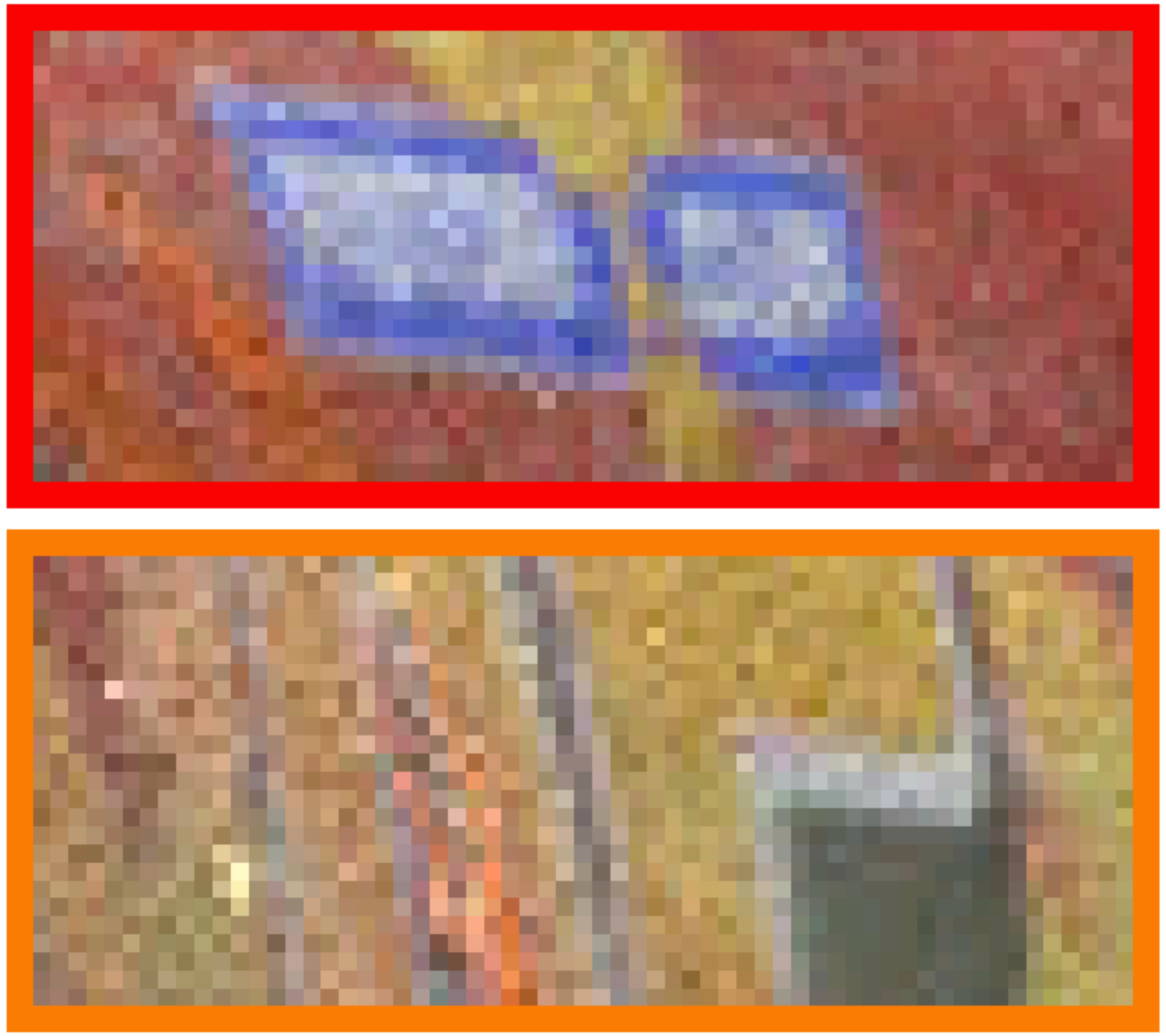
On Cars 3 dataset, KPCN converges 5-6x faster

Results



Ours
Input (128 spp)
NFOR (log)
Ours
Ref. (32K spp)
relative ℓ_2
$1 - \mathrm{SSIM}$
29.15e-3
0.562
0.90e-3
0.019
0.69e-3
0.017
relative ℓ_2
$1 - \mathrm{SSIM}$
38.57e-3
0.552
1.12e-3
0.025
0.92e-3
0.024

Results

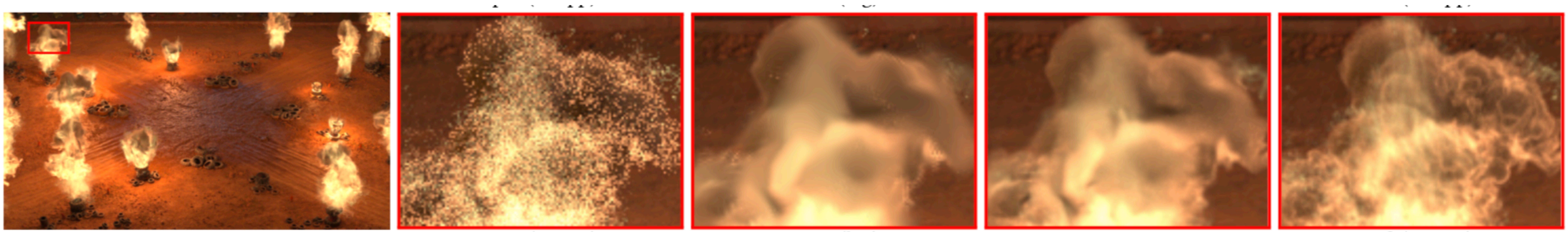


Input (32 spp)

Results

w/o Decomposition, w/o Albedo divide | w/ Decomposition, w/o Albedo divide | w/o Decomposition, w/ Albedo divide | w/ Decomposition, w/ Albedo divide | Ref. (2K spp)

Results



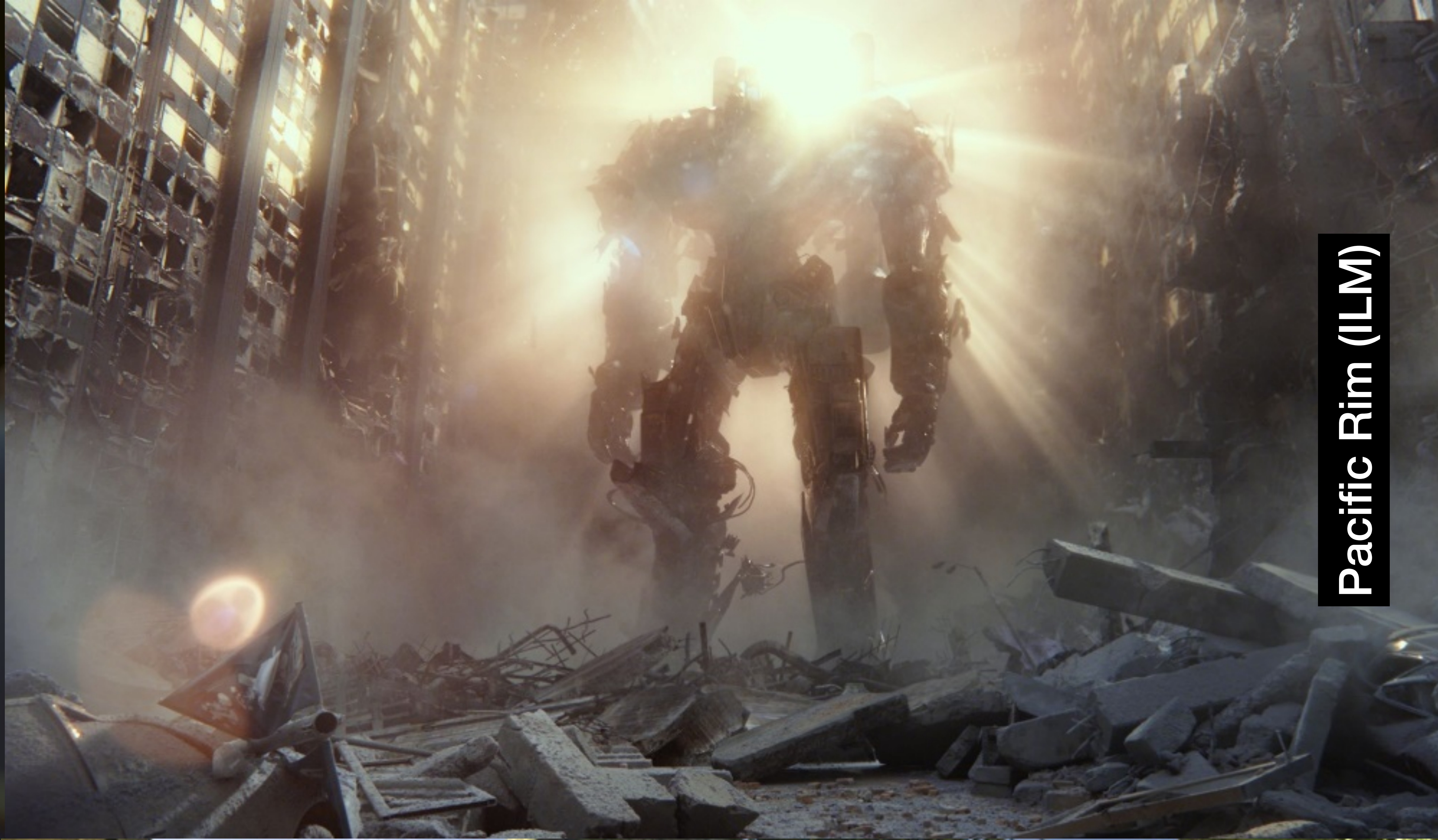
Ours | Input (32 spp) | NFOR (log) | Ours | Ref. (8K spp)

