EEP590 Spring 2022

Deep Learning for Embedded Real Time Intelligence

Lecture 4: Convolutional Neural Networks

Prof. Richard Shi Department of Electrical and Computer Engineering
(cjshi@uw.edu)

with the material adapted from **Joseph Redmon** and **Ali Farhadi** at **UW CSE**

ELECTRICAL & COMPUTER ENGINEERING

Neural Networks and Images

Neural networks are densely connected

Each neuron in layer i connected to every neuron in layer i+1

HOG features: 36 features / 8 x 8 patch

Neural Networks and Images

Neural networks are densely connected

Each neuron in layer i connected to every neuron in layer i+1

HOG features: 36 features / 8 x 8 patch

Standard CV feature extraction technique

Neural Networks and Images

Neural networks are densely connected

Each neuron in layer i connected to every neuron in layer i+1

HOG features: 36 features / 8 x 8 patch (Histogram of Oriented Gradients): a feature descriptor like the Canny Edge Detector, SIFT (Scale Invariant and Feature Transform) . For object detection, the HOG technique counts occurrences of gradient orientation in the localized portion of an image. The HOG descriptor focuses on the structure or the shape of an object. it uses magnitude as well as angle of the gradient to compute the features. For the regions of the image it generates histograms using the magnitude and orientations of the gradient.

Say we want to process images:

Input : 256 x 256 x 3 RGB image

Hidden : 32 x 32 x 36 feature map?

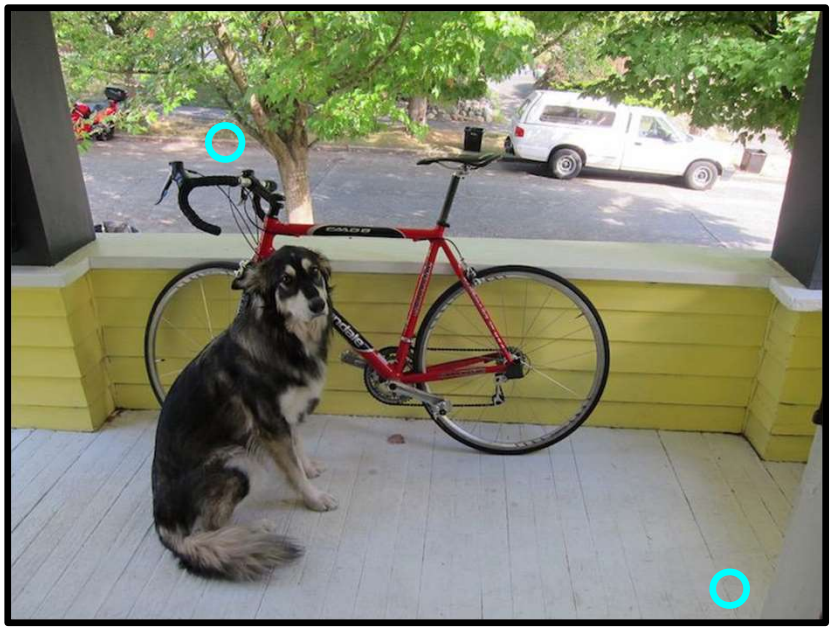
Output : 1000 classes

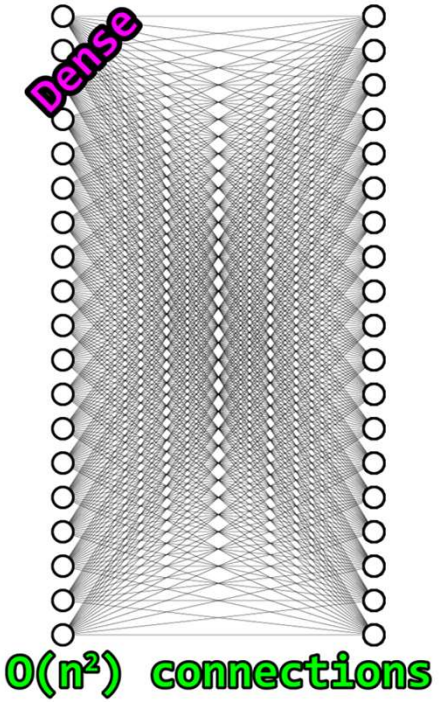
Input -> hidden is 7.2 billion connections!

Full Connection: Too Many Weights!

Neural networks are densely connected

But is this really what we want when processing images?





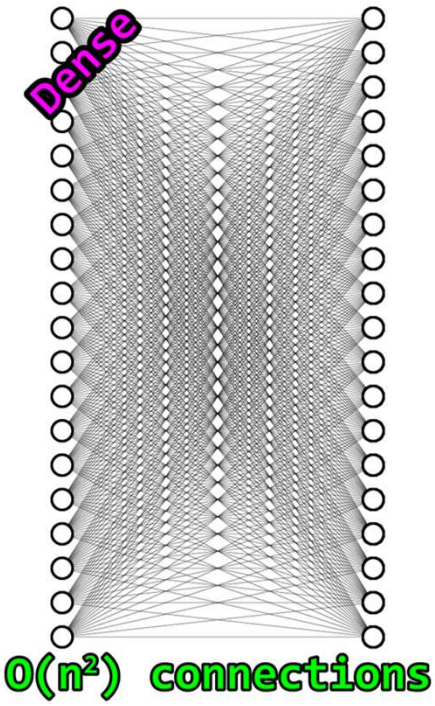
Dense Weights → Shared Sparse Structured Weights

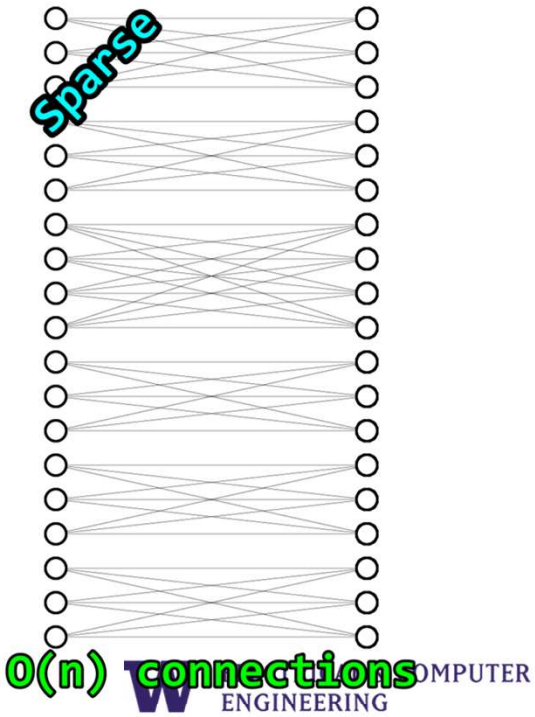
Would rather have sparse connections

- Fewer weights
- Nearby regions - related
- Far apart - not related

Operator: Convolutions!

- Just weighted sums of small areas in image
- Weight sharing in different locations in image





CNNs: Convolutional Neural Networks

Use *convolutions* instead of *dense connections* to process images

Take advantage of structure in the data!

Impose an assumption on the model:

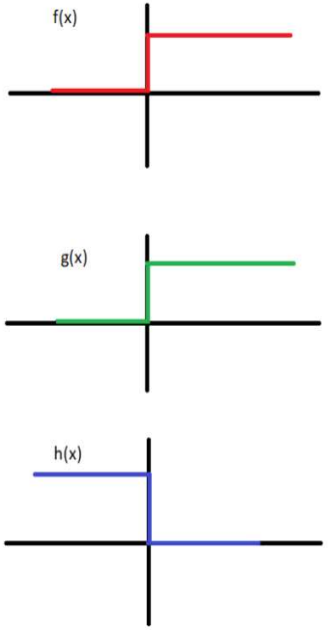
Nearby pixels are related, far apart ones are less related

Share weights among regions

Going Back to 1-D: Cross-Correlation vs Convolution

Consider *f(x) and g(x)* to be step functions but they can be different, and ***h(x)*** *is a time-reversed (flipped) version of* ***g(x)***.

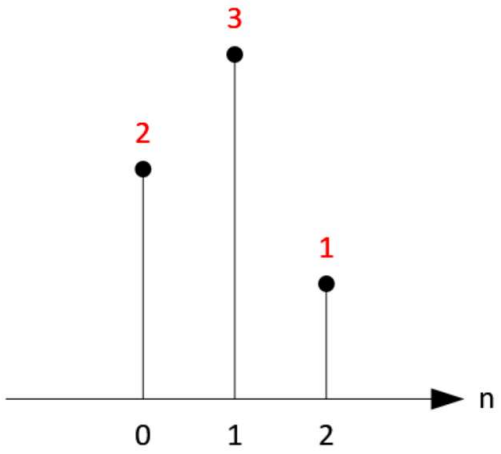
- ***Cross-Correlation of f(x) and g(x)*** *is found by fixing any one of the two functions, let's say we fix* ***f(x)***, *and slide the other,* ***g(x)***, *over* ***f(x)*** *while multiplying the functions and adding at each shift.*
- *Cross-Correlation of two functions gives a* ***measure of similarity*** *between the functions.*



Convolution of *f(x)* and *g(x)* *is found by fixing any one of the two functions, let's say we fix* ***f(x)***, *and* **flipping the other function, *g(x)*, about the y-axis** *and then sliding it over* ***f(x)*** *while multiplying the functions and adding at each shift.*
Convolution *of two functions gives the* ***output fuction*** *when one function is* ***transformed*** *by the other.*

Convolution of f(x) and g(x) = **Cross-Correlation of *f(x)* and *g(- x)***

1D Representation of Time Series Data



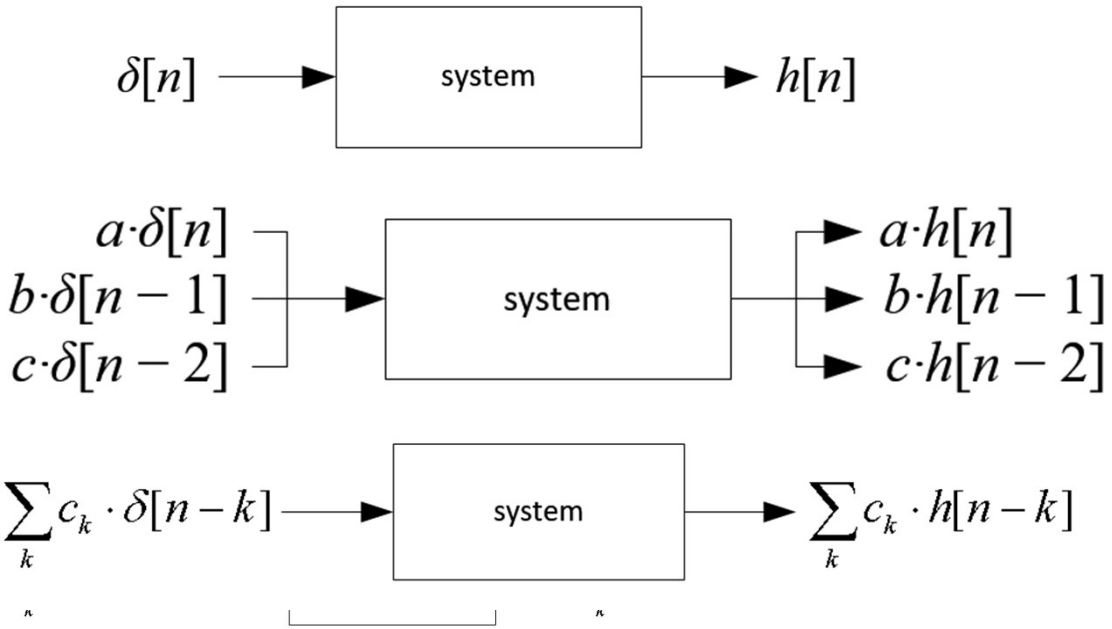
$$x[0] = x[0] \cdot \delta[n] \quad = 2 \cdot \delta[n-0]$$
$$x[1] = x[1] \cdot \delta[n-1] = 3 \cdot \delta[n-1]$$
$$x[2] = x[2] \cdot \delta[n-2] = 1 \cdot \delta[n-2]$$

$$x[n] = x[0] \cdot \delta[n-0] + x[1] \cdot \delta[n-1] + x[2] \cdot \delta[n-2]$$

In general, a signal can be written as sum of scaled and shifted delta functions;

$$x[n] = \sum_{k=-\infty}^{\infty} x[k] \cdot \delta[n-k]$$

Transfer Function of a Linear System



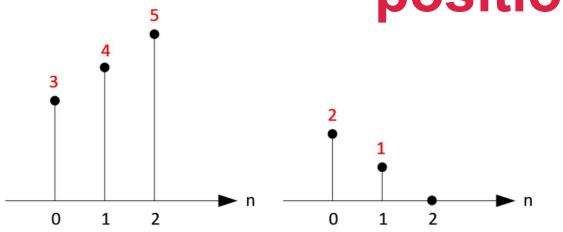
For example, if input signal is $x[n] = 2 \cdot \delta[n] + 3 \cdot \delta[n-1] + 1 \cdot \delta[n-2]$, then the output is simply $y[n] = 2 \cdot h[n] + 3 \cdot h[n-1] + 1 \cdot h[n-2]$.

