Deep Learning IV

CS771: Introduction to Machine Learning

Purushottam Kar

Announcements

Assignment 3 released

Deadline November 23, 2019, 9:59PM IST

Very close to grade submission deadline – do not rely on “extensions”

Use non-linear methods (kernels, NN) judiciously for this assignment

Do not forget to also explore LwP, linear methods

In ML, good and insightful feature engineering can beat the best of algos ☺

Fully exploit the simplicities and structure in given data

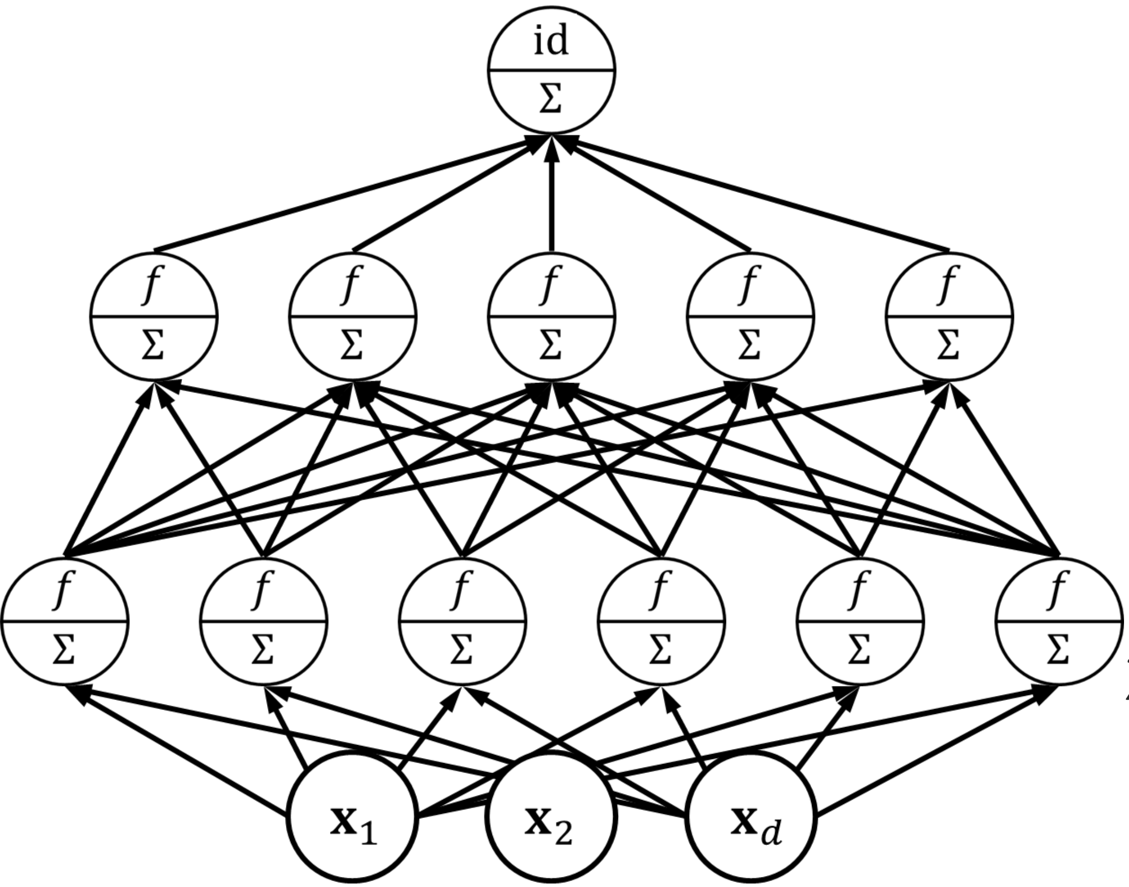
Recap of Last Lecture

Chain rule for multivariate functions (notion of Jacobian)

Backprogation rule for training DN using GD

Generative models via NN – autoencoders, GANs

Feedforward Networks can be massive



Fully connected layers are powerful

Allow all possible combinations of input dims to create new features which are functions of any subset of $\mathbf{x}_1, \mathbf{x}_2, \dots, \mathbf{x}_d$

New features of the form $f(\mathbf{w}^\top \mathbf{x})$

Also very unnecessary for apps where input has lots of structure

Make networks very bulky – e.g. the AE

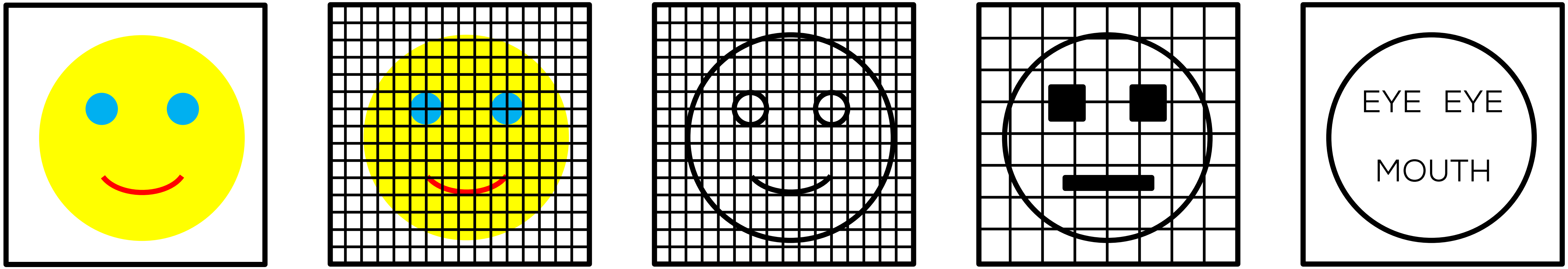
784 →1000→500→250→30→250→500→1000→784

needs 2.8 million edge weights to be learnt

From only 60 thousand data points ☹

Also require tons of data to train so many edge weights otherwise NN may overfit

Structured Data needs Local Features



Highly unlikely that top left and bottom right pixels would need to be considered together right at the first hidden layer to detect edges

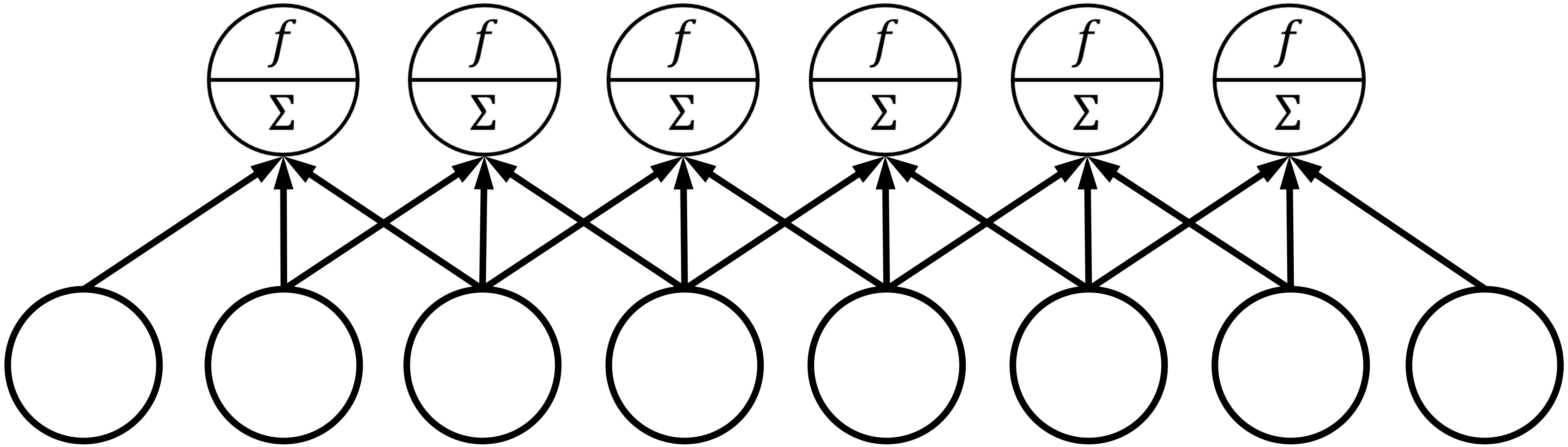
Only neighboring pixels need to talk to each other to detect edges. Also edge detection happens via "filters" – same filter needs to be applied everywhere

Then, need to aggregate info to detect structures

Then, need to detect even higher level features

Distant pixels are jointly considered, but at a much later stage (deeper layer)

Convolution Operation

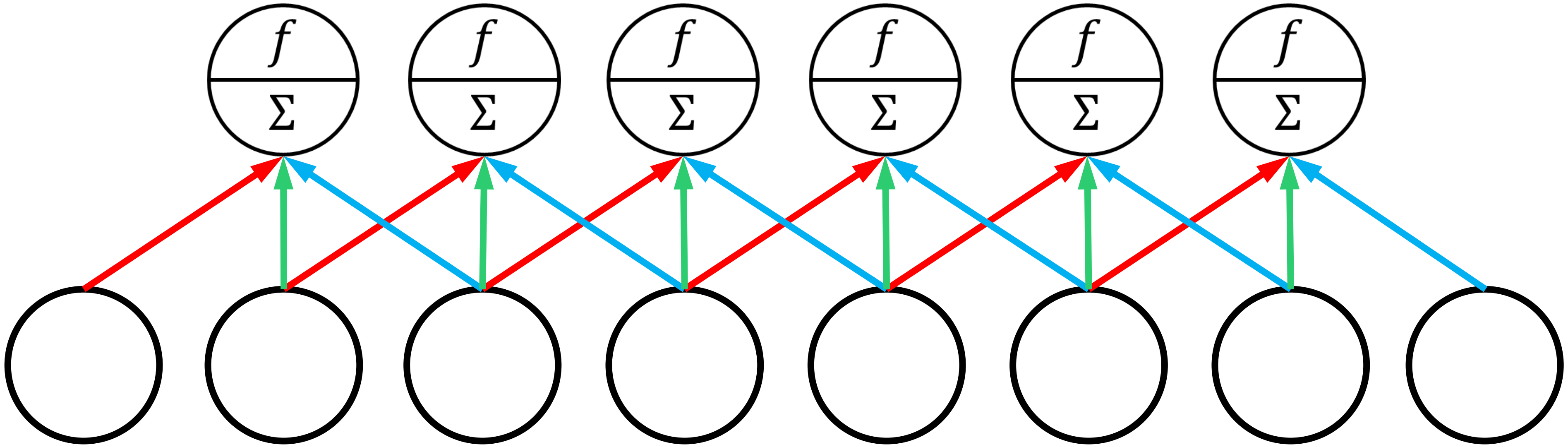


Convolutions are at the heart of signal processing and CNNs

Convolutions create layers which are sparsely connected

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

Convolution Operation



Convolutions are at the heart of signal processing and CNNs

Convolutions create layers which are sparsely connected

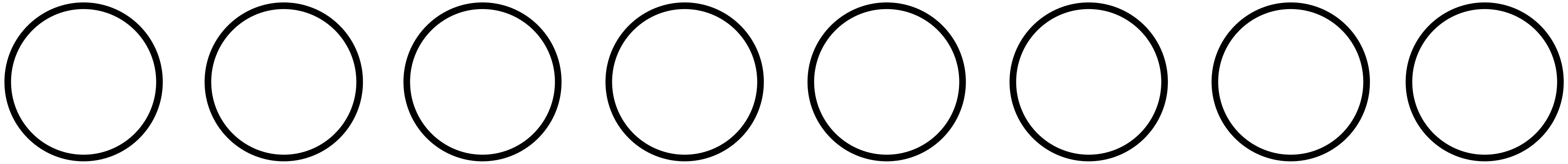
On top of that they force equality constraints among weights in that layer

All green edges forced to have the same weight, all red edges forced to have the same weight, all blue edges …

So effectively only 3 edge weights to be learnt for this layer!

A fully connected layer would have had 48 edges

Convolution Operation



Convolutions are at the heart of signal processing and CNNs

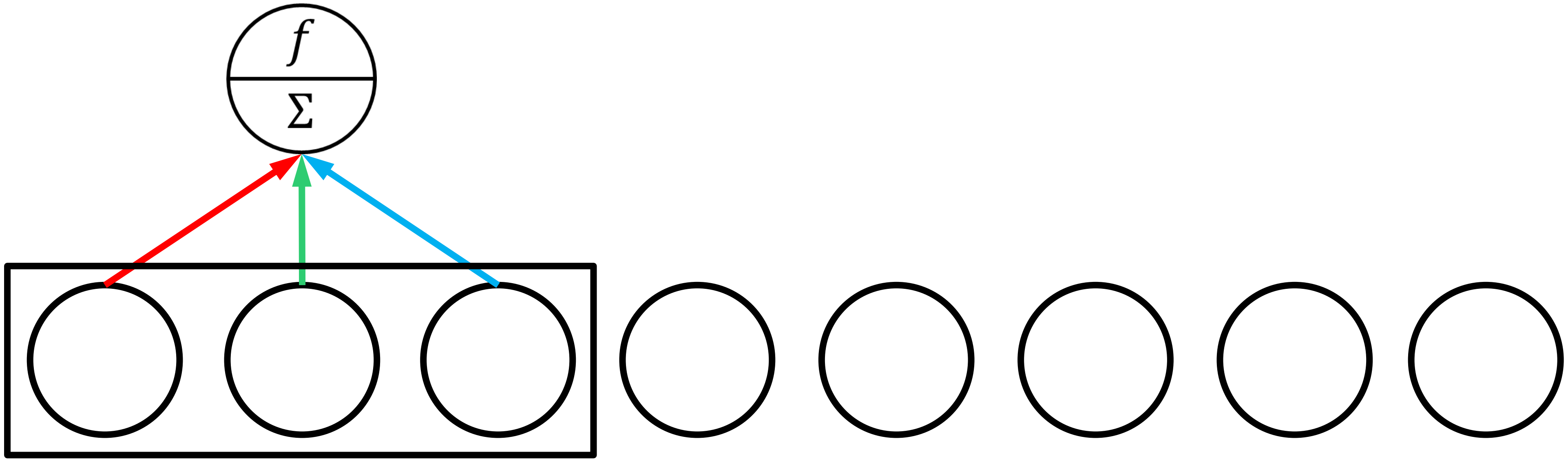
Convolutions create layers which are sparsely connected

On top of that they force equality constraints among weights in that layer

Doing so efficiently and cleverly so that we don't land up with a nasty constrained optimization problem is the key to the success of CNNs

$\mathbf{h}_i = f(\mathbf{x}_i \cdot \mathbf{w}_1 + \mathbf{x}_{i+1} \cdot \mathbf{w}_2 + \mathbf{x}_{i+2} \cdot \mathbf{w}_3)$ *i.e.* $\mathbf{h}_i = f\left(\sum_{j=1}^{3} \mathbf{x}_{i+j-1} \cdot \mathbf{w}_j\right)$

Convolution Operation



Convolutions are at the heart of signal processing and CNNs

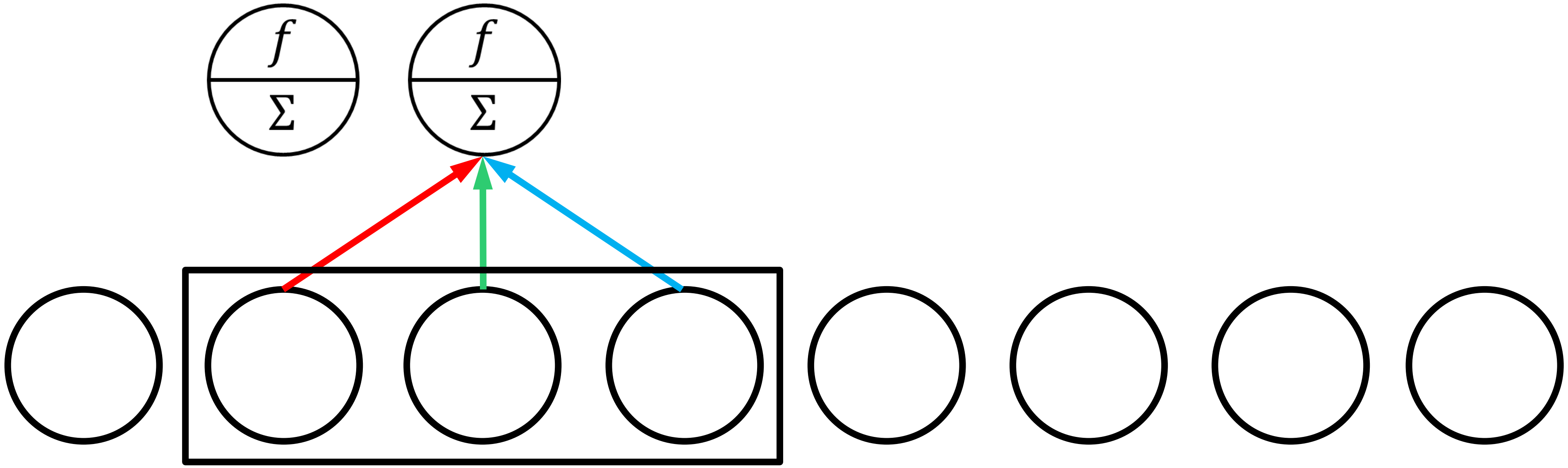
Convolutions create layers which are sparsely connected

On top of that they force equality constraints among weights in that layer

Doing so efficiently and cleverly so that we don't land up with a nasty constrained optimization problem is the key to the success of CNNs

$\mathbf{h}_i = f(\mathbf{x}_i \cdot \mathbf{w}_1 + \mathbf{x}_{i+1} \cdot \mathbf{w}_2 + \mathbf{x}_{i+2} \cdot \mathbf{w}_3)$ *i.e.* $\mathbf{h}_i = f\left(\sum_{j=1}^{3} \mathbf{x}_{i+j-1} \cdot \mathbf{w}_j\right)$

Convolution Operation



Convolutions are at the heart of signal processing and CNNs

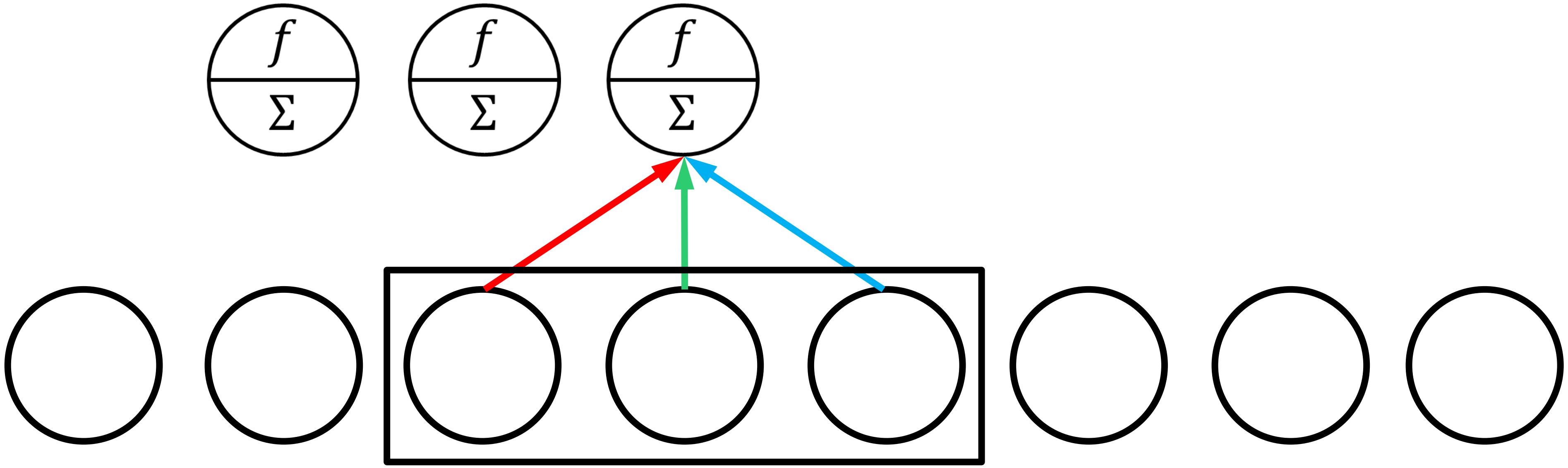
Convolutions create layers which are sparsely connected

On top of that they force equality constraints among weights in that layer

Doing so efficiently and cleverly so that we don't land up with a nasty constrained optimization problem is the key to the success of CNNs

$\mathbf{h}_i = f(\mathbf{x}_i \cdot \mathbf{w}_1 + \mathbf{x}_{i+1} \cdot \mathbf{w}_2 + \mathbf{x}_{i+2} \cdot \mathbf{w}_3)$ *i.e.* $\mathbf{h}_i = f\left(\sum_{j=1}^{3} \mathbf{x}_{i+j-1} \cdot \mathbf{w}_j\right)$

Convolution Operation



Convolutions are at the heart of signal processing and CNNs

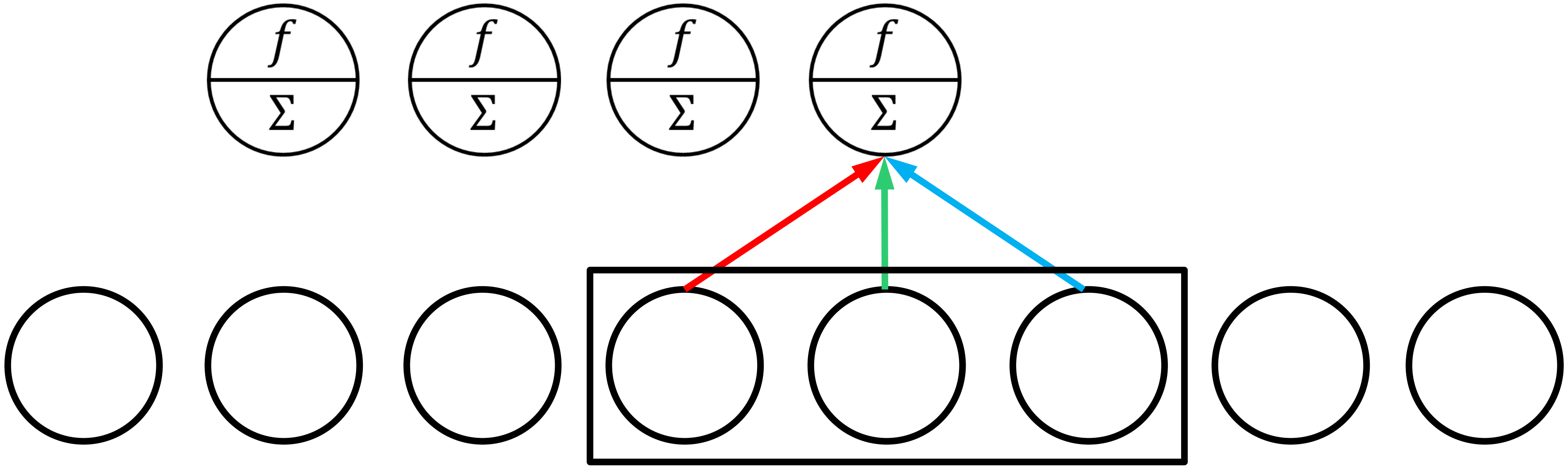
Convolutions create layers which are sparsely connected

On top of that they force equality constraints among weights in that layer

Doing so efficiently and cleverly so that we don't land up with a nasty constrained optimization problem is the key to the success of CNNs

$\mathbf{h}_i = f(\mathbf{x}_i \cdot \mathbf{w}_1 + \mathbf{x}_{i+1} \cdot \mathbf{w}_2 + \mathbf{x}_{i+2} \cdot \mathbf{w}_3)$ *i.e.* $\mathbf{h}_i = f\left(\sum_{j=1}^{3} \mathbf{x}_{i+j-1} \cdot \mathbf{w}_j\right)$

