University of Minnesota
Driven to Discover℠

# EE 5561: Image Processing and Applications

## Lecture 18

Mehmet Akçakaya

# Recap of Last Lecture

- Backpropagation
  - Concept & examples for clarification
  - In practice: All done automatically in the learning framework
- Practical training points
  - Optimization algorithms
  - Hyperparameters/initialization
  - Regularization

- Today: Convolutional neural networks (CNNs)

# Feed-Forward Networks

- So far, feed-forward networks

$$f_{\boldsymbol{\theta}}(\mathbf{x}) = \eta\Bigg(\mathbf{W}^{[n]}\Big(\ldots\eta\Big(\mathbf{W}^{[2]}\eta(\mathbf{W}^{[1]}\mathbf{x} + \mathbf{b}^{[1]})\big) + \mathbf{b}^{[2]}\Big)\cdots + \mathbf{b}^{[n]}\Big)\Bigg)$$

  - Here $\mathbf{W}^{[k]}$ is a matrix, with no particular structure
  - Per neuron we have

$$a = \eta\big(\mathbf{w}^T\mathbf{x} + b\big)$$

    - # parameters to learn = size of image
    - Does not work with changing input sizes
    - Not clear how it behaves under image shifts
    - ….

# CNNs

- Instead we will be interested in a special case

$$a = \eta\left(\mathbf{w} * \mathbf{x} + b\right)$$

  - i.e. implemented via convolutions
- Why convolutional layers?
- 4 important ideas:
  - Sparse weights
  - Parameter sharing
  - Translation invariance/equivariance
  - Works with inputs of different sizes

# CNNs

- Instead we will be interested in a special case

$$a = \eta\big(\mathbf{w} * \mathbf{x} + b\big)$$

  - i.e. implemented via convolutions
- In general, convolutions are implemented as a sliding dot-product (~cross-correlation)
  - As we saw before, no flipping the kernel (unlike EE 3015)
- We do convolutions in multiple dimensions (e.g. 3D)

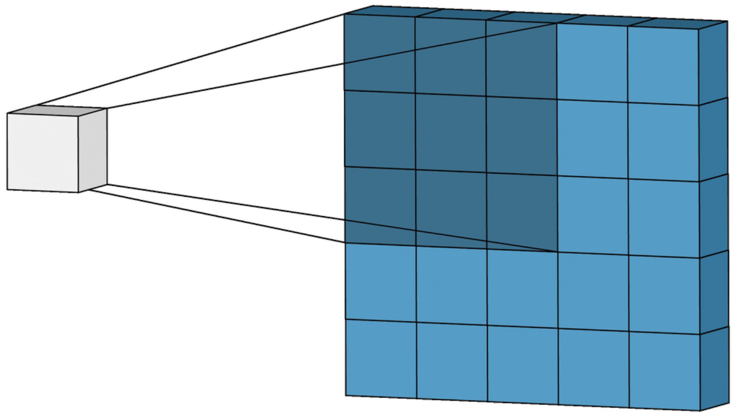
$n_x \times n_y \times n_0$ kernel

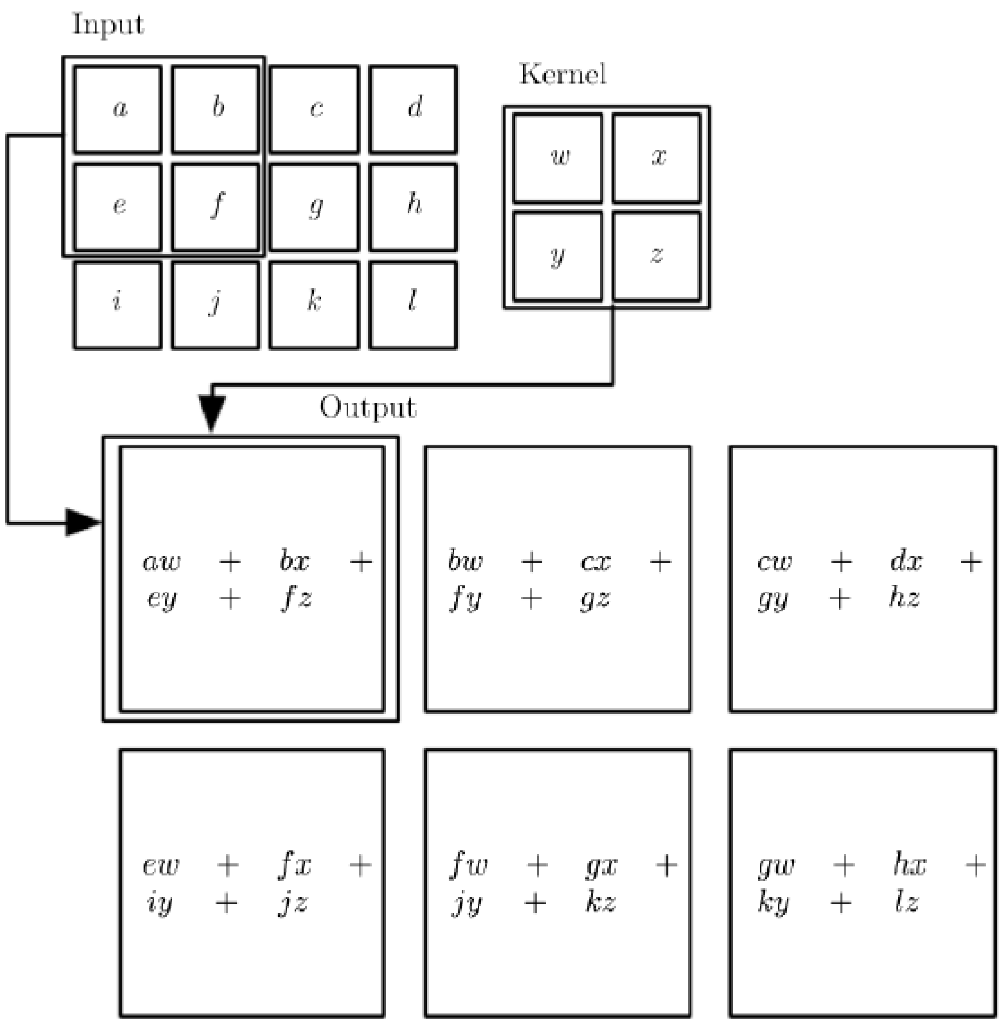
$$\begin{aligned} S(i,j,k) &= I(i,j,k) * H(i,j,k) \\ &= \sum_{l=1}^{n_0} \sum_{n=1}^{n_y} \sum_{m=1}^{n_x} I(i+m, j+n, l) K(m,n,l) \end{aligned}$$

