Fuzzy Set Theory in Computer Vision: **Example 3, Part II**

Derek T. Anderson and James M. Keller

FUZZ-IEEE, July 2017

Overview

- Resource; CS231n: Convolutional Neural Networks for Visual Recognition
  - https://github.com/tuanavu/Stanford-CS231/tree/master/lectures
- Resource; CNN tutorial
  - http://cs231n.github.io/convolutional-networks/

DL, you need lots of data...

- Well, maybe ...

DL, you need lots of data...

- Well, maybe ...
- Most folks do NOT train from scratch

DL, you need lots of data...

- Well, maybe ...
- Most folks do NOT train from scratch
  - For example, maybe you train on ImageNet, which has a lot of data, and then you fine tune your network relative to your specific data/problem

DL, you need lots of data...

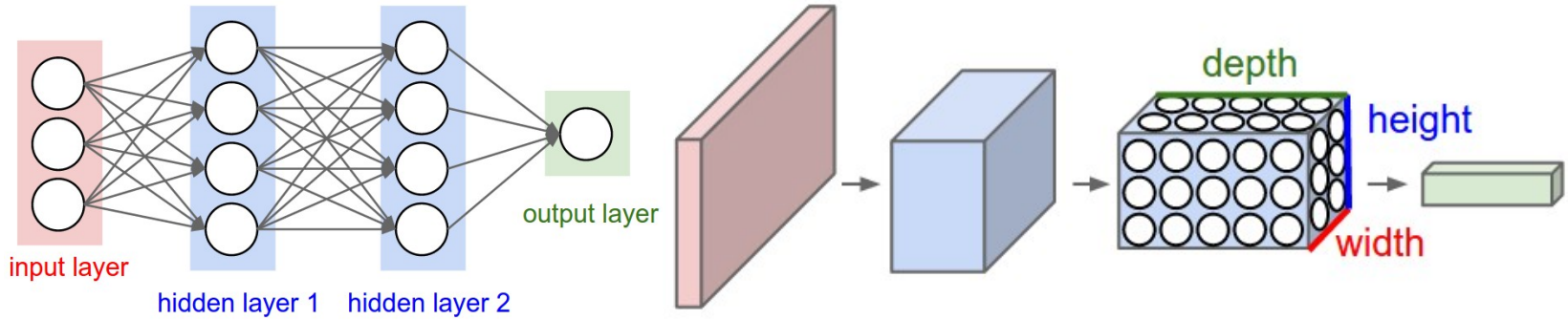
- Well, maybe ...
- Most folks do NOT train from scratch
  - For example, maybe you train on ImageNet, which has a lot of data, and then you fine tune your network relative to your specific data/problem
- Transfer learning

DL, you need lots of data...

- Well, maybe ...
- Most folks do NOT train from scratch
  - For example, maybe you train on ImageNet, which has a lot of data, and then you fine tune your network relative to your specific data/problem
- Transfer learning
  - Hot field, recommend you do your homework

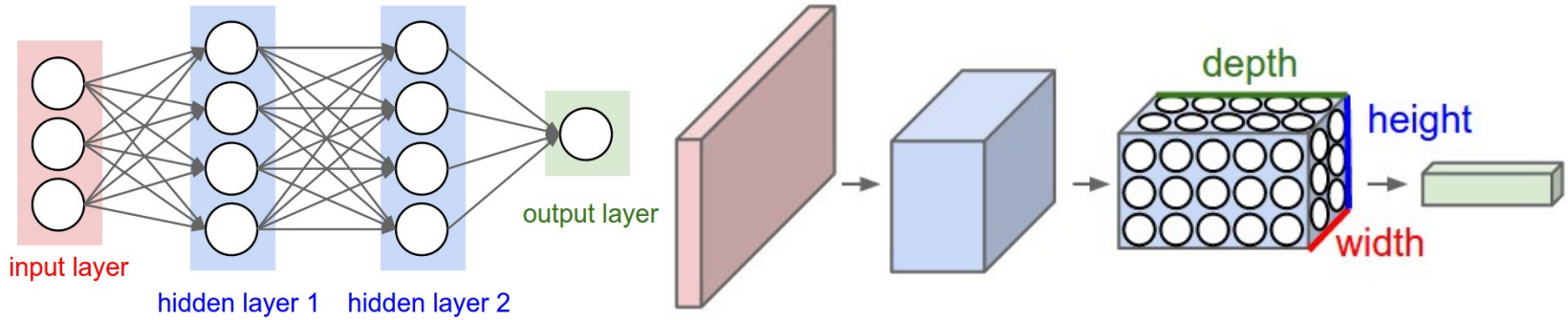
DL, you need lots of data...

