Learning Recursive Filters for Low-Level Vision via a Hybrid Neural Network

Sifei Liu[1] Jinshan Pan[12] Ming-Hsuan Yang[1]

[1]University of California at Merced [2]Dalian University of Technology

Introduction

- Learning recursive filters
  - An important type of filter in signal processing
  - Estimating the coefficients of recursive filters
    - Various optimization methods in frequency/temporal domain
- Applications for computer vision
  - Image filtering, denoising, inpainting, color interpolation, etc.

Introduction

- Learning recursive filters
  - An important type of filter in signal processing
  - Estimating the coefficients of recursive filters
    - Various optimization methods in frequency/temporal domain
    - Deep neural network?

- Applications for computer vision
  - Image filtering, denoising, inpainting, color interpolation, etc.

Low-Level Vision Problems: Filtering

Low-Level Vision Problems: Enhancement

Low-Level Vision Problems: Image Denoising

Low-Level Vision Problems: Image Inpainting

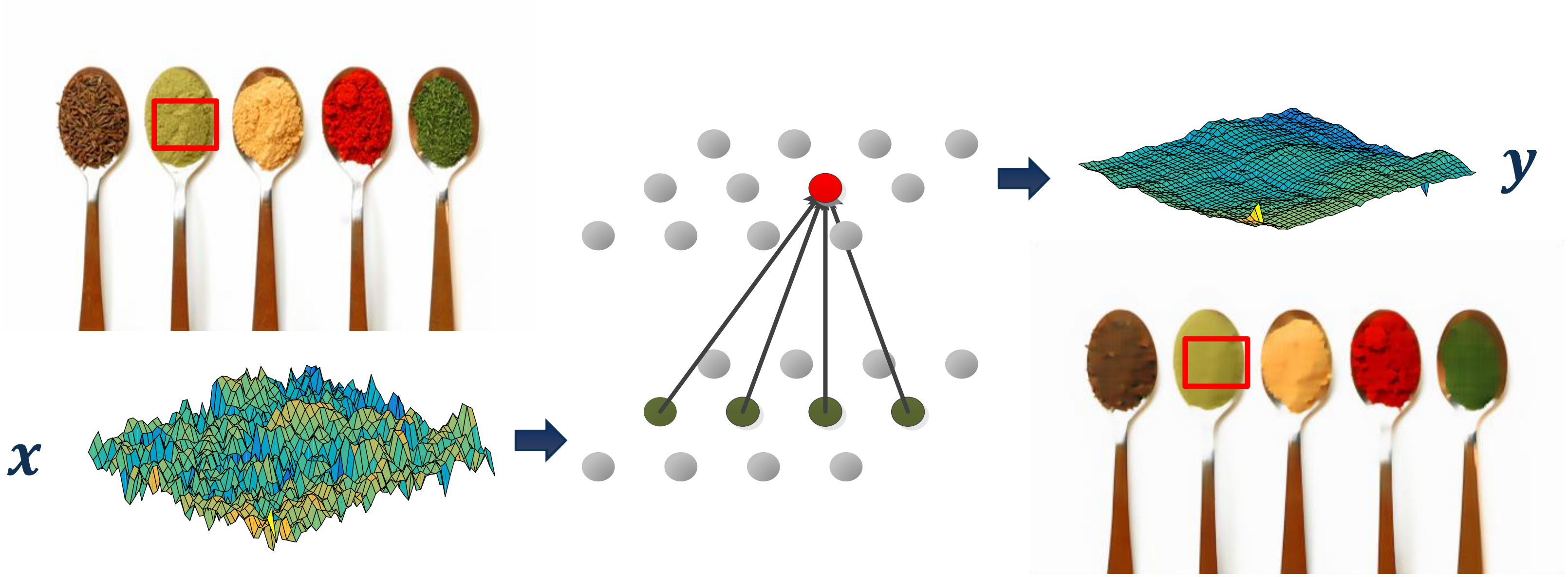
Low-Level Vision Problems: Color Interpolation

Contributions

- A general framework:
  - ➢ Convolutional + recurrent networks (CNN + RNN)
- Small model
- Real-time on QVGA (320×240) images