  - Usually the non-spatial dimension sizes match between input and kernel (*k* dimension here)
  - e.g. If *I* is an RGB image (3 input channels), then we will convolve it with $n_x \times n_y \times 3$ kernels

# Convolutions in CNNs

Pictorially (from Lecture 4)





# Convolutions in CNNs

Input (5x5x2)

| | | | | |
|---|---|---|---|---|
| 1 | 2 | 1 | 2 | 0 |
| 2 | 2 | 0 | 0 | 2 |
| 2 | 1 | 0 | 2 | 1 |
| 0 | 2 | 0 | 2 | 0 |
| 1 | 1 | 1 | 0 | 1 |

| | | | | |
|---|---|---|---|---|
| 2 | 2 | 1 | 1 | 0 |
| 3 | 0 | 0 | 1 | 1 |
| 2 | 0 | 0 | 1 | 1 |
| 1 | 2 | 0 | 1 | 1 |
| 1 | 1 | 2 | 0 | 2 |

Filter $W_0$ (3x3x2)

| | | |
|---|---|---|
| 1 | -1 | -1 |
| -1 | 1 | 0 |
| 0 | 0 | 1 |

| | | |
|---|---|---|
| 1 | 0 | 0 |
| -1 | 0 | 0 |
| 1 | 1 | -1 |

Filter $W_1$ (3x3x2)

| | | |
|---|---|---|
| 1 | 2 | 1 |
| 2 | 4 | 2 |
| 1 | 2 | 1 |

Gaussian filter

| | | |
|---|---|---|
| 0 | -1 | 0 |
| -1 | 4 | -1 |
| 0 | -1 | 0 |

Laplace filter

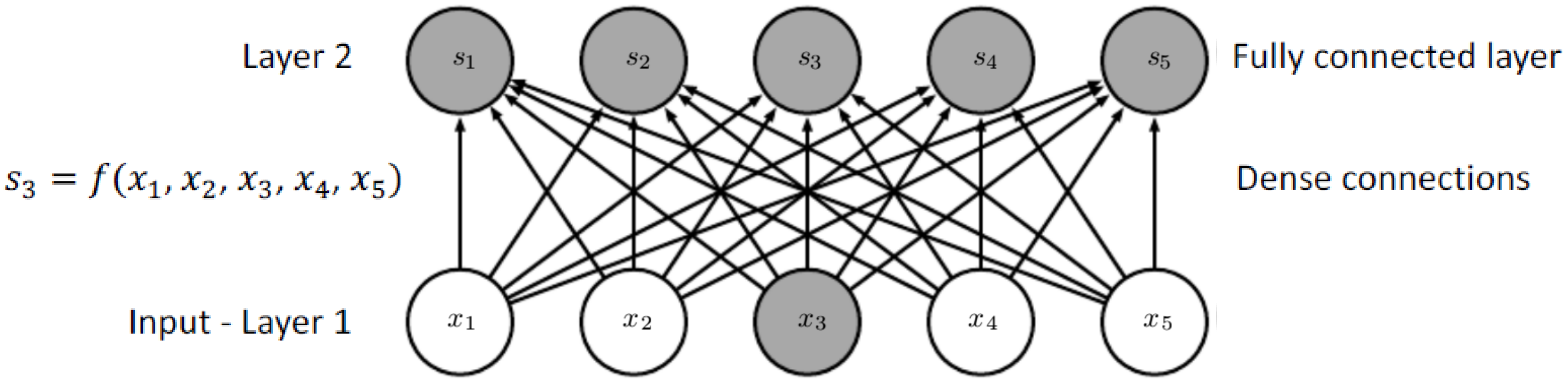
Output ( 3 x 3 x 2 )

| | | |
|---|---|---|
| -1 | 0 | 3 |
| 3 | 6 | 1 |
| 5 | 0 | 0 |

| | | |
|---|---|---|
| 17 | 11 | 15 |
| 14 | 11 | 17 |
| 22 | 9 | 17 |

# Why Convolutional Layers?

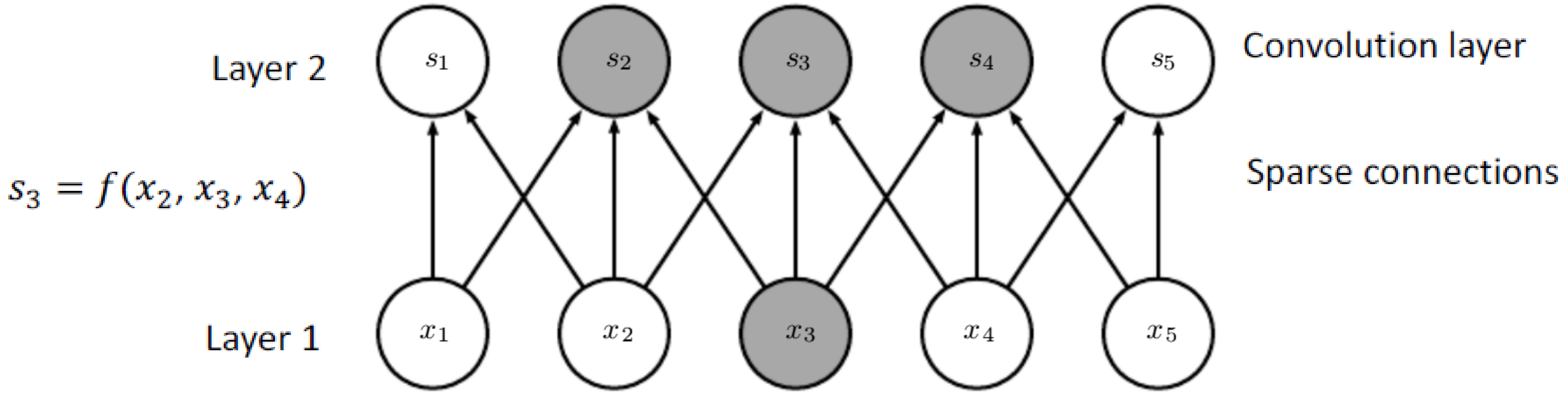
- Back to why:
  - Sparse weights
  - With fully-connected layers, we have




