Non-linear Models-VII

CS771: Introduction to Machine Learning

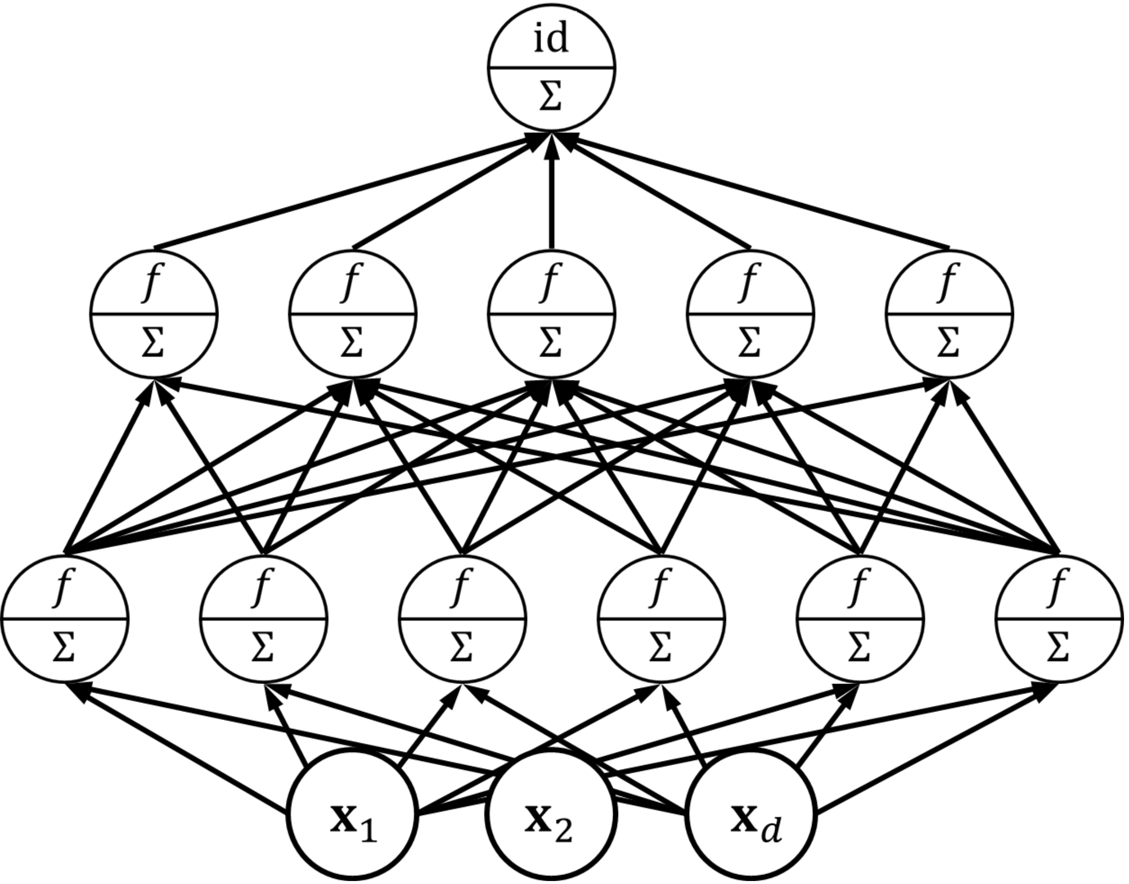
Purushottam Kar

Announcements

- Discussion session this Sunday, Nov 5, 6PM, RM101
- Please submit questions to http://tinyurl.com/ml17-18ads2
- Please (re)upload your project proposals to GS by Sun, Nov 5
- Make sure all teammates are linked to the (group) submission

Convolutional Neural Networks

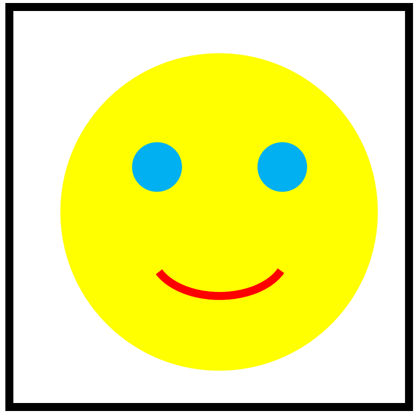
Feedforward Networks can be an overkill!



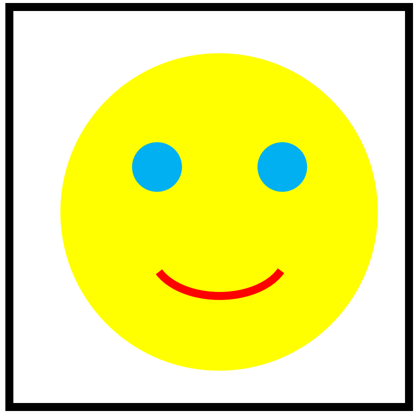
- Fully connected layers are powerful
- Allow all possible combinations of input dims to create new features
  - $\mathbf{x}_1$ can talk to $\mathbf{x}_2$ as well as $\mathbf{x}_d$
- Allow all possible combinations of hidden layer outputs too
- Also very unnecessary for apps where input has structure
- Make networks very bulky
- Also require tons of data to train so many edge weights

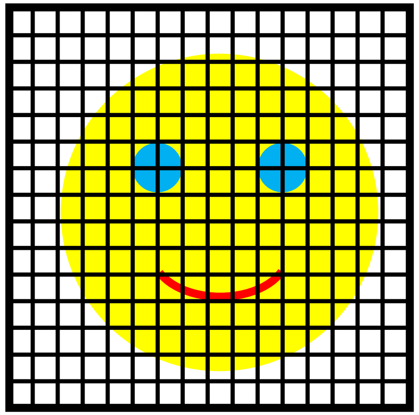
Images demand Local Features

Images demand Local Features

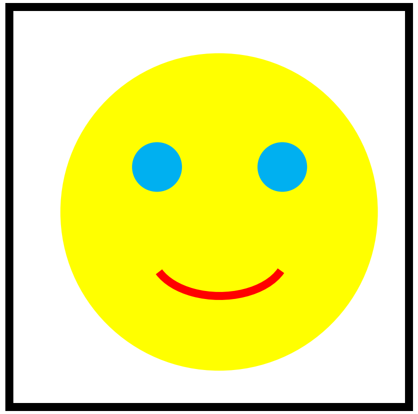


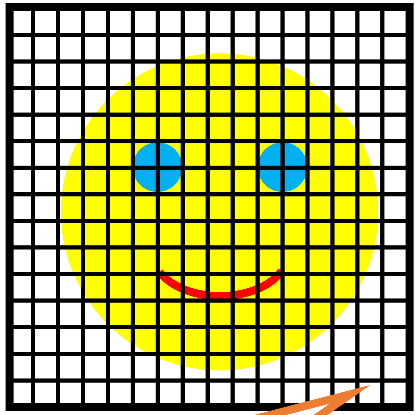
Images demand Local Features





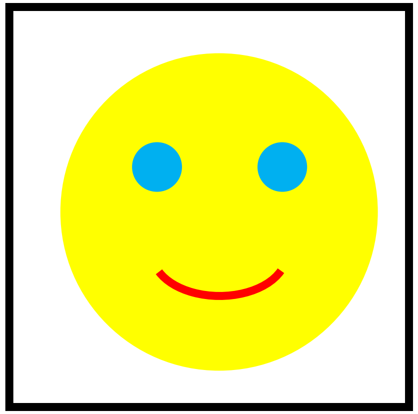
Images demand Local Features

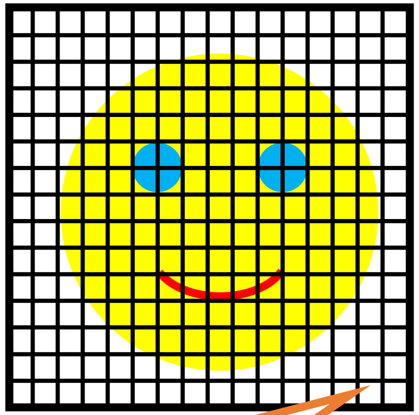




The top left and bottom right pixels do not need to be considered together right at the beginning

Images demand Local Features

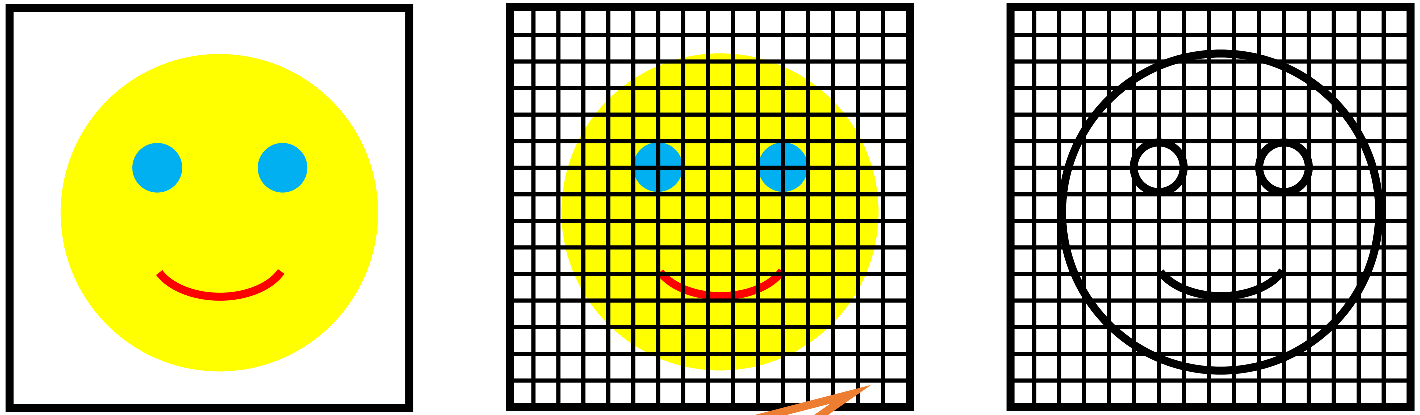




The top left and bottom right pixels do not need to be considered together right at the beginning

First, only neighboring pixels need to talk to each other to detect edges

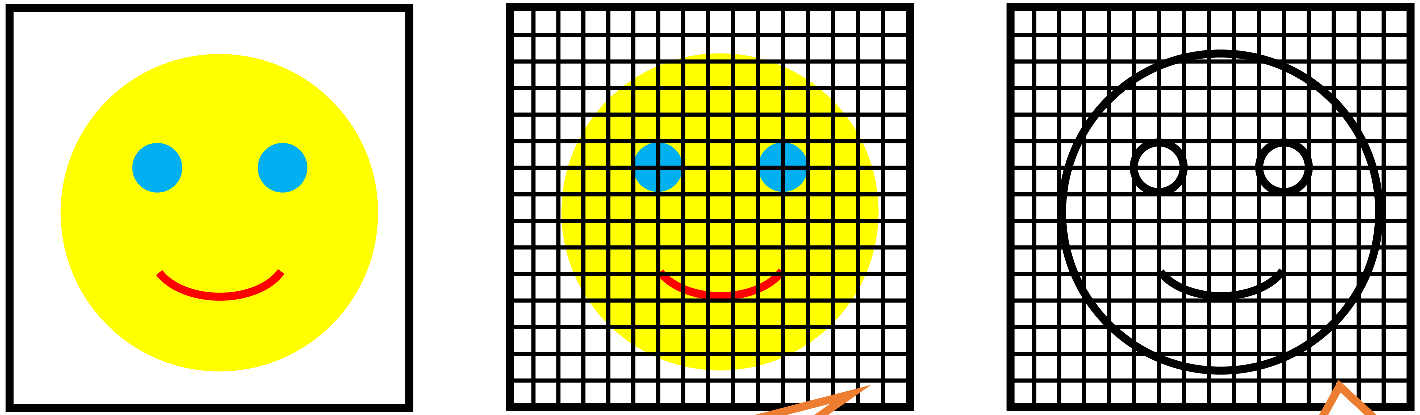
Images demand Local Features



The top left and bottom right pixels do not need to be considered together right at the beginning

First, only neighboring pixels need to talk to each other to detect edges

Images demand Local Features

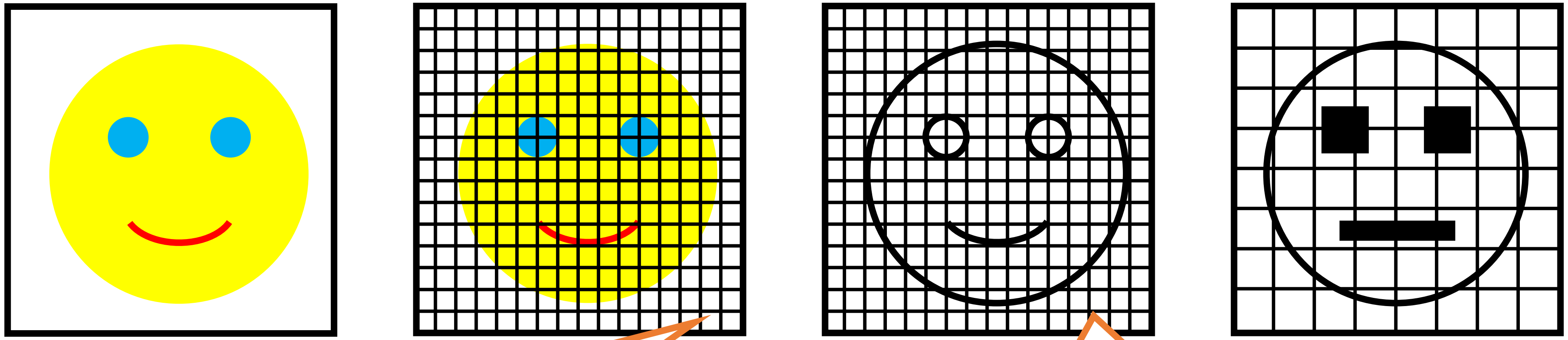


The top left and bottom right pixels do not need to be considered together right at the beginning

First, only neighboring pixels need to talk to each other to detect edges

Then, need to aggregate info to detect structures

Images demand Local Features

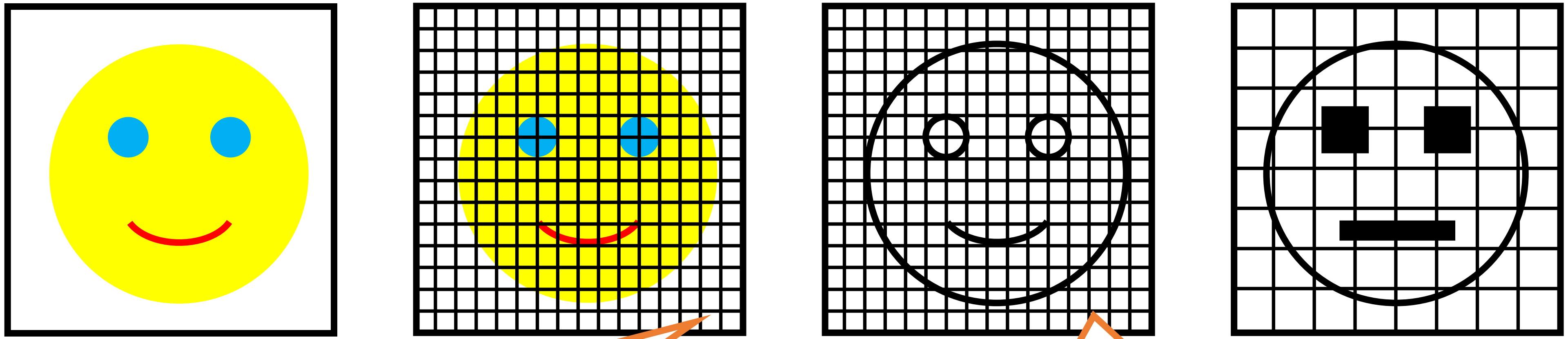


The top left and bottom right pixels do not need to be considered together right at the beginning

First, only neighboring pixels need to talk to each other to detect edges

Then, need to aggregate info to detect structures

Images demand Local Features



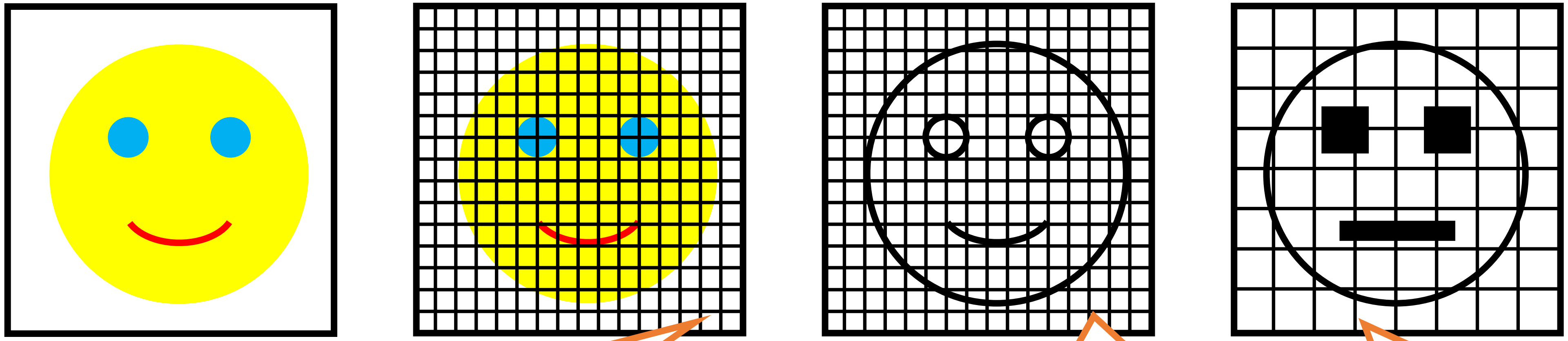
The top left and bottom right pixels do not need to be considered together right at the beginning

First, only neighboring pixels need to talk to each other to detect edges

Then, need to aggregate info to detect structures

Then, need to detect even higher level features

Images demand Local Features



The top left and bottom right pixels do not need to be considered together right at the beginning

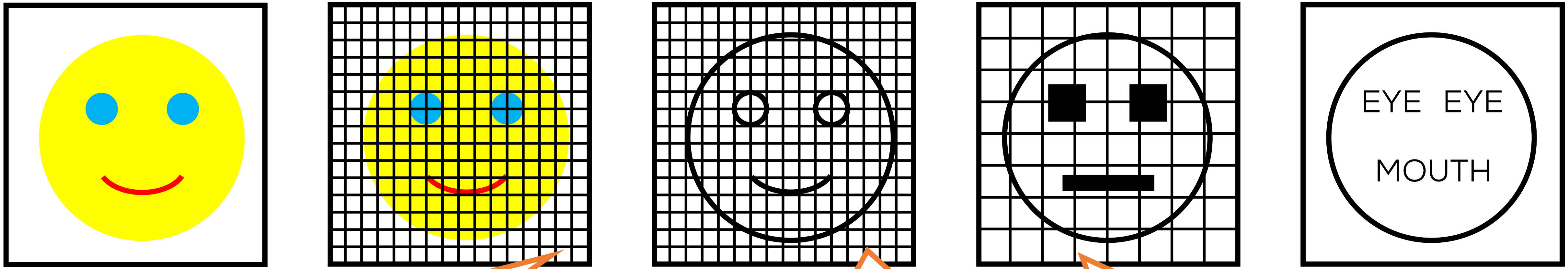
First, only neighboring pixels need to talk to each other to detect edges

Then, need to aggregate info to detect structures

Then, need to detect even higher level features

Then, need to detect even higher level structures

Images demand Local Features



The top left and bottom right pixels do not need to be considered together right at the beginning

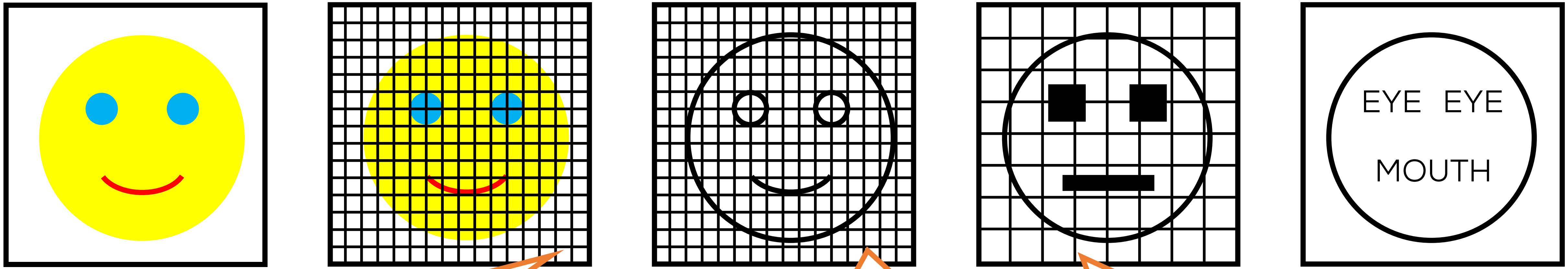
First, only neighboring pixels need to talk to each other to detect edges

Then, need to aggregate info to detect structures

Then, need to detect even higher level features

Then, need to detect even higher level structures

Images demand Local Features



The top left and bottom right pixels do not need to be considered together right at the beginning

First, only neighboring pixels need to talk to each other to detect edges

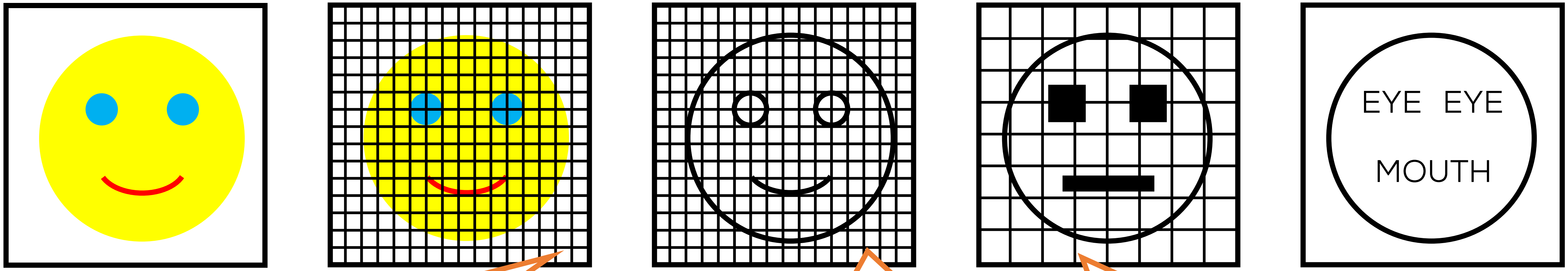
Then, need to aggregate info to detect structures

Then, need to detect even higher level features

Then, need to detect even higher level structures

Distant pixels do communicate, but at a much later stage

Images demand Local Features



The top left and bottom right pixels do not need to be considered together right at the beginning

First, only neighboring pixels need to talk to each other to detect edges

The same procedure is used to detect edges all over the image

Then, need to aggregate info to detect structures

Then, need to detect even higher level features

Then, need to detect even higher level structures

Distant pixels do communicate, but at a much later stage

Text demands Local Features

Text demands Local Features

The quick brown fox jumps over the lazy dog

Text demands Local Features

The quick brown fox jumps over the lazy dog

Clues from neighbouring words help identify part of speech, adjective, noun etc

Text demands Local Features

The quick brown fox jumps over the lazy dog

DET ADJ ADJ NN VV PP DET ADJ NN

Clues from neighbouring words help identify part of speech, adjective, noun etc

Text demands Local Features

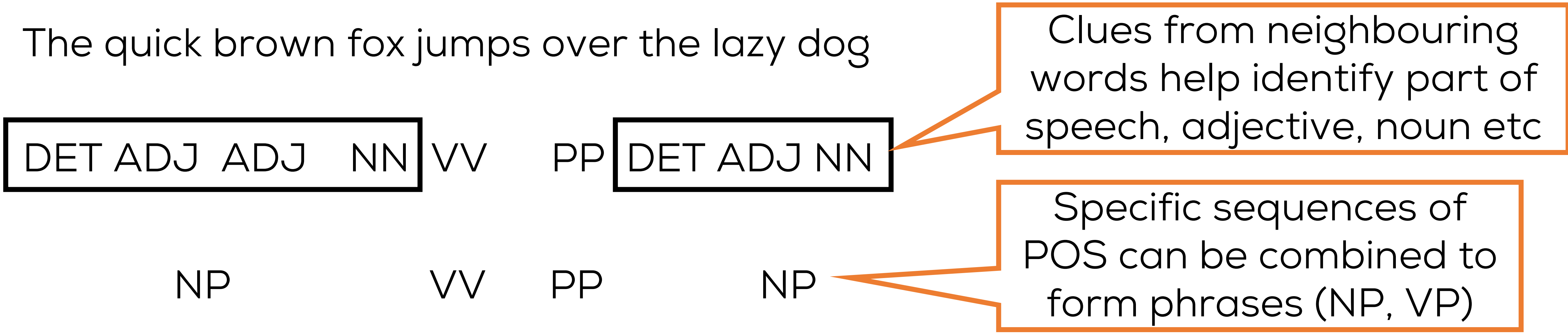
The quick brown fox jumps over the lazy dog

DET ADJ ADJ NN VV PP DET ADJ NN

Clues from neighbouring words help identify part of speech, adjective, noun etc

Specific sequences of POS can be combined to form phrases (NP, VP)

Text demands Local Features



The quick brown fox jumps over the lazy dog

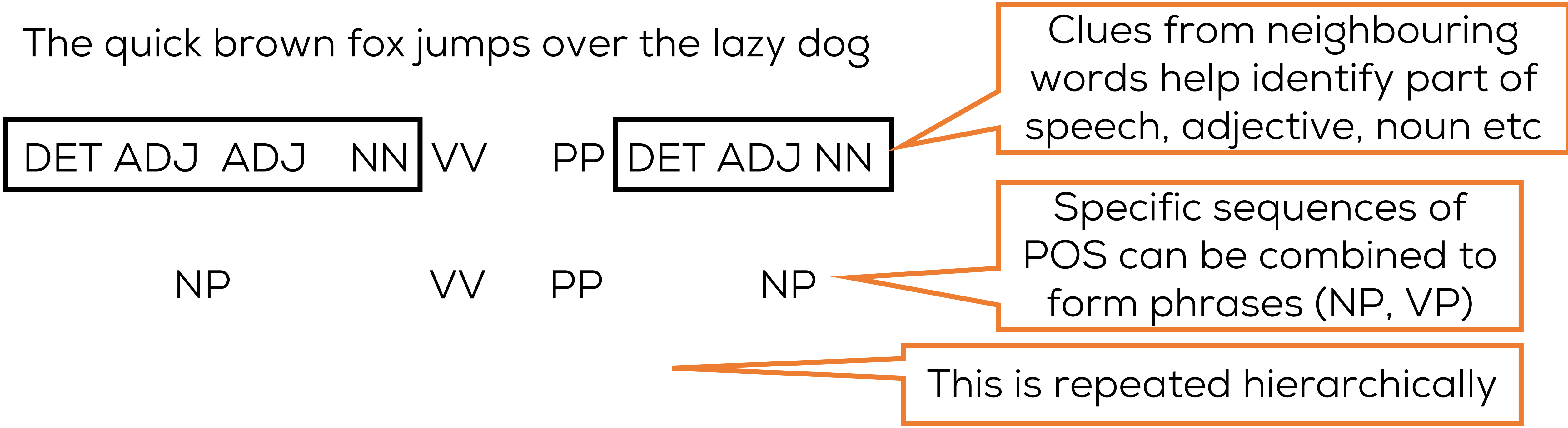
DET ADJ ADJ NN VV PP DET ADJ NN

NP VV PP NP

Clues from neighbouring words help identify part of speech, adjective, noun etc

Specific sequences of POS can be combined to form phrases (NP, VP)

Text demands Local Features



The quick brown fox jumps over the lazy dog

DET ADJ ADJ NN VV PP DET ADJ NN

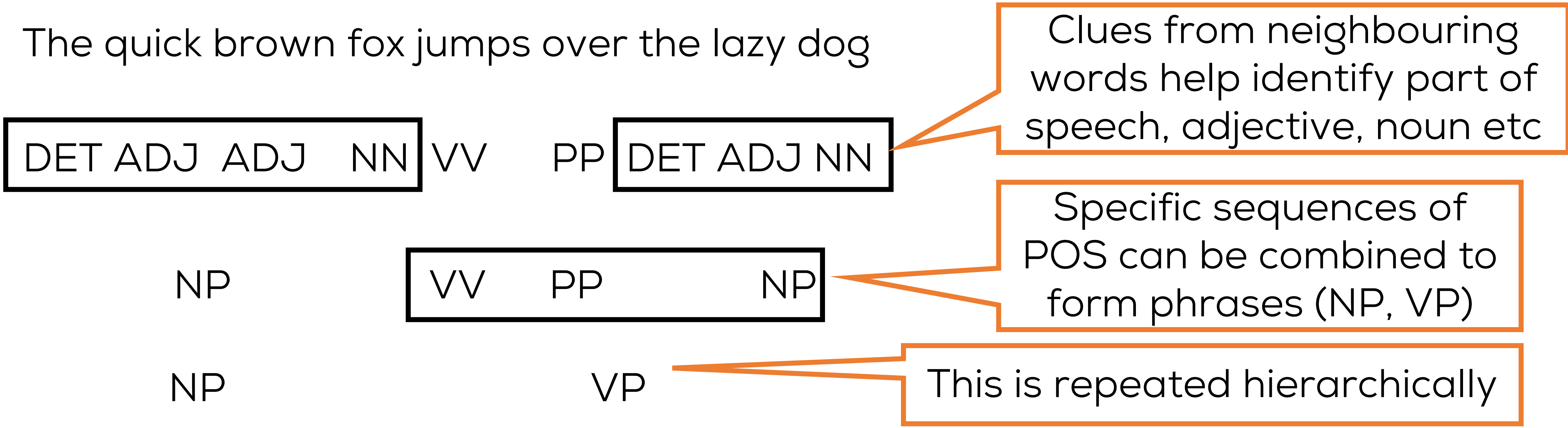
NP VV PP NP

Clues from neighbouring words help identify part of speech, adjective, noun etc

Specific sequences of POS can be combined to form phrases (NP, VP)

This is repeated hierarchically

Text demands Local Features



The quick brown fox jumps over the lazy dog

DET ADJ ADJ NN VV PP DET ADJ NN

NP VV PP NP

NP VP

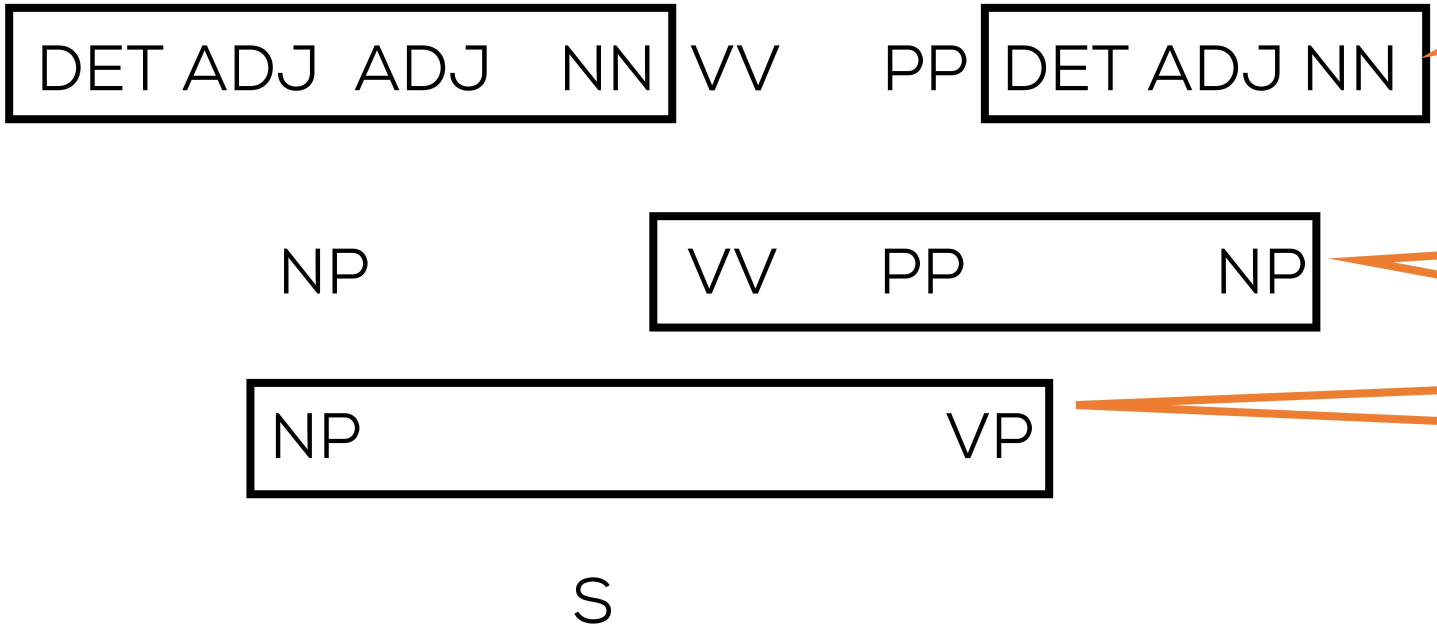
Clues from neighbouring words help identify part of speech, adjective, noun etc

Specific sequences of POS can be combined to form phrases (NP, VP)

This is repeated hierarchically

Text demands Local Features

The quick brown fox jumps over the lazy dog



DET ADJ ADJ NN VV PP DET ADJ NN

NP VV PP NP

NP VP

S

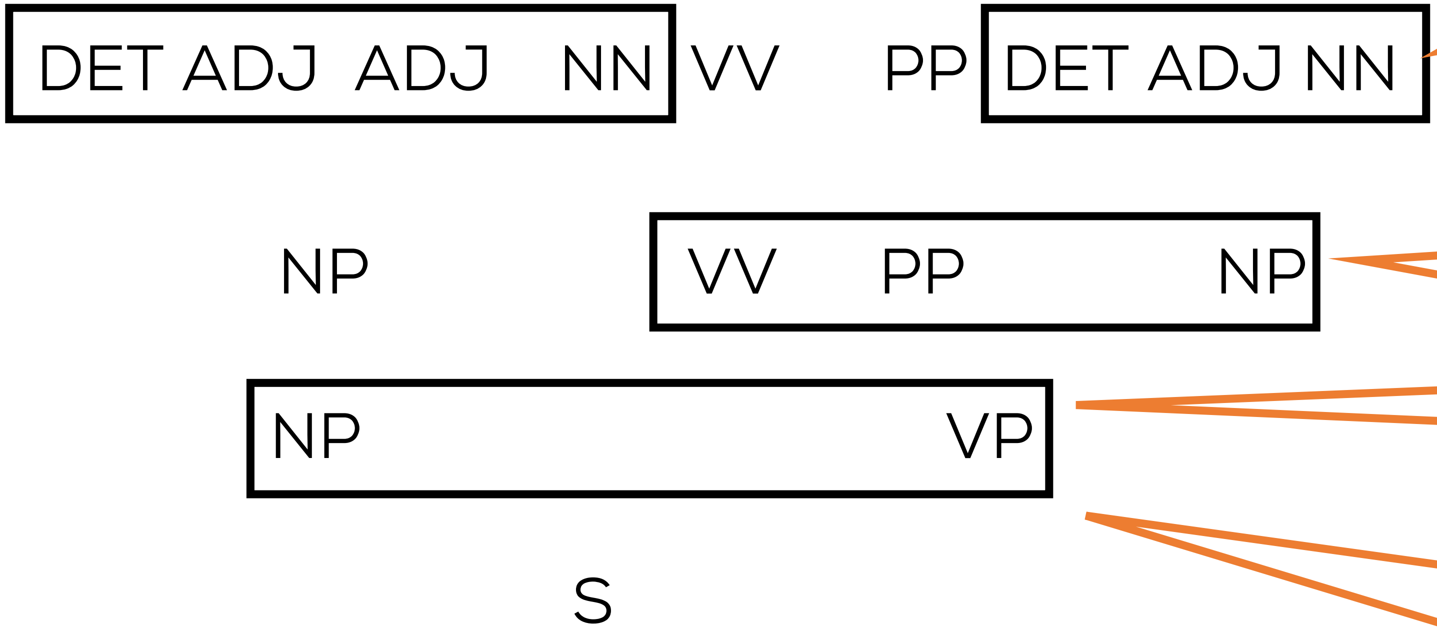
Clues from neighbouring words help identify part of speech, adjective, noun etc

Specific sequences of POS can be combined to form phrases (NP, VP)