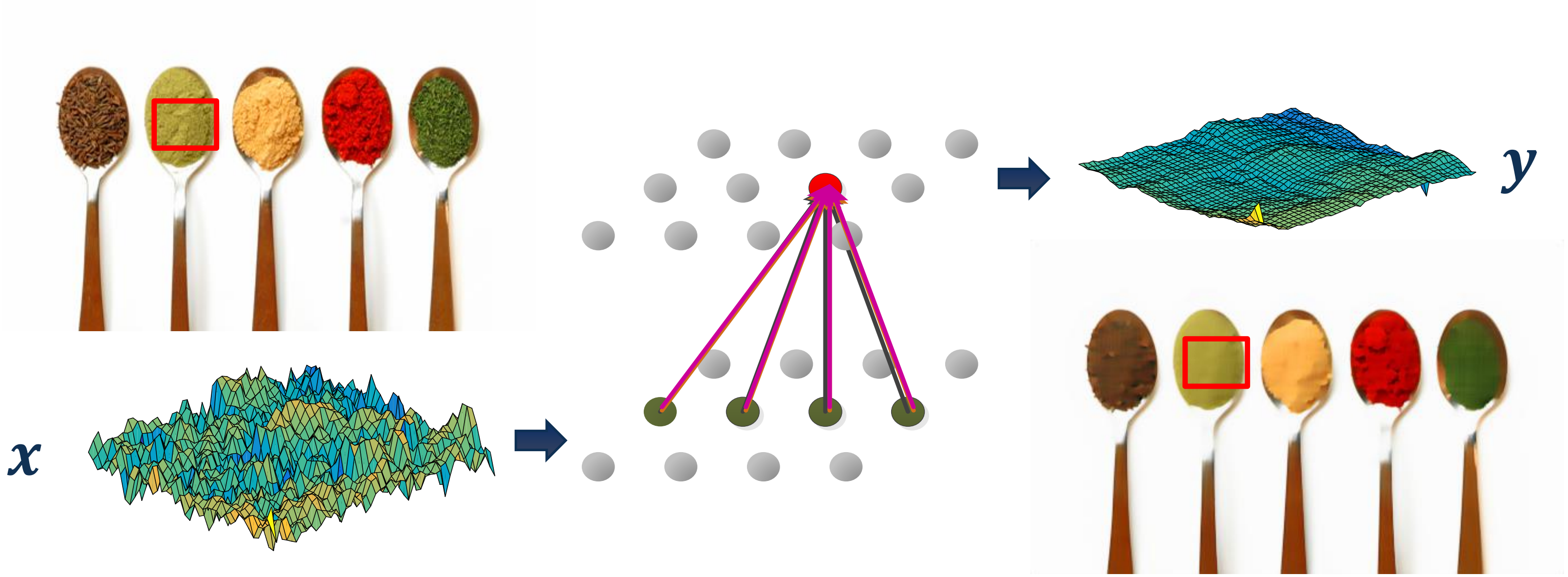
Convolutional Filter



$$y[k] = \sum_{i=-M}^{M} a_i x[k-i]$$

- ✓ Easy to design
- × Large number of parameters
- × Many groups of filters

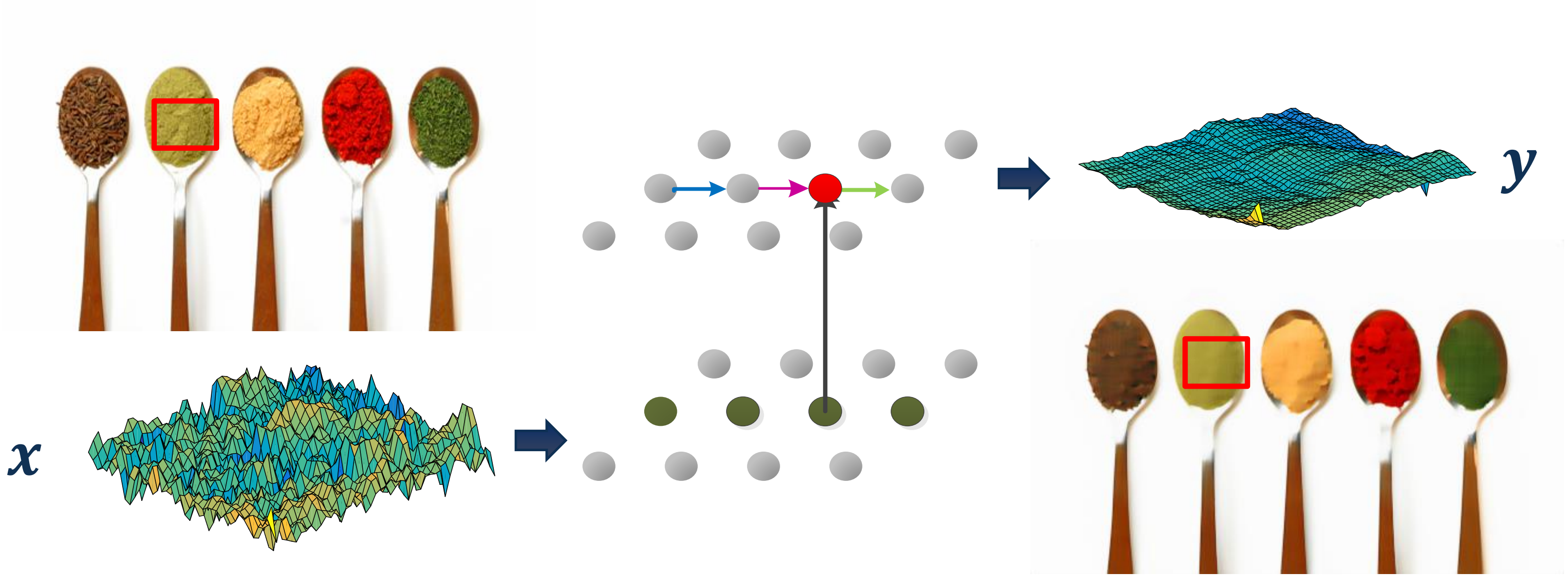
Convolutional Filter



$$y[k] = \sum_{i=-M}^{M} a_i x[k-i]$$

- ✓ Easy to design
- × Large number of parameters
- × Many groups of filters

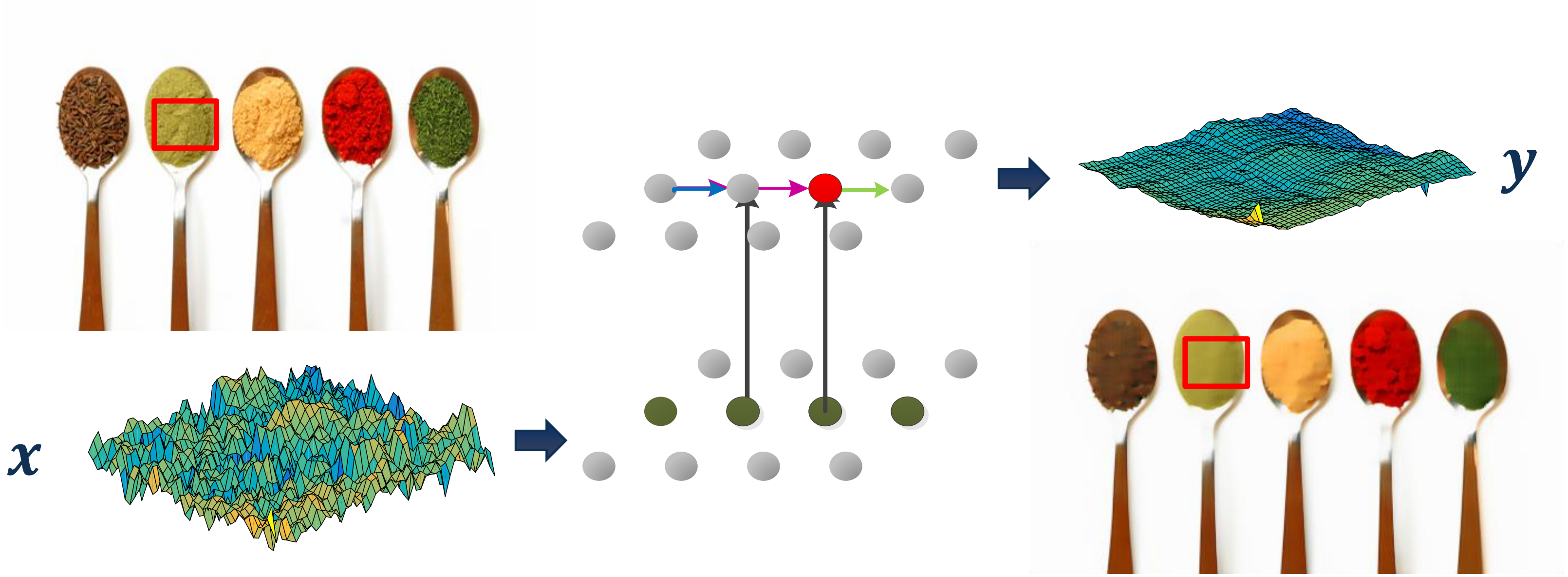
Recursive Filter



$$y[k] = a_k x[k] + p_k y[k-1]$$

- ✓ Small number of parameters
- × Difficult to design

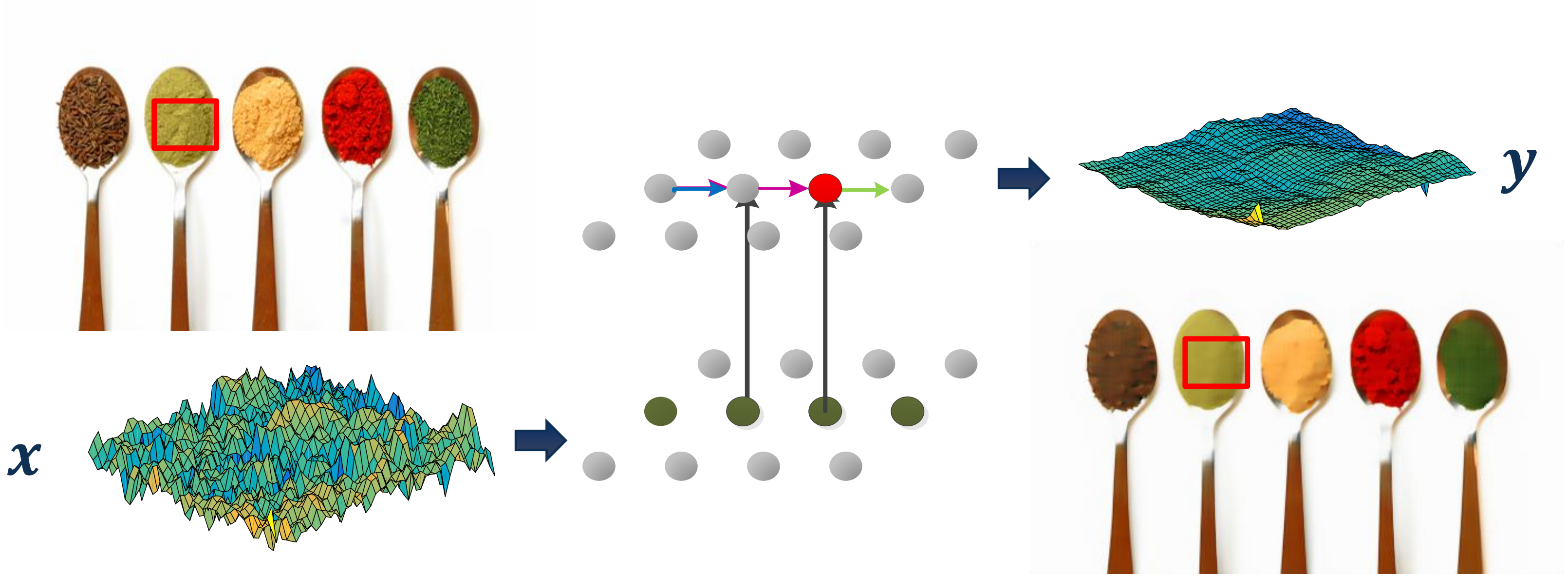
Recursive Filter



$$y[k] = a_k x[k] + p_k y[k-1]$$

- ✓ Small number of parameters
- × Difficult to design

Recursive Filter

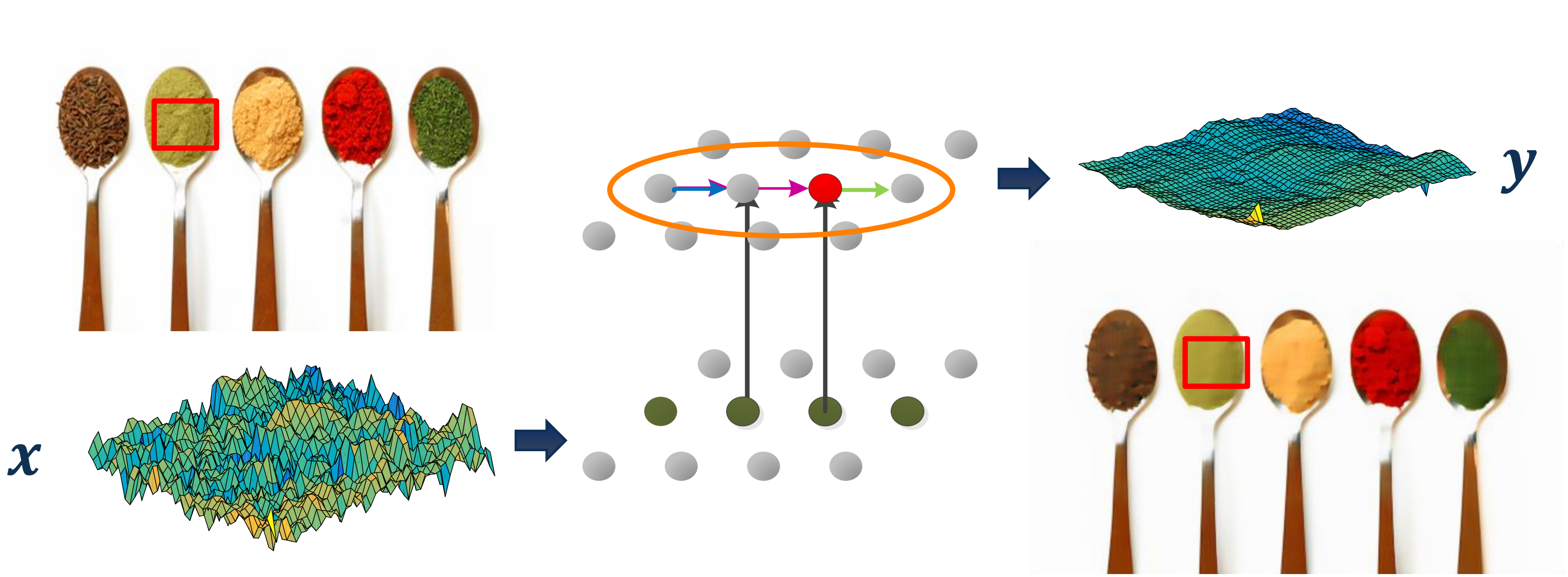


$$y[k] = a_k x[k] + p_k y[k-1]$$

Linear recurrent neural network (LRNN)