Convolution Operation



Convolutions are at the heart of signal processing and CNNs

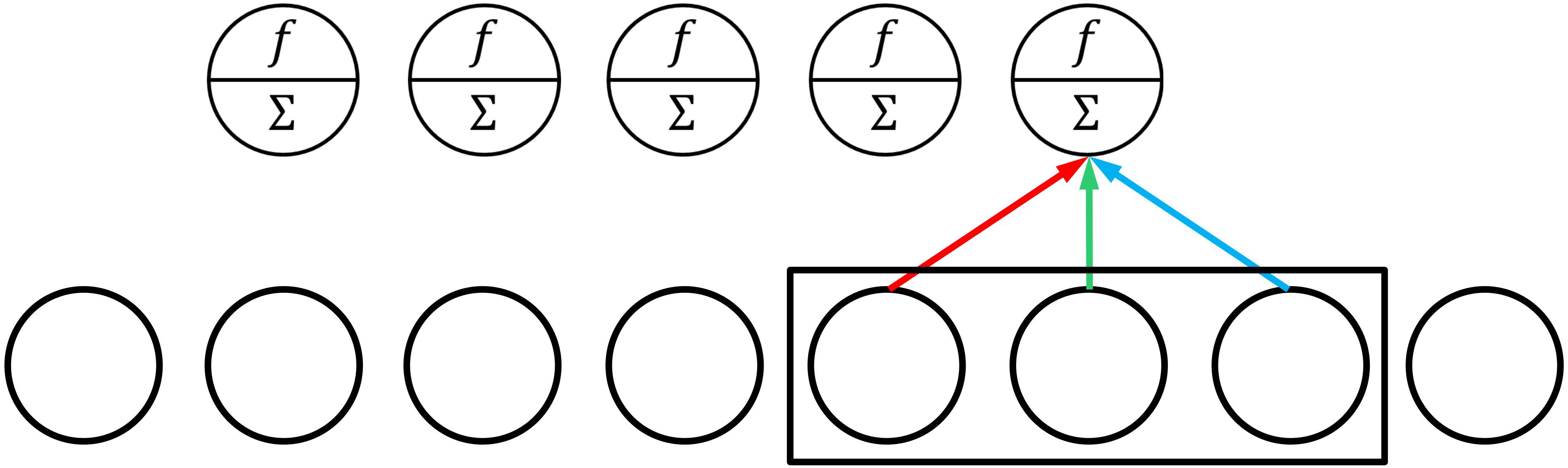
Convolutions create layers which are sparsely connected

On top of that they force equality constraints among weights in that layer

Doing so efficiently and cleverly so that we don't land up with a nasty constrained optimization problem is the key to the success of CNNs

$\mathbf{h}_i = f(\mathbf{x}_i \cdot \mathbf{w}_1 + \mathbf{x}_{i+1} \cdot \mathbf{w}_2 + \mathbf{x}_{i+2} \cdot \mathbf{w}_3)$ *i.e.* $\mathbf{h}_i = f\left(\sum_{j=1}^{3} \mathbf{x}_{i+j-1} \cdot \mathbf{w}_j\right)$

Convolution Operation



Convolutions are at the heart of signal processing and CNNs

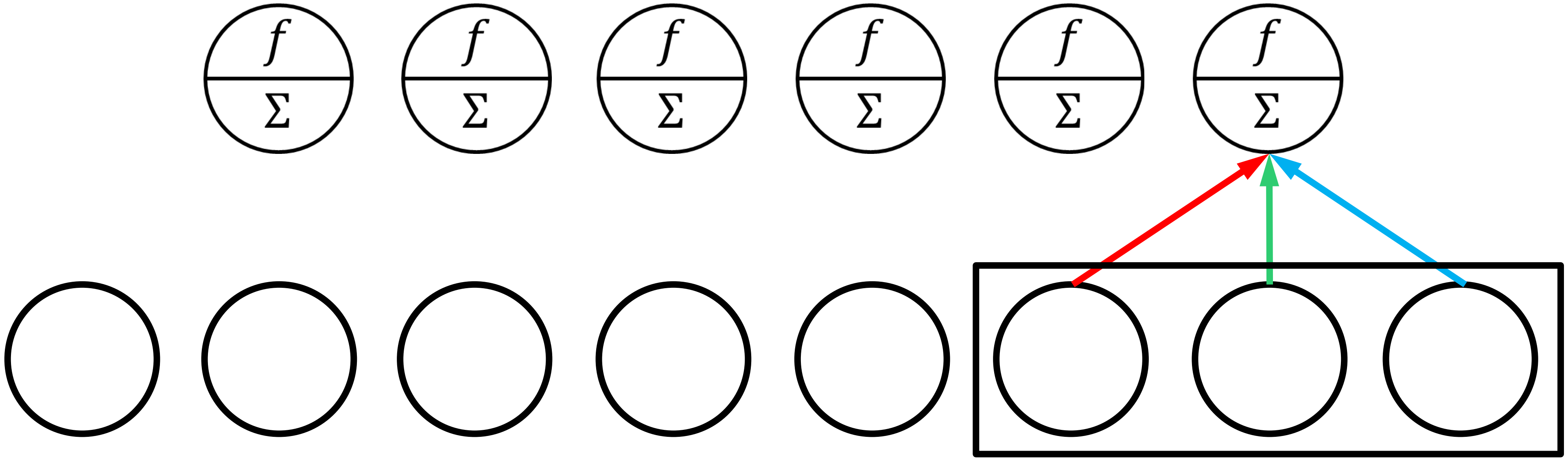
Convolutions create layers which are sparsely connected

On top of that they force equality constraints among weights in that layer

Doing so efficiently and cleverly so that we don't land up with a nasty constrained optimization problem is the key to the success of CNNs

$\mathbf{h}_i = f(\mathbf{x}_i \cdot \mathbf{w}_1 + \mathbf{x}_{i+1} \cdot \mathbf{w}_2 + \mathbf{x}_{i+2} \cdot \mathbf{w}_3)$ *i.e.* $\mathbf{h}_i = f\left(\sum_{j=1}^{3} \mathbf{x}_{i+j-1} \cdot \mathbf{w}_j\right)$

Convolution Operation



Convolutions are at the heart of signal processing and CNNs

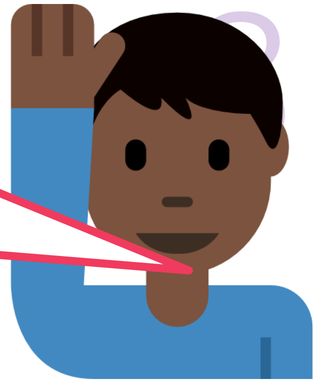
Convolutions create layers which are sparsely connected

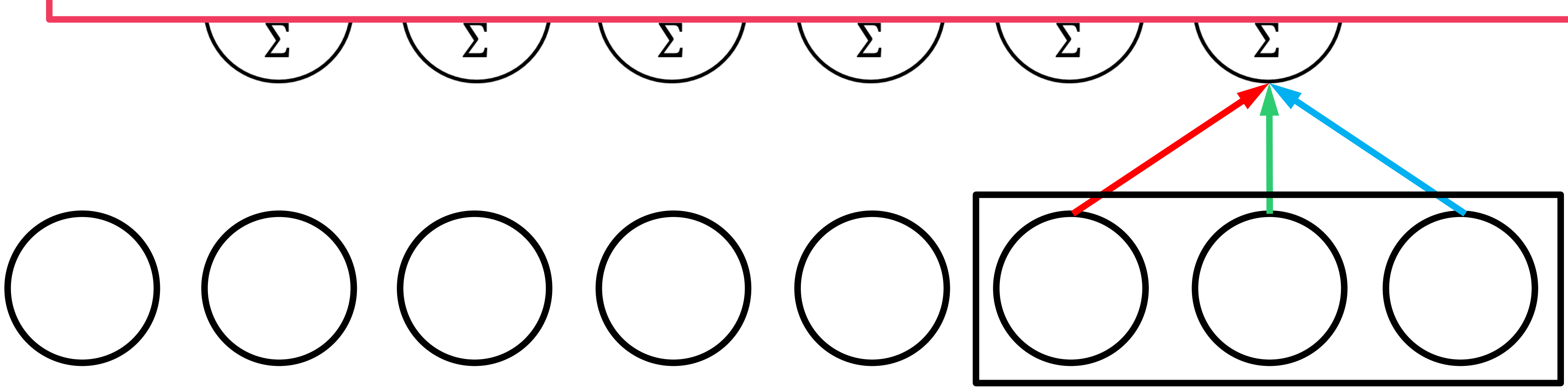
On top of that they force equality constraints among weights in that layer

Doing so efficiently and cleverly so that we don't land up with a nasty constrained optimization problem is the key to the success of CNNs

$\mathbf{h}_i = f(\mathbf{x}_i \cdot \mathbf{w}_1 + \mathbf{x}_{i+1} \cdot \mathbf{w}_2 + \mathbf{x}_{i+2} \cdot \mathbf{w}_3)$ *i.e.* $\mathbf{h}_i = f\left(\sum_{j=1}^{3} \mathbf{x}_{i+j-1} \cdot \mathbf{w}_j\right)$

The formal name for such an operation which sweeps a function (in this case $f(\mathbf{w}^\top \cdot)$ across a signal or an array is *convolution*. The vector w is often called the *kernel* of the convolution – don't confuse this with Mercer kernels though!





Convolutions are at the heart of signal processing and CNNs

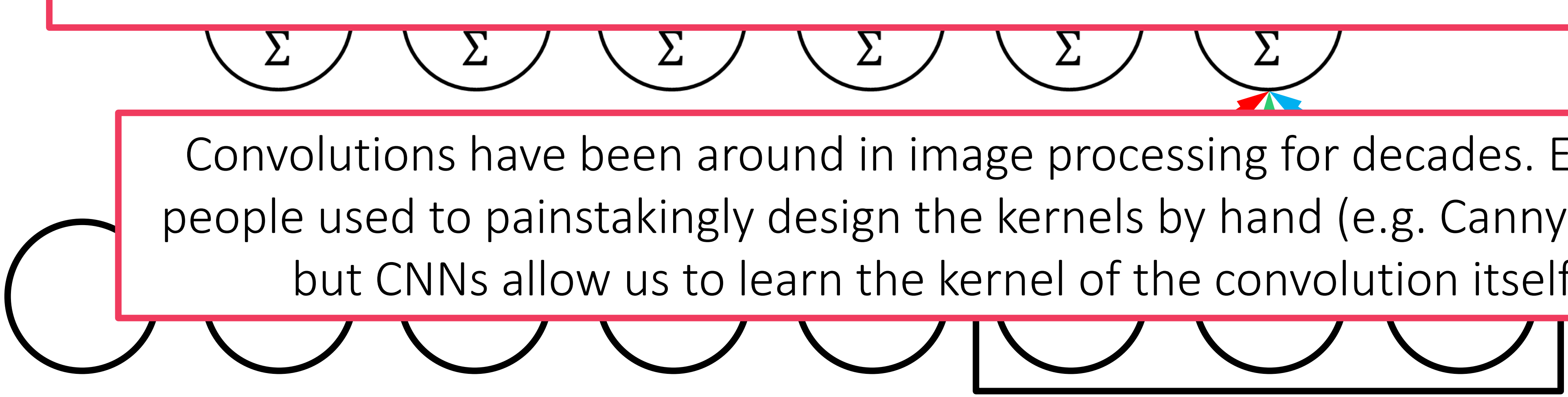
Convolutions create layers which are sparsely connected

On top of that they force equality constraints among weights in that layer

Doing so efficiently and cleverly so that we don't land up with a nasty constrained optimization problem is the key to the success of CNNs

$$\mathbf{h}_i = f(\mathbf{x}_i \cdot \mathbf{w}_1 + \mathbf{x}_{i+1} \cdot \mathbf{w}_2 + \mathbf{x}_{i+2} \cdot \mathbf{w}_3) \text{ i.e. } \mathbf{h}_i = f\left(\sum_{j=1}^{3} \mathbf{x}_{i+j-1} \cdot \mathbf{w}_j\right)$$

The formal name for such an operation which sweeps a function (in this case $f(\mathbf{w}^\top \cdot)$) across a signal or an array is *convolution*. The vector w is often called the *kernel* of the convolution – don’t confuse this with Mercer kernels though!



Convolutions have been around in image processing for decades. Earlier, people used to painstakingly design the kernels by hand (e.g. Canny filters) but CNNs allow us to learn the kernel of the convolution itself.

Convolutions are at the heart of signal processing and CNNs

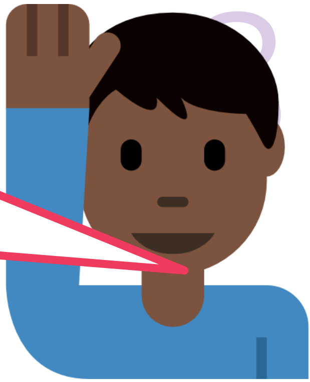
Convolutions create layers which are sparsely connected

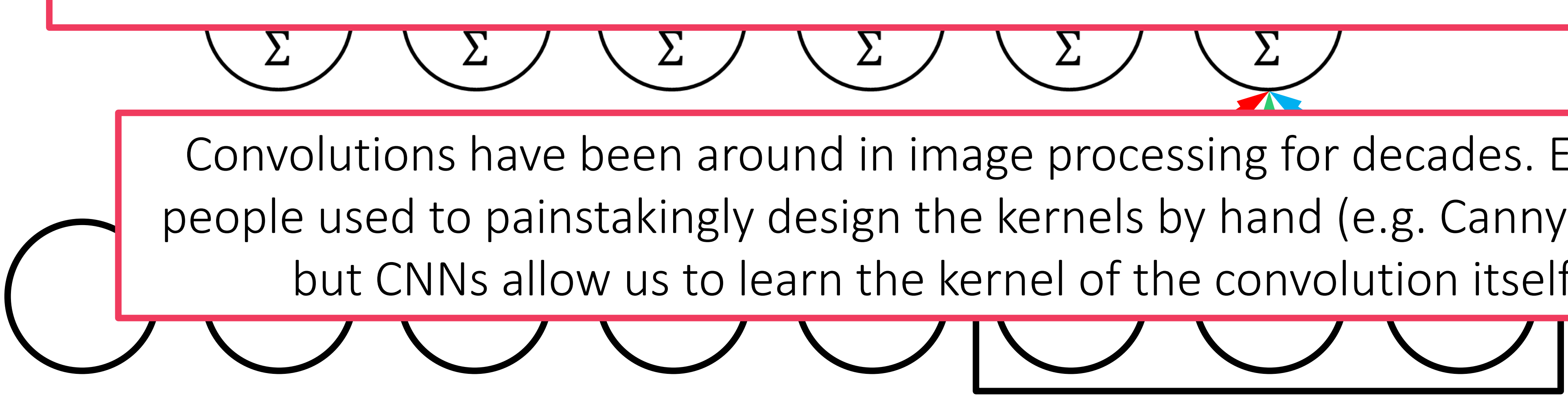
On top of that they force equality constraints among weights in that layer

Doing so efficiently and cleverly so that we don’t land up with a nasty constrained optimization problem is the key to the success of CNNs

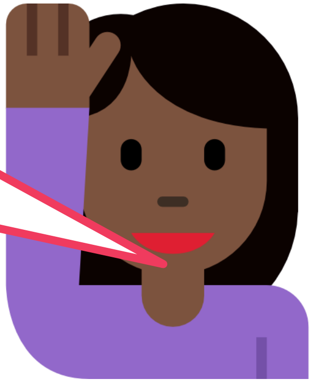
$$\mathbf{h}_i = f(\mathbf{x}_i \cdot \mathbf{w}_1 + \mathbf{x}_{i+1} \cdot \mathbf{w}_2 + \mathbf{x}_{i+2} \cdot \mathbf{w}_3) \text{ i.e. } \mathbf{h}_i = f\left(\sum_{j=1}^{3} \mathbf{x}_{i+j-1} \cdot \mathbf{w}_j\right)$$

The formal name for such an operation which sweeps a function (in this case $f(\mathbf{w}^\top \cdot)$ across a signal or an array is *convolution*. The vector w is often called the *kernel* of the convolution – don't confuse this with Mercer kernels though!



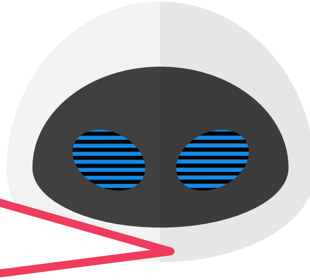


Convolutions have been around in image processing for decades. Earlier, people used to painstakingly design the kernels by hand (e.g. Canny filters) but CNNs allow us to learn the kernel of the convolution itself.



Conv

True, the idea is to allow difficult problems to be solved with a customized filter that works best for that problem. However, handcrafted filters may still offer good results for simple problems (and offer faster training since learning filters is not an easy task – the backprop becomes complicated)



Co

Or

Do

constrained optimization problem is the key to the success of CNNs

$$\mathbf{h}_i = f(\mathbf{x}_i \cdot \mathbf{w}_1 + \mathbf{x}_{i+1} \cdot \mathbf{w}_2 + \mathbf{x}_{i+2} \cdot \mathbf{w}_3) \text{ i.e. } \mathbf{h}_i = f\left(\sum_{j=1}^{3} \mathbf{x}_{i+j-1} \cdot \mathbf{w}_j\right)$$

2D Convolutions

Convolutions make sense for 2D, 3D, nD data as well

A fully conn. layer would've needed 576 weights. A conv. needs only 9 weights

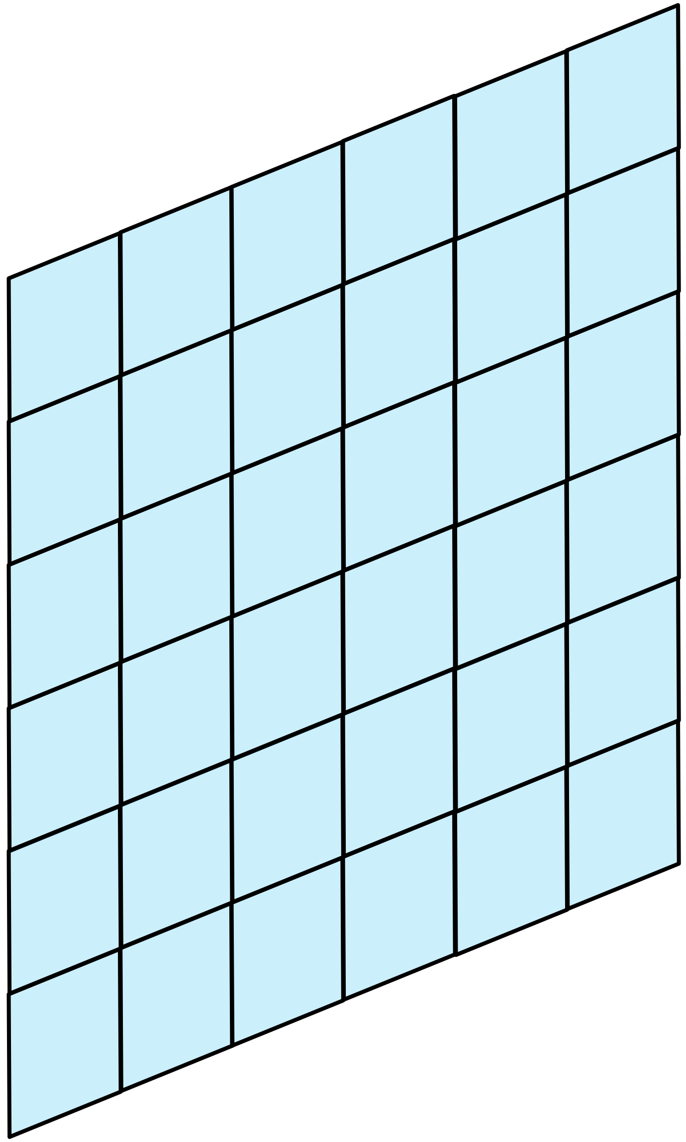
Such local operations are exactly what we need for detecting local patterns

Edges, boundaries, textures

Can apply convolutions to 3D layers as well

E.g. Video data is 3D

2D Convolutions



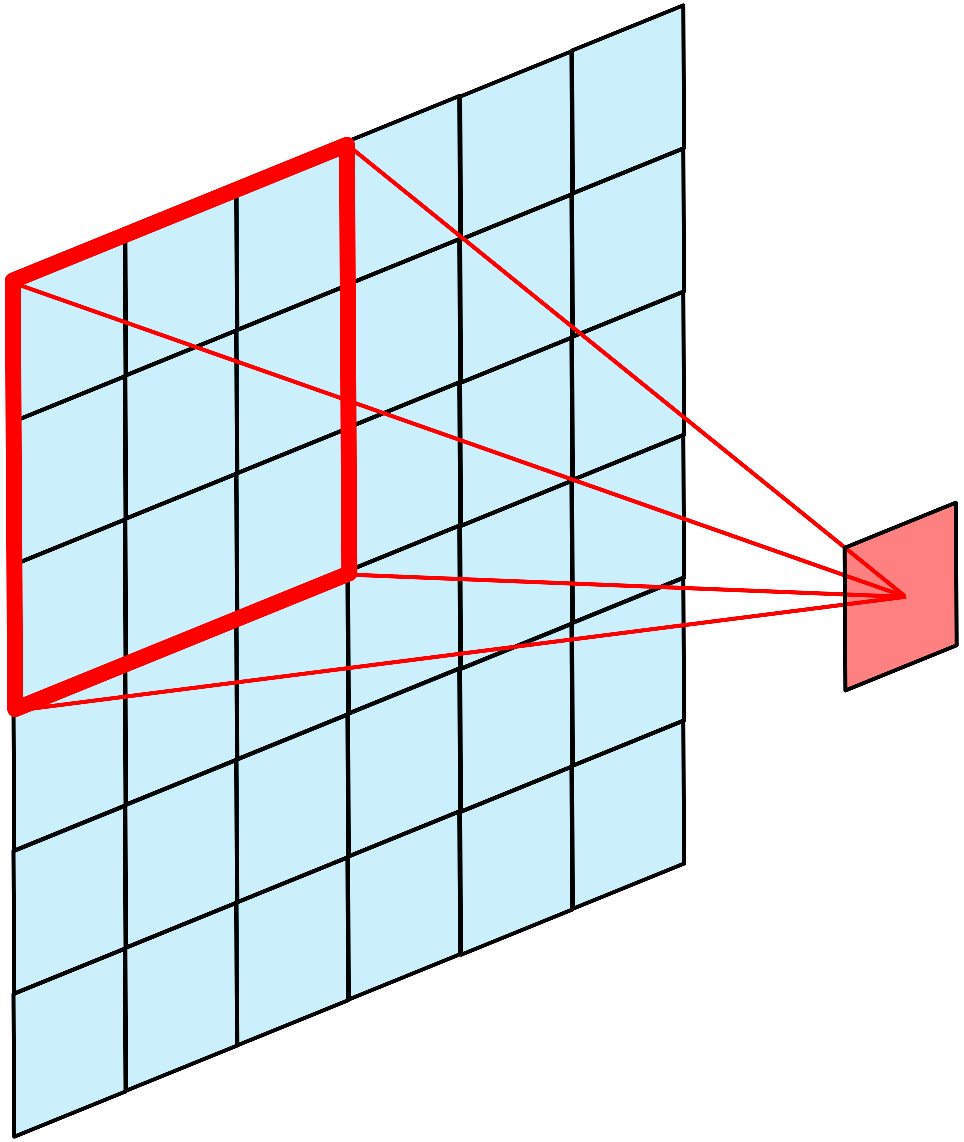
Convolutions make sense for 2D, 3D, nD data as well

- *A fully conn. layer would've needed 576 weights. A conv. needs only 9 weights*

Such local operations are exactly what we need for detecting local patterns

- *Edges, boundaries, textures*
- *Can apply convolutions to 3D layers as well*
- *E.g. Video data is 3D*