- Well, maybe ...
- Most folks do NOT train from scratch
  - For example, maybe you train on ImageNet, which has a lot of data, and then you fine tune your network relative to your specific data/problem
- Transfer learning
  - Hot field, recommend you do your homework
  - For example, on small data sets maybe you just retrain the classifier? However, for medium size data sets maybe you retrain a bigger portion of the network or maybe all of it.



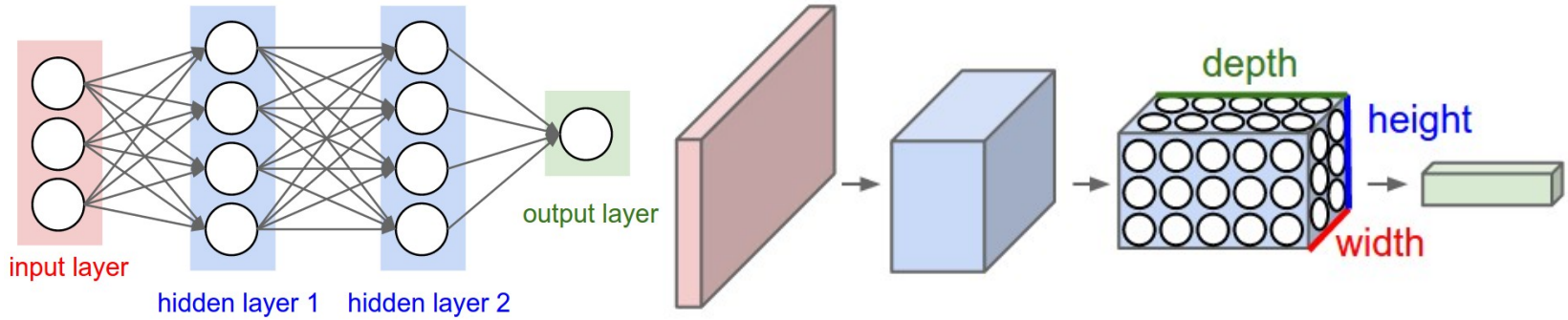
CNN basics

- Neurons typically arranged in width, height and "depth"