Goodfellow, "Deep Learning," Chapter 9

# Why Convolutional Layers?

- Back to why:
  - Sparse weights
  - With CNNs, we have




Goodfellow, "Deep Learning," Chapter 9

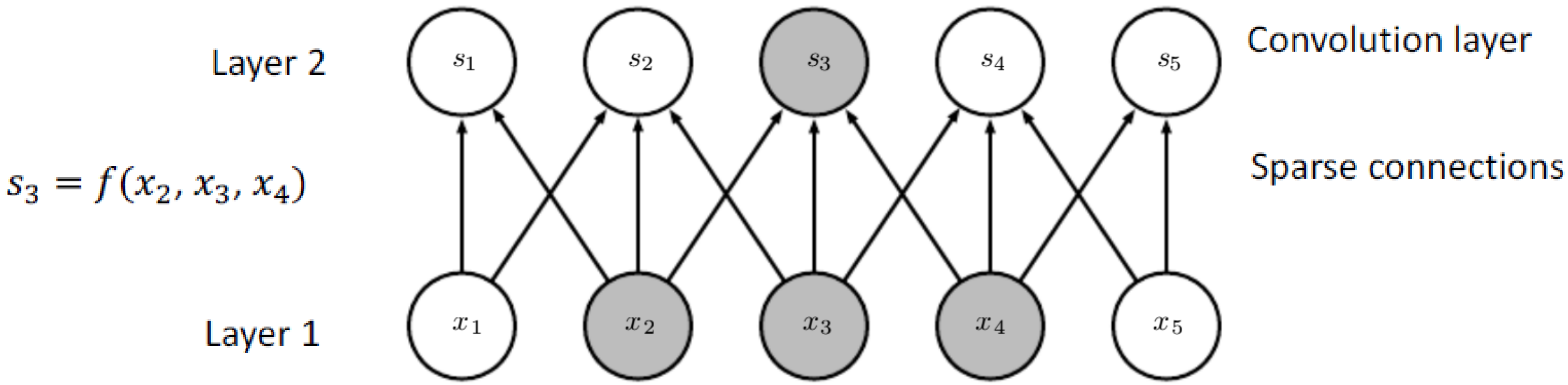
# Why Convolutional Layers?

- Back to why:
  - Sparse weights
  - Why does it matter?
  - A typical chest x-ray is ~3000×2000 pixels
    - Dense connections: 6,000,000 connections per node!
    - Sparse connections, e.g. k = 3×3 connections per node

| Memory | # operations |
| --- | --- |
| ↑ | O(n) – image size |
| ↓ | O(k) |

# Receptive Field

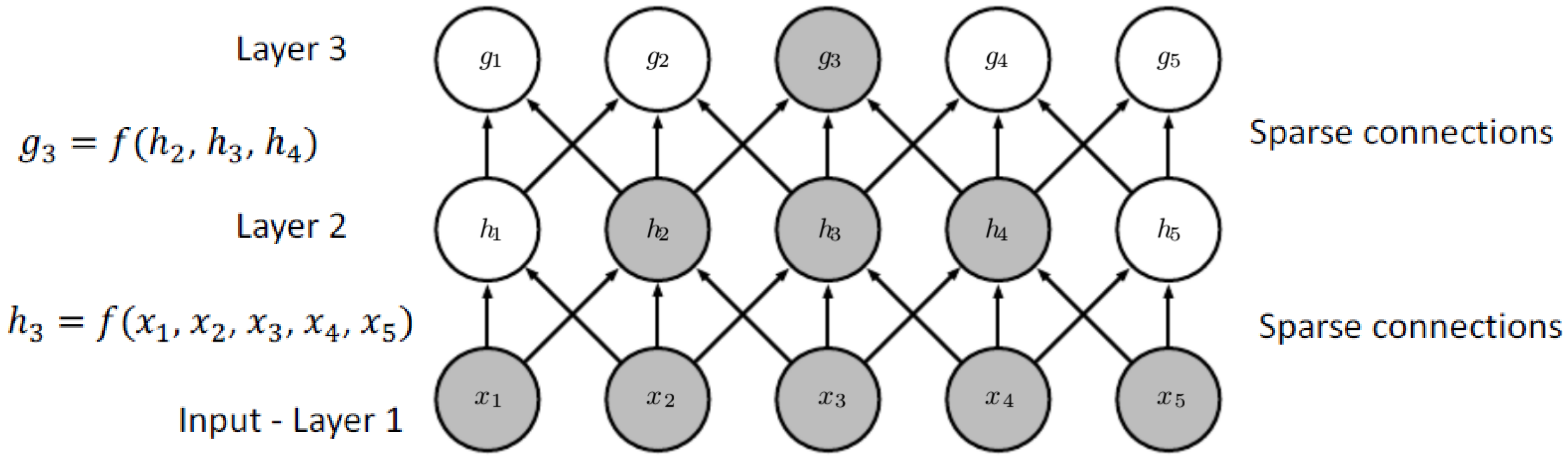
- By not having dense connections, neurons in later layers receive "stimulus" or input from only a subset of neurons
- Called the receptive field




Goodfellow, "Deep Learning," Chapter 9

# Receptive Field

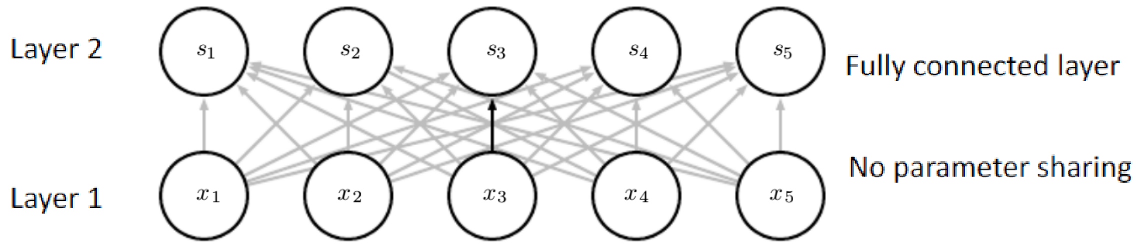
- By not having dense connections, neurons in later layers receive "stimulus" or input from only a subset of neurons
- Called the receptive field