This is repeated hierarchically

Text demands Local Features

The quick brown fox jumps over the lazy dog



Clues from neighbouring words help identify part of speech, adjective, noun etc

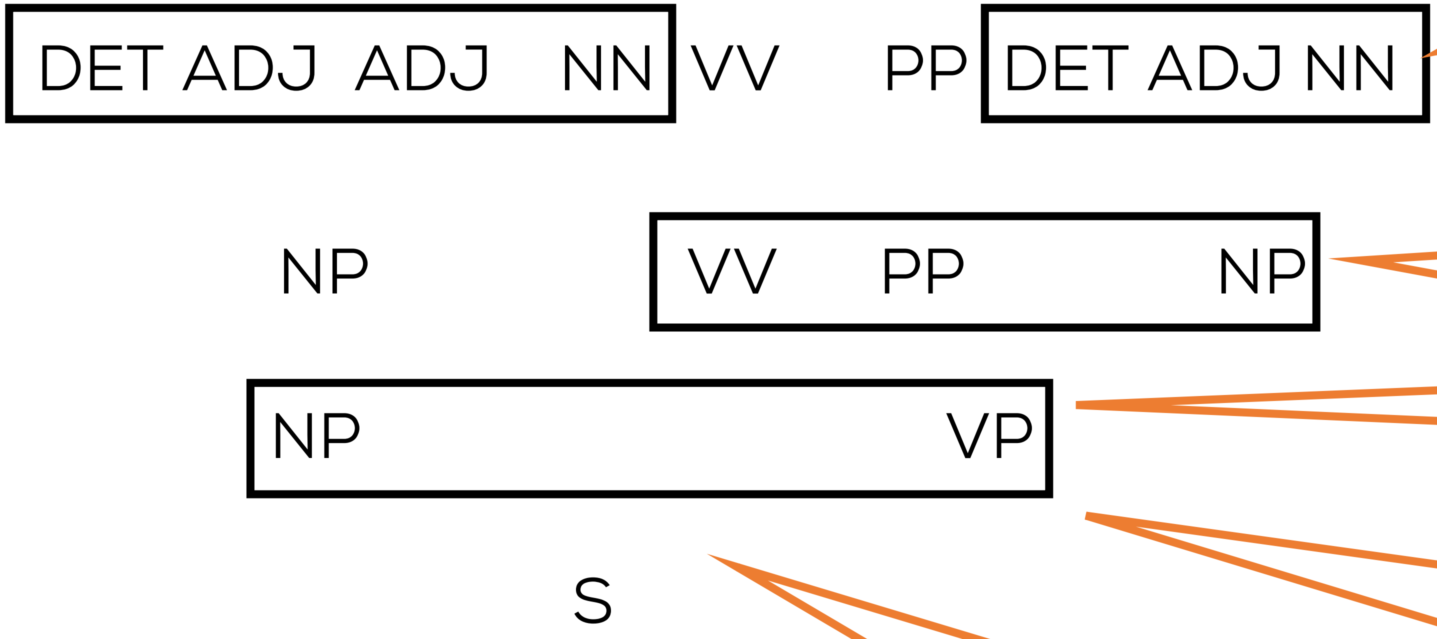
Specific sequences of POS can be combined to form phrases (NP, VP)

This is repeated hierarchically

Note that “fox” and “dog” interact only at the “sentence” level

Text demands Local Features

The quick brown fox jumps over the lazy dog



Clues from neighbouring words help identify part of speech, adjective, noun etc

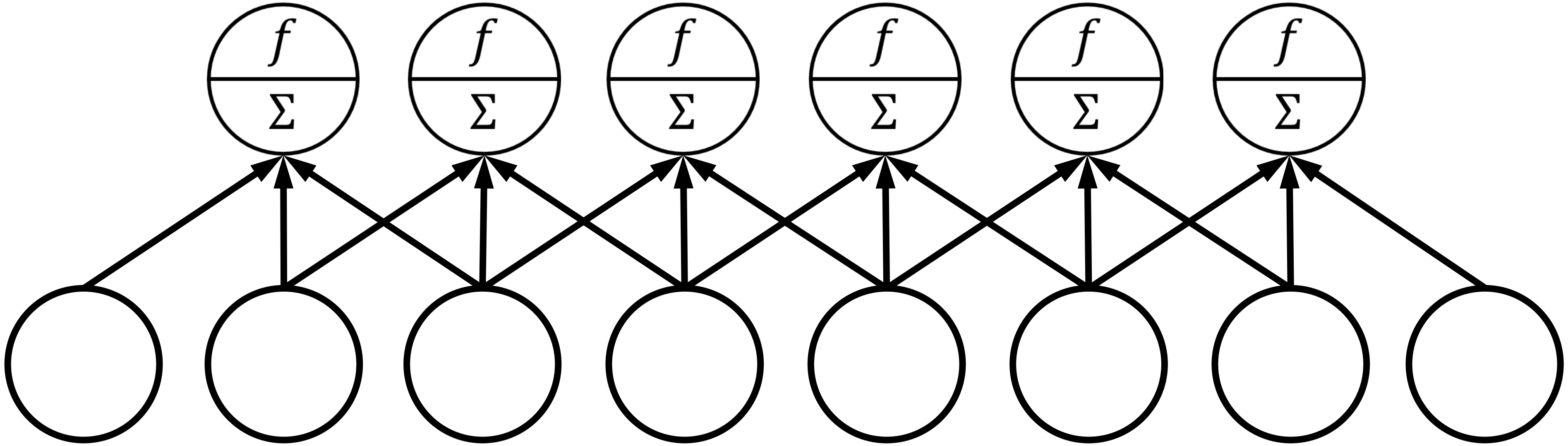
Specific sequences of POS can be combined to form phrases (NP, VP)

This is repeated hierarchically

Note that "fox" and "dog" interact only at the "sentence" level

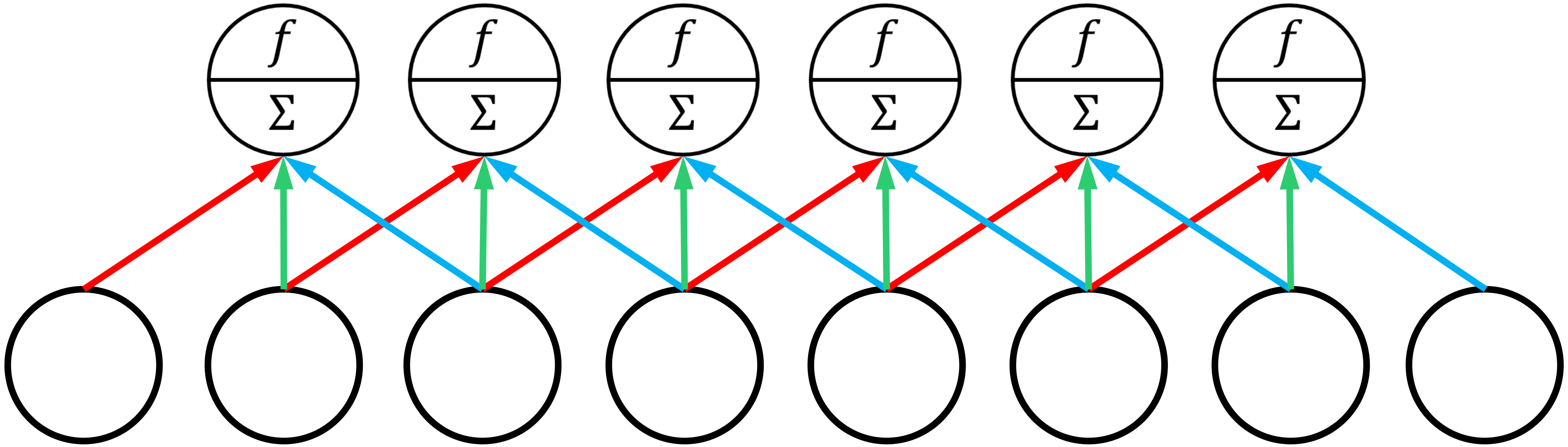
More importantly in a context free grammar, the same rules apply to, e.g., detect a noun phrase no matter where in the sentence we are looking!

Convolution Operation



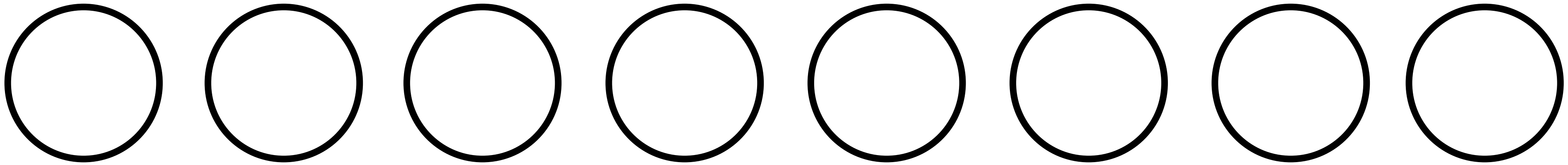
- Only 18 edges, fully connected layer would have had 48 edges

Convolution Operation



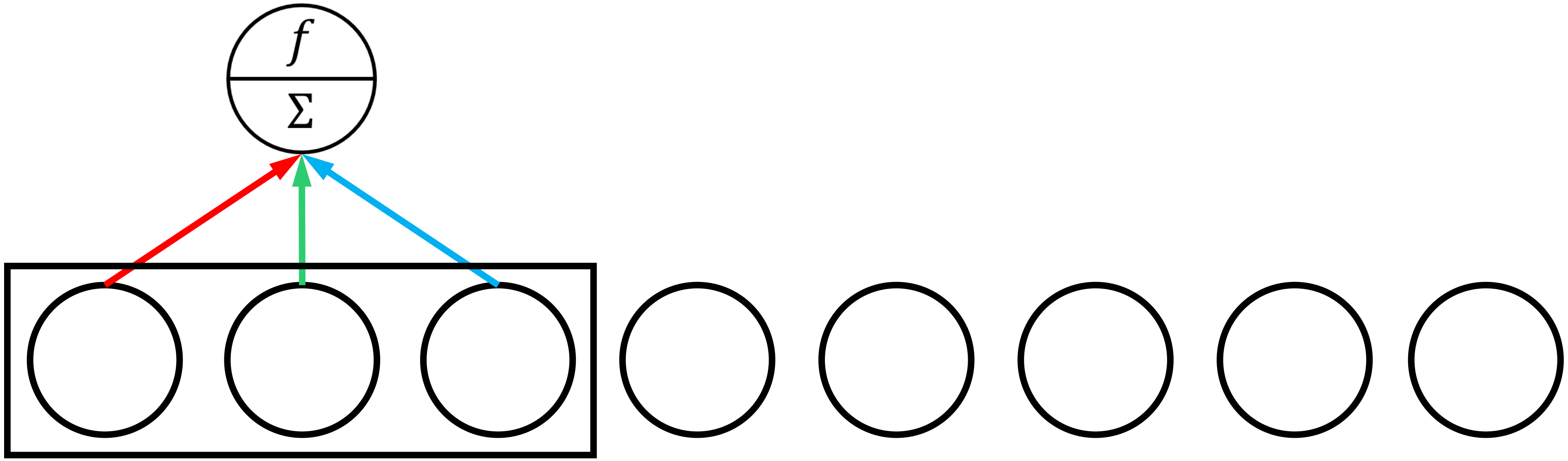
- Only 18 edges, fully connected layer would have had 48 edges
- All green edges forced to have the same weight, all red edges forced to have the same weight, all green edges …
- So effectively only 3 edge weights to be learnt for this layer!

Convolution Operation



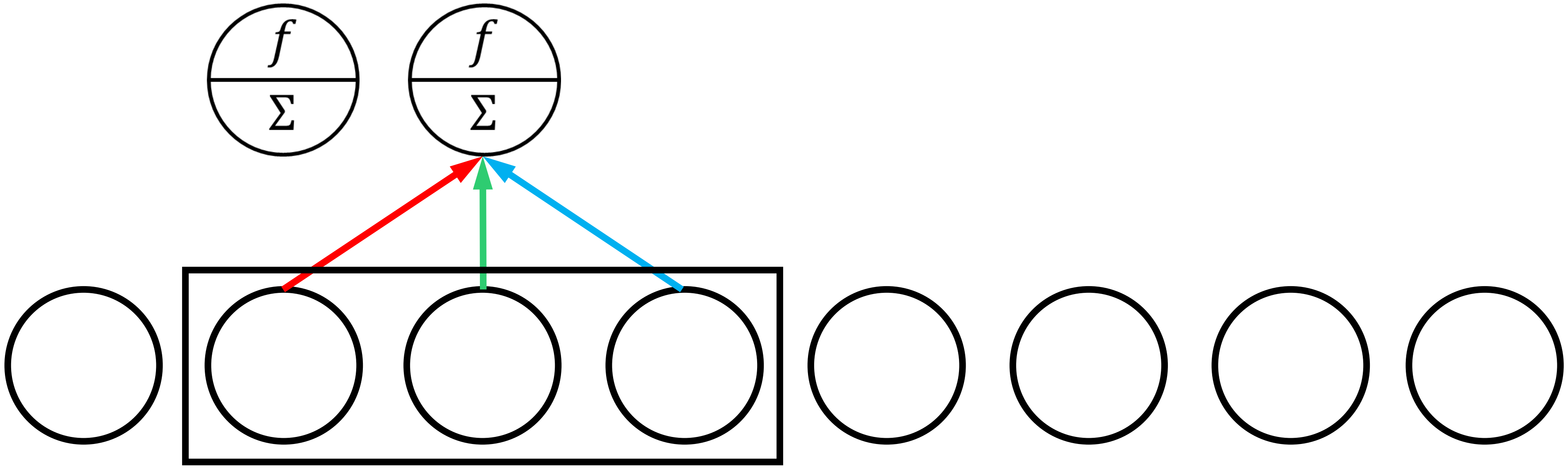
- Only 18 edges, fully connected layer would have had 48 edges
- All green edges forced to have the same weight, all red edges forced to have the same weight, all green edges …
- So effectively only 3 edge weights to be learnt for this layer!

Convolution Operation



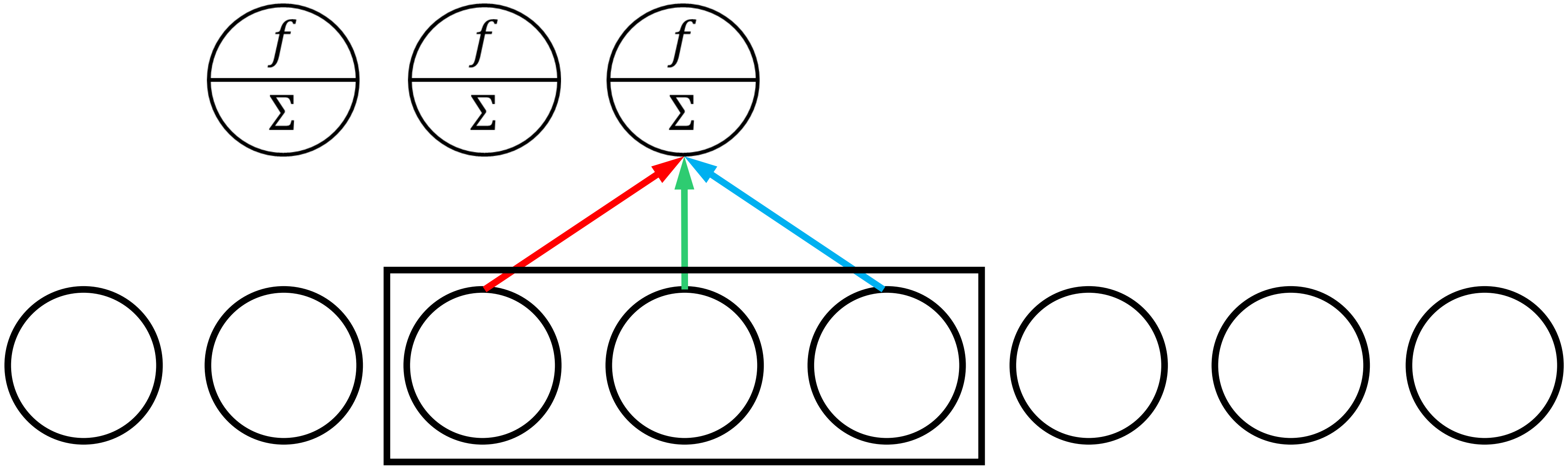
- Only 18 edges, fully connected layer would have had 48 edges
- All green edges forced to have the same weight, all red edges forced to have the same weight, all green edges …
- So effectively only 3 edge weights to be learnt for this layer!

Convolution Operation



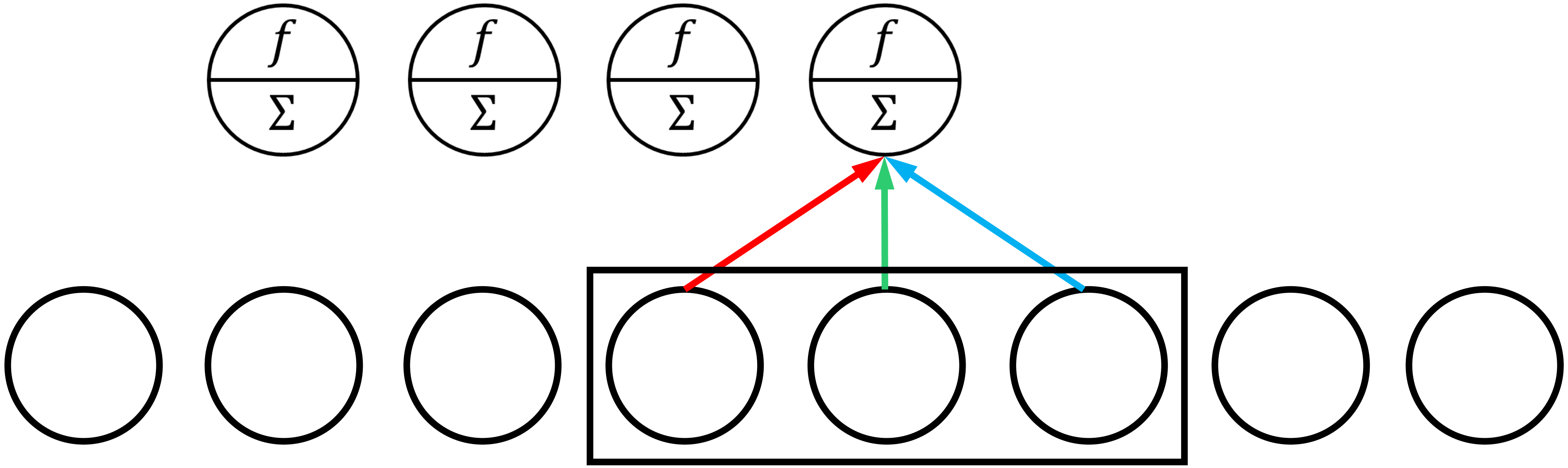
- Only 18 edges, fully connected layer would have had 48 edges
- All green edges forced to have the same weight, all red edges forced to have the same weight, all green edges …
- So effectively only 3 edge weights to be learnt for this layer!

Convolution Operation



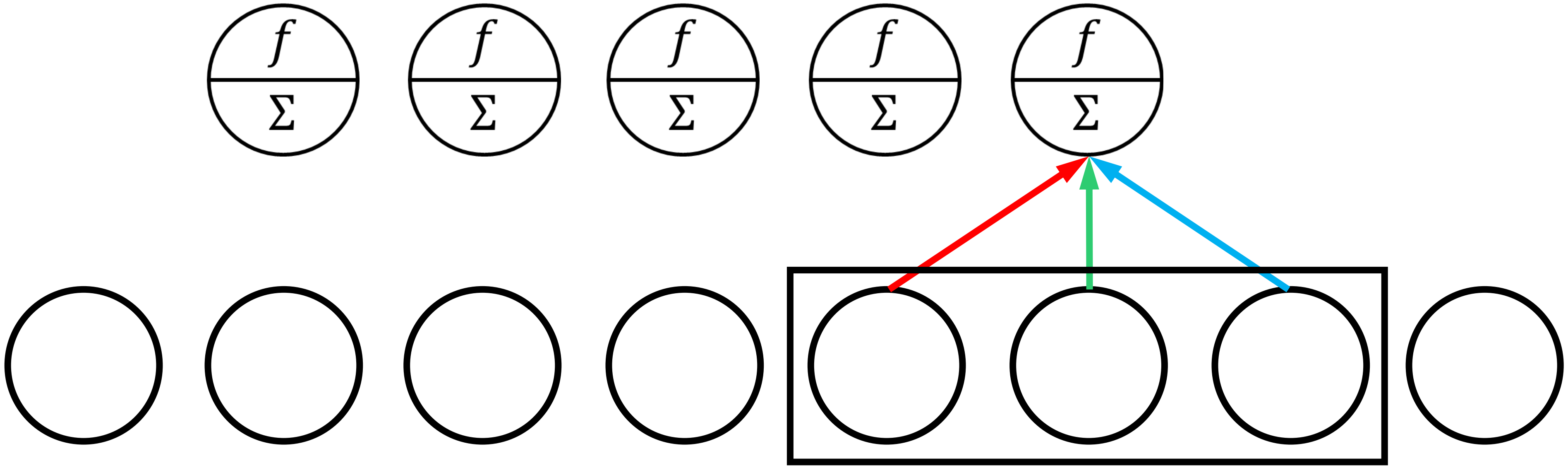
- Only 18 edges, fully connected layer would have had 48 edges
- All green edges forced to have the same weight, all red edges forced to have the same weight, all green edges …
- So effectively only 3 edge weights to be learnt for this layer!

Convolution Operation



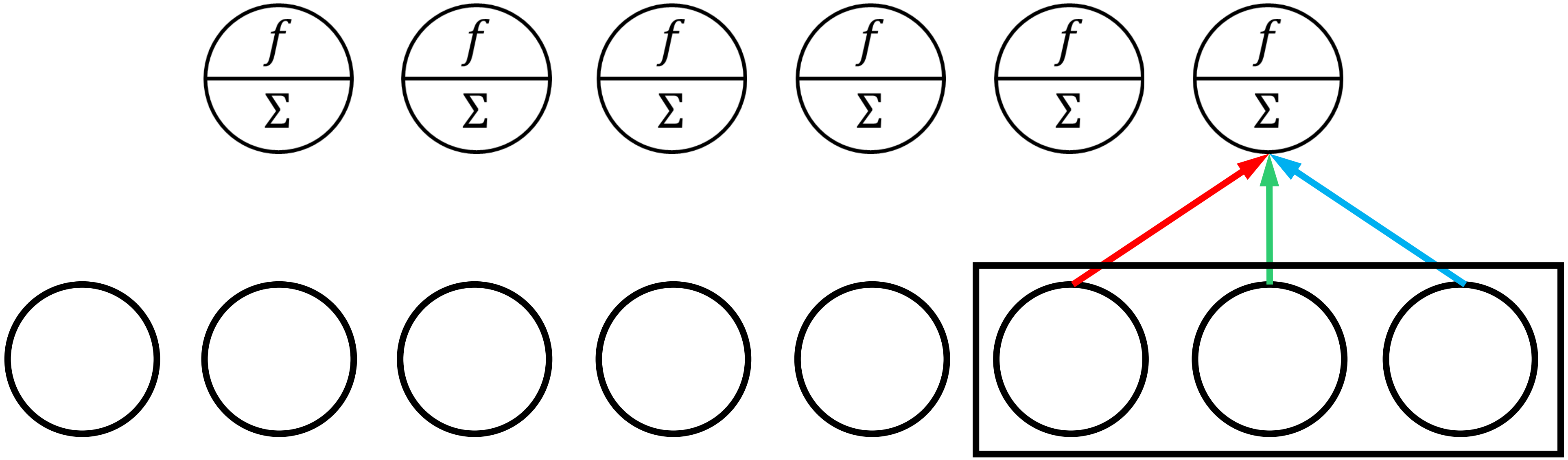
- Only 18 edges, fully connected layer would have had 48 edges
- All green edges forced to have the same weight, all red edges forced to have the same weight, all green edges …
- So effectively only 3 edge weights to be learnt for this layer!

Convolution Operation



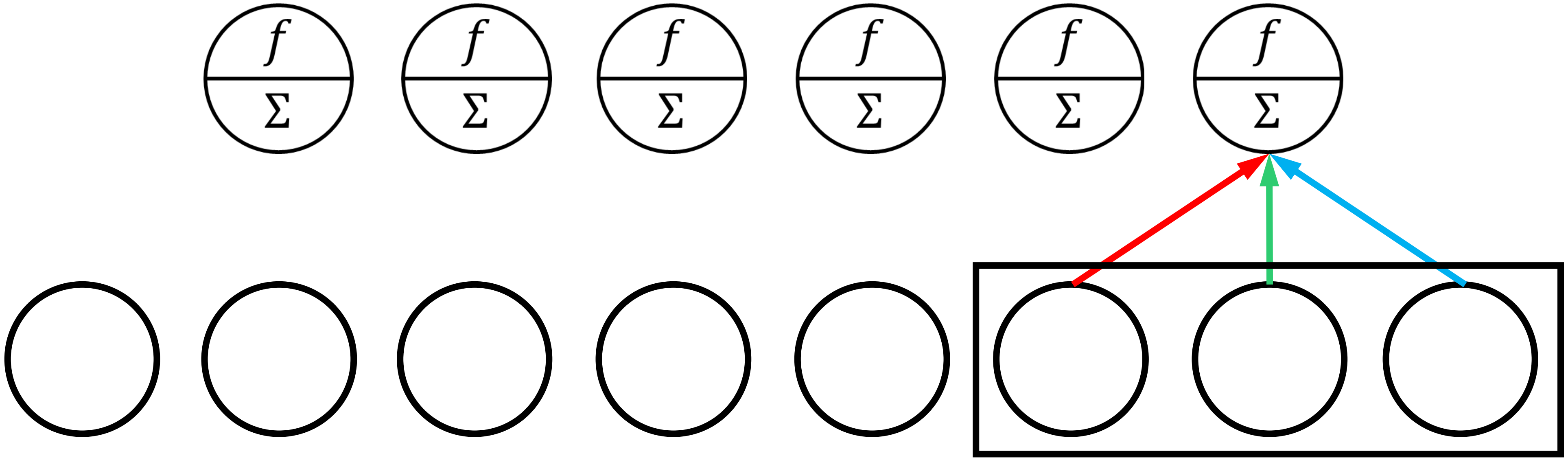
- Only 18 edges, fully connected layer would have had 48 edges
- All green edges forced to have the same weight, all red edges forced to have the same weight, all green edges …
- So effectively only 3 edge weights to be learnt for this layer!

Convolution Operation



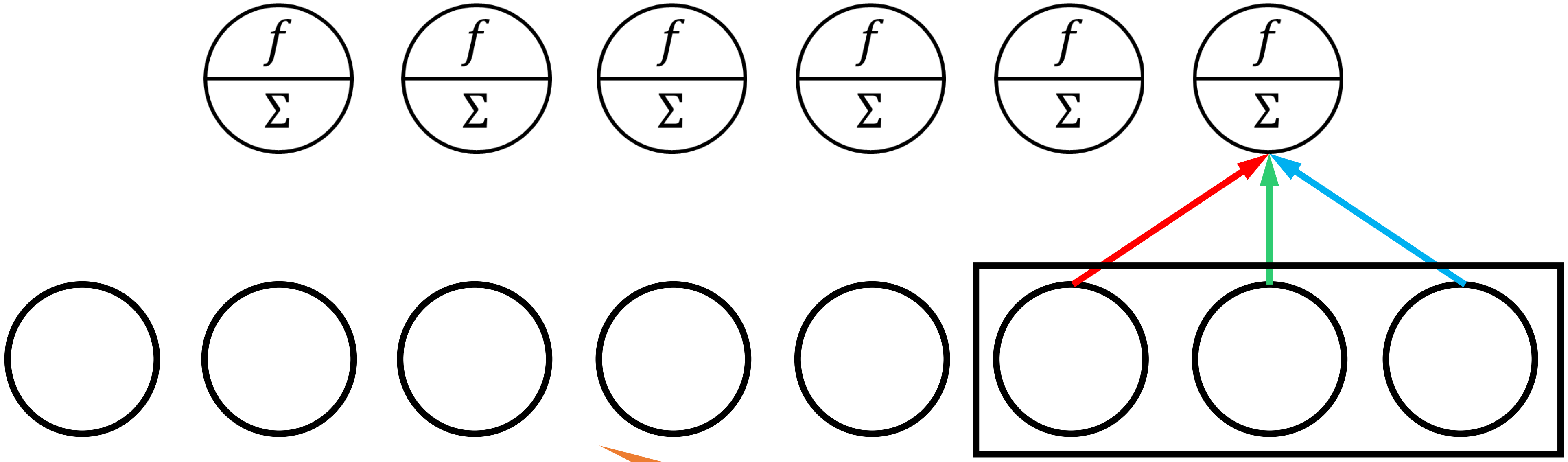
- Only 18 edges, fully connected layer would have had 48 edges
- All green edges forced to have the same weight, all red edges forced to have the same weight, all green edges …
- So effectively only 3 edge weights to be learnt for this layer!

Convolution Operation



- Only 18 edges, fully connected layer would have had 48 edges
- All green edges forced to have the same weight, all red edges forced to have the same weight, all green edges …
- So effectively only 3 edge weights to be learnt for this layer!
- Formally called a convolution operation!

Convolution Operation

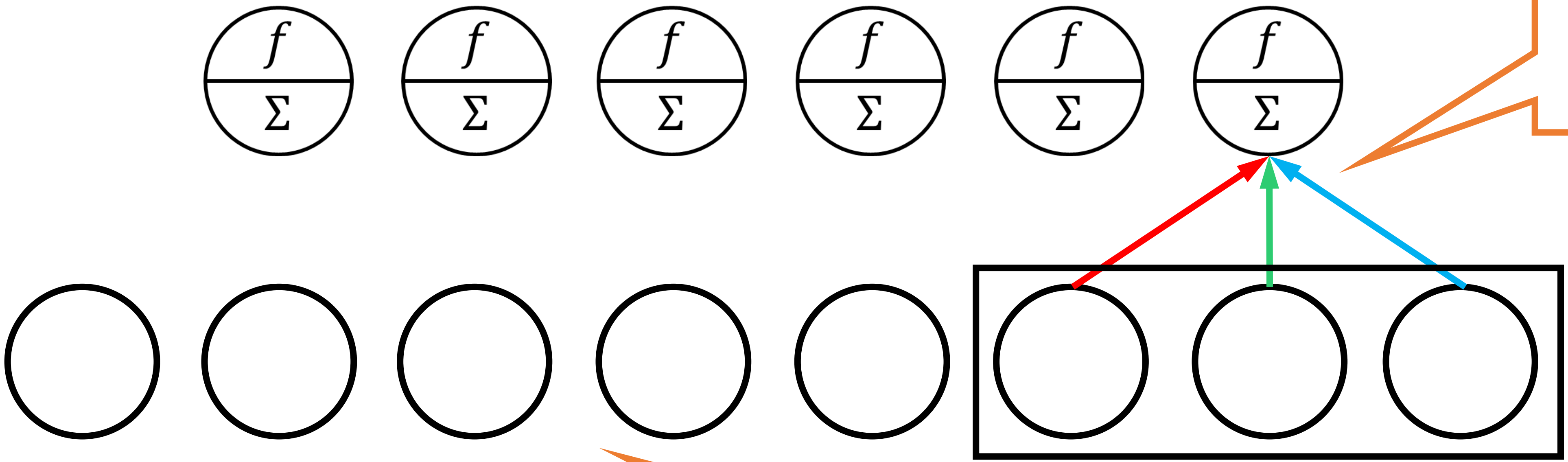


$$\mathbf{h}_i = f(\mathbf{x}_i \cdot \mathbf{w}_1 + \mathbf{x}_{i+1} \cdot \mathbf{w}_2 + \mathbf{x}_{i+3} \cdot \mathbf{w}_3)$$

$$\mathbf{h}_i = f\left(\sum_{j=1}^{3} \mathbf{x}_{i+j} \cdot \mathbf{w}_j\right)$$

- Only 18 edges, fully con ted
- All green edges forced to have forced to have the same weigh
- So effectively only 3 edge weig
- Formally called a convolution operation!

Convolution Operation



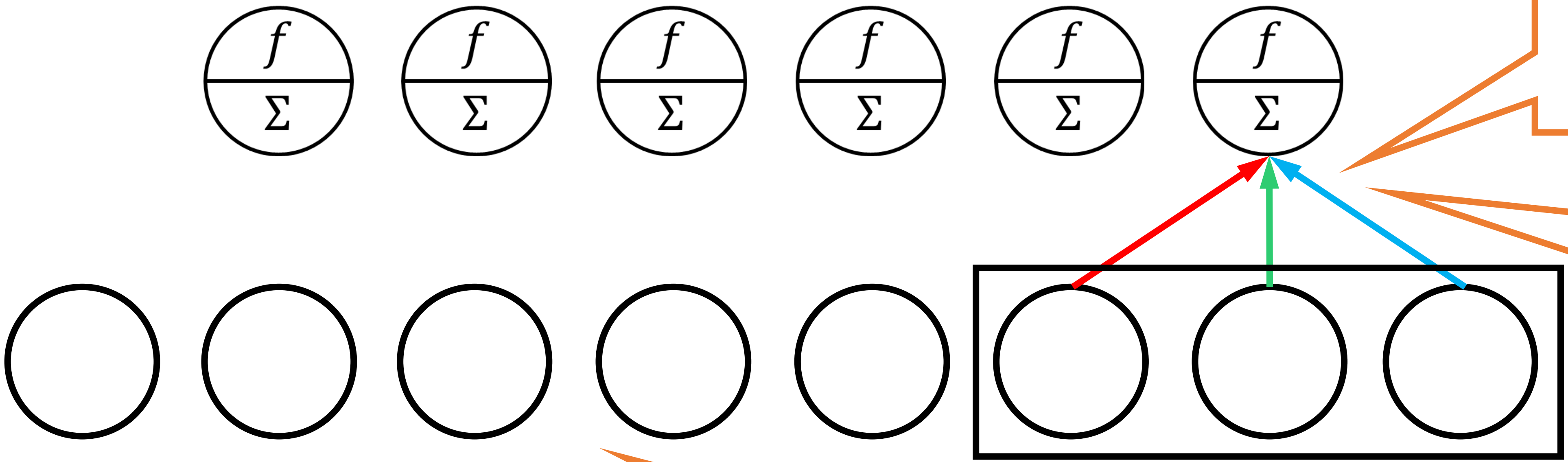
w often called a convolutional "kernel"

- Only 18 edges, fully conn...ted
- All green edges forced to have... forced to have the same weigh...
- So effectively only 3 edge weig...
- Formally called a convolution operation!

$$\mathbf{h}_i = f(\mathbf{x}_i \cdot \mathbf{w}_1 + \mathbf{x}_{i+1} \cdot \mathbf{w}_2 + \mathbf{x}_{i+3} \cdot \mathbf{w}_3)$$

$$\mathbf{h}_i = f\left(\sum_{j=1}^{3} \mathbf{x}_{i+j} \cdot \mathbf{w}_j\right)$$

Convolution Operation



w often called a convolutional “kernel”

Don’t confuse with Mercer kernels

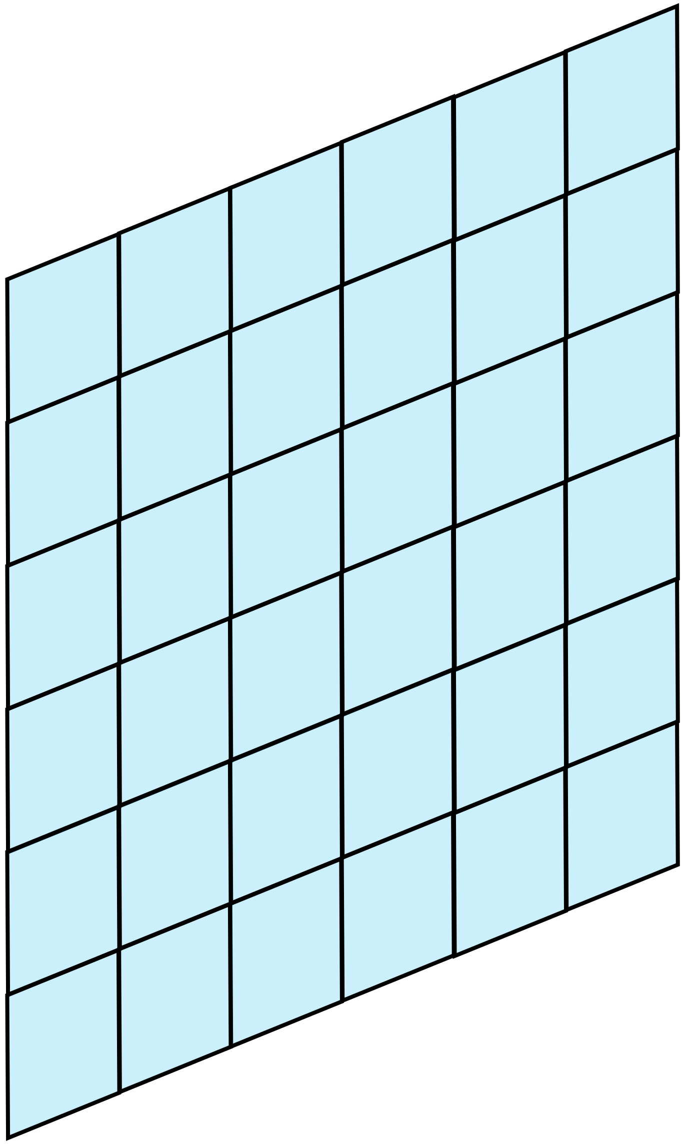
$$\mathbf{h}_i = f(\mathbf{x}_i \cdot \mathbf{w}_1 + \mathbf{x}_{i+1} \cdot \mathbf{w}_2 + \mathbf{x}_{i+3} \cdot \mathbf{w}_3)$$

$$\mathbf{h}_i = f\left(\sum_{j=1}^{3} \mathbf{x}_{i+j} \cdot \mathbf{w}_j\right)$$

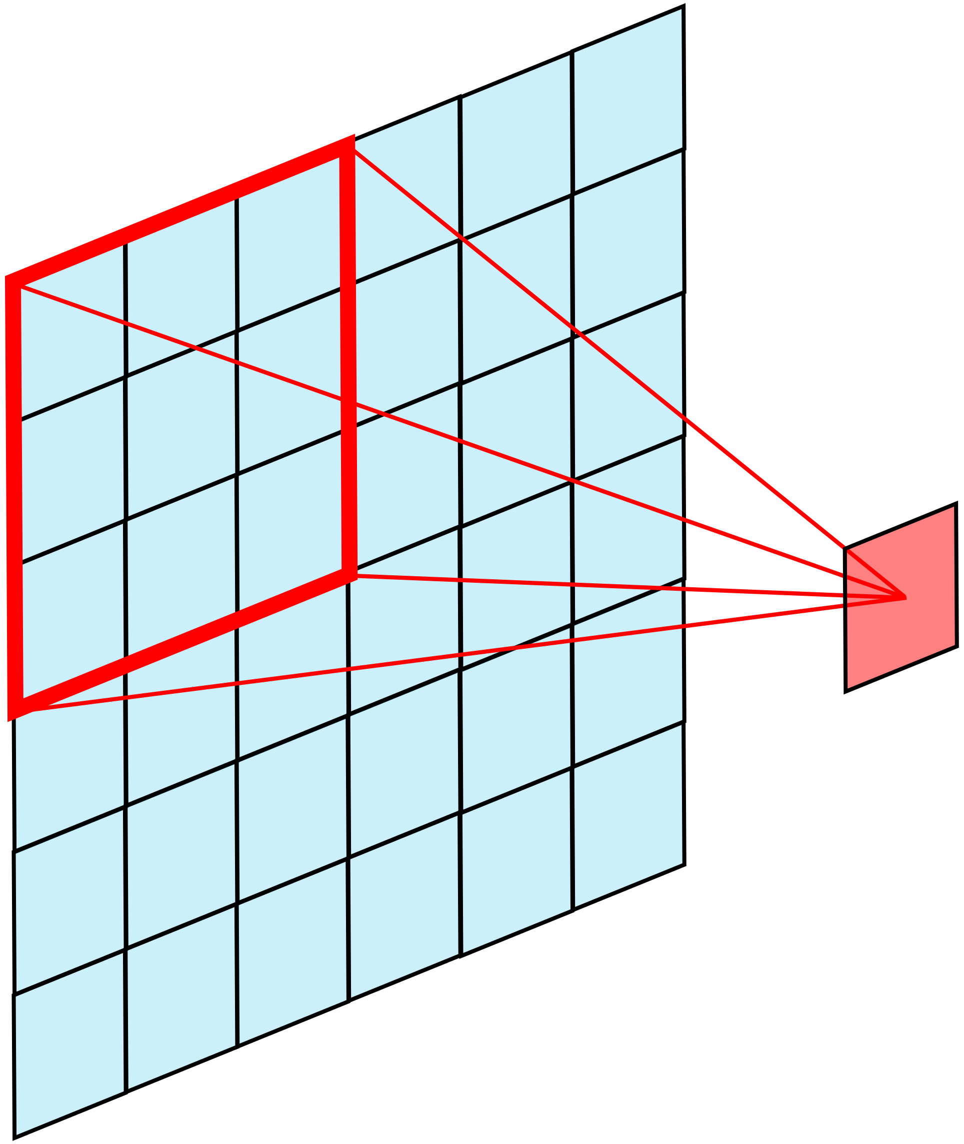
- Only 18 edges, fully con ted
- All green edges forced to have forced to have the same weig
- So effectively only 3 edge weig
- Formally called a convolution operation!