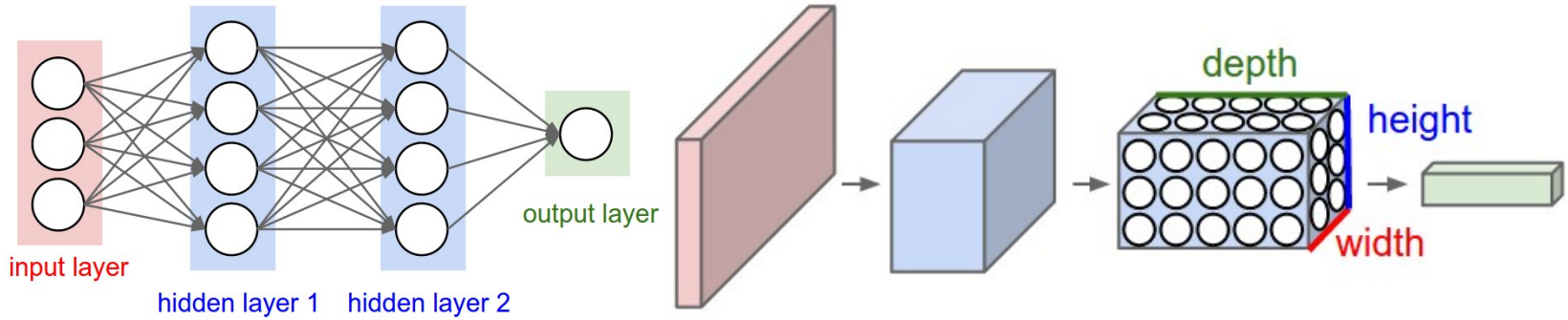
CNN basics

- Neurons typically arranged in width, height and "depth"
- For example
  - Input: RGB, maybe $256x256x3$



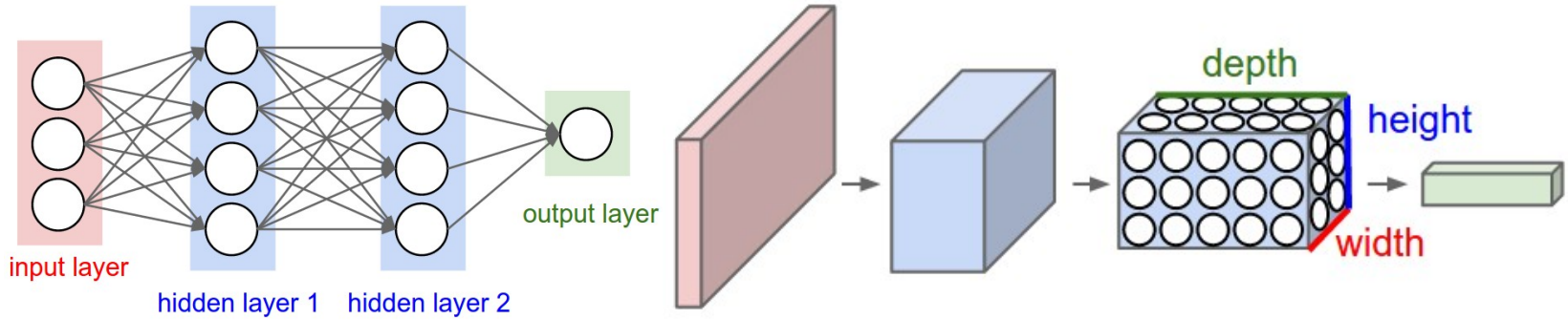
CNN basics

- Neurons typically arranged in width, height and "depth"
- For example
  - Input: RGB, maybe 256x256x3
  - First conv: maybe 32 filters of size 5x5x3



CNN basics

- Neurons typically arranged in width, height and "depth"
- For example
  - Input: RGB, maybe 256x256x3
  - First conv: maybe 32 filters of size 5x5x3
  - Output: filter maps, size 251x251x32



CNN basics

- Neurons typically arranged in width, height and "depth"
- For example
  - Input: RGB, maybe 256x256x3
  - First conv: maybe 32 filters of size 5x5x3
  - Output: filter maps, size 251x251x32
- Many different types of layers

Trivial example architecture

- Input, [32x32x3]

Trivial example architecture

- Input, [32x32x3]
- CONV layer, [32x32x12] (12 filters)

Trivial example architecture

- Input, [32x32x3]
- CONV layer, [32x32x12] (12 filters)
- RELU layer, maybe max(0,x) (stays [32x32x12])

Trivial example architecture

- Input, [32x32x3]
- CONV layer, [32x32x12] (12 filters)
- RELU layer, maybe max(0,x) (stays [32x32x12])
- Max POOL layer, downsample, maybe get [16x16x12]

Trivial example architecture

- Input, [32x32x3]
- CONV layer, [32x32x12] (12 filters)
- RELU layer, maybe max(0,x) (stays [32x32x12])
- Max POOL layer, downsample, maybe get [16x16x12]
- FC layer, MLP, output 5 neurons for 5 classes