Ours | Input (32 spp) | Ours | Ref. (1K spp)

Also works on Piper short movie frames

Interactive Reconstruction of Monte Carlo Sequences

Chaitanya et al. [2017]


Toy Story (Pixar)

Pacific Rim (ILM)

Crisis 3 (Crytek)

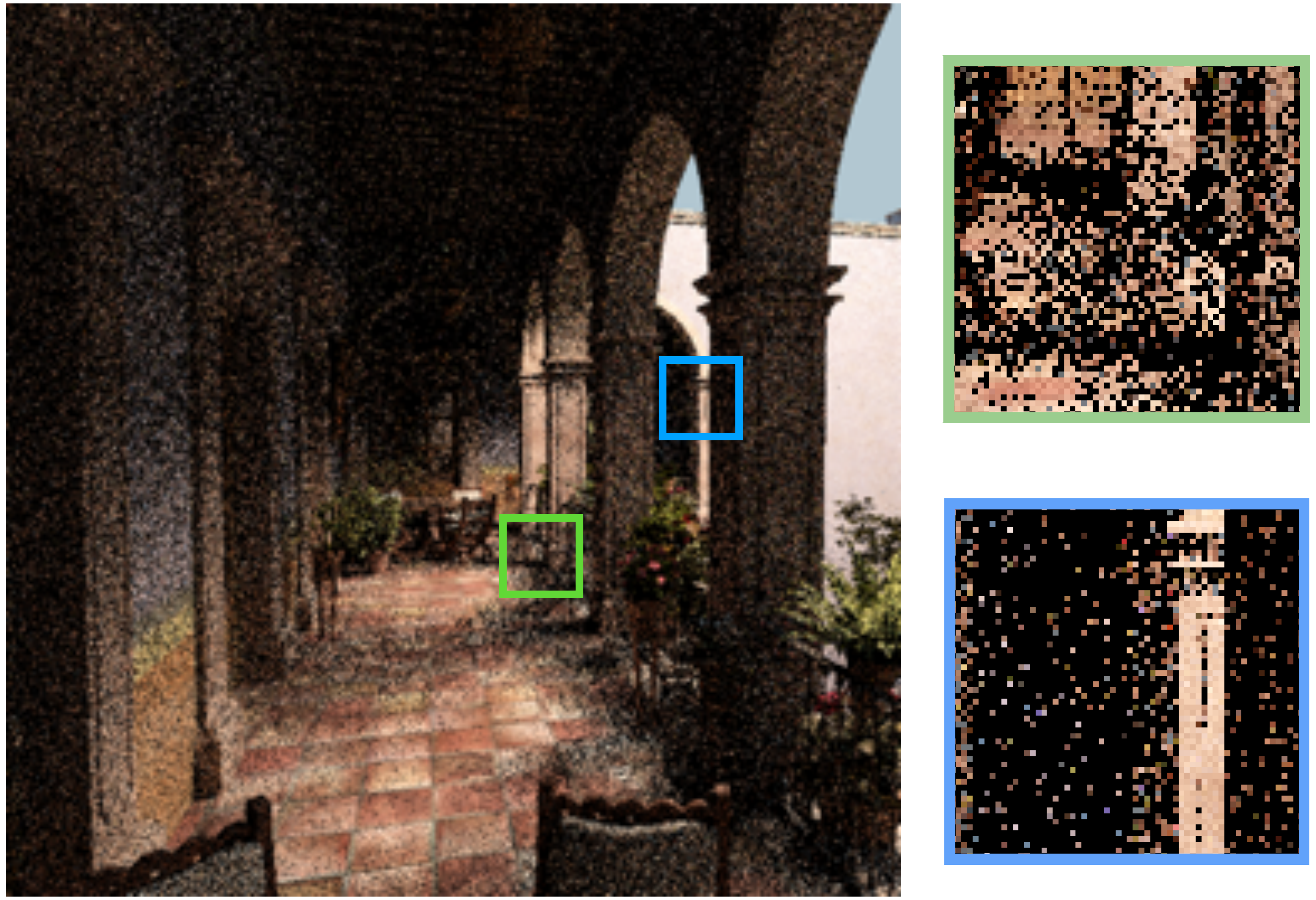
Halo 3 (Bungie)