2D Convolution Operation

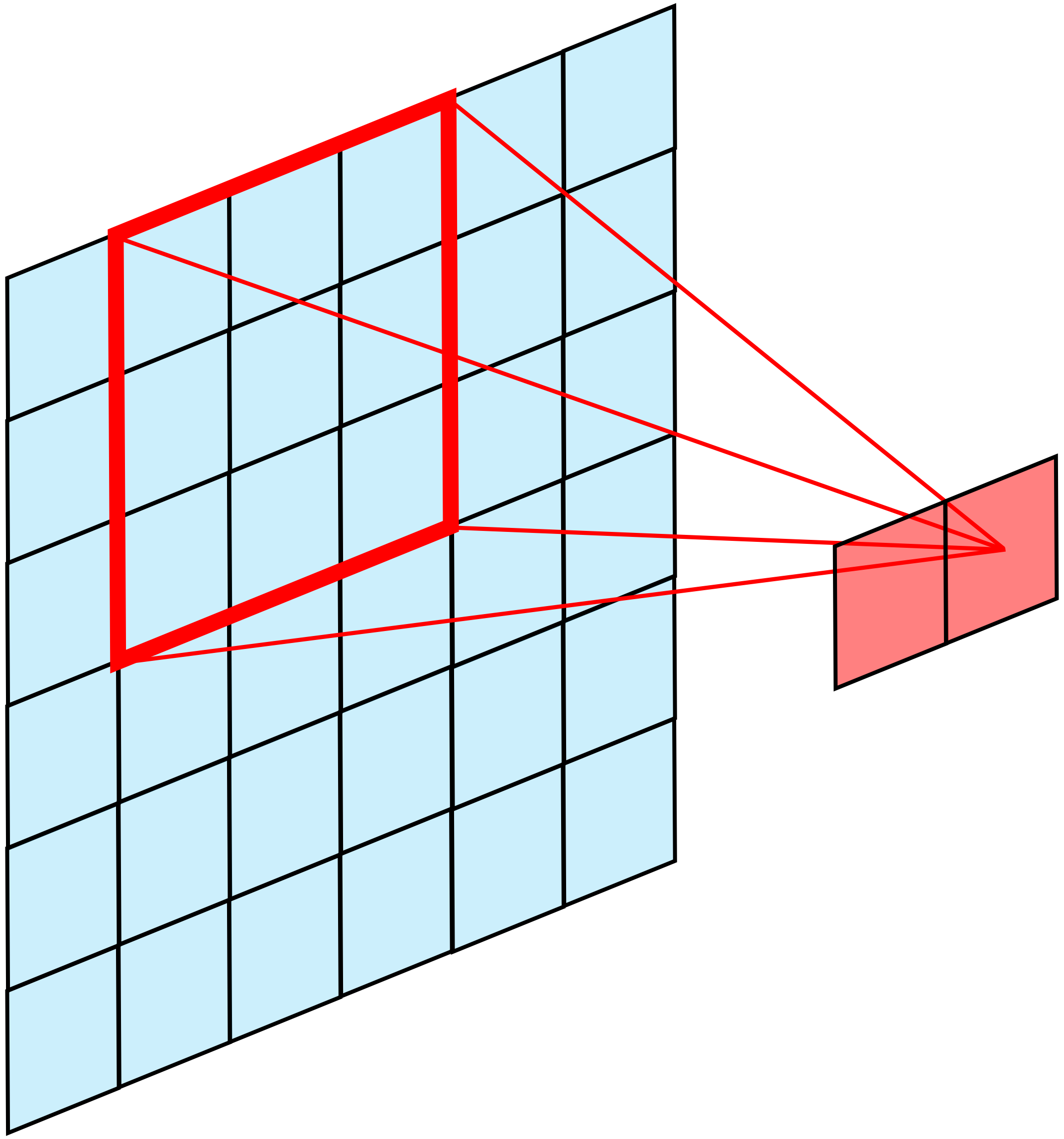
2D Convolution Operation



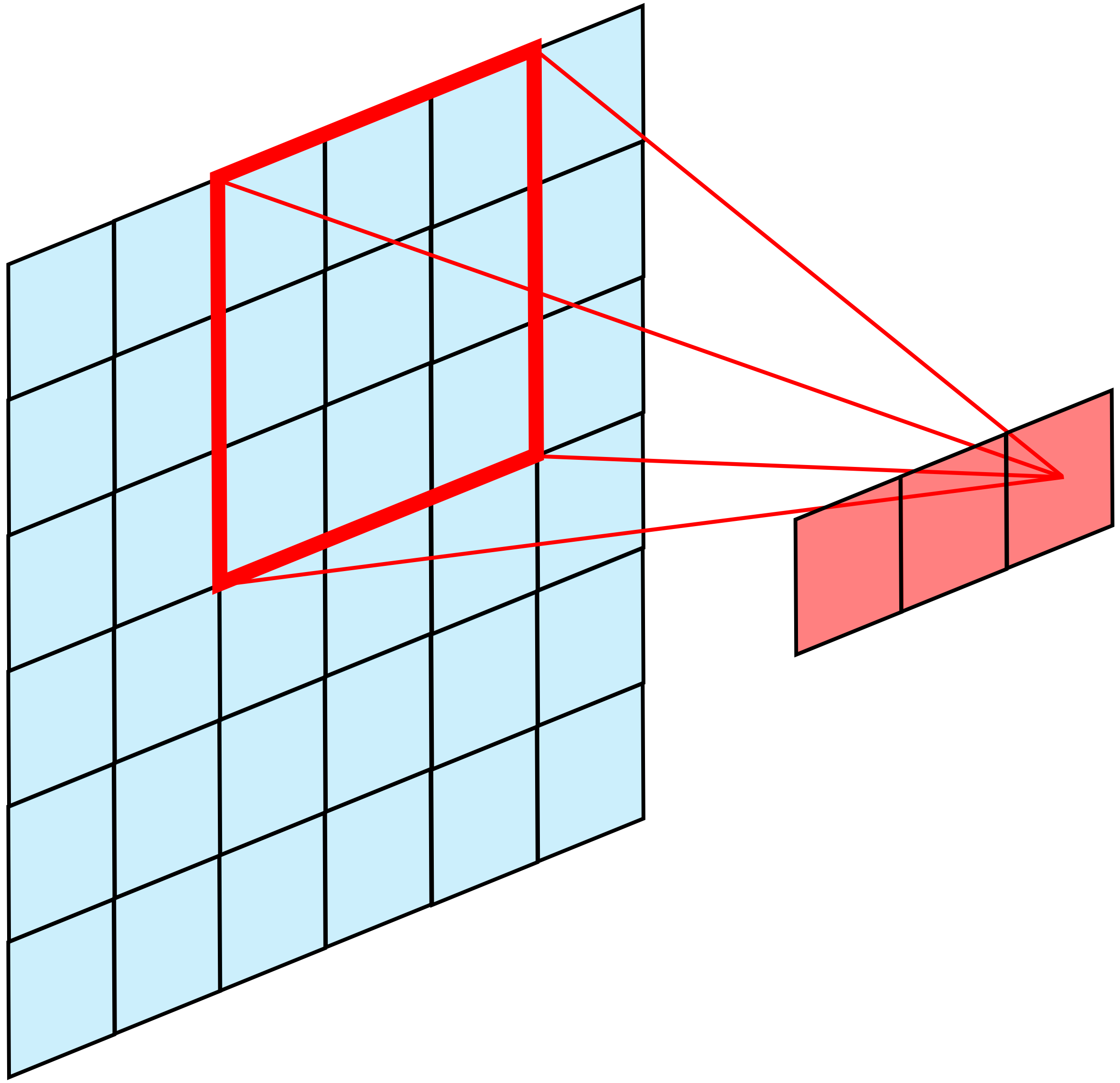
2D Convolution Operation



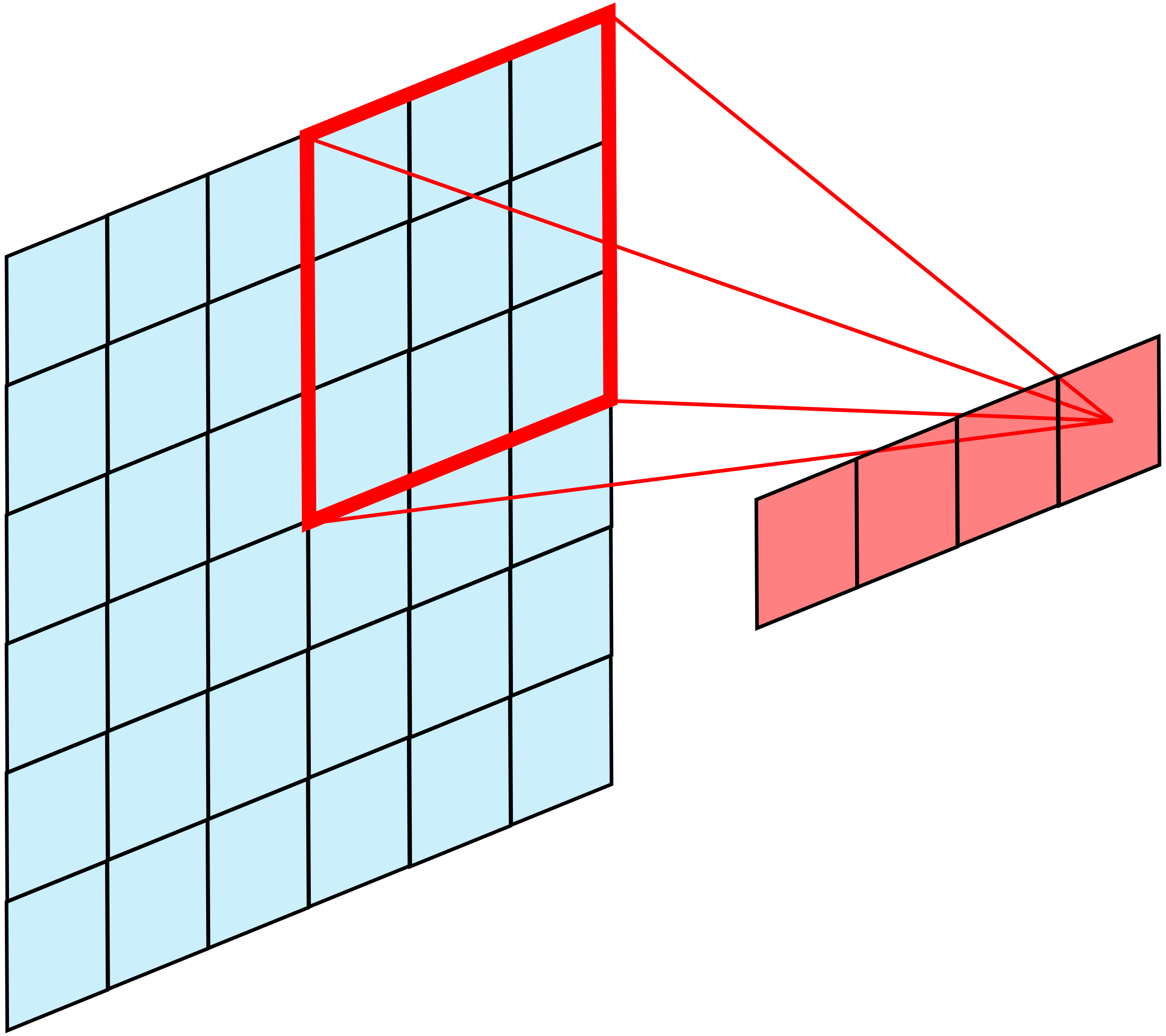
2D Convolution Operation



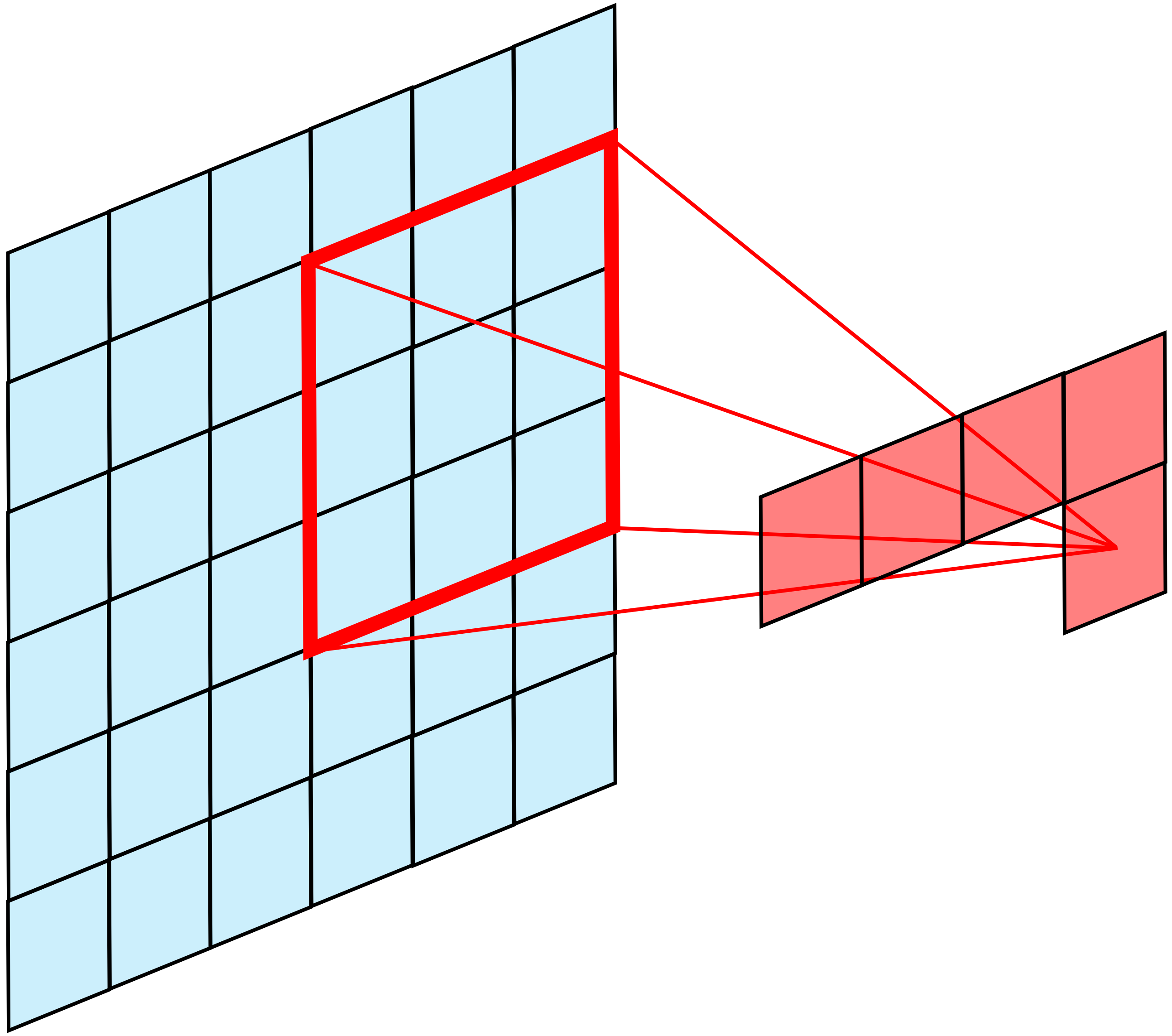
2D Convolution Operation



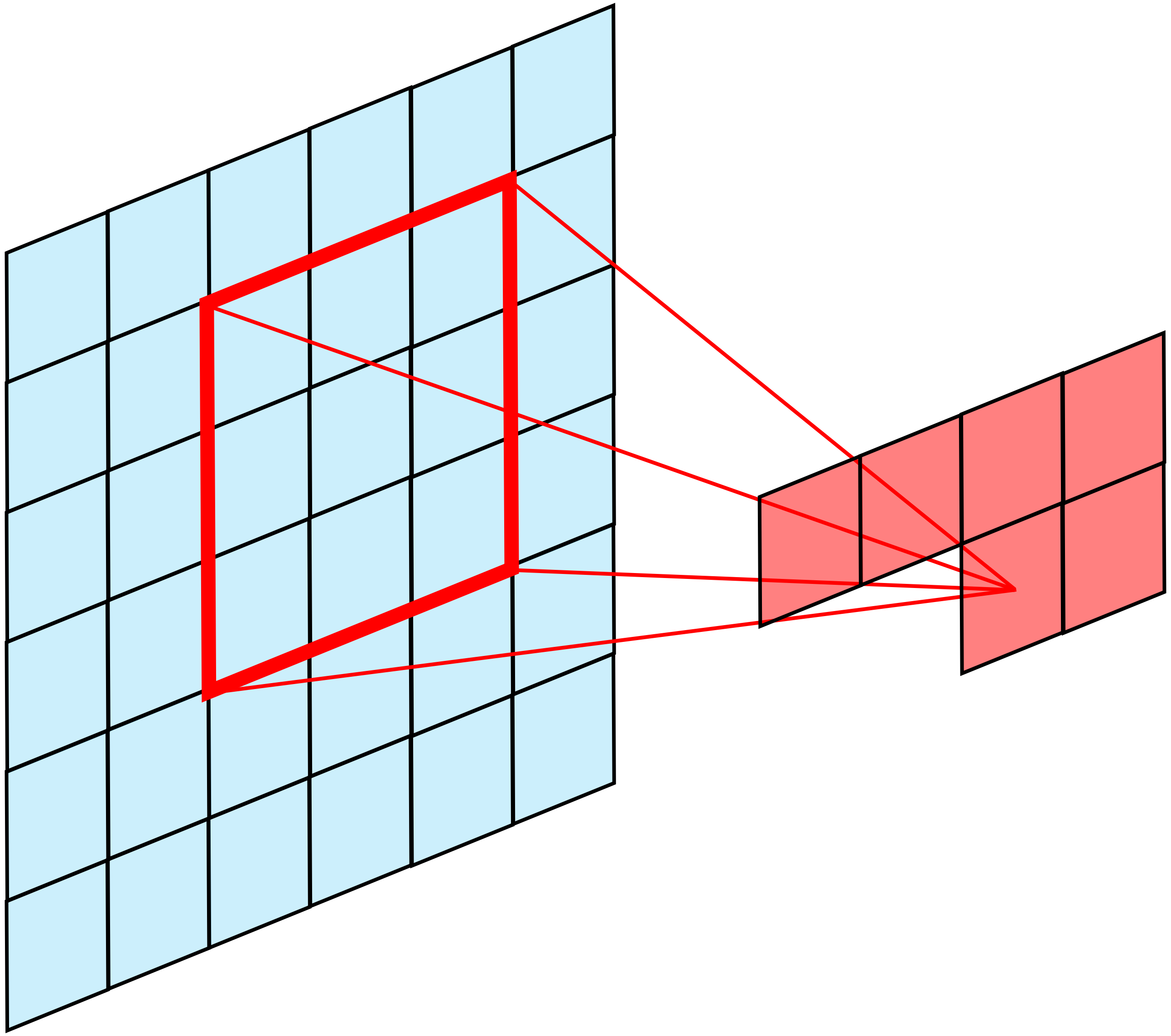
2D Convolution Operation



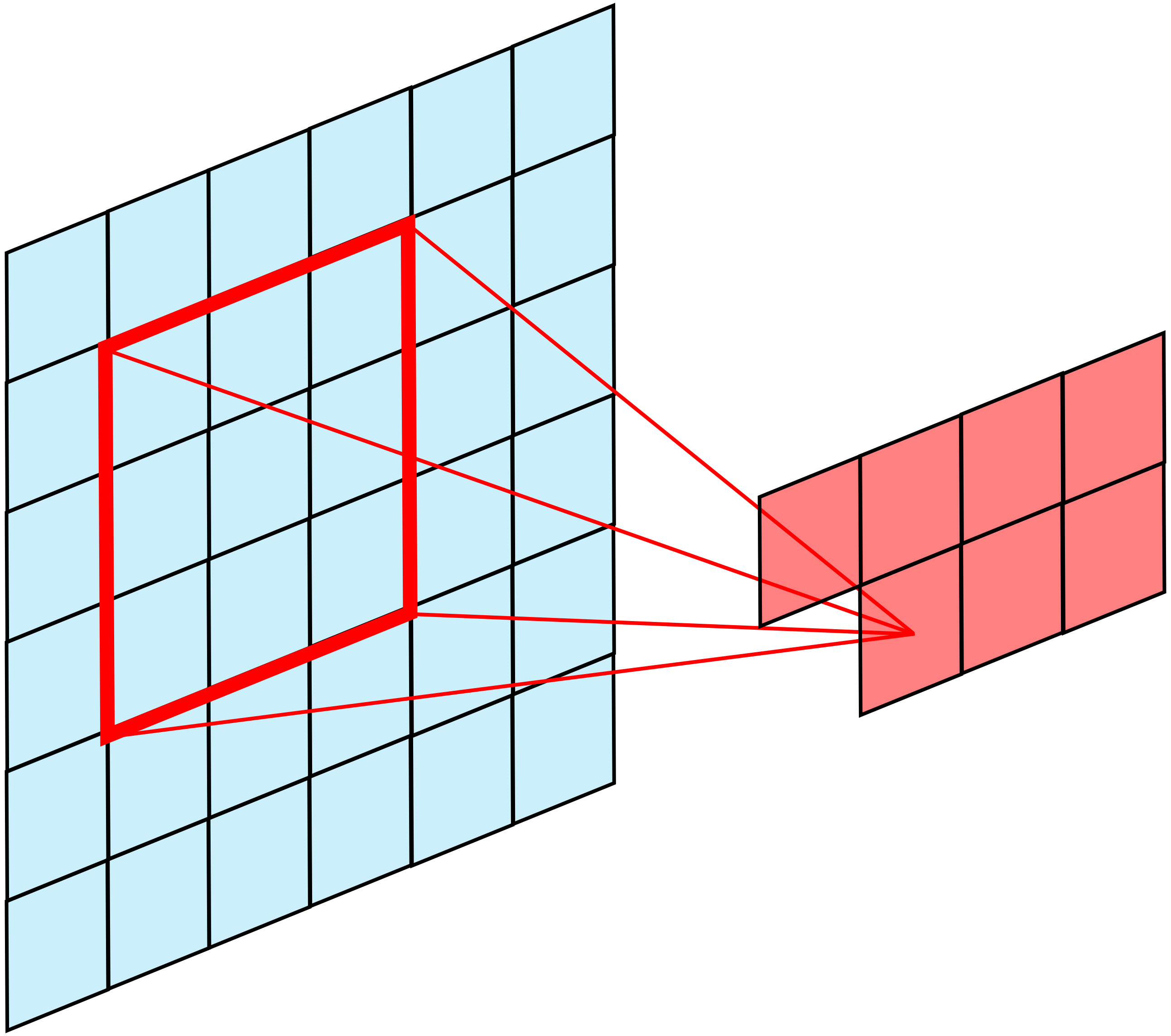
2D Convolution Operation



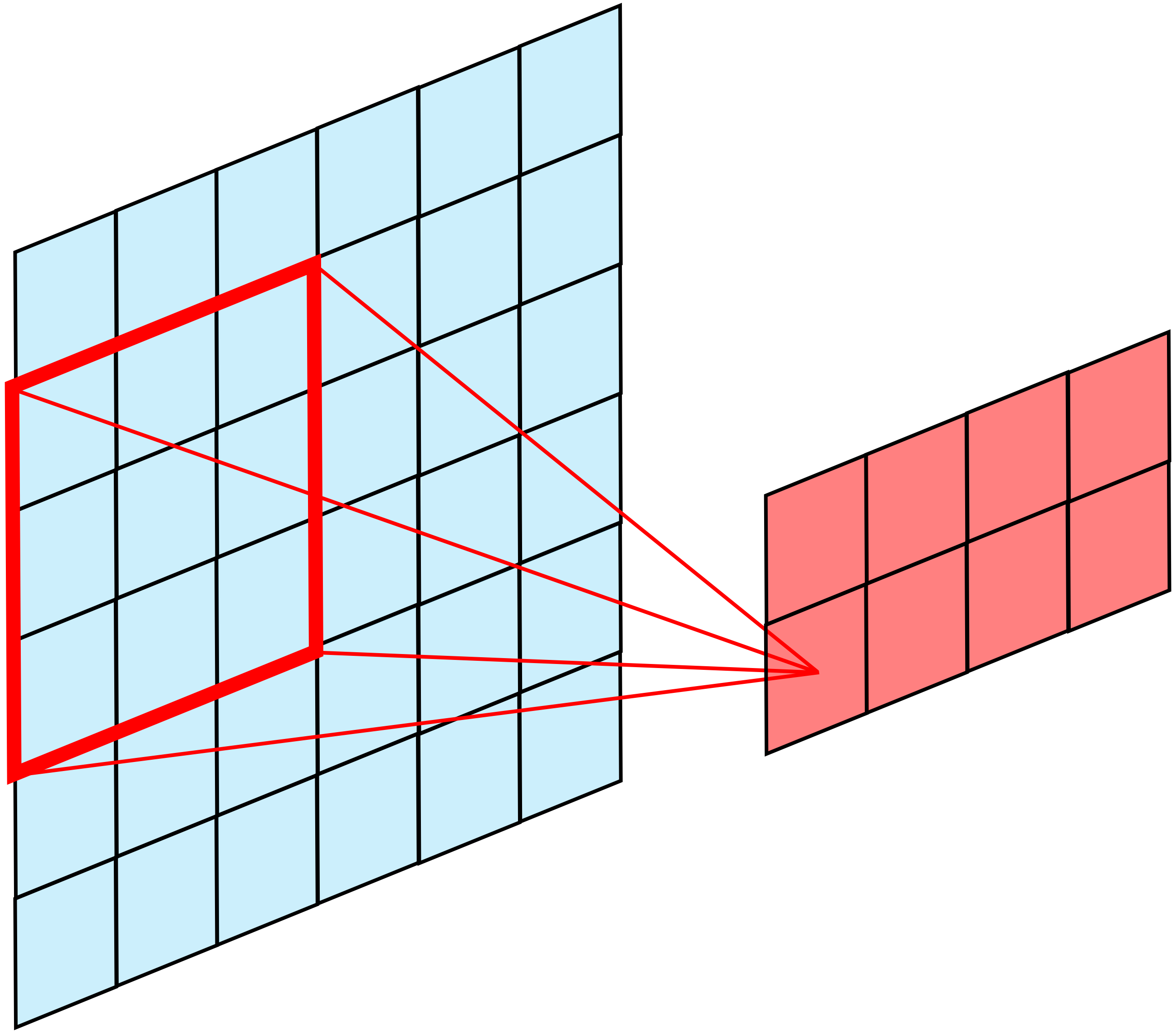
2D Convolution Operation



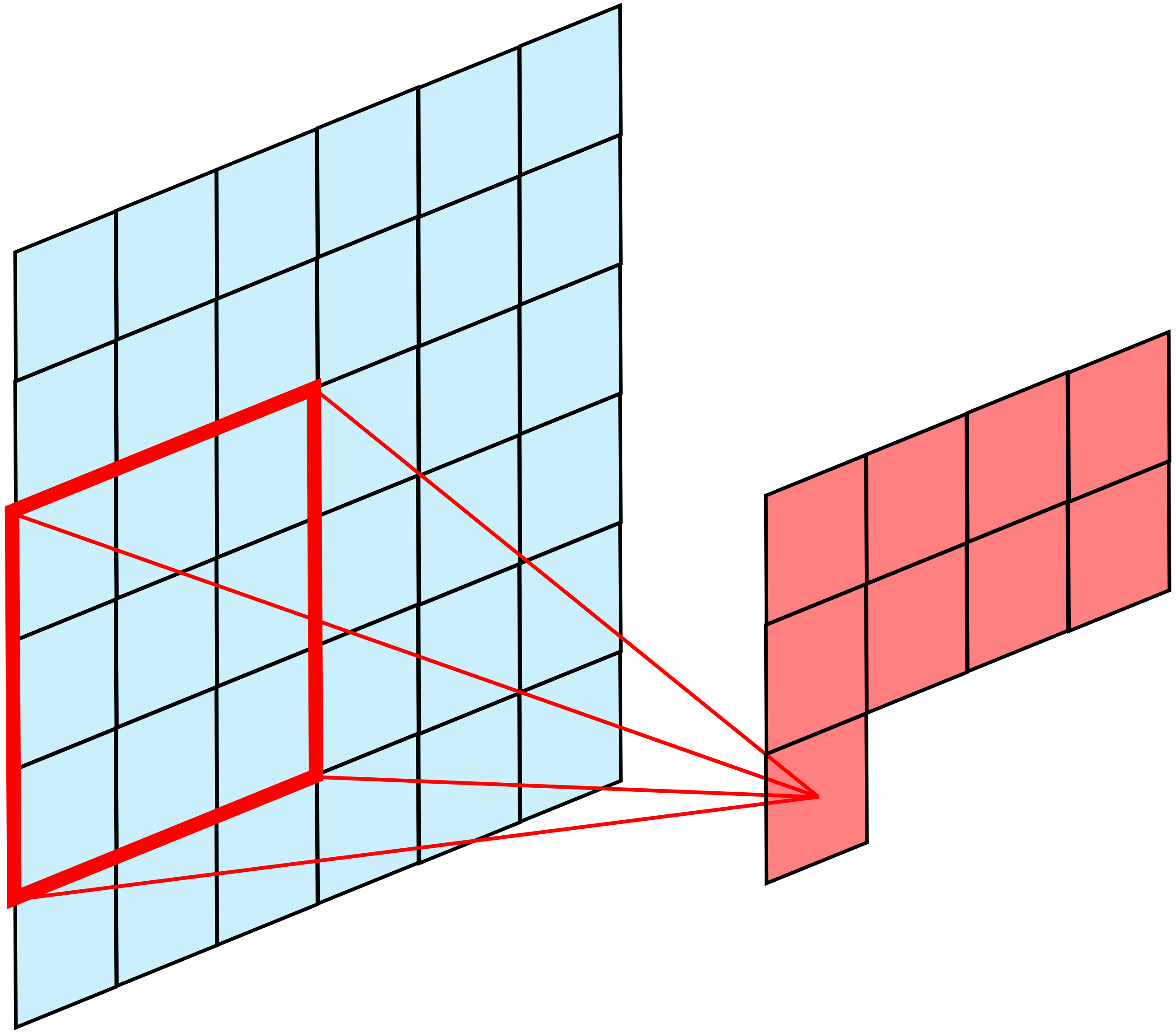
2D Convolution Operation



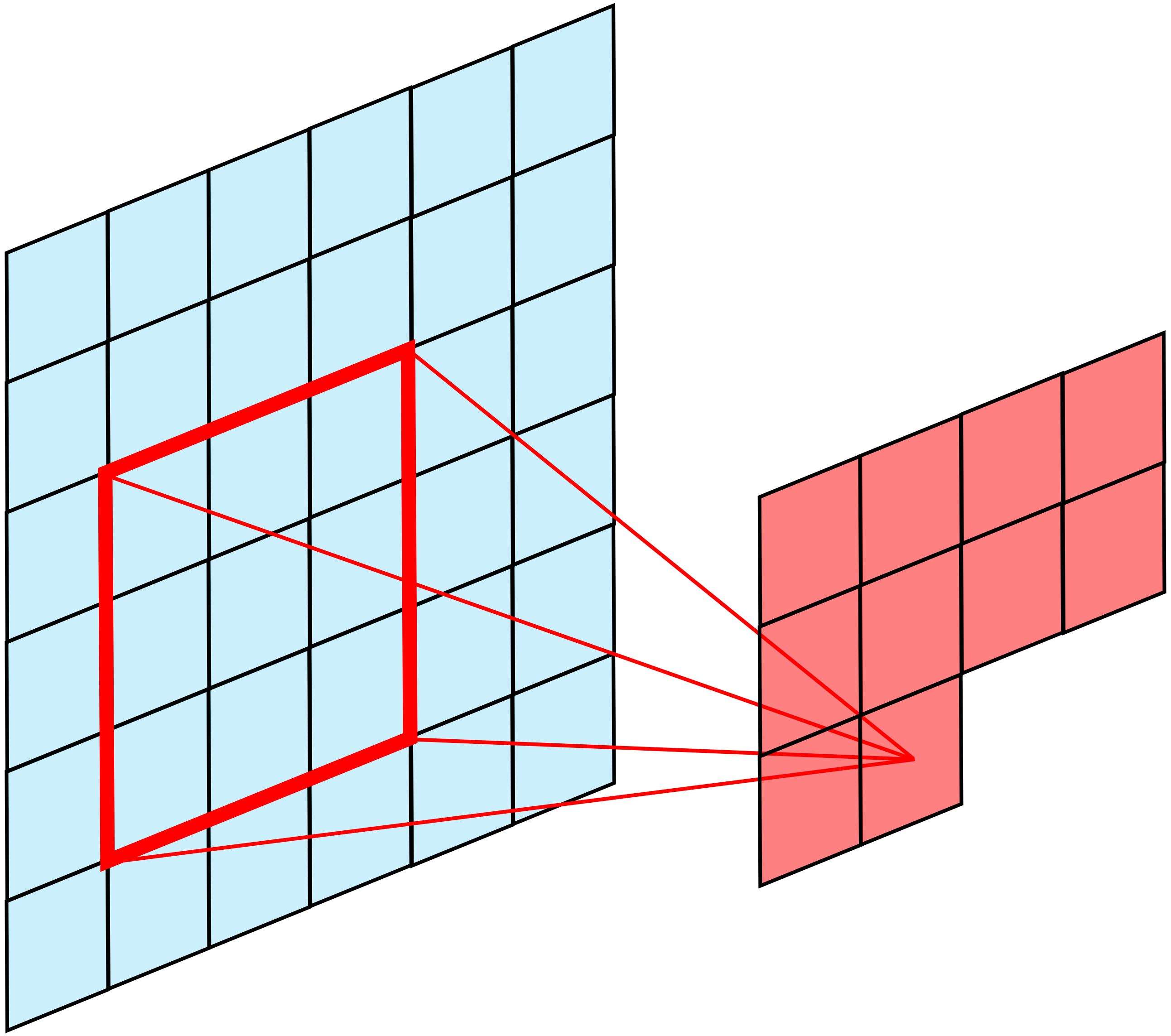
2D Convolution Operation



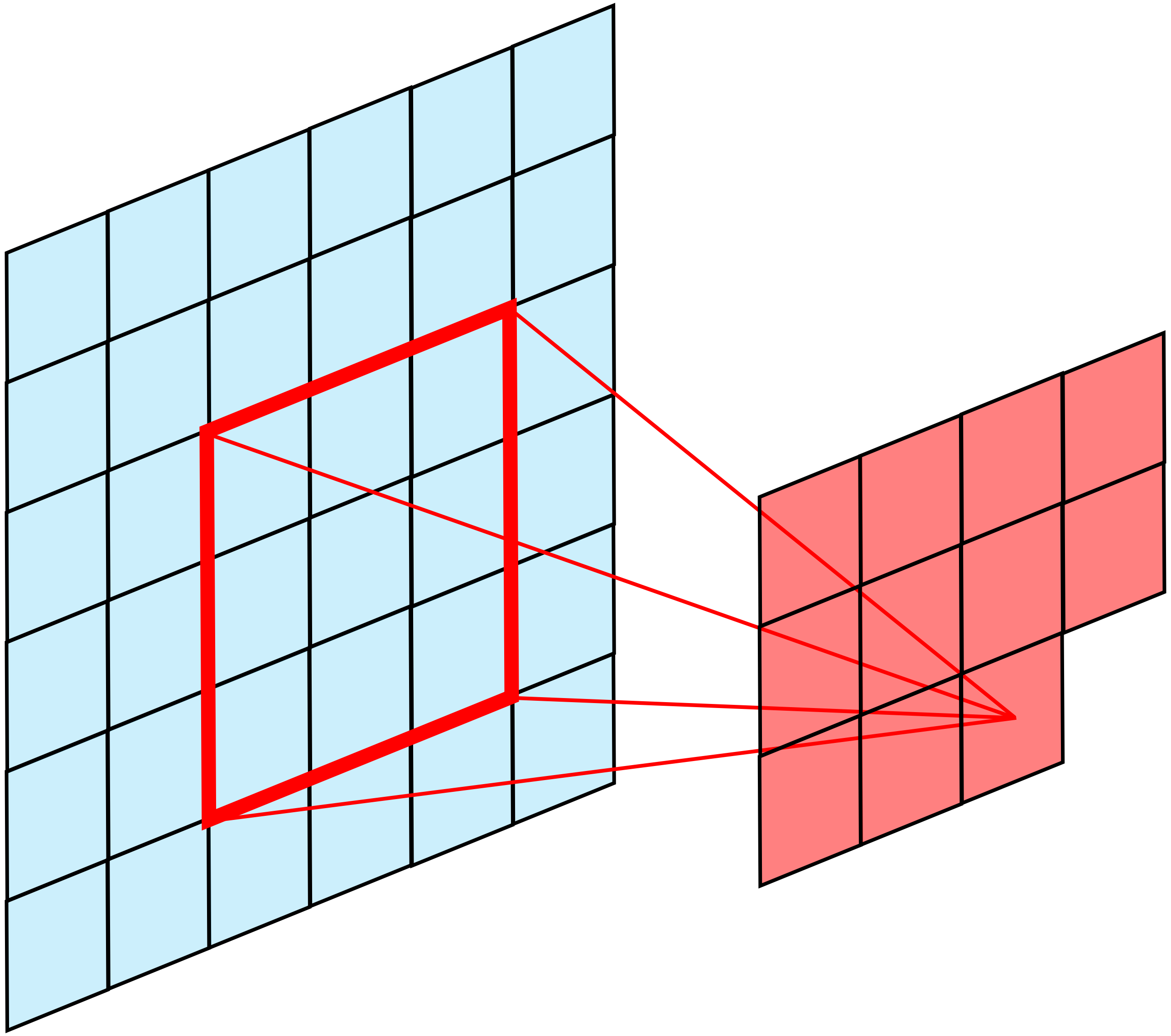
2D Convolution Operation