- Hence deeper layers indirectly interact with a larger portion of the input

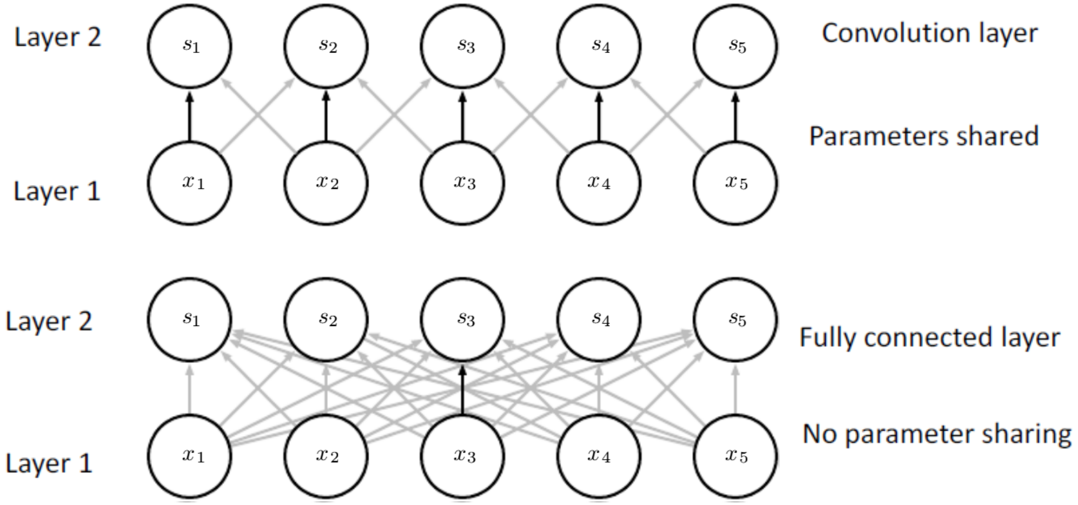
# Why Convolutional Layers?

- Back to why:
  - Parameter sharing
  - In a fully-connected layer, each parameter is used once when computing the output and never revisited



# Why Convolutional Layers?

- Back to why:
  - Parameter sharing
  - In a fully-connected layer, each parameter is used once when computing the output and never revisited
  - In contrast, each parameter of the convolutional kernel is used at every point of the input



# Why Convolutional Layers?

- Back to why:
  - Parameter sharing
  - In a fully-connected layer, each parameter is used once when computing the output and never revisited
  - In contrast, each parameter of the convolutional kernel is used at every point of the input
  - i.e. We maintain the same filter/feature detector used in one part of the input data across other sections of the input
  - Also more efficient

# Why Convolutional Layers?

- Back to why:
  - Parameter sharing efficiency

Kernel

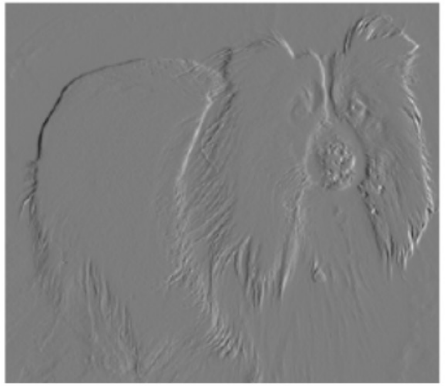
| | |
|---|---|
| -1 | 1 |

vertically oriented
edge detection

Original Image

280 px
320 px

Filtered Image



| | Convolutions | Matrix multiplications |
|---|---|---|
| Floating point operations | 319×280×3 | 320×280×319×280 |

# Why Convolutional Layers?

- Back to why:
  - Equivariance/invariance to translation
  - Definition: A function *f* is equivariant to some transformation *T* if

$$f(T(x)) = T(f(x))$$

    - e.g. *f:* DFT, *T*: rotation
  - Side note: In signals and systems, we called this "invariance", but invariance is often defined as

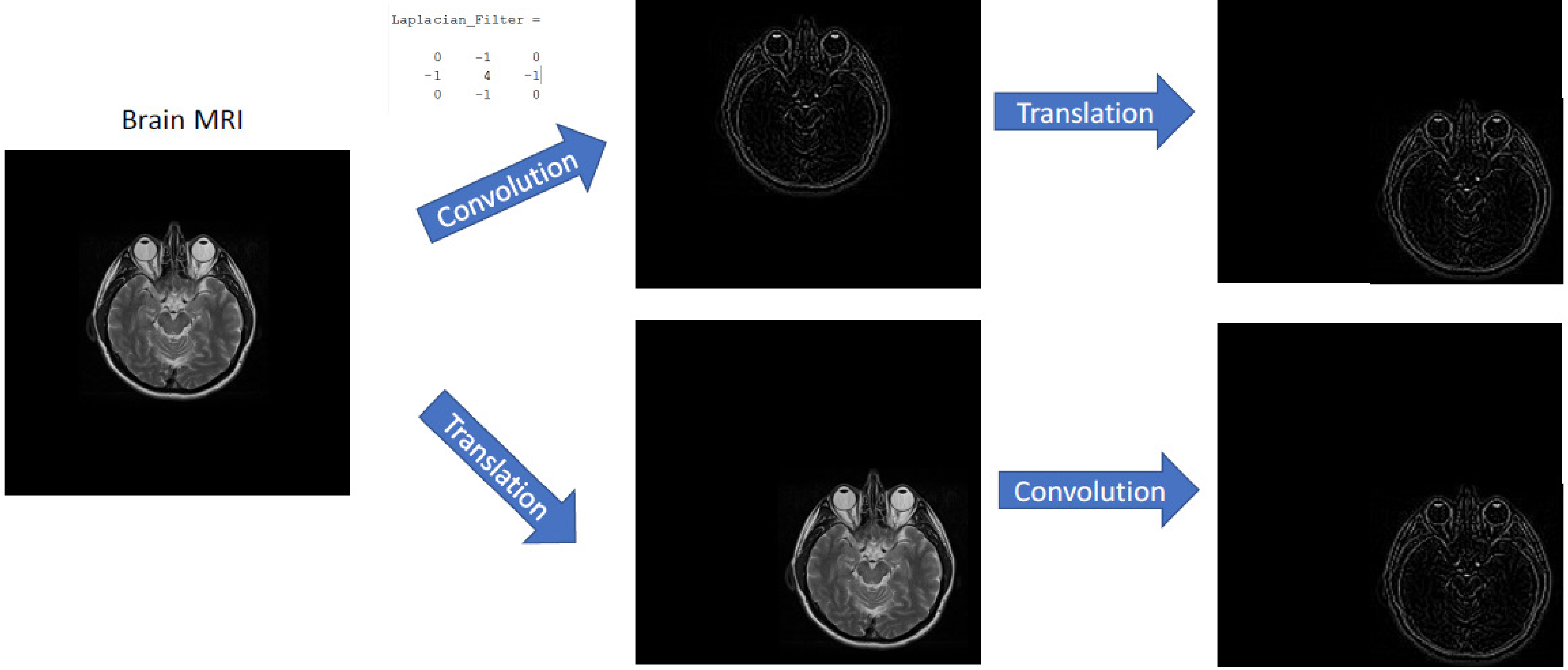
$$f(T(x)) = f(x)$$

    - e.g. *f*: max value of the image, *T*: translation
  - Convolutions are shift/translation equivariant