Therefore, we now clearly see that if the input signal is $x[n] = \sum x[k] \cdot \delta[n-k]$, then the output will be $y[n] = \sum x[k] \cdot h[n-k]$. Note one condition; convolution works on the linear and time invariant system.

To summarize, a signal is decomposed into a set of impulses and the output signal can be computed by adding the scaled and shifted impulse responses.

1D Convolution Example

$x[n] = \{ 3, 4, 5 \}$
$h[n] = \{ 2, 1 \}$



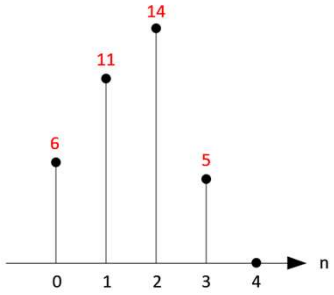
Input: $x[n]$ Impulse Response: $h[n]$

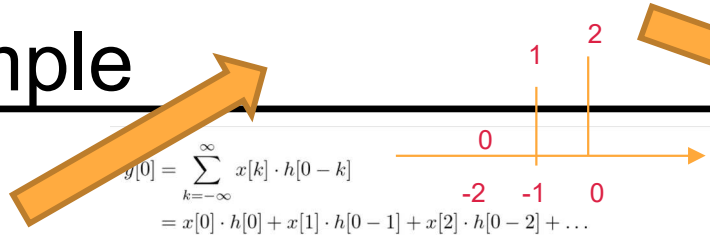
One thing to note before we move on: Try to figure out you

Shifted by n positions

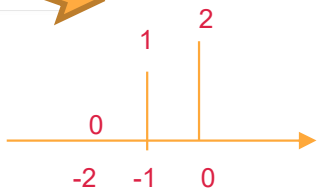
flipped

$$y[n] = \sum x[k] \cdot h[n-k]$$

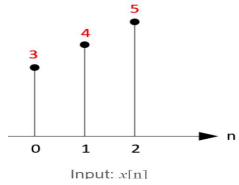




$$y[0] = \sum_{k=-\infty}^{\infty} x[k] \cdot h[0-k]$$
$$= x[0] \cdot h[0] + x[1] \cdot h[0-1] + x[2] \cdot h[0-2] + \dots$$
$$= x[0] \cdot h[0]$$
$$= 3 \cdot 2$$
$$= 6$$



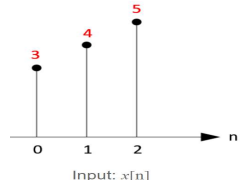
$$y[1] = \sum_{k=-\infty}^{\infty} x[k] \cdot h[1-k]$$
$$= x[0] \cdot h[1-0] + x[1] \cdot h[1-1] + x[2] \cdot h[1-2] + \dots$$
$$= x[0] \cdot h[1] + x[1] \cdot h[0]$$
$$= 3 \cdot 1 + 4 \cdot 2$$
$$= 11$$



Input: $x[n]$

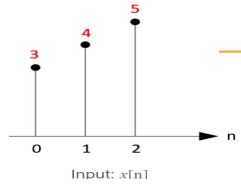
Shifted Vector Dot product

$$y[2] = \sum_{k=-\infty}^{\infty} x[k] \cdot h[2-k]$$
$$= x[0] \cdot h[2-0] + x[1] \cdot h[2-1] + x[2] \cdot h[2-2] + \dots$$
$$= x[0] \cdot h[2] + x[1] \cdot h[1] + x[2] \cdot h[0]$$
$$= 3 \cdot 0 + 4 \cdot 1 + 5 \cdot 2$$
$$= 14$$

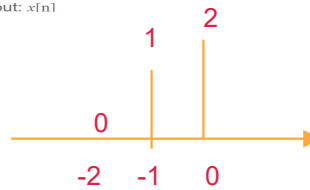


Input: $x[n]$

$$y[3] = \sum_{k=-\infty}^{\infty} x[k] \cdot h[3-k]$$
$$= x[0] \cdot h[3-0] + x[1] \cdot h[3-1] + x[2] \cdot h[3-2] + x[3] \cdot h[3-3] + \dots$$
$$= x[0] \cdot h[3] + x[1] \cdot h[2] + x[2] \cdot h[1] + x[3] \cdot h[0]$$
$$= 3 \cdot 0 + 4 \cdot 0 + 5 \cdot 1 + 0 \cdot 2$$
$$= 5$$



Input: $x[n]$



$$y[4] = \sum_{k=-\infty}^{\infty} x[k] \cdot h[4-k]$$
$$= x[0] \cdot h[4-0] + x[1] \cdot h[4-1] + x[2] \cdot h[4-2] + x[3] \cdot h[4-3] + x[4] \cdot h[4-4] + .$$
$$= x[0] \cdot h[4] + x[1] \cdot h[3] + x[2] \cdot h[2] + x[3] \cdot h[1] + x[4] \cdot h[0]$$
$$= 3 \cdot 0 + 4 \cdot 0 + 5 \cdot 0 + 0 \cdot 2 + 0 \cdot 1 + 0 \cdot 0$$
$$= 0$$

ELECTRICAL & COMPUTER ENGINEERING

C++ Implementation of 1-D Convoluion

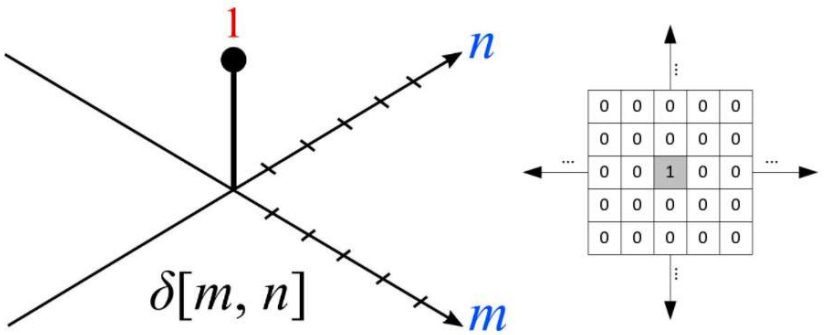
C++ Implementation for Convolution 1D

Implementing the convolution algorithm is quite simple. The code snippet is following;

```
for ( i = 0; i < sampleCount; i++ )
{
    y[i] = 0;                          // set to zero before sum
    for ( j = 0; j < kernelCount; j++ )
    {
        y[i] += x[i - j] * h[j];       // convolve: multiply and accumulate
    }
}
```

Shifted Vector Dot product

2D Signal Representation



Delta function (Impulse function) in 2D space

The second image is 2D matrix representation of impulse function. The shaded center point is the origin where m=n=0.

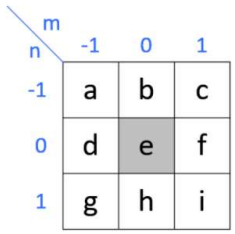
Once again, a signal can be decomposed into a sum of scaled and shifted impulse (delta) functions;

$$x[m,n] = \sum_{j=-\infty}^{\infty} \sum_{i=-\infty}^{\infty} x[i,j] \cdot \delta[m-i, n-j]$$

For example, $x[0, 0]$ is $x[0, 0]\cdot\delta[m, n]$, $x[1, 2]$ is $x[1, 2]\cdot\delta[m-1, n-2]$, and so on. Note that the matrices are referenced here as [column, row], not [row, column]. m is horizontal (column) direction and n is vertical (row) direction.

And, the output of linear and time invariant system can be written by convolution of input signal $x[m, n]$, and impulse response, $h[m, n]$;

$$y[m,n] = x[m,n] * h[m,n] = \sum_{j=-\infty}^{\infty} \sum_{i=-\infty}^{\infty} x[i,j] \cdot h[m-i, n-j]$$



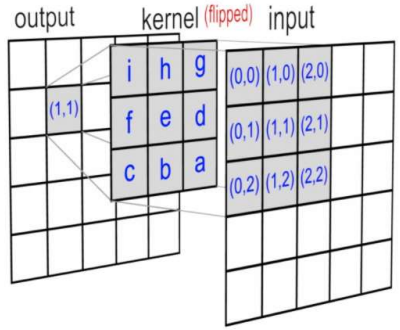
Examine an example to clarify how to convolve in 2D space.
Let's say that the size of impulse response (kernel) is 3x3, and it's values are a, b, c, d,...

Notice the origin (0,0) is located in the center of the kernel.

Let's pick a simplest sample and compute convolution, for instance, the output at (1, 1) will be;

$$\begin{aligned} y[1,1] = \sum_{j=-\infty}^{\infty} \sum_{i=-\infty}^{\infty} x[i,j] \cdot h[1-i, 1-j] \\ = \; x[0,0]\cdot h[1,1] \;+ x[1,0]\cdot h[0,1] \;+ x[2,0]\cdot h[-1,1] \\ + x[0,1]\cdot h[1,0] \;+ x[1,1]\cdot h[0,0] \;+ x[2,1]\cdot h[-1,0] \\ + x[0,2]\cdot h[1,-1] + x[1,2]\cdot h[0,-1] + x[2,2]\cdot h[-1,-1] \end{aligned}$$

It results in sum of 9 elements of scaled and shifted impulse responses. The following image shows the graphical representation of 2D convolution.



2D Convolution

Notice that the kernel matrix is flipped both horizontal and vertical direction before multiplying the overlapped input data, because $x[0,0]$ is multiplied by the last sample of impulse response, $h[1,1]$. And $x[2,2]$ is multiplied by the first sample, $h[-1,-1]$.

2D Convolution

Shifted Matrix Dot product

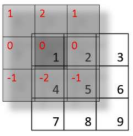
Input		
1	2	3
4	5	6
7	8	9

n \ m	-1	0	1
-1	-1	-2	-1
0	0	0	0
1	1	2	1

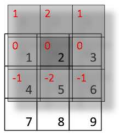
Kernel

Output		
-13	-20	-17
-18	-24	-18
13	20	17

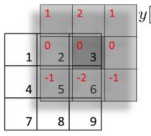
flipped



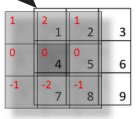
$$
\begin{aligned}
y[0,0] &= \sum_j \sum_i x[i,j] \cdot h[0-i, 0-j] \\
&= \quad x[-1,-1] \cdot h[1,1] + x[0,-1] \cdot h[0,1] + x[1,-1] \cdot h[-1,1] \\
&\quad + x[-1,0] \cdot h[1,0] \quad + x[0,0] \cdot h[0,0] \quad + x[1,0] \cdot h[-1,0] \\
&\quad + x[-1,1] \cdot h[1,-1] + x[0,1] \cdot h[0,-1] + x[1,1] \cdot h[-1,-1] \\
&= \quad 0 \cdot 1 + 0 \cdot 2 + 0 \cdot 1 \\
&\quad + 0 \cdot 0 + 1 \cdot 0 + 2 \cdot 0 \\
&\quad + 0 \cdot (-1) + 4 \cdot (-2) + 5 \cdot (-1) \\
&= -13
\end{aligned}
$$



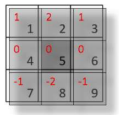
$$
\begin{aligned}
y[1,0] &= \sum_j \sum_i x[i,j] \cdot h[1-i, 0-j] \\
&= \quad x[0,-1] \cdot h[1,1] + x[1,-1] \cdot h[0,1] + x[2,-1] \cdot h[-1,1] \\
&\quad + x[0,0] \cdot h[1,0] \quad + x[1,0] \cdot h[0,0] \quad + x[2,0] \cdot h[-1,0] \\
&\quad + x[0,1] \cdot h[1,-1] + x[1,1] \cdot h[0,-1] + x[2,1] \cdot h[-1,-1] \\
&= \quad 0 \cdot 1 + 0 \cdot 2 + 0 \cdot 1 \\
&\quad + 1 \cdot 0 + 2 \cdot 0 + 3 \cdot 0 \\
&\quad + 4 \cdot (-1) + 5 \cdot (-2) + 6 \cdot (-1) \\
&= -20
\end{aligned}
$$