- ✓ Small number of parameters
- × Difficult to design

Recursive Filter

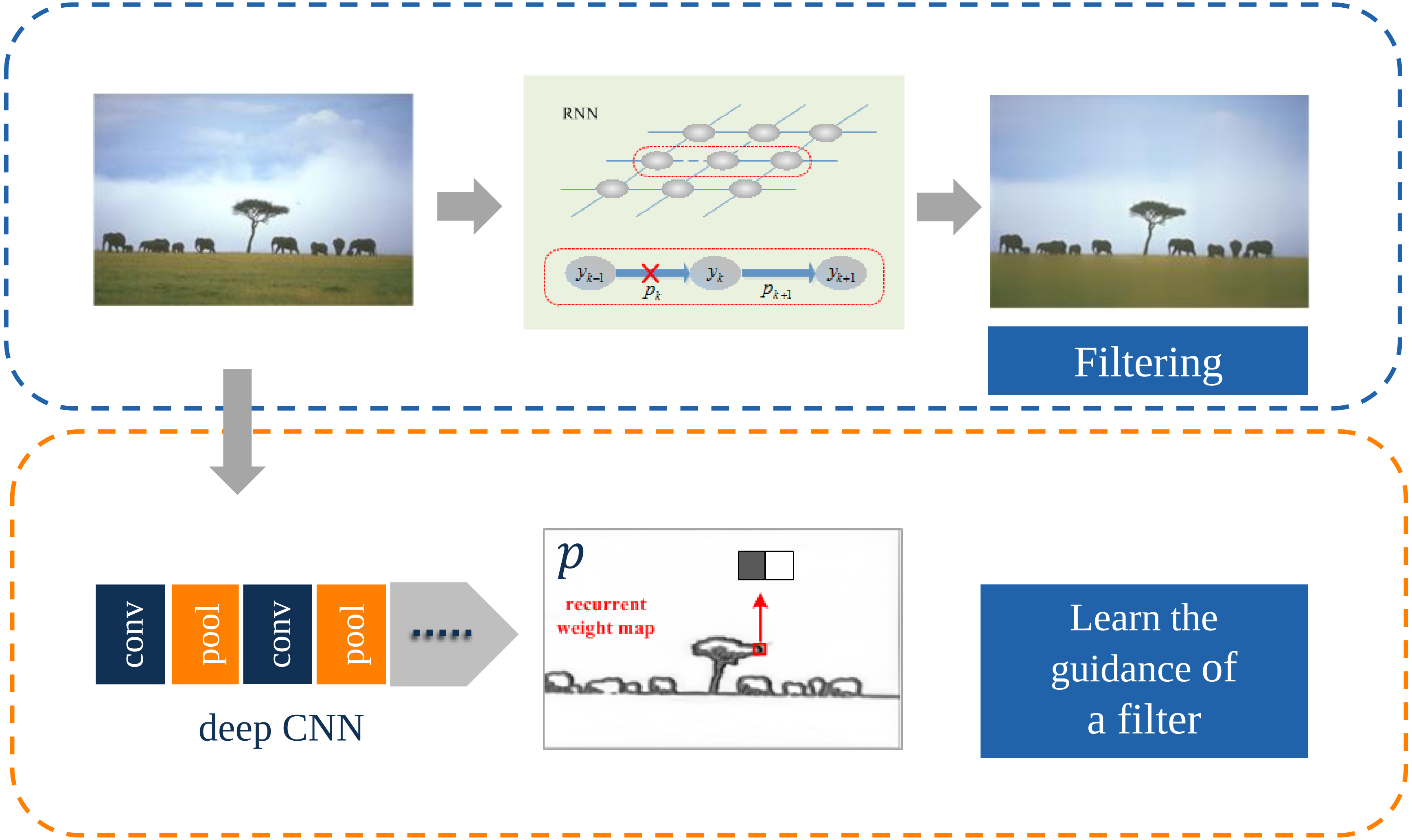


$$y[k] = a_k x[k] + p_k y[k-1]$$

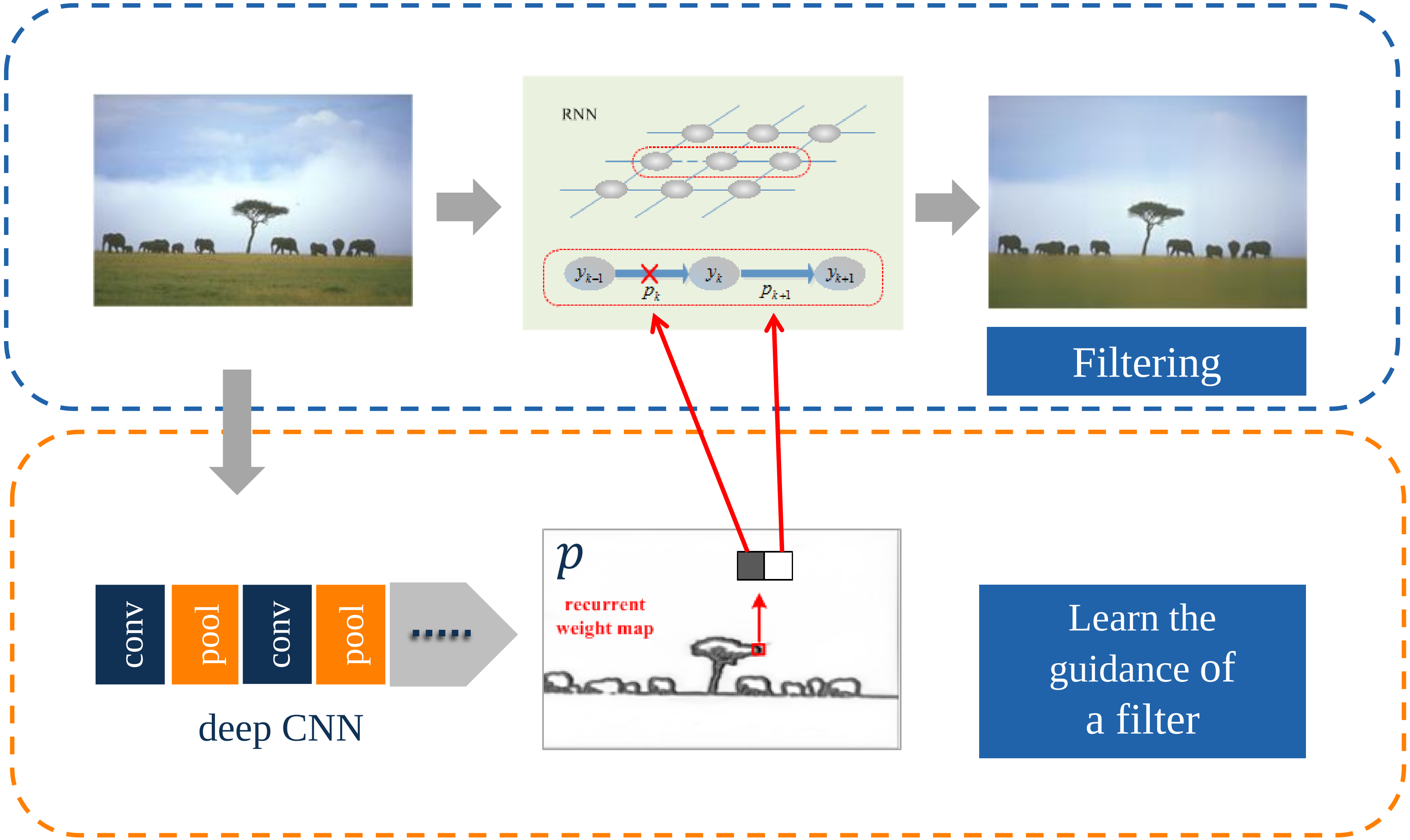
Linear recurrent neural network (LRNN)