# Why Convolutional Layers?

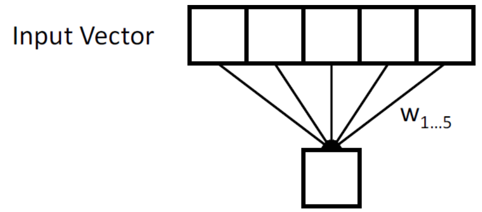
- Back to why:
  - Equivariance to translation

$$f(T(x)) = T(f(x))$$

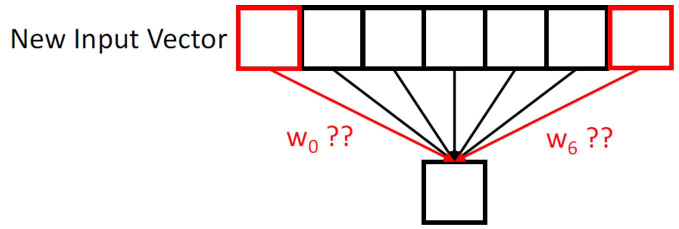


# Why Convolutional Layers?

- Back to why:
  - Can handle inputs of variable sizes
  - Consider a fully-connected layer
    - Trained for a specific input size

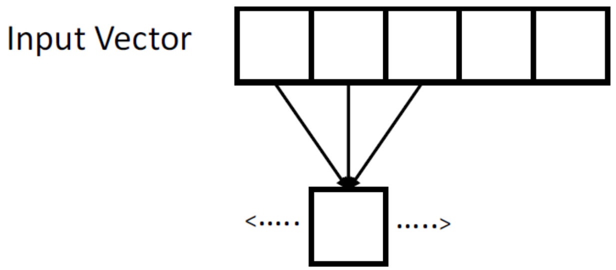



    - If a new input vector has a different size, what happens?
    - Need to retrain

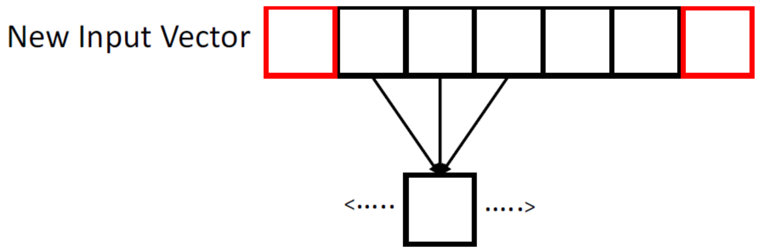


# Why Convolutional Layers?

- Back to why:
  - Can handle inputs of variable sizes
  - This is not an issue for a convolutional layer
    - If trained on input of one size



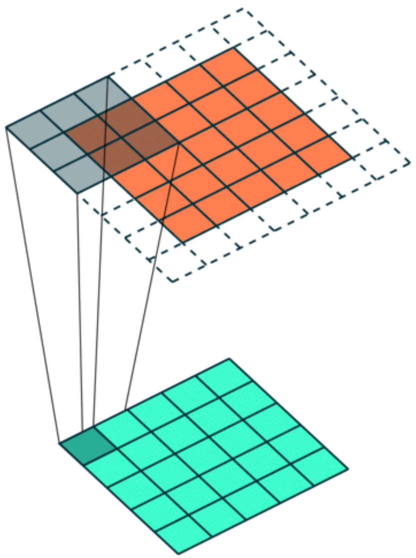

    - Can still be applied to input of different size



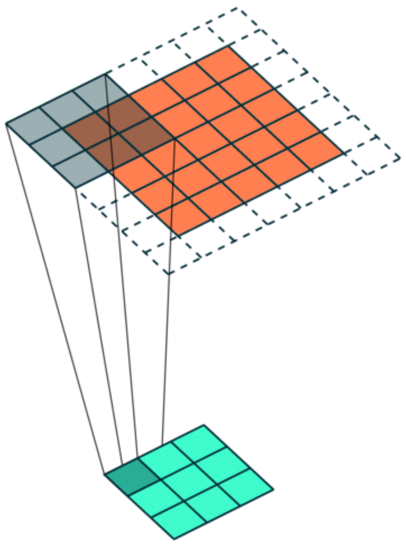
# Variations on Convolutions

- So far we considered convolutions as a sliding window
- In certain cases, we may slide the window by more than one pixel
  - Called stride
  - Computational load & output size reduced

Stride = 1



Stride = 2



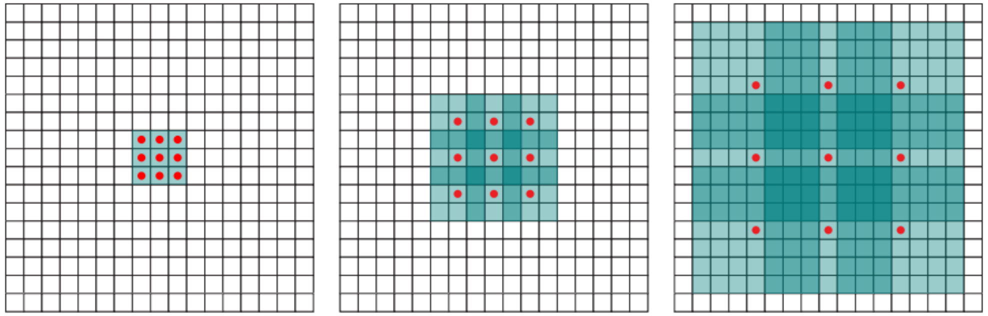
https://m-alcu.github.io/blog/2018/01/13/neural-layers/