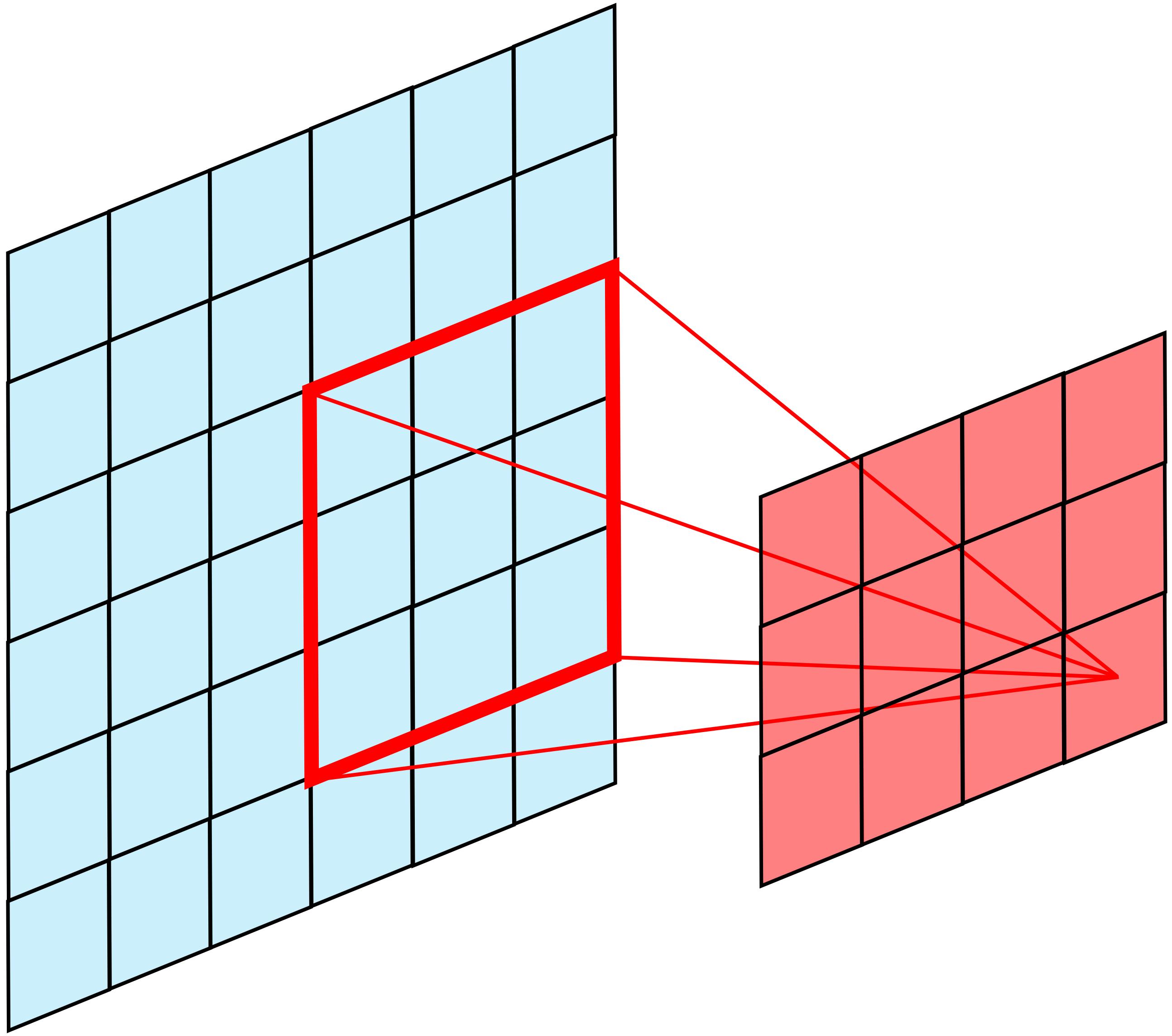
2D Convolution Operation



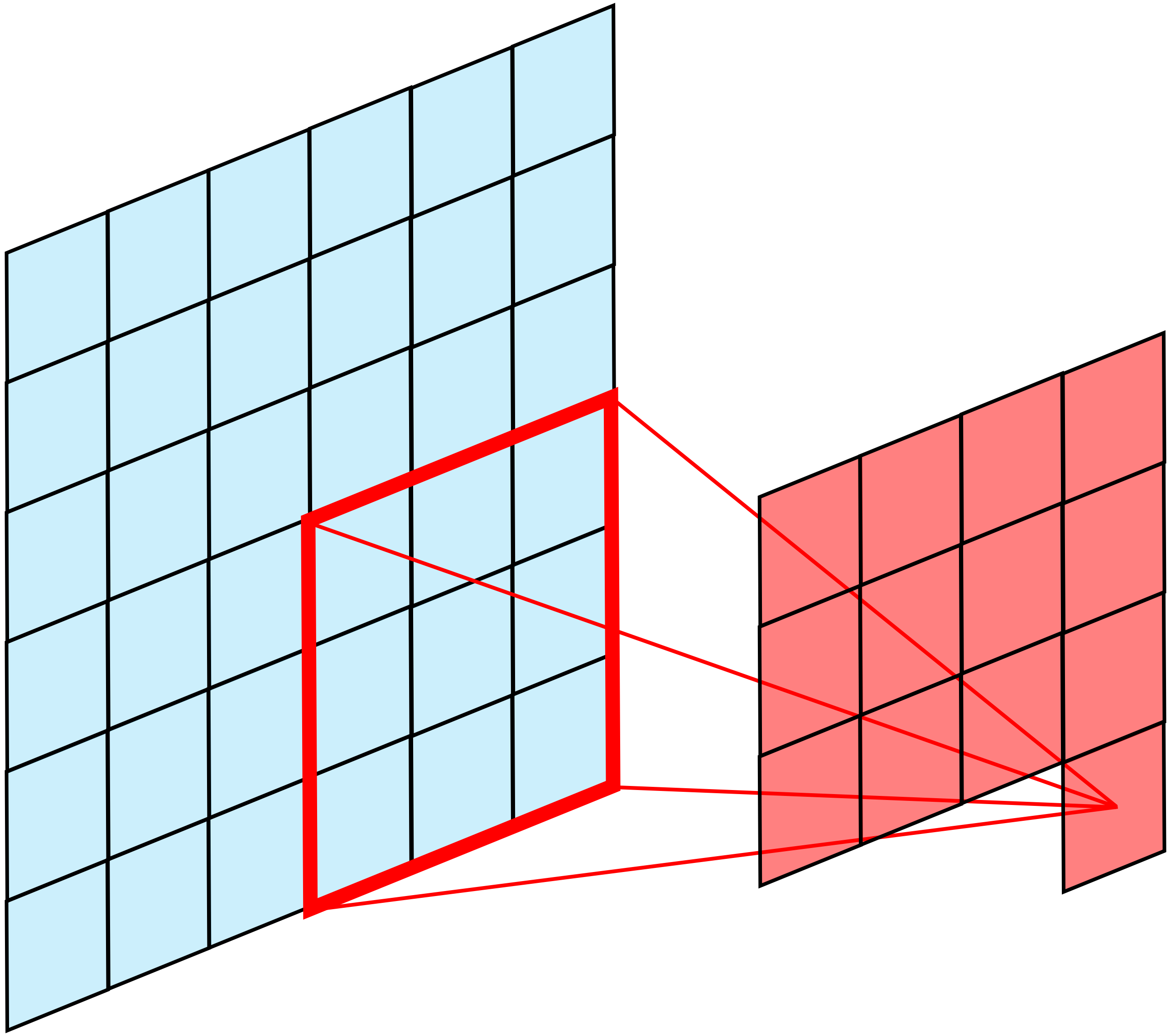
2D Convolution Operation



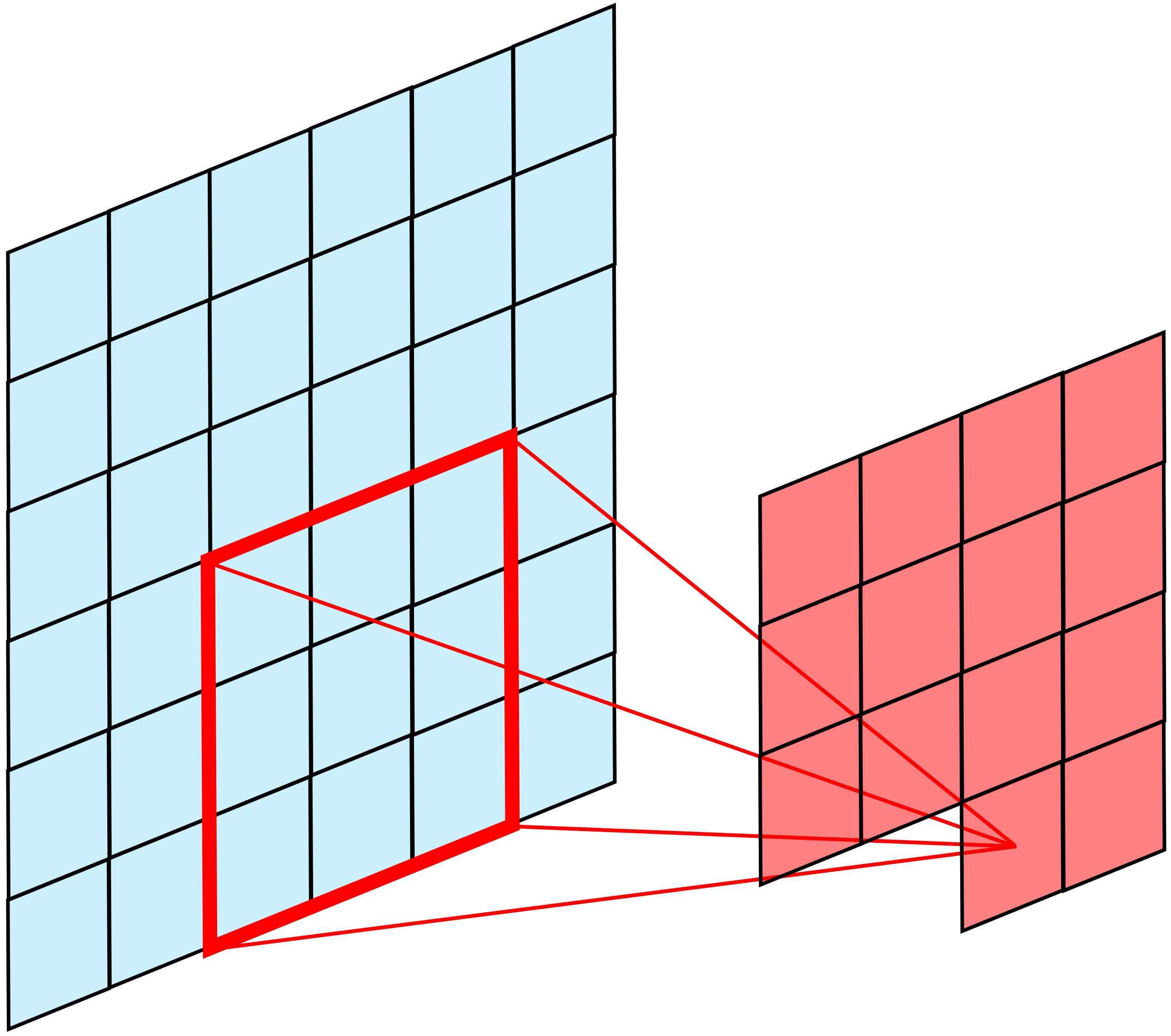
2D Convolution Operation



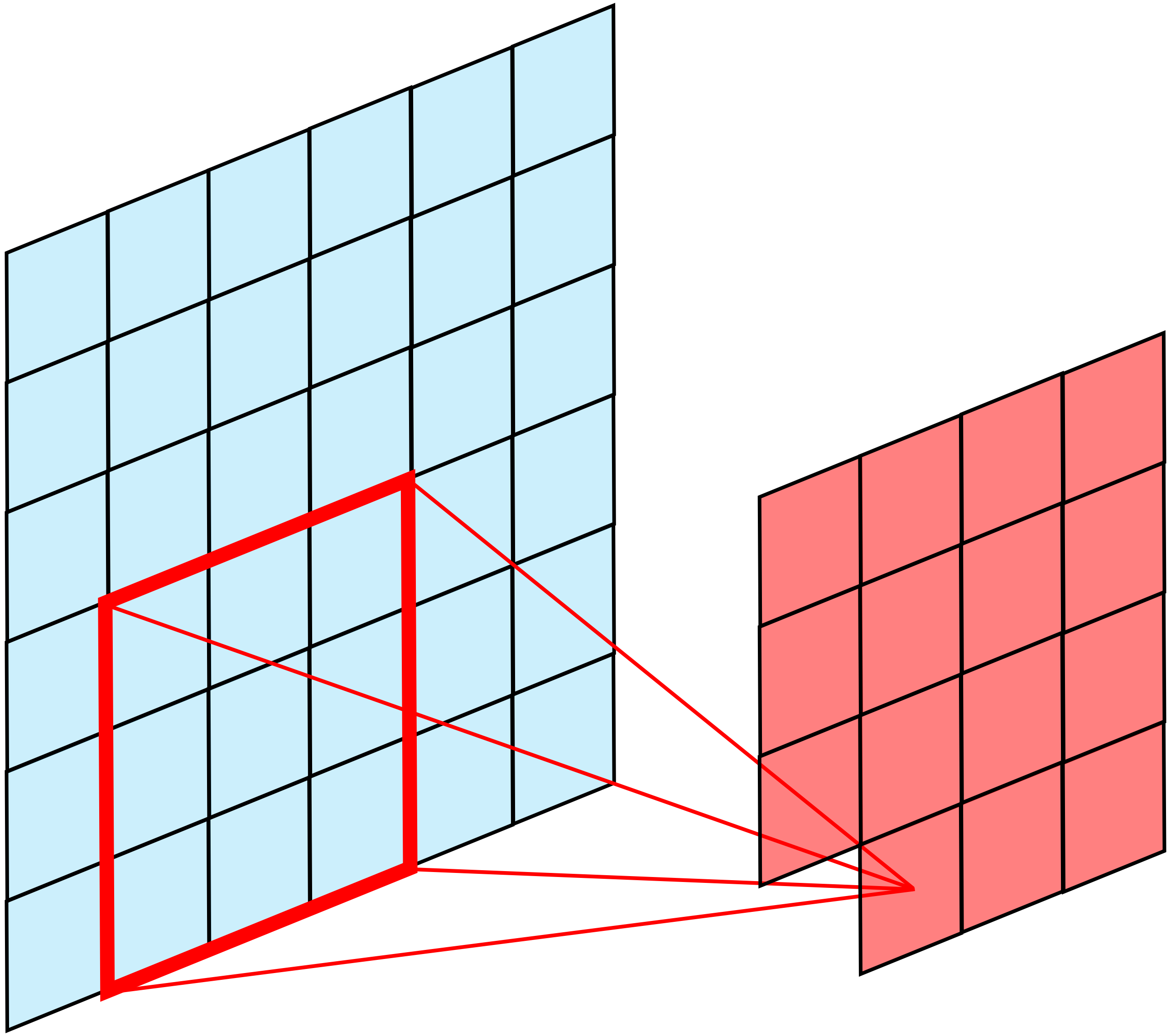
2D Convolution Operation



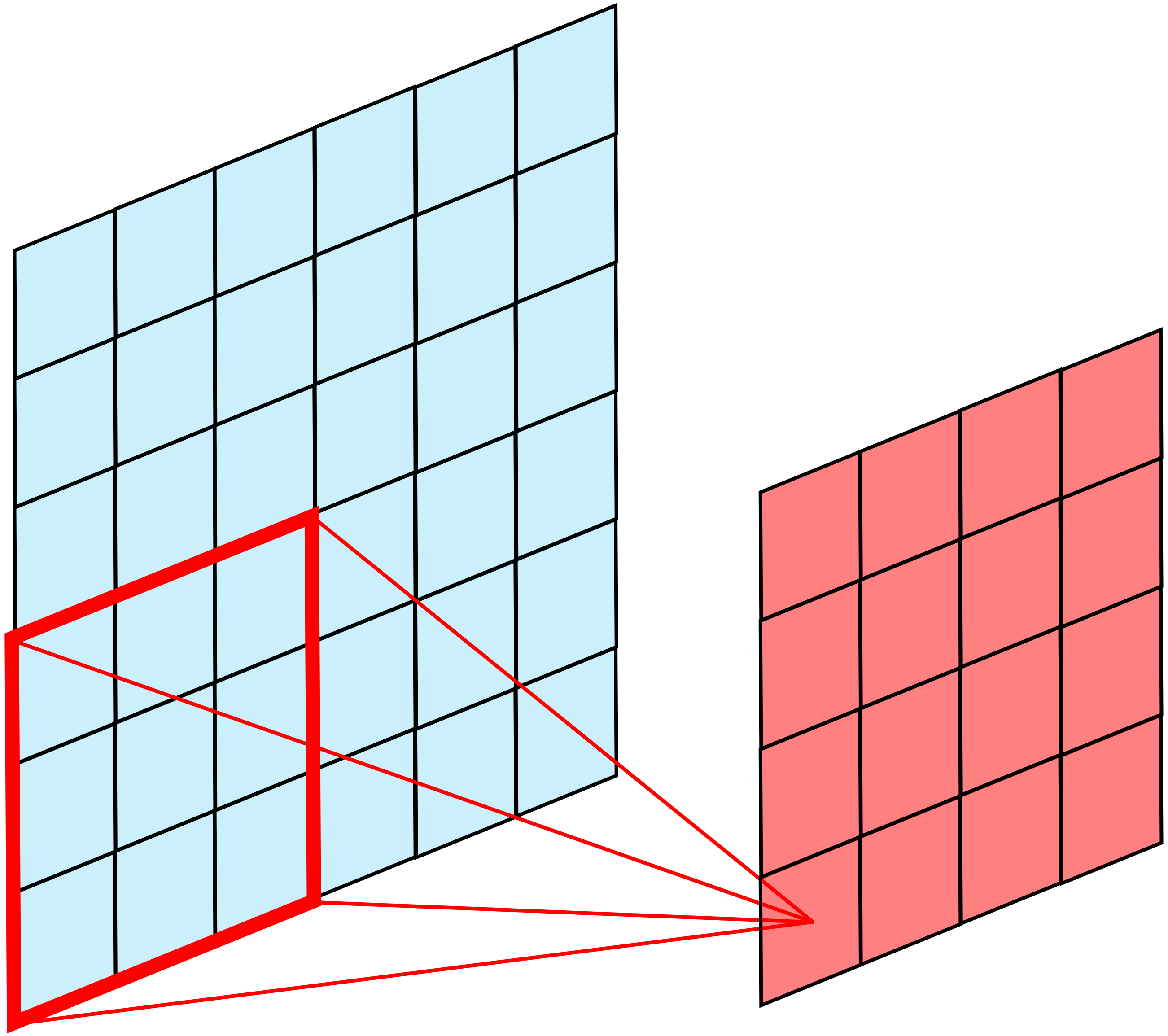
2D Convolution Operation



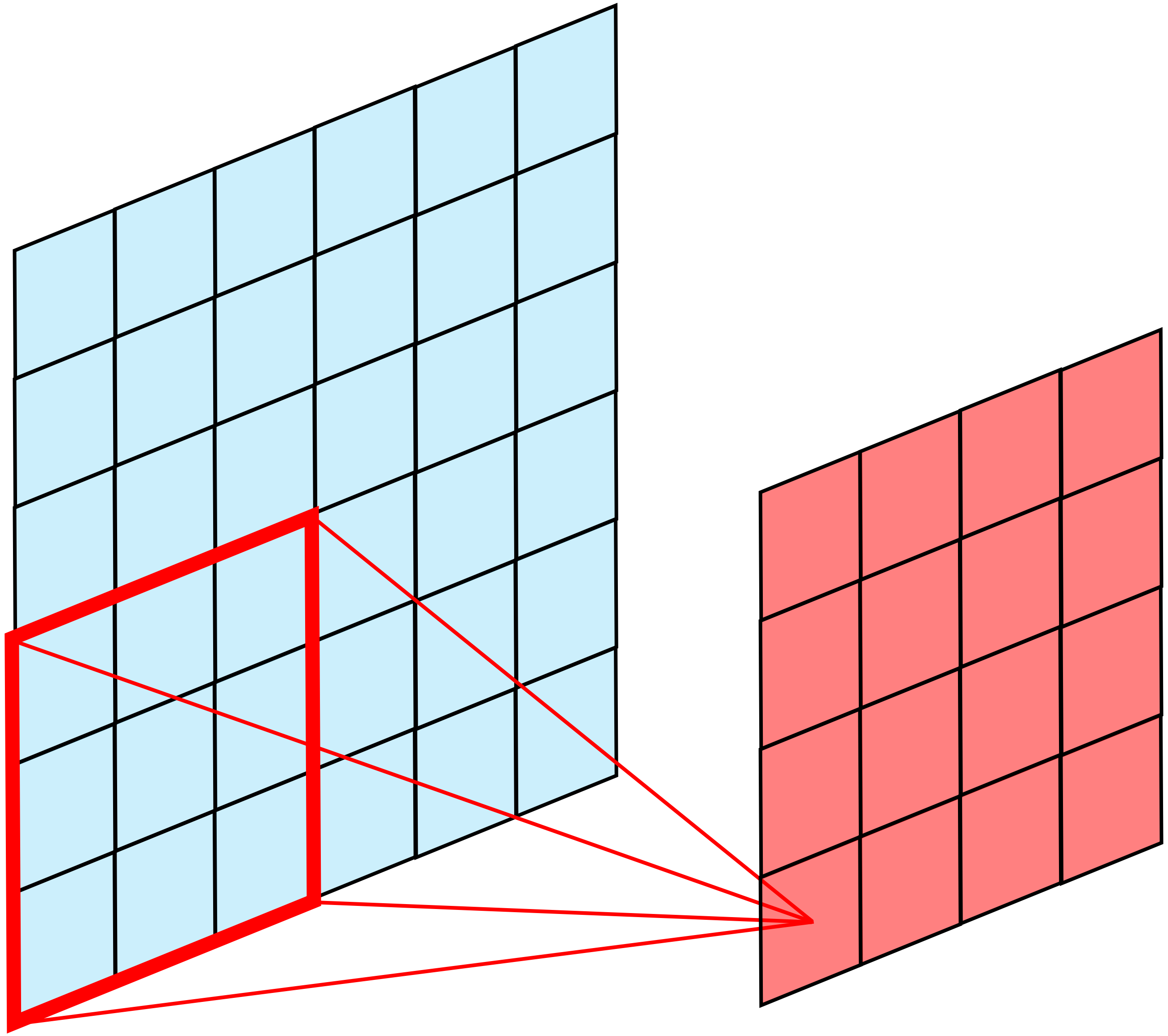
2D Convolution Operation



2D Convolution Operation

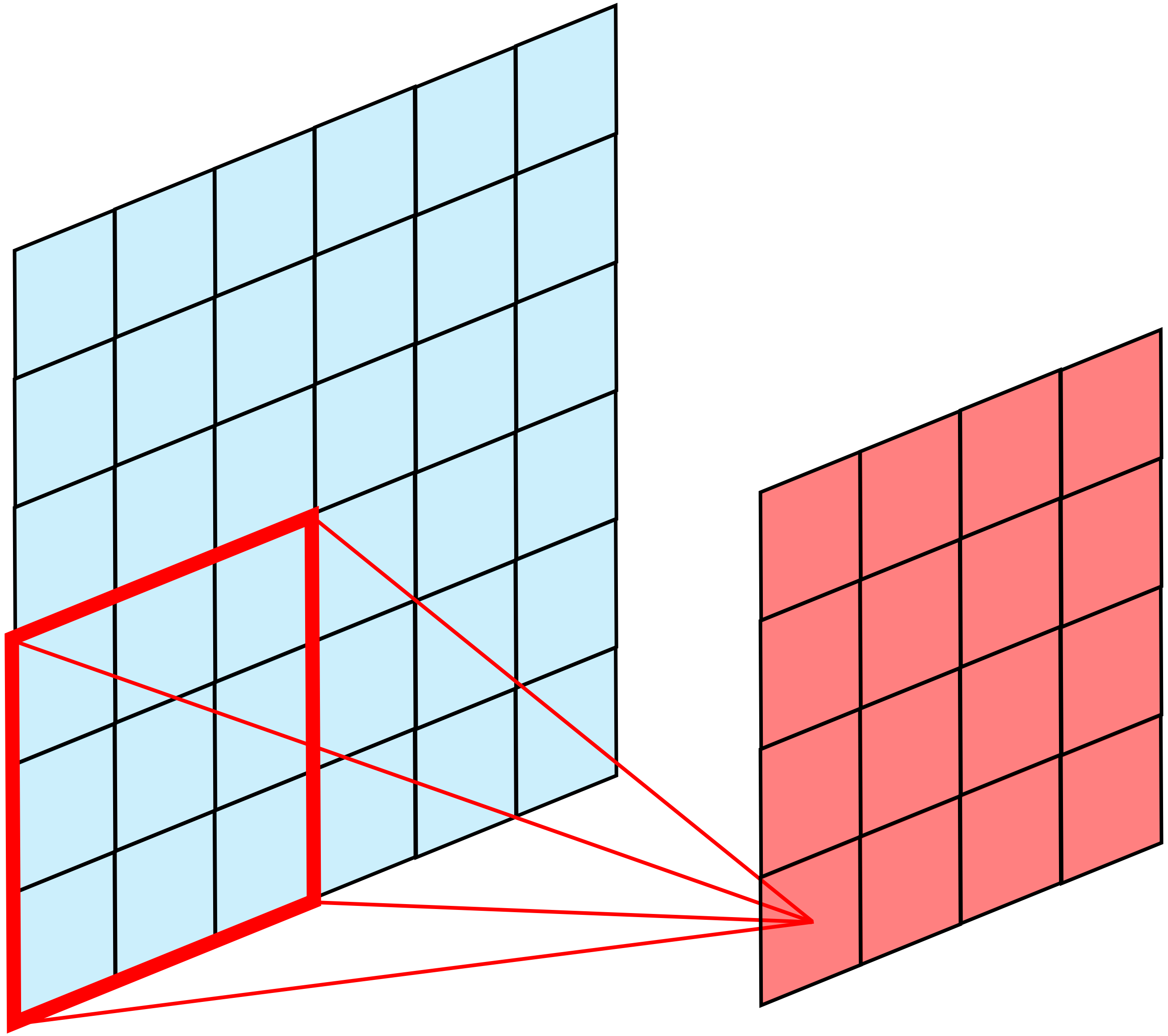


2D Convolution Operation



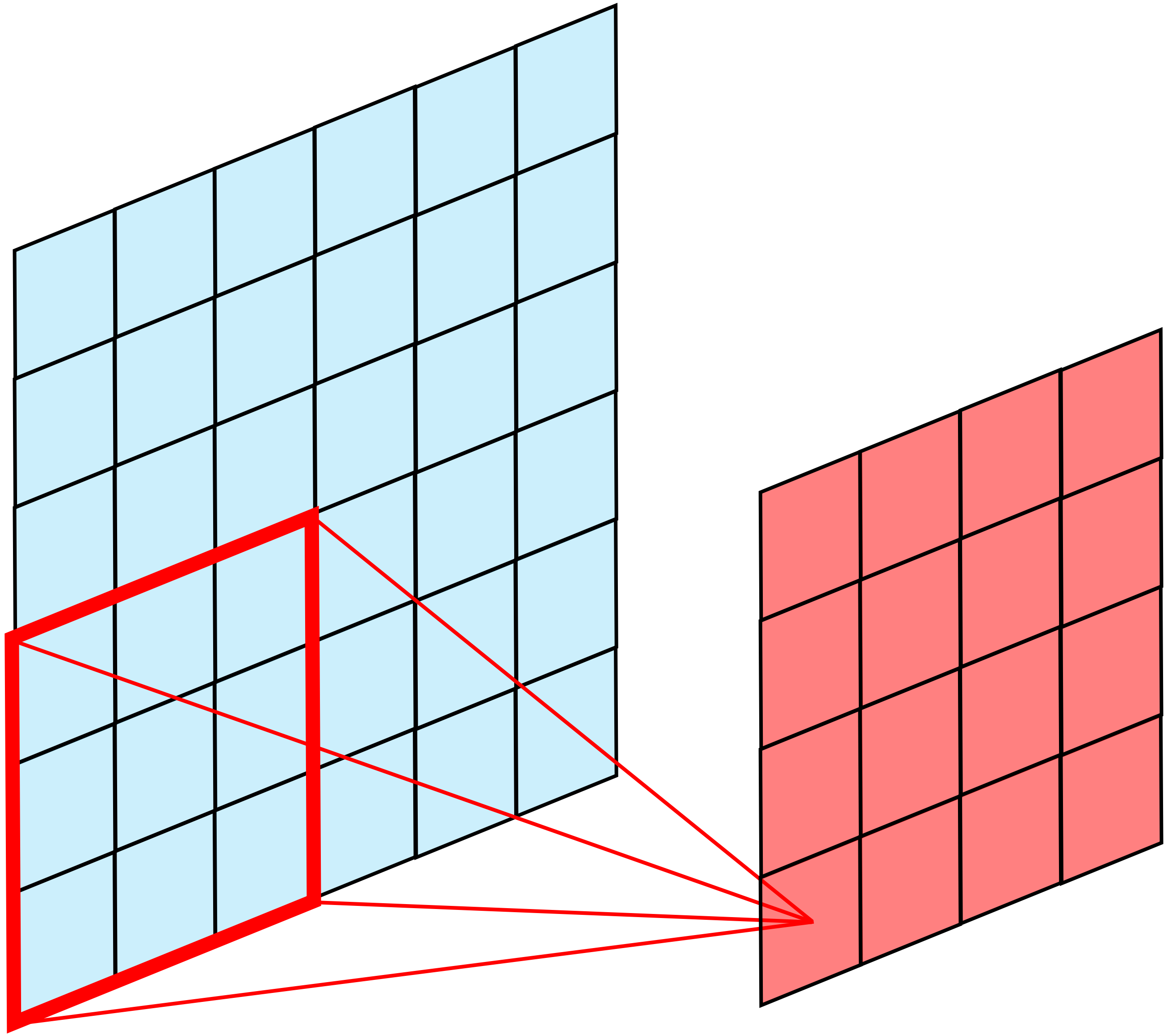
- Fully connected layer would need 576 weights

2D Convolution Operation



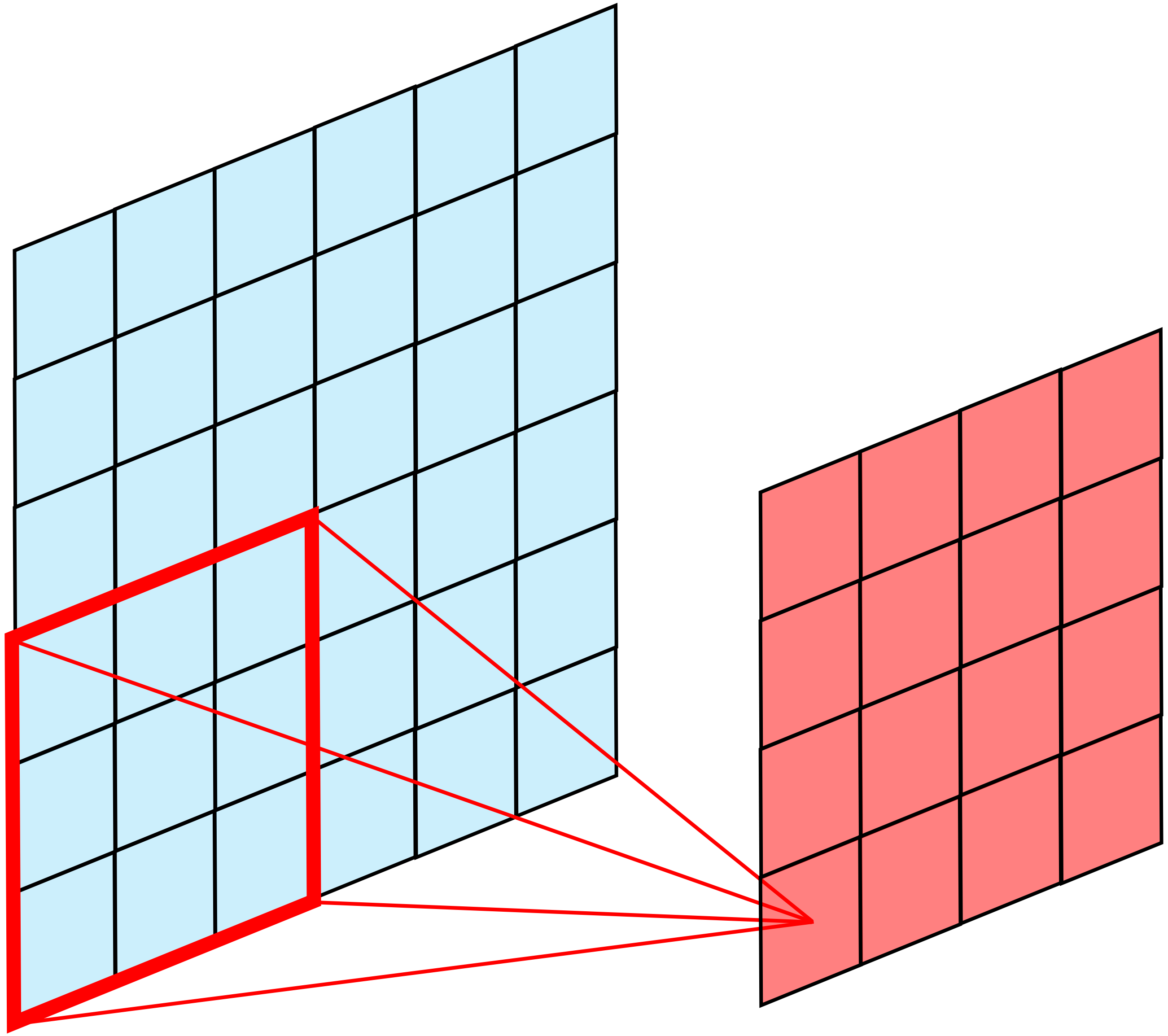
- Fully connected layer would need 576 weights
- Convolution needs only 9