- ✓ Small number of parameters
- × Difficult to design

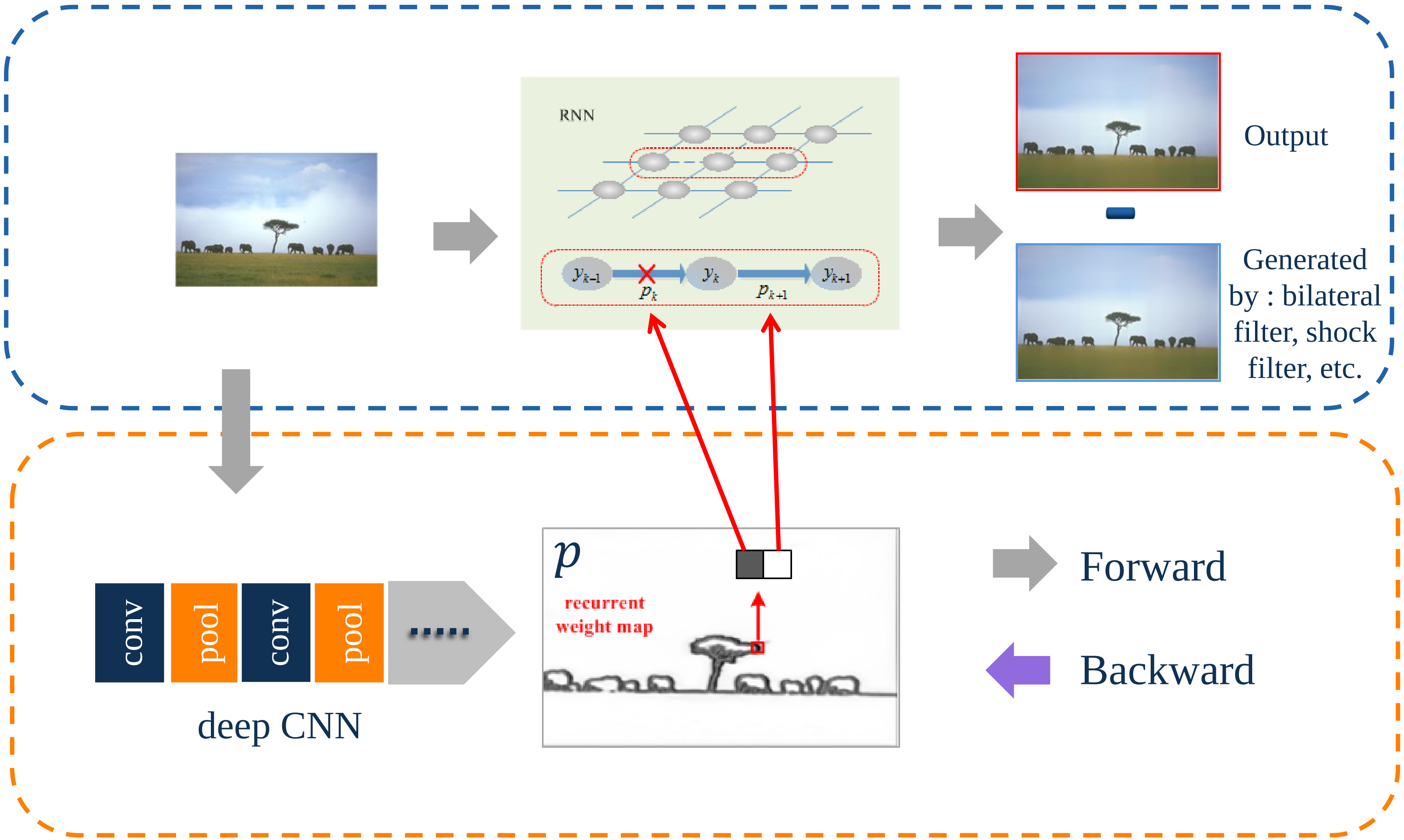
Hybrid Network



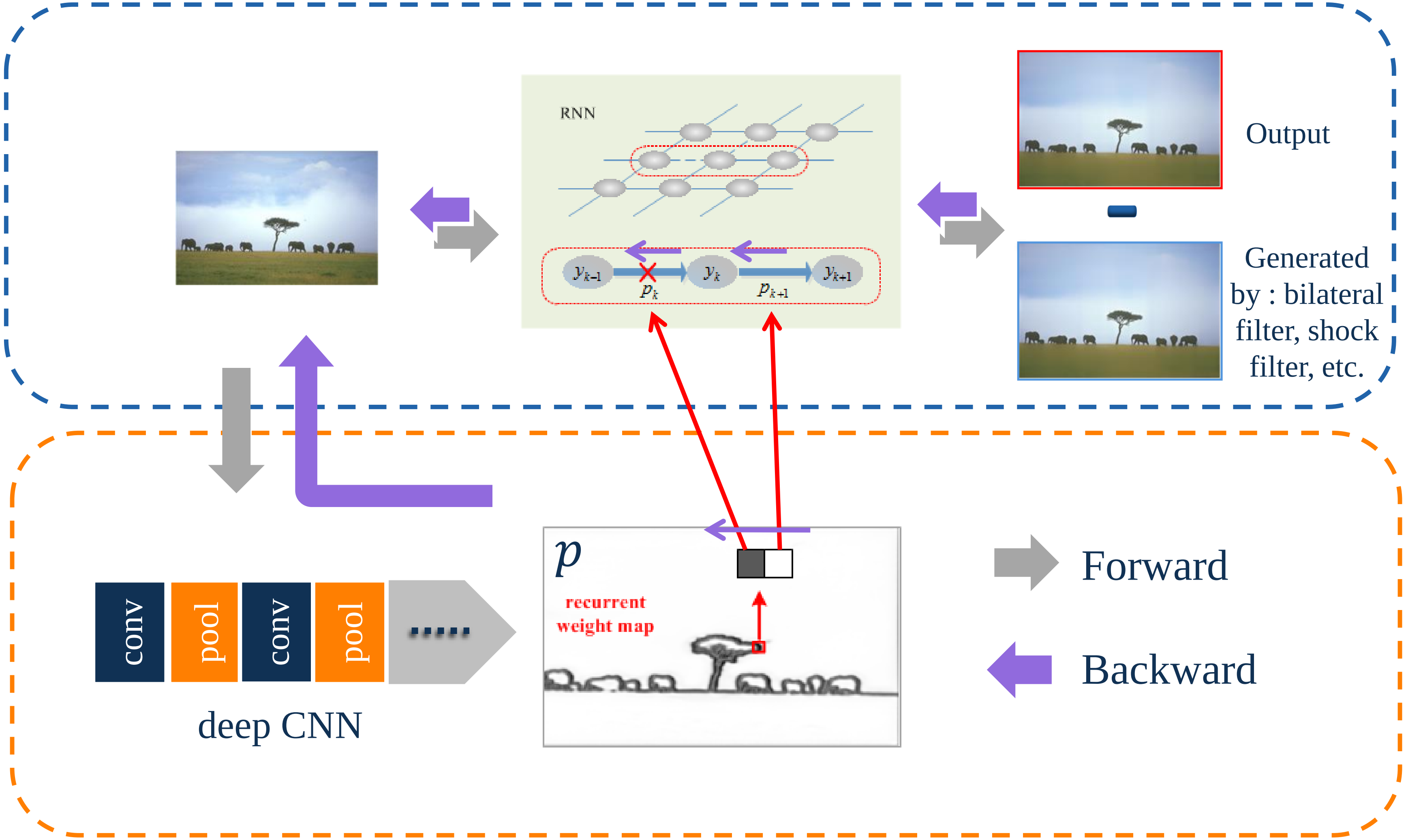
Hybrid Network



Framework of Hybrid Network



Framework of Hybrid Network



Perspective from Signal Processing

- Temporal domain

A general recursive filter

$$y[k] = \sum_{i=0}^{P} a_i x[k-i] + \sum_{j=1}^{Q} b_j y[k-j]$$

$$k = 0, \ldots, N$$

- Z domain

$$H_c = \sum_{i=0}^{P-Q} h_i z^{-i}$$

Perspective from Signal Processing

- Temporal domain

A general recursive filter

$$y[k] = \sum_{i=0}^{P} a_i x[k-i] + \sum_{j=1}^{Q} b_j y[k-j]$$

$$k = 0, \dots, N$$

A recursive unit:

$$y[k] = gx[k] + py[k-1]$$

- Z domain

$$H_c = \sum_{i=0}^{P-Q} h_i z^{-i}$$

Perspective from Signal Processing

- Temporal domain

A general recursive filter

$$y[k] = \sum_{i=0}^{P} a_i x[k-i] + \sum_{j=1}^{Q} b_j y[k-j]$$

$$k = 0, \ldots, N$$

A recursive unit:

$$y[k] = gx[k] + py[k-1]$$

- Z domain

Z-transform

$$H(z) = \frac{\sum_{i=0}^{P} a_i z^{-i}}{1 - \sum_{j=1}^{Q} b_j z^{-j}}$$

$$H_c = \sum_{i=0}^{P-Q} h_i z^{-i}$$

Perspective from Signal Processing

- Temporal domain

A general recursive filter

$$y[k] = \sum_{i=0}^{P} a_i x[k-i] + \sum_{j=1}^{Q} b_j y[k-j]$$
$$k = 0, \dots, N$$

A recursive unit:

$$y[k] = gx[k] + py[k-1]$$

- Z domain

Z-transform

$$H(z) = \frac{\sum_{i=0}^{P} a_i z^{-i}}{1 - \sum_{j=1}^{Q} b_j z^{-j}}$$

Cascade: $H(z) = H_r(z)H_c(z)$

$$H_r = \prod_{j=1}^{Q} \boxed{\frac{g_j}{1 - p_j z^{-1}}} \quad H_c = \prod_{i=1}^{P} h_i(1 - q_i z^{-1})$$

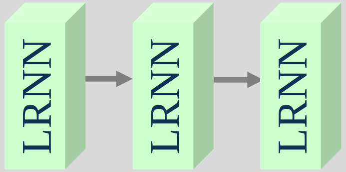
Parallel: $H(z) = H_r(z) + H_c(z)$

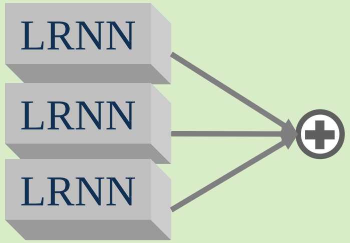
$$H_r = \sum_{j=1}^{Q} \boxed{\frac{g_j}{1 - p_j z^{-1}}} \quad H_c = \sum_{i=0}^{P-Q} h_i z^{-i}$$

Perspective from Signal Processing

- A general recursive filter is equivalent to the combination of multiple linear RNNs in cascade or parallel form.

Cascade: $$H_r = \prod_{j=1}^{Q} \frac{g_j}{1 - p_j z^{-1}}$$





Parallel: $$H_r = \sum_{j=1}^{Q} \frac{g_j}{1 - p_j z^{-1}}$$

Perspective from Signal Processing

- Temporal domain

A general recursive filter

$$y[k] = \sum_{i=0}^{P} a_i x[k-i] + \sum_{j=1}^{Q} b_j y[k-j]$$

$$k = 0, \dots, N$$

- Z domain

Z-transform

$$H(z) = \frac{\sum_{i=0}^{P} a_i z^{-i}}{1 - \sum_{j=1}^{Q} b_j z^{-j}}$$

Cascade: $H(z) = H_r(z) H_c(z)$

$$H_r = \prod_{j=1}^{Q} \frac{g_j}{1 - p_j z^{-1}} \qquad \boxed{H_c = \prod_{i=1}^{P} h_i (1 - q_i z^{-1})}$$

Parallel: $H(z) = H_r(z) + H_c(z)$

$$H_r = \sum_{j=1}^{Q} \frac{g_j}{1 - p_j z^{-1}} \qquad \boxed{H_c = \sum_{i=0}^{P-Q} h_i z^{-i}}$$

Combination of convolutional filters:
not applied in this work

Perspective from Signal Processing

- Temporal domain

A general recursive filter

$$y[k] = \sum_{i=0}^{P} a_i x[k-i] + \sum_{j=1}^{Q} b_j y[k-j]$$

$$k = 0, \ldots, N$$

- Z domain

Z-transform

$$H(z) = \frac{\sum_{i=0}^{P} a_i z^{-i}}{1 - \sum_{j=1}^{Q} b_j z^{-j}}$$

Cascade: $H(z) = H_r(z) H_c(z)$

$$H_r = \prod_{j=1}^{Q} \frac{g_j}{1 - p_j z^{-1}} \qquad \boxed{H_c = \prod_{i=1}^{P} h_i (1 - q_i z^{-1})}$$

Parallel: $H(z) = H_r(z) + H_c(z)$

$$H_r = \sum_{j=1}^{Q} \frac{g_j}{1 - p_j z^{-1}} \qquad \boxed{H_c = \sum_{i=0}^{P-Q} h_i z^{-i}}$$

Combination of convolutional filters:
not applied in this work

Perspective from Signal Processing

- Temporal domain

A general recursive filter

$$y[k] = \sum_{i=0}^{P} a_i x[k-i] + \sum_{j=1}^{Q} b_j y[k-j]$$

$$k = 0, \dots, N$$

Combination of convolutional filters:
not applied in this work

- Z domain

Z-transform

$$H(z) = \frac{\sum_{i=0}^{P} a_i z^{-i}}{1 - \sum_{j=1}^{Q} b_j z^{-j}}$$

Cascade: $H(z) = H_r(z)\cancel{H_c(z)}$

$$H_r = \prod_{j=1}^{Q} \frac{g_j}{1 - p_j z^{-1}} \qquad \cancel{\boxed{H_c = \prod_{i=1}^{P} h_i(1 - q_i z^{-1})}}$$

Low-pass filter

Parallel: $H(z) = H_r(z) \cancel{+ H_c(z)}$

$$H_r = \sum_{j=1}^{Q} \frac{g_j}{1 - p_j z^{-1}} \qquad \cancel{\boxed{H_c = \sum_{i=0}^{P-Q} h_i z^{-i}}}$$