Motivation: Interactive Reconstruction

Limited to a few rays per pixel @ 1080p @ 30Hz

Never enough to reconstruct an image

Deep learning approach for interactive graphics

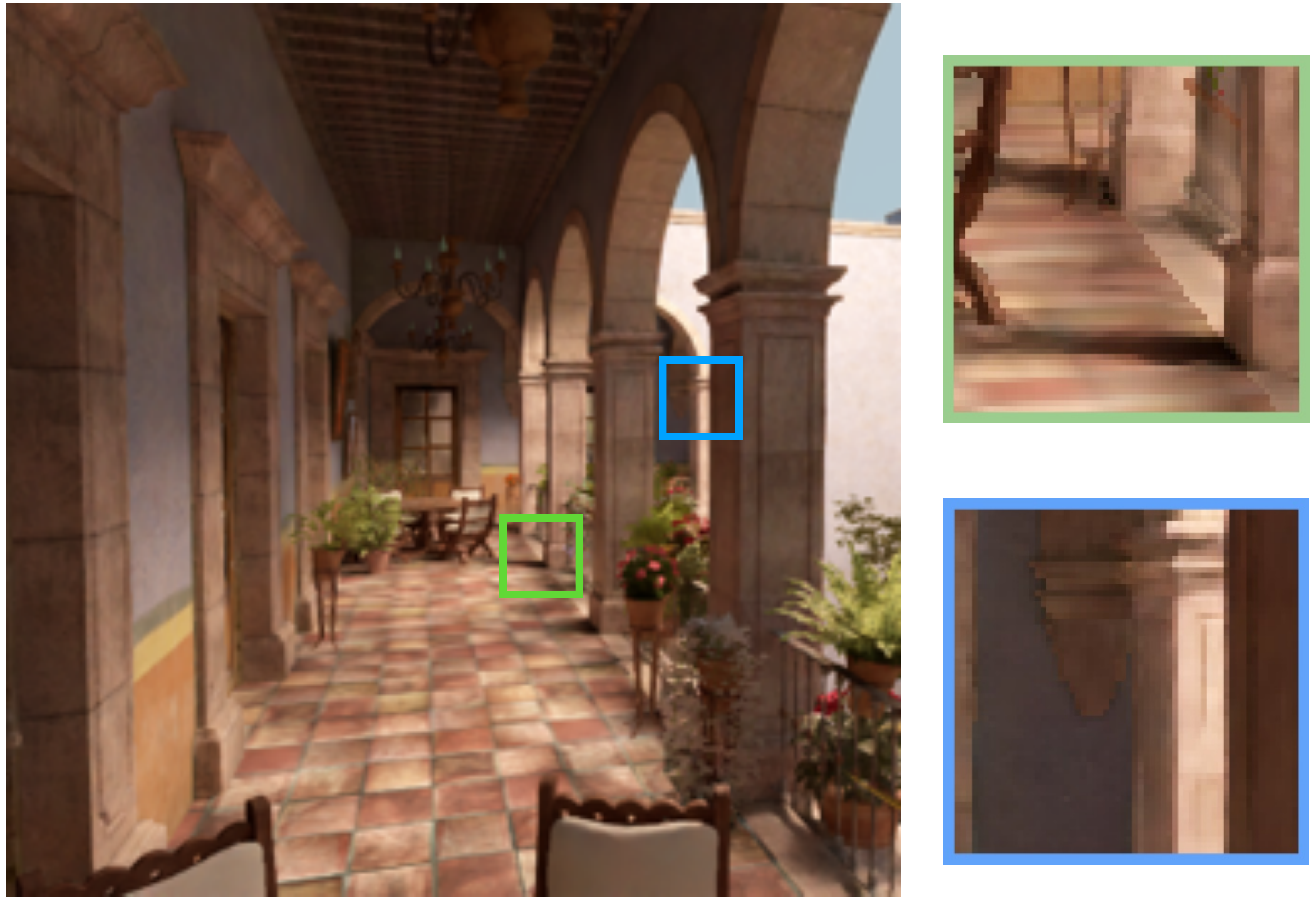


Motivation: Interactive Reconstruction

Limited to a few rays per pixel @ 1080p @ 30Hz

Never enough to reconstruct an image

Deep learning approach for interactive graphics

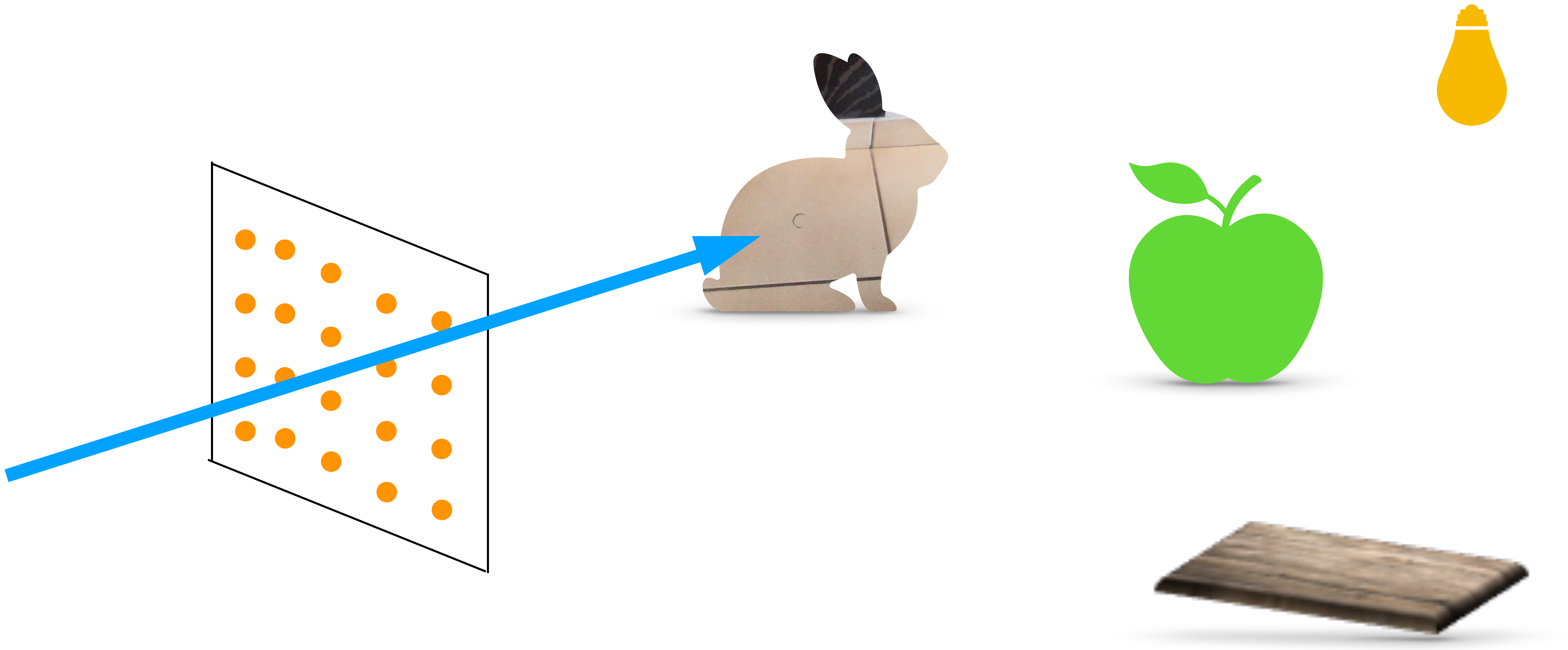


Problem Statement

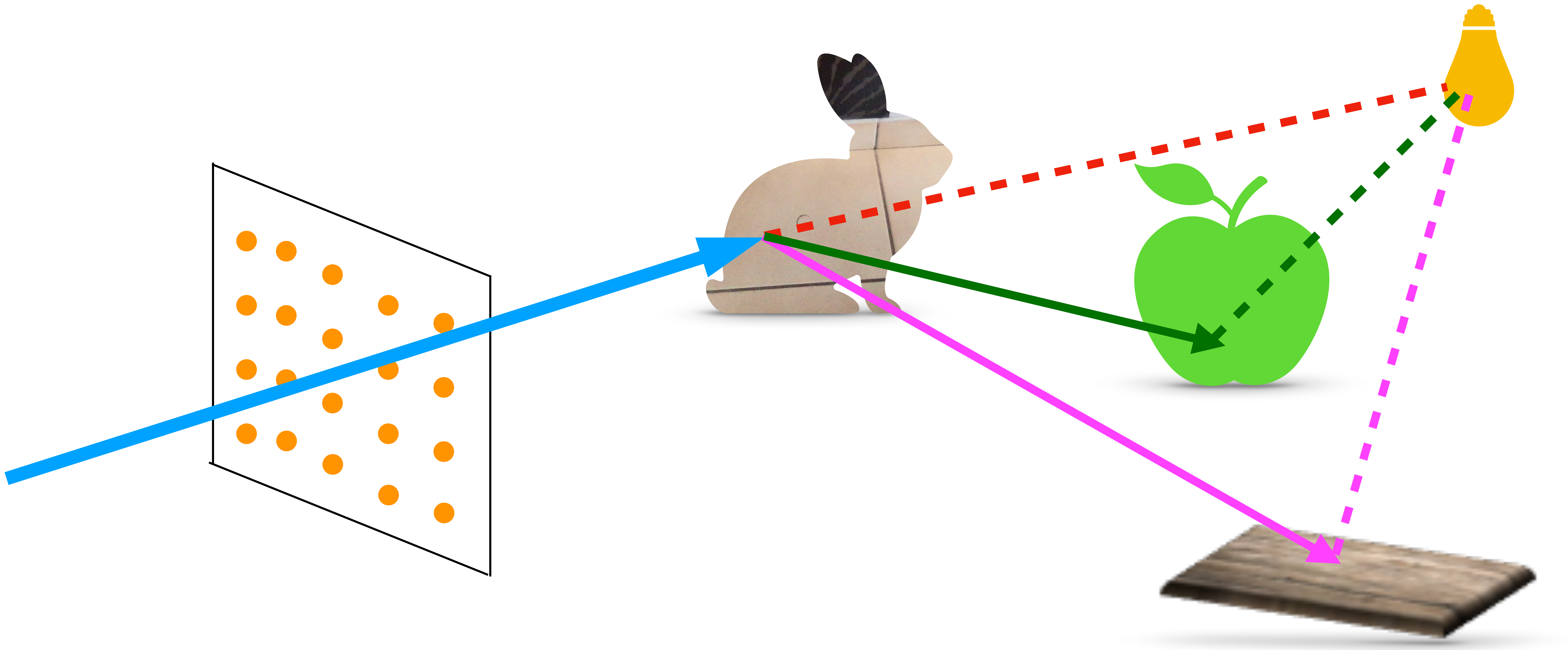
Handle generic effects:

- Soft shadows
- Diffuse and specular reflections
- Global illumination (one-bounce)
- No Motion blur or depth of field

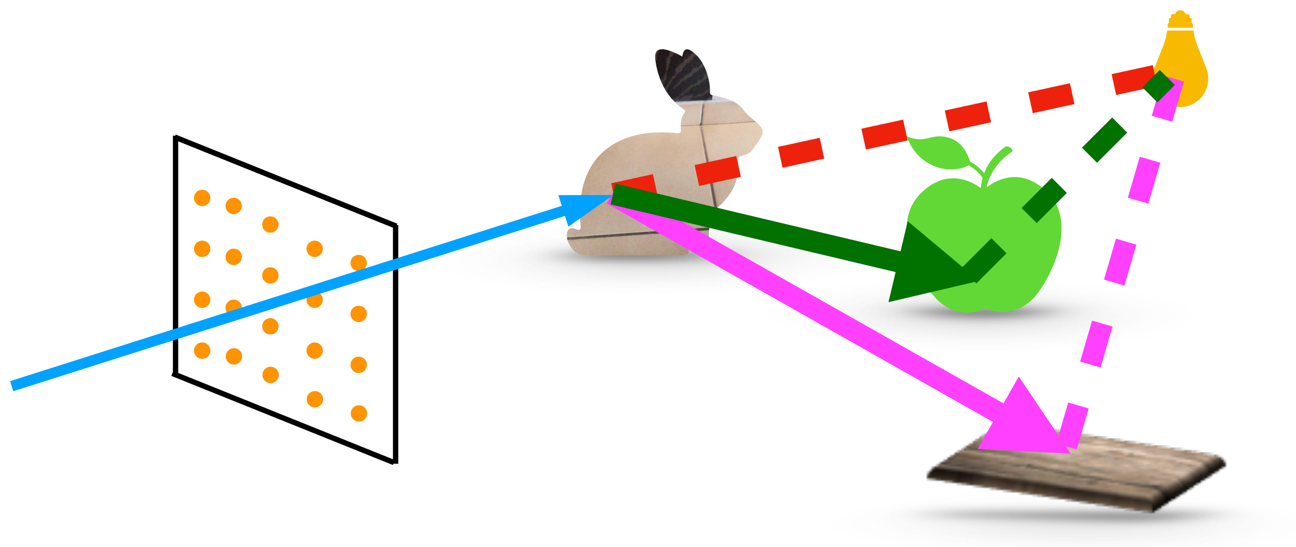
System setup: Path tracing



System setup: Path tracing



System setup: Path tracing



Rasterize primary hits in G-buffers

Path-tracing from the primary paths

- 1 ray for direct shadows
- 2 rays for indirect (sample + connect)

1 direct + 1 indirect path (spp)

Denoising Autoencoder (DAE)

Recurrent Autoencoder [Chaitanya et al. 2017]