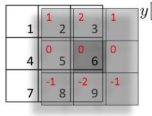
$$
\begin{aligned}
y[2,0] &= \sum_j \sum_i x[i,j] \cdot h[2-i, 0-j] \\
&= \quad x[1,-1] \cdot h[1,1] + x[2,-1] \cdot h[0,1] + x[3,-1] \cdot h[-1,1] \\
&\quad + x[1,0] \cdot h[1,0] \quad + x[2,0] \cdot h[0,0] \quad + x[3,0] \cdot h[-1,0] \\
&\quad + x[1,1] \cdot h[1,-1] + x[2,1] \cdot h[0,-1] + x[3,1] \cdot h[-1,-1] \\
&= \quad 0 \cdot 1 + 0 \cdot 2 + 0 \cdot 1 \\
&\quad + 2 \cdot 0 + 3 \cdot 0 + 0 \cdot 0 \\
&\quad + 5 \cdot (-1) + 6 \cdot (-2) + 0 \cdot (-1) \\
&= -17
\end{aligned}
$$



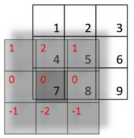
$$
\begin{aligned}
y[0,1] &= \sum_j \sum_i x[i,j] \cdot h[0-i, 1-j] \\
&= \quad x[-1,0] \cdot h[1,1] \quad + x[0,0] \cdot h[0,1] \quad + x[1,0] \cdot h[-1,1] \\
&\quad + x[-1,1] \cdot h[1,0] \quad + x[0,1] \cdot h[0,0] \quad + x[1,1] \cdot h[-1,0] \\
&\quad + x[-1,2] \cdot h[1,-1] + x[0,2] \cdot h[0,-1] + x[1,2] \cdot h[-1,-1] \\
&= \quad 0 \cdot 1 + 1 \cdot 2 + 2 \cdot 1 \\
&\quad + 0 \cdot 0 + 4 \cdot 0 + 5 \cdot 0 \\
&\quad + 0 \cdot (-1) + 7 \cdot (-2) + 8 \cdot (-1) \\
&= -18
\end{aligned}
$$



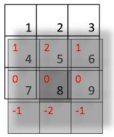
$$
\begin{aligned}
y[1,1] &= \sum_j \sum_i x[i,j] \cdot h[1-i, 1-j] \\
&= \quad x[0,0] \cdot h[1,1] \quad + x[1,0] \cdot h[0,1] \quad + x[2,0] \cdot h[-1,1] \\
&\quad + x[0,1] \cdot h[1,0] \quad + x[1,1] \cdot h[0,0] \quad + x[2,1] \cdot h[-1,0] \\
&\quad + x[0,2] \cdot h[1,-1] + x[1,2] \cdot h[0,-1] + x[2,2] \cdot h[-1,-1] \\
&= \quad 1 \cdot 1 + 2 \cdot 2 + 3 \cdot 1 \\
&\quad + 4 \cdot 0 + 5 \cdot 0 + 6 \cdot 0 \\
&\quad + 7 \cdot (-1) + 8 \cdot (-2) + 9 \cdot (-1) \\
&= -24
\end{aligned}
$$



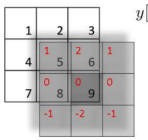
$$
\begin{aligned}
y[2,1] &= \sum_j \sum_i x[i,j] \cdot h[2-i, 1-j] \\
&= \quad x[1,0] \cdot h[1,1] \quad + x[2,0] \cdot h[0,1] \quad + x[3,0] \cdot h[-1,1] \\
&\quad + x[1,1] \cdot h[1,0] \quad + x[2,1] \cdot h[0,0] \quad + x[3,1] \cdot h[-1,0] \\
&\quad + x[1,2] \cdot h[1,-1] + x[2,2] \cdot h[0,-1] + x[3,2] \cdot h[-1,-1] \\
&= \quad 2 \cdot 1 + 3 \cdot 2 + 0 \cdot 1 \\
&\quad + 5 \cdot 0 + 6 \cdot 0 + 0 \cdot 0 \\
&\quad + 8 \cdot (-1) + 9 \cdot (-2) + 0 \cdot (-1) \\
&= -18
\end{aligned}
$$



$$
\begin{aligned}
y[0,2] &= \sum_j \sum_i x[i,j] \cdot h[0-i, 2-j] \\
&= \quad x[-1,1] \cdot h[1,1] \quad + x[0,1] \cdot h[0,1] \quad + x[1,1] \cdot h[-1,1] \\
&\quad + x[-1,2] \cdot h[1,0] \quad + x[0,2] \cdot h[0,0] \quad + x[1,2] \cdot h[-1,0] \\
&\quad + x[-1,3] \cdot h[1,-1] + x[0,3] \cdot h[0,-1] + x[1,3] \cdot h[-1,-1] \\
&= \quad 0 \cdot 1 + 4 \cdot 2 + 5 \cdot 1 \\
&\quad + 0 \cdot 0 + 7 \cdot 0 + 8 \cdot 0 \\
&\quad + 0 \cdot (-1) + 0 \cdot (-2) + 0 \cdot (-1) \\
&= 13
\end{aligned}
$$



$$
\begin{aligned}
y[1,2] &= \sum_j \sum_i x[i,j] \cdot h[1-i, 2-j] \\
&= \quad x[0,1] \cdot h[1,1] \quad + x[1,1] \cdot h[0,1] \quad + x[2,1] \cdot h[-1,1] \\
&\quad + x[0,2] \cdot h[1,0] \quad + x[1,2] \cdot h[0,0] \quad + x[2,2] \cdot h[-1,0] \\
&\quad + x[0,3] \cdot h[1,-1] + x[1,3] \cdot h[0,-1] + x[2,3] \cdot h[-1,-1] \\
&= \quad 4 \cdot 1 + 5 \cdot 2 + 6 \cdot 1 \\
&\quad + 7 \cdot 0 + 8 \cdot 0 + 9 \cdot 0 \\
&\quad + 0 \cdot (-1) + 0 \cdot (-2) + 0 \cdot (-1) \\
&= 20
\end{aligned}
$$



$$
\begin{aligned}
y[2,2] &= \sum_j \sum_i x[i,j] \cdot h[2-i, 2-j] \\
&= \quad x[1,1] \cdot h[1,1] \quad + x[2,1] \cdot h[0,1] \quad + x[3,1] \cdot h[-1,1] \\
&\quad + x[1,2] \cdot h[1,0] \quad + x[2,2] \cdot h[0,0] \quad + x[3,2] \cdot h[-1,0] \\
&\quad + x[1,3] \cdot h[1,-1] + x[2,3] \cdot h[0,-1] + x[3,3] \cdot h[-1,-1] \\
&= \quad 5 \cdot 1 + 6 \cdot 2 + 0 \cdot 1 \\
&\quad + 8 \cdot 0 + 9 \cdot 0 + 0 \cdot 0 \\
&\quad + 0 \cdot (-1) + 0 \cdot (-2) + 0 \cdot (-1) \\
&= 17
\end{aligned}
$$

C++ Implementation of 2-D Convoluion

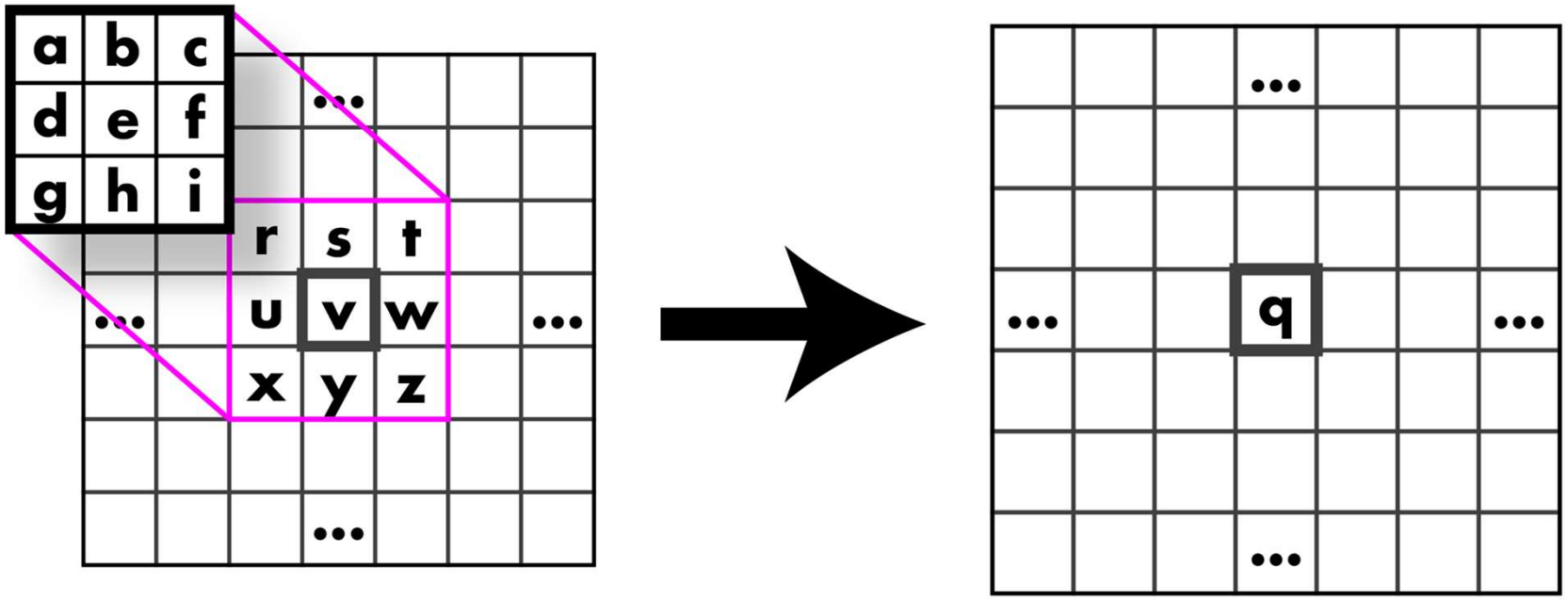
C++ Algorithm for Convolution 2D

We need 4 nested loops for 2D convolution instead of 2 loops in 1D convolution.

```
// find center position of kernel (half of kernel size)
kCenterX = kCols / 2;
kCenterY = kRows / 2;

for(i=0; i < rows; ++i)              // rows
{
    for(j=0; j < cols; ++j)          // columns
    {
        for(m=0; m < kRows; ++m)     // kernel rows
        {
            mm = kRows - 1 - m;      // row index of flipped kernel

            for(n=0; n < kCols; ++n) // kernel columns
            {
                nn = kCols - 1 - n;  // column index of flipped kernel

                // index of input signal, used for checking boundary
                ii = i + (kCenterY - mm);
                jj = j + (kCenterX - nn);

                // ignore input samples which are out of bound
                if( ii >= 0 && ii < rows && jj >= 0 && jj < cols )
                    out[i][j] += in[ii][jj] * kernel[mm][nn];
            }
        }
    }
}
```