High-pass filter

Spatially Variant Linear RNN

A 1st recursive filter

$$y[k] = gx[k] + py[k-1]$$

Normalized filter

$g = 1 - p$

$$y[k] = (1-p)x[k] + py[k-1]$$

Spatially variant

$$y[k] = (1-p[k])x[k] + p[k]y[k-1]$$

Back propagation

$$\sigma[k] = \theta[k]\{y[k-1] - x[k]\}$$

Spatially Variant Linear RNN

A 1st recursive filter

$$y[k] = gx[k] + py[k-1]$$

Normalized filter

$g = 1 - p$

$$y[k] = (1-p)x[k] + py[k-1]$$

Spatially variant

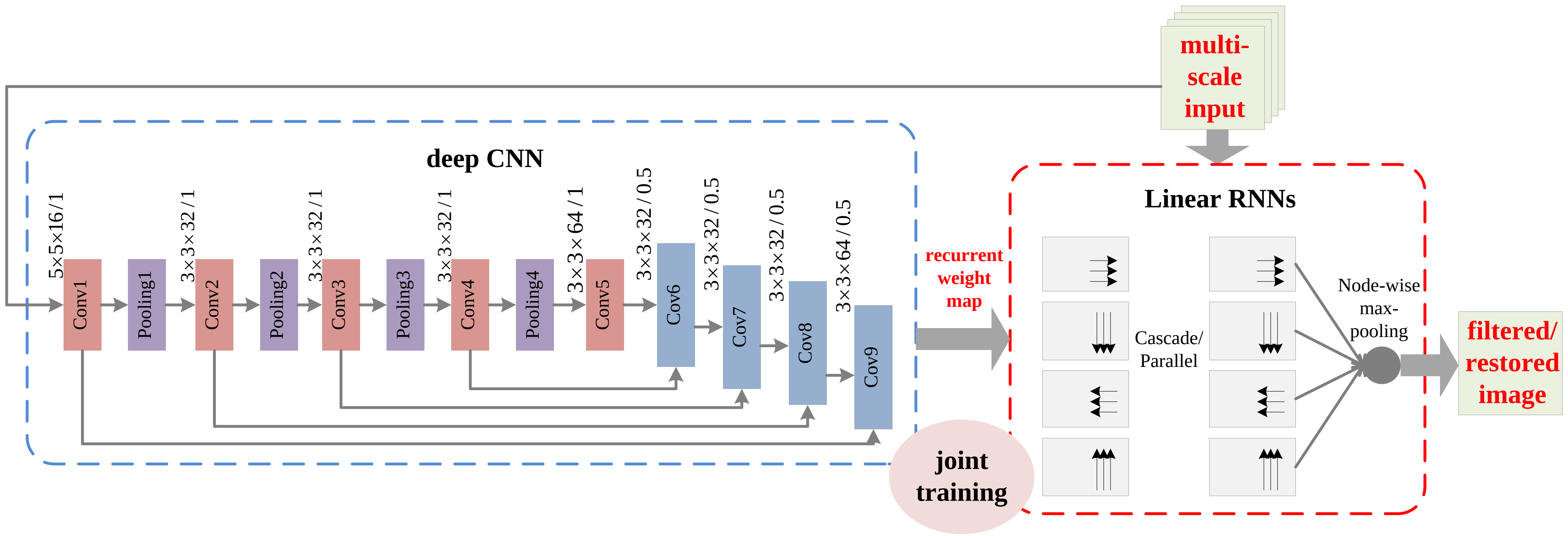
$$y[k] = (1-p[k])x[k] + p[k]y[k-1]$$

$\boldsymbol{p[k]}$

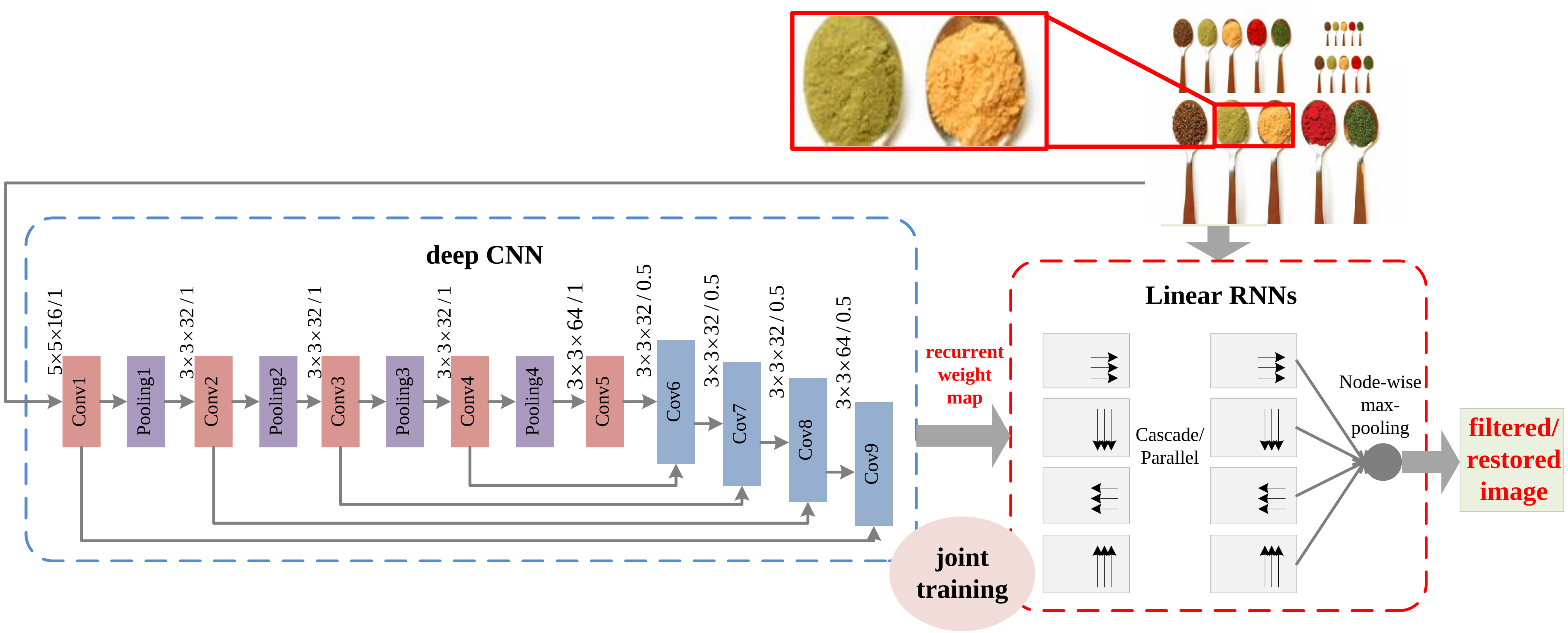
Back propagation

$$\sigma[k] = \theta[k]\{y[k-1] - x[k]\}$$

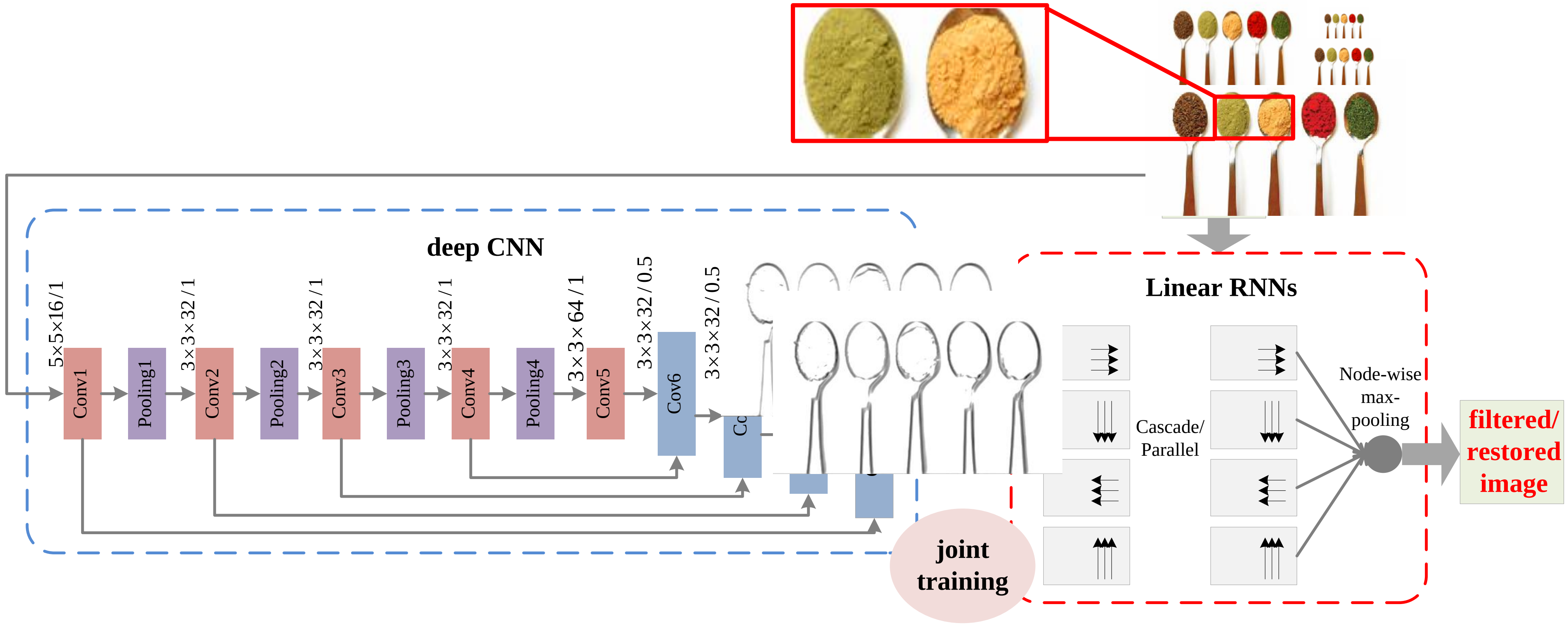
Hybrid Network: Joint Training



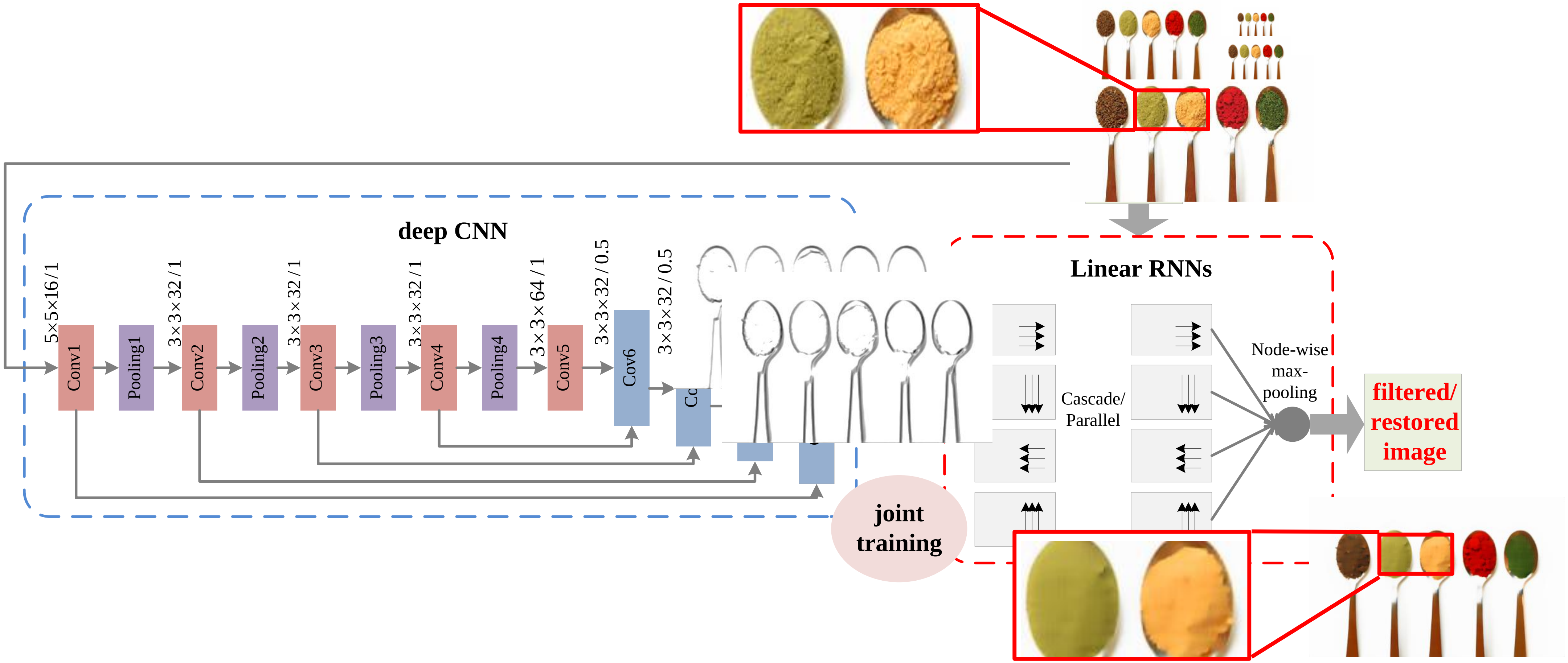