2D Convolutions



Convolutions make sense for 2D, 3D, nD data as well

A fully conn. layer would've needed 576 weights. A conv. needs only 9 weights

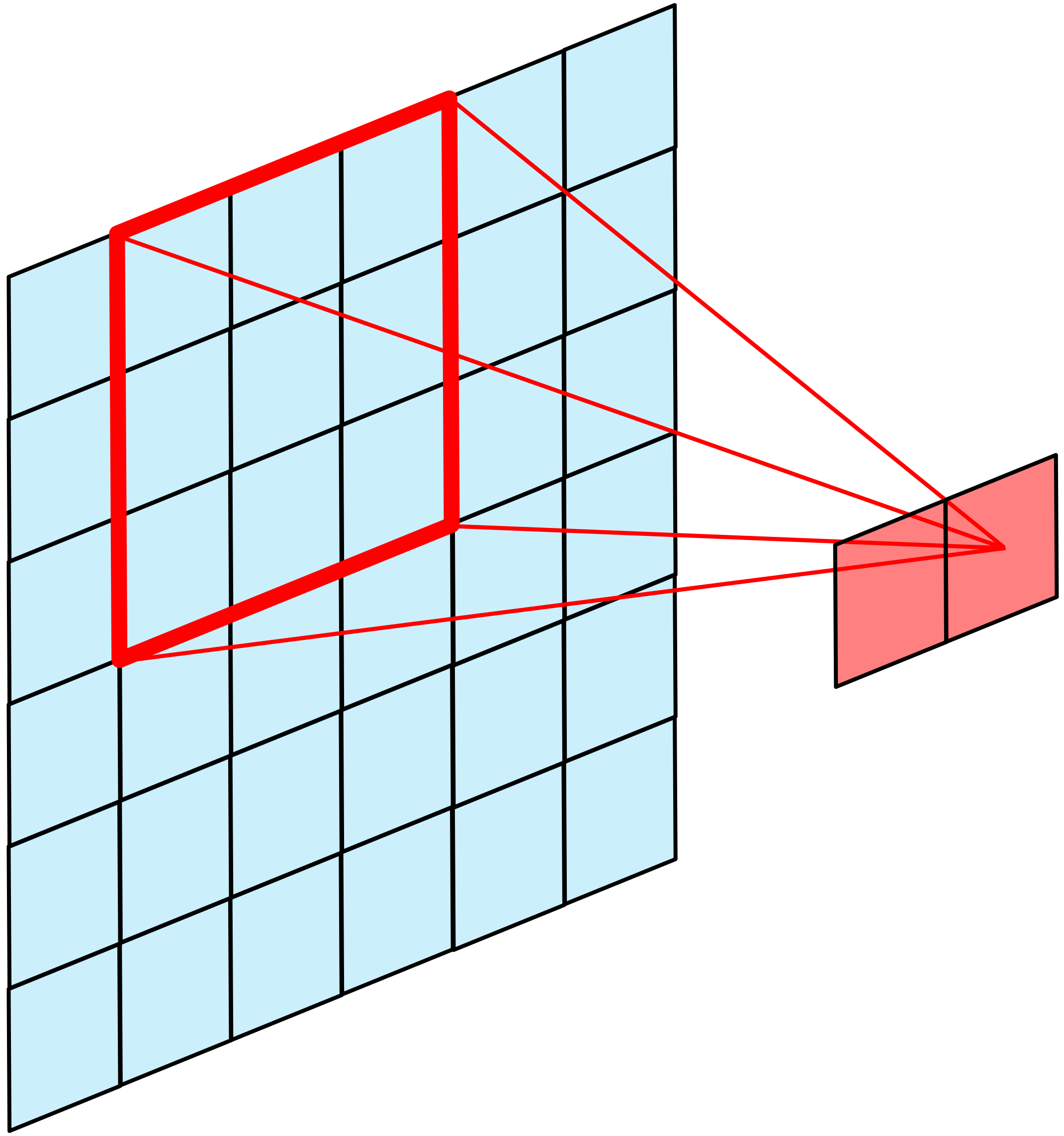
Such local operations are exactly what we need for detecting local patterns

Edges, boundaries, textures

Can apply convolutions to 3D layers as well

E.g. Video data is 3D

2D Convolutions



Convolutions make sense for 2D, 3D, nD data as well

A fully conn. layer would've needed 576 weights. A conv. needs only 9 weights

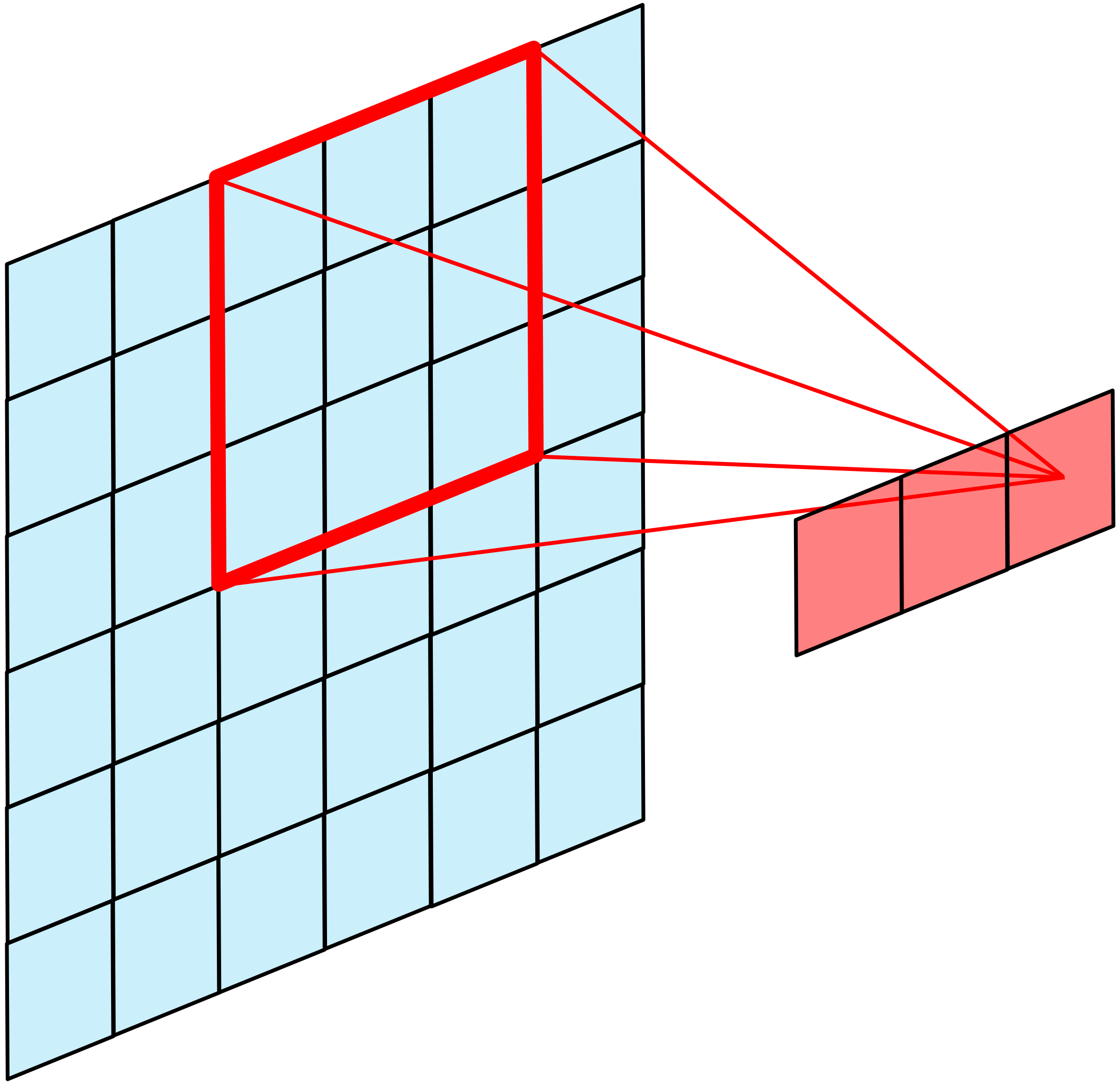
Such local operations are exactly what we need for detecting local patterns

Edges, boundaries, textures

Can apply convolutions to 3D layers as well

E.g. Video data is 3D

2D Convolutions



Convolutions make sense for 2D, 3D, nD data as well

A fully conn. layer would've needed 576 weights. A conv. needs only 9 weights

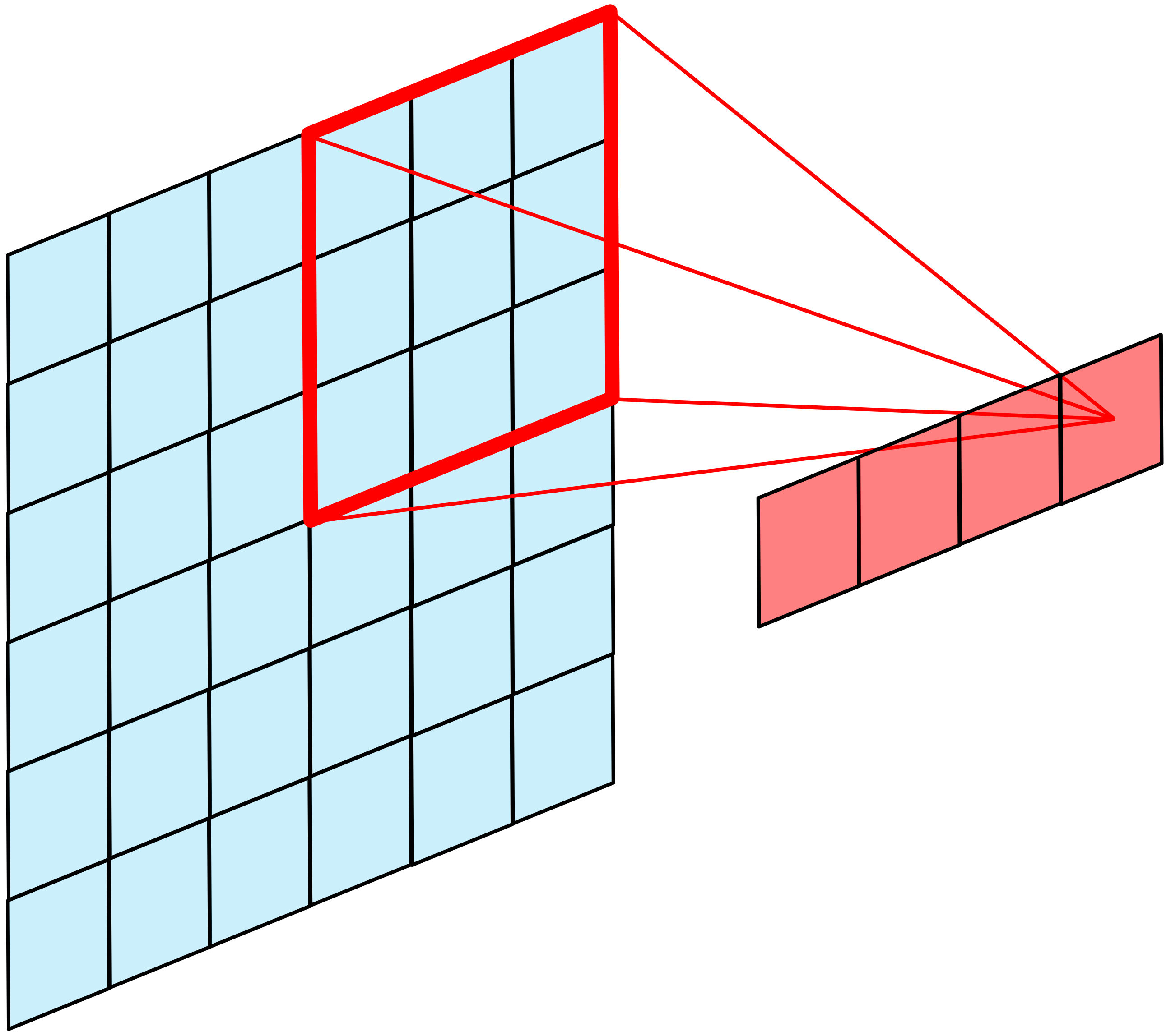
Such local operations are exactly what we need for detecting local patterns

Edges, boundaries, textures

Can apply convolutions to 3D layers as well

E.g. Video data is 3D

2D Convolutions



Convolutions make sense for 2D, 3D, nD data as well

A fully conn. layer would've needed 576 weights. A conv. needs only 9 weights

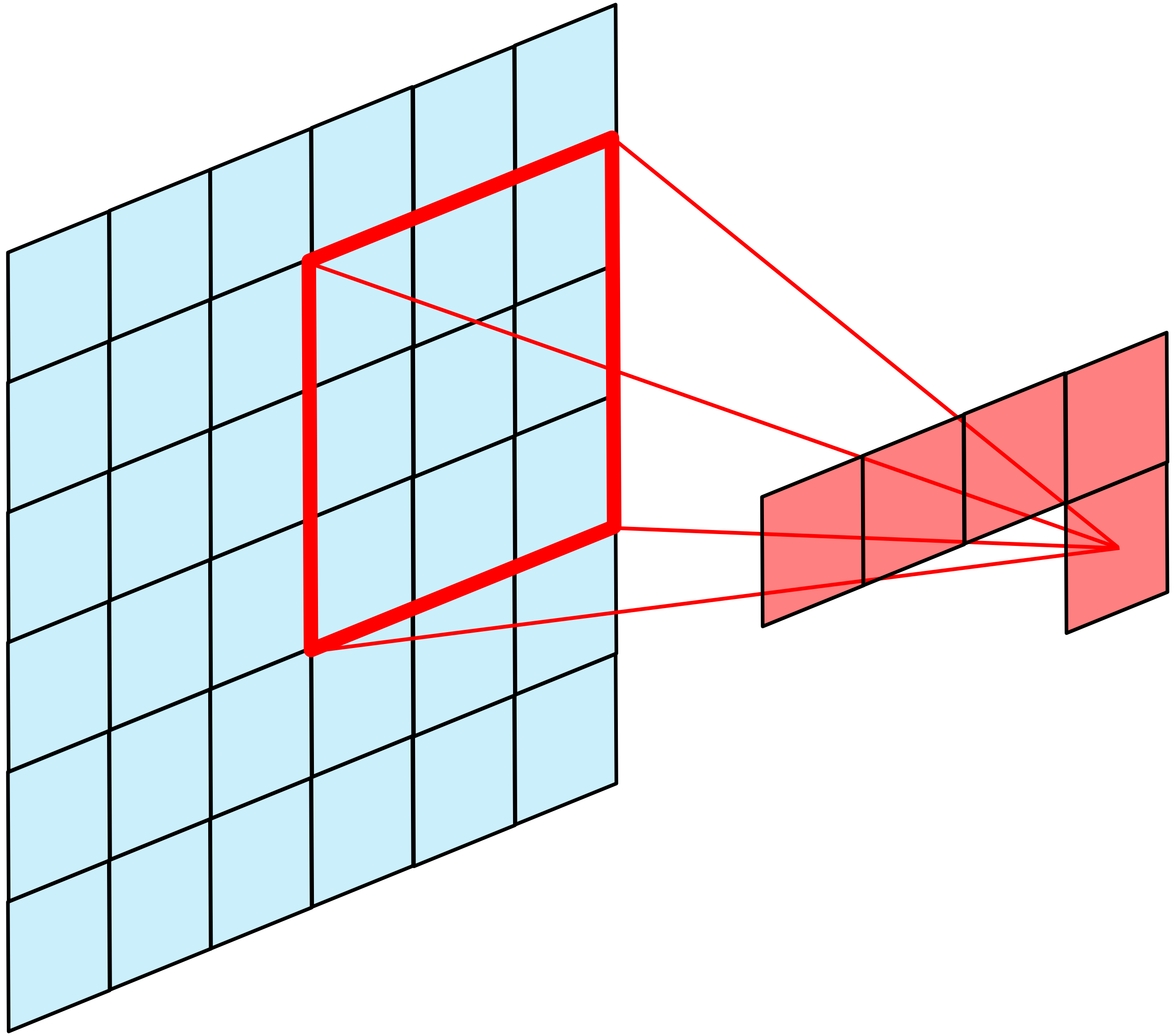
Such local operations are exactly what we need for detecting local patterns

Edges, boundaries, textures

Can apply convolutions to 3D layers as well

E.g. Video data is 3D

2D Convolutions



Convolutions make sense for 2D, 3D, nD data as well

A fully conn. layer would've needed 576 weights. A conv. needs only 9 weights

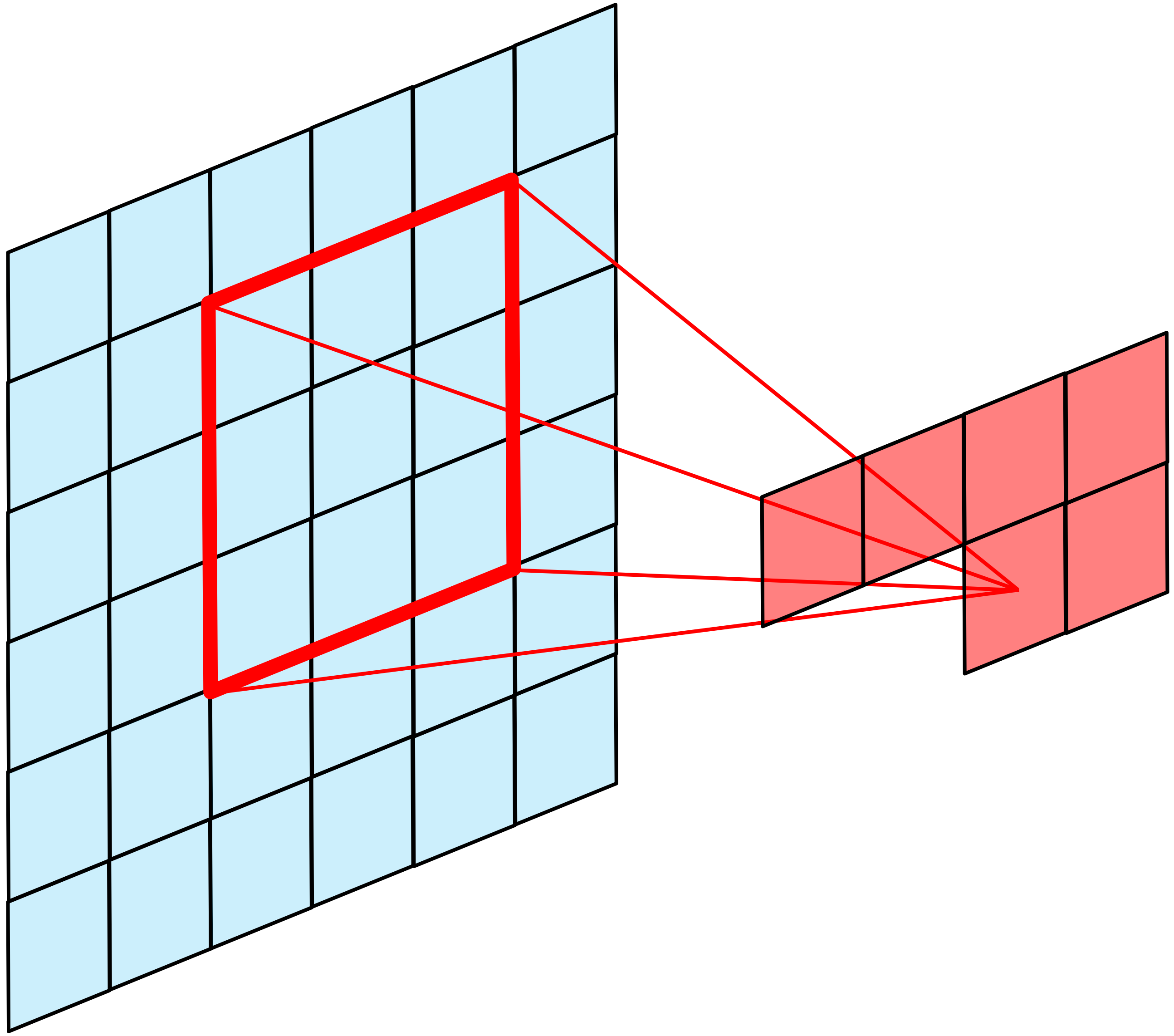
Such local operations are exactly what we need for detecting local patterns

Edges, boundaries, textures

Can apply convolutions to 3D layers as well

E.g. Video data is 3D

2D Convolutions



Convolutions make sense for 2D, 3D, nD data as well

A fully conn. layer would've needed 576 weights. A conv. needs only 9 weights

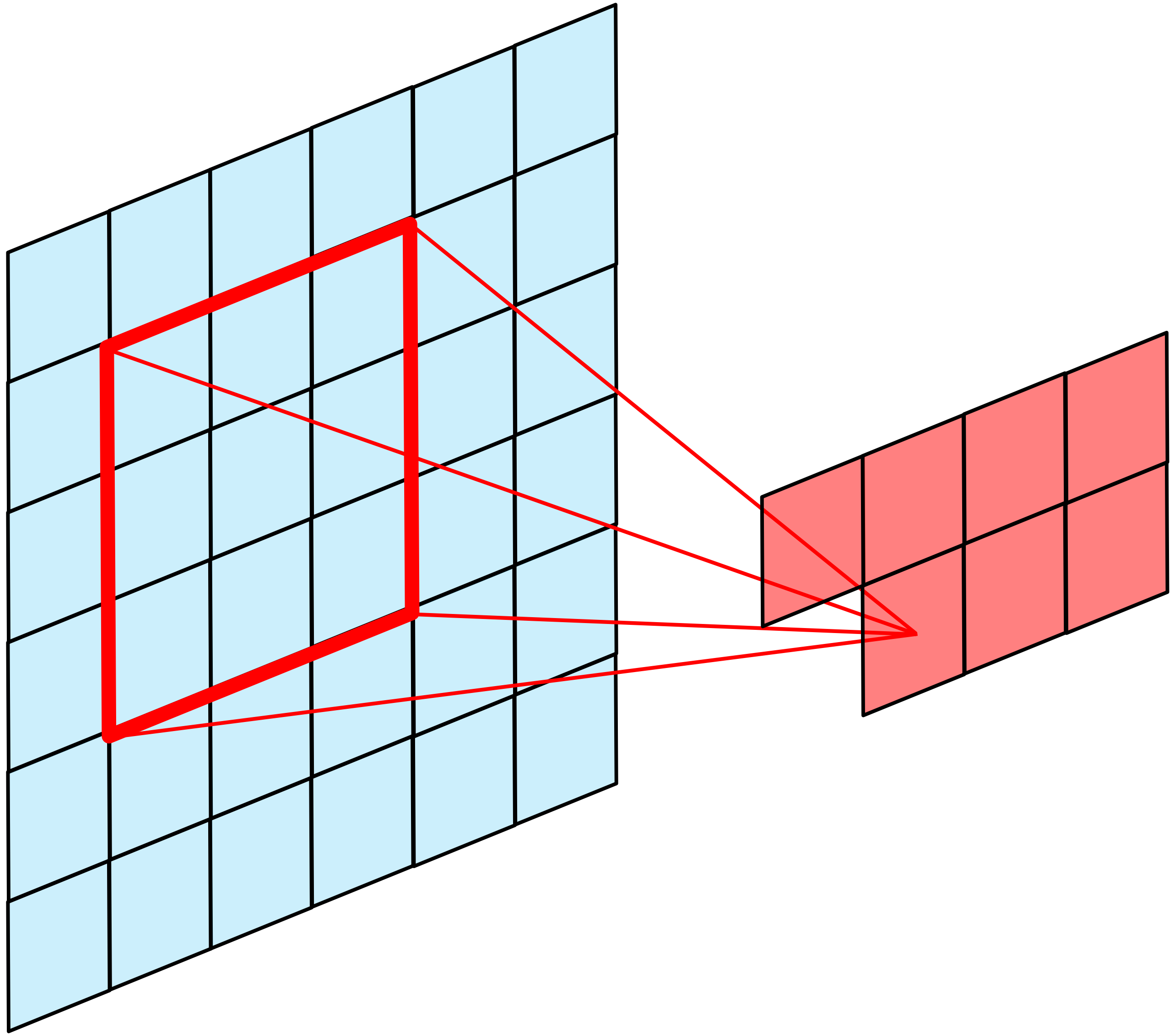
Such local operations are exactly what we need for detecting local patterns