Convolution

- Weight sharing (vs fully connected)

Convolution

- Weight sharing (vs fully connected)
- For example, if 32 filters and if each filter is say $7x5$ size, then each filter results in a "filter map" of size $(N-(7-1))x(M-(5-1))$ if the input was size NxM and if no padding is used (varies per implementation)

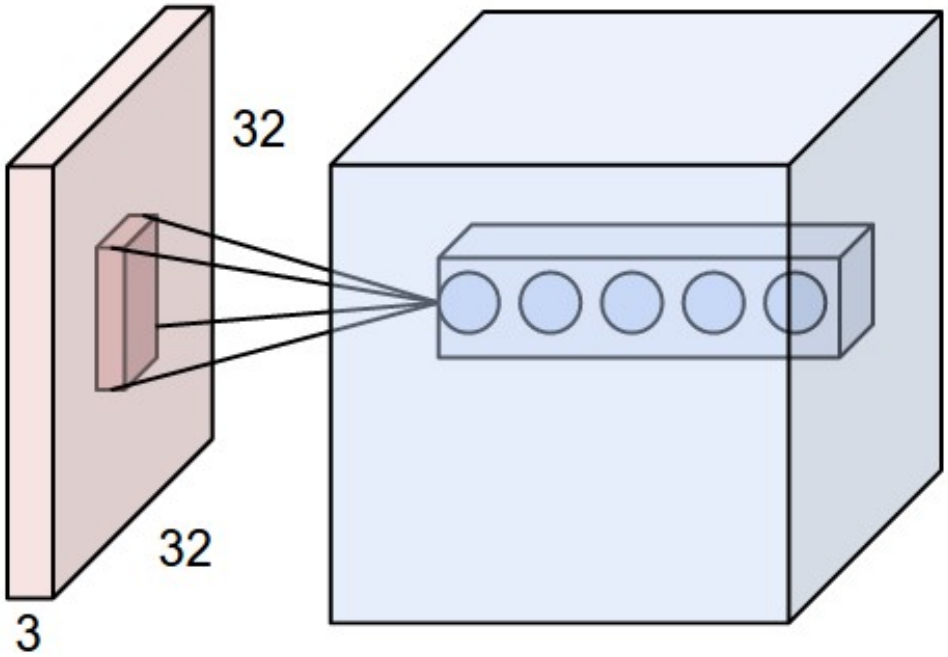
Convolution

- Weight sharing (vs fully connected)
- For example, if 32 filters and if each filter is say $7x5$ size, then each filter results in a "filter map" of size $(N-(7-1))x(M-(5-1))$ if the input was size NxM and if no padding is used (varies per implementation)
- Convolution "slides" the $7x5$ filter (pattern signal) over every pixel (unless a stride, jump, is used) and it produces a value at each location–telling us how similar the local spatial region is to that filter