Hybrid Network: Joint Training



Hybrid Network: Joint Training

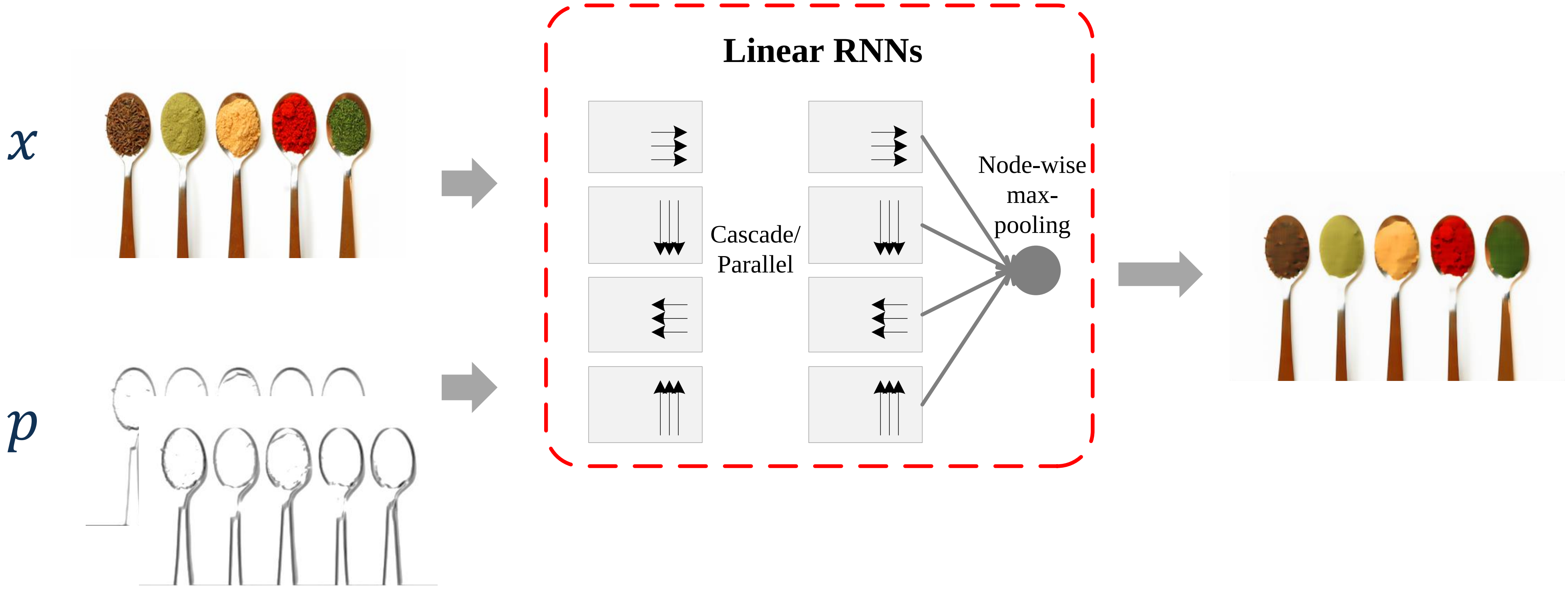


Hybrid Network: Joint Training



Hybrid Network: Linear RNNs

1D filters in 4 directions

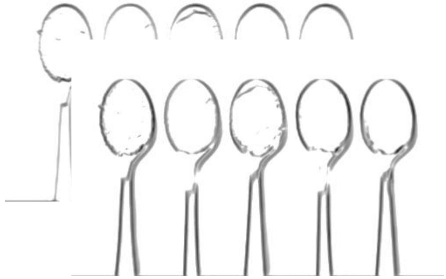


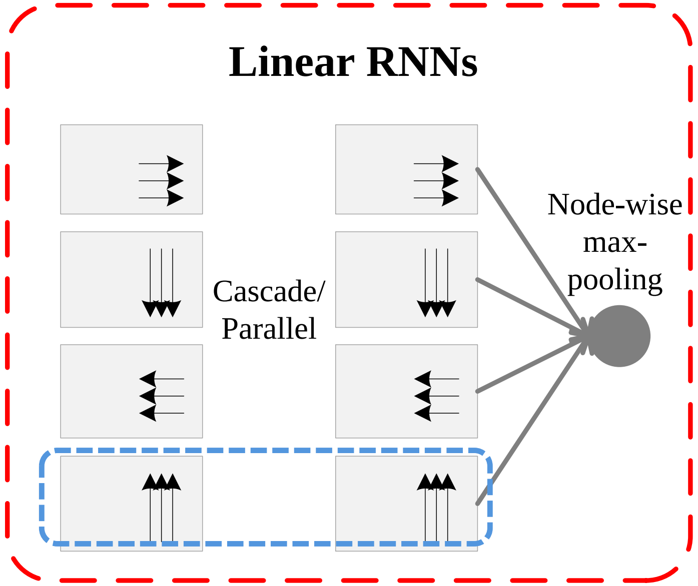
Hybrid Network: Linear RNNs

1D filters in 4 directions

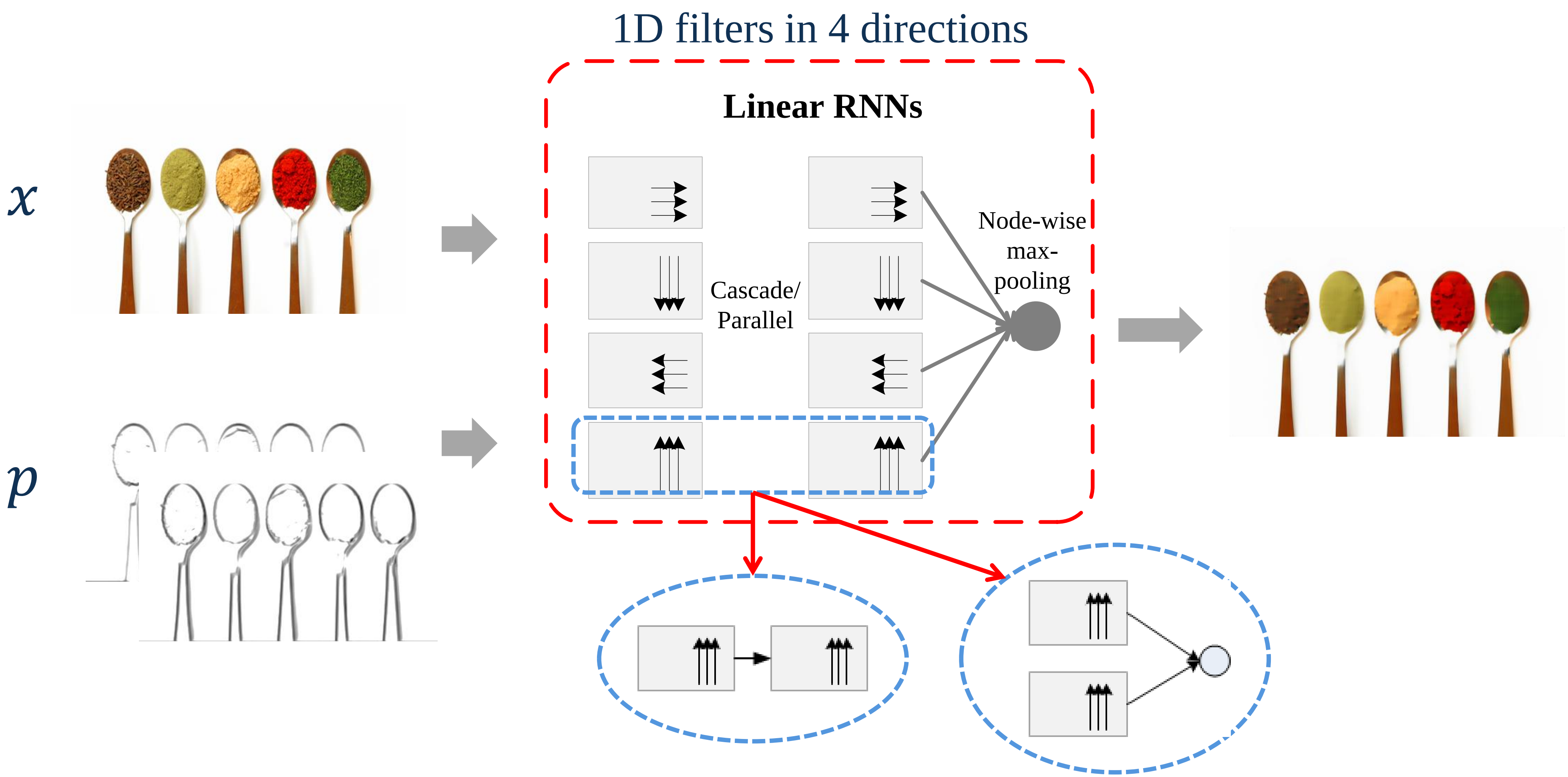
x

p

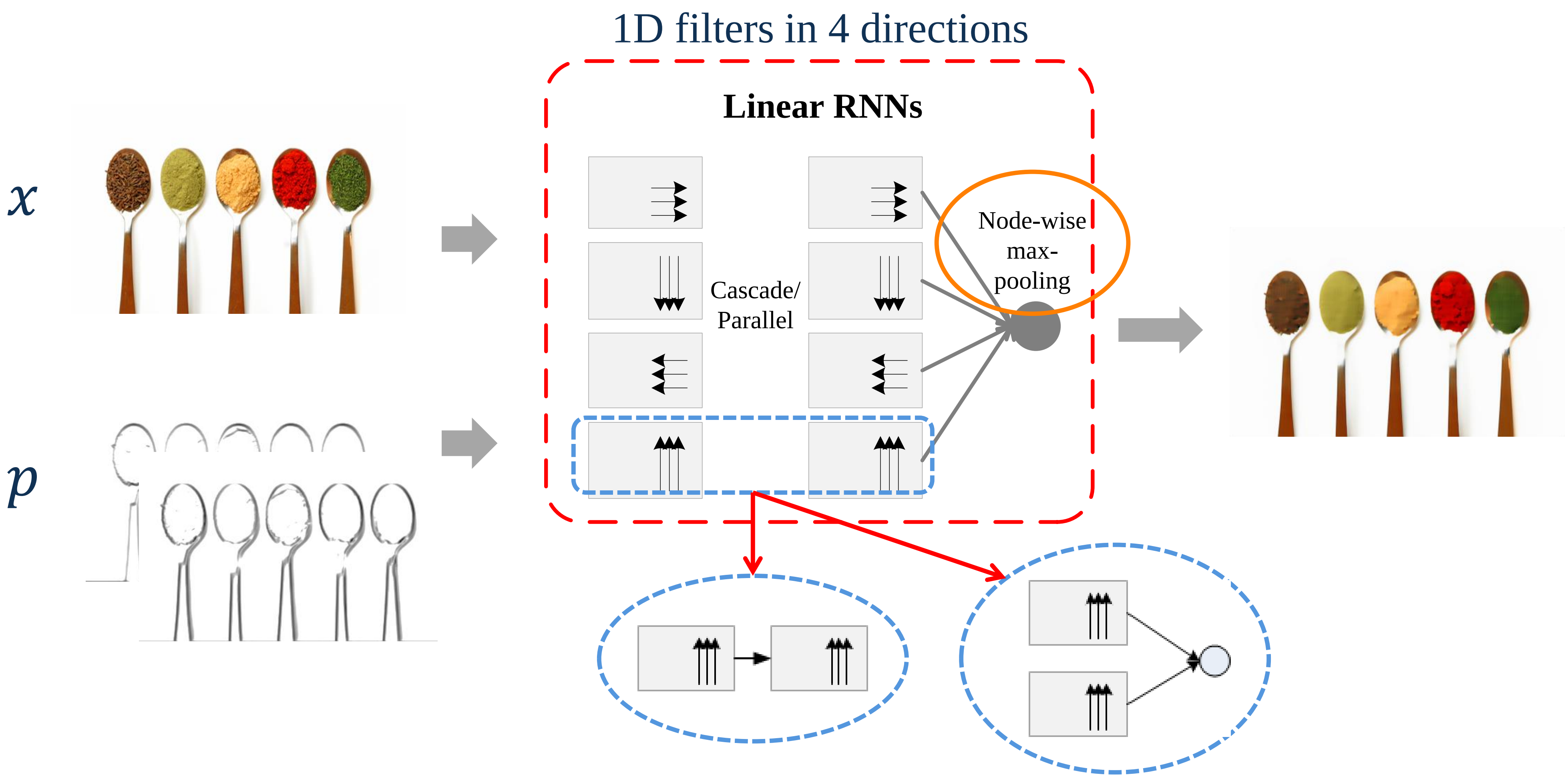




Hybrid Network: Linear RNNs

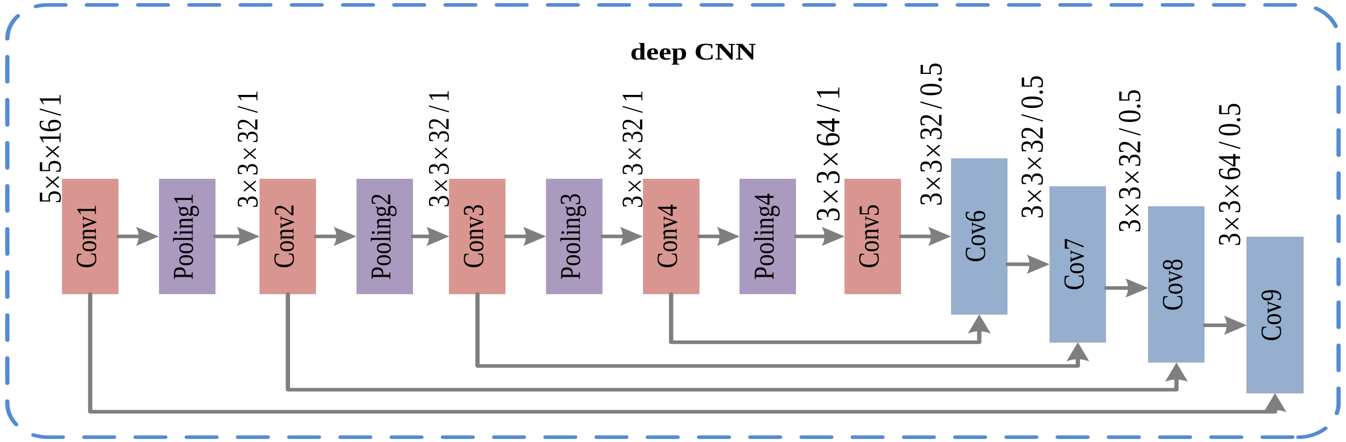


Hybrid Network: Linear RNNs

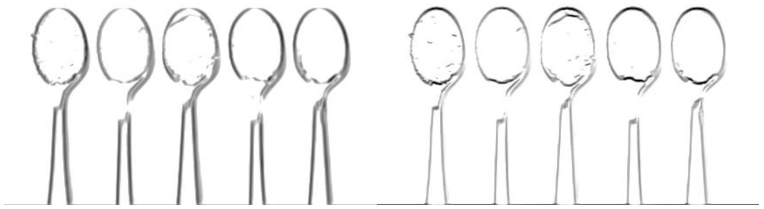


Hybrid Network: CNN

Input



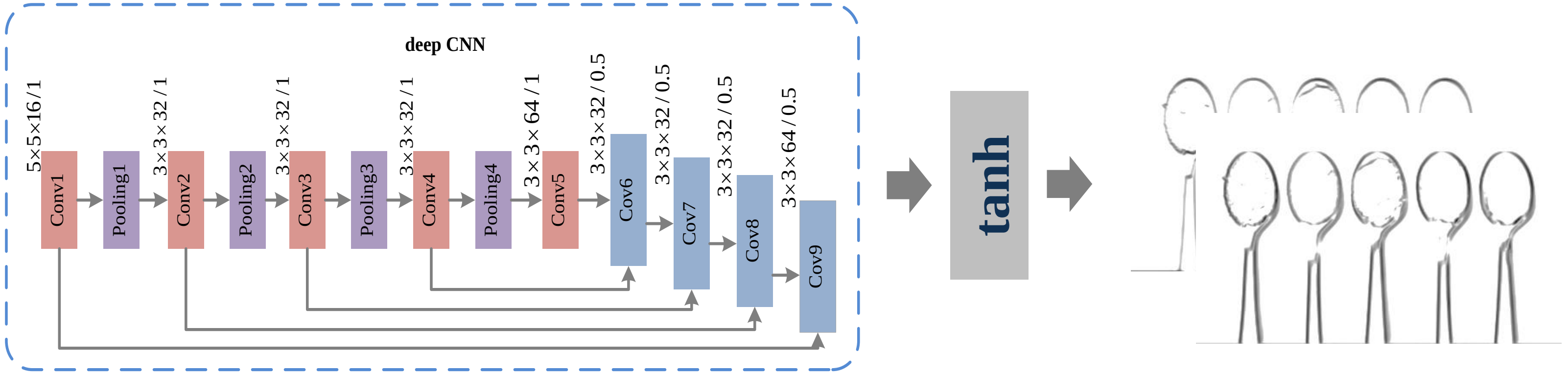
Output



x-axis y-axis

Model Stability

- Vanilla RNN: nonlinearity function (e.g., sigmoid, tanh, etc.)