Edges, boundaries, textures

Can apply convolutions to 3D layers as well

E.g. Video data is 3D

2D Convolutions



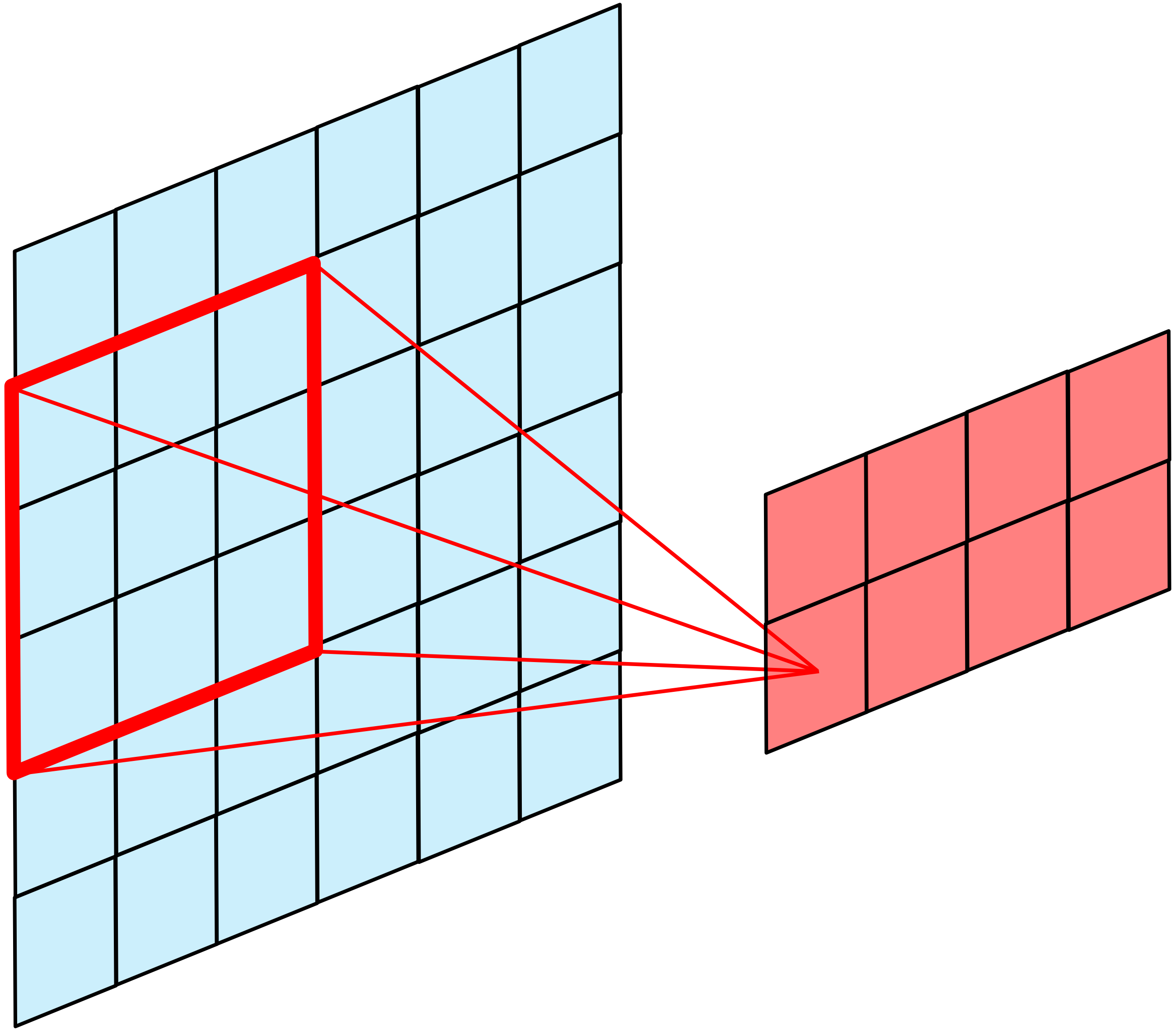
Convolutions make sense for 2D, 3D, nD data as well

- *A fully conn. layer would've needed 576 weights. A conv. needs only 9 weights*

Such local operations are exactly what we need for detecting local patterns

- *Edges, boundaries, textures*
- *Can apply convolutions to 3D layers as well*
- *E.g. Video data is 3D*

2D Convolutions



Convolutions make sense for 2D, 3D, nD data as well

A fully conn. layer would've needed 576 weights. A conv. needs only 9 weights

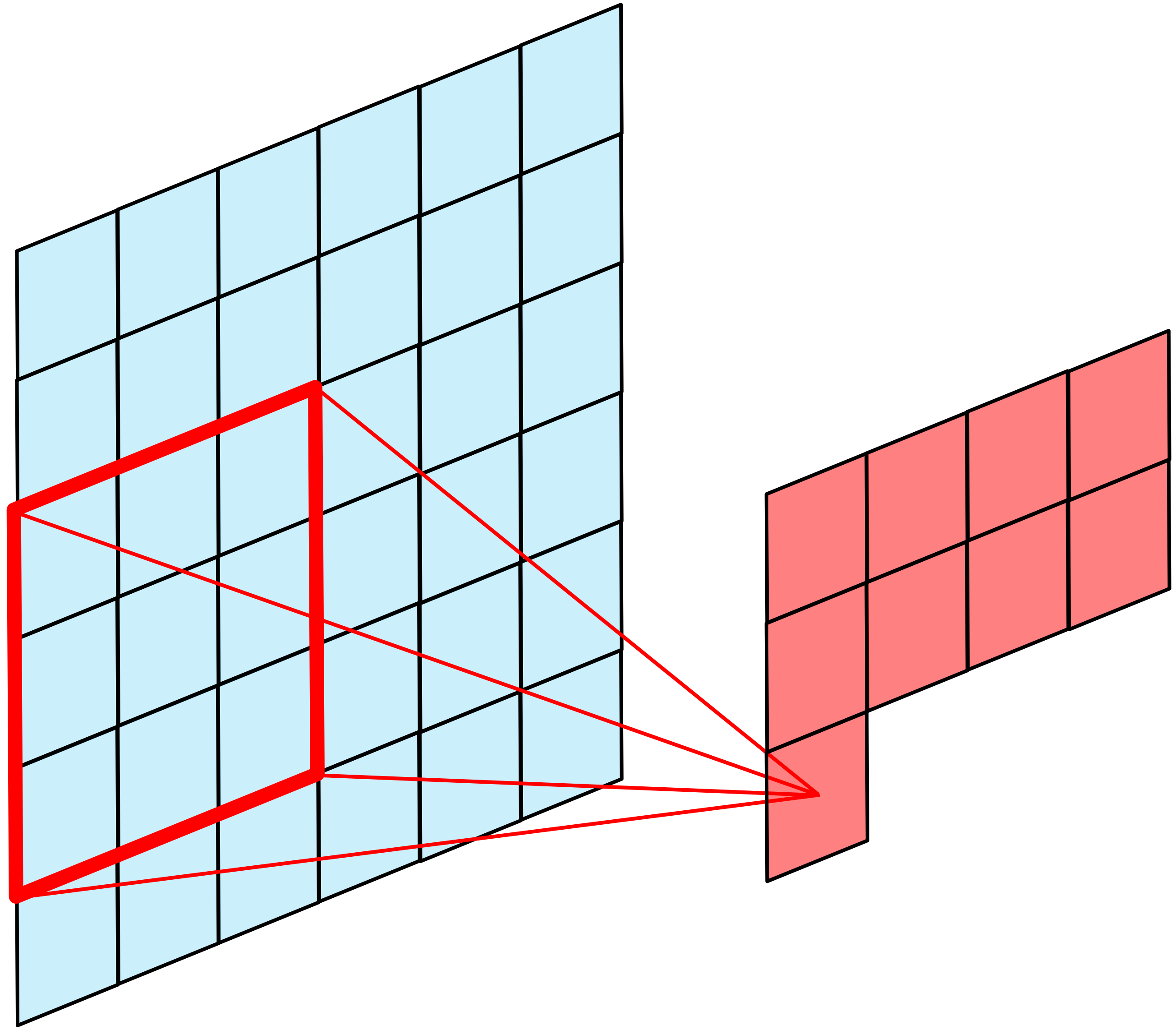
Such local operations are exactly what we need for detecting local patterns

Edges, boundaries, textures

Can apply convolutions to 3D layers as well

E.g. Video data is 3D

2D Convolutions



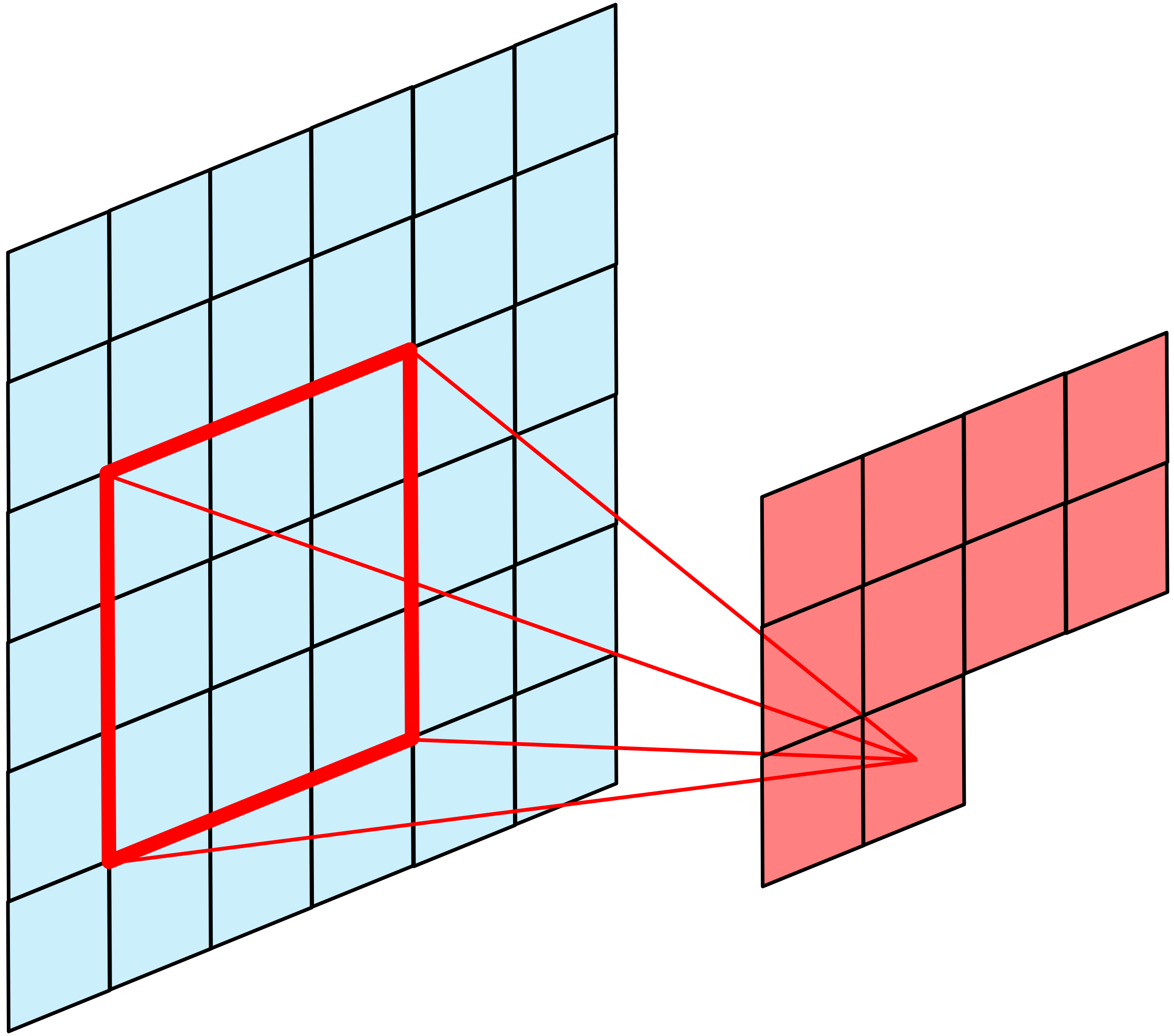
Convolutions make sense for 2D, 3D, nD data as well

- *A fully conn. layer would've needed 576 weights. A conv. needs only 9 weights*

Such local operations are exactly what we need for detecting local patterns

- *Edges, boundaries, textures*
- *Can apply convolutions to 3D layers as well*
- *E.g. Video data is 3D*

2D Convolutions



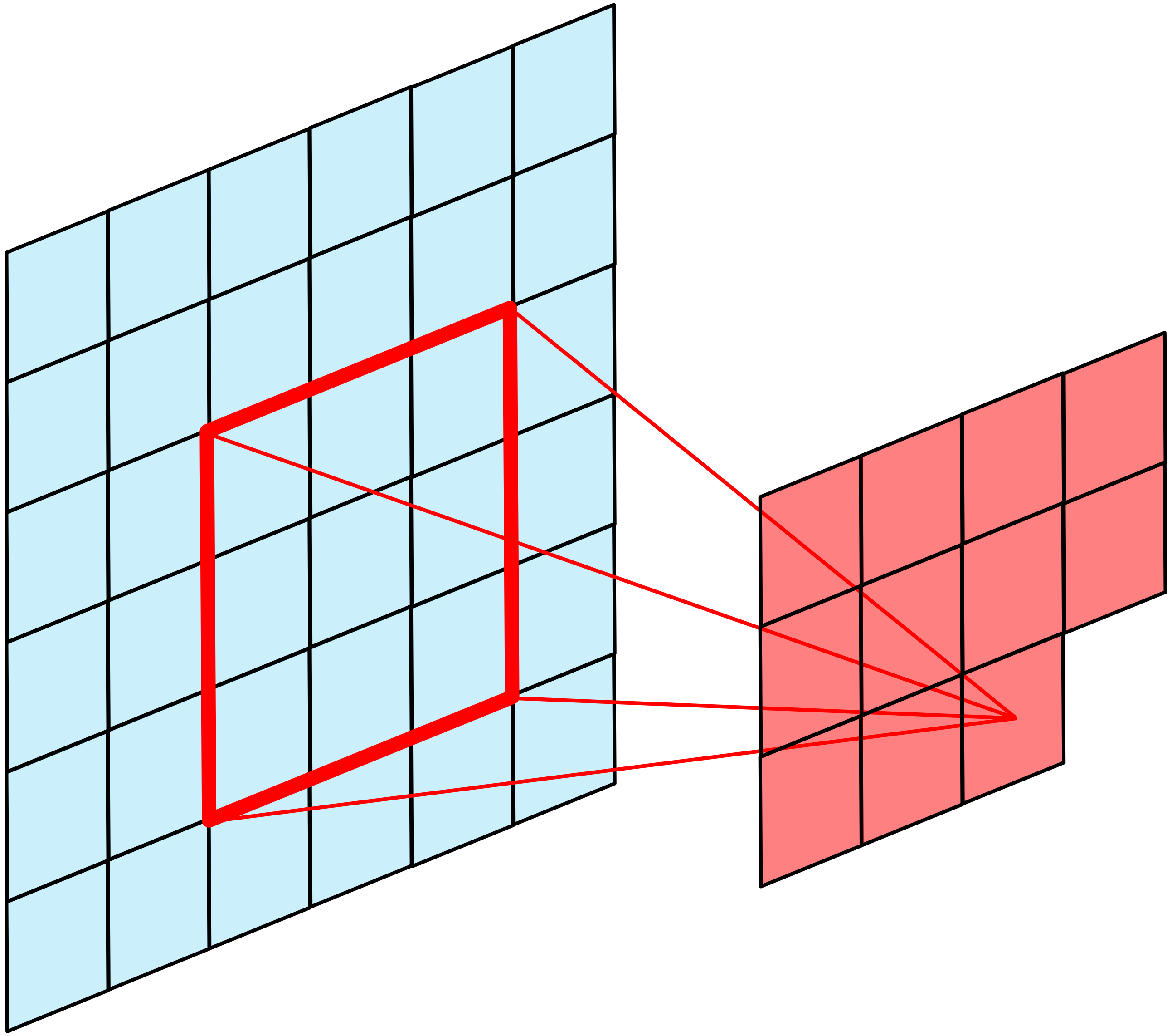
Convolutions make sense for 2D, 3D, nD data as well

- *A fully conn. layer would've needed 576 weights. A conv. needs only 9 weights*

Such local operations are exactly what we need for detecting local patterns

- *Edges, boundaries, textures*
- *Can apply convolutions to 3D layers as well*
- *E.g. Video data is 3D*

2D Convolutions



Convolutions make sense for 2D, 3D, nD data as well

A fully conn. layer would've needed 576 weights. A conv. needs only 9 weights

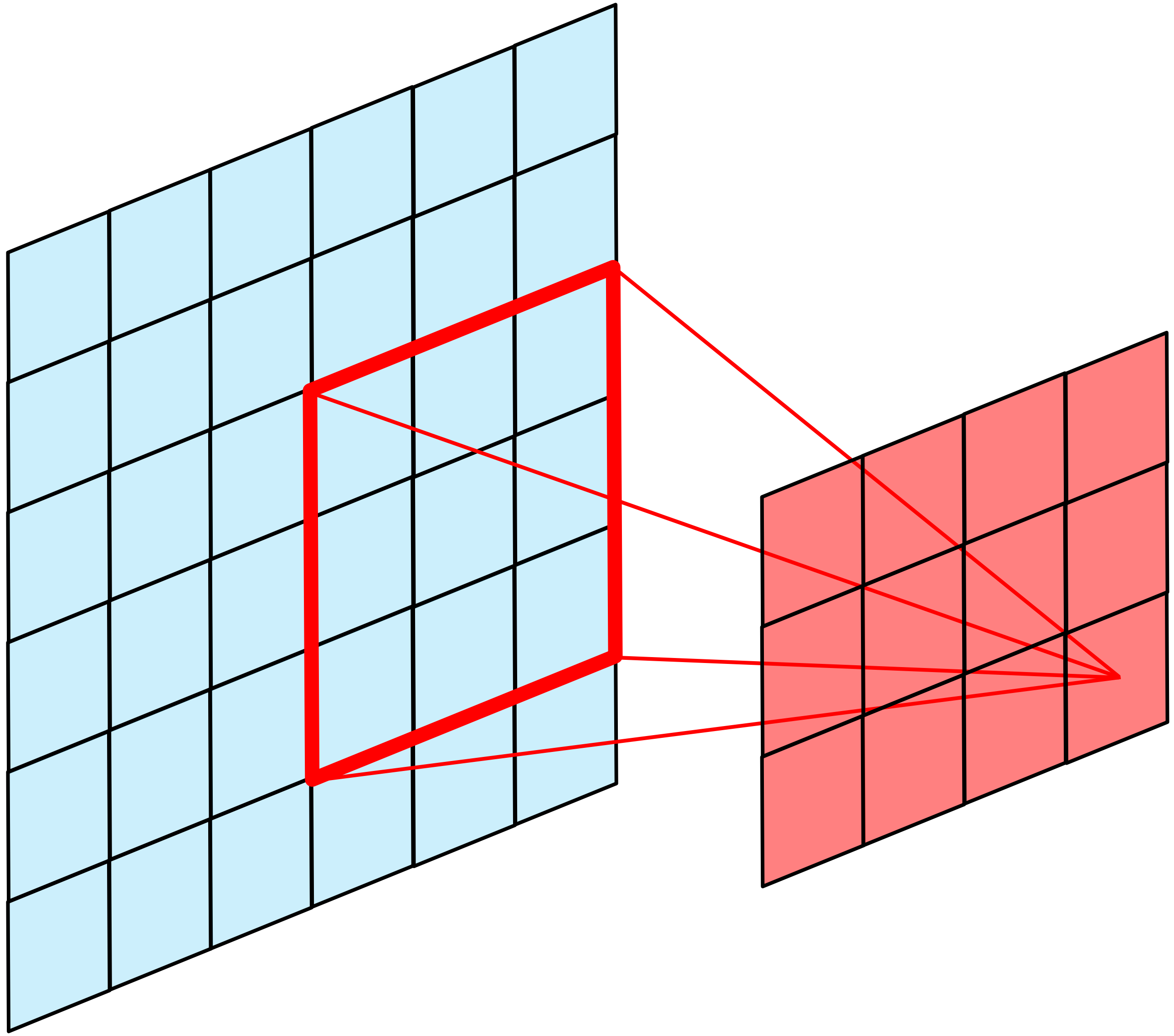
Such local operations are exactly what we need for detecting local patterns

Edges, boundaries, textures

Can apply convolutions to 3D layers as well

E.g. Video data is 3D

2D Convolutions



Convolutions make sense for 2D, 3D, nD data as well

A fully conn. layer would've needed 576 weights. A conv. needs only 9 weights

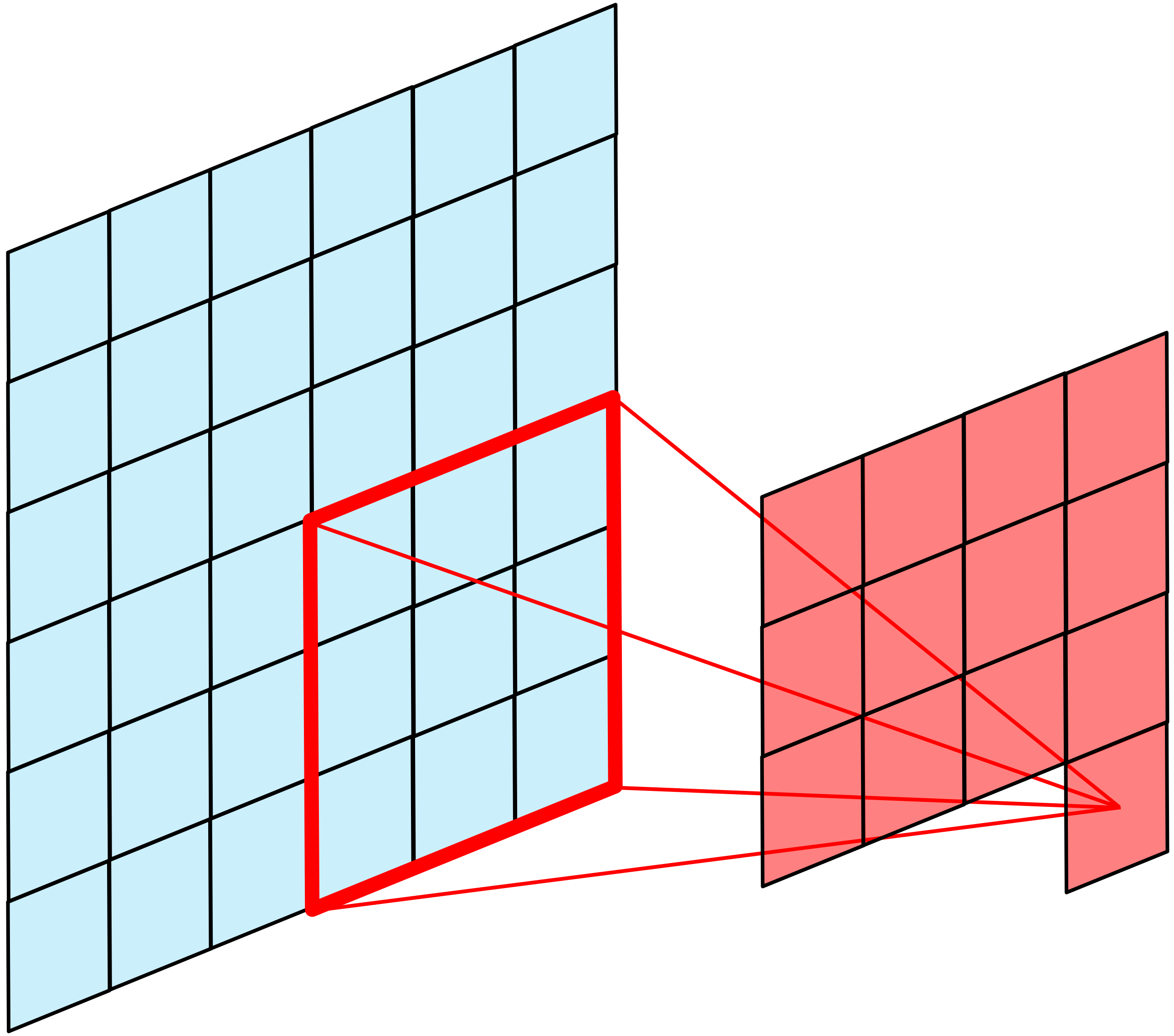
Such local operations are exactly what we need for detecting local patterns