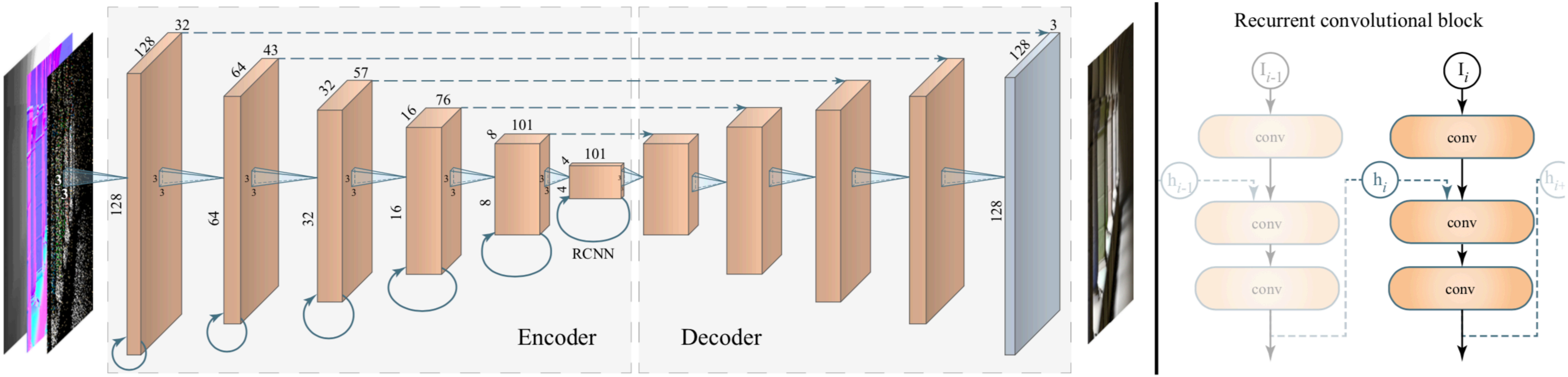
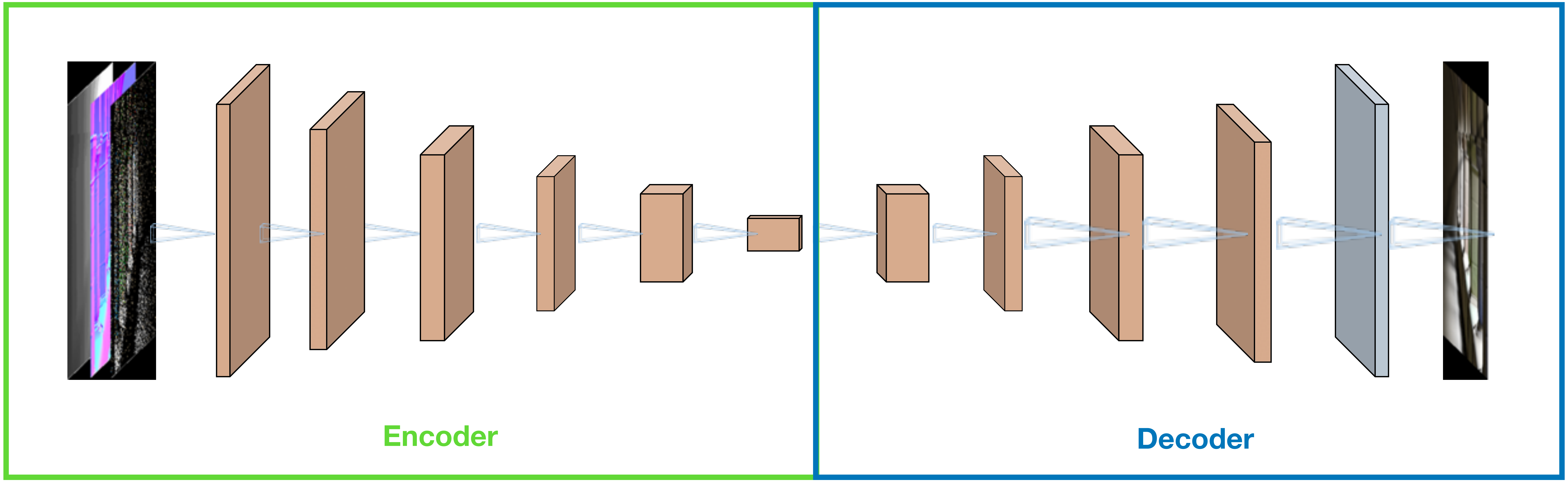


Fig. 2. Architecture of our recurrent autoencoder. The input is 7 scalar values per pixel (noisy RGB, normal vector, depth, roughness). Each encoder stage has a convolution and 2×2 max pooling. A decoder stage applies a 2×2 nearest neighbor upsampling, concatenates the per-pixel feature maps from a skip 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.

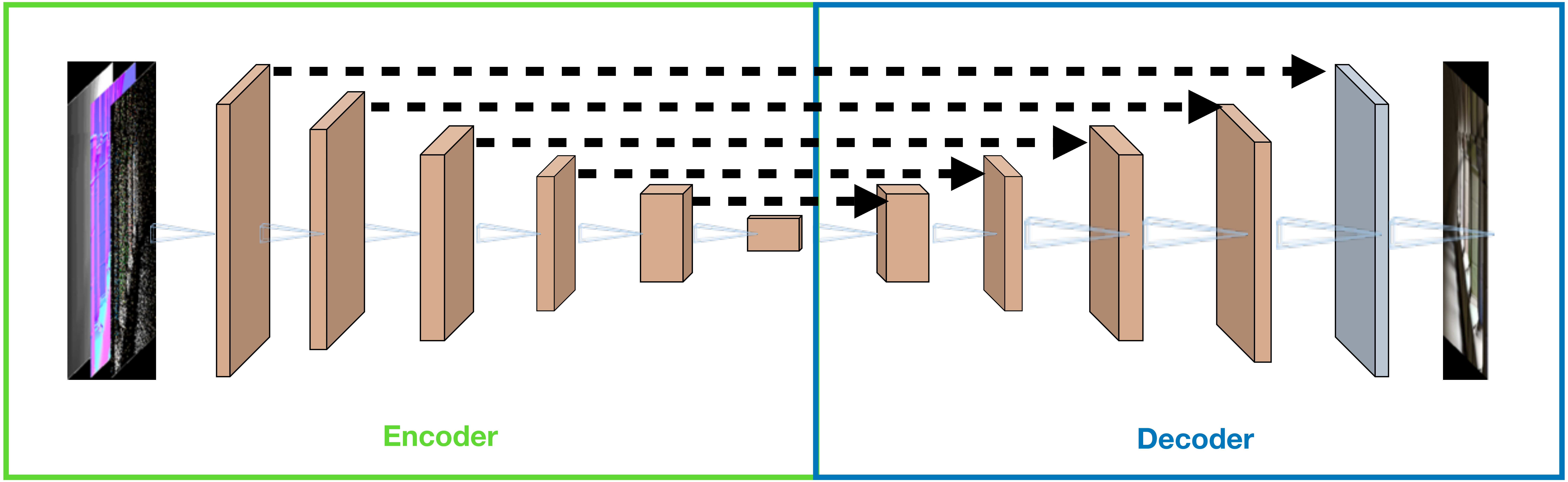
Recurrent Neural Networks

Encoder and decoder stages for dimensionality reduction



Recurrent Neural Networks

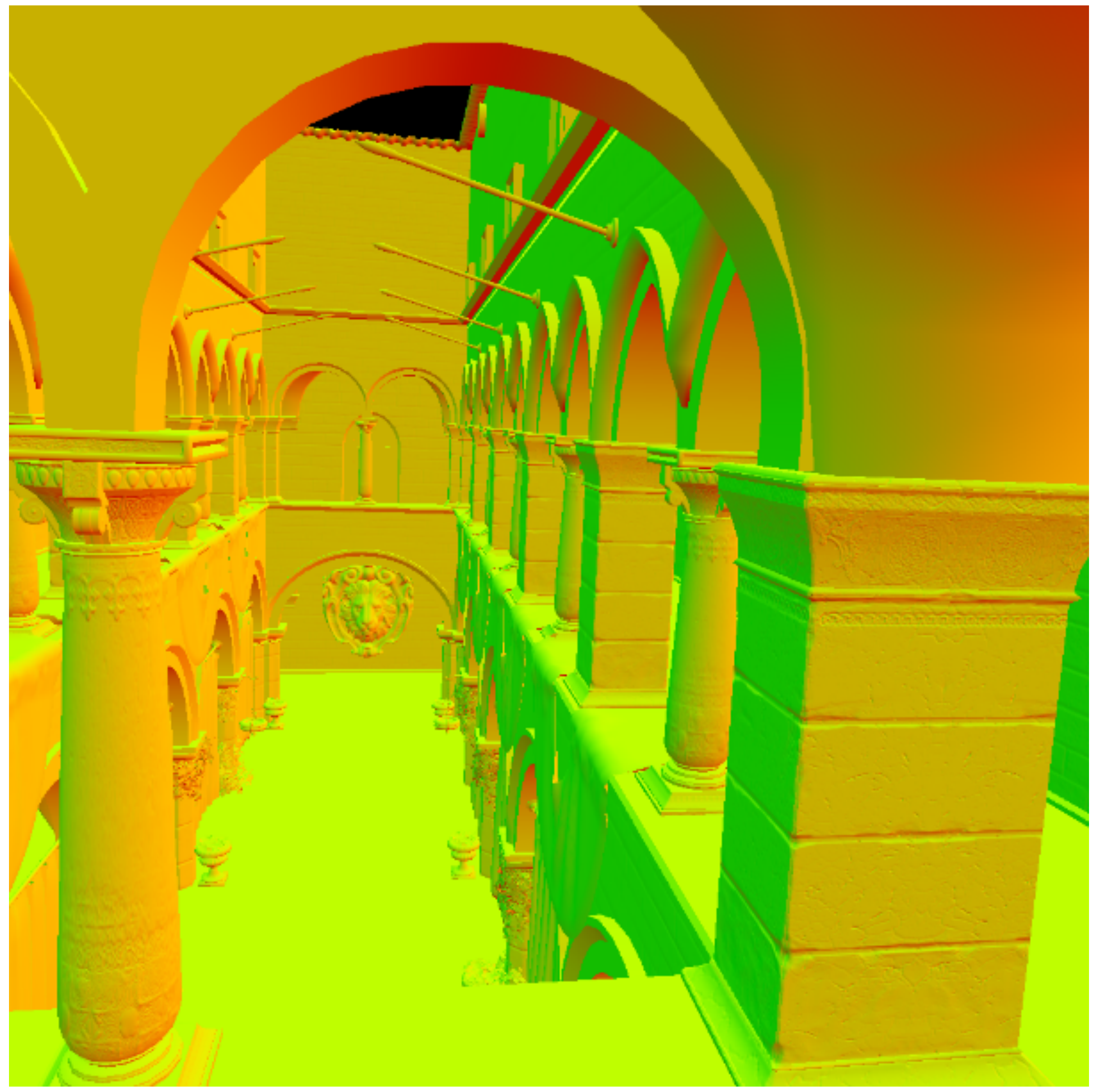
Encoder and decoder stages for dimensionality reduction