# Variations on Convolutions

- Boundary processing
  - Did this before
  - If nothing is done, image size will shrink
  - Zeropadding is commonly used
    - For filter size $K$ (assumed to be odd), zeropad by $(K\text{-}1)/2$
  - Periodic/mirror boundary processing are other options

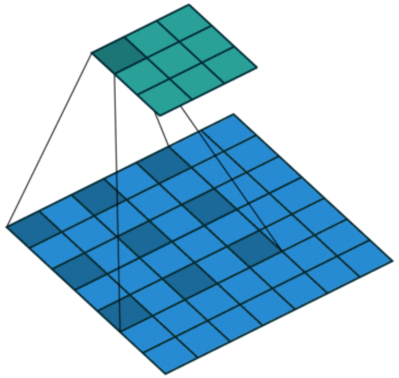
# Variations on Convolutions

- Dilated convolutions
  - Increase the spacing between the values in a kernel



  - Larger receptive field at the same computational cost

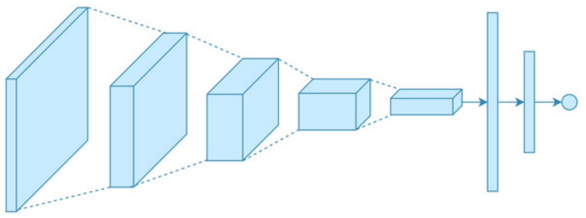
Dilation Factor = 2

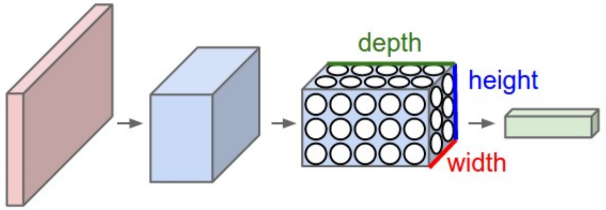


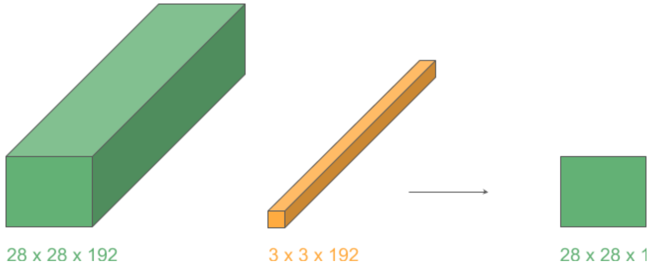
https://arxiv.org/pdf/1511.07122.pdf

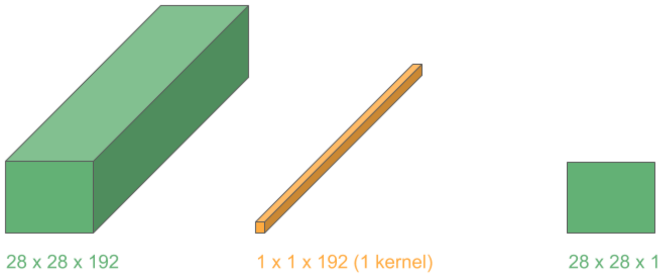
# Visualizing Convolutional Layers

- Often we will see figures that look like
  - These inherently visualize certain parameters
  - In general for a convolutional layer, we will talk about height, width, depth (number of channels)
  - So far we characterized convolution with one kernel
  - In general, we will have multiple of these kernels per layer
    - Called number of channels or features → depth of the layer





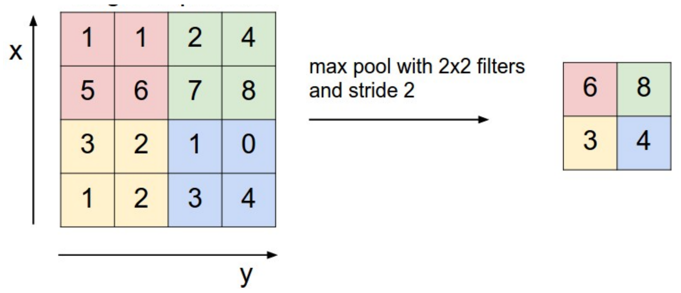




https://www.baeldung.com/cs/ml-understanding-dimensions-cnn

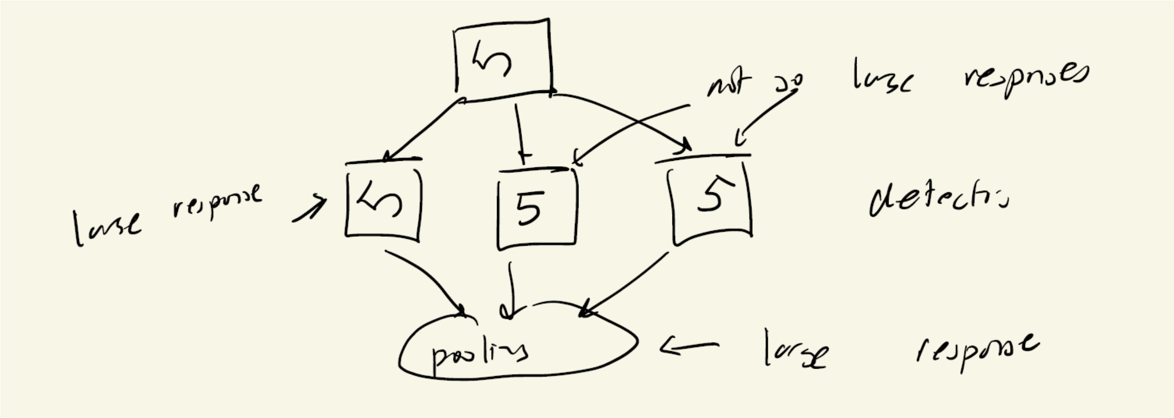
# Other CNN Components

- One common layer is a *pooling* layer
  - Replaces output of previous layer by “summary statistics” of nearby outputs
  - Several versions
    - Max pooling
    - Average pooling
    - $l_2$ norm
    - Weighted average
  - Typically used with downsampling
    - We’ve seen it before (e.g. wavelets)
    - Most common version is max pooling with 2×2 filters
      - Take max of 4 elements
      - For non-overlapping blocks (i.e. stride = 2)