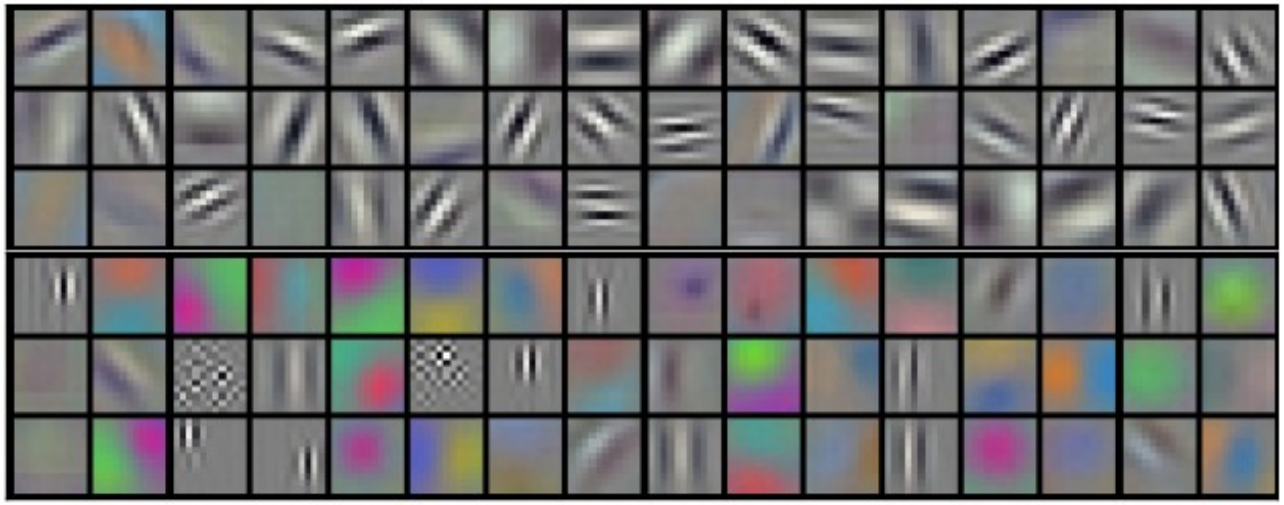
Convolution

- Weight sharing (vs fully connected)
- For example, if 32 filters and if each filter is say $7x5$ size, then each filter results in a "filter map" of size $(N-(7-1))x(M-(5-1))$ if the input was size NxM and if no padding is used (varies per implementation)
- Convolution "slides" the $7x5$ filter (pattern signal) over every pixel (unless a stride, jump, is used) and it produces a value at each location–telling us how similar the local spatial region is to that filter
- The deeper you go in the net, typically you move from "micro" to "macro" features, e.g., edges to maybe eyes to maybe faces, and many claim the deeper you go the more "semantic" information you start to mine as well ...