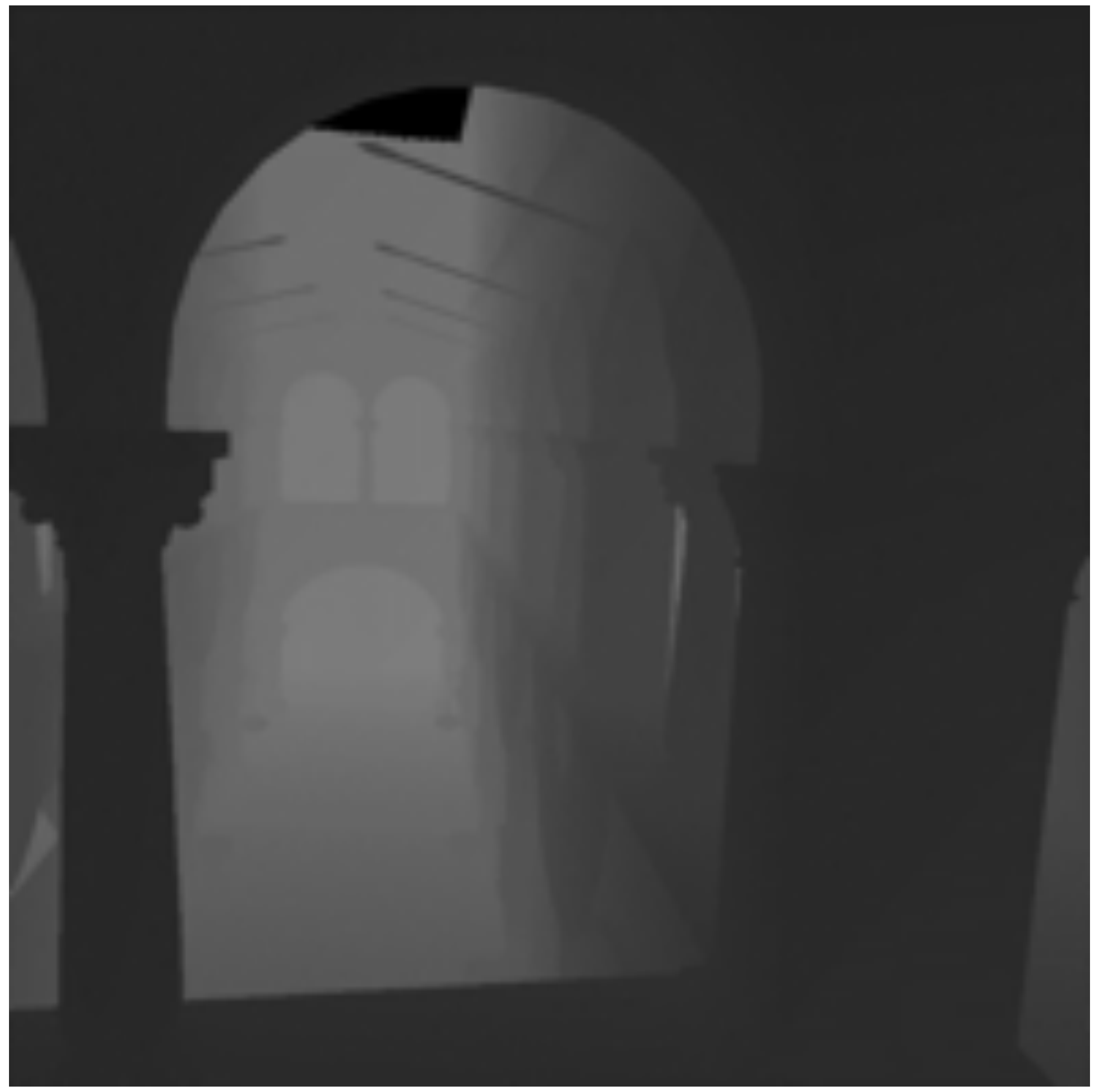


Skip connections to reintroduce lost information

Auxillary Features


Untextured color


View space normals


Linearize depth

Training sequences

SponzaDiffuse

SponzaGlossy

Classroom

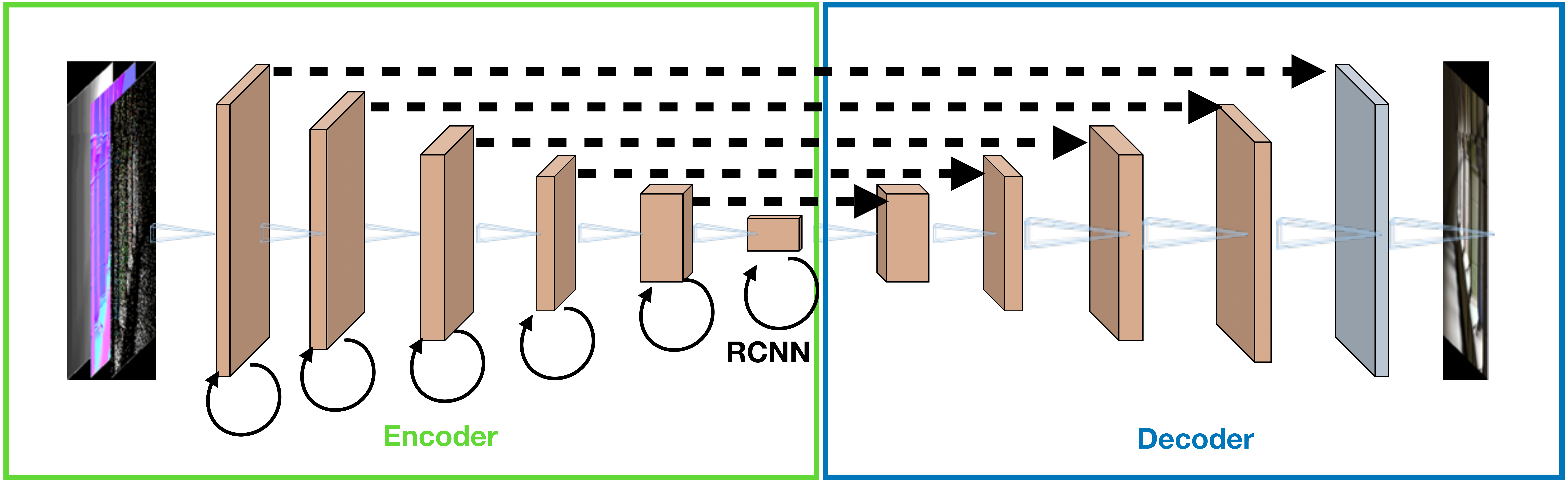

1spp approx. 70 ms


DAE 1spp
approx. 70 ms + approx. 60 ms


Reference 1024 spp
approx. 240 ms

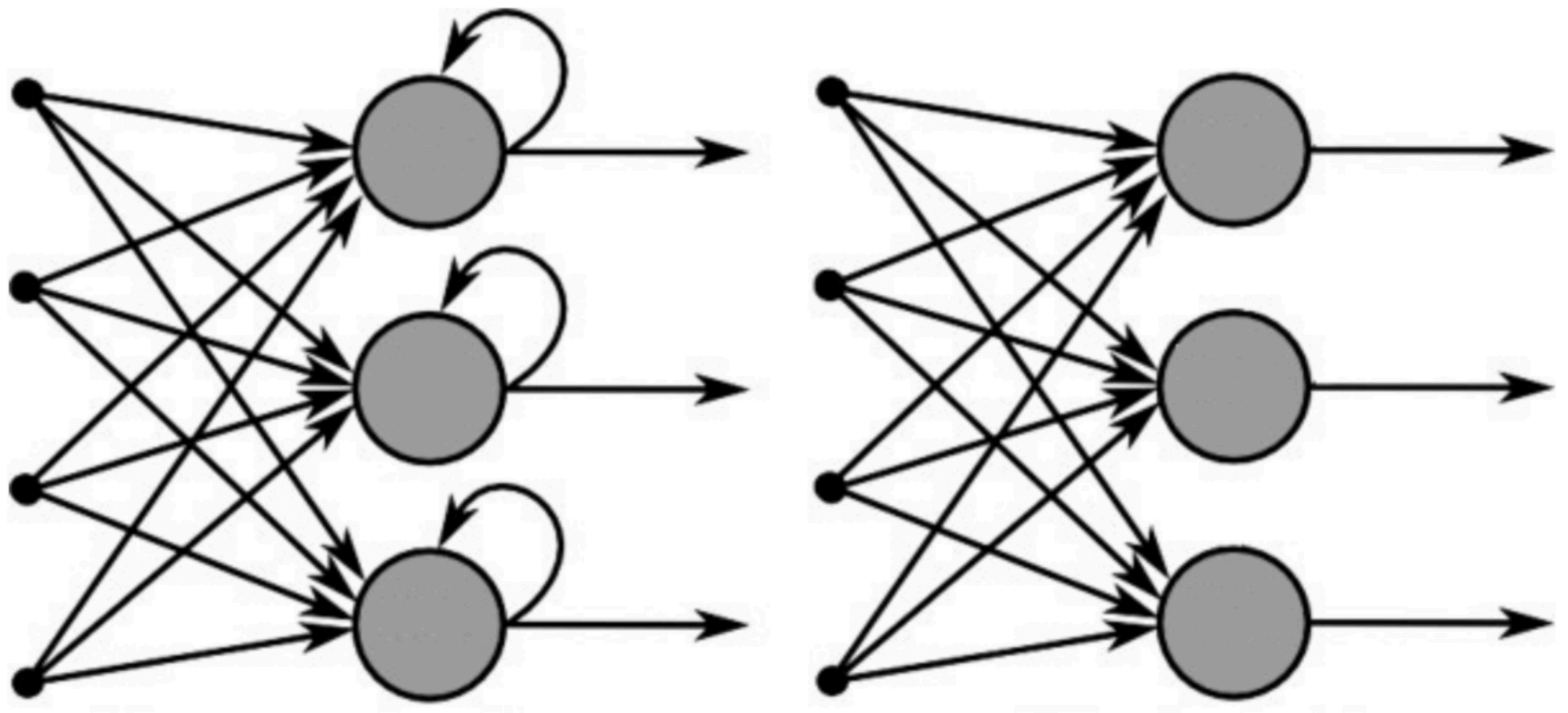
Recurrent Denoising Autoencoder

Feedback loops to retain important information after every encoding stage