- Linear RNN: $|p| < 1$, so that all poles lie inside the unit circle
  - If p is trainable (e.g., the output of a CNN), the stability can be maintained by regularizing its value through a tanh layer: $p \in (-1,1)$



Weight Maps with Single LRNN

- Learning the Relative Total Variation (RTV) filter (Xu et al. SIGGRAPH ASIA 2012)

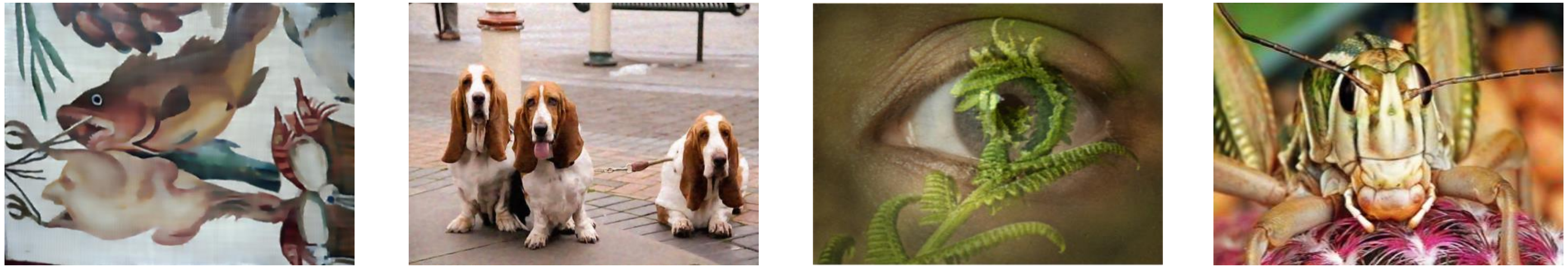
Weight Maps with Single LRNN

- Learning the L0 filter (Xu et al. ICML 2015)

x-axis

y-axis

Low-Level Vision Tasks



	Filter	Denoising	Interpolation
Input	Original image	Degraded image	Degraded image + mask
Output	Filtered image	Restored image	Restored color image

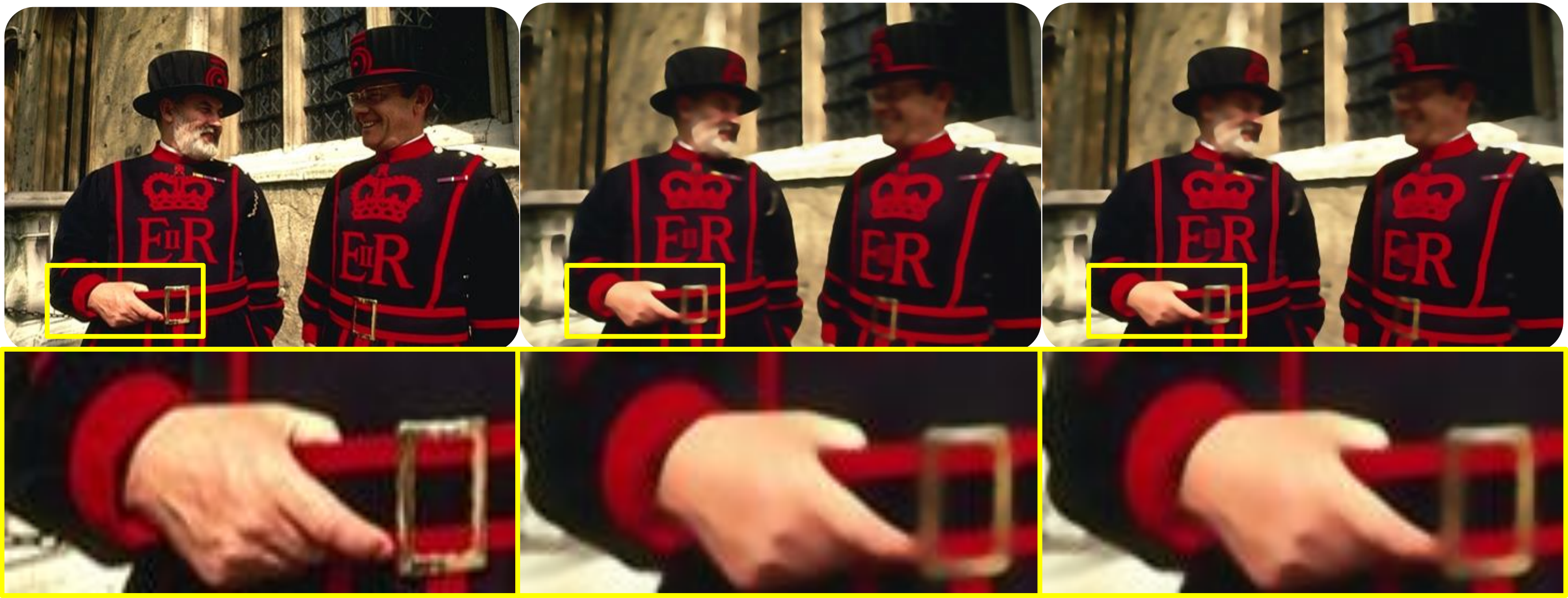
Edge-Preserving Smoothing

- Generally outperform the CNN filter (Xu et al. ICML 2015)