Example of convolution at one spatial location

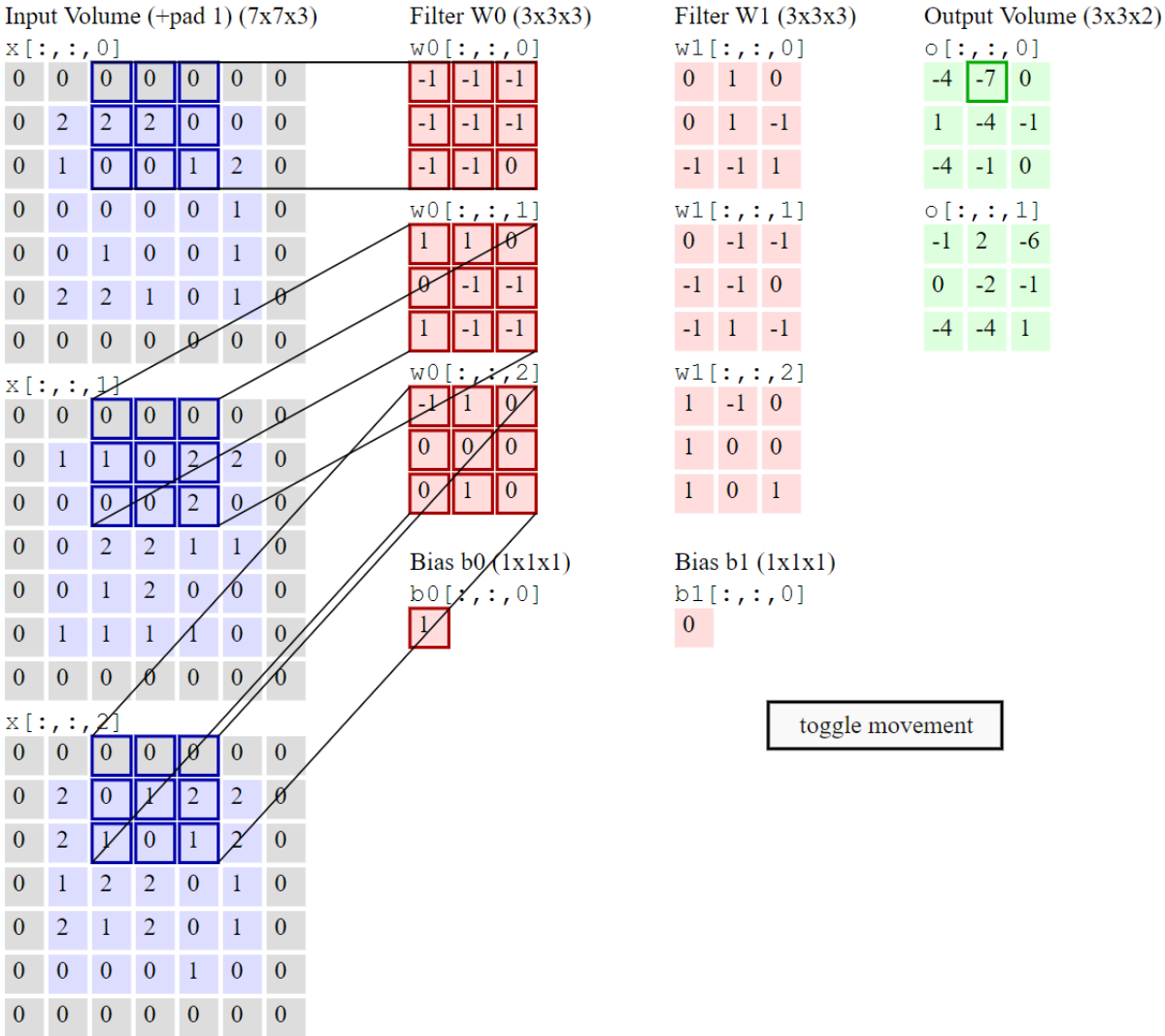


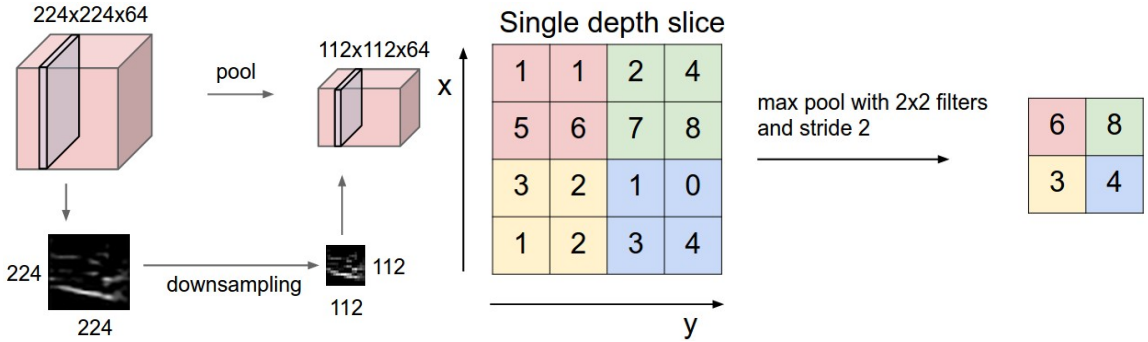
Example filters



Example filters learned by Krizhevsky et al (AlexNet). Each of the 96 filters shown here is of size $[11x11x3]$, and each one is shared by the $55 * 55$ neurons in one depth slice.