Recurrent Neural Networks vs. Simple Feed-Forward NN

Source link

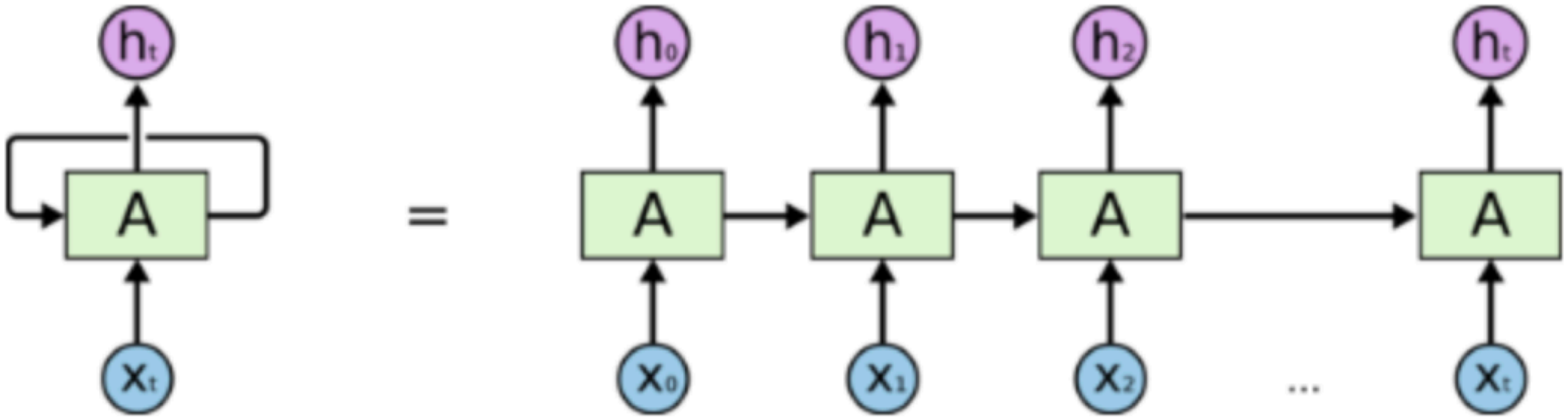


Recurrent Neural Network

Feed-Forward Neural Network

Recurrent Neural Networks

Source link



An unrolled recurrent neural network.

Recurrent Neural Networks

Source link

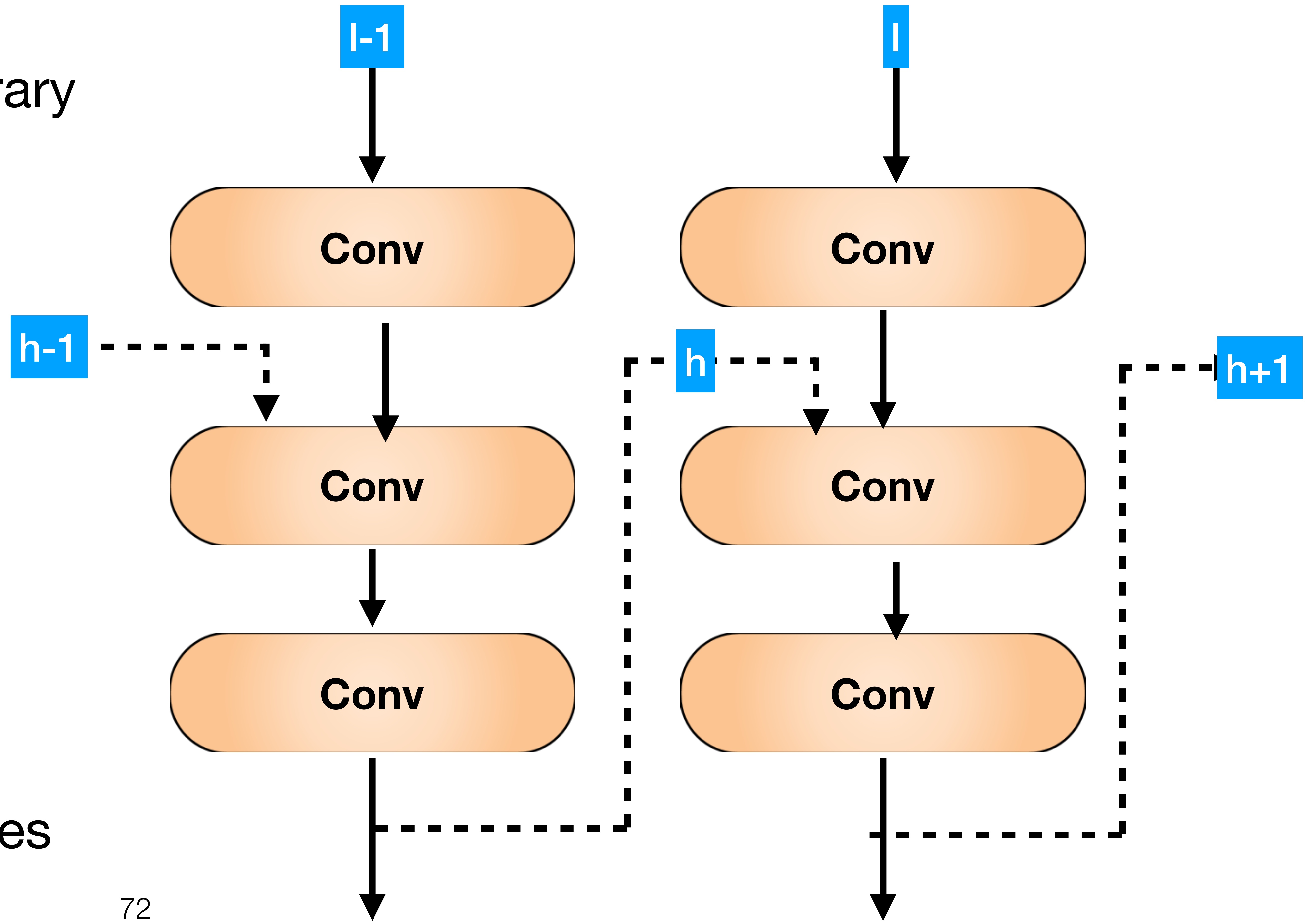
Fully convolutional blocks to support arbitrary image resolution

6 RNN blocks, one per pool layer in the encoder

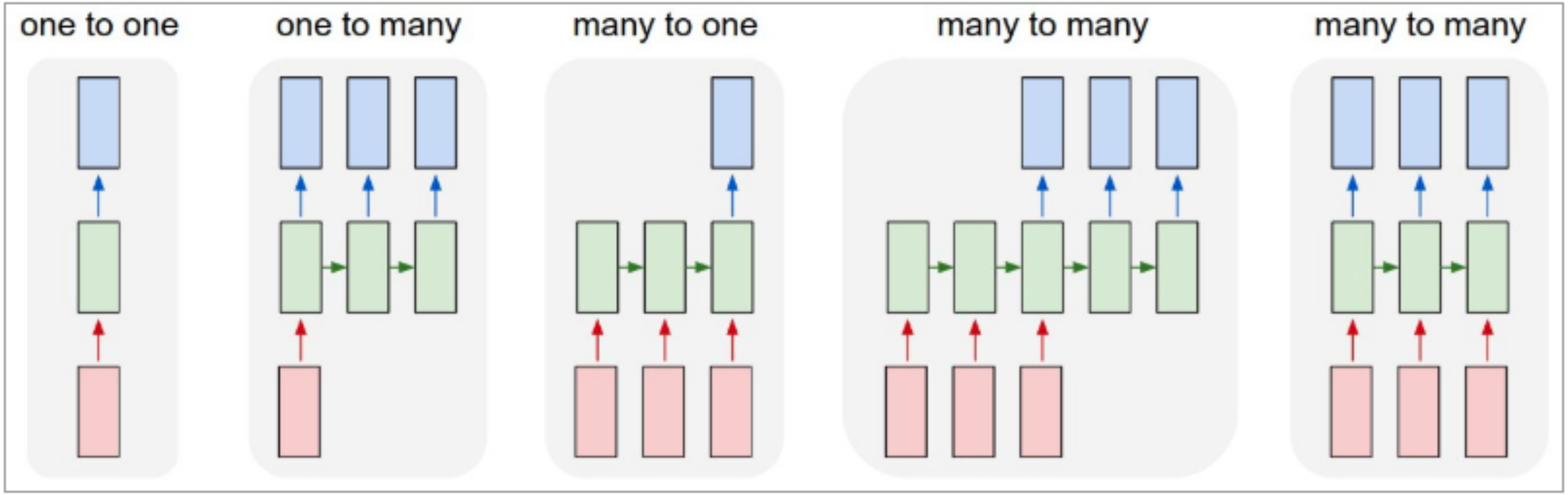
Design:

- 1 conv layer (3x3) for current features

- 2 conv layers (3x3) for previous features

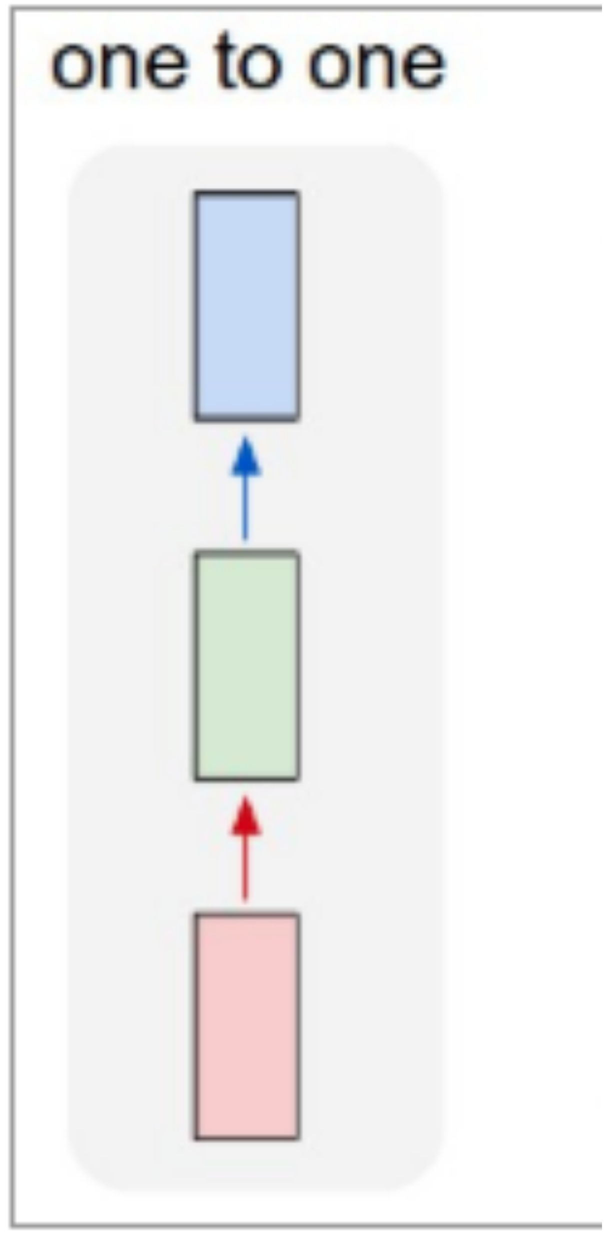


Recurrent Neural Networks



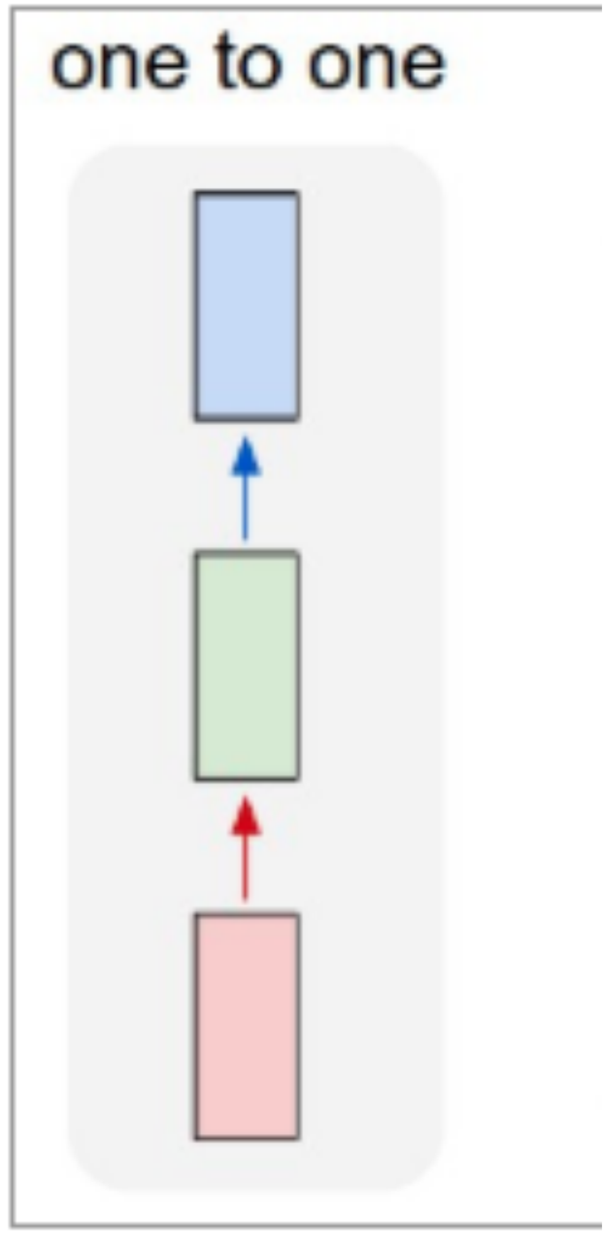
Recurrent Neural Networks

CNNs,
fixed input,
fixed output



Recurrent Neural Networks

CNNs,
fixed input,
fixed output



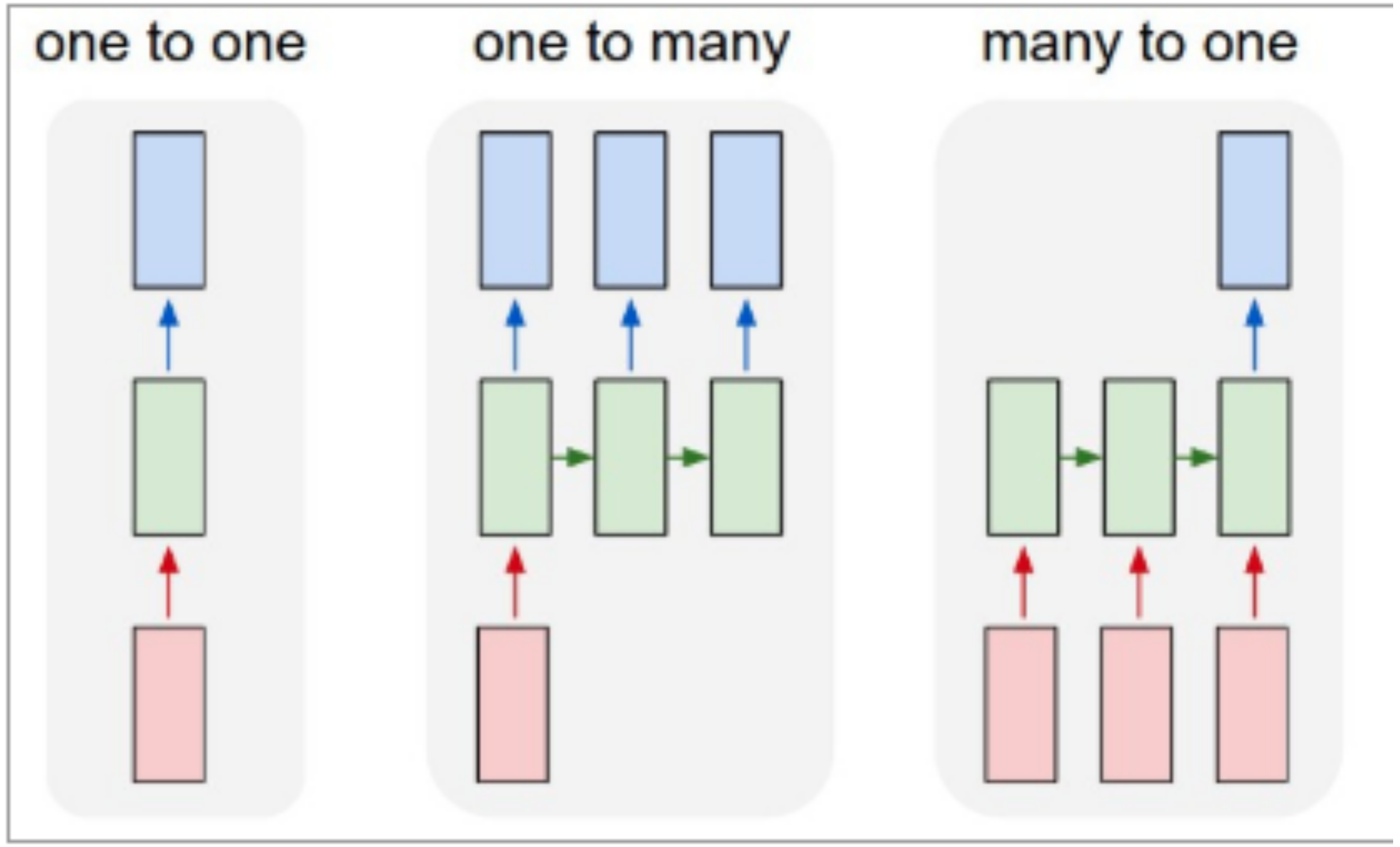
Se

e.g., image captioning takes an image as input and outputs a sentence of words

Recurrent Neural Networks

CNNs,
fixed input,
fixed output

Sequence input



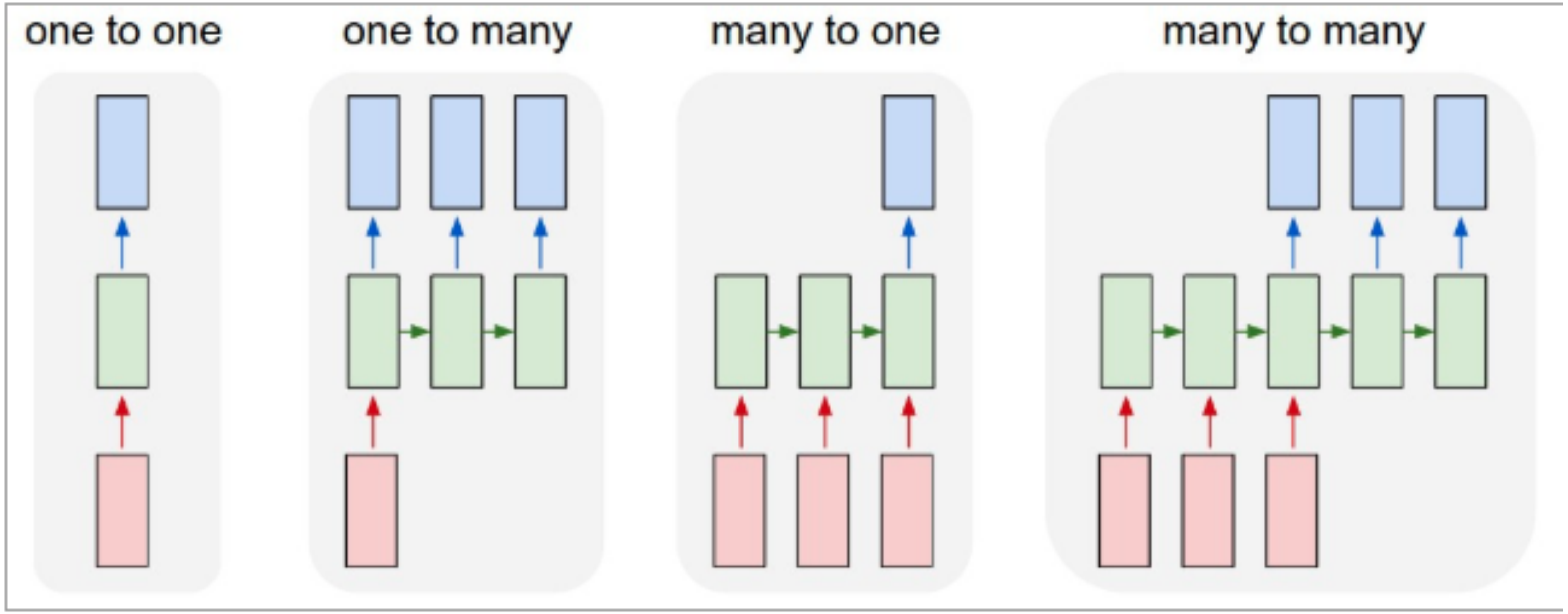
Sequence output

e.g., to know the sentiments of a sentence

Recurrent Neural Networks

CNNs,
fixed input,
fixed output

Sequence input



Sequence output

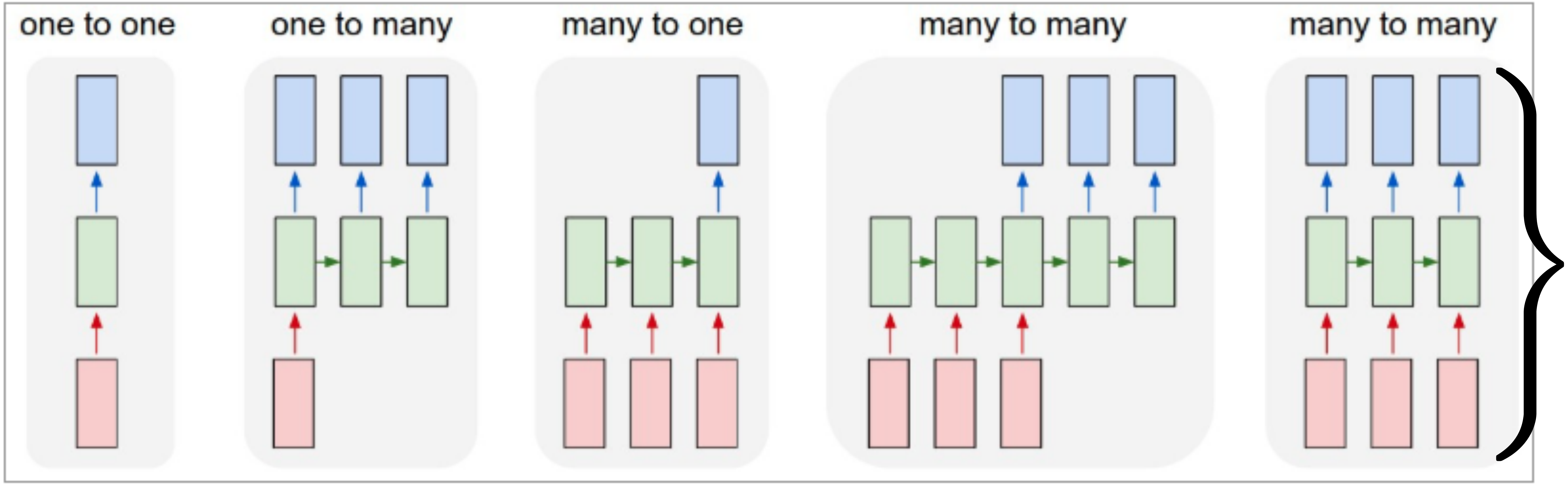
Sequence input,
Sequence output.
e.g. Machine translation

Recurrent Neural Networks

CNNs,
fixed input,
fixed output

Sequence input

Synced sequence
Input & output



e.g., video classification where we want to label each frame

Sequence output

Sequence input,
Sequence output.
e.g. Machine translation

Training

Input is a sequence of 7 frames

128x128 random image crop per sequence

Play the sequence forward/backward

Each frame advance the camera or random seed

Loss Functions