Edges, boundaries, textures

Can apply convolutions to 3D layers as well

E.g. Video data is 3D

2D Convolutions



Convolutions make sense for 2D, 3D, nD data as well

A fully conn. layer would've needed 576 weights. A conv. needs only 9 weights

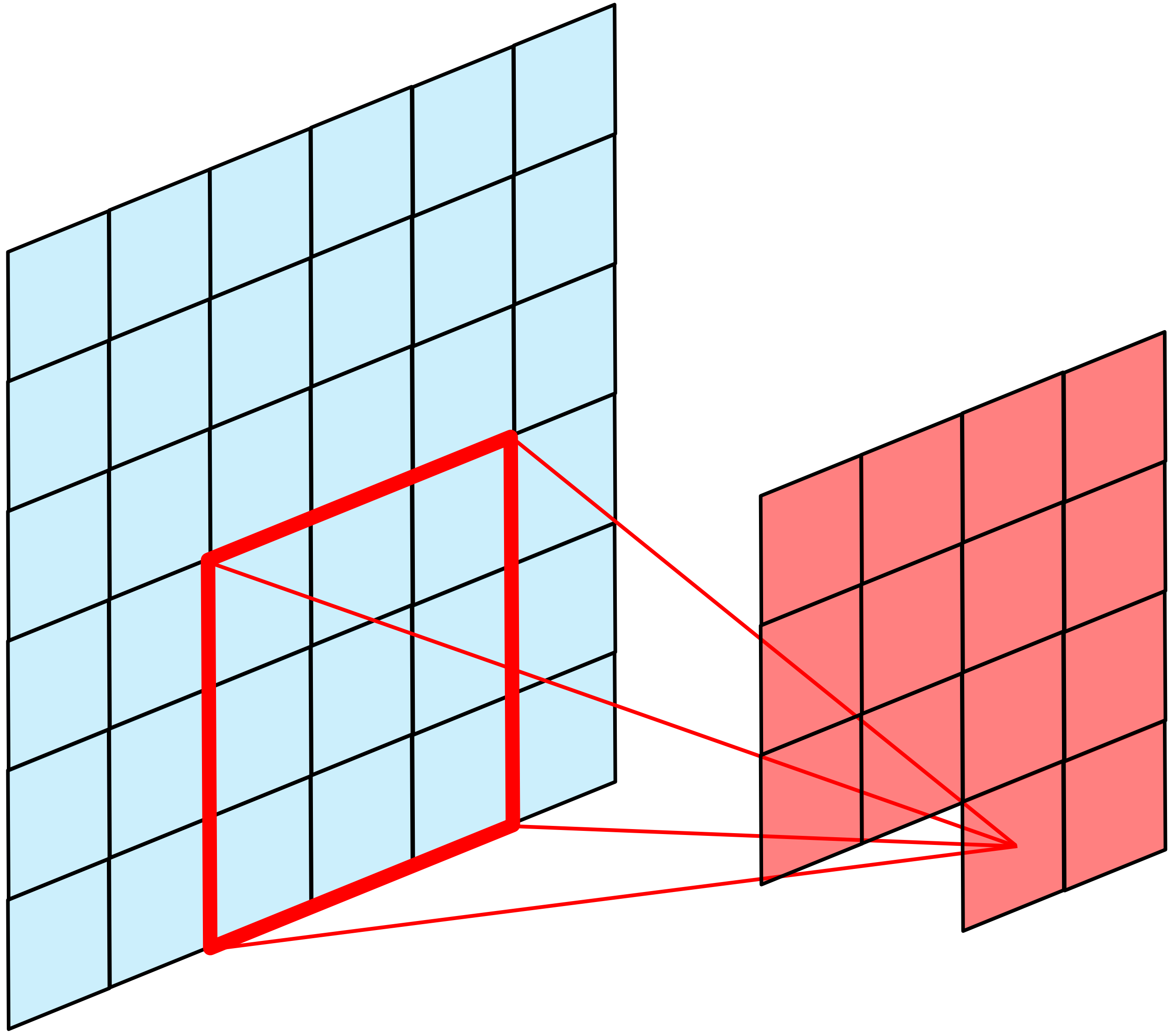
Such local operations are exactly what we need for detecting local patterns

Edges, boundaries, textures

Can apply convolutions to 3D layers as well

E.g. Video data is 3D

2D Convolutions



Convolutions make sense for 2D, 3D, nD data as well

A fully conn. layer would've needed 576 weights. A conv. needs only 9 weights

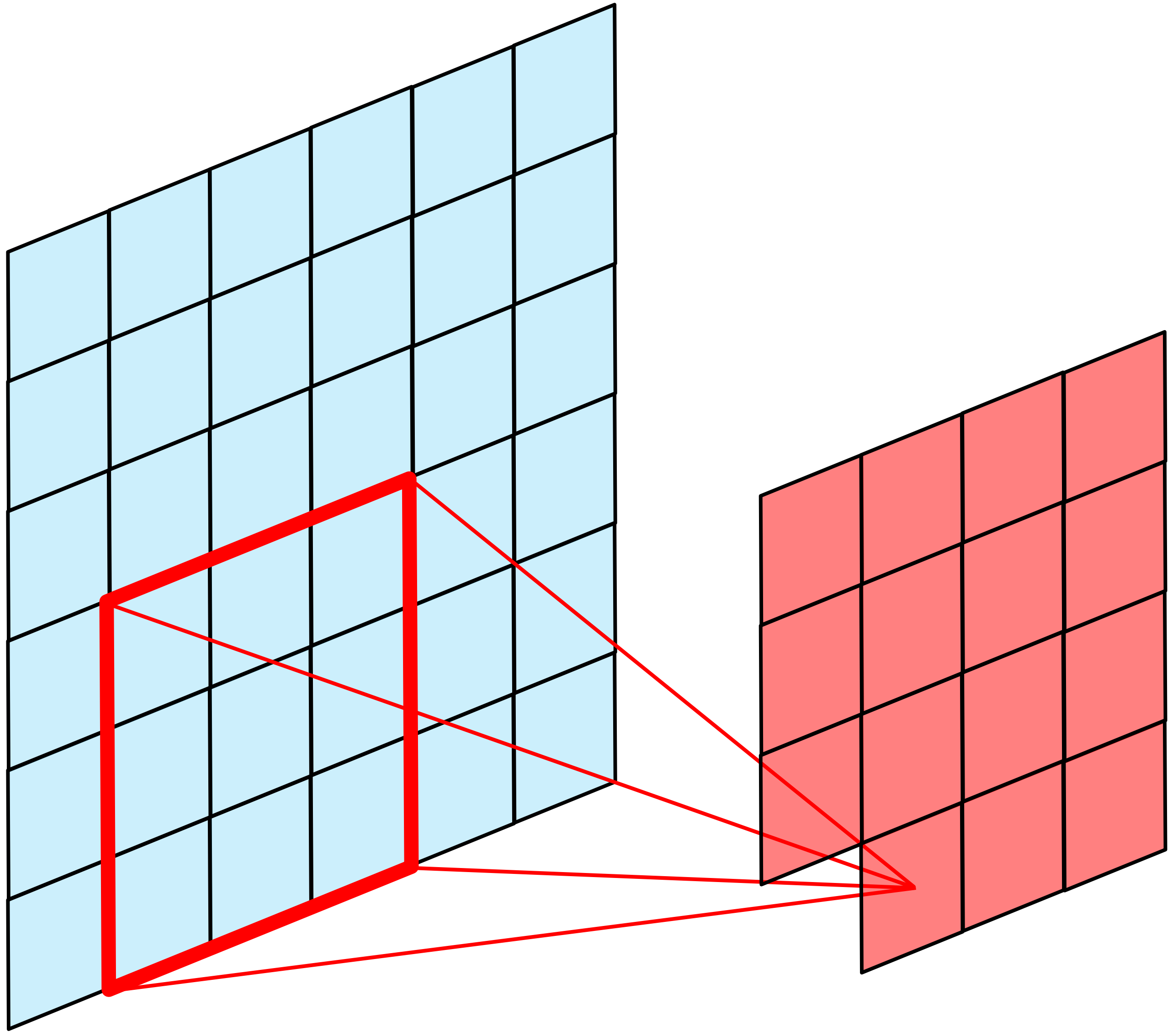
Such local operations are exactly what we need for detecting local patterns

Edges, boundaries, textures

Can apply convolutions to 3D layers as well

E.g. Video data is 3D

2D Convolutions



Convolutions make sense for 2D, 3D, nD data as well

A fully conn. layer would've needed 576 weights. A conv. needs only 9 weights

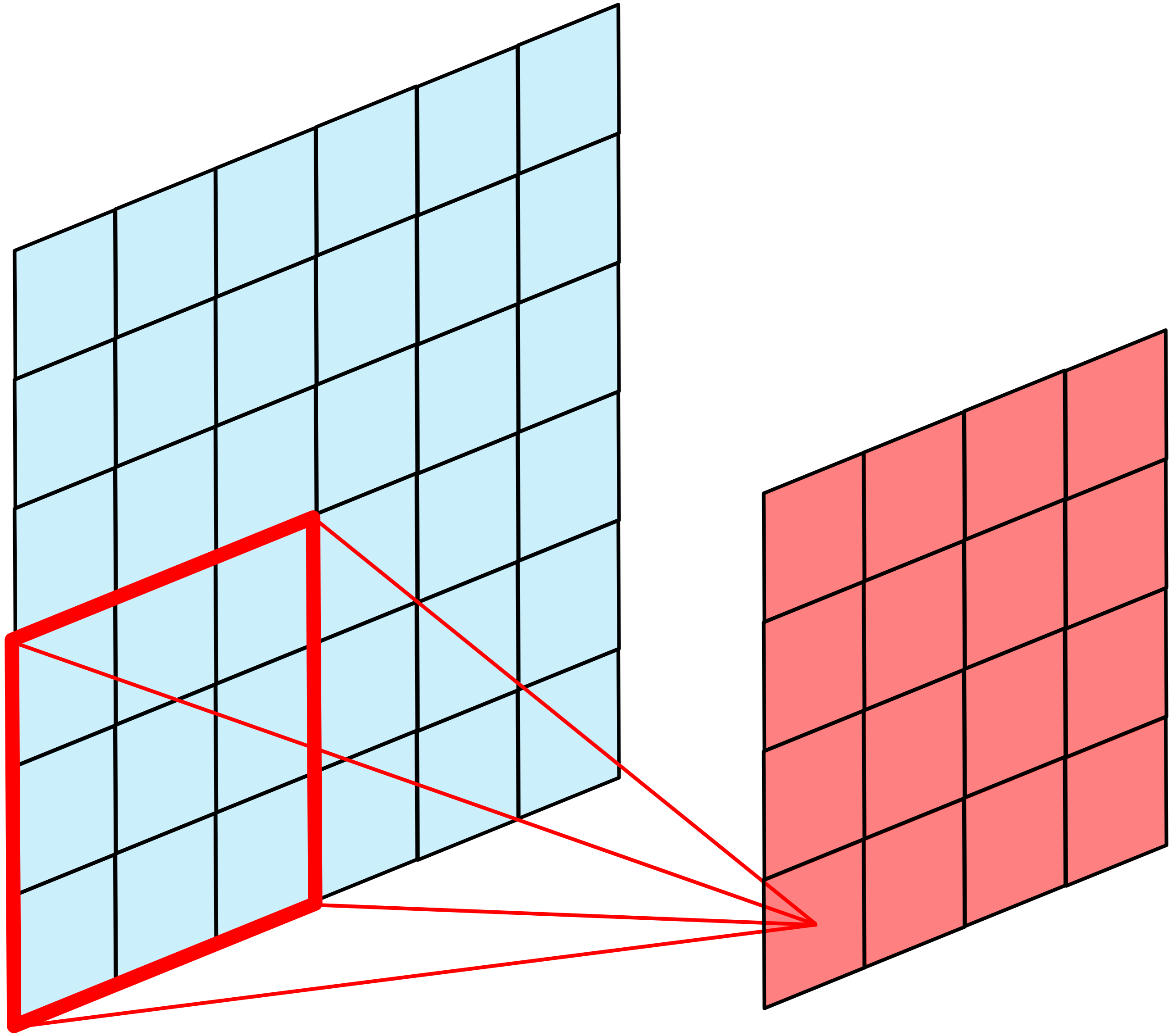
Such local operations are exactly what we need for detecting local patterns

Edges, boundaries, textures

Can apply convolutions to 3D layers as well

E.g. Video data is 3D

2D Convolutions



Convolutions make sense for 2D, 3D, nD data as well

A fully conn. layer would've needed 576 weights. A conv. needs only 9 weights

Such local operations are exactly what we need for detecting local patterns

Edges, boundaries, textures

Can apply convolutions to 3D layers as well

E.g. Video data is 3D

Convolutional Neural Network

Popular where the raw data has strong spatial structure e.g. images have 2D structure, text has linear structure, video has 3D 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, CNNs use the definition $(\mathbf{u} * \mathbf{v})_i = \sum_{j=1}^{n} \mathbf{u}_{i+j} \cdot \mathbf{v}_j$

In signal processing literature this operation is actually called cross-correlation

Pooling Operations

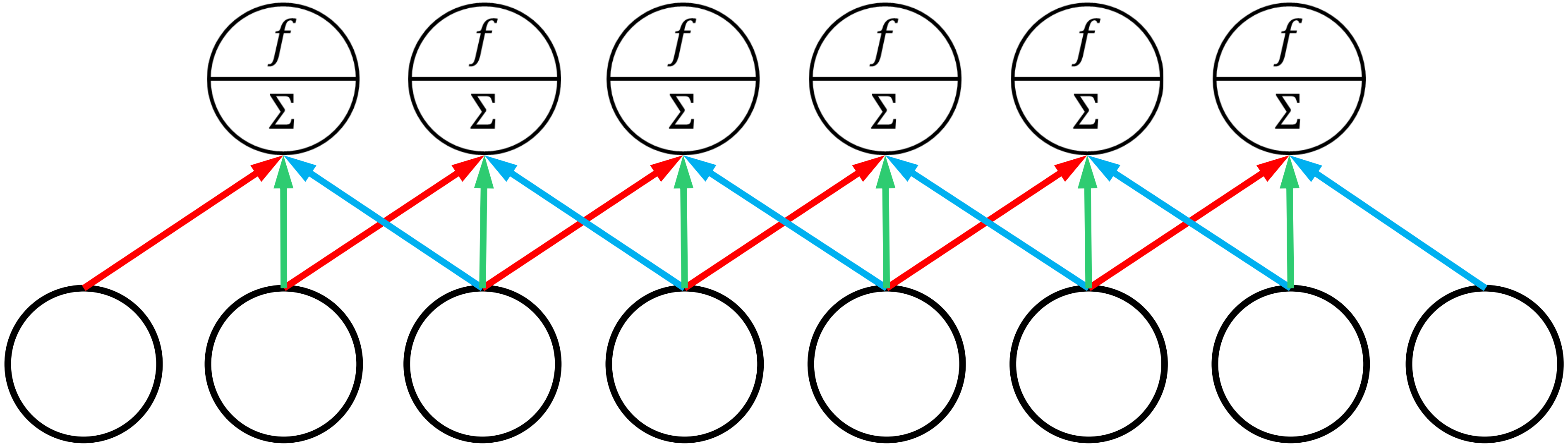
Reduce sensitivity of the network to small shifts/errors in image

Max-pooling and average pooling most common

Pooling Operations

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

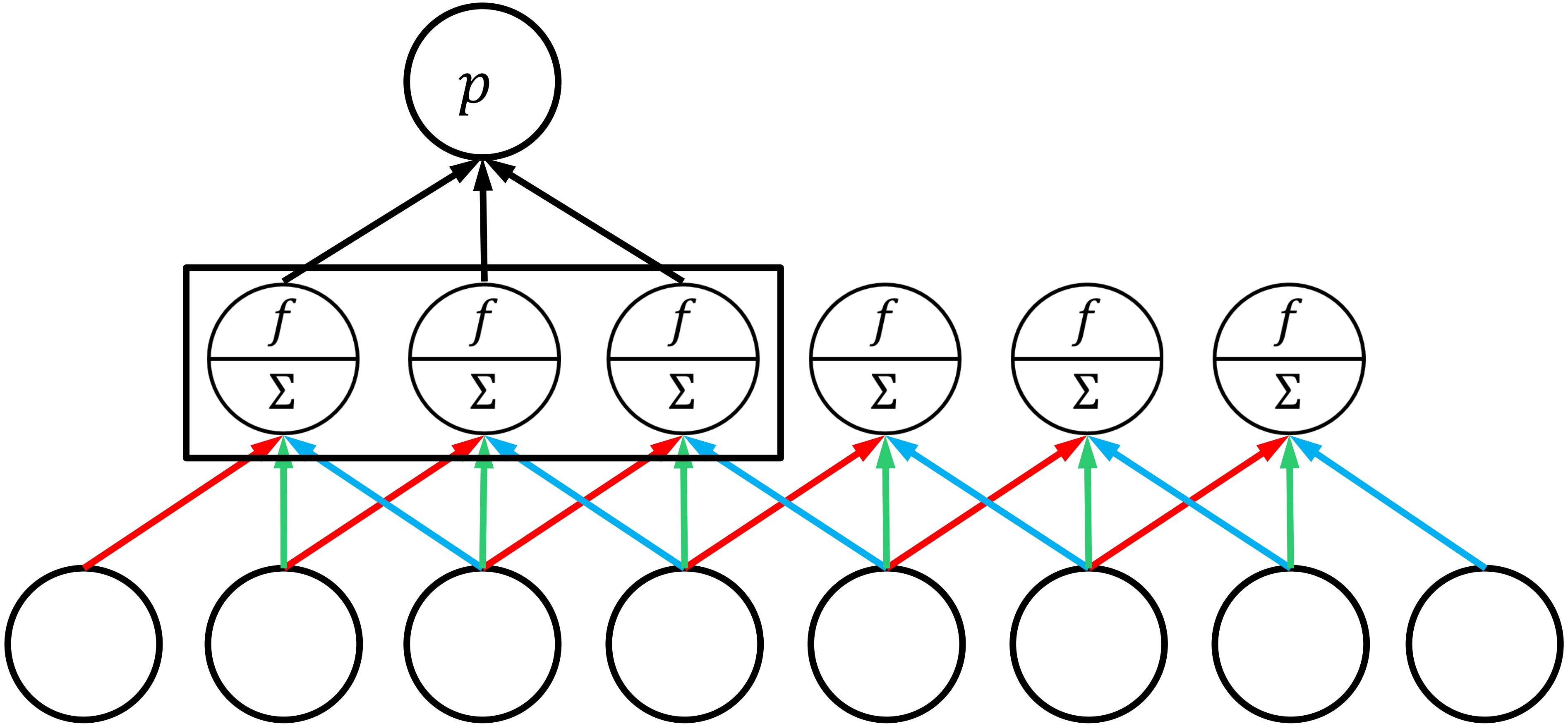
Max-pooling and average pooling most common



Pooling Operations

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

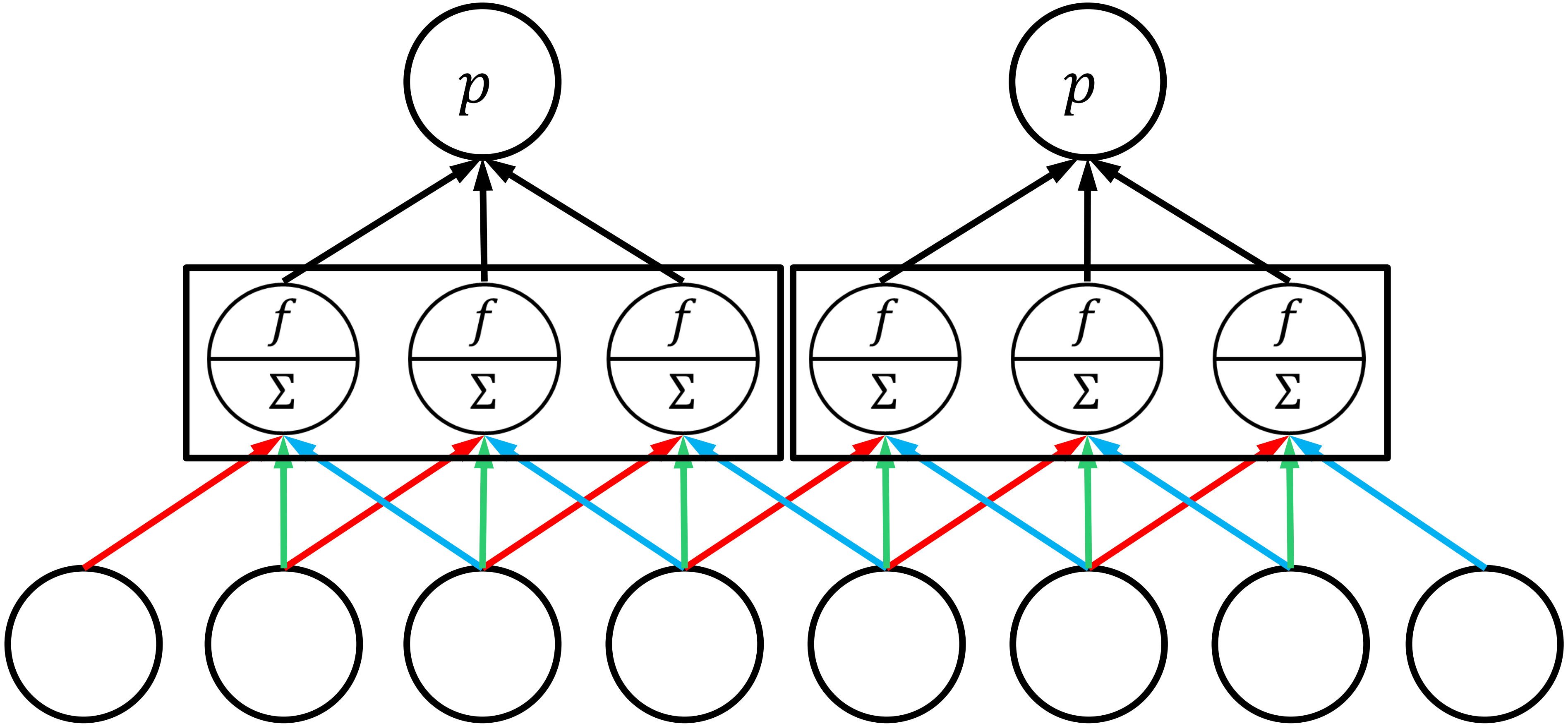
Max-pooling and average pooling most common



Pooling Operations

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

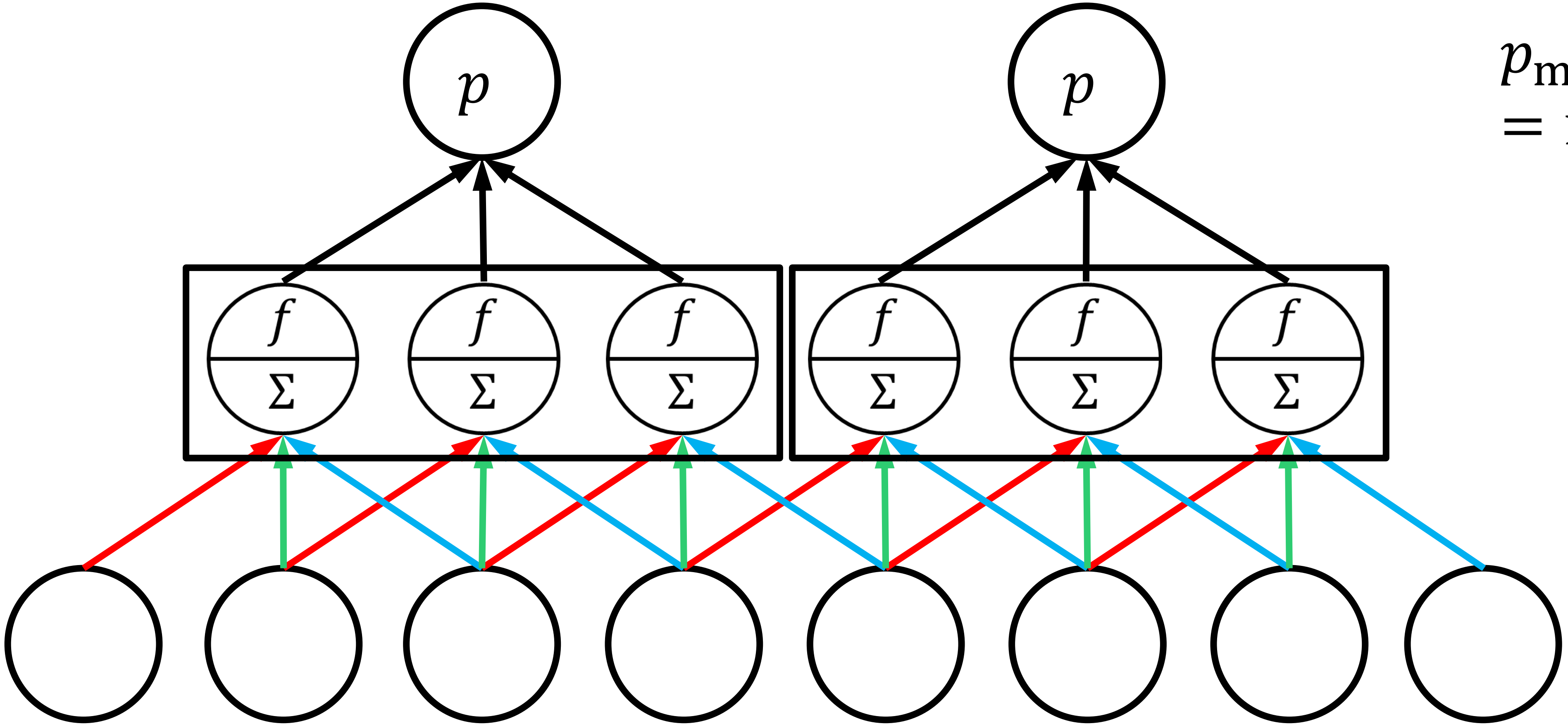
Max-pooling and average pooling most common