PSNR	L0	BLF	RTV	RGF	WLS	WMF	Shock
Xu et al.	32.8	38.4	32.1	35.9	36.2	31.6	30.0
Ours	30.9	**38.6**	**37.1**	**42.2**	**39.4**	**34.0**	**31.8**

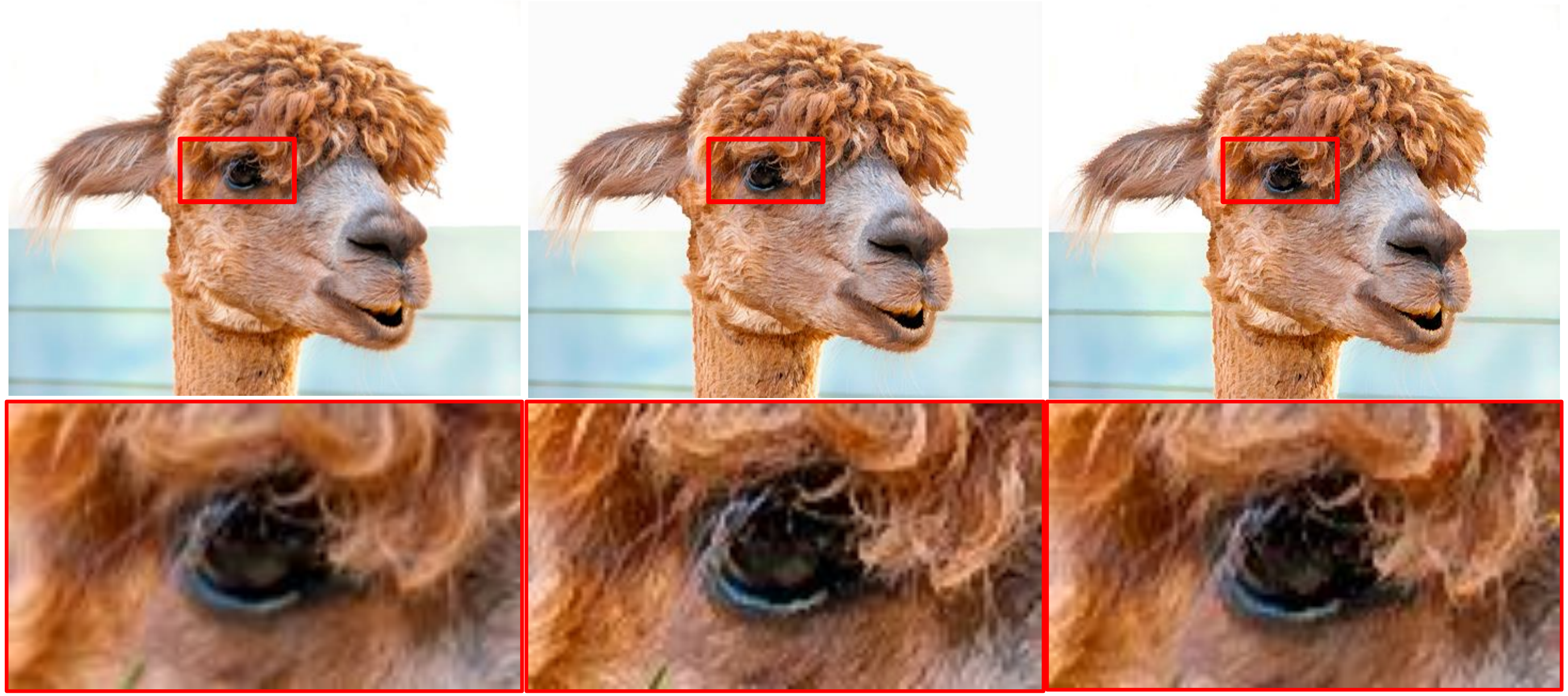
- BLF: Bilateral filter (Yang et al. ECCV 2013)
- RTV: Relative total variation filter (Xu et al. SIGGRAPH ASIA 2012)
- RGF: Rolling guidance filter (Zhang et al. ECCV 2014)
- WLS: Weighted least squares filter (Farbman et al. SIGGRAPH 2008)
- WMF: Weighted median filter (Zhang et al. CVPR 2014)
- Shock: Shock filter

Edge-Preserving Smoothing: Rolling Guidance Filter



Original Proposed RGF

Edge-Preserving Enhancement: Shock Filter



Original

Proposed

Shock

Image Denoising

Noisy

EPLL (Zoran et al) PSNR: 31.0

CNN (Ren et al) PSNR:31.0

Ours PSNR:32.3

Image Pixel Propagation: 50% Random Pixels

Original

Restored

Image Pixel Propagation: Character Inpainting

Original

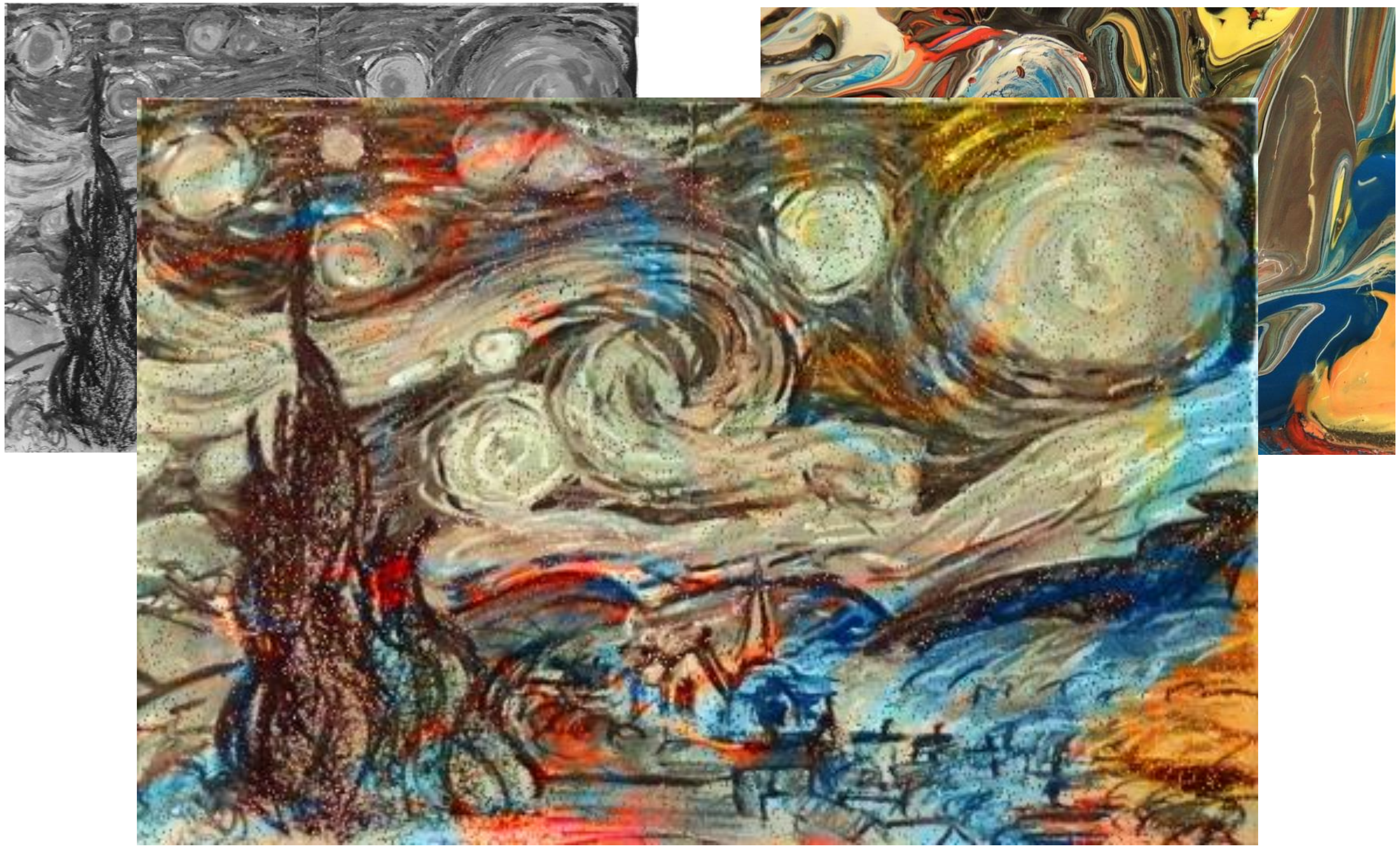
Restored

Color Pixel Propagation: 3% Color Retained

Color Pixel Propagation: 3% Color Retained

Re-colorization

Re-colorization



Run Time and Model Size

- Ten times smaller than the CNN filter (**0.54** vs. **5.60** MB)
- Real-time with QVGA images

Second/ MB	**BLF**	**WLS**	**RTV**	**WMF**	**EPLL**	**Levin**	**Xu et al.**	**Ours**
QVGA (320×240)	0.46	0.71	1.22	0.94	33.82	2.10	0.23	**0.05**
VGA (640×480)	1.41	3.25	6.26	3.54	466.79	9.24	0.83	**0.16**
720p (1280×720)	3.18	9.42	16.26	4.98	1395.61	31.09	2.11	**0.37**

Concluding Remarks

- Learning image filters by a hybrid neural network
  - Convolutional neural network
  - Recurrent neural network
- Address the issues with state-of-the-art convolutional filters
  - Slow speed
  - Large model size
  - Do not exploit structural information

Demo: Cartooning

Code and datasets available at:
http://www.sifeiliu.net/linear-rnn
http://vllab.ucmerced.edu

LRNN vs. Vanilla RNN

$$h[k] = (1 - p[k]) \cdot x[k] + p[k] \cdot h[k-1] \quad \Big| \quad h[k] = f\{W_x x[k] + W_h (h[k-1] + b)\}$$

- Spatially variant filter
  - LRNN is spatially variant w.r.t the spatial location *k* where each *k* is controlled by a different recursive filter.
- Infinite-term dependency
  - Compared to the vanilla RNN with short-term dependency, or even long short-term memory (LSTM) with long-term dependency, the LRNN does not contain any *W* that formulates an exponentially decreasing influence.
  - Instead when *p* reaches 1, the value of *h* can propagate with infinite steps.
- Linear system
  - LRNN is a linear system with trainable coefficient.
  - Its linearity applies to many low-level problem such as filtering/denoising/interpolation, compared to the Vanilla RNN/LSTM.

LRNN vs. Pixel RNN