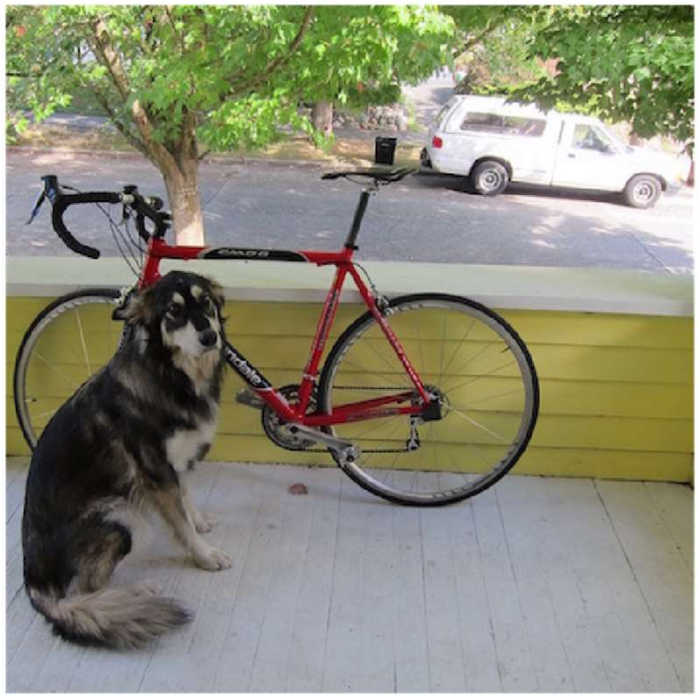
Matrix Vector product

2-D Convolution: Small-Region Filter Sliding Through Image

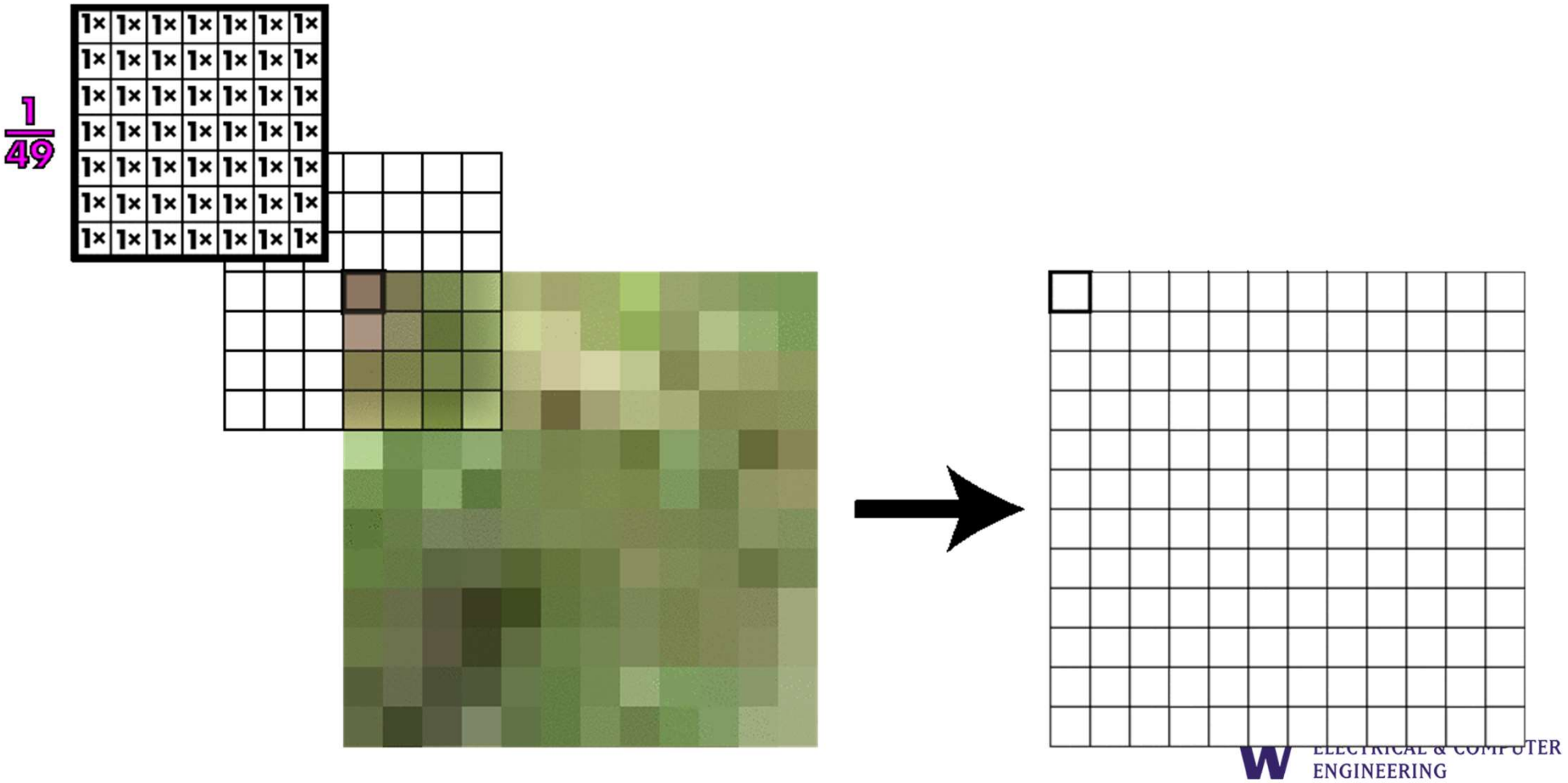


Convolutions on Larger Images

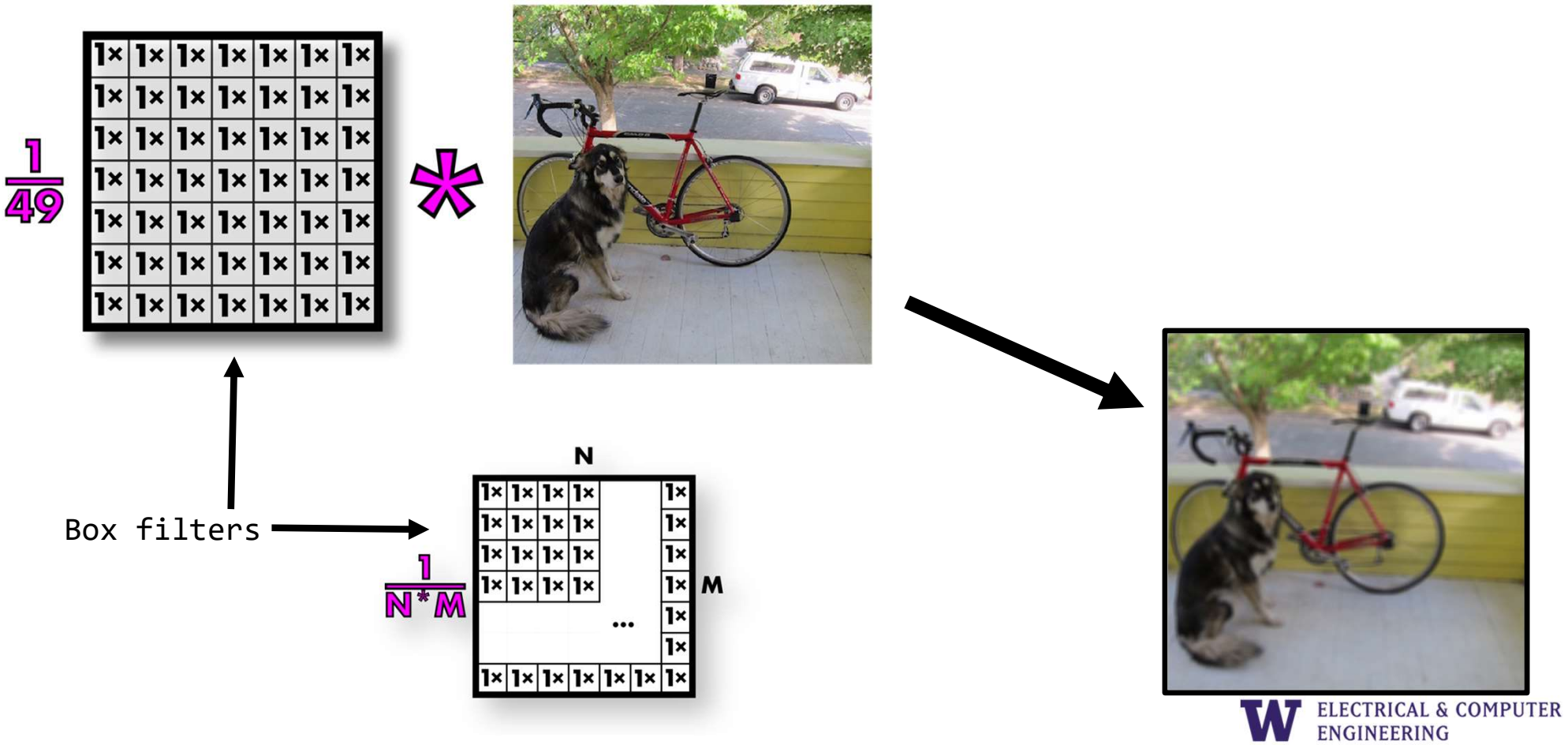
$$\frac{1}{49} \begin{bmatrix} 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 & 1 & 1 \end{bmatrix} * \quad$$



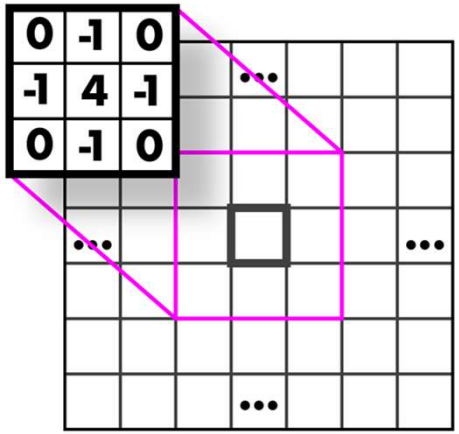
Kernel Slides Across Image



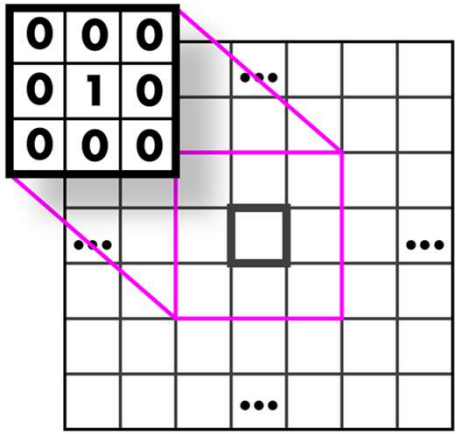
Box Filter



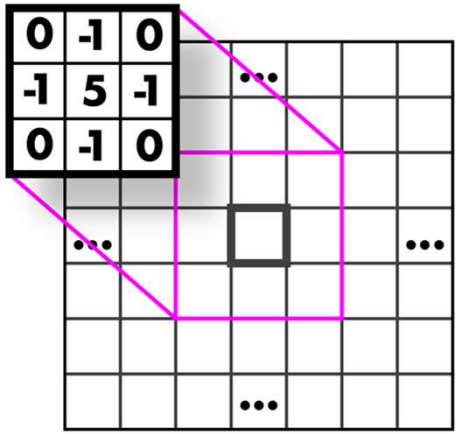
Highpass Kernel: Extract Edges



Identity Kernel: Does Nothing!

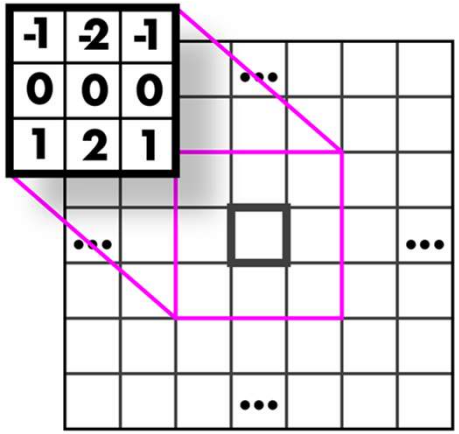


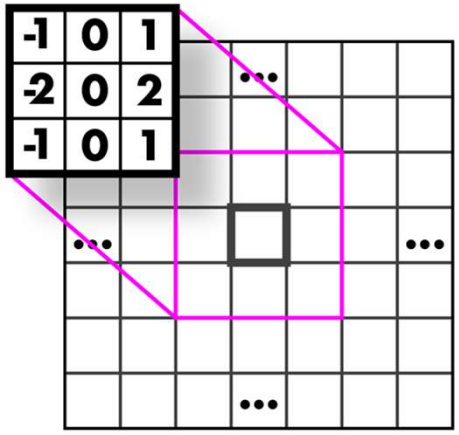
Sharpen Kernel



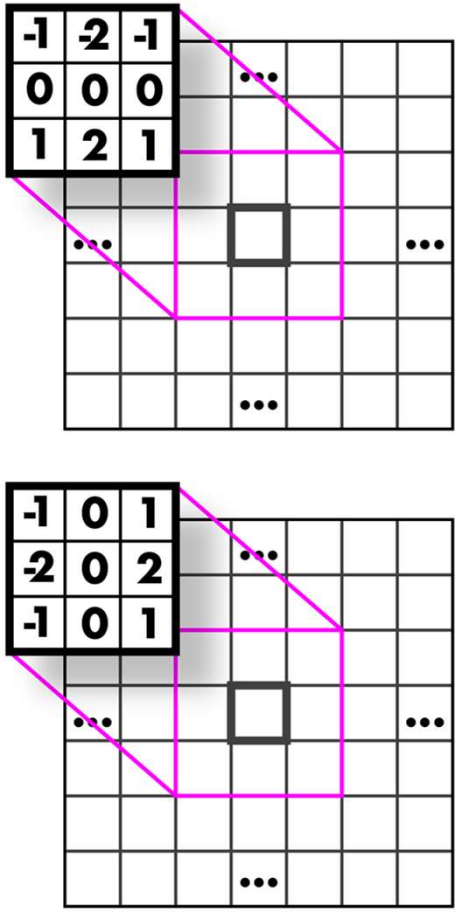
Note: sharpen = highpass + identity!