Pooling Operations

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

Max-pooling and average pooling most common

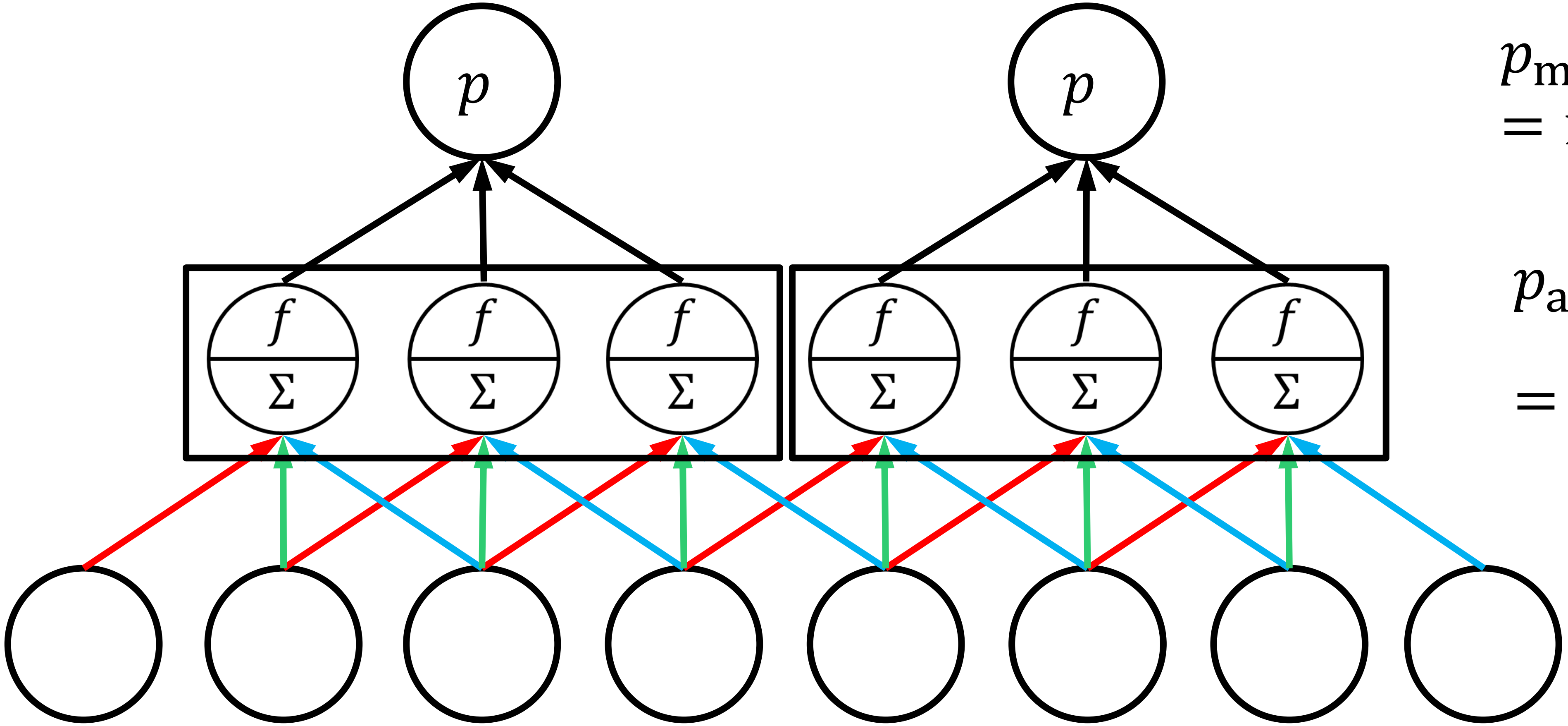


$$p_{\max}(\mathbf{h}_1, \mathbf{h}_2, \mathbf{h}_3) = \max\{\mathbf{h}_1, \mathbf{h}_2, \mathbf{h}_3\}$$

Pooling Operations

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

Max-pooling and average pooling most common



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

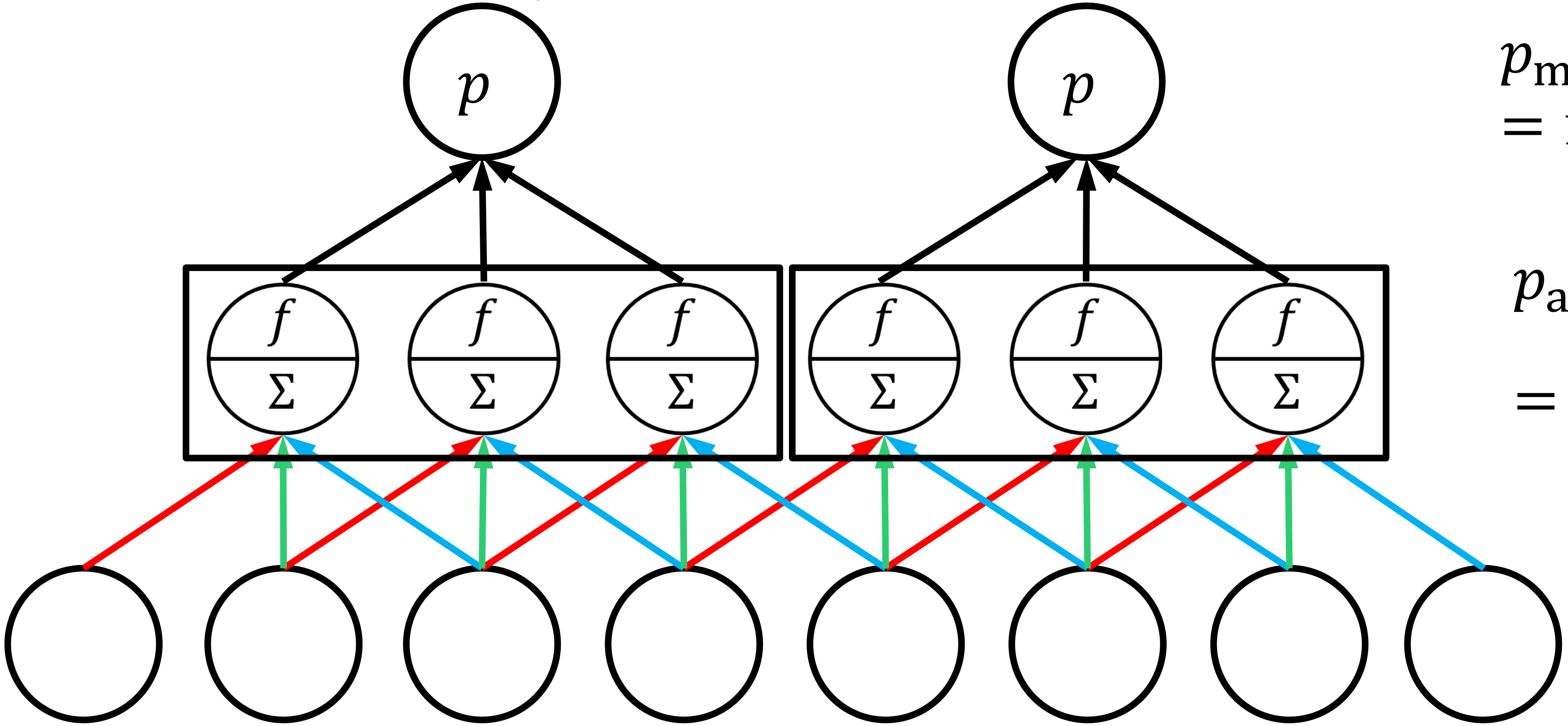
$$p_{\text{avg}}(\mathbf{h}_1, \mathbf{h}_2, \mathbf{h}_3) = \frac{1}{3}(\mathbf{h}_1 + \mathbf{h}_2 + \mathbf{h}_3)$$

Pooling Operations

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

Max-pooling and averag

"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

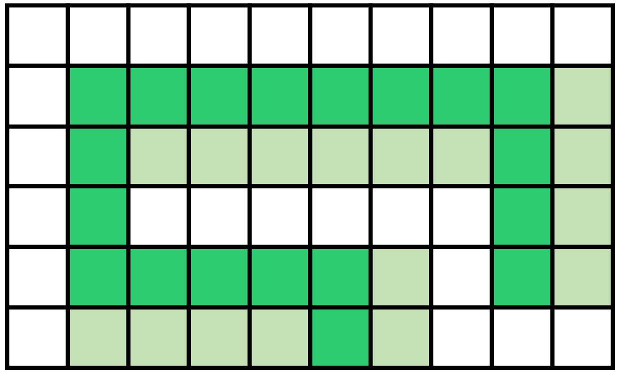
Kernels

Convolved Image

Max Pooling
(stride 1x2)

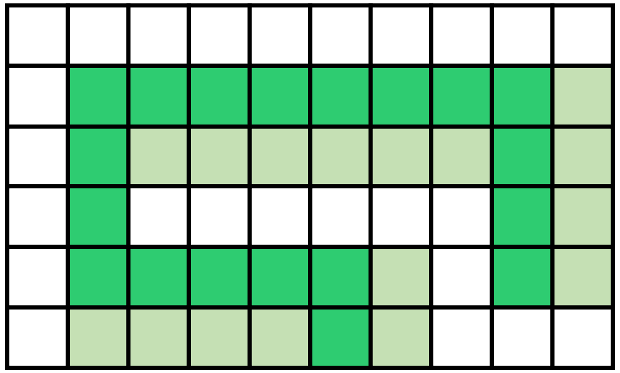
Simple Operations using Conv and Pool

Raw Image | Kernels | Convolved Image | Max Pooling (stride 1x2)



Simple Operations using Conv and Pool

Raw Image | Kernels | Convolved Image | Max Pooling (stride 1x2)



$= +1$

$= +0.5$

$= 0$

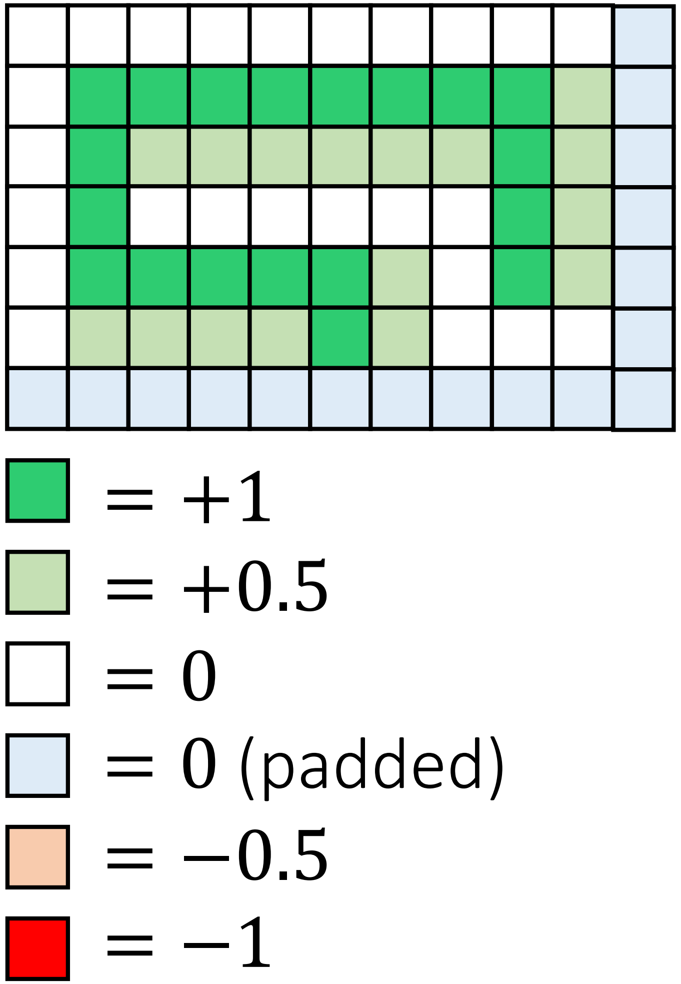
$= 0$ (padded)

$= -0.5$

$= -1$

Simple Operations using Conv and Pool

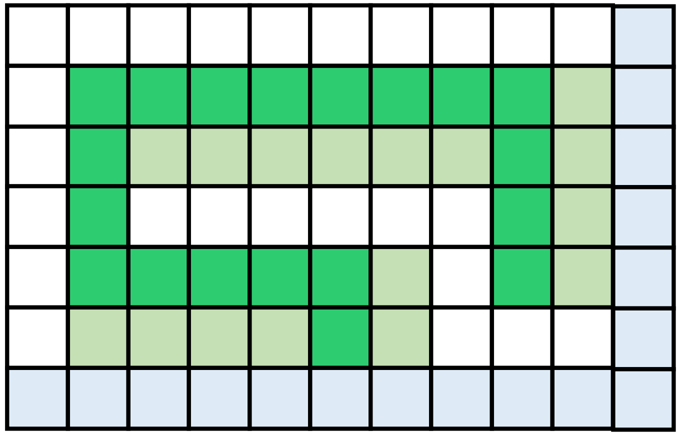
Raw Image | Kernels | Convolved Image | Max Pooling (stride 1x2)



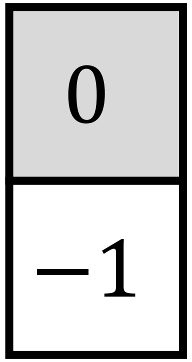
Images are often padded with zero pixels so that convolved image is of same size

Simple Operations using Conv and Pool

Raw Image



Kernels



Detects horizontal edges!

Convolved Image

Max Pooling (stride 1x2)

$= +1$

$= +0.5$

$= 0$

$= 0$ (padded)

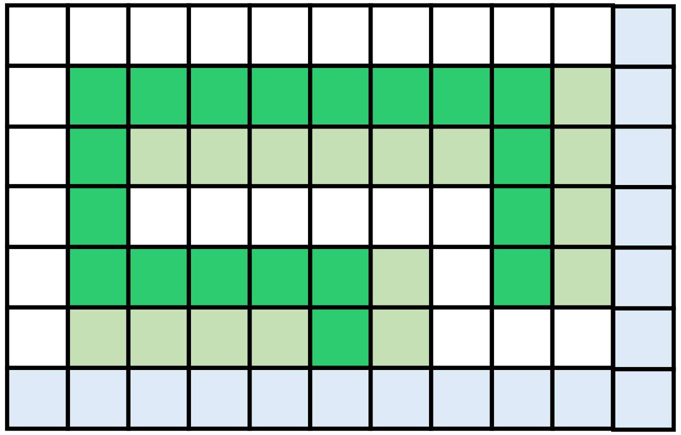
$= -0.5$

$= -1$

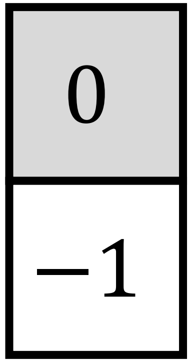
Images are often padded with zero pixels so that convolved image is of same size

Simple Operations using Conv and Pool

Raw Image

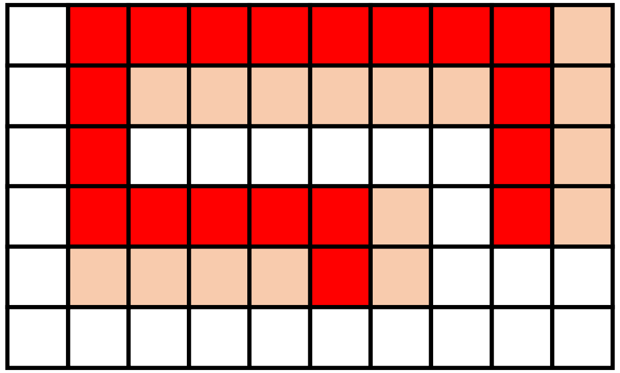


Kernels



Detects horizontal edges!

Convolved Image



Max Pooling (stride 1x2)

$= +1$

$= +0.5$

$= 0$

$= 0$ (padded)

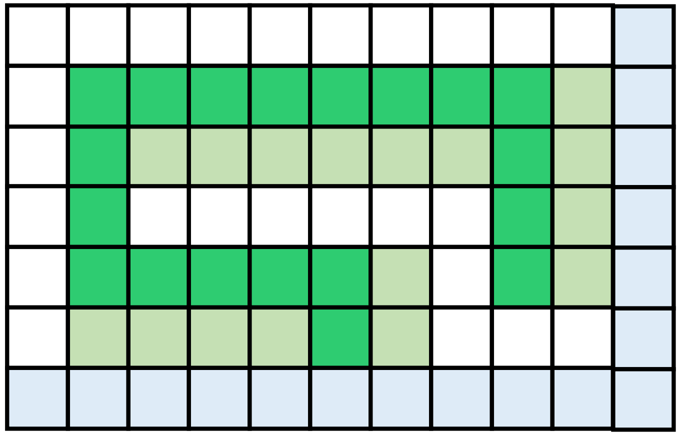
$= -0.5$

$= -1$

Images are often padded with zero pixels so that convolved image is of same size

Simple Operations using Conv and Pool

Raw Image



$= +1$

$= +0.5$

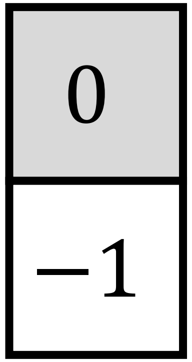
$= 0$

$= 0$ (padded)

$= -0.5$

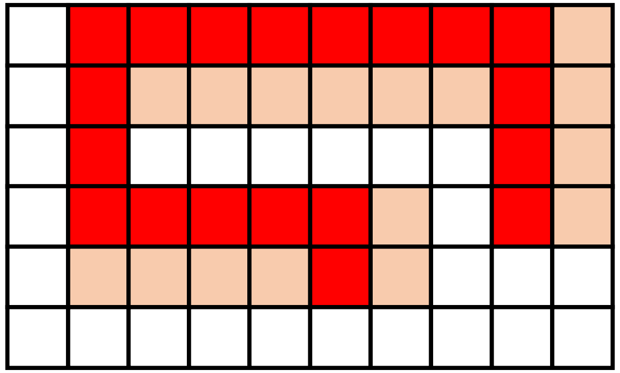
$= -1$

Kernels

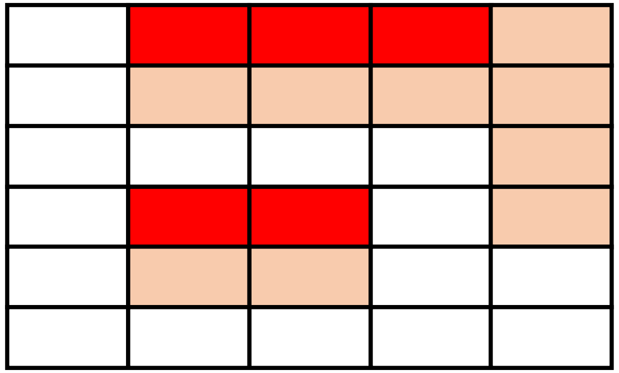


Detects horizontal edges!

Convolved Image



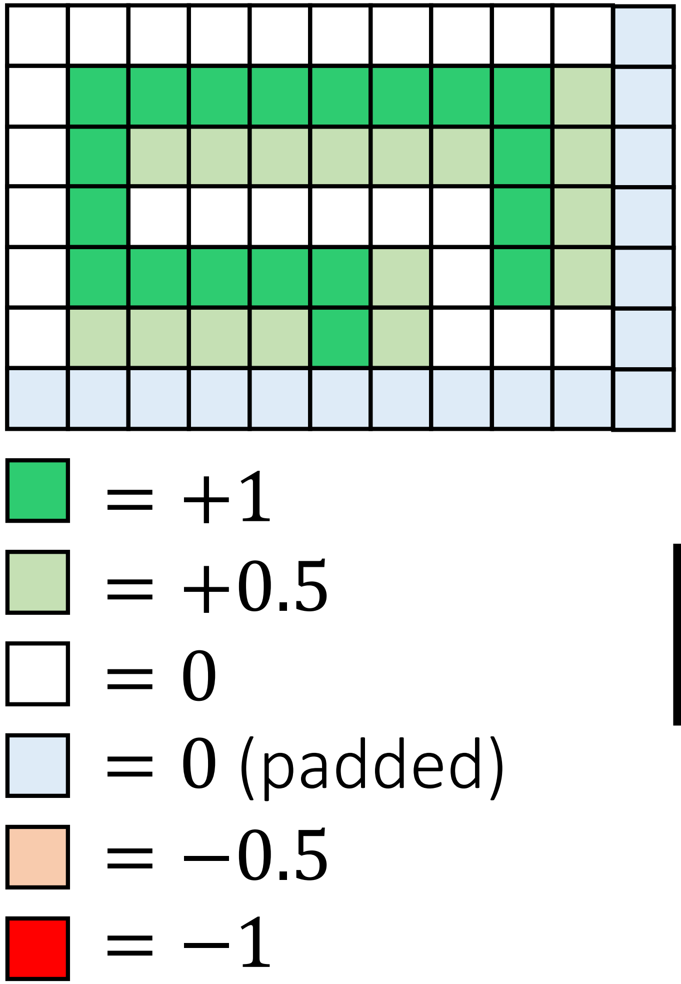
Max Pooling (stride 1x2)



Images are often padded with zero pixels so that convolved image is of same size

Simple Operations using Conv and Pool

Raw Image



Kernels

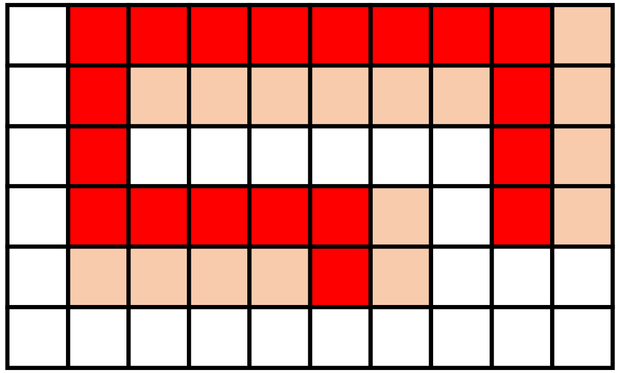
$$\begin{bmatrix} 0 \\ -1 \end{bmatrix}$$

Detects horizontal edges!

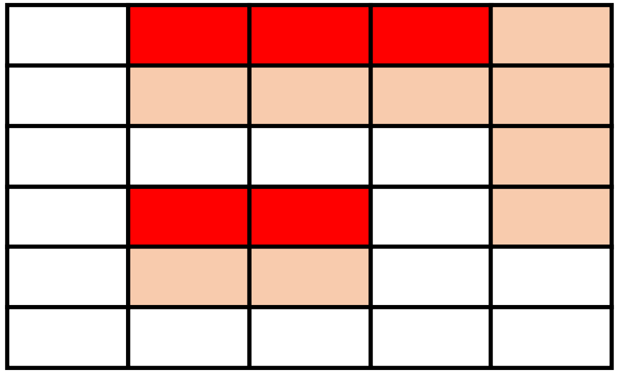
$$\begin{bmatrix} 0 & -1 \end{bmatrix}$$

Detects vertical edges!