2D Convolution Operation



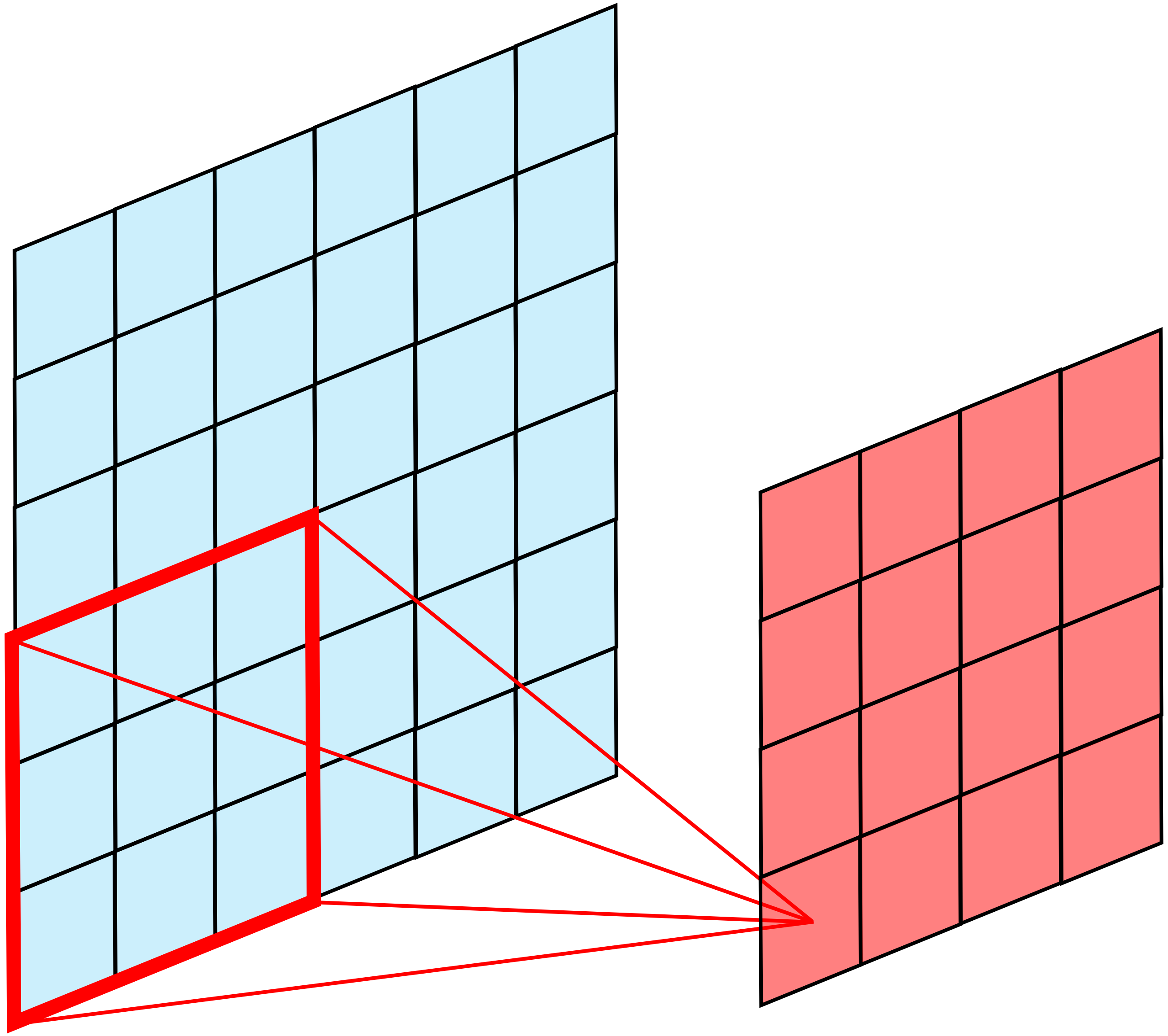
- Fully connected layer would need 576 weights
- Convolution needs only 9
- Such local operations are exactly what we need for detecting local patterns

2D Convolution Operation



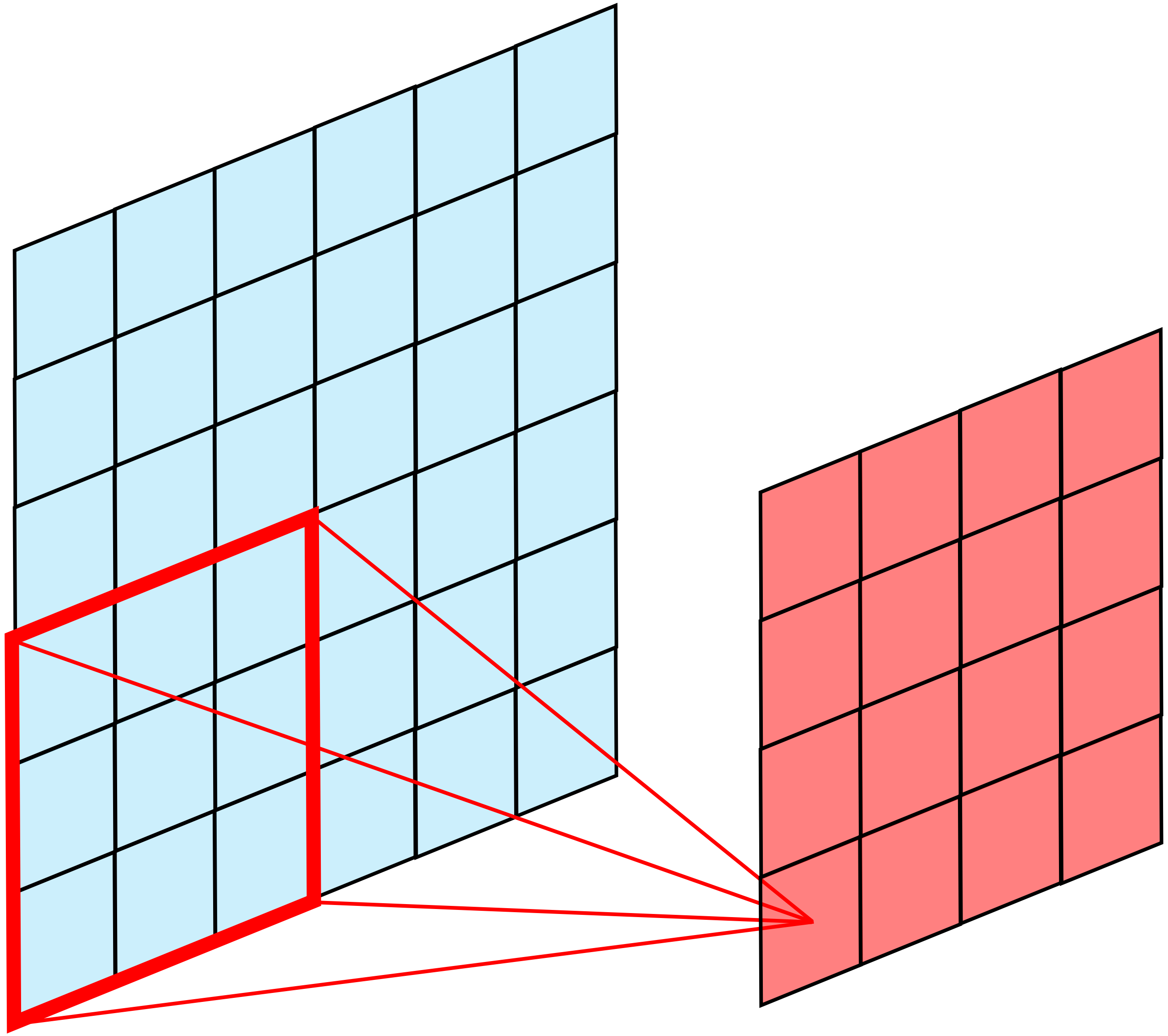
- Fully connected layer would need 576 weights
- Convolution needs only 9
- Such local operations are exactly what we need for detecting local patterns
- Edges, phrases etc

2D Convolution Operation



- Fully connected layer would need 576 weights
- Convolution needs only 9
- Such local operations are exactly what we need for detecting local patterns
- Edges, phrases etc
- Can apply convolutions to 3D layers as well

2D Convolution Operation



- Fully connected layer would need 576 weights
- Convolution needs only 9
- Such local operations are exactly what we need for detecting local patterns
- Edges, phrases etc
- Can apply convolutions to 3D layers as well
- E.g. Video data is 3D

Convolutional Neural Network

- Used widely in cases where the raw input has strong spatial structure e.g. images have 2D structure, text has linear structure
- Greatly reduces the number of parameters to be learnt
- Layers sparsely connected and aggressive parameter sharing
- Note: notion of "convolution" used in CNNs is non-standard
- Standard notion of convolution of two vectors $\mathbf{u}, \mathbf{v} \in \mathbb{R}^n$ is another vector $\mathbf{s} \in \mathbb{R}^n$ denoted as $\mathbf{s} = \mathbf{u} * \mathbf{v}$ such that

$$\mathbf{s}_i = \sum_{j=1}^{n} \mathbf{u}_j \cdot \mathbf{v}_{i-j}$$

- However, CNN uses $(\mathbf{u} * \mathbf{v})_i = \sum_{j=1}^{n} \mathbf{u}_{i+j} \cdot \mathbf{v}_j$ (cross-correlation)

Pooling Operations

Pooling Operations

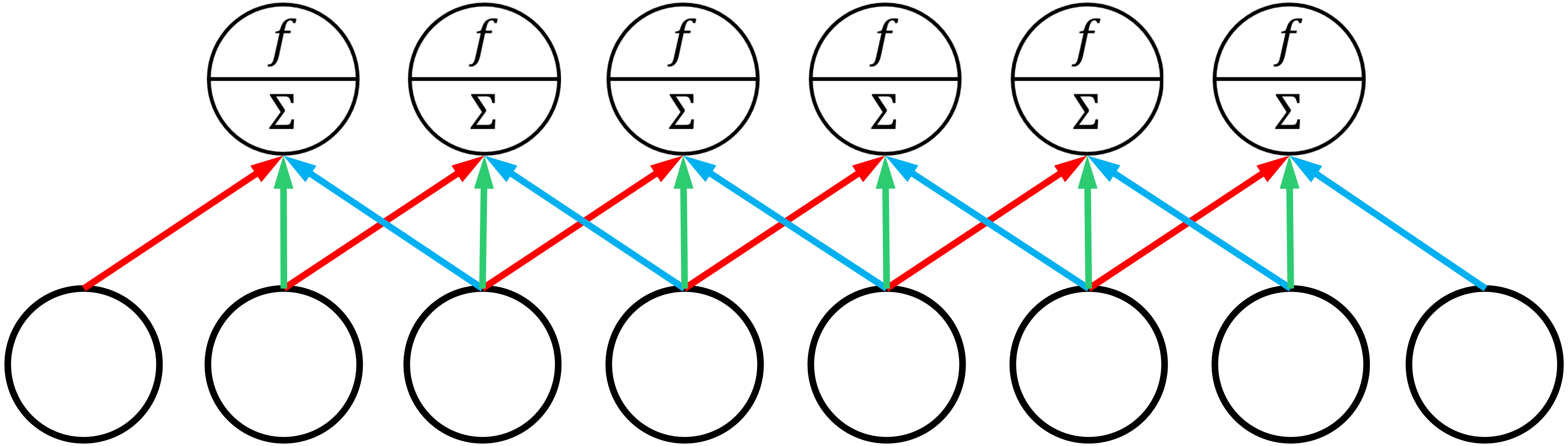
- Reduce sensitivity of the network to small shifts/errors in image

Pooling Operations

- Reduce sensitivity of the network to small shifts/errors in image
- Max-pooling and average pooling most common

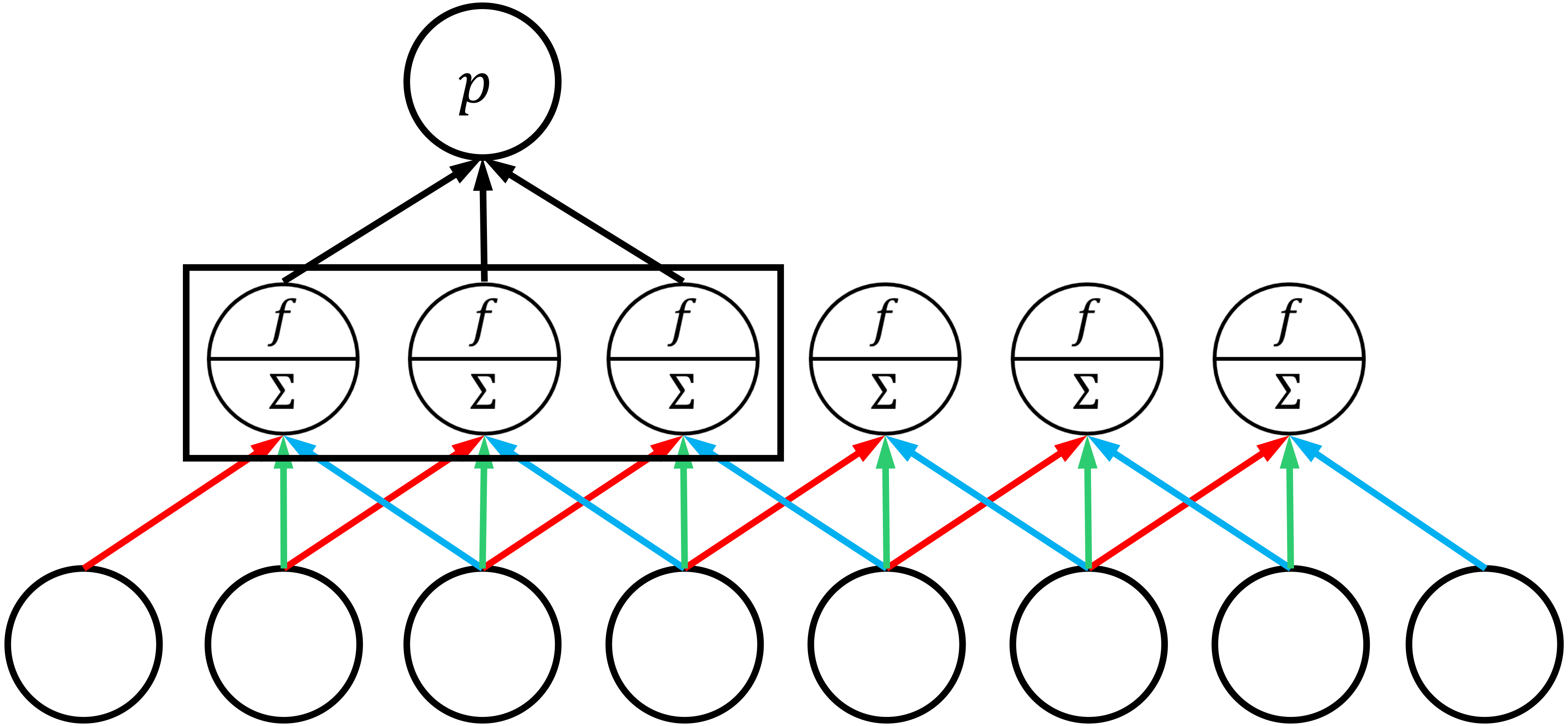
Pooling Operations

- Reduce sensitivity of the network to small shifts/errors in image
- Max-pooling and average pooling most common



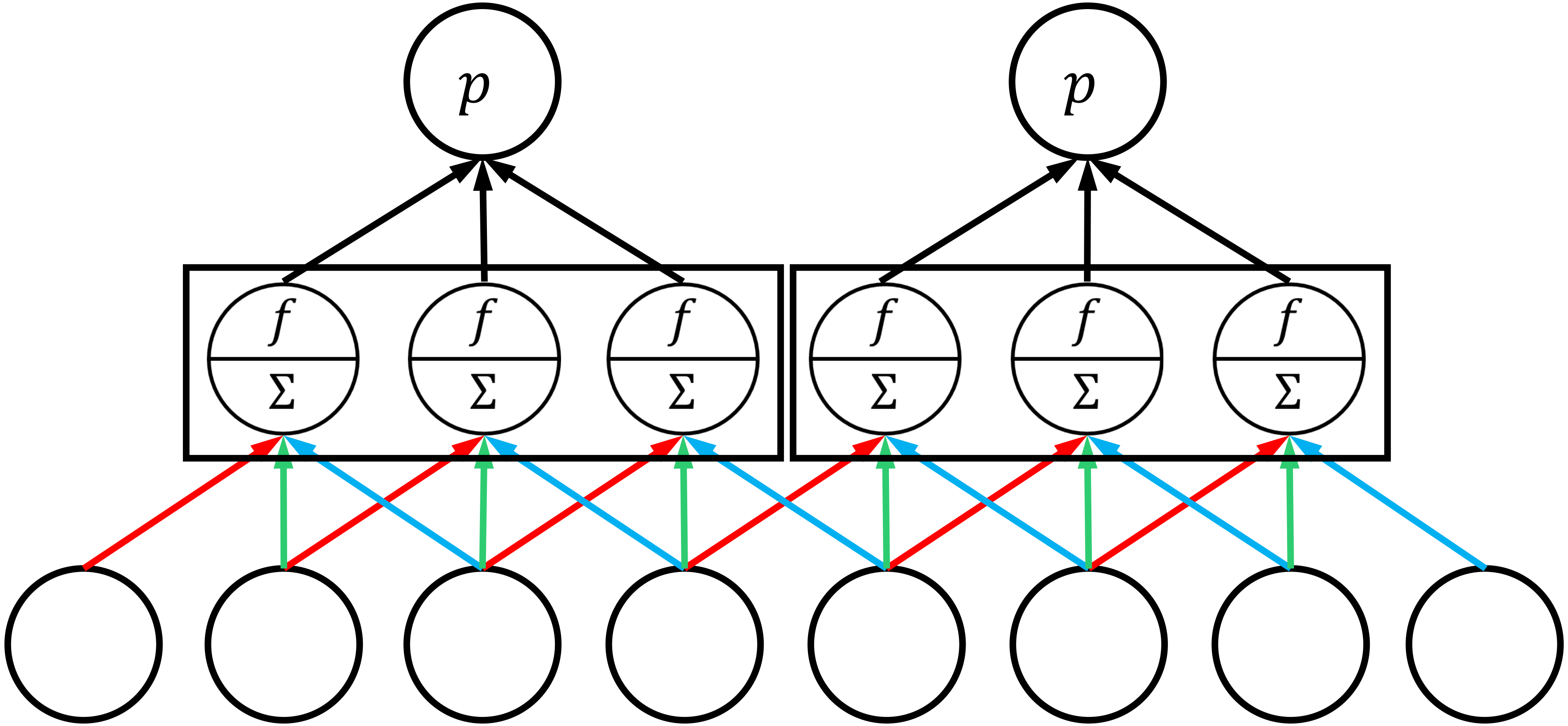
Pooling Operations

- Reduce sensitivity of the network to small shifts/errors in image
- Max-pooling and average pooling most common



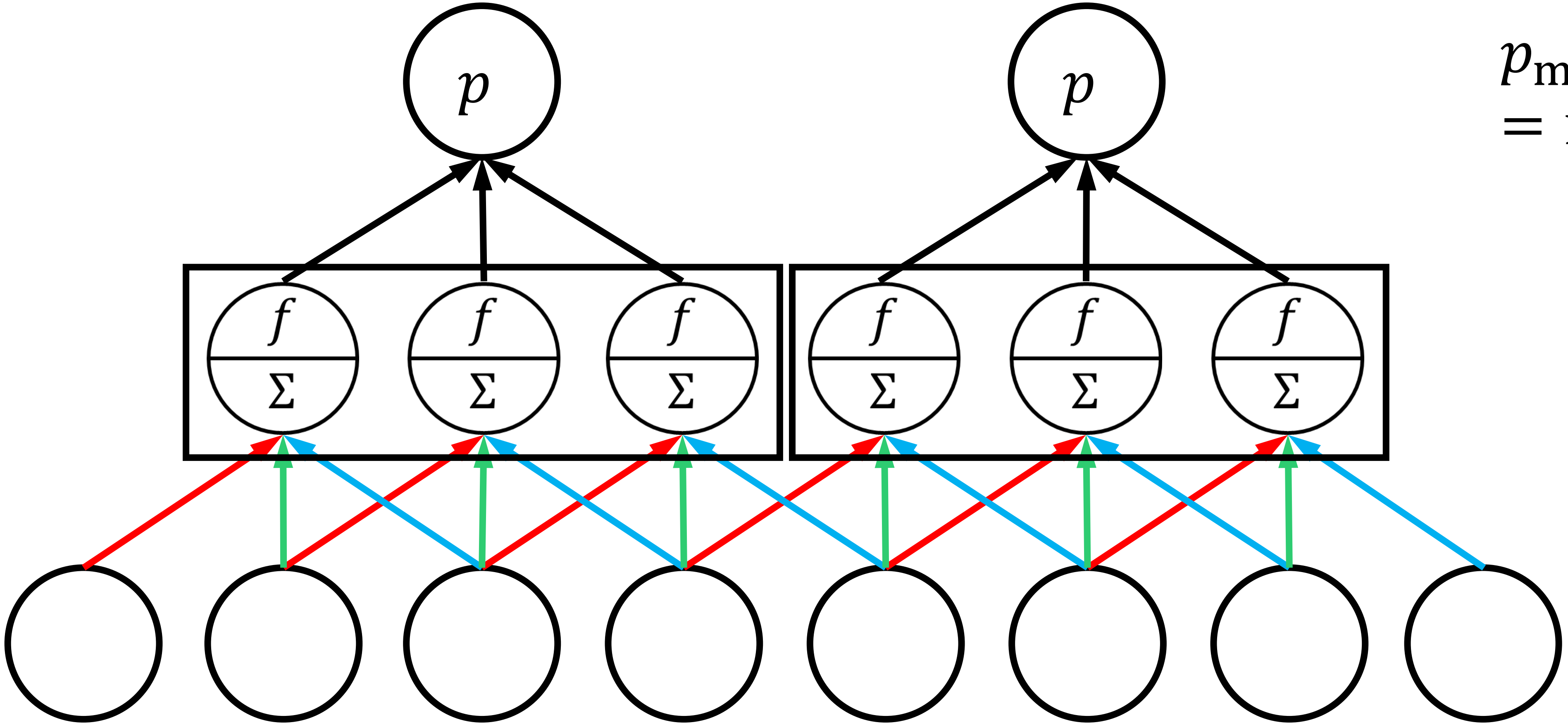
Pooling Operations

- Reduce sensitivity of the network to small shifts/errors in image
- Max-pooling and average pooling most common



Pooling Operations

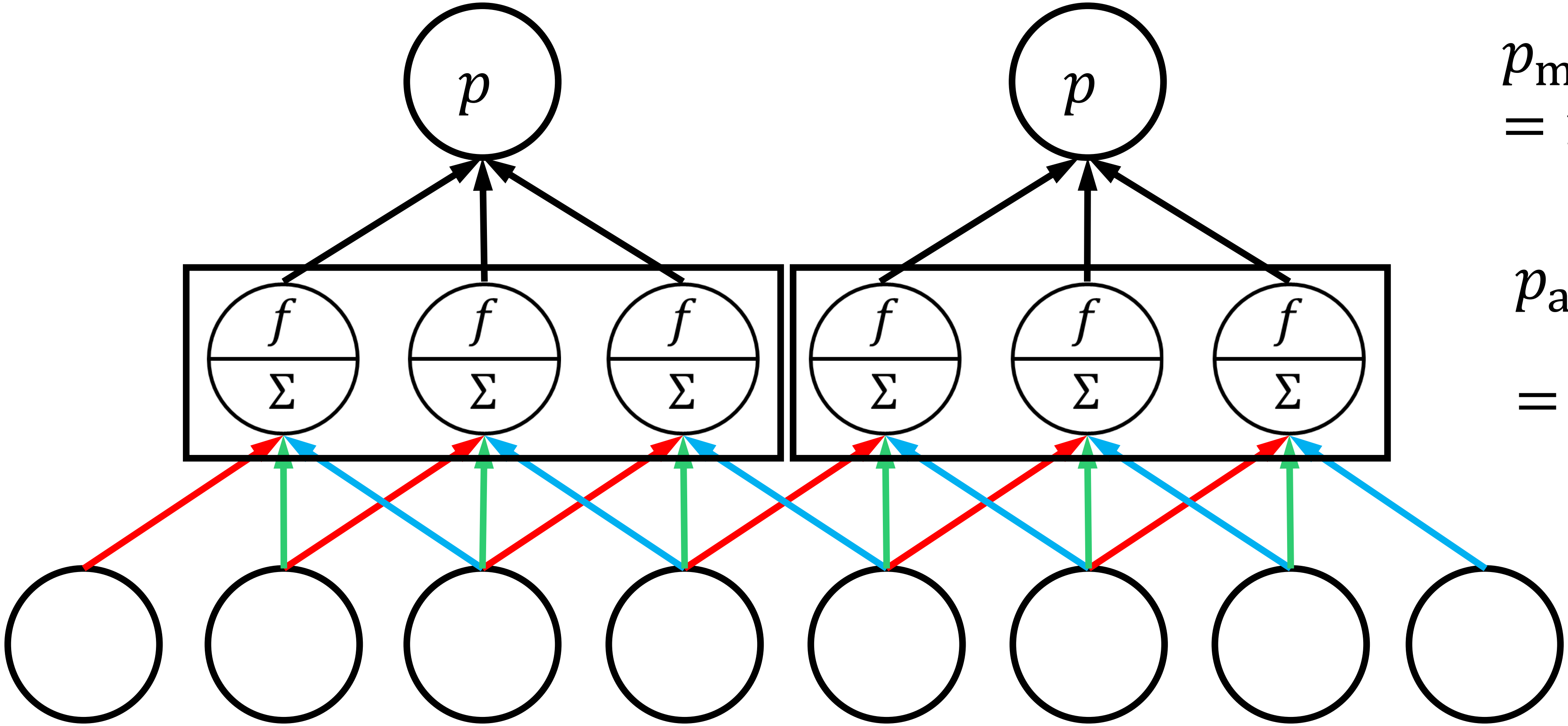
- Reduce sensitivity of the network to small shifts/errors in image
- Max-pooling and average pooling most common



$$p_{\max}(\mathbf{h}_1, \mathbf{h}_2, \mathbf{h}_3) \\ = \max\{\mathbf{h}_1, \mathbf{h}_2, \mathbf{h}_3\}$$

Pooling Operations

- Reduce sensitivity of the network to small shifts/errors in image
- Max-pooling and average pooling most common



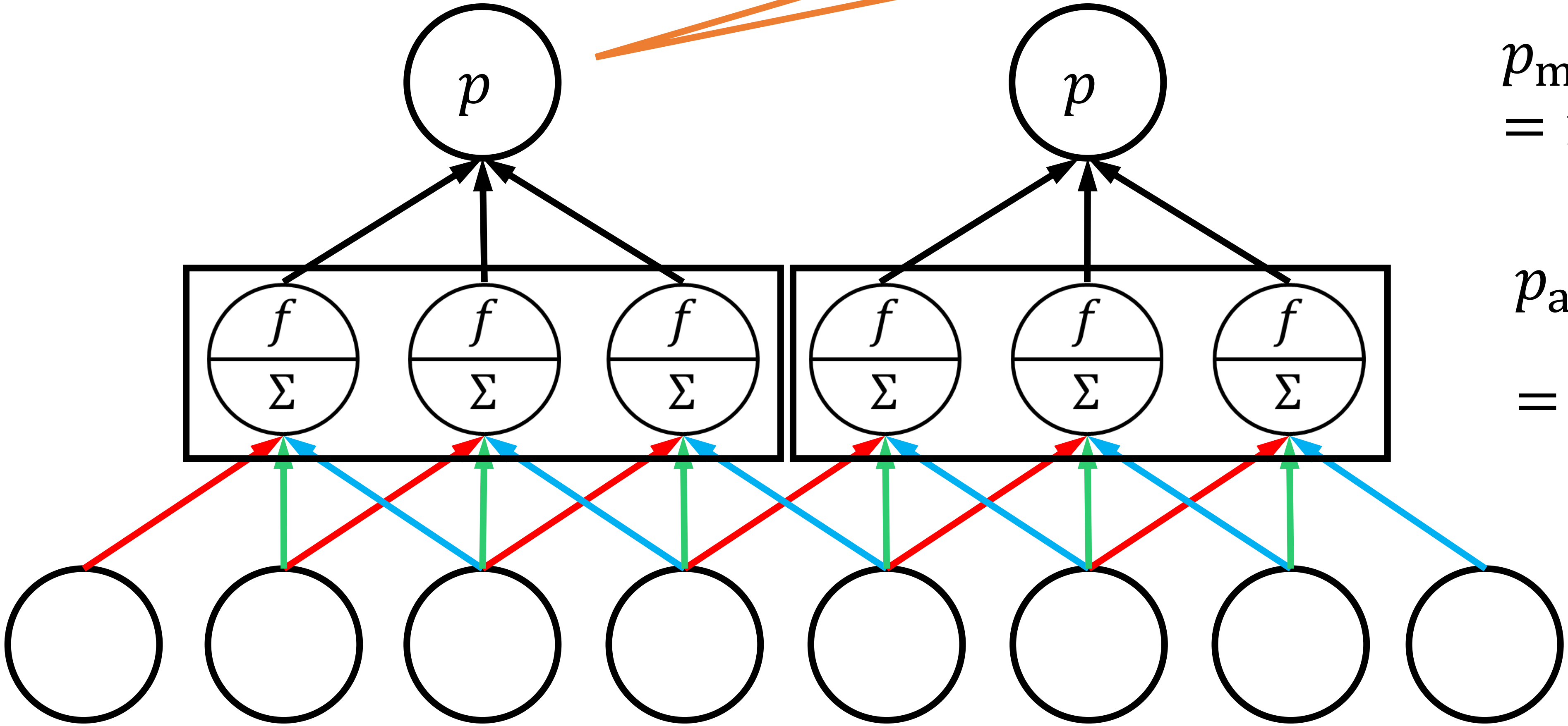
$$p_{\max}(\mathbf{h}_1, \mathbf{h}_2, \mathbf{h}_3) \\ = \max\{\mathbf{h}_1, \mathbf{h}_2, \mathbf{h}_3\}$$

$$p_{\text{avg}}(\mathbf{h}_1, \mathbf{h}_2, \mathbf{h}_3) \\ = \frac{1}{3}(\mathbf{h}_1 + \mathbf{h}_2 + \mathbf{h}_3)$$

Pooling Operations

- Reduce sensitivity of the net image
- Max-pooling and average pooling most common

“Stride” length – number of nodes after which a new “pool” is started. Stride =3 here



$$p_{\max}(\mathbf{h}_1, \mathbf{h}_2, \mathbf{h}_3)$$
$$= \max\{\mathbf{h}_1, \mathbf{h}_2, \mathbf{h}_3\}$$

$$p_{\text{avg}}(\mathbf{h}_1, \mathbf{h}_2, \mathbf{h}_3)$$
$$= \frac{1}{3}(\mathbf{h}_1 + \mathbf{h}_2 + \mathbf{h}_3)$$

Simple Operations using Conv and Pool

Simple Operations using Conv and Pool

Raw Image

Kernels

Convolved Image

Max Pooling (stride 1x2)

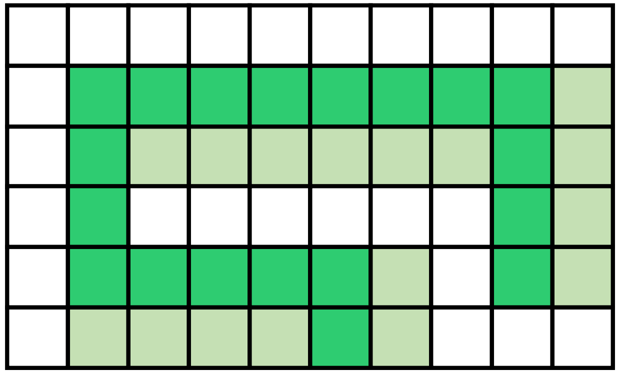
Simple Operations using Conv and Pool

Raw Image

Kernels

Convolved Image

Max Pooling (stride 1x2)



Simple Operations using Conv and Pool

Raw Image

Kernels

Convolved Image

Max Pooling (stride 1x2)

□	□	□	□	□	□	□	□	□	□
□	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	▨
□	🟩	▨	▨	▨	▨	▨	▨	🟩	▨
□	🟩	□	□	□	□	□	□	🟩	▨
□	🟩	🟩	🟩	🟩	🟩	▨	□	🟩	▨
□	▨	▨	▨	▨	🟩	▨	□	□	□

🟩 $= +1$

▨ $= +0.5$

□ $= 0$

🟦 $= 0$ (padded)

🟧 $= -0.5$

🟥 $= -1$

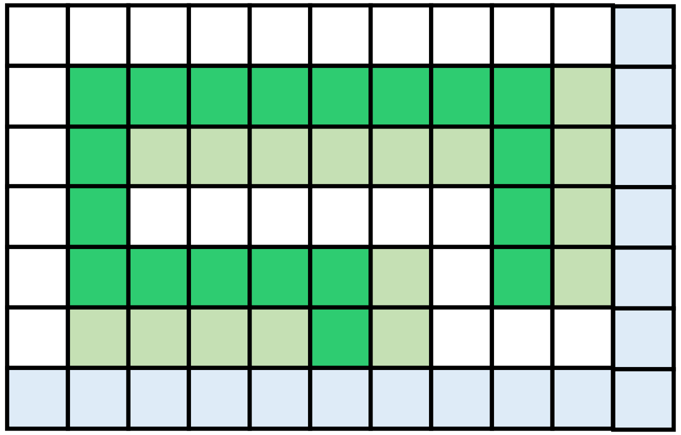
Simple Operations using Conv and Pool

Raw Image

Kernels

Convolved Image

Max Pooling (stride 1x2)



🟩 $= +1$

▨ $= +0.5$

□ $= 0$

🟦 $= 0$ (padded)

🟧 $= -0.5$

🟥 $= -1$

Image padded with zero pixels so that convolved image is of same size

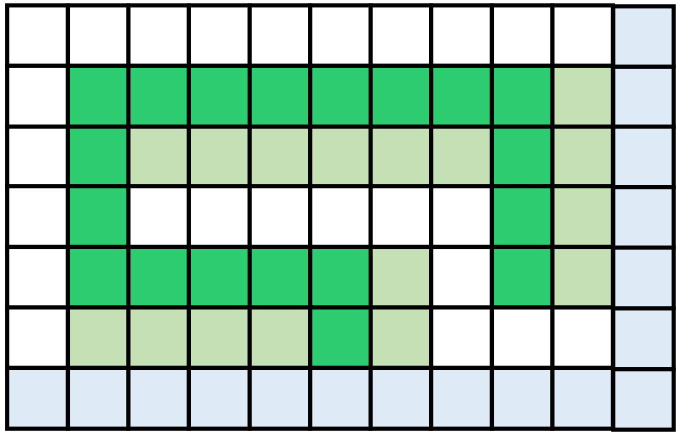
Simple Operations using Conv and Pool

Raw Image

Kernels

Convolved Image

Max Pooling (stride 1x2)



0
-1

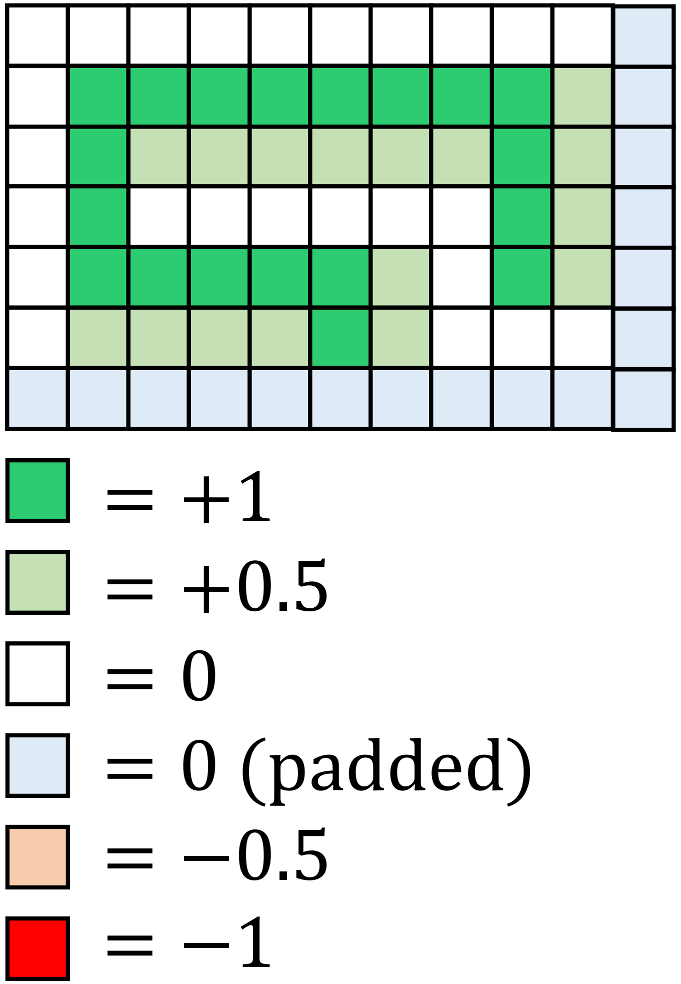
Detects horizontal edges!