Sobel Kernels: Edges and…





Sobel Kernels: Edges and Gradient!



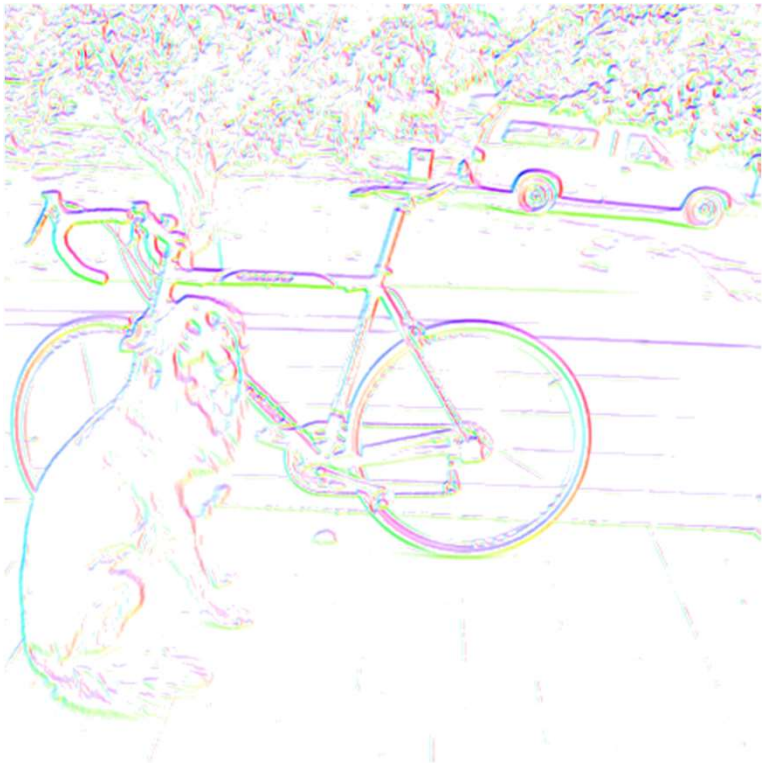


Illustration of Various Spatial Filters

$$\begin{bmatrix} 0 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 0 \end{bmatrix}$$ **Identy**

$$\begin{bmatrix} 1 & 1 & 1 \\ 1 & 2 & 1 \\ 1 & 1 & 1 \end{bmatrix}$$ **Smoothing**

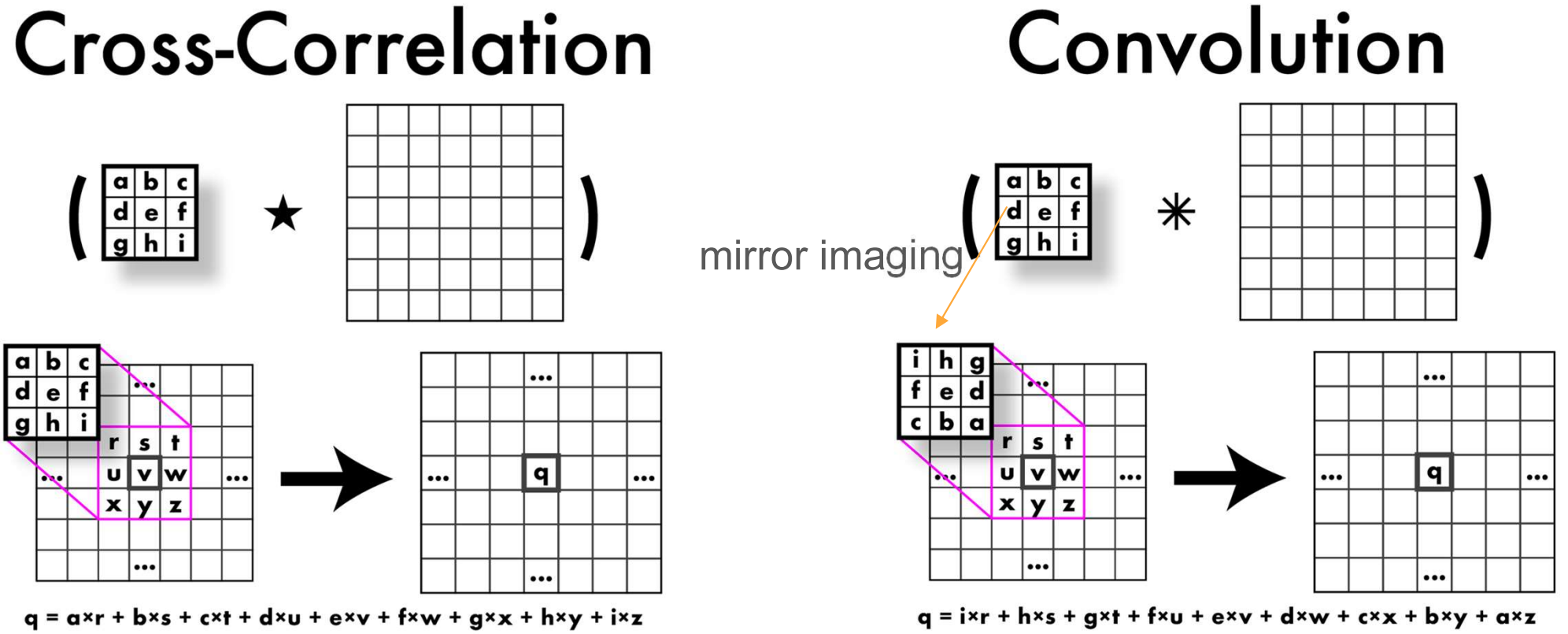
$$\begin{bmatrix} -1 & -1 & -1 \\ -1 & 9 & -1 \\ -1 & -1 & -1 \end{bmatrix}$$ **Sharpening**

$$\begin{bmatrix} 0 & 0 & -2 \\ 0 & 2 & 0 \\ 1 & 0 & 0 \end{bmatrix}$$ **Raised**

$$\begin{bmatrix} 0 & 0 & 1 \\ 0 & 0 & 0 \\ 1 & 0 & 0 \end{bmatrix}$$ **Motion-blur**

$$\begin{bmatrix} -1 & -1 & -1 \\ -1 & 8 & -1 \\ -1 & -1 & -1 \end{bmatrix}$$ **Edge detection**

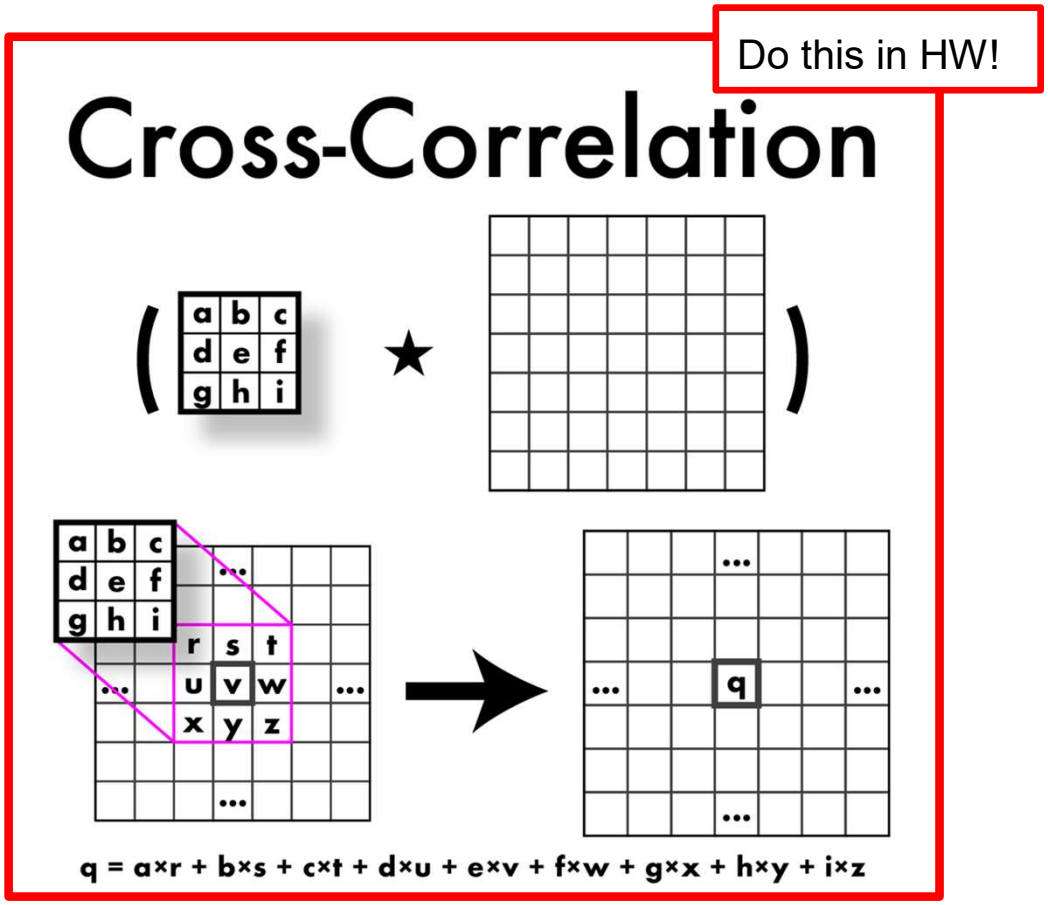
Cross-Correlation vs Convolution

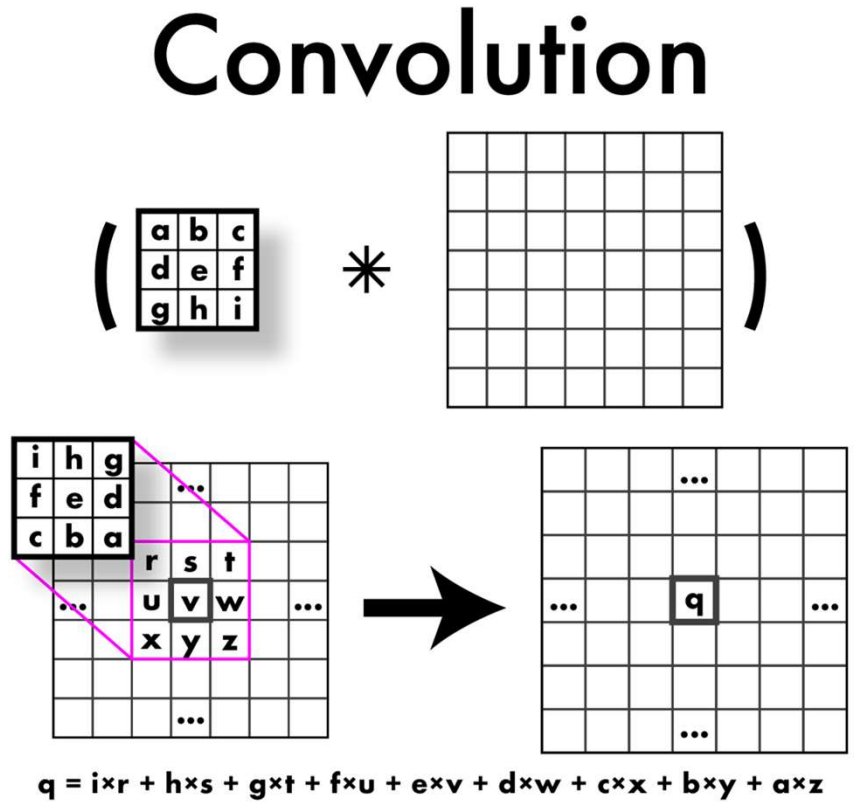


Often, deep learning like in TensorFlow does not actually distinguish these two operations