# Pooling Layer

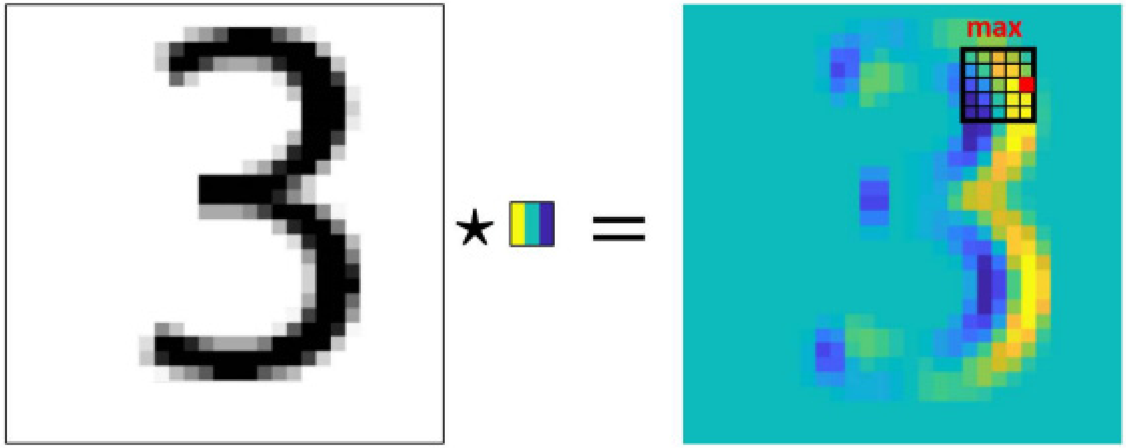
- Why is max pooling useful?
  - Produces a large response if any response coming into it is large




  - Good for detection

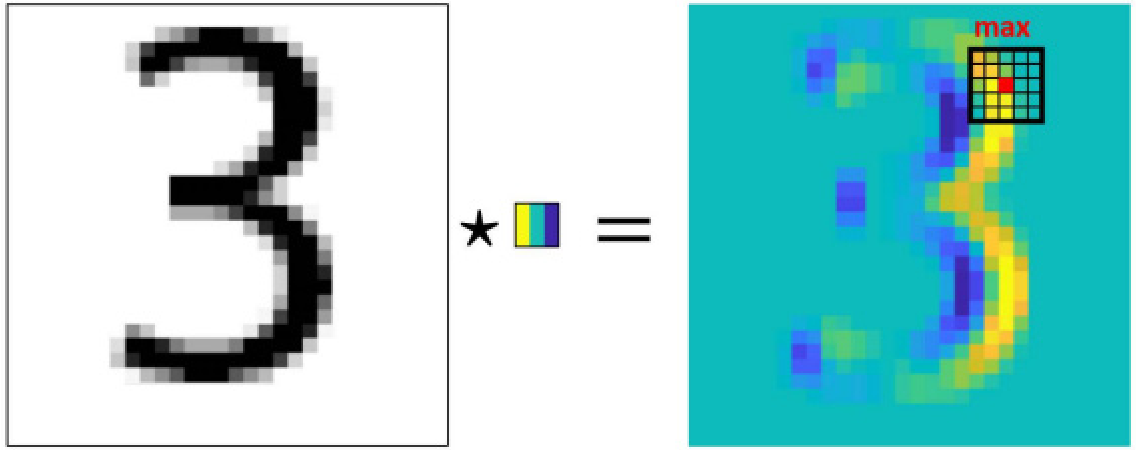
# Pooling Layer

- Why is max pooling useful?
  - Approximately shift invariant
  - Think of the max example from earlier
  - Convolution followed by max pooling:



# Pooling Layer

- Why is max pooling useful?
  - Approximately shift invariant
  - Think of the max example from earlier
  - Convolution followed by max pooling *for shifted input*:



# Pooling Layer

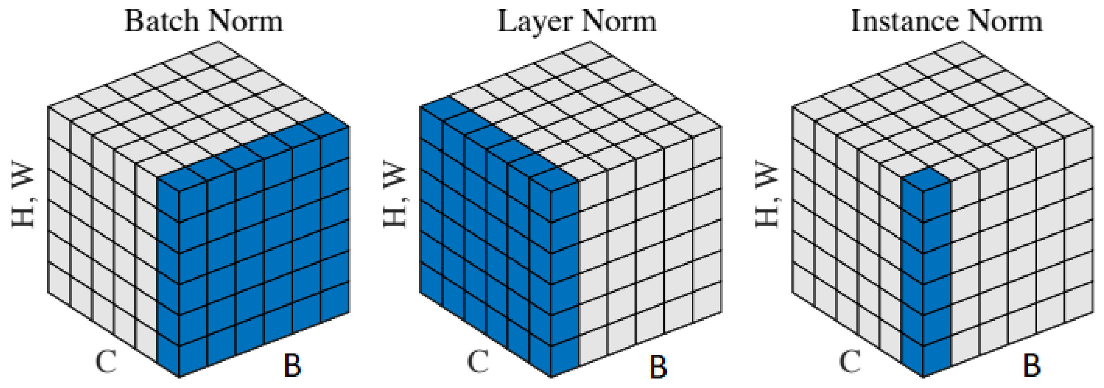
- Why is max pooling useful?
  - Approximately shift invariant
  - Think of the max example from earlier
  - Note it is not shift equivariant
- While they were commonly used in earlier CNNs, some recent methods do not necessarily employ pooling
  - If size reduction/downsampling is needed, this may be achieved by using larger stride in convolutions

# Other CNN Components

- Another common layer is a *normalization* layer
- Hard to train deep networks: All parameters updated simultaneously
  - If distribution of output changes of an earlier layer affects subsequent layers, often with amplification based on the distance
- Normalization: Statistics estimated from the input and used to re-parametrize the input
  - Idea: Subsequent layers are less sensitive to changes in previous layers
  - Faster & more stable training
- Several versions based on which dimension the normalization is performed over