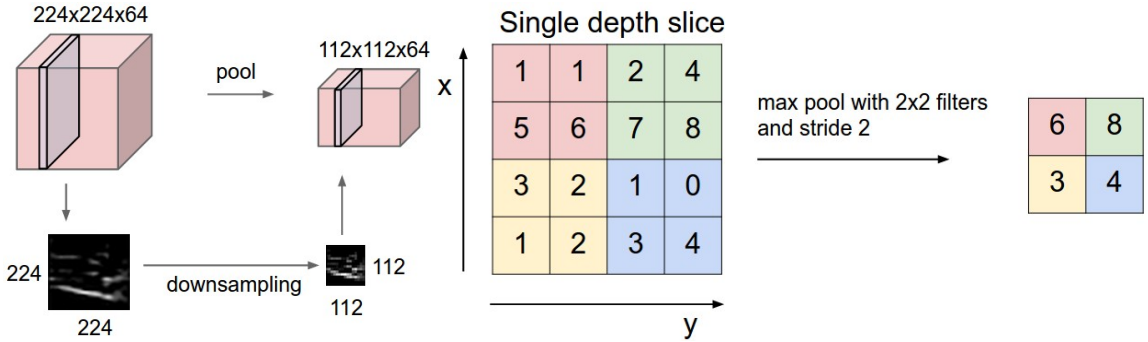
Full convolution





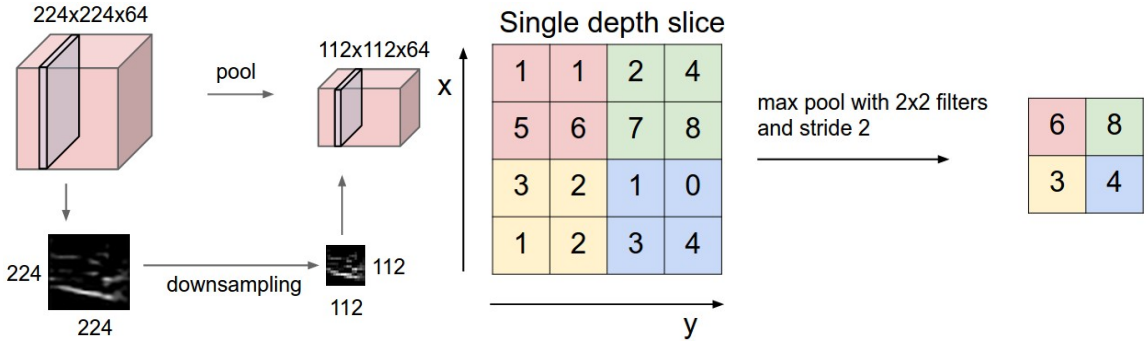
Pooling

- Progressively nonlinearlly reduces the spatial size of our signal



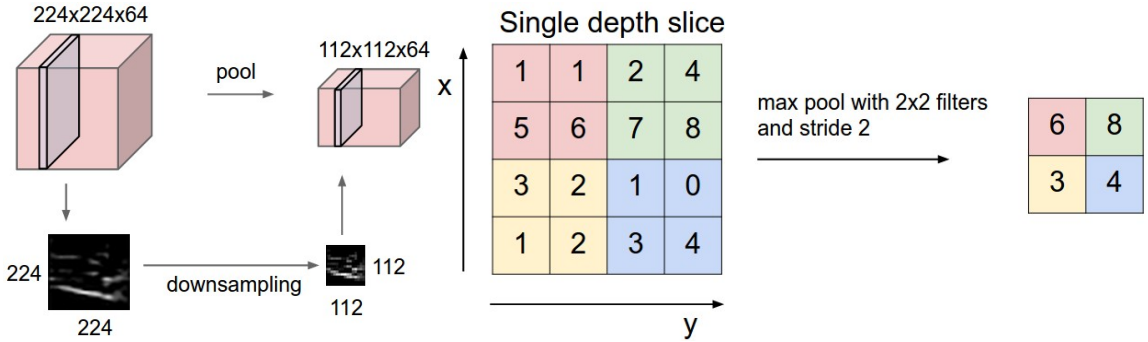
Pooling

- Progressively nonlinearlly reduces the spatial size of our signal
- Eliminate non-maximal values - reduce comp. in upper layers



Pooling

- Progressively nonlinearlly reduces the spatial size of our signal
- Eliminate non-maximal values - reduce comp. in upper layers
- Helps control overfitting as well to some extent



Pooling

- Progressively nonlinearlly reduces the spatial size of our signal
- Eliminate non-maximal values - reduce comp. in upper layers
- Helps control overfitting as well to some extent
- Translation and rotation and ...