Convolved Image



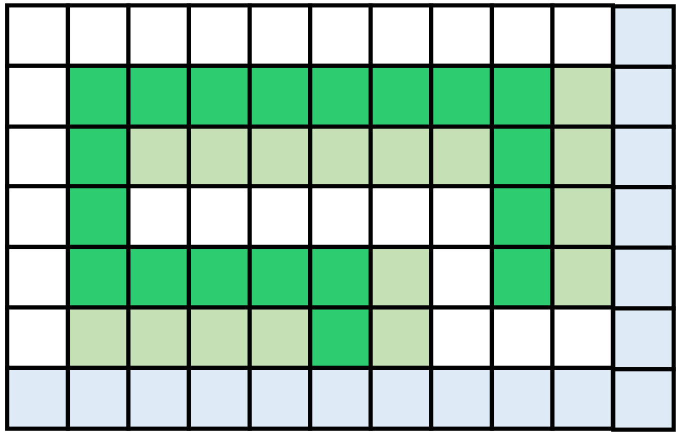
Max Pooling (stride 1x2)

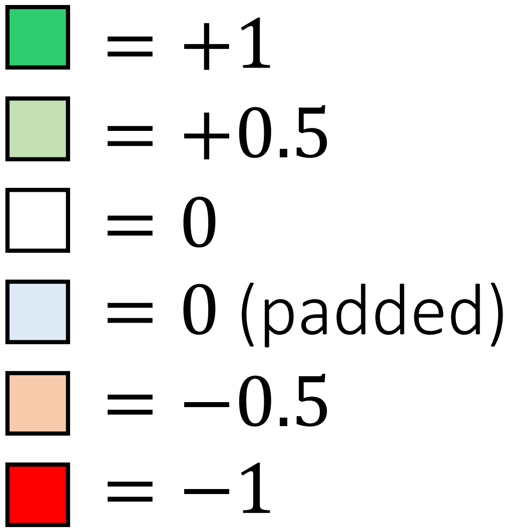


Images are often padded with zero pixels so that convolved image is of same size

Simple Operations using Conv and Pool

Raw Image	Kernels	Convolved Image	Max Pooling (stride 1x2)



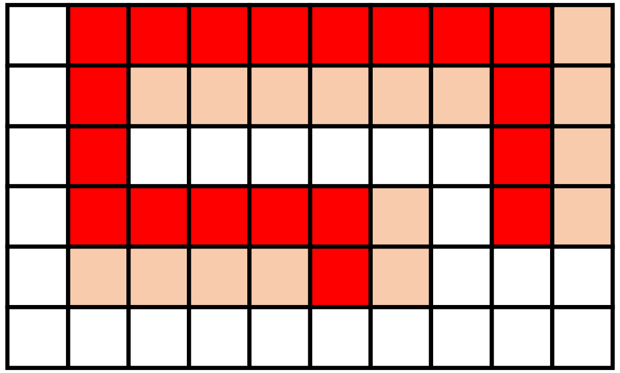


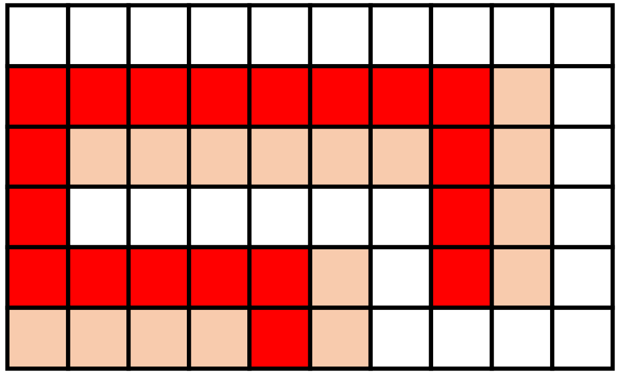
$$\begin{bmatrix}0\\-1\end{bmatrix}$$

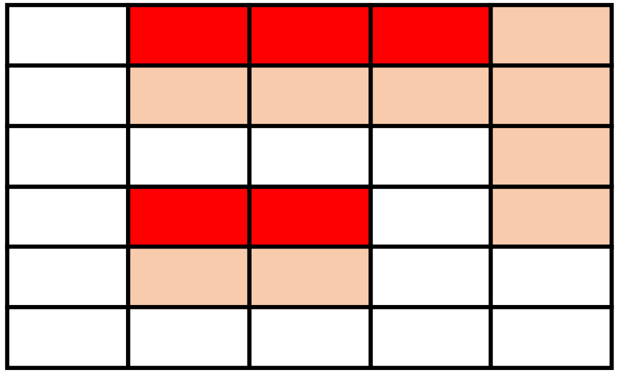
Detects horizontal edges!

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

Detects vertical edges!



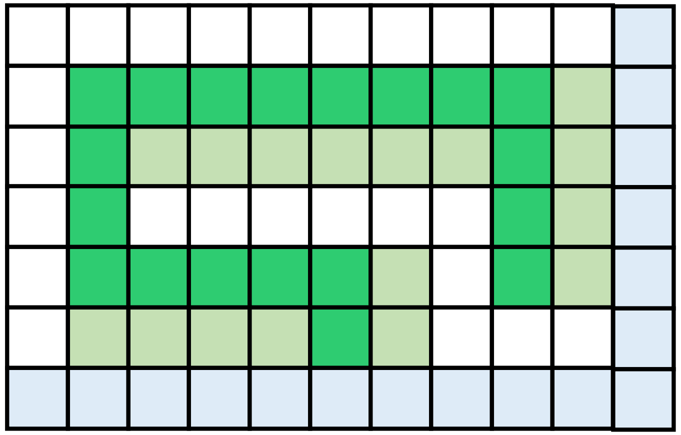




Images are often padded with zero pixels so that convolved image is of same size

Simple Operations using Conv and Pool

Raw Image



= +1

= +0.5

= 0

= 0 (padded)

= −0.5

= −1

Kernels

0
−1

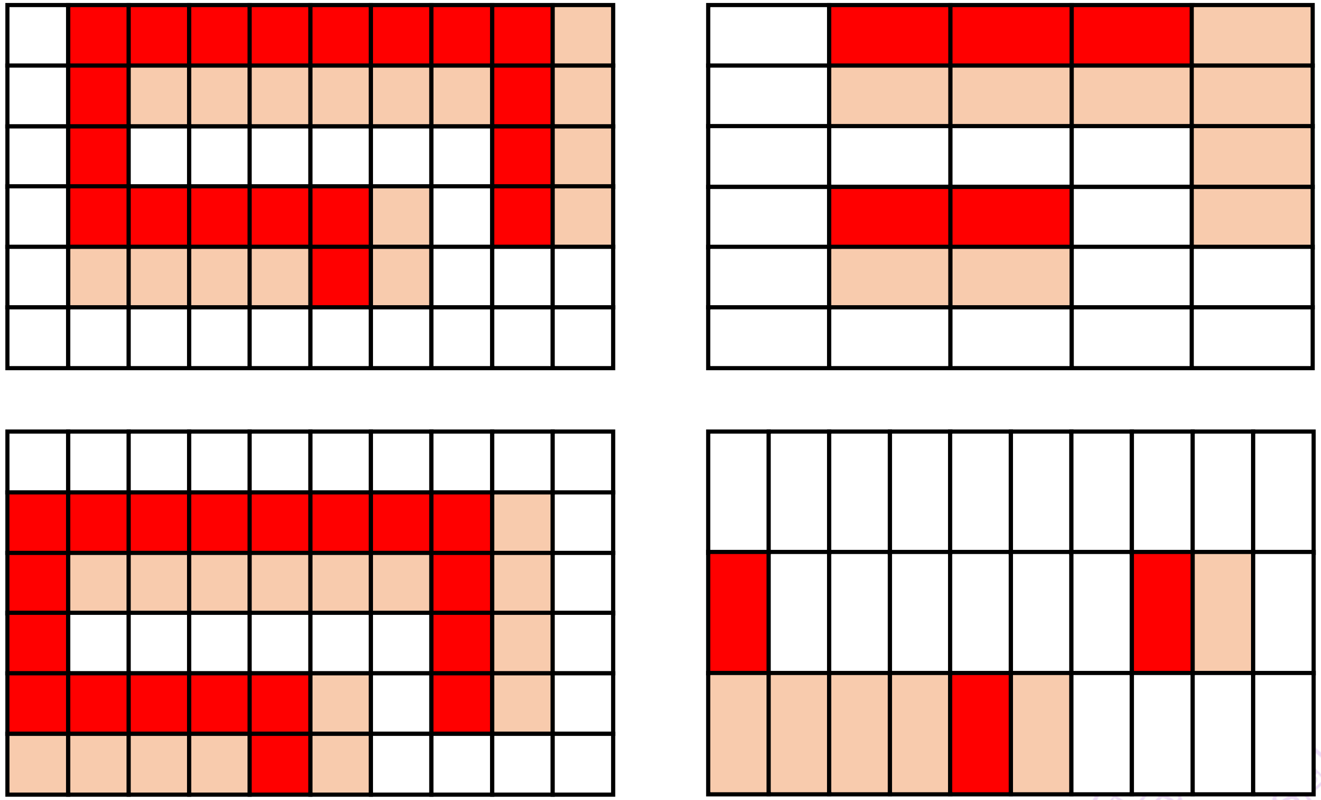
Detects horizontal edges!

0	−1

Detects vertical edges!

Convolved Image

Max Pooling (stride 1x2)



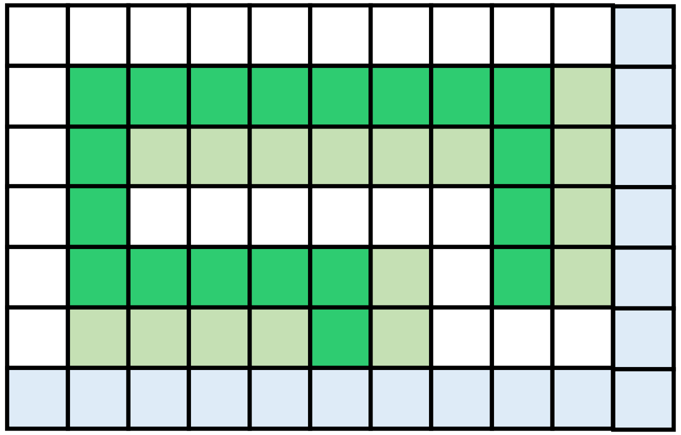
Images are often padded with zero pixels so that convolved image is of same size

Max Pooling (stride 2x1)

Simple Operations using Conv and Pool

R

We may verify that 2x2 stride leads to too much info loss



$= +1$

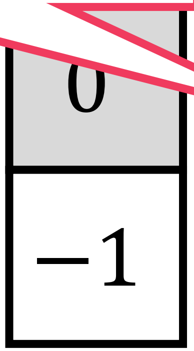
$= +0.5$

$= 0$

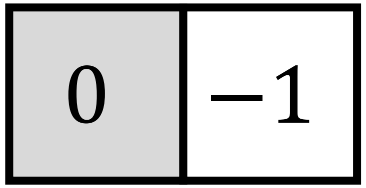
$= 0$ (padded)

$= -0.5$

$= -1$

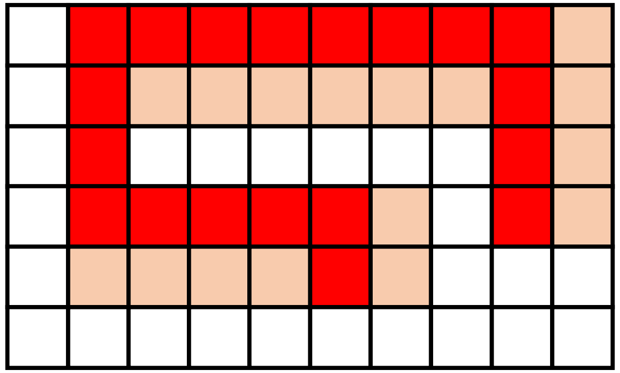


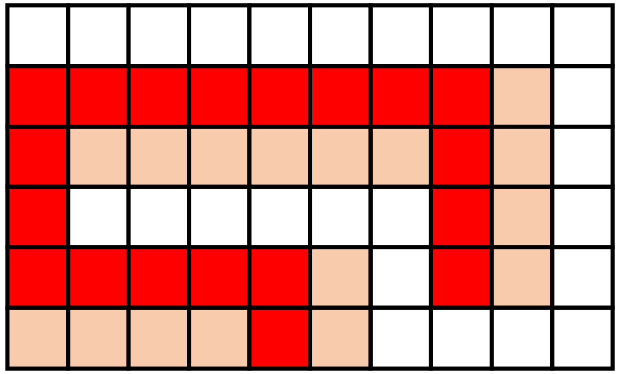
Detects horizontal edges!



Detects vertical edges!

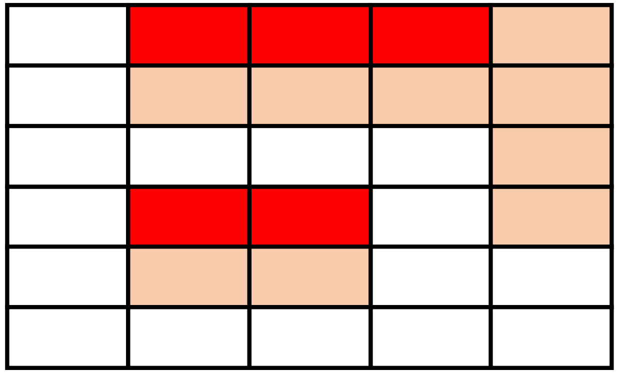
Convolved Image

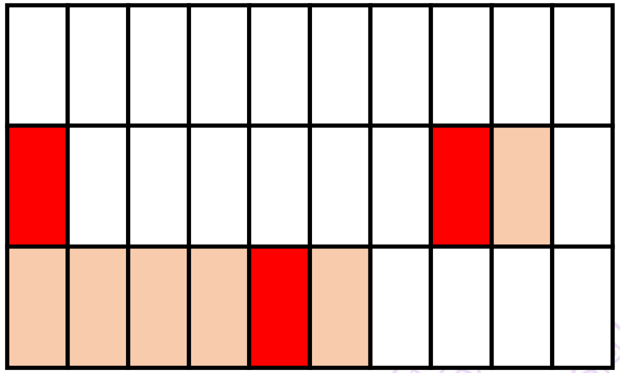




Images are often padded with zero pixels so that convolved image is of same size

Max Pooling (stride 1x2)

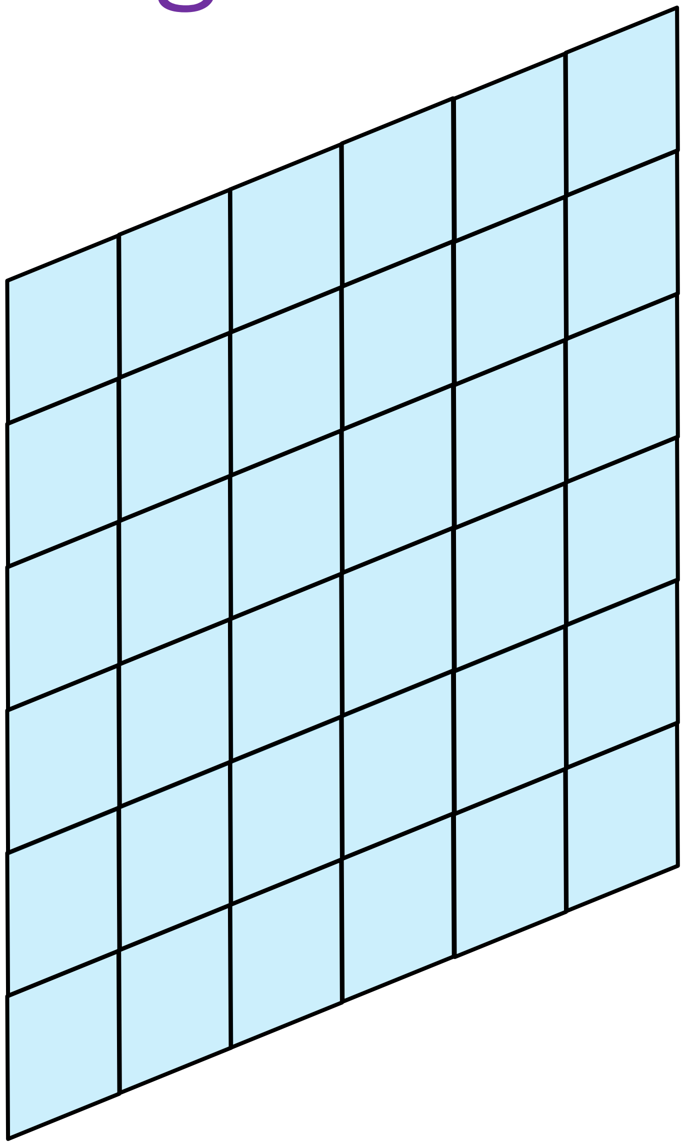




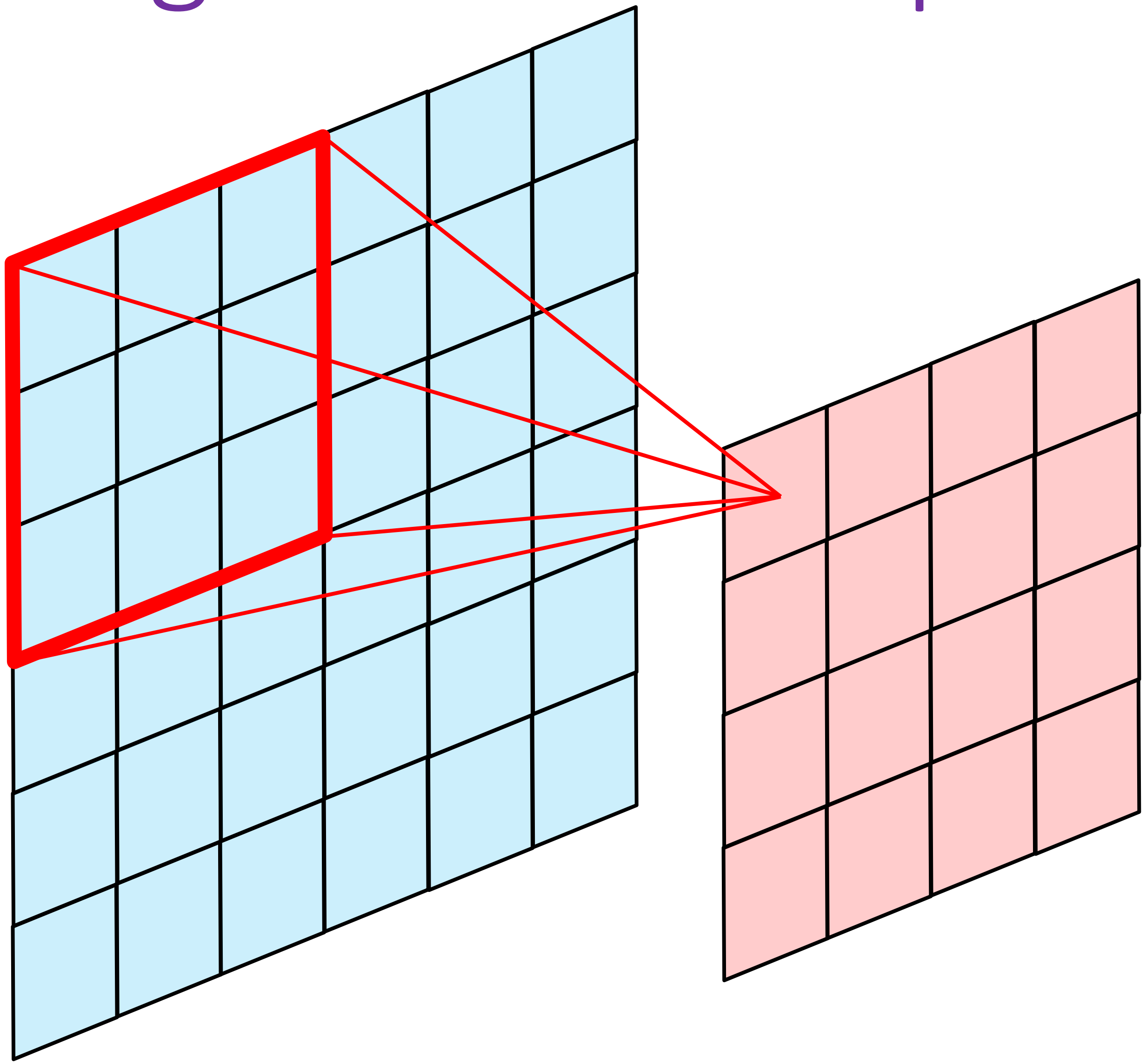
Max Pooling (stride 2x1)