- 🟩 $= +1$
- ▨ $= +0.5$
- □ $= 0$
- 🟦 $= 0$ (padded)
- 🟧 $= -0.5$
- 🟥 $= -1$

Image padded with zero pixels so that convolved image is of same size

Simple Operations using Conv and Pool

Raw Image

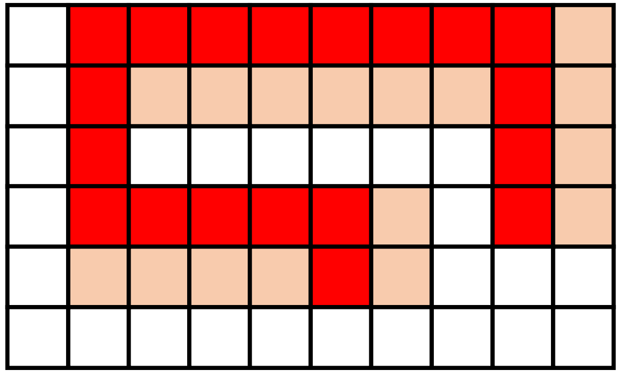


Kernels

$$\begin{bmatrix} 0 \\ -1 \end{bmatrix}$$

Detects horizontal edges!

Convolved Image



Max Pooling (stride 1x2)

Image padded with zero pixels so that convolved image is of same size

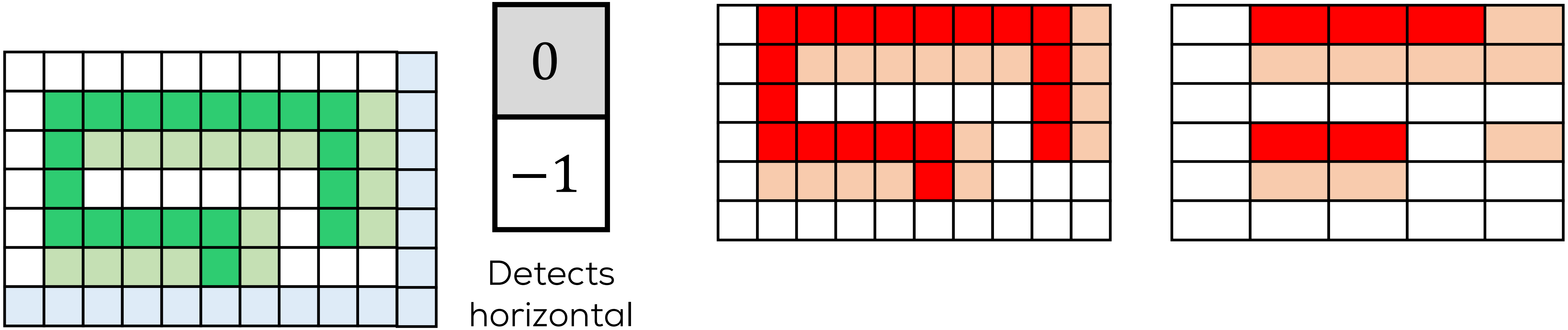
Simple Operations using Conv and Pool

Raw Image

Kernels

Convolved Image

Max Pooling (stride 1x2)



Detects horizontal edges!

🟩 $= +1$

▨ $= +0.5$

□ $= 0$

▢ $= 0$ (padded)

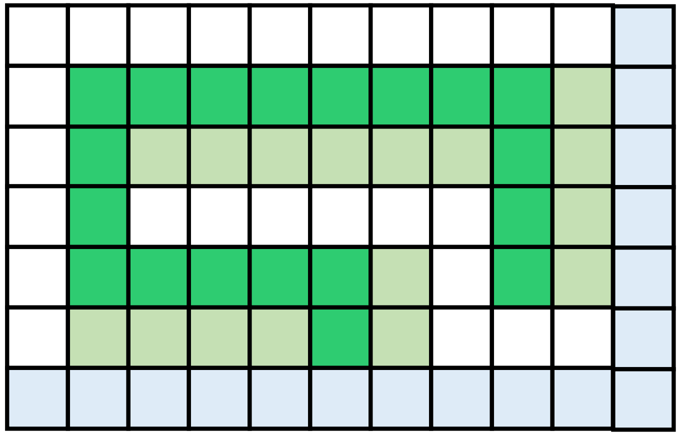
🟧 $= -0.5$

🟥 $= -1$

Image padded with zero pixels so that convolved image is of same size

Simple Operations using Conv and Pool

Raw Image



- 🟩 $= +1$
- 🟩 $= +0.5$
- □ $= 0$
- □ $= 0$ (padded)
- 🟧 $= -0.5$
- 🟥 $= -1$

Kernels

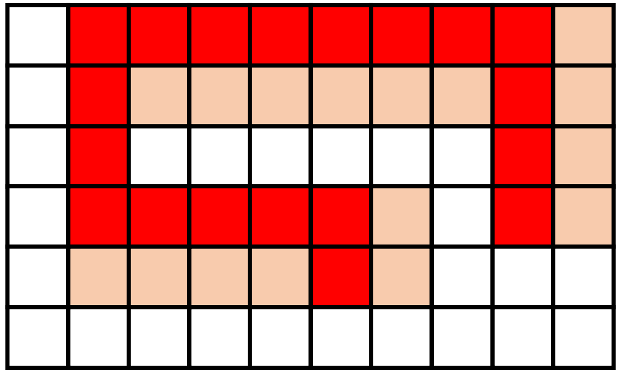
$$\begin{bmatrix} 0 \\ -1 \end{bmatrix}$$

Detects horizontal edges!

$$\begin{bmatrix} 0 & -1 \end{bmatrix}$$

Detects vertical edges!

Convolved Image



Max Pooling (stride 1x2)

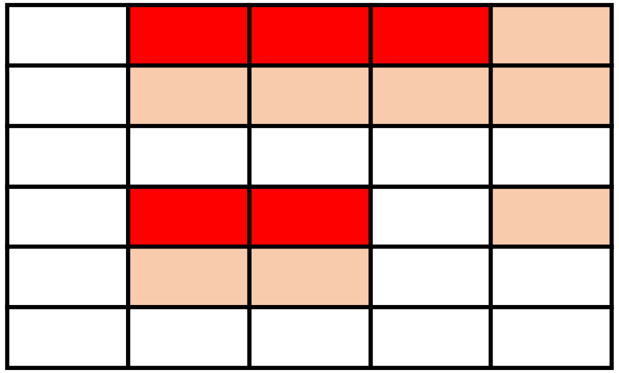


Image padded with zero pixels so that convolved image is of same size

Simple Operations using Conv and Pool

Raw Image

□	□	□	□	□	□	□	□	□	□	🟦
□	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	▨	🟦
□	🟩	▨	▨	▨	▨	▨	▨	🟩	▨	🟦
□	🟩	□	□	□	□	□	□	🟩	▨	🟦
□	🟩	🟩	🟩	🟩	🟩	▨	□	🟩	▨	🟦
□	▨	▨	▨	▨	🟩	▨	□	□	□	🟦
🟦	🟦	🟦	🟦	🟦	🟦	🟦	🟦	🟦	🟦	🟦

🟩 $= +1$

▨ $= +0.5$

□ $= 0$

🟦 $= 0$ (padded)

🟧 $= -0.5$

🟥 $= -1$

Kernels

0
-1

Detects horizontal edges!

0	-1

Detects vertical edges!

Convolved Image

□	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟧
□	🟥	🟧	🟧	🟧	🟧	🟧	🟧	🟥	🟧
□	🟥	□	□	□	□	□	□	🟥	🟧
□	🟥	🟥	🟥	🟥	🟥	🟧	□	🟥	🟧
□	🟧	🟧	🟧	🟧	🟥	🟧	□	□	□
□	□	□	□	□	□	□	□	□	□

□	□	□	□	□	□	□	□	□	□
🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟧	□
🟥	🟧	🟧	🟧	🟧	🟧	🟧	🟥	🟧	□
🟥	□	□	□	□	□	□	🟥	🟧	□
🟥	🟥	🟥	🟥	🟥	🟧	□	🟥	🟧	□
🟧	🟧	🟧	🟧	🟥	🟧	□	□	□	□

Max Pooling (stride 1x2)

□	🟥	🟥	🟥	🟧
□	🟧	🟧	🟧	🟧
□	□	□	□	□
□	🟥	🟥	□	🟧
□	🟧	🟧	□	□
□	□	□	□	□

Image padded with zero pixels so that convolved image is of same size

Simple Operations using Conv and Pool

Raw Image

⬜	⬜	⬜	⬜	⬜	⬜	⬜	⬜	⬜	⬜	🟦
⬜	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	▩	🟦
⬜	🟩	▩	▩	▩	▩	▩	▩	🟩	▩	🟦
⬜	🟩	⬜	⬜	⬜	⬜	⬜	⬜	🟩	▩	🟦
⬜	🟩	🟩	🟩	🟩	🟩	▩	⬜	🟩	▩	🟦
⬜	▩	▩	▩	▩	🟩	▩	⬜	⬜	⬜	🟦
🟦	🟦	🟦	🟦	🟦	🟦	🟦	🟦	🟦	🟦	🟦

🟩 $= +1$

▩ $= +0.5$

⬜ $= 0$

🟦 $= 0$ (padded)

🟧 $= -0.5$

🟥 $= -1$

Kernels

0
-1

Detects horizontal edges!

0	-1

Detects vertical edges!

Convolved Image

⬜	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟧
⬜	🟥	🟧	🟧	🟧	🟧	🟧	🟧	🟥	🟧
⬜	🟥	⬜	⬜	⬜	⬜	⬜	⬜	🟥	🟧
⬜	🟥	🟥	🟥	🟥	🟥	🟧	⬜	🟥	🟧
⬜	🟧	🟧	🟧	🟧	🟥	🟧	⬜	⬜	⬜
⬜	⬜	⬜	⬜	⬜	⬜	⬜	⬜	⬜	⬜

⬜	⬜	⬜	⬜	⬜	⬜	⬜	⬜	⬜	⬜
🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟧	⬜
🟥	🟧	🟧	🟧	🟧	🟧	🟧	🟥	🟧	⬜
🟥	⬜	⬜	⬜	⬜	⬜	⬜	🟥	🟧	⬜
🟥	🟥	🟥	🟥	🟥	🟧	⬜	🟥	🟧	⬜
🟧	🟧	🟧	🟧	🟥	🟧	⬜	⬜	⬜	⬜

Max Pooling (stride 1x2)

⬜	🟥	🟥	🟥	🟧
⬜	🟧	🟧	🟧	🟧
⬜	⬜	⬜	⬜	⬜
⬜	🟥	🟥	⬜	🟧
⬜	🟧	🟧	⬜	⬜
⬜	⬜	⬜	⬜	⬜

⬜	⬜	⬜	⬜	⬜	⬜	⬜	⬜	⬜	⬜
🟥	⬜	⬜	⬜	⬜	⬜	⬜	🟥	🟧	⬜
🟧	🟧	🟧	🟧	🟥	🟧	⬜	⬜	⬜	⬜

Max Pooling (stride 2x1)

Image padded with zero pixels so that convolved image is of same size

Simple Operations using Conv and Pool

Verify that 2x2 stride leads to too much info loss

s

□	□	□	□	□	□	□	□	□	□	🟦
□	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	▧	🟦
□	🟩	▧	▧	▧	▧	▧	▧	🟩	▧	🟦
□	🟩	□	□	□	□	□	□	🟩	▧	🟦
□	🟩	🟩	🟩	🟩	🟩	▧	□	🟩	▧	🟦
□	▧	▧	▧	▧	🟩	▧	□	□	□	🟦
🟦	🟦	🟦	🟦	🟦	🟦	🟦	🟦	🟦	🟦	🟦

🟩 $= +1$

▧ $= +0.5$

□ $= 0$

🟦 $= 0$ (padded)

🟧 $= -0.5$

🟥 $= -1$

$$\begin{bmatrix} 0 \\ -1 \end{bmatrix}$$

Detects horizontal edges!

Convolved Image

□	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟧
□	🟥	🟧	🟧	🟧	🟧	🟧	🟧	🟥	🟧
□	🟥	□	□	□	□	□	□	🟥	🟧
□	🟥	🟥	🟥	🟥	🟥	🟧	□	🟥	🟧
□	🟧	🟧	🟧	🟧	🟥	🟧	□	□	□
□	□	□	□	□	□	□	□	□	□

Max Pooling (stride 1x2)

□	🟥	🟥	🟥	🟧
□	🟧	🟧	🟧	🟧
□	□	□	□	□
□	🟥	🟥	□	🟧
□	🟧	🟧	□	□
□	□	□	□	□

$$\begin{bmatrix} 0 & -1 \end{bmatrix}$$

Detects vertical edges!

□	□	□	□	□	□	□	□	□	□
🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟧	□
🟥	🟧	🟧	🟧	🟧	🟧	🟧	🟥	🟧	□
🟥	□	□	□	□	□	□	🟥	🟧	□
🟥	🟥	🟥	🟥	🟥	🟧	□	🟥	🟧	□
🟧	🟧	🟧	🟧	🟥	🟧	□	□	□	□

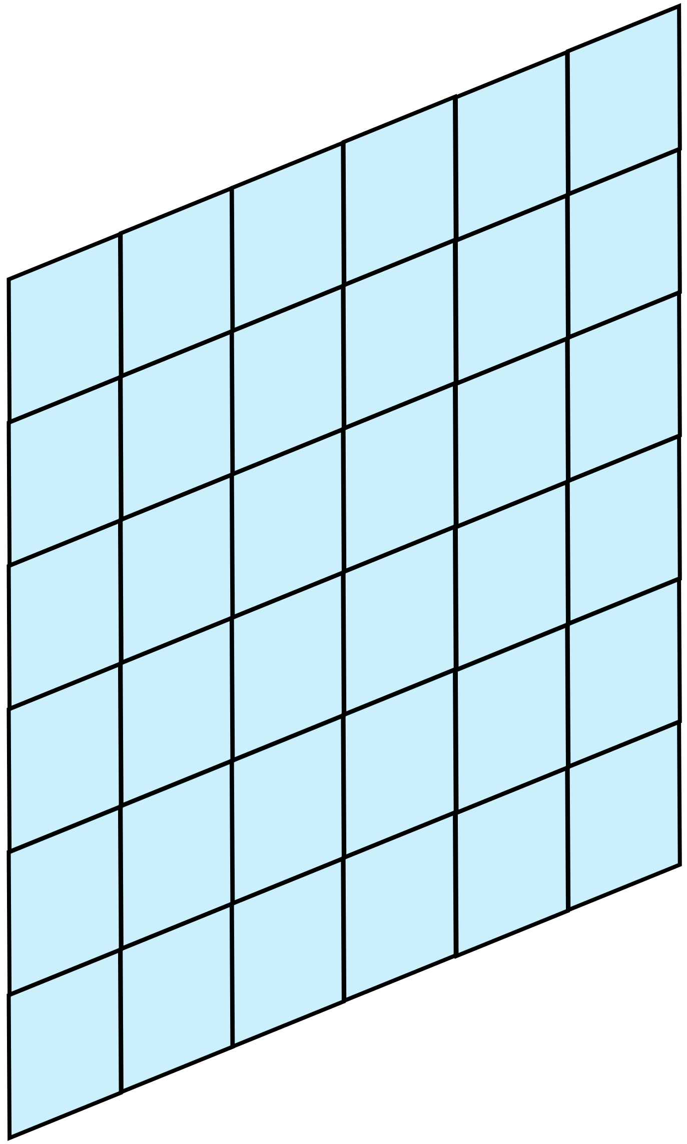
□	□	□	□	□	□	□	□	□	□
🟥	□	□	□	□	□	□	🟥	🟧	□
🟧	🟧	🟧	🟧	🟥	🟧	□	□	□	□

Max Pooling (stride 2x1)

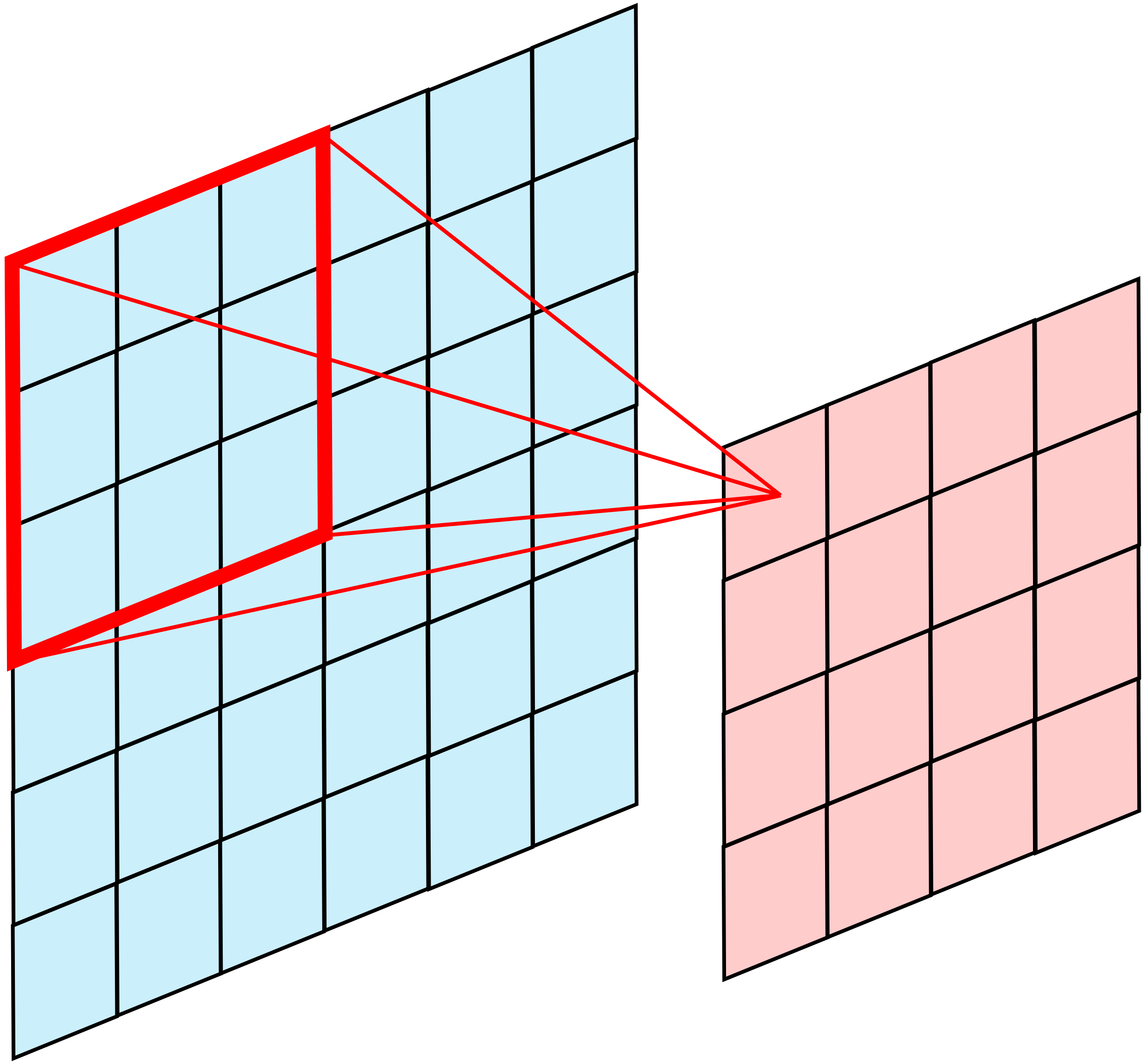
Image padded with zero pixels so that convolved image is of same size