Spatial Loss to emphasize more the dark regions

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

Loss Functions

Spatial Loss to emphasize more the dark regions

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

Temporal loss

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

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

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

Loss Functions

Spatial Loss to emphasize more the dark regions

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

Temporal loss

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

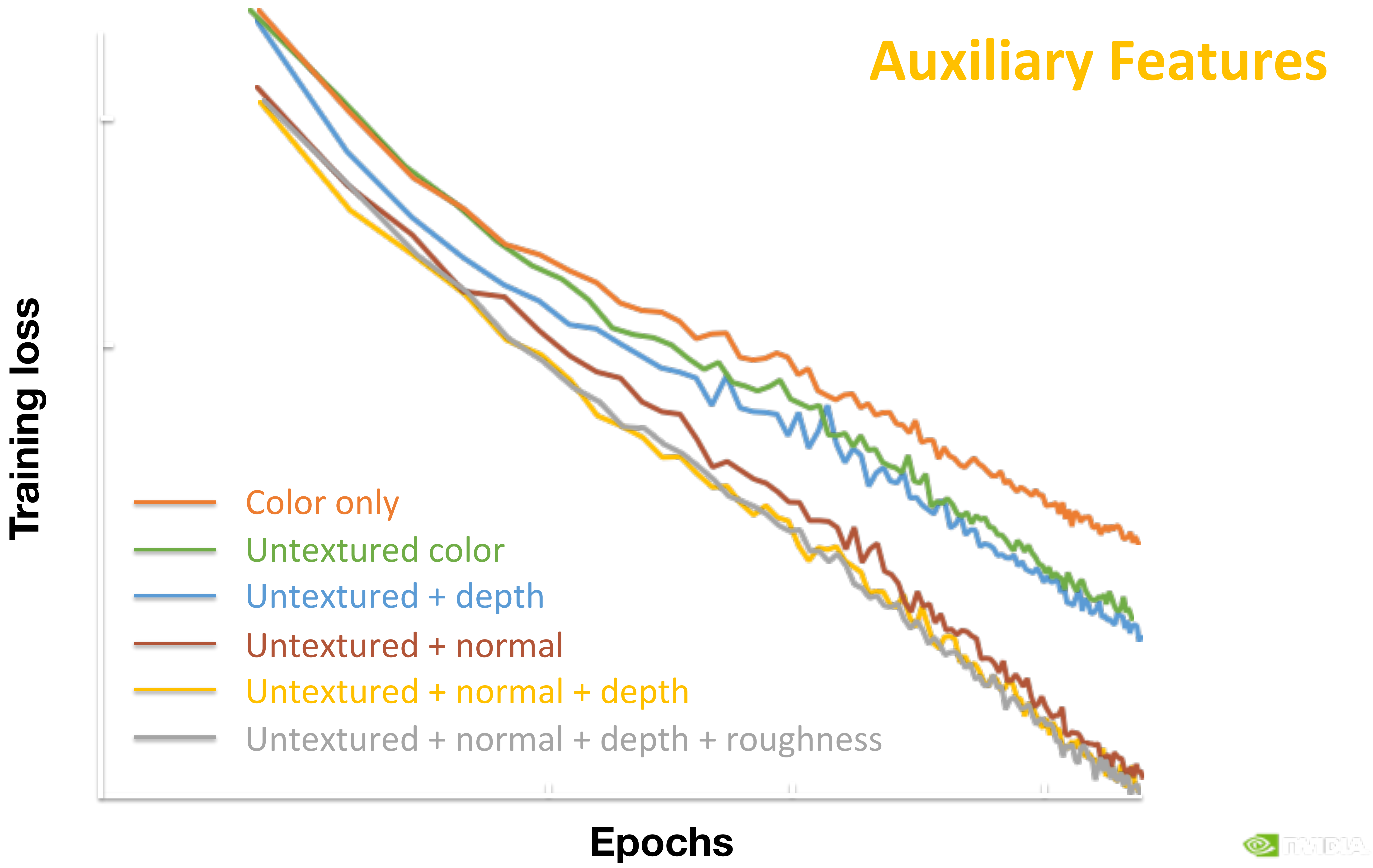
High frequency error norm loss for stable edges

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

Final Loss is a weighted averaged of above losses

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

Training Loss depends on Auxiliary Features



Temporal Stability


Recurrent autoencoder
with temporal AA
Recurrent autoencoder
Autoencoder
with skips


1 sample/pixel input


Recurrent autoencoder
1 sample/pixel input

Introduction to CNNs

Kernel Predicting Denoising

Sample-based MC Denoising

(next lecture)

Acknowledgments

Thanks to Chaitanya and colleagues for making their slides publicly available.