Using CNNs in Computer Vision

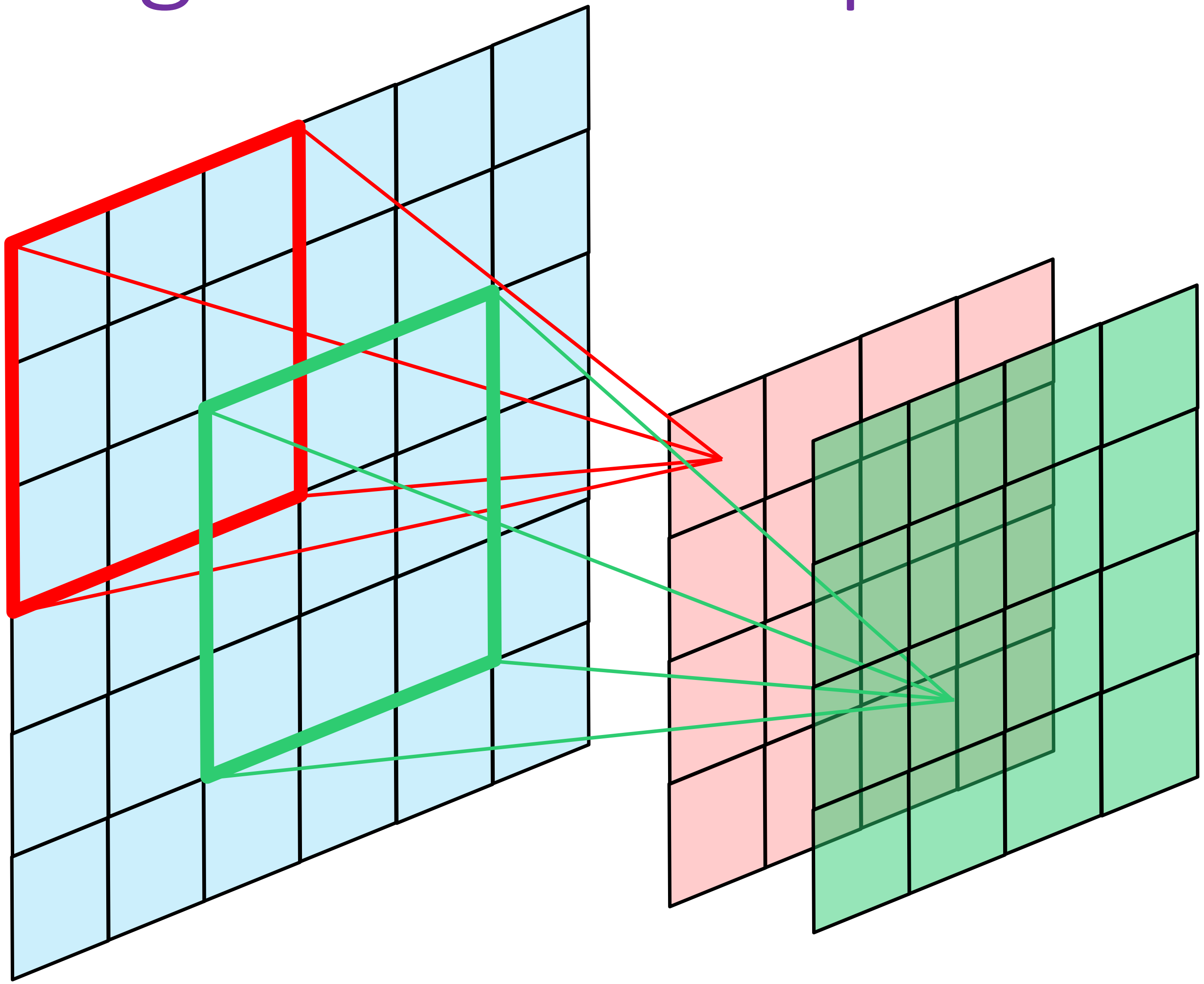
Using CNNs in Computer Vision



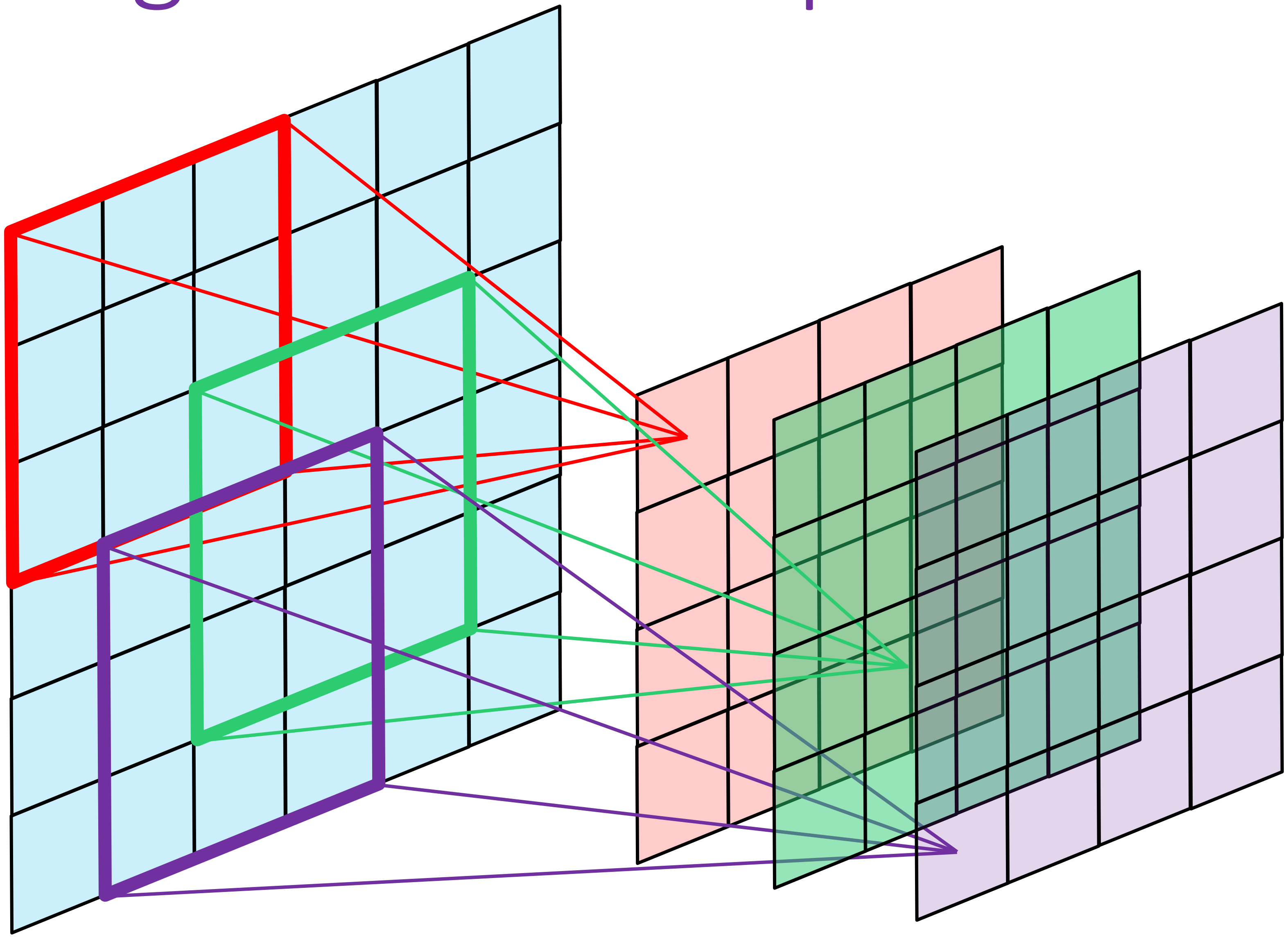
Using CNNs in Computer Vision



Using CNNs in Computer Vision

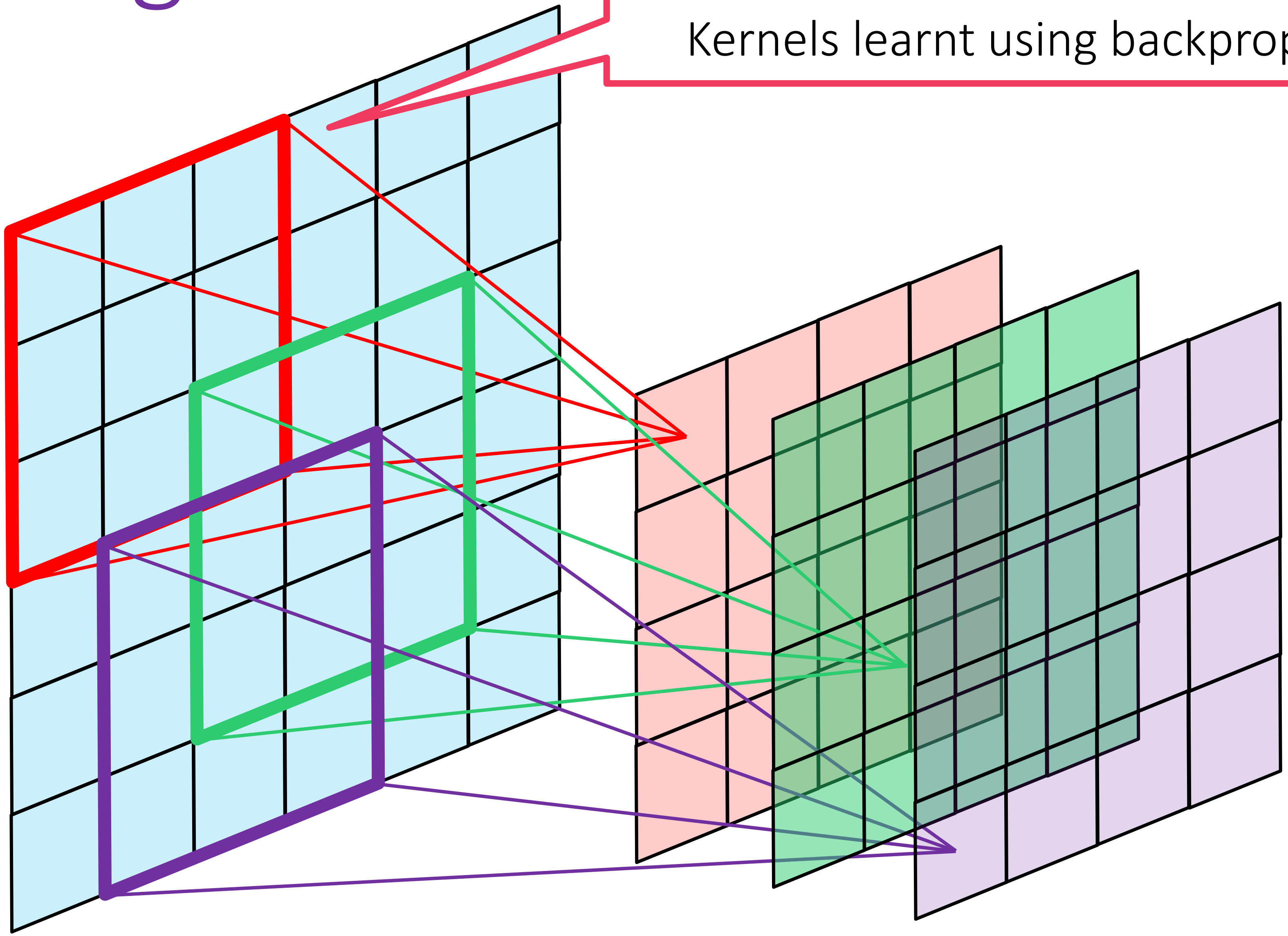


Using CNNs in Computer Vision

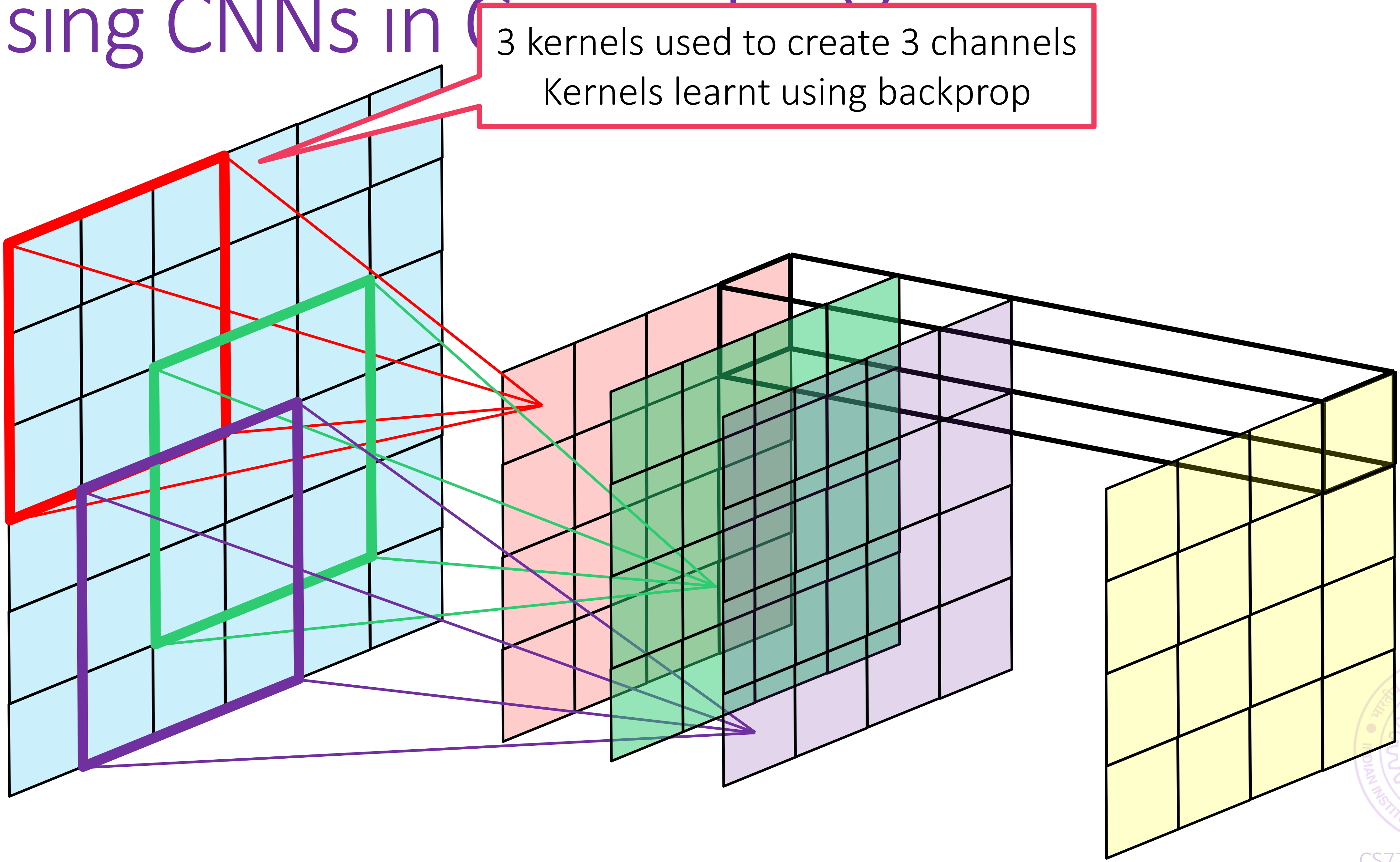


Using CNNs in C[…] Vi[…]

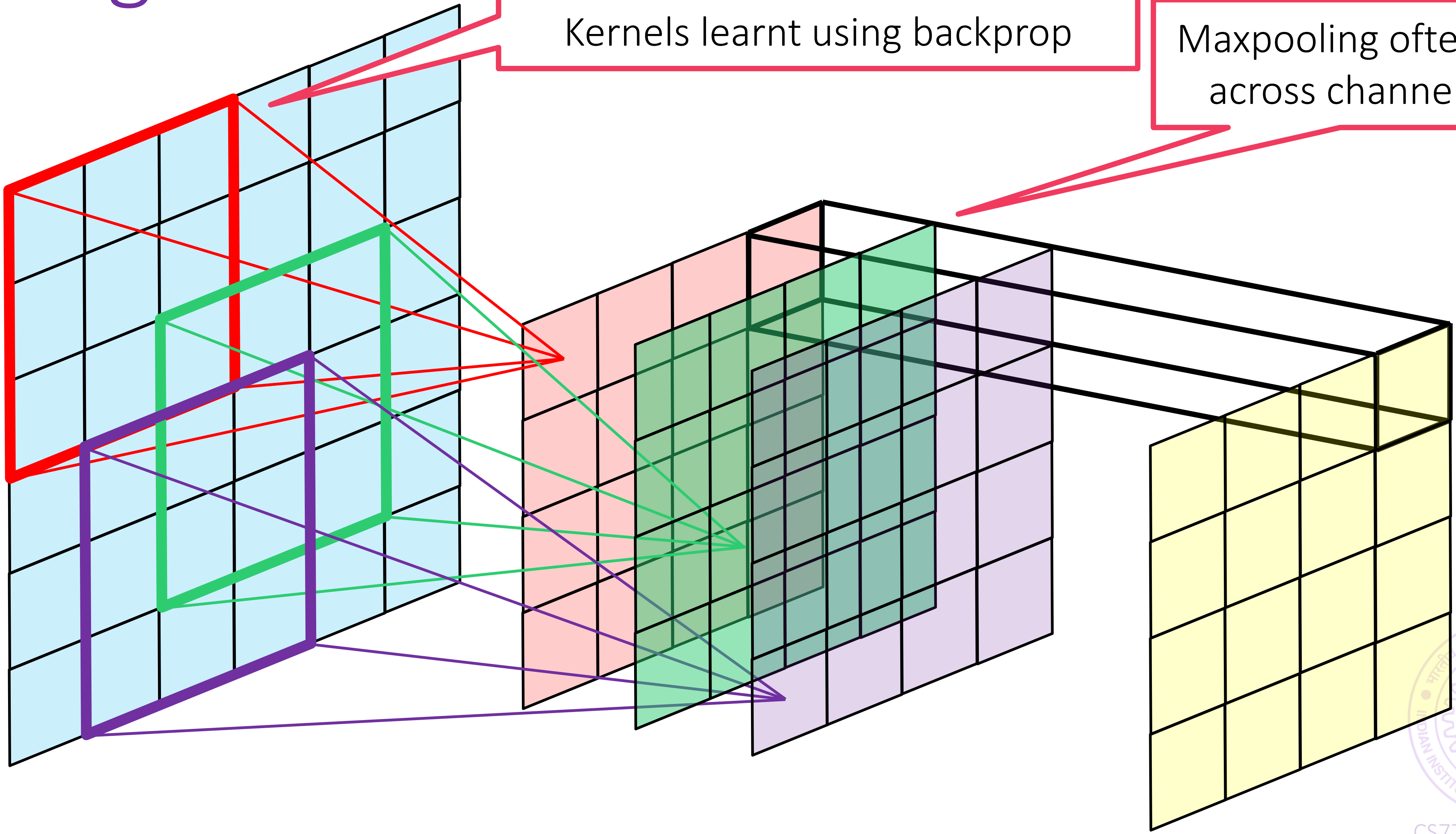
3 kernels used to create 3 channels
Kernels learnt using backprop



Using CNNs in C…



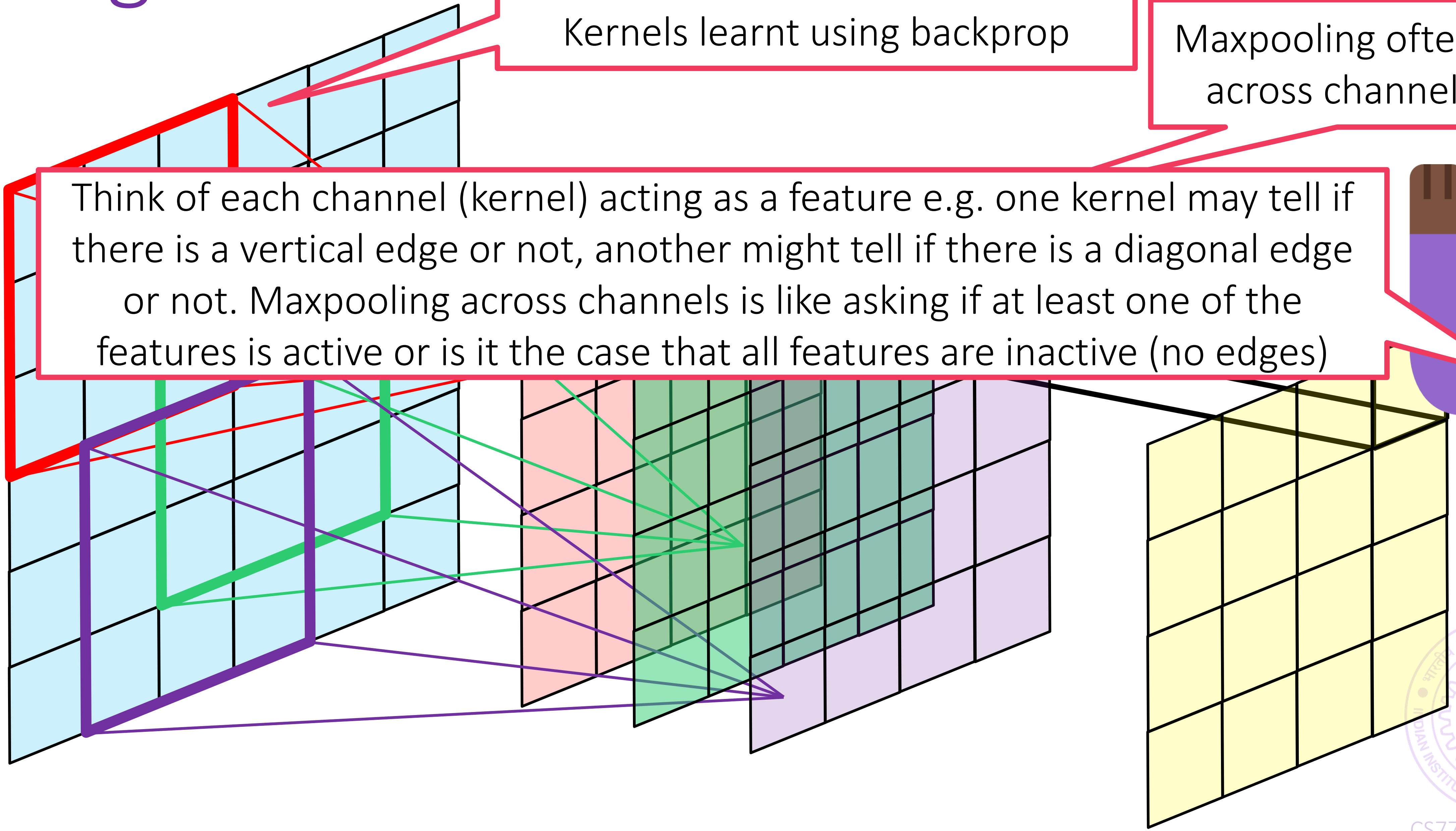
Using CNNs in C



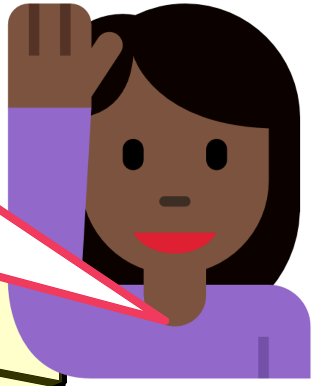
Using CNNs in C

3 kernels used to create 3 channels
Kernels learnt using backprop

Maxpooling often done
across channels too



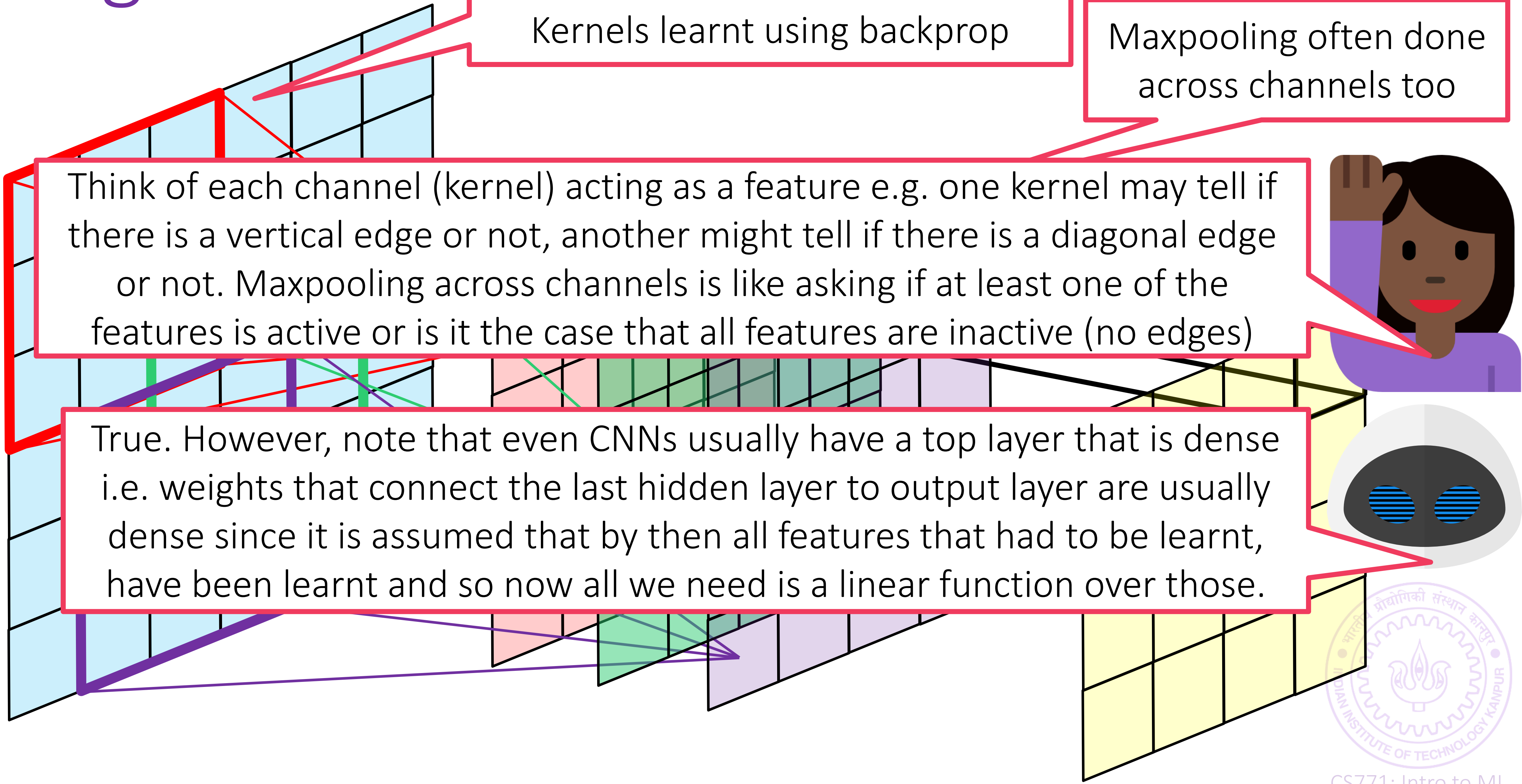
Think of each channel (kernel) acting as a feature e.g. one kernel may tell if there is a vertical edge or not, another might tell if there is a diagonal edge or not. Maxpooling across channels is like asking if at least one of the features is active or is it the case that all features are inactive (no edges)



Using CNNs in C

3 kernels used to create 3 channels
Kernels learnt using backprop

Maxpooling often done across channels too



Think of each channel (kernel) acting as a feature e.g. one kernel may tell if there is a vertical edge or not, another might tell if there is a diagonal edge or not. Maxpooling across channels is like asking if at least one of the features is active or is it the case that all features are inactive (no edges)

True. However, note that even CNNs usually have a top layer that is dense i.e. weights that connect the last hidden layer to output layer are usually dense since it is assumed that by then all features that had to be learnt, have been learnt and so now all we need is a linear function over those.