Usage in Computer Vision

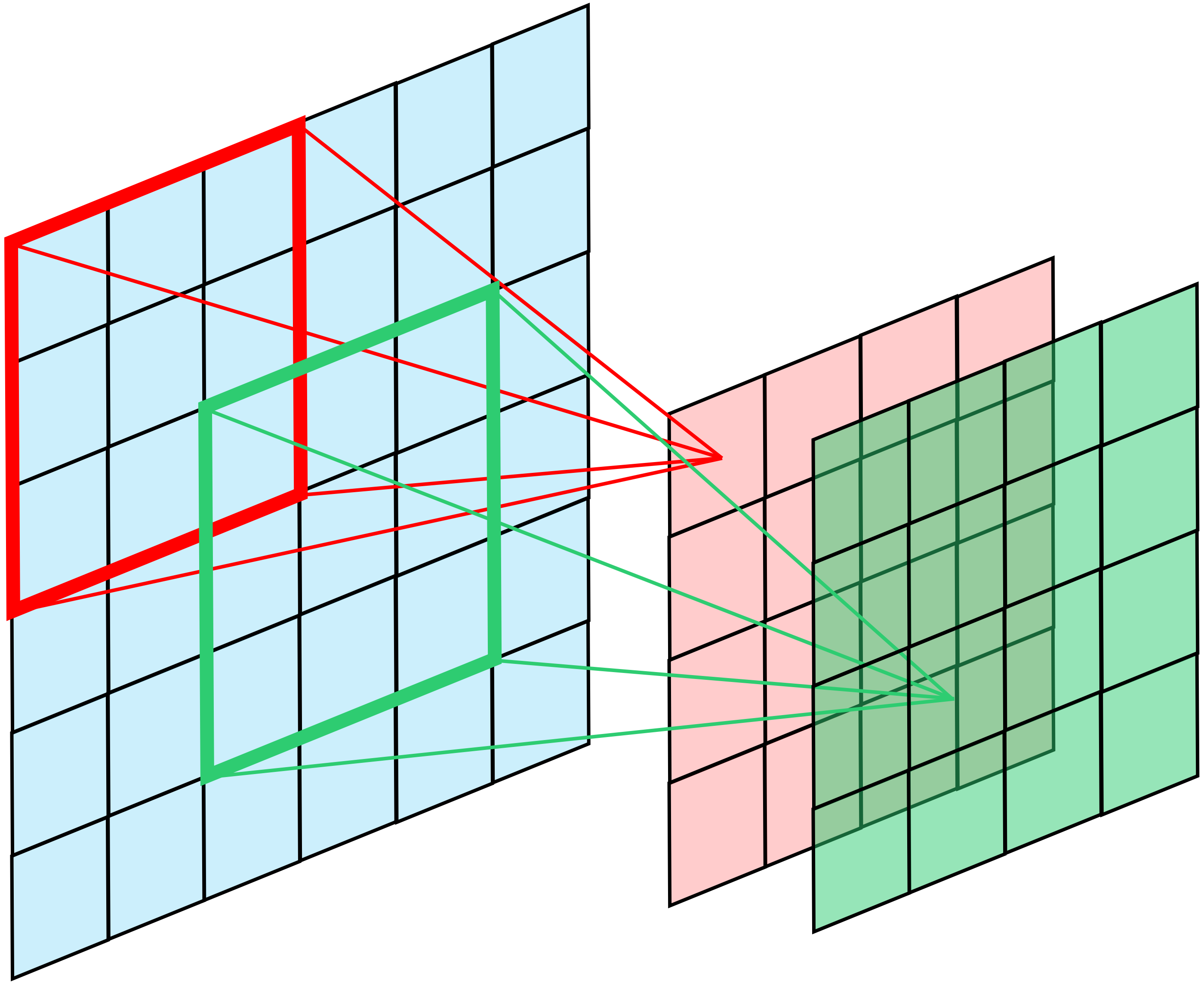
Usage in Computer Vision



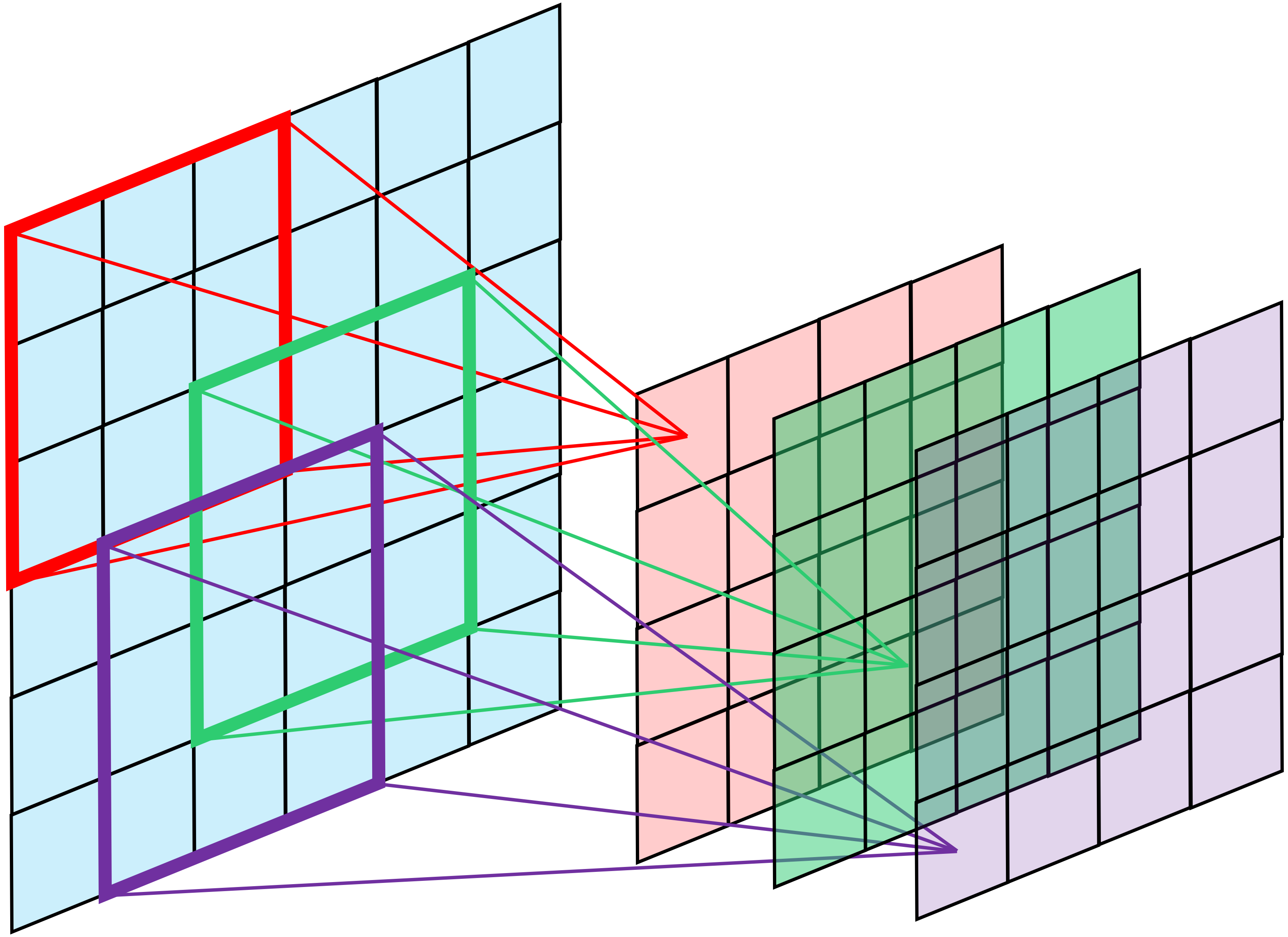
Usage in Computer Vision



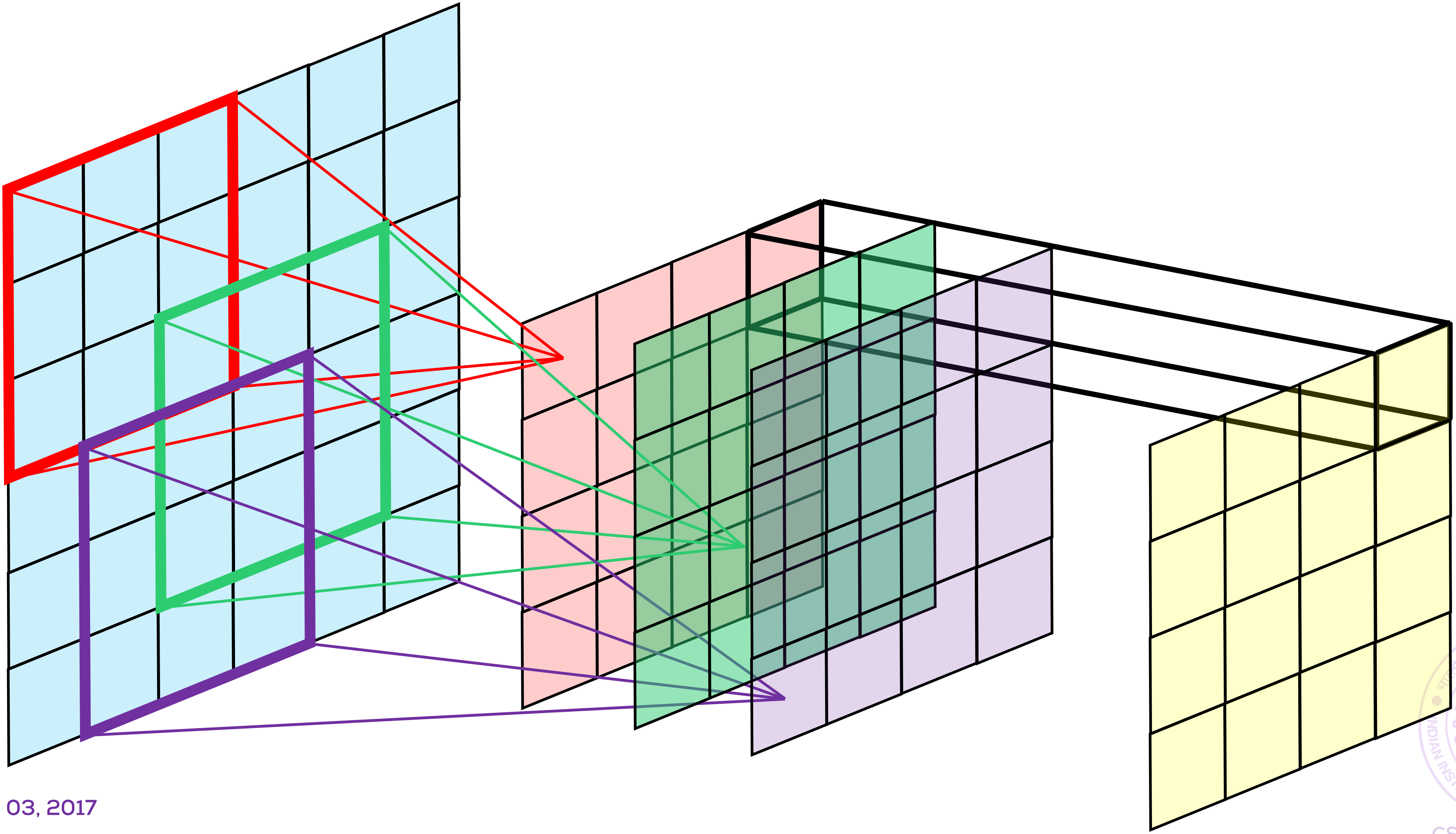
Usage in Computer Vision



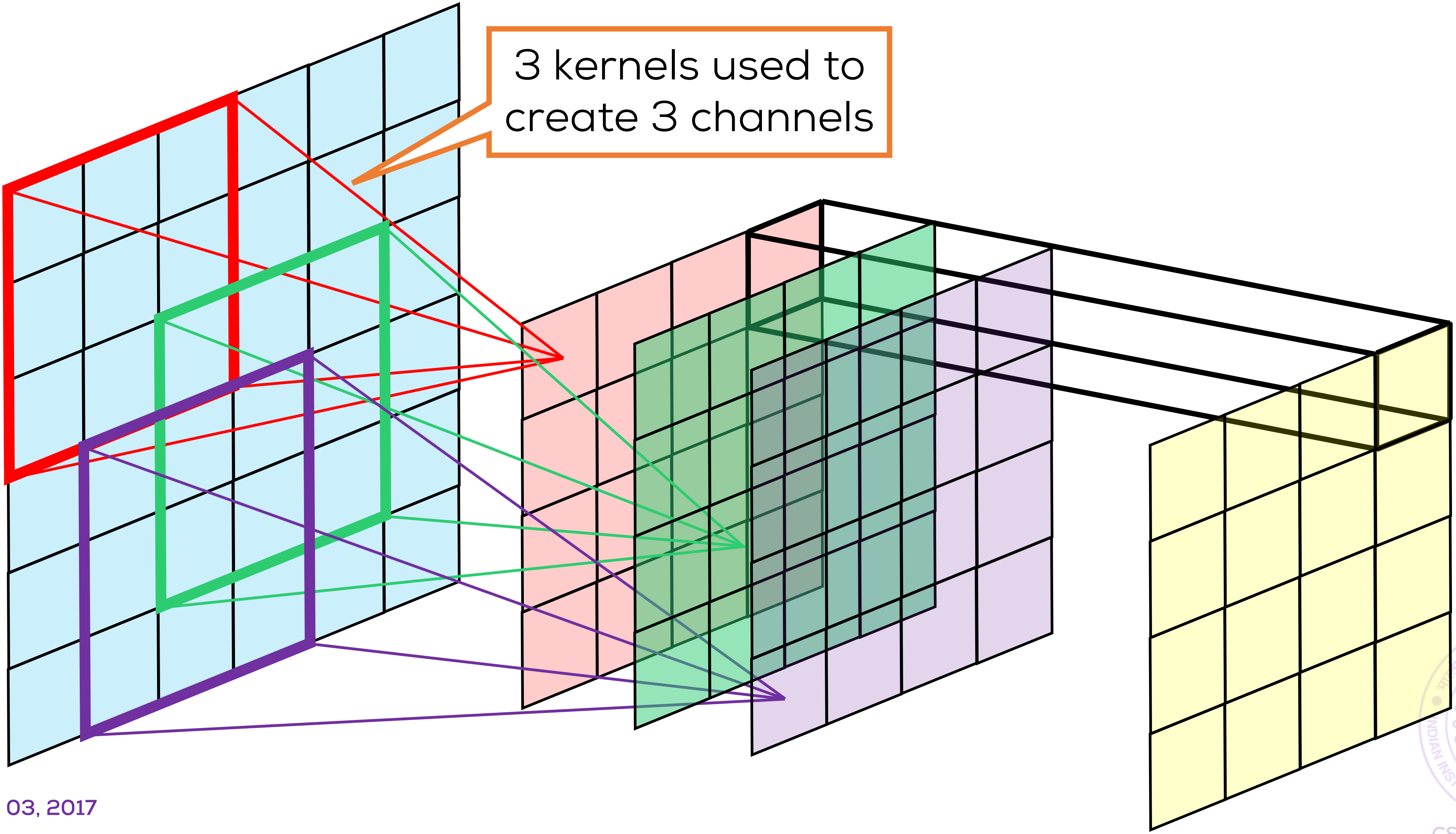
Usage in Computer Vision



Usage in Computer Vision

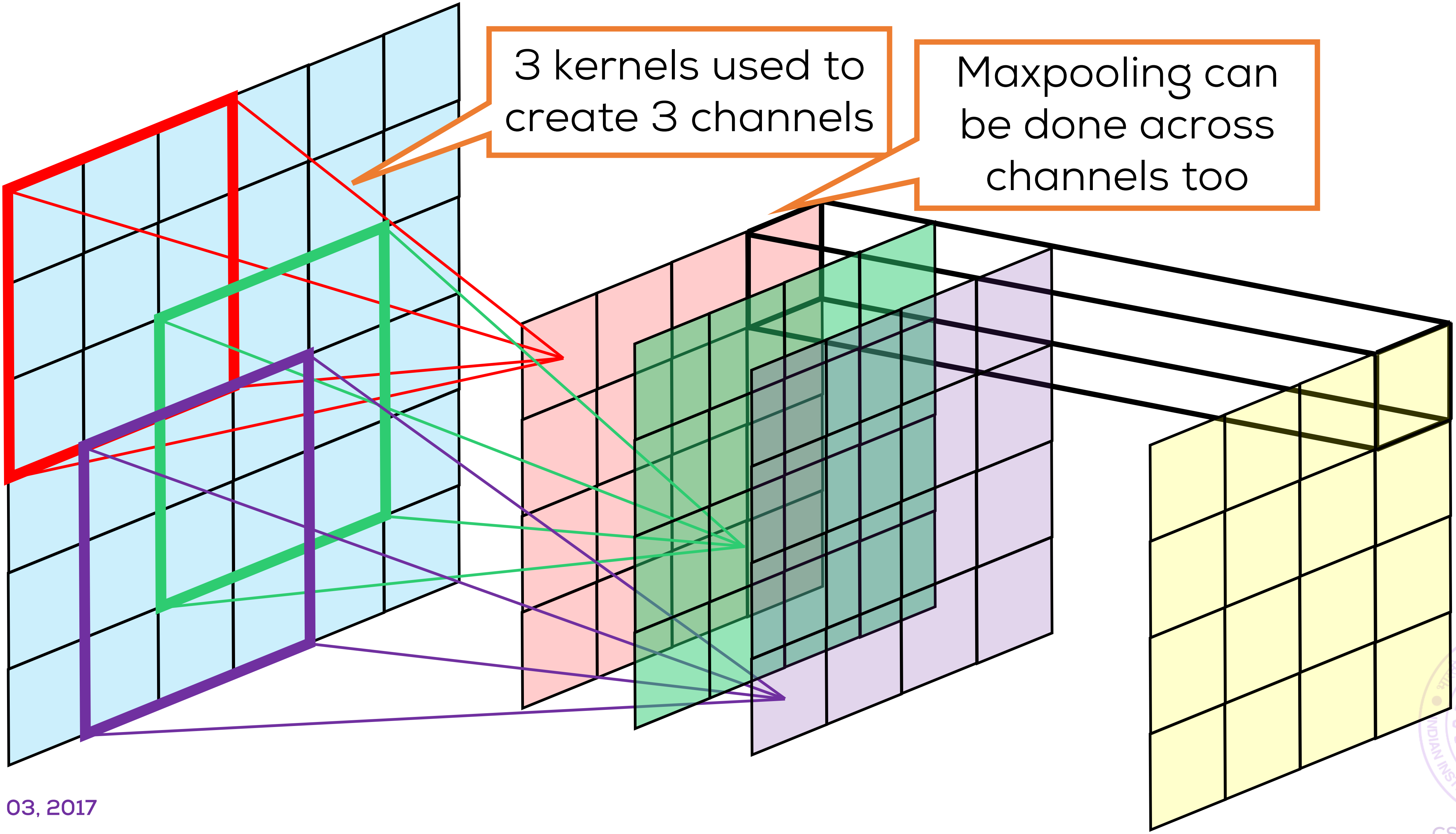


Usage in Computer Vision



3 kernels used to create 3 channels

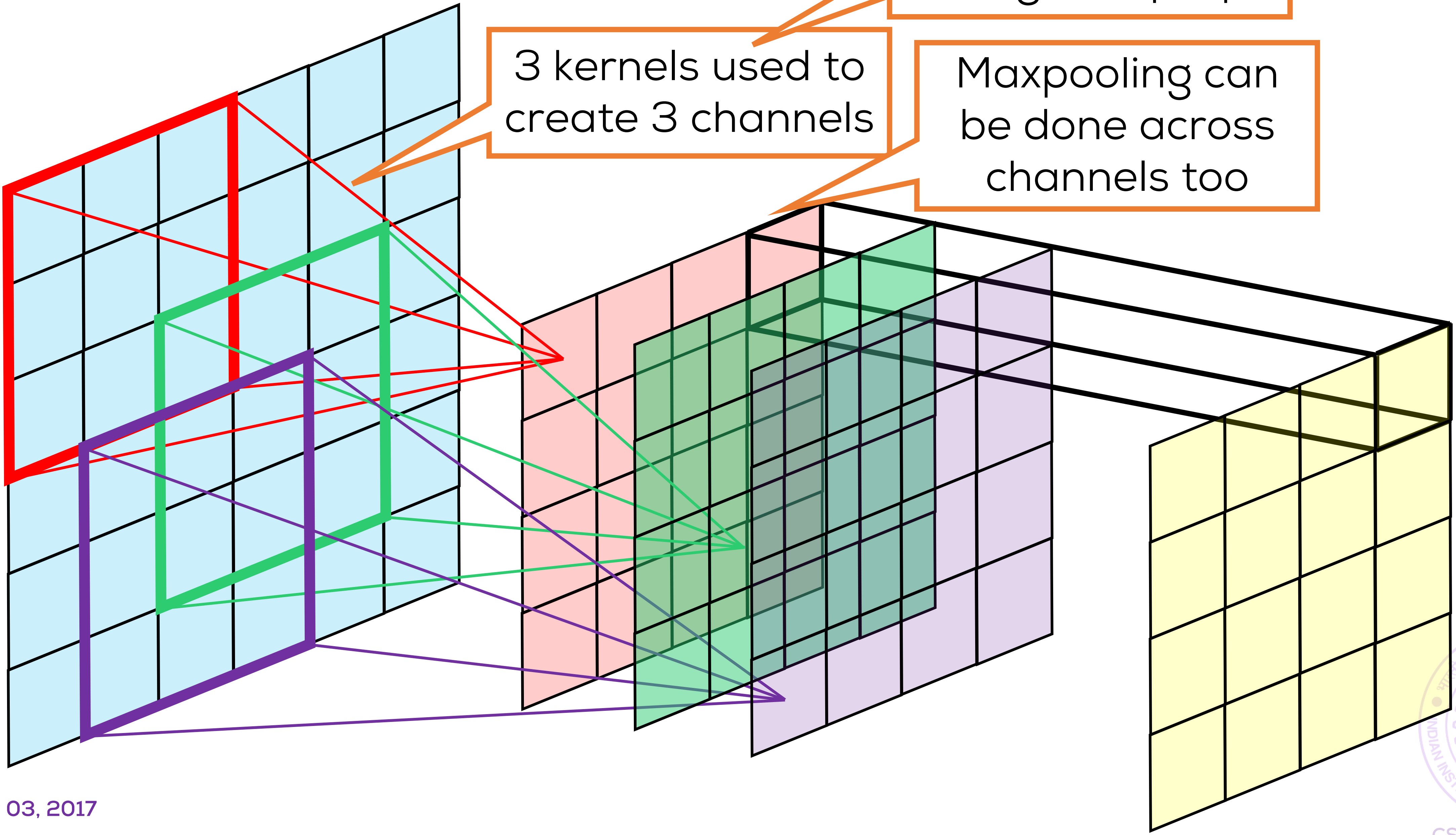
Usage in Computer Vision



3 kernels used to create 3 channels

Maxpooling can be done across channels too

Usage in Computer Vision

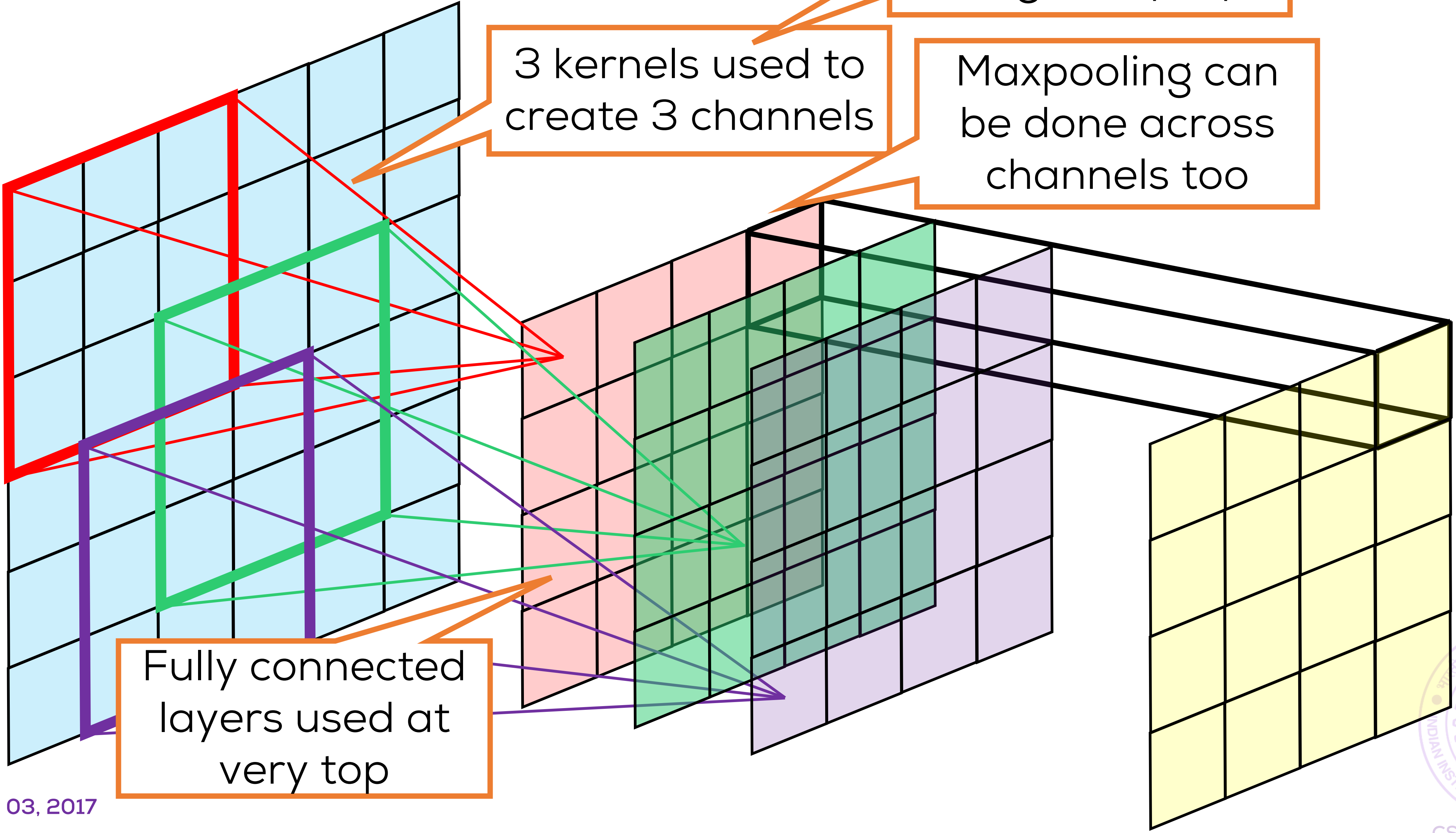


Kernels learnt using backprop

3 kernels used to create 3 channels

Maxpooling can be done across channels too

Usage in Computer Vision



Kernels learnt using backprop

3 kernels used to create 3 channels

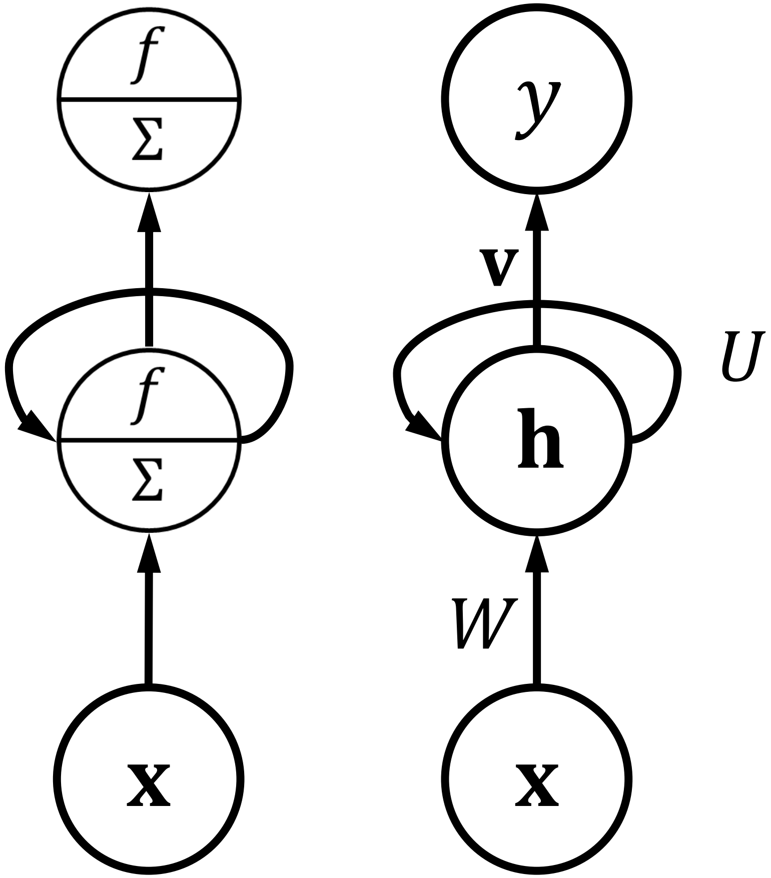
Maxpooling can be done across channels too

Fully connected layers used at very top

Recurrent Neural Networks

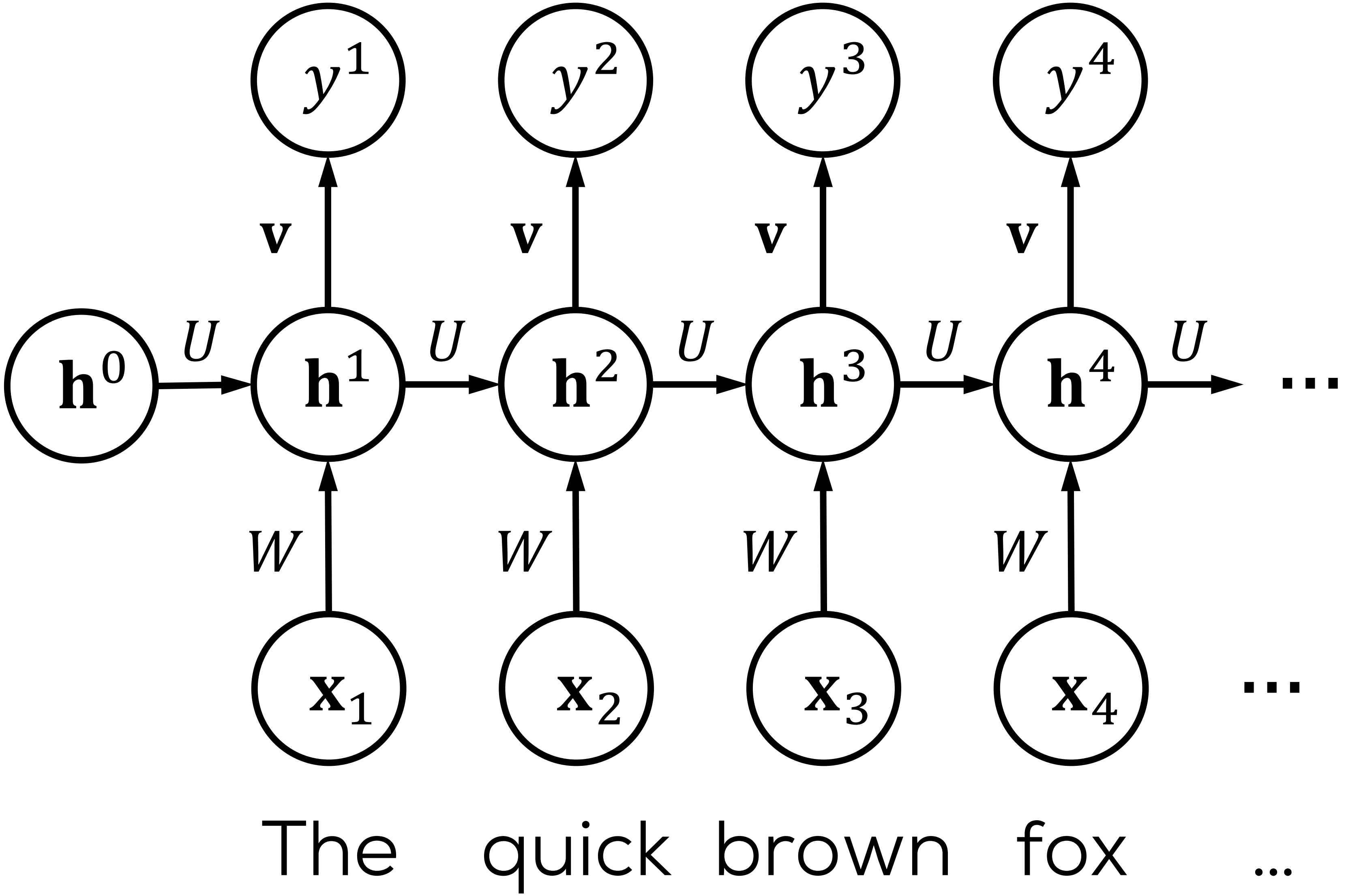
Learning with Sequence Data

- Textual data, time series data (stock values etc) are best represented as a sequence
- All NNs seen till now require fixed dimensional input
- Violated if working with sequence data (length of sequence varies)
- Recurrent neural networks handle this by violating the no feedback loop rule of feedforward networks
- For sake of clarity, we will represent entire layers by a single node
- … and change depiction of NN a bit



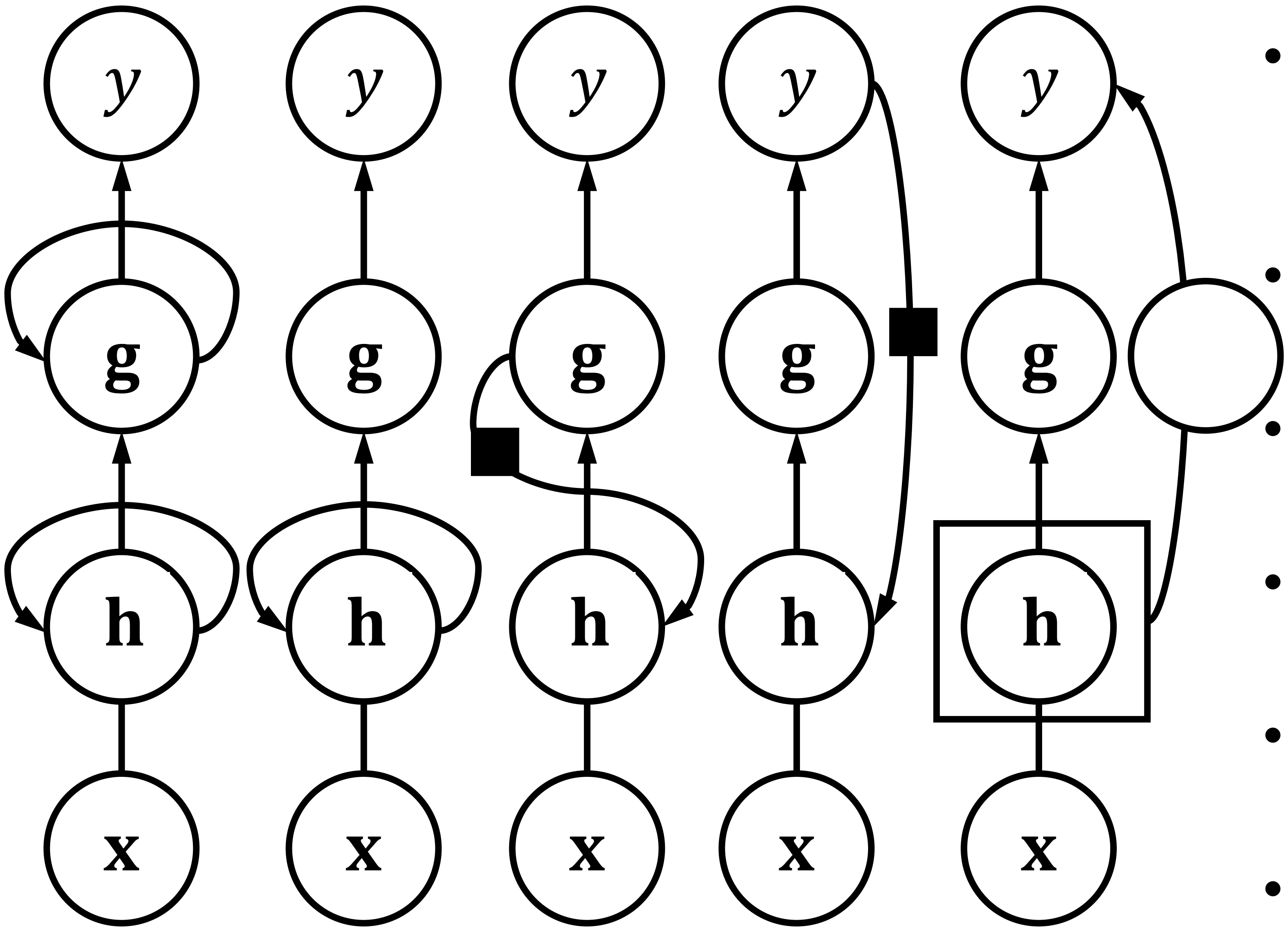
$$\hat{y} = \langle \mathbf{v}, \mathbf{h} \rangle$$
$$\mathbf{h} = f(W\mathbf{x} + U\mathbf{h})$$

An RNN in Action!



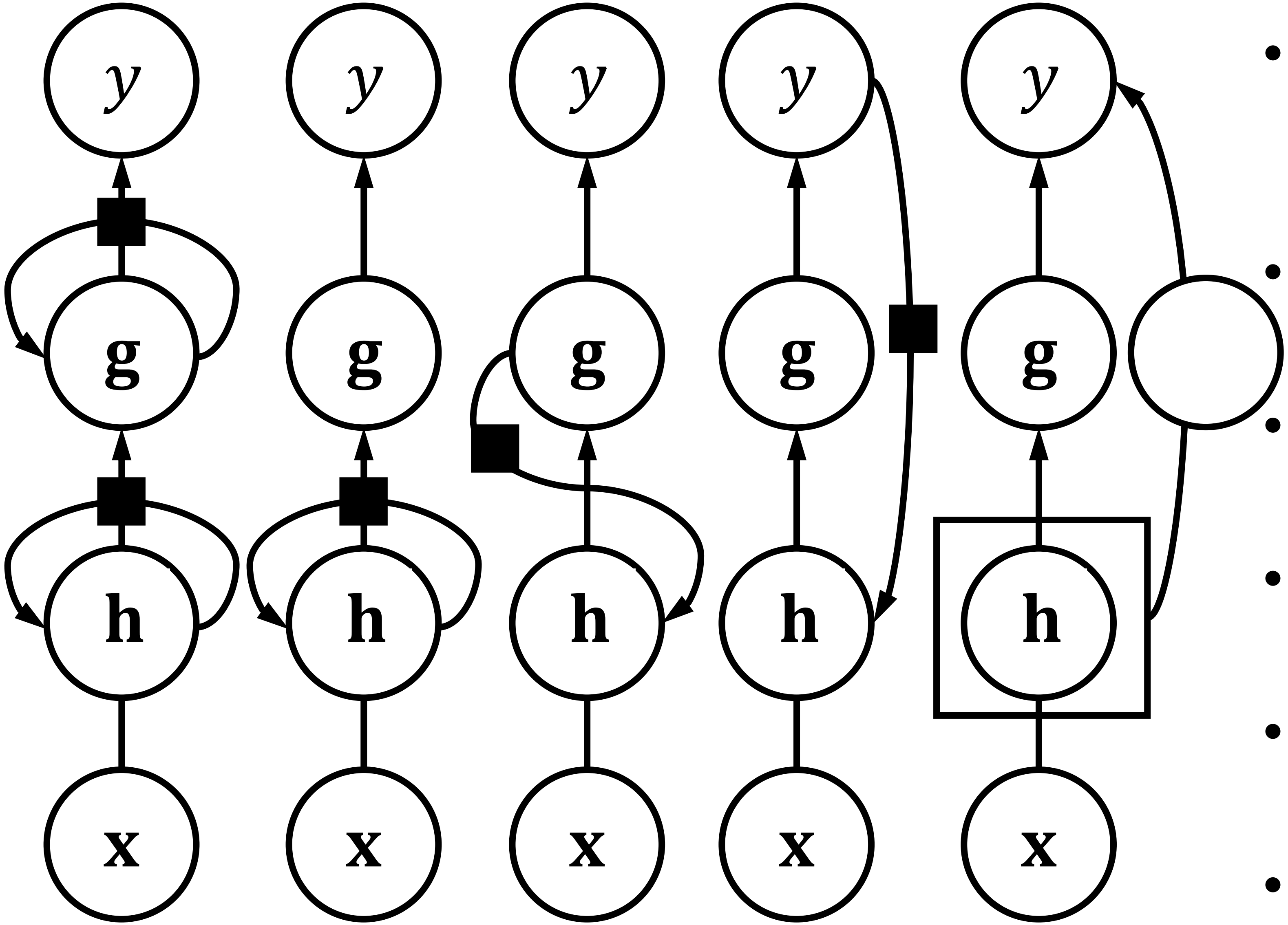
- $\hat{y}^t = \langle \mathbf{v}, \mathbf{h}^t \rangle$
- $\mathbf{h}^t = f(W\mathbf{x}^t + U\mathbf{h}^{t-1})$
- $\hat{y}^t$ can be used to do POS tagging
- $\hat{y}^t$ can even be a vector $\hat{\mathbf{y}}^t$
- Can have several hidden layers
- If several hidden layers, some maybe non-recurrent
- … or cross connect

RNN Variants



- ■ indicates a time lag. $\mathbf{g}^t / y^t$ is passed onto $\mathbf{h}^{t+1}$ not $\mathbf{h}^t$ (omitted often)
- The fourth variant is called “teacher forcing”
- At test time since y^t is not available, $\hat{y}^t$ passed
- The last variant is called an *attention* mechanism
- Very powerful, popular. It is all you need!
- Can be used with recurrent nodes as well

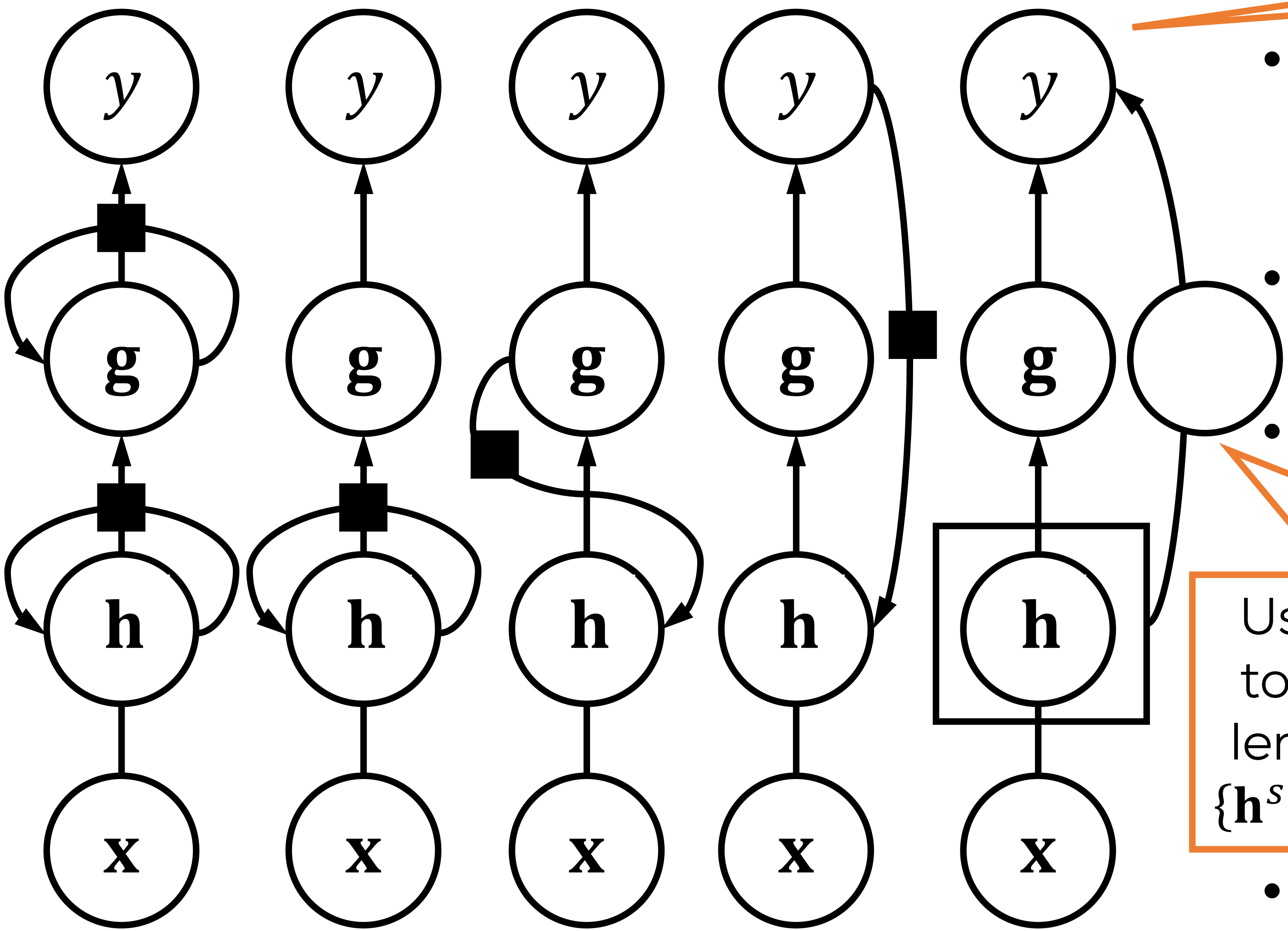
RNN Variants



- ■ indicates a time lag. $\mathbf{g}^t/y^t$ is passed onto $\mathbf{h}^{t+1}$ not $\mathbf{h}^t$ (omitted often)
- The fourth variant is called "teacher forcing"
- At test time since y^t is not available, $\hat{y}^t$ passed
- The last variant is called an *attention* mechanism
- Very powerful, popular. It is all you need!
- Can be used with recurrent nodes as well

RNN Variants

Note: while predicting y^t can access $\mathbf{h}^s$ for $s \neq t$ as well!



- ■ indicates a time lag. $\mathbf{g}^t / y^t$ is passed onto $\mathbf{h}^{t+1}$ not $\mathbf{h}^t$ (omitted often)
- The fourth variant is called "teacher forcing"
- At test time since y^t is not available, $\hat{y}^t$ passed

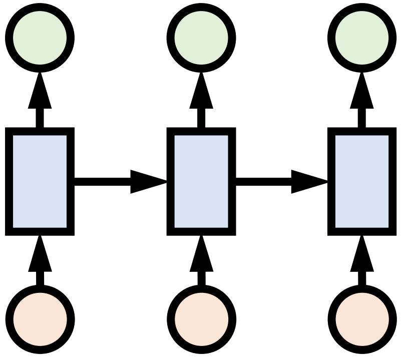
Usually a separate NN is used to select a subset $S_t \subset [T]$ (T is length of seq) such that tokens $\{\mathbf{h}^s : s \in S_t\}$ useful in predicting y^t

- Can be used with recurrent nodes as well

RNN Applications

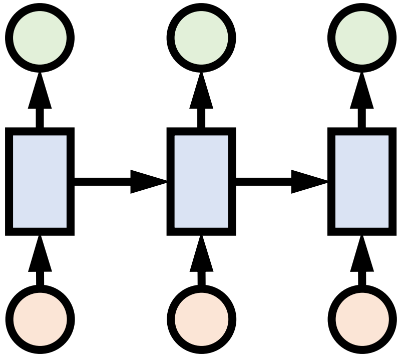
RNN Applications

Aligned Seq2Seq



RNN Applications

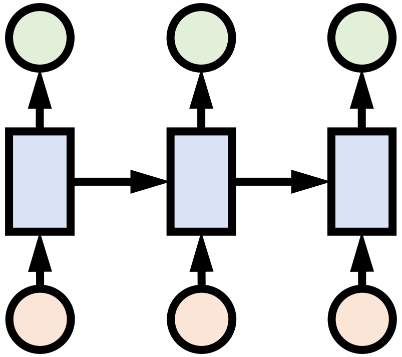
Aligned Seq2Seq



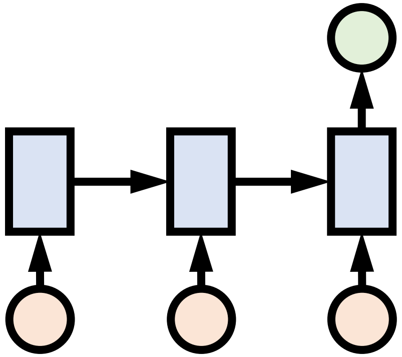
- POS tagging, predicting next word, language model learning, labelling frames of a video

RNN Applications

Aligned Seq2Seq



Sequence to Single

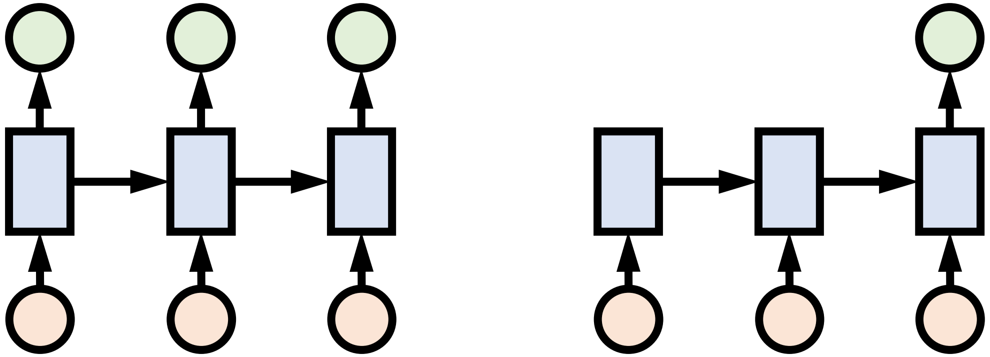


- POS tagging, predicting next word, language model learning, labelling frames of a video

RNN Applications

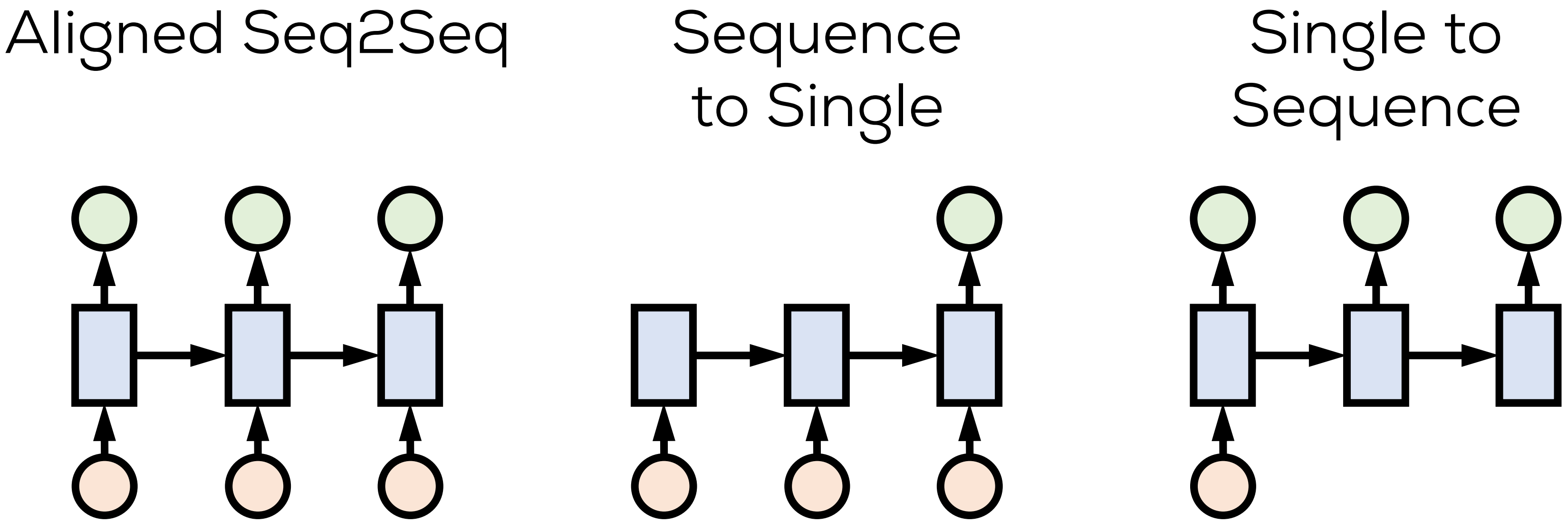
Aligned Seq2Seq

Sequence to Single



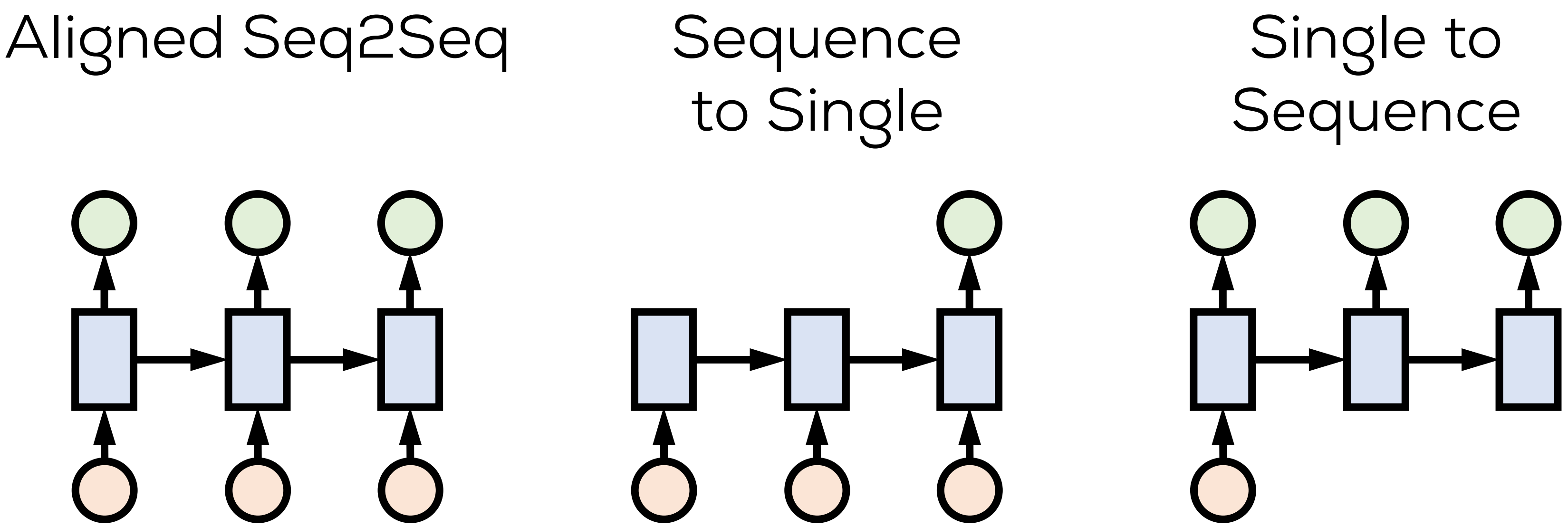
- POS tagging, predicting next word, language model learning, labelling frames of a video
- Sentiment analysis, video/document classification

RNN Applications



- POS tagging, predicting next word, language model learning, labelling frames of a video
- Sentiment analysis, video/document classification

RNN Applications



Aligned Seq2Seq

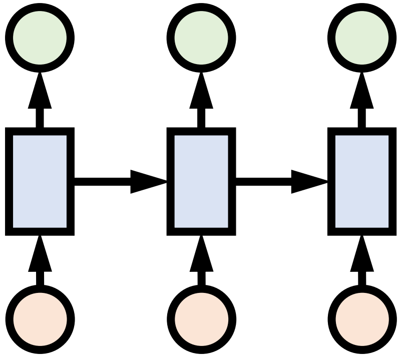
Sequence to Single

Single to Sequence

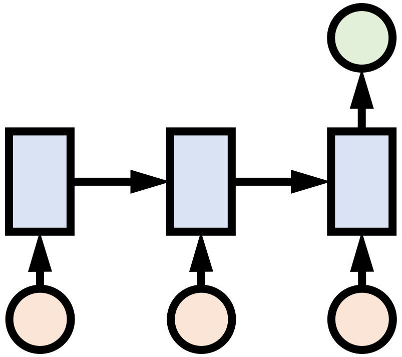
- POS tagging, predicting next word, language model learning, labelling frames of a video
- Sentiment analysis, video/document classification
- Image captioning