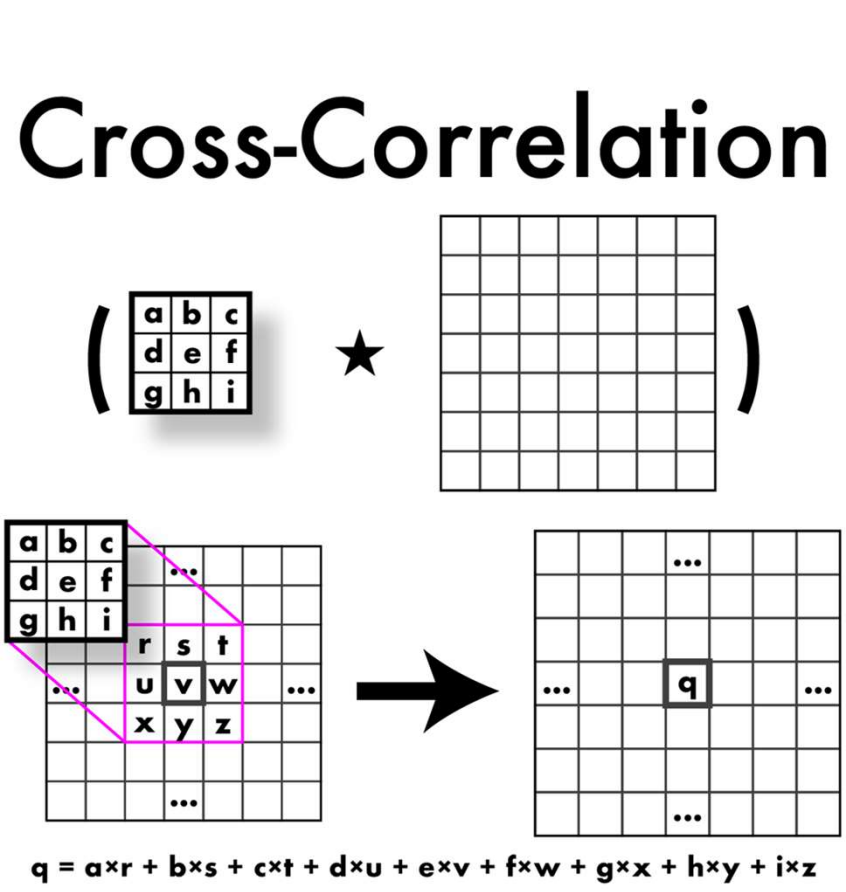
Cross-Correlation vs Convolution

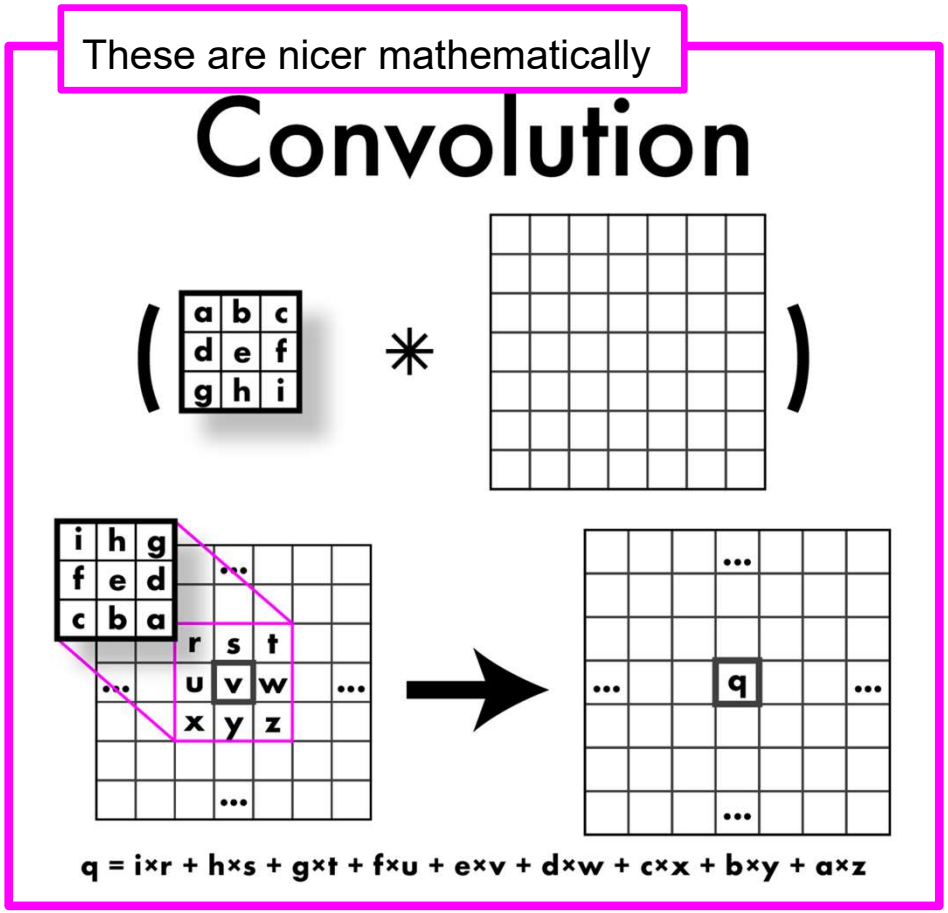
Do this in HW!





Cross-Correlation vs Convolution





Nice Mathematical Properties of Convolution

- Commutative
  - A*B = B*A
- Associative
  - A*(B*C) = (A*B)*C
- Distributes over addition
  - A*(B+C) = A*B + A*C
- Plays well with scalars
  - x(A*B) = (xA)*B = A*(xB)

So...Combining Neural Networks with Convoluions → CNN

Neural Networks

Can learn from data

- Learn to extract features that are good for a specific task
- Too many connections for images

Convolutions

Good at extracting features from images

- Static, hand designed
- Same for every task

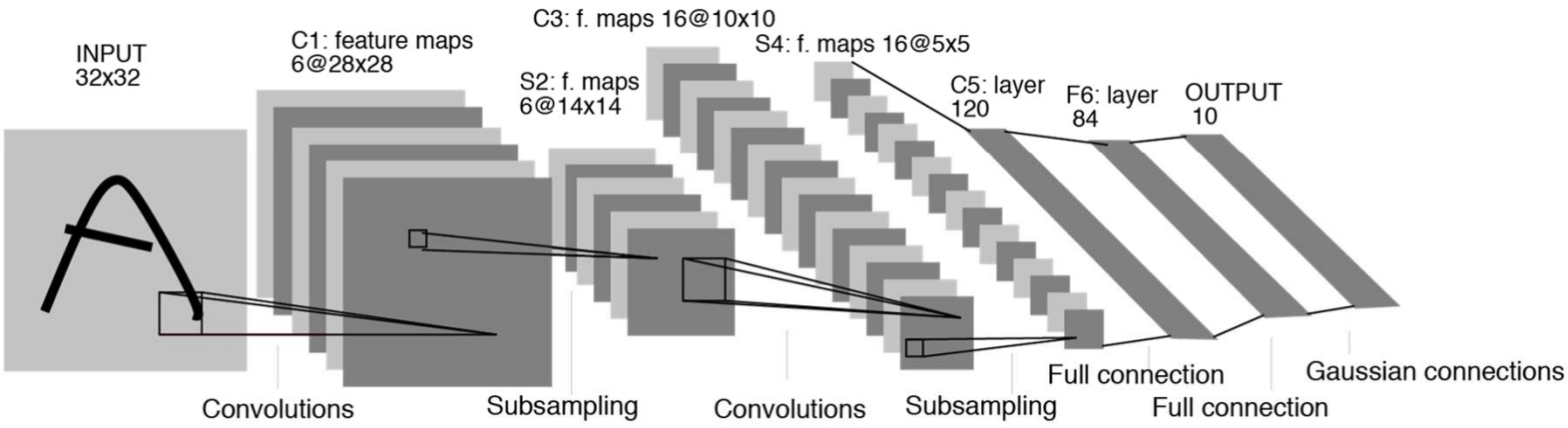
Convolutional Layer

Input: an image

Processing: convolution with multiple filters

Output: an image, # channels = # filters

Output still weighted sum of input (w/ activation)



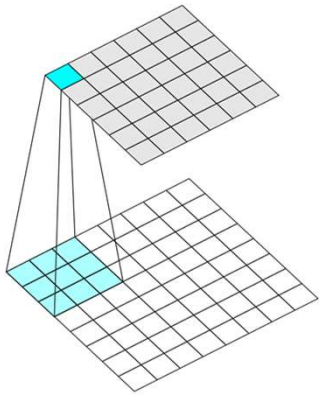
Kernel Size

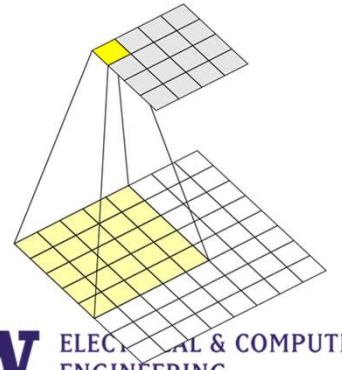
How big the filter for a layer is

Typically 1 x 1 <-> 11 x 11

1 x 1 is just linear combination of channels in previous image (no spatial processing)

Filters usually have same
number of channels as
input image.



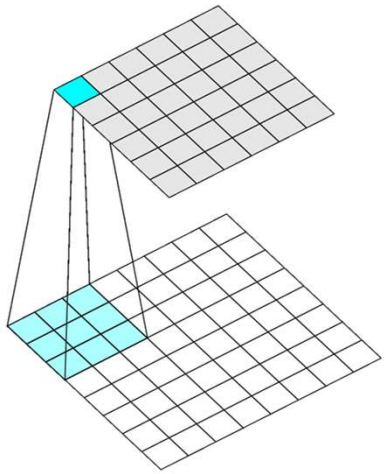


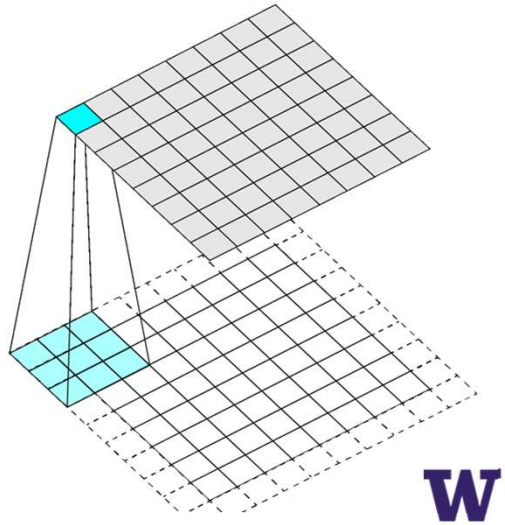
Padding

Convolutions have problems on edges

Do nothing: output a little smaller than input

Pad: add extra pixels on edge





Stride

How far to move filter between applications

We've done stride 1 convolutions up until now, approximately preserves image size

Could move filter further, downsample image

Implement Using Matrices!

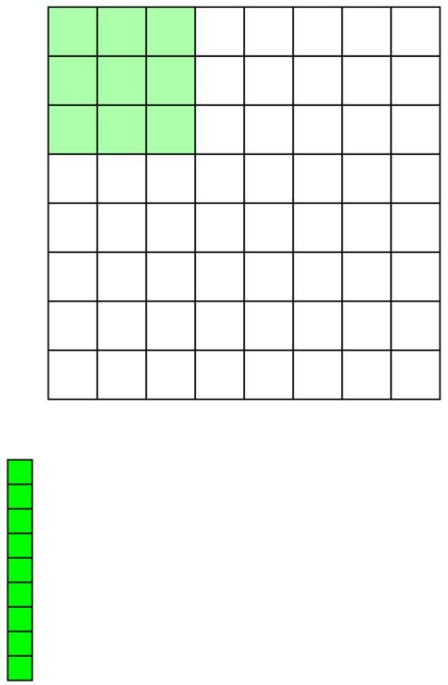
We want convolution to be matrix operations because matrices are fast. How?

Im2col: Rearrange Image

Take spatial blocks of image and put them into columns of a matrix.

Im2col handles kernel size, stride, padding, etc.

Makes matrix with all relevant pixels in the image.