Pooling

- Death in the details

Pooling

- Death in the details
- The backward pass for a max(x,y) operation has a simple interpretation as only routing the gradient to the input that had the highest value in the forward pass. Hence, during the forward pass of a pooling layer it is common to keep track of the index of the max activation (sometimes also called the switches) so that gradient routing is efficient during backpropagation.

Pooling

- Death in the details
- The backward pass for a max(x,y) operation has a simple interpretation as only routing the gradient to the input that had the highest value in the forward pass. Hence, during the forward pass of a pooling layer it is common to keep track of the index of the max activation (sometimes also called the switches) so that gradient routing is efficient during backpropagation.
- Many use this with deconvolution (a different topic!) to decide how to reverse engineer the filters for purposes like visualization (see Zeiler et al., reverses via the switches, that induces a sparse map and various ways to impute the missing values)

Rectified linear unit (ReLU)

- Non-linear gating

Rectified linear unit (ReLU)

- Non-linear gating
- $f(x) = max(0, x)$
  - Activation is simply thresholded at zero

Rectified linear unit (ReLU)

- Non-linear gating
- $f(x) = max(0, x)$
  - Activation is simply thresholded at zero
  - Does not saturate in positive region

Rectified linear unit (ReLU)

- Non-linear gating
- $f(x) = max(0, x)$
  - Activation is simply thresholded at zero
  - Does not saturate in positive region
- Was found to accelerate (factor of 6 inKrizhevsky et al.) convergence of stochastic gradient descent compared to sigmoid/tanh functions

Rectified linear unit (ReLU)

- Non-linear gating
- $f(x) = max(0, x)$
  - Activation is simply thresholded at zero
  - Does not saturate in positive region
- Was found to accelerate (factor of 6 inKrizhevsky et al.) convergence of stochastic gradient descent compared to sigmoid/tanh functions
- Simple to implement (versus alternative solutions, e.g., tanh/sigmoid)

Rectified linear unit (ReLU)

- Non-linear gating
- $f(x) = max(0, x)$
  - Activation is simply thresholded at zero
  - Does not saturate in positive region
- Was found to accelerate (factor of 6 inKrizhevsky et al.) convergence of stochastic gradient descent compared to sigmoid/tanh functions
- Simple to implement (versus alternative solutions, e.g., tanh/sigmoid)
- Leaky ReLU ("dying ReLU problem")
  - $f(x) = 1(x < 0)(ax) + 1(x >= 0)(x)$ [a is a small constant]

Rectified linear unit (ReLU)

- Non-linear gating
- $f(x) = max(0, x)$
  - Activation is simply thresholded at zero
  - Does not saturate in positive region
- Was found to accelerate (factor of 6 inKrizhevsky et al.) convergence of stochastic gradient descent compared to sigmoid/tanh functions
- Simple to implement (versus alternative solutions, e.g., tanh/sigmoid)
- Leaky ReLU ("dying ReLU problem")
  - $f(x) = 1(x < 0)(ax) + 1(x >= 0)(x)$ [a is a small constant]
- Other ones in the lit

Your all good, right?

- Nope ... but a start

Your all good, right?

- Nope ... but a start
- Stochastic gradient descent

Your all good, right?

- Nope ... but a start
- Stochastic gradient descent
- Batch normalization

Your all good, right?

- Nope ... but a start
- Stochastic gradient descent
- Batch normalization
- Architectures

Your all good, right?

- Nope ... but a start
- Stochastic gradient descent
- Batch normalization
- Architectures
- Deconvolution

Your all good, right?

- Nope ... but a start
- Stochastic gradient descent
- Batch normalization
- Architectures
- Deconvolution
- GPU acceleration

Your all good, right?

- Nope ... but a start
- Stochastic gradient descent
- Batch normalization
- Architectures
- Deconvolution
- GPU acceleration
- Regularization and drop out
- ...