# Normalization Layers

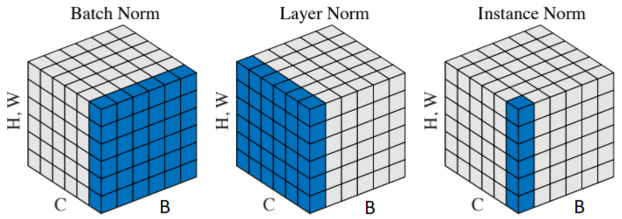
- General idea
  - We will view our data as a 3D tensor with dimensions
    - x-y dimension (vectorize for visualization – not in practice)
    - Batch dimension
    - Channel dimension
  - This allows us to view different normalization methods in the same framework




https://arxiv.org/pdf/1803.08494.pdf

# Normalization Layers

- General idea
  - We normalize in one dimension, D (i.e. HW, C or B) by
    - First normalize it to have zero mean and variance 1




$$\mathbf{z} = \frac{\mathbf{x} - \mathbb{E}_D[\mathbf{x}]}{\sqrt{\mathrm{Var}_D[\mathbf{x}] + \epsilon}}$$

sample mean along D

sample variance along D

for numerical stability

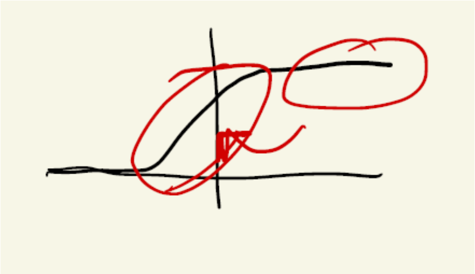
    - But zero mean and variance 1 may not be the best normalization for the network
    - So make these learnable

$$\hat{\mathbf{x}} = \gamma \mathbf{z} + \beta \qquad \gamma, \beta \text{ learnable}$$

    - If the original activation was the best, the learnable parameters can recover that too

# Normalization Layers

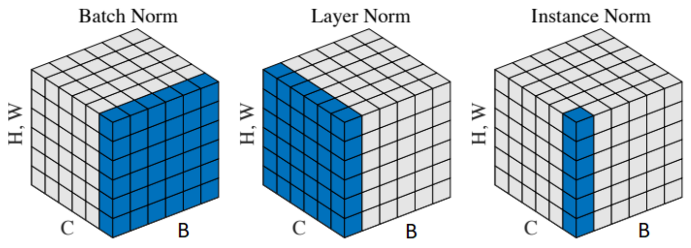
- For instance, for saturating activations, the normalization can force the inputs to be in the linear regime



→ easier backpropagation

- Normalization may allow for
  - Higher learning rates
  - Being less careful about initialization

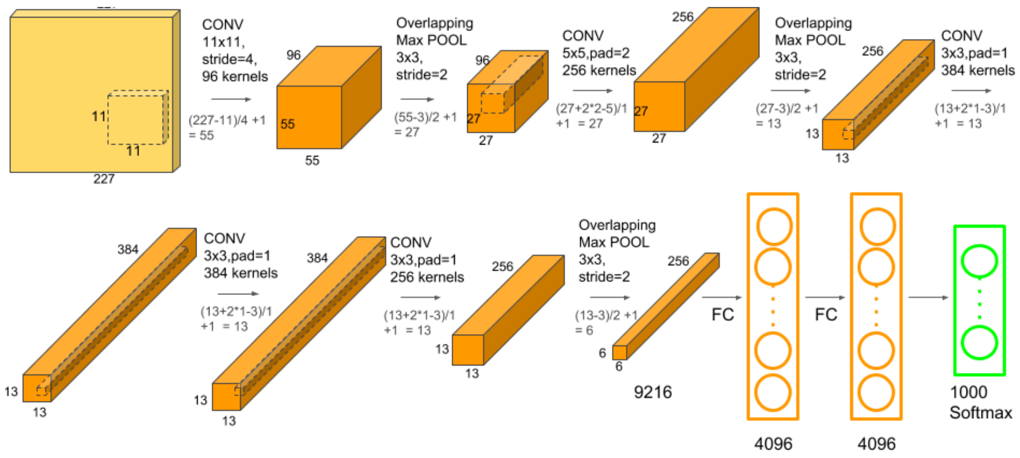
# Normalization Layers



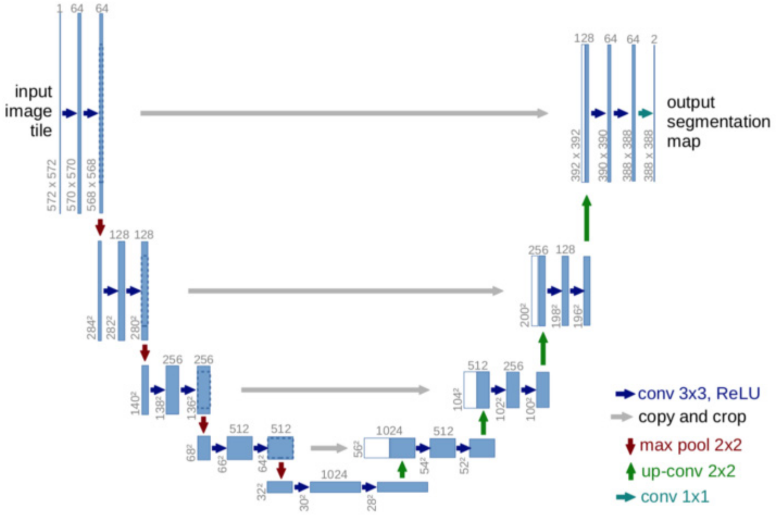

- Batch normalization: per mini-batch (first work)
  - Advantages mentioned earlier
  - Issues:
    - Unstable with small batch sizes (sample mean/variance not accurate)
    - Difference in training vs. testing (train with large batch size, test with one data → mismatch)
    - Does not work well on some architectures (application-dependent)
- Layer normalization: Normalizes input along feature channels of a layer
  - Improved performance for some architectures
- Instance normalization: Normalizes input across each feature channel
  - Applied at training & testing
  - Designed to make network agnostic to contrast in the images

# Visualizing CNNs

− In a few lectures we will look at figures like




AlexNet




UNet

# Recap

- Convolutional neural networks
  - Why convolutions instead of fully-connected layers?
    - Sparse weights
    - Parameter sharing
    - Translation equivariance
    - Works with inputs of different sizes
  - Variations on convolutions & visualizing convolutional layers
  - Other CNN components
    - Pooling layers
    - Normalization layers
    - Activation layers (from earlier)

# Course Announcements

- I am traveling M-W next week
  - No office hours
- Tuesday lecture [like last time]:
  - I will post a pre-recorded version from last year online
  - Merve will give an in-person lecture going over the same slides
    - If you have questions, this will be more interactive
- Good: No quiz for Tuesday