RNN Applications

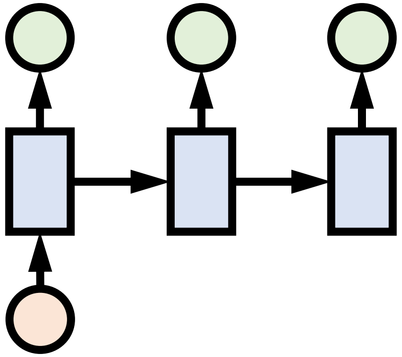
Aligned Seq2Seq



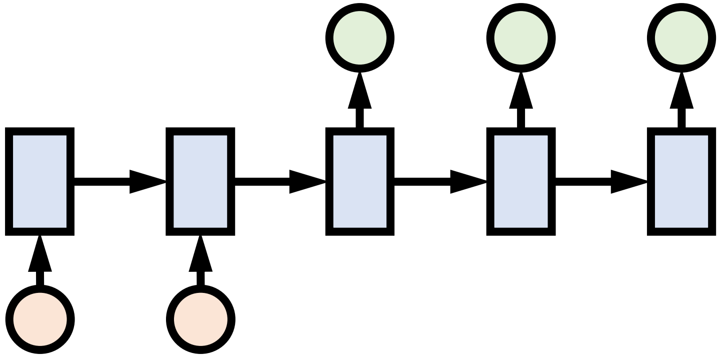
Sequence to Single



Single to Sequence



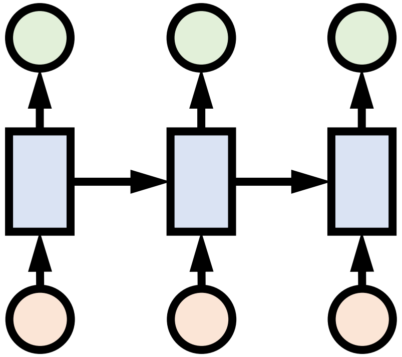
Non-aligned Seq2Seq



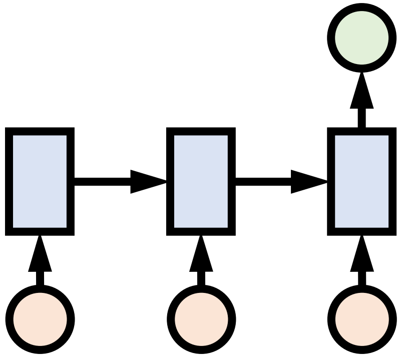
- POS tagging, predicting next word, language model learning, labelling frames of a video
- Sentiment analysis, video/document classification
- Image captioning

RNN Applications

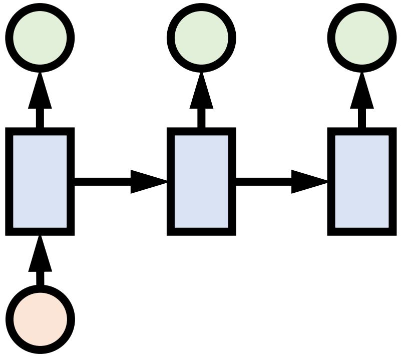
Aligned Seq2Seq



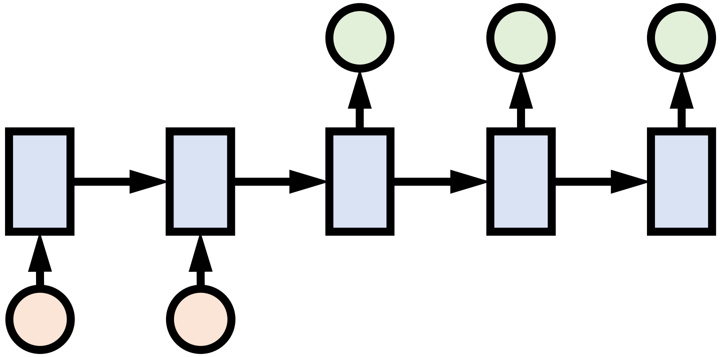
Sequence to Single



Single to Sequence



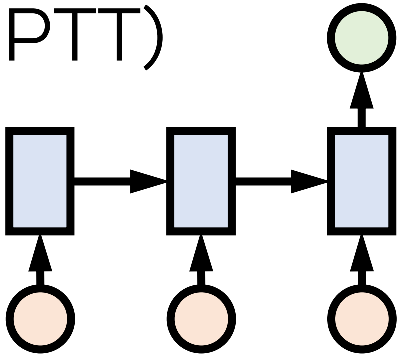
Non-aligned Seq2Seq



- POS tagging, predicting next word, language model learning, labelling frames of a video
- Sentiment analysis, video/document classification
- Image captioning
- Machine translation, query rewriting, error correction in input seq

Training RNNs

- A bit tricky since the simple network is “rolled” out across time
- Hence have to do ”Backpropagation Through Time” (BPTT)
- Lets look at only sequence to single prediction now
- We have $\hat{y} = \langle \mathbf{v}, \mathbf{h}^T \rangle$, and $\mathbf{h}^t = f(W\mathbf{x}^t + U\mathbf{h}^{t-1})$
- Need to be very careful about chain rule now



$$\frac{d\ell}{d\mathbf{v}} = \frac{d\ell}{d\hat{y}} \cdot \frac{d\hat{y}}{d\mathbf{v}} = \ell'(\hat{y}) \cdot \mathbf{h}^T$$

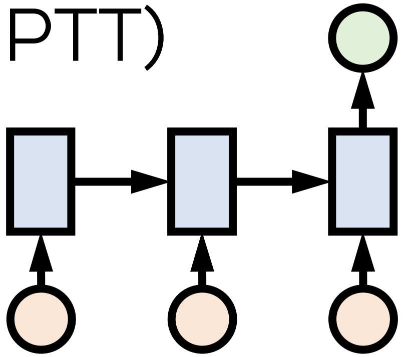
$$\frac{d\ell}{dW} = \frac{d\ell}{d\hat{y}} \cdot \frac{d\hat{y}}{dW} = \frac{d\ell}{d\hat{y}} \cdot \frac{d\hat{y}}{d\mathbf{h}^T} \cdot \frac{d\mathbf{h}^T}{dW} = \ell'(\hat{y}) \cdot \mathbf{v} \cdot \frac{d\mathbf{h}^T}{dW}$$

$$\frac{d\mathbf{h}^T}{dW} = \frac{d\mathbf{h}^T}{d\mathbf{z}^t} \cdot \frac{d\mathbf{z}^t}{dW} = J^f \cdot \left(\mathbf{x}^t + U \cdot \frac{d\mathbf{h}^{T-1}}{dW} \right) = \cdots$$

Training RNNs

Remember, ignore quantities $\mathbf{q}$ from chain rule if $\frac{d\mathbf{q}}{dW} = \mathbf{0}$

- A bit tricky since the simple network is "rolled" out across time
- Hence have to do "Backpropagation Through Time" (BPTT)
- Lets look at only sequence to single prediction now
- We have $\hat{y} = \langle \mathbf{v}, \mathbf{h}^T \rangle$, and $\mathbf{h}^t = f(W\mathbf{x}^t + U\mathbf{h}^{t-1})$
- Need to be very careful about chain rule now



Let $z^t = W\mathbf{x}^t + U\mathbf{h}^{t-1}$

$$\frac{d\ell}{d\mathbf{v}} = \frac{d\ell}{d\hat{y}} \cdot \frac{d\hat{y}}{d\mathbf{v}} = \ell'(\hat{y}) \cdot \mathbf{h}^T$$

$$\frac{d\ell}{dW} = \frac{d\ell}{d\hat{y}} \cdot \frac{d\hat{y}}{dW} = \frac{d\ell}{d\hat{y}} \cdot \frac{d\hat{y}}{d\mathbf{h}^T} \cdot \frac{d\mathbf{h}^T}{dW} = \ell'(\hat{y}) \cdot \mathbf{v} \cdot \frac{d\mathbf{h}^T}{dW}$$

$$\frac{d\mathbf{h}^T}{dW} = \frac{d\mathbf{h}^T}{d\mathbf{z}^t} \cdot \frac{d\mathbf{z}^t}{dW} = J^f \cdot \left(\mathbf{x}^t + U \cdot \frac{d\mathbf{h}^{T-1}}{dW} \right) = \cdots$$

Long Short-term Memory (LSTM)

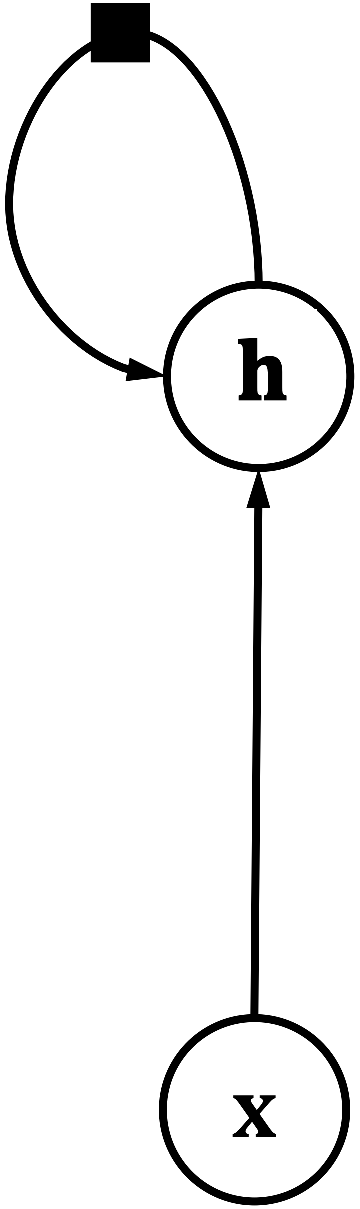
- Notice that

$$\frac{d\mathbf{h}^T}{dW} = J^f_{\mathbf{z}^T} \cdot U \cdot \frac{d\mathbf{h}^{T-1}}{dW} + \text{blah} = J^f_{\mathbf{z}^T} \cdot U \cdot J^f_{\mathbf{z}^{T-1}} \cdot U \cdot \frac{d\mathbf{h}^{T-2}}{dW} + \text{blah}$$

$$= J^f_{\mathbf{z}^T} \cdot U \cdot J^f_{\mathbf{z}^{T-1}} \cdot U \cdot J^f_{\mathbf{z}^{T-2}} \cdot U \cdot \frac{d\mathbf{h}^{T-3}}{dW} + \text{blah}$$

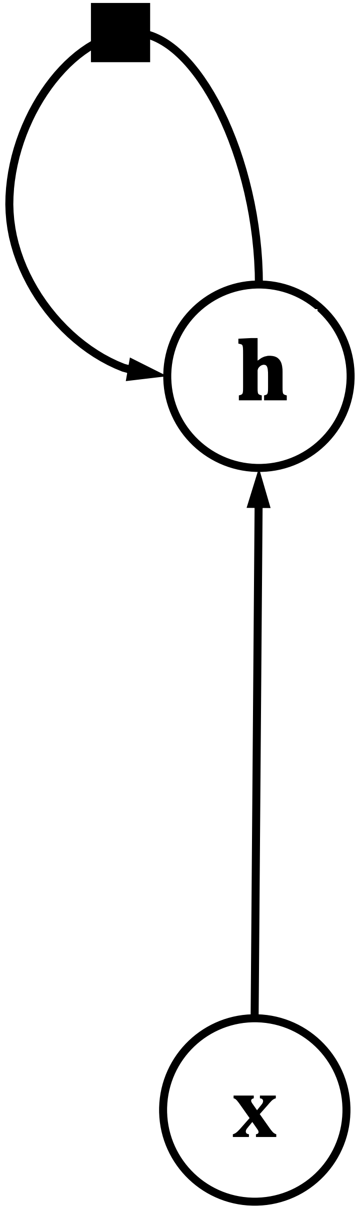
- Perfect recipe for gradients to either blow up or vanish entirely
- Many solutions: echo networks, skip connections, leaky units
- LSTMs found to be most successful
- Implement “gates” to stop/allow flow of data through time
- Other variants like Gated Recurrrent Units also work well

The LSTM Cell



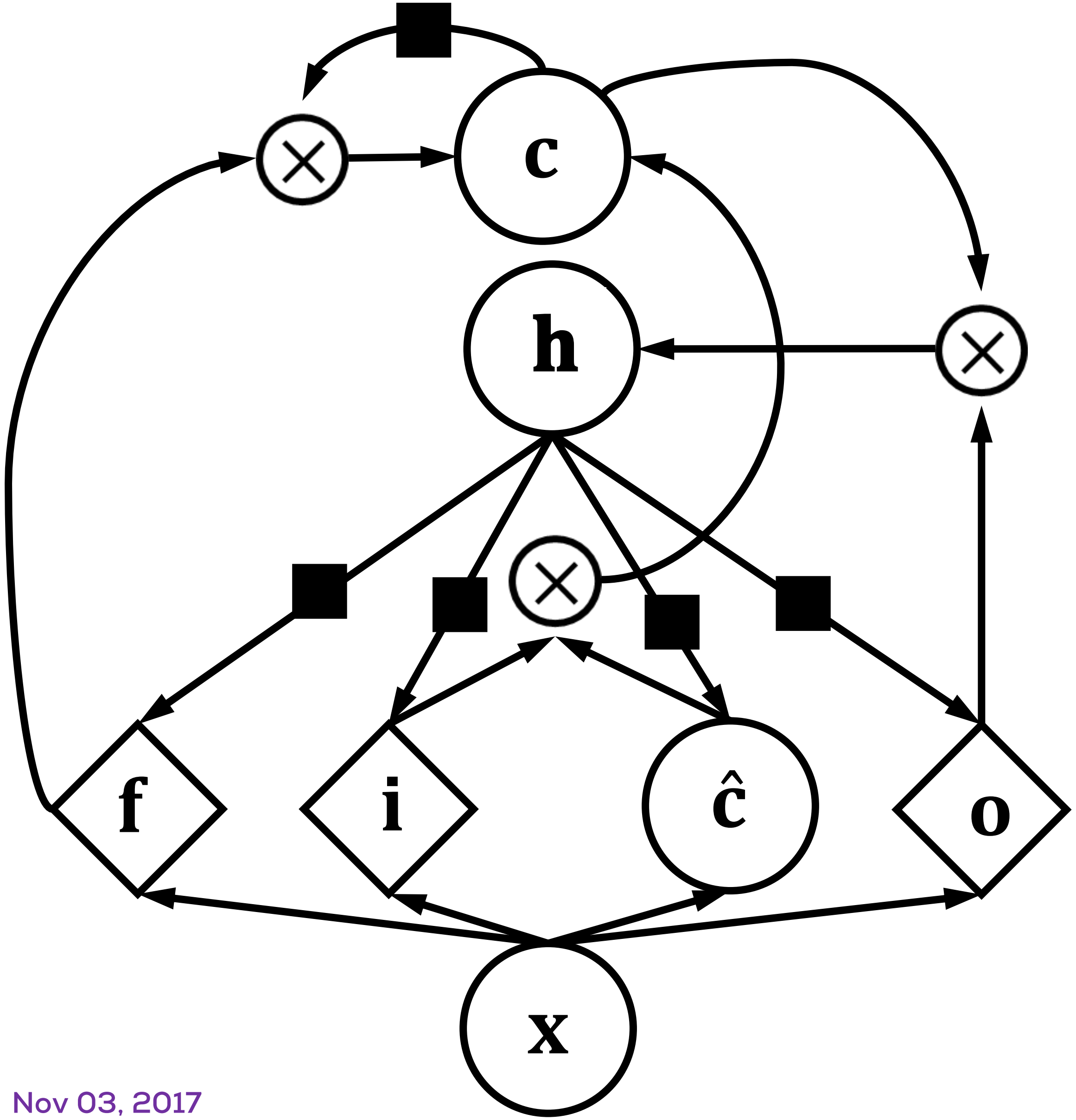
- Earlier $\mathbf{h}^t = f(W\mathbf{x}^t + U\mathbf{h}^{t-1})$

The LSTM Cell



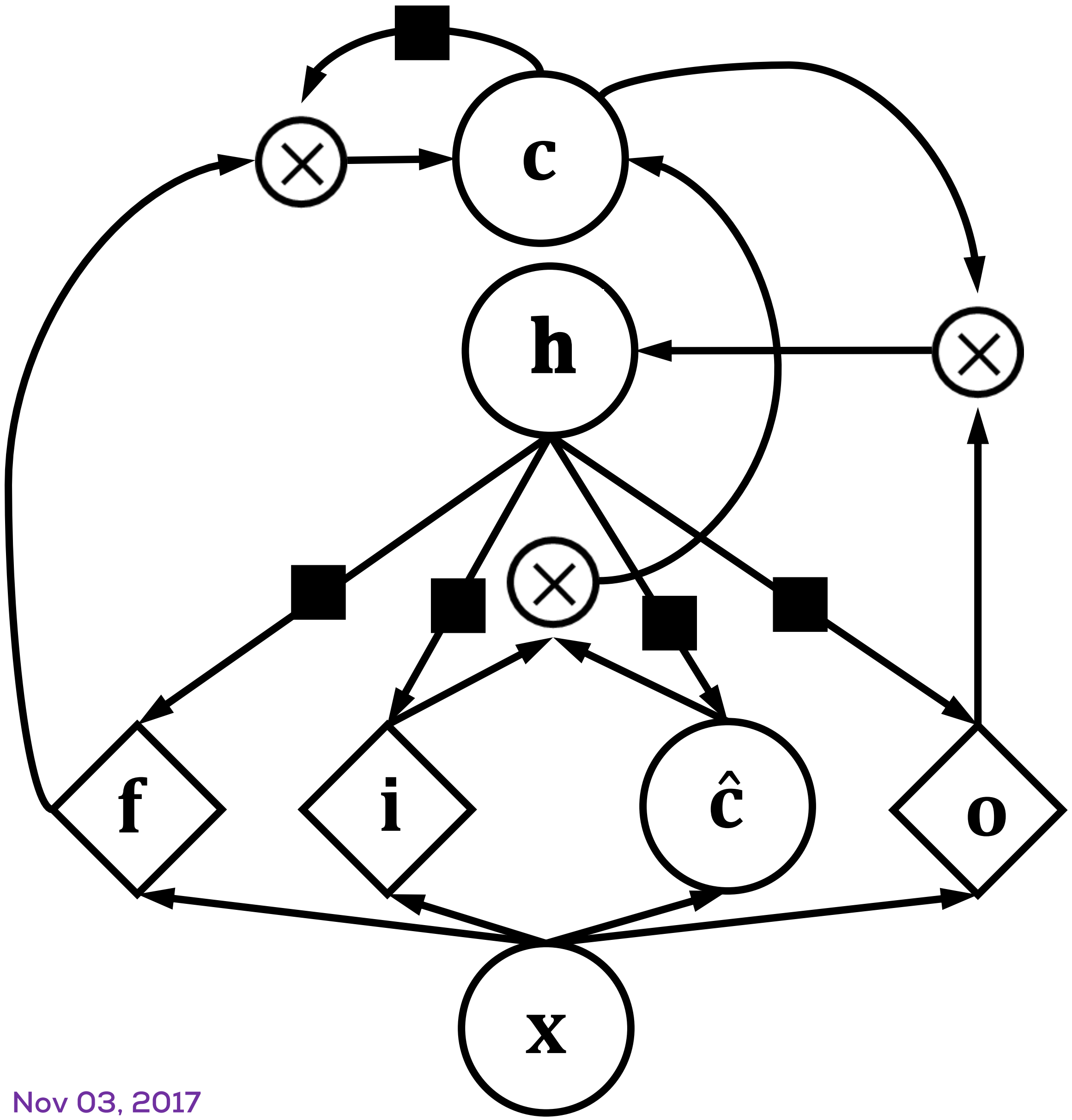
- Earlier $\mathbf{h}^t = f(W\mathbf{x}^t + U\mathbf{h}^{t-1})$
- Now we have a cell state $\mathbf{c}^t$

The LSTM Cell



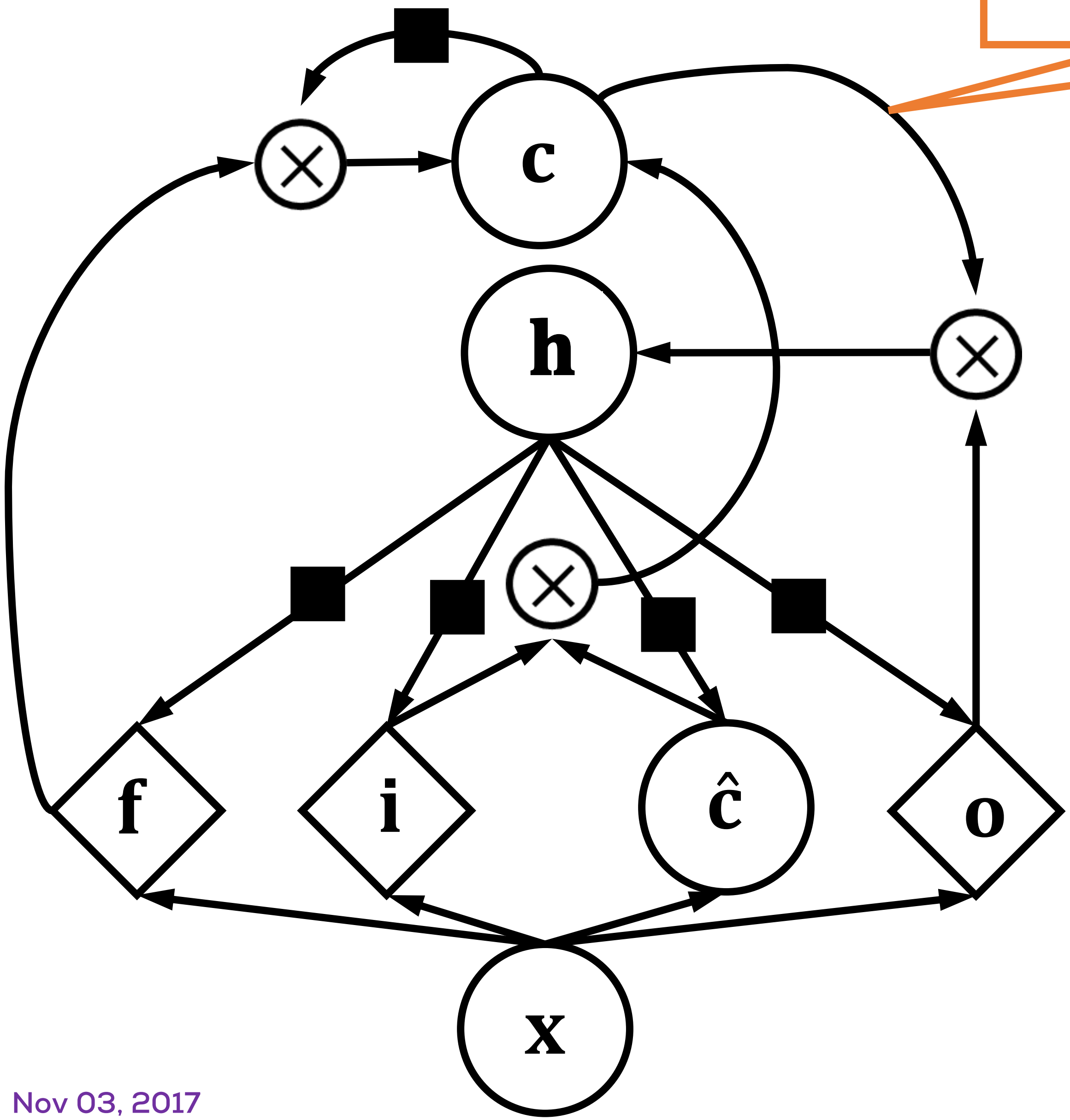
- Earlier $\mathbf{h}^t = f(W\mathbf{x}^t + U\mathbf{h}^{t-1})$
- Now we have a cell state $\mathbf{c}^t$

The LSTM Cell



- Earlier $\mathbf{h}^t = f(W\mathbf{x}^t + U\mathbf{h}^{t-1})$
- Now we have a cell state $\mathbf{c}^t$
- $\mathbf{c}^t = \mathbf{c}^{t-1} \otimes \mathbf{f}^t + \hat{\mathbf{c}}^t \otimes \mathbf{i}^t$
- $\hat{\mathbf{c}}^t = f(W^c\mathbf{x}_t + U^c\mathbf{h}^{t-1})$
- $\mathbf{i}^t = \sigma\left(W^i\mathbf{x}_t + U^i\mathbf{h}^{t-1}\right)$
- $\mathbf{o}^t = \sigma(W^o\mathbf{x}_t + U^o\mathbf{h}^{t-1})$
- $\mathbf{f}^t = \sigma\left(W^f\mathbf{x}_t + U^f\mathbf{h}^{t-1}\right)$
- $\mathbf{h}^t = \boldsymbol{o}^{\mathrm{t}} \otimes f(\mathbf{c}^t)$
- Output is $\hat{y}^t = \langle \mathbf{v}, \mathbf{h}^t \rangle$ as before

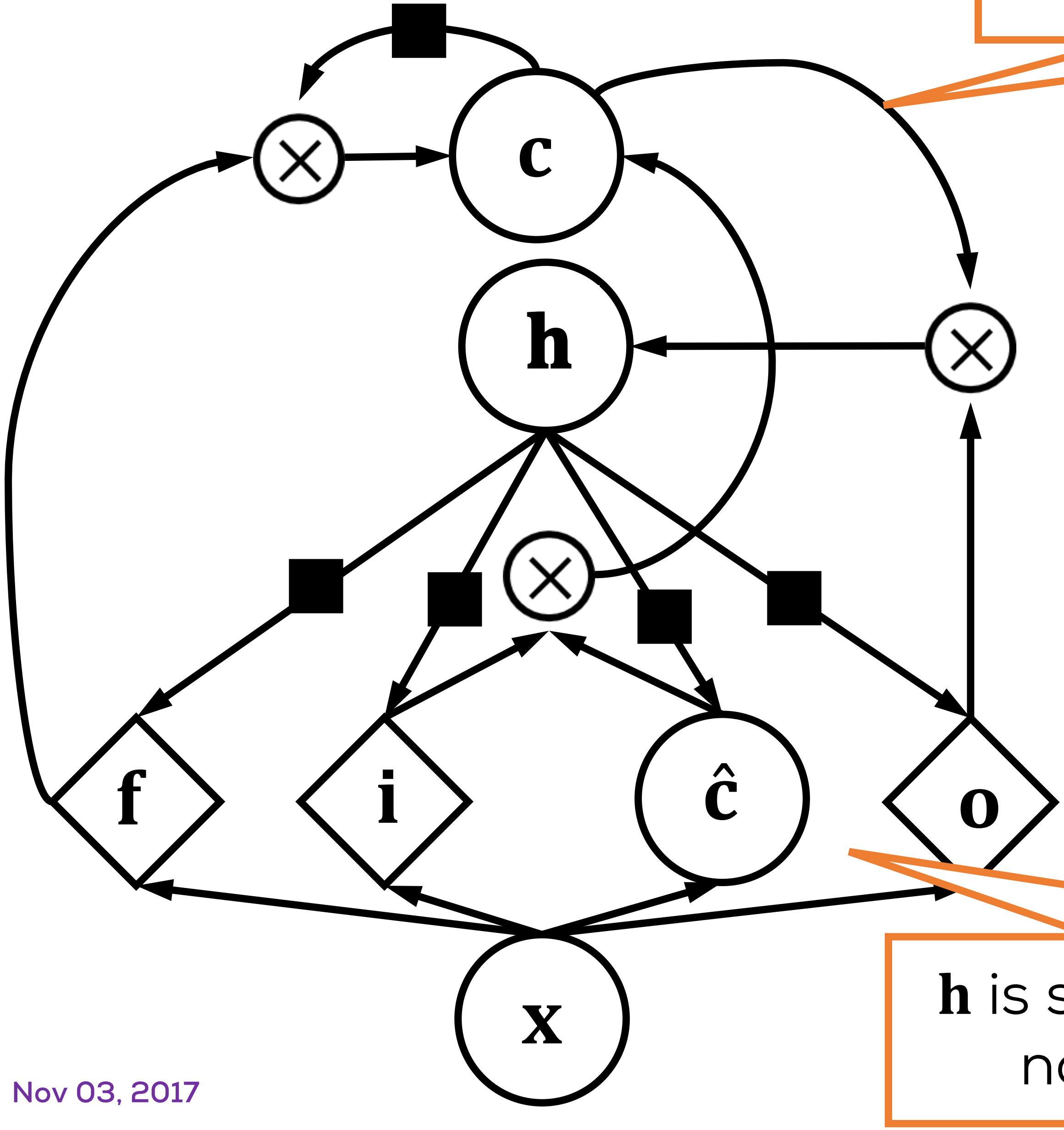
The LSTM Cell



Details are indeed a bit tedious but main idea is simple

- Earlier $\mathbf{h}^t = f(W\mathbf{x}^t + U\mathbf{h}^{t-1})$
- Now we have a cell state $\mathbf{c}^t$
- $\mathbf{c}^t = \mathbf{c}^{t-1} \otimes \mathbf{f}^t + \hat{\mathbf{c}}^t \otimes \mathbf{i}^t$
- $\hat{\mathbf{c}}^t = f(W^c\mathbf{x}_t + U^c\mathbf{h}^{t-1})$
- $\mathbf{i}^t = \sigma\left(W^i\mathbf{x}_t + U^i\mathbf{h}^{t-1}\right)$
- $\mathbf{o}^t = \sigma(W^o\mathbf{x}_t + U^o\mathbf{h}^{t-1})$
- $\mathbf{f}^t = \sigma\left(W^f\mathbf{x}_t + U^f\mathbf{h}^{t-1}\right)$
- $\mathbf{h}^t = \boldsymbol{o}^{\mathrm{t}} \otimes f(\mathbf{c}^t)$
- Output is $\hat{y}^t = \langle \mathbf{v}, \mathbf{h}^t \rangle$ as before

The LSTM Cell

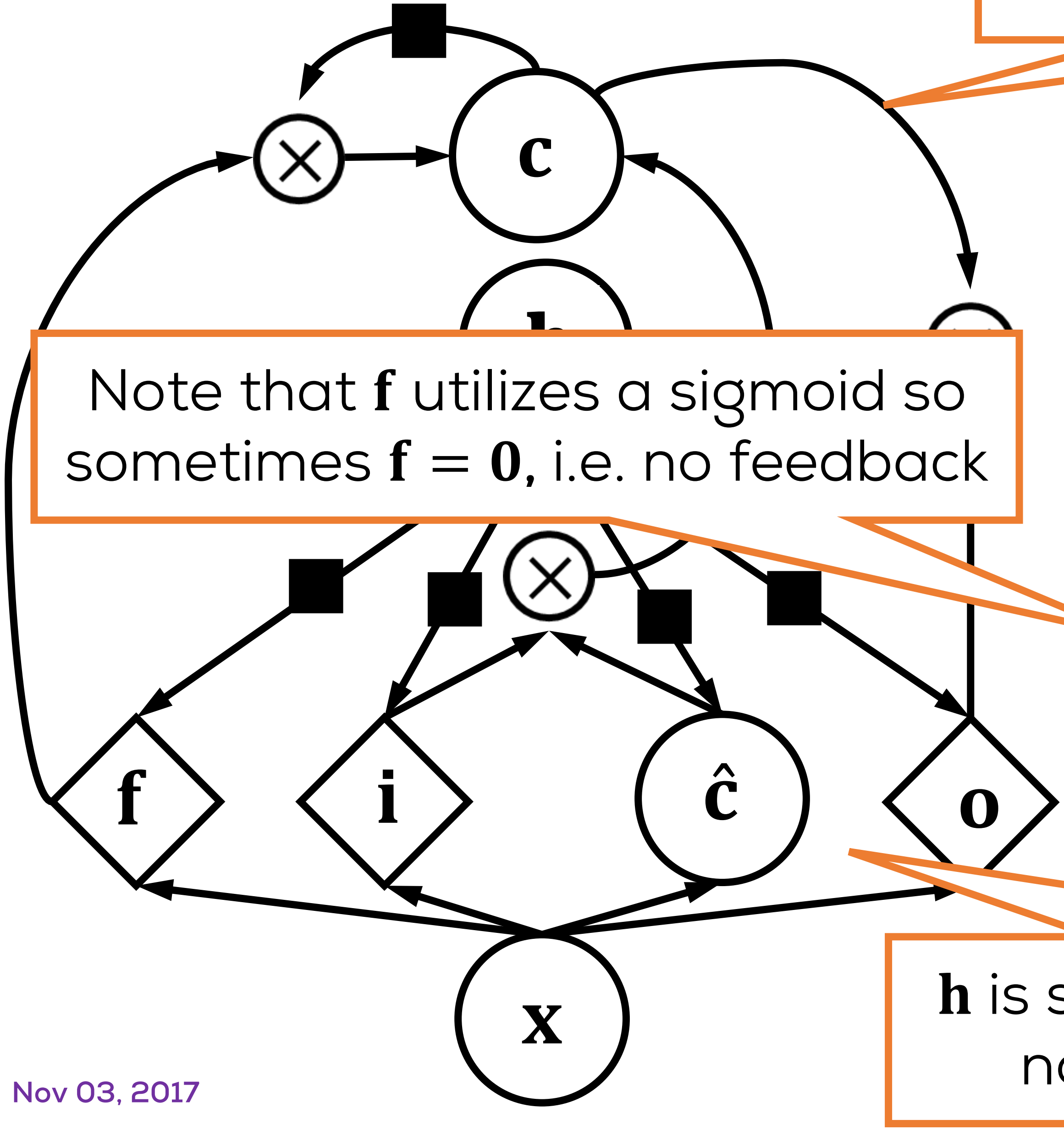


Details are indeed a bit tedious but main idea is simple

- Earlier $\mathbf{h}^t = f(W\mathbf{x}^t + U\mathbf{h}^{t-1})$
- Now we have a cell state $\mathbf{c}^t$
- $\mathbf{c}^t = \mathbf{c}^{t-1} \otimes \mathbf{f}^t + \hat{\mathbf{c}}^t \otimes \mathbf{i}^t$
- $\hat{\mathbf{c}}^t = f(W^c\mathbf{x}_t + U^c\mathbf{h}^{t-1})$
- $\mathbf{i}^t = \sigma\left(W^i\mathbf{x}_t + U^i\mathbf{h}^{t-1}\right)$
- $\mathbf{o}^t = \sigma(W^o\mathbf{x}_t + U^o\mathbf{h}^{t-1})$
- $\mathbf{f}^t = \sigma\left(W^f\mathbf{x}_t + U^f\mathbf{h}^{t-1}\right)$
- $\mathbf{h}^t = \boldsymbol{o}^{\mathrm{t}} \otimes f(\mathbf{c}^t)$
- Output is $\hat{y}^t = \langle \mathbf{v}, \mathbf{h}^t \rangle$ as before

h is still sending itself a self feedback but now it is routed through **c**, **o**, **i** and **f**

The LSTM Cell



Details are indeed a bit tedious but main idea is simple

- Earlier $\mathbf{h}^t = f(W\mathbf{x}^t + U\mathbf{h}^{t-1})$
- Now we have a cell state $\mathbf{c}^t$
- $\mathbf{c}^t = \mathbf{c}^{t-1} \otimes \mathbf{f}^t + \hat{\mathbf{c}}^t \otimes \mathbf{i}^t$
- $\hat{\mathbf{c}}^t = f(W^c\mathbf{x}_t + U^c\mathbf{h}^{t-1})$
- $\mathbf{i}^t = \sigma\left(W^i\mathbf{x}_t + U^i\mathbf{h}^{t-1}\right)$
- $\mathbf{o}^t = \sigma(W^o\mathbf{x}_t + U^o\mathbf{h}^{t-1})$
- $\mathbf{f}^t = \sigma\left(W^f\mathbf{x}_t + U^f\mathbf{h}^{t-1}\right)$
- $\mathbf{h}^t = \boldsymbol{o}^{\mathrm{t}} \otimes f(\mathbf{c}^t)$
- Output is $\hat{y}^t = \langle \mathbf{v}, \mathbf{h}^t \rangle$ as before

Note that $\mathbf{f}$ utilizes a sigmoid so sometimes $\mathbf{f} = \mathbf{0}$, i.e. no feedback

$\mathbf{h}$ is still sending itself a self feedback but now it is routed through $\mathbf{c}, \mathbf{o}, \mathbf{i}$ and $\mathbf{f}$

The LSTM Cell

- $\mathbf{f}$ is a forget gate. Sometimes it tells the LSTM to forget to receive feedback from the previous hidden state
- Note that $\mathbf{o}$ also acts like a forget gate. It sometimes tells the LSTM to forget to send feedback to the next hidden state
- $\mathbf{i}$ is also a forget gate. It sometimes tells the LSTM to forget to take the input into account when computing the hidden state
- All these put together prevent gradients from blowing up or diminishing to nothingness
- Caution: in textbooks, the “output” of the network is used to refer to $\mathbf{h}^t$ and not $\hat{y}^\mathrm{t}$. The output forget gate $\mathbf{o}$ also directly controls $\mathbf{h}^t$

Deep Learning

- Very active area right now
- Too vast to be covered in few lectures
- Rules-of-thumb, accepted practices changing rapidly
- Keeping up with published literature only way to stay fresh
- Exciting applications to reinforcement learning, question-answering, “artificial intelligence”

Please give your Feedback

http://tinyurl.com/ml17-18afb