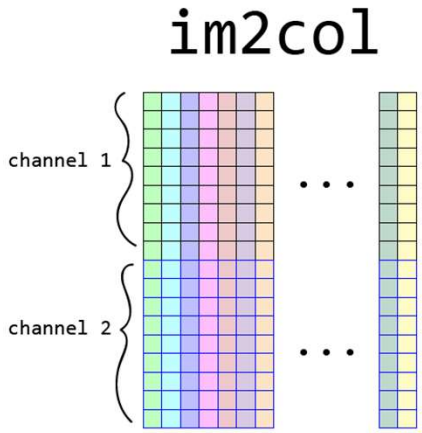
Im2col: Rearrange Image

Take spatial blocks of image and put them into columns of a matrix.

Im2col handles kernel size, stride, padding, etc.

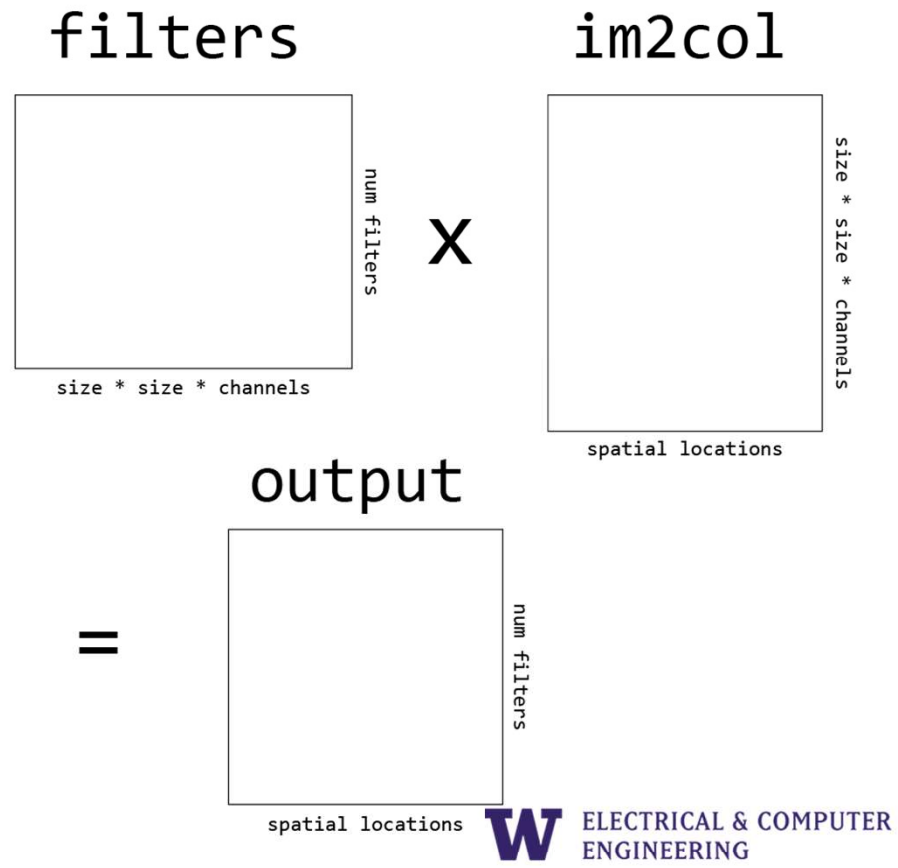
Makes matrix with all relevant pixels in the image.

Multiple channel image stacked by channel.



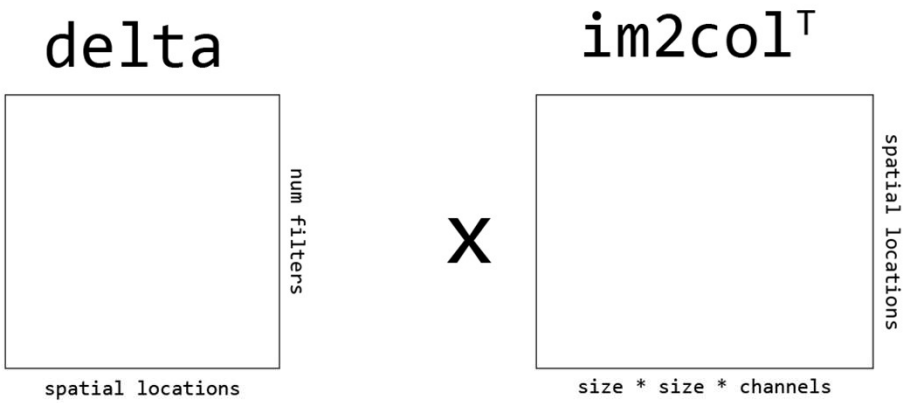
Im2col: Rearrange Image

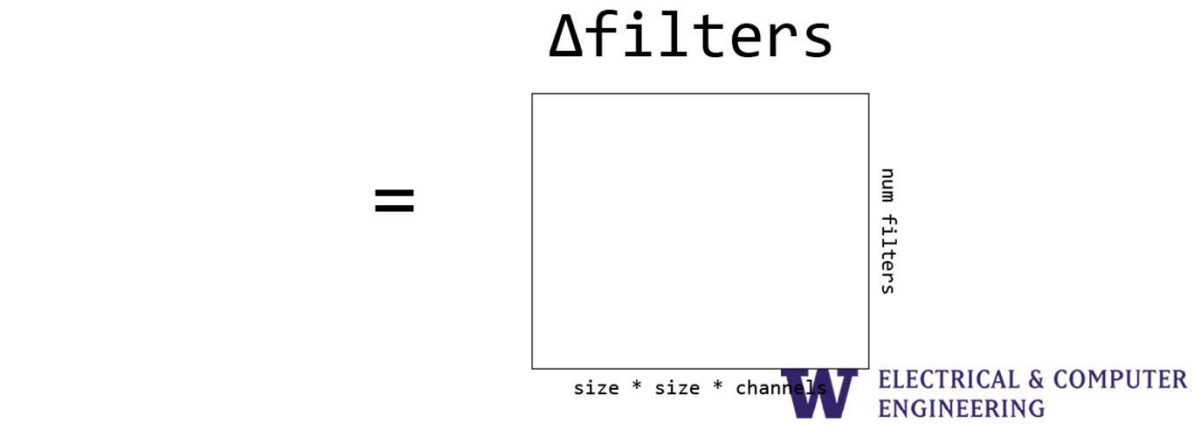
Now we just multiply by our filter matrix to do convolutions.



Im2col: Rearrange Image

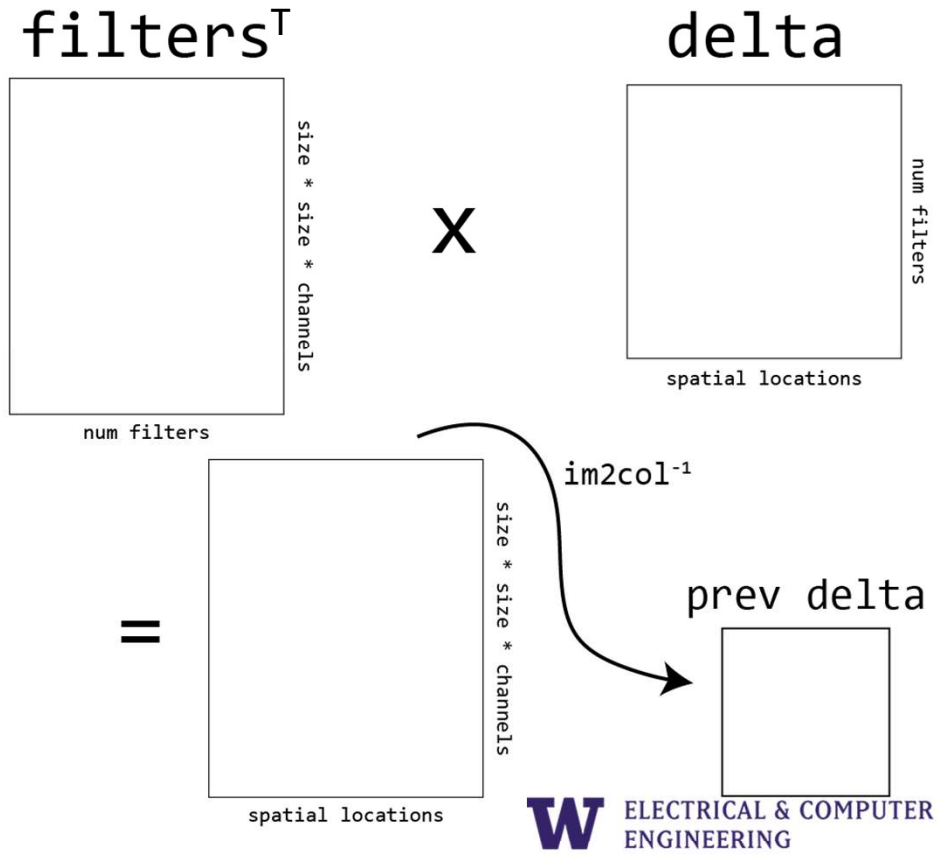
Can calculate our weight updates as well by multiplying the delta of a layer by the input.





Im2col: Rearrange Image

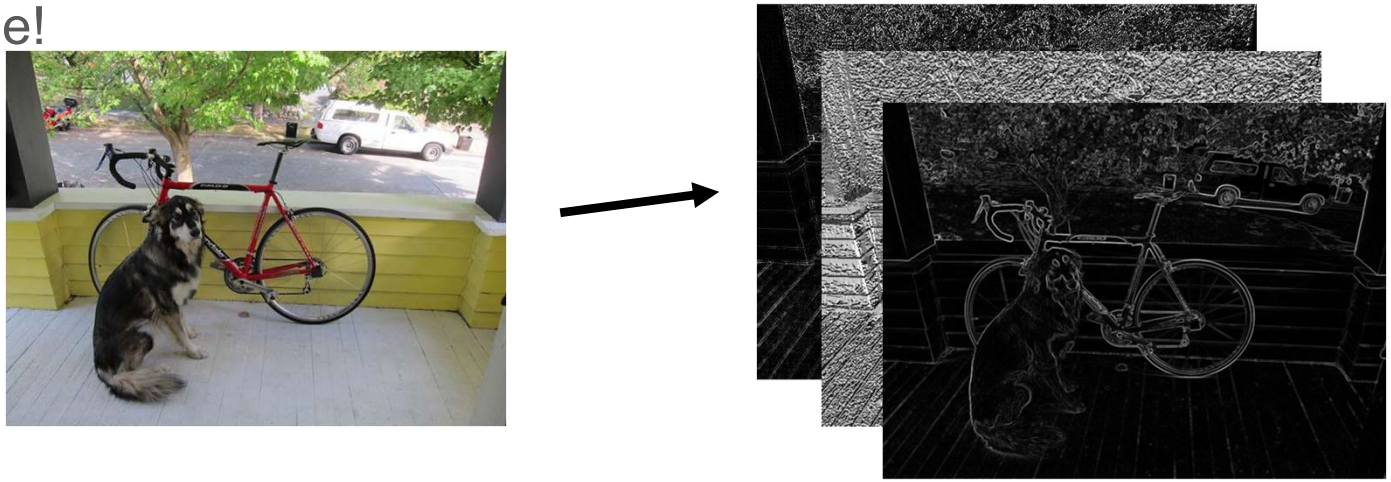
Finally, can calculate backpropagated delta by multiplying the current delta by weights and doing the reverse of im2col.



Images Are BIG

Even a 256 x 256 images has hundreds of thousands of pixels and that's considered a small image!

Convolution:



Aggregate information, maybe we don't need all of the image, can subsample without throwing away useful information

Pooling Layer

Input: an image

Processing: pool pixel values over region

Output: an image, shrunk by a factor of the stride

Hyperparameters:

- What kind of pooling? Average, mean, max, min
- How big of stride? Controls downsampling
- How big of region? Usually not much bigger than stride

Most common: 2x2 or 3x3 maxpooling, stride of 2

Maxpooling Layer, 2x2 Stride 2

-7	6	-1	3	9	9	6	-9
3	-8	0	7	10	8	-3	10
-4	2	-6	4	-7	5	5	7
-3	-9	1	8	-8	9	-1	-5
-7	10	-9	-5	9	-8	-7	10
-5	5	9	4	10	-8	7	6
-3	8	0	2	2	-3	-2	5
4	-6	7	-3	1	4	10	0

6			

Maxpooling Layer, 2x2 stride 2

-7	6	-1	3	9	9	6	-9
3	-8	0	7	10	8	-3	10
-4	2	-6	4	-7	5	5	7
-3	-9	1	8	-8	9	-1	-5
-7	10	-9	-5	9	-8	-7	10
-5	5	9	4	10	-8	7	6
-3	8	0	2	2	-3	-2	5
4	-6	7	-3	1	4	10	0

6	7		

Maxpooling Layer, 2x2 Stride 2

-7	6	-1	3	9	9	6	-9
3	-8	0	7	10	8	-3	10
-4	2	-6	4	-7	5	5	7
-3	-9	1	8	-8	9	-1	-5
-7	10	-9	-5	9	-8	-7	10
-5	5	9	4	10	-8	7	6
-3	8	0	2	2	-3	-2	5
4	-6	7	-3	1	4	10	0

6	7	10	10
2			

Maxpooling Layer, 2x2 Stride 2

-7	6	-1	3	9	9	6	-9
3	-8	0	7	10	8	-3	10
-4	2	-6	4	-7	5	5	7
-3	-9	1	8	-8	9	-1	-5
-7	10	-9	-5	9	-8	-7	10
-5	5	9	4	10	-8	7	6
-3	8	0	2	2	-3	-2	5
4	-6	7	-3	1	4	10	0

6	7	10	10
2	8	9	7
10	9	10	10
8	7	4	10

Pooling Layer

Input: an image

Processing: pool pixel values over region

Output: an image, shrunk by a factor of the stride

Hyperparameters:

- What kind of pooling? Average, mean, max, min
- How big of stride? Controls downsampling
- How big of region? Usually not much bigger than stride

Most common: 2x2 or 3x3 maxpooling, stride of 2

(Fully) Connected Layer

The standard neural network layer where every input neuron connects to every output neuron

Often used to go from image feature map -> final output or map image features to a single vector

Eliminates spatial information

Convnet Building Blocks

Convolutional layers:

- Connections are convolutions
- Used to extract features

Pooling layers:

- Used to downsample feature maps, make processing more efficient
- Most common: maxpool, avgpool sometimes used at end

Connected layers:

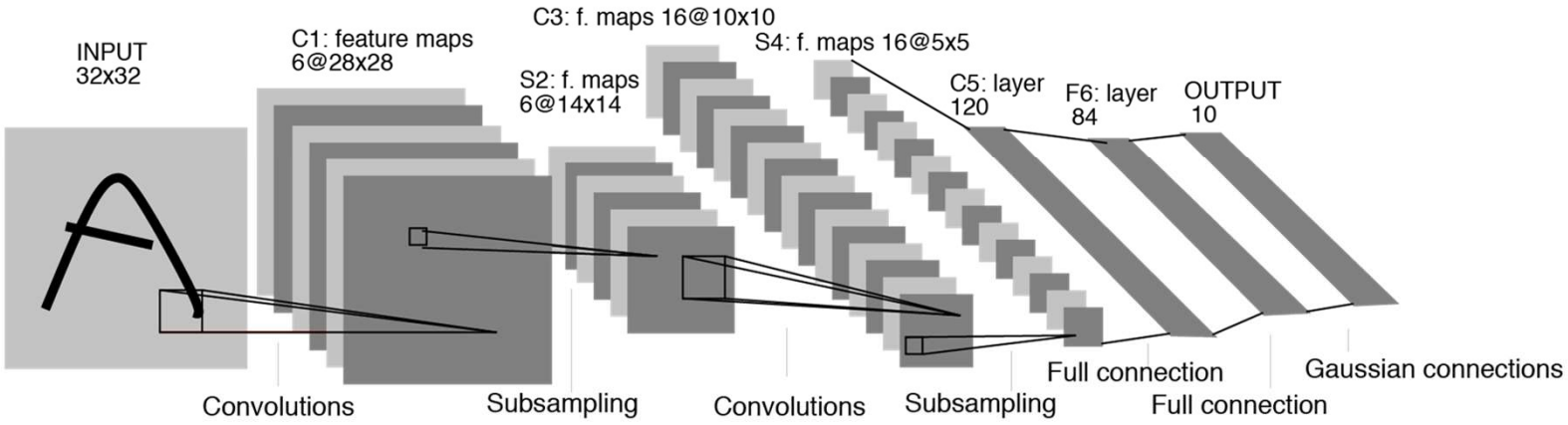
- Often used as last layer, to map image features -> prediction
- No spatial information
- Inefficient: lots of weights, no weight sharing

LeNet: First Convnet for Images*

99% accuracy on MNIST (Yann LeCun 1998)

Has all elements of modern convnet

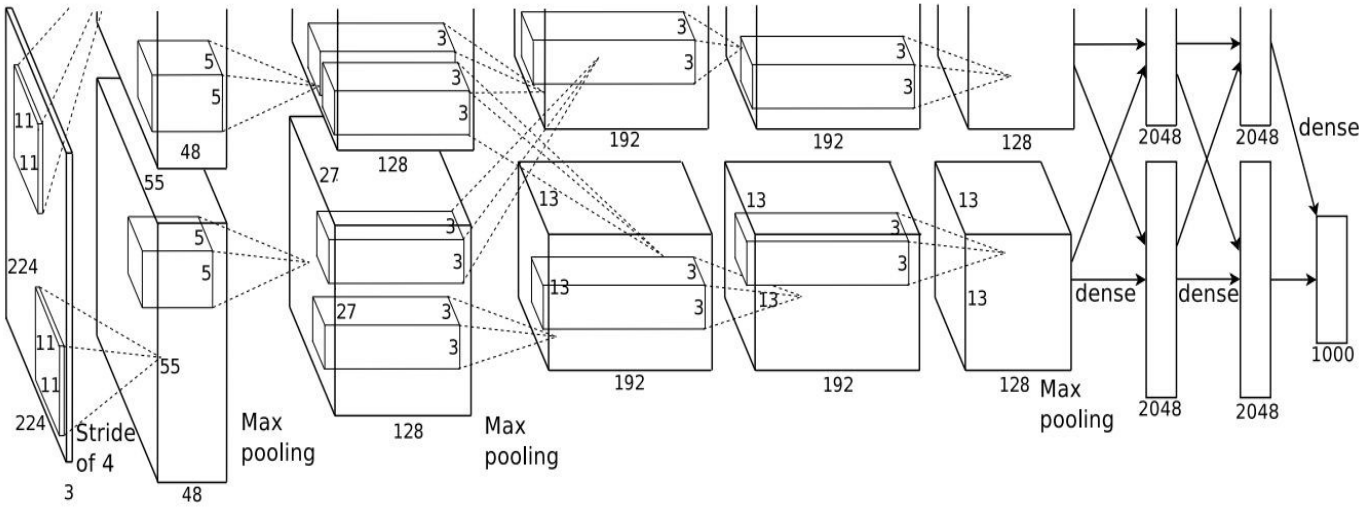
- Convolutions, maxpooling, fully connected layers
- Logistic activations after pooling layers (nowadays use RELU)
- Weight updates through backpropagation



MINIST Handwriting Recognition: 50,000 images of handwriting; 28 x 28 x 1 (grayscale); Numbers 0-9

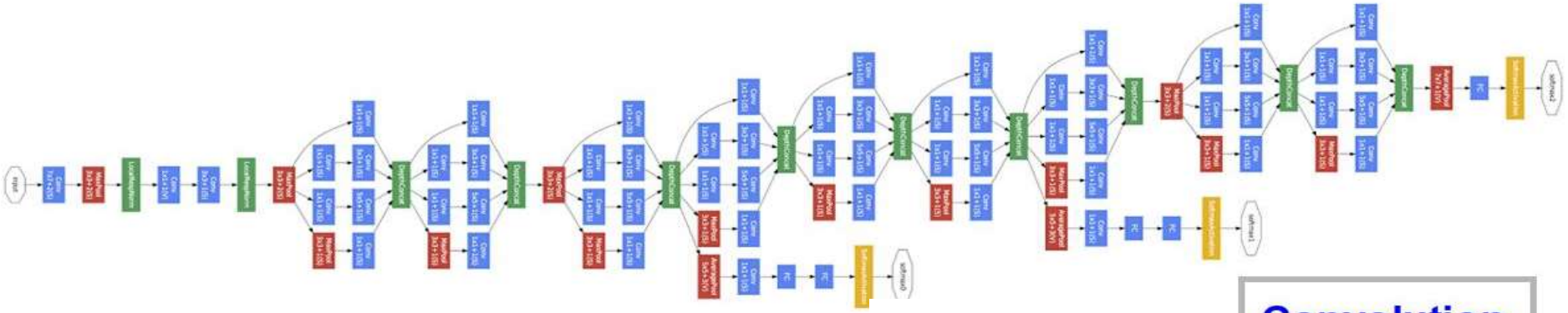
*probably? Maybe neocognitron but not trained w/ backprop Fukushima, Kunihiko (1980). "Neocognitron: A Self-organizing Neural Network Model for a Mechanism of Pattern Recognition Unaffected by Shift in Position" (PDF). Biological Cybernetics. 36 (4): 193–202.

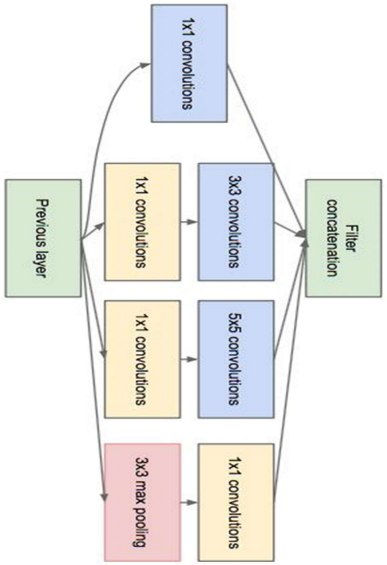
AlexNet*


224
224
3
11
11
Stride
of 4
48
55
5
5
48
Max
pooling
27
128
3
3
3
128
Max
pooling
13
192
3
3
3
192
3
3
3
192
13
128
Max
pooling
dense
2048
2048
dense
dense
2048
2048
1000


params
AlexNet
FLOPs
4M
FC 1000
4M
16M
FC 4096 / ReLU
16M
37M
FC 4096 / ReLU
37M
Max Pool 3x3s2
442K
Conv 3x3s1, 256 / ReLU
74M
1.3M
Conv 3x3s1, 384 / ReLU
112M
884K
Conv 3x3s1, 384 / ReLU
149M
Max Pool 3x3s2
Local Response Norm
307K
Conv 5x5s1, 256 / ReLU
223M
Max Pool 3x3s2
Local Response Norm
35K
Conv 11x11s4, 96 / ReLU
105M

GoogLeNet



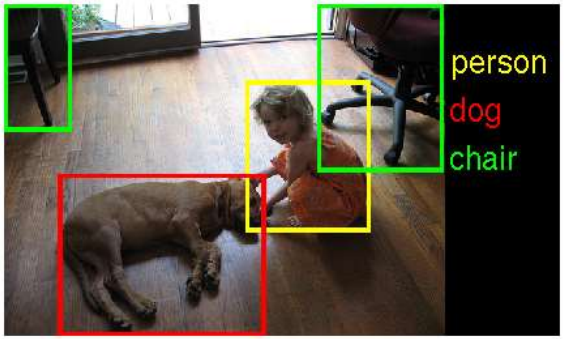


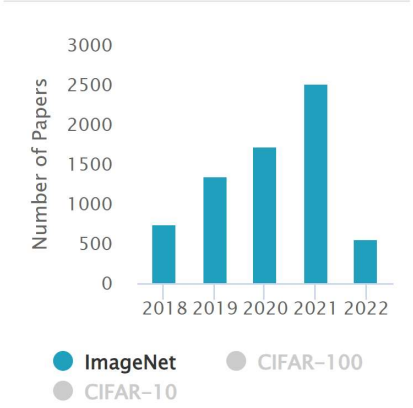
Convolution
Pooling
Softmax
Other

Christian Szegedy, 2014

ImageNet: Really Big Image Dataset

- The **ImageNet** dataset contains 14,197,122 annotated images according to the WordNet hierarchy. Since 2010 the dataset is used in the ImageNet Large Scale Visual Recognition Challenge (ILSVRC), a benchmark in image classification and object detection.
- Total number of non-empty WordNet synsets: 21841
- Total number of images: 14,197,122
- Number of images with bounding box annotations: 1,034,908
- Number of synsets with SIFT features: 1000
- Number of images with SIFT features: 1.2 million





ELECTRICAL & COMPUTER ENGINEERING

Typically in a *Deep* Learning Network

- Most *feature extraction* is done in the model
- Multiple layers with deeper layers having more semantically meaningful content
- Has figures of architectures that look like: