DEEP LEARNING

Convolutional Neural Networks

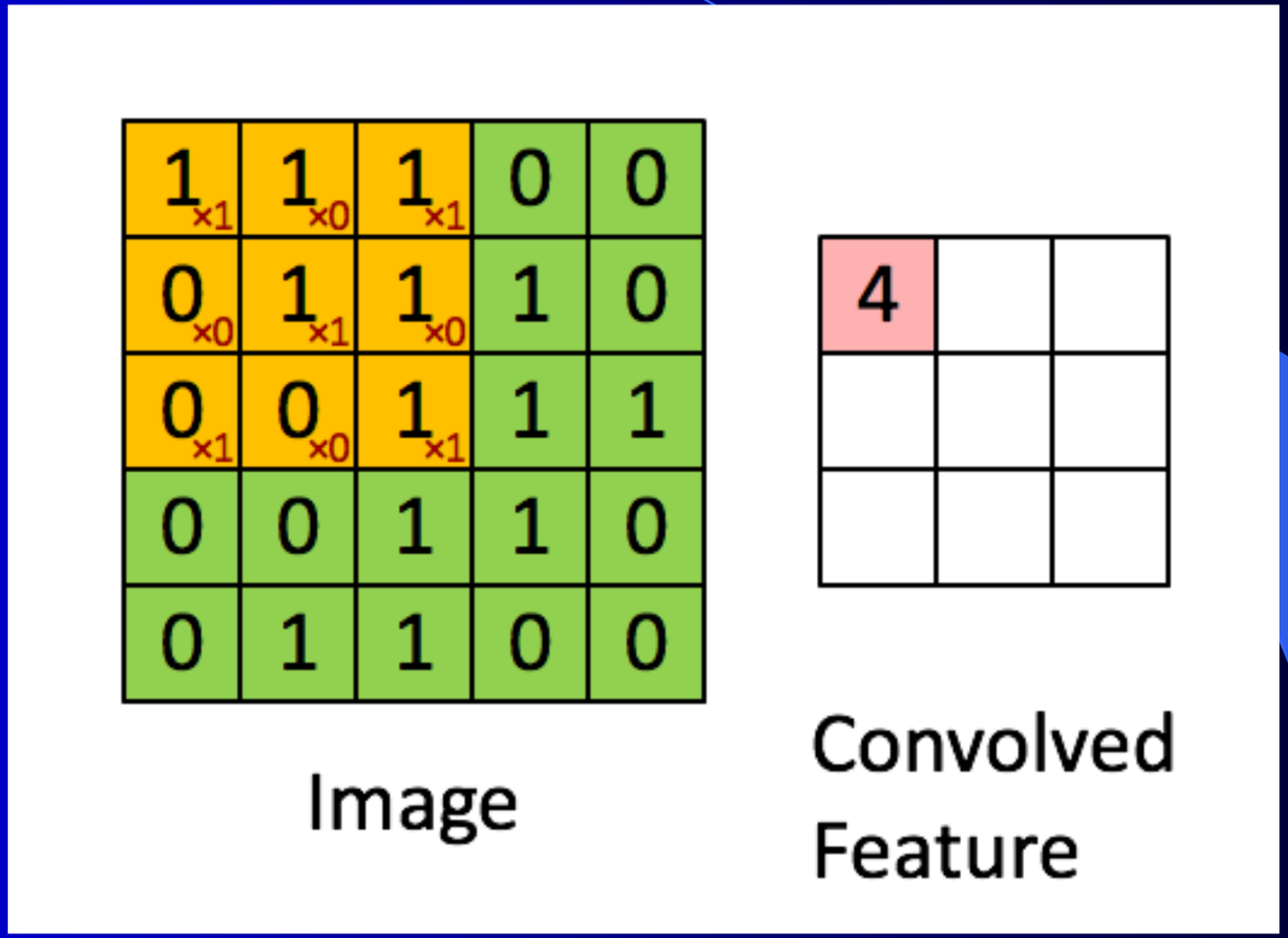
Convolutional Neural Networks

- "Niche" networks built specifically for problems with low dimensional (e.g. 2-*d*) grid-like local structure – e.g. Images
  - Vision, character recognition, speech, games, images - where neighboring features have high correlations (pixels, words, etc.), while distant features (pixels, words) are less correlated
    - Typically just uses raw features (e.g. pixels) with no preprocessing
  - Natural images have the property of being stationary, meaning that the statistics of one part of the image are the same as any other part
  - While standard NN nodes take input from all nodes in the previous layer, CNNs enforce that a node receives only a small set of features which are spatially or temporally close to each other called *kernels* from one layer to the next (e.g. 3x3, 5x5), thus enabling ability to handle local 2-D structure.
    - Can find edges, corners, endpoints, etc.
    - Good for problems with local 2-D structure, but lousy for general learning with abstract features having no prescribed feature ordering or locality

Convolutions

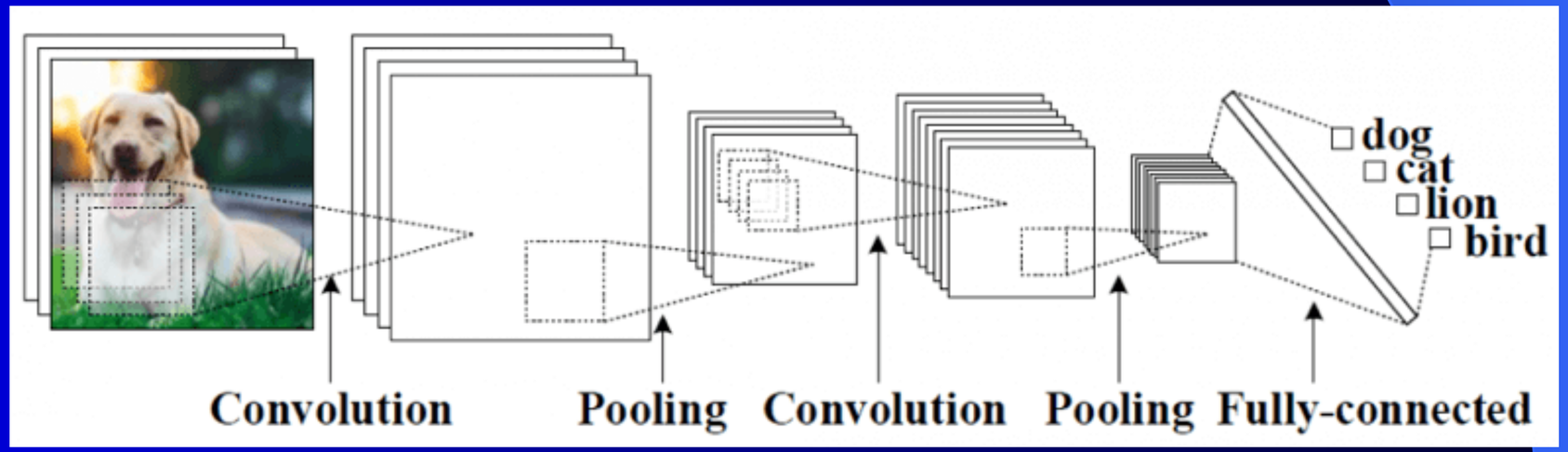
- Typical MLPs have a connection from every node in the previous layer, and the net value for a node is the scalar dot product of the inputs and weights (e.g. matrix multiply). Convolutional nets are somewhat different:
  - Nodes still do a scalar dot product from the previous layer, but with only a small portion (kernel) of the nodes in the previous layer – *Sparse representation*
  - Every node in a feature map has the exact same weight values from the preceding layer – *Shared parameters*, tied weights, a LOT less unique weight values. Regularization by having same weights looking at lots of input positions (Convolutional filter – same weights)
  - Each node has its shared weight convolution computed on a kernel slightly shifted, from that of its neighbor, in the previous layer – *Translation invariance*.
  - Each node's convolution scalar (*net* value) is then passed through a non-linear activation function (ReLU, tanh, sigmoid, etc.)

Convolution Example



CNN

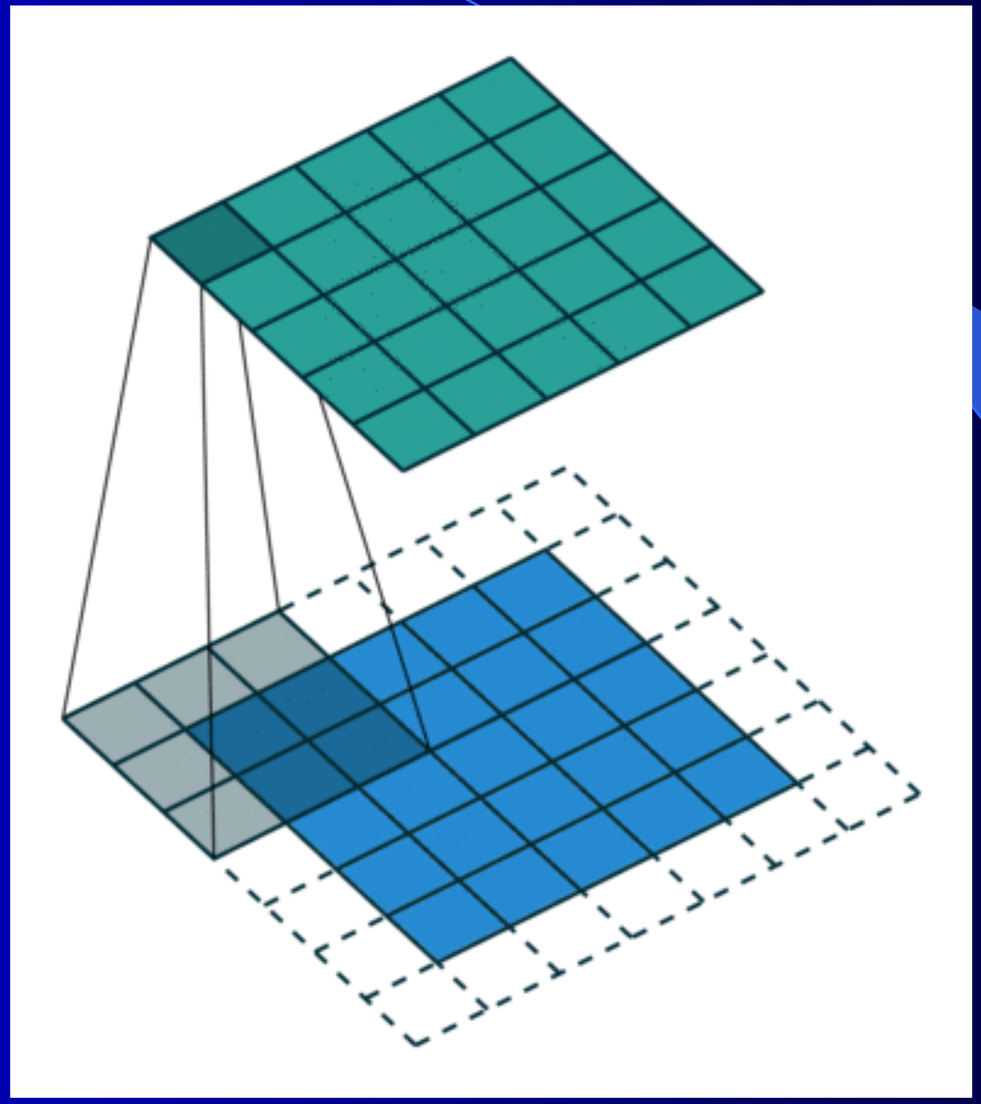
- The 2-*d* planes of nodes (or their outputs) at subsequent layers in a CNN are called *feature maps*
- Thus each *feature map* searches the full previous layer to see if, where, and how often its feature occurs (precise position less critical)
  - The output will be high at each node in the map corresponding to a kernel where the feature occurs (e.g. edge, curve)
  - Convolution layers search across all feature maps of the previous layer
  - Later layers can concern themselves with higher order combinations of features and rough relative positions – e.g. eyes next to each other with nose below



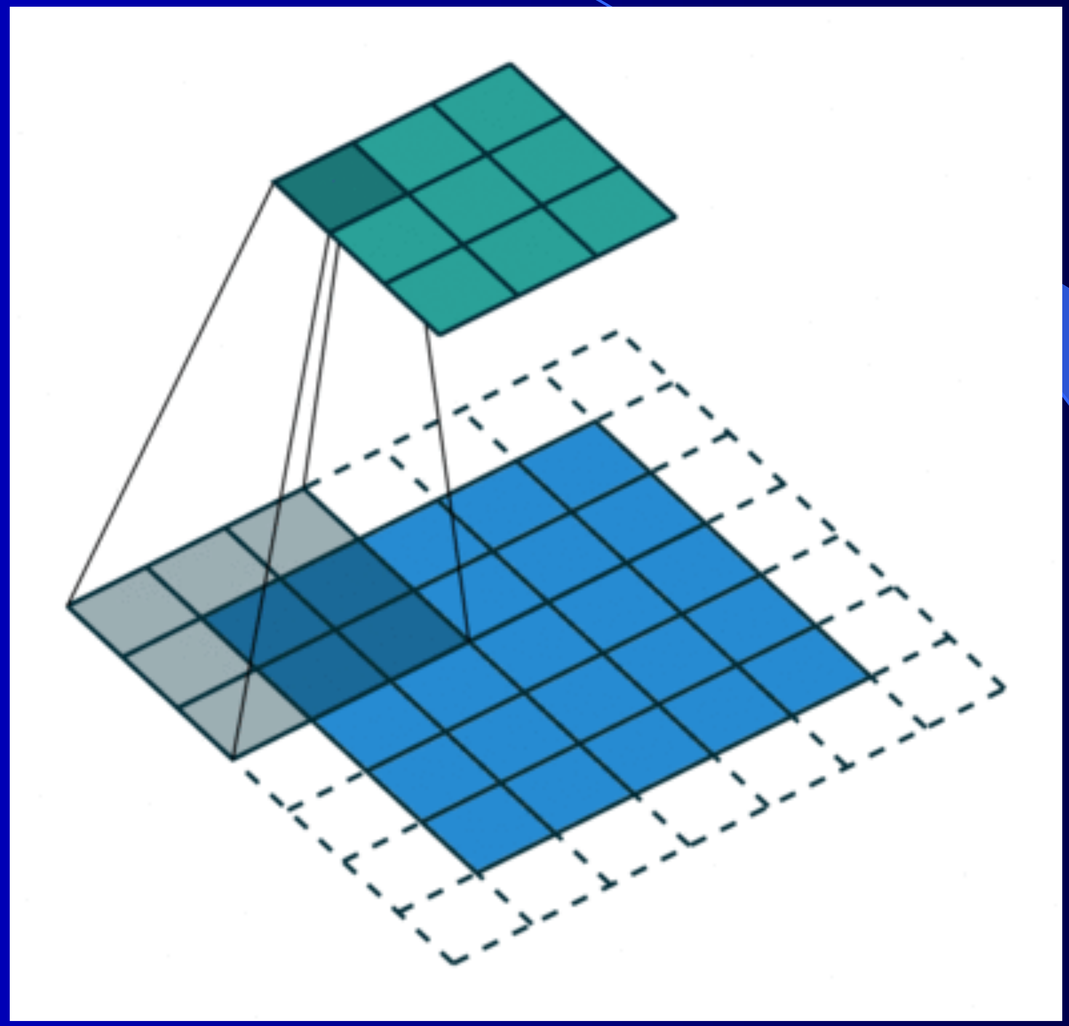
CNN Hyperparameters

- Structure itself, number of layers, size of filters, number of feature maps in convolution layers, connectivity between layers, activation functions, final supervised layers, etc.
- Drop-out often used in final fully connected layers for overfit avoidance – less critical in convolution/pooling layers which already regularize due to weight sharing
- As is, the feature map would always decrease in volume which is not usually desirable - *Zero-padding* avoids this and lets us maintain up to the same volume
  - Would shrink fast for large kernel/filter sizes and would limit the depth (number of layers) in the network, smaller kernels common (3x3)
  - Also allows the different filter sizes to fit arbitrary map widths
- *Stride* – Don't have to test every location for the feature (i.e. stride = 1), could sample more coarsely
  - Another option (besides pooling) for down-sampling

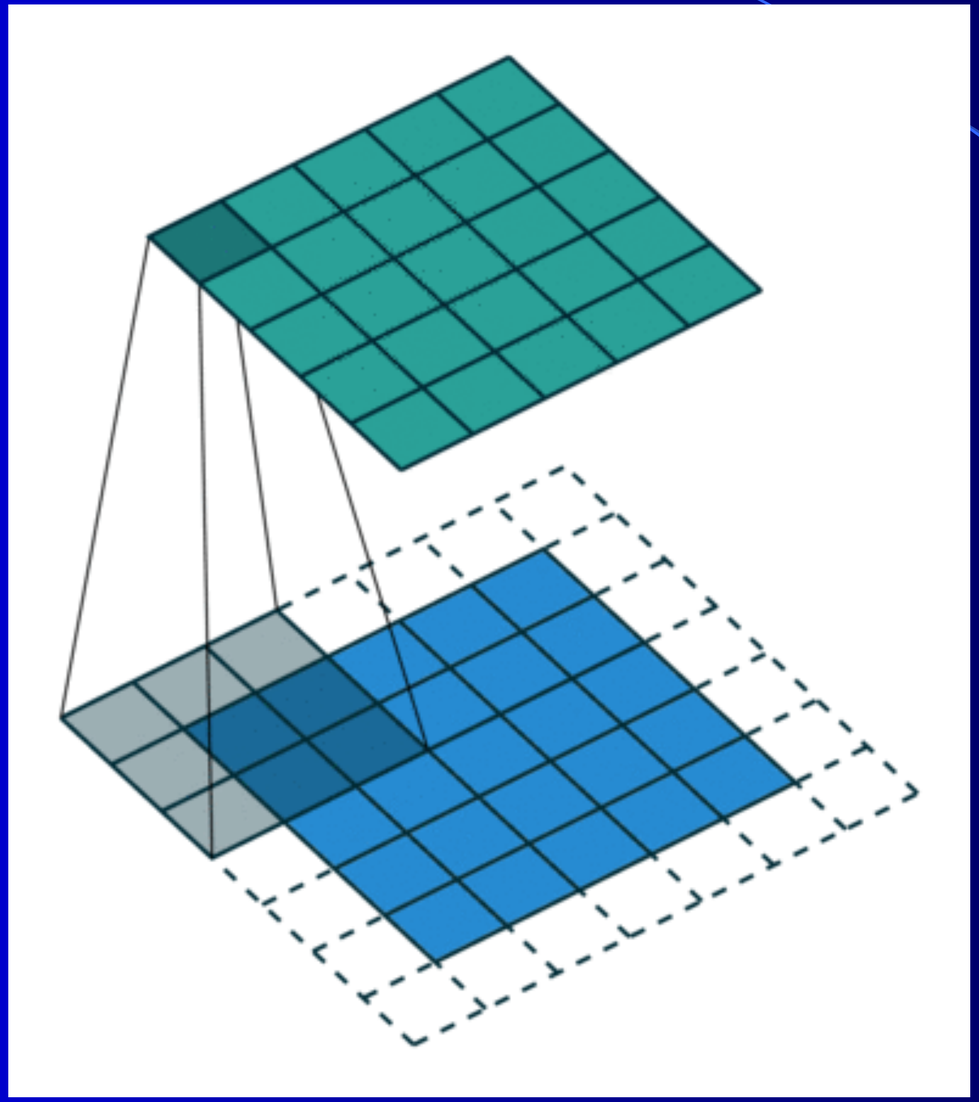
Convolutional Example
0 padding = 1 and stride = 1



Zero-Pad = 1, Stride = 2



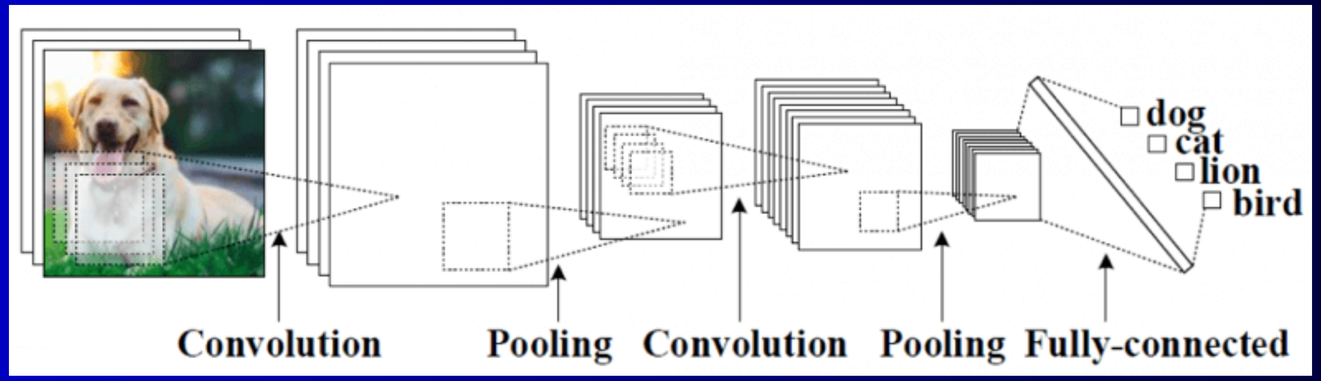
Convolutional Example
Zero pad = 1 and stride = 1



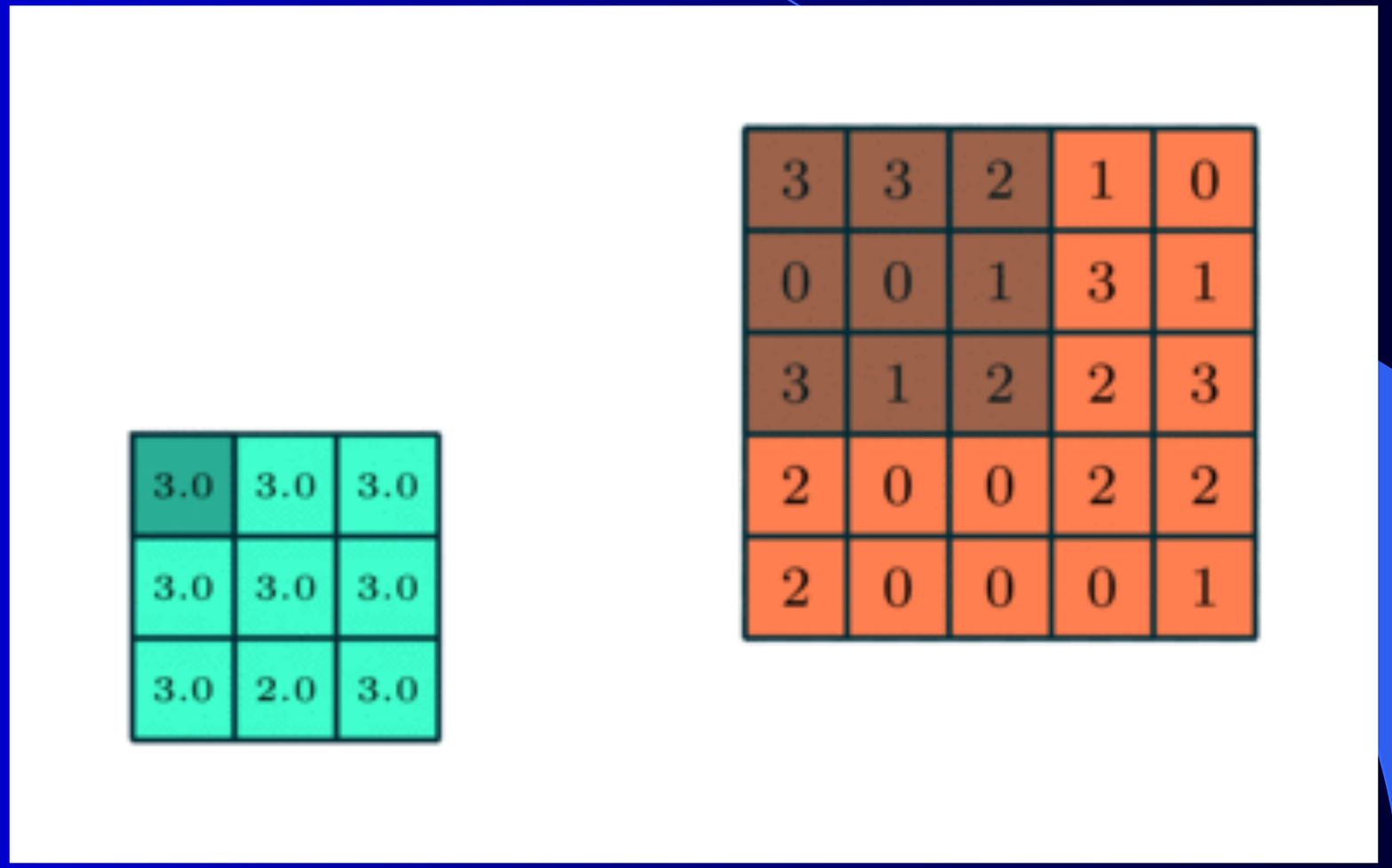
Note that the next layer (top) is still 5x5 due to the zero padding. What size would the next layer have been without zero padding?

Sub-Sampling (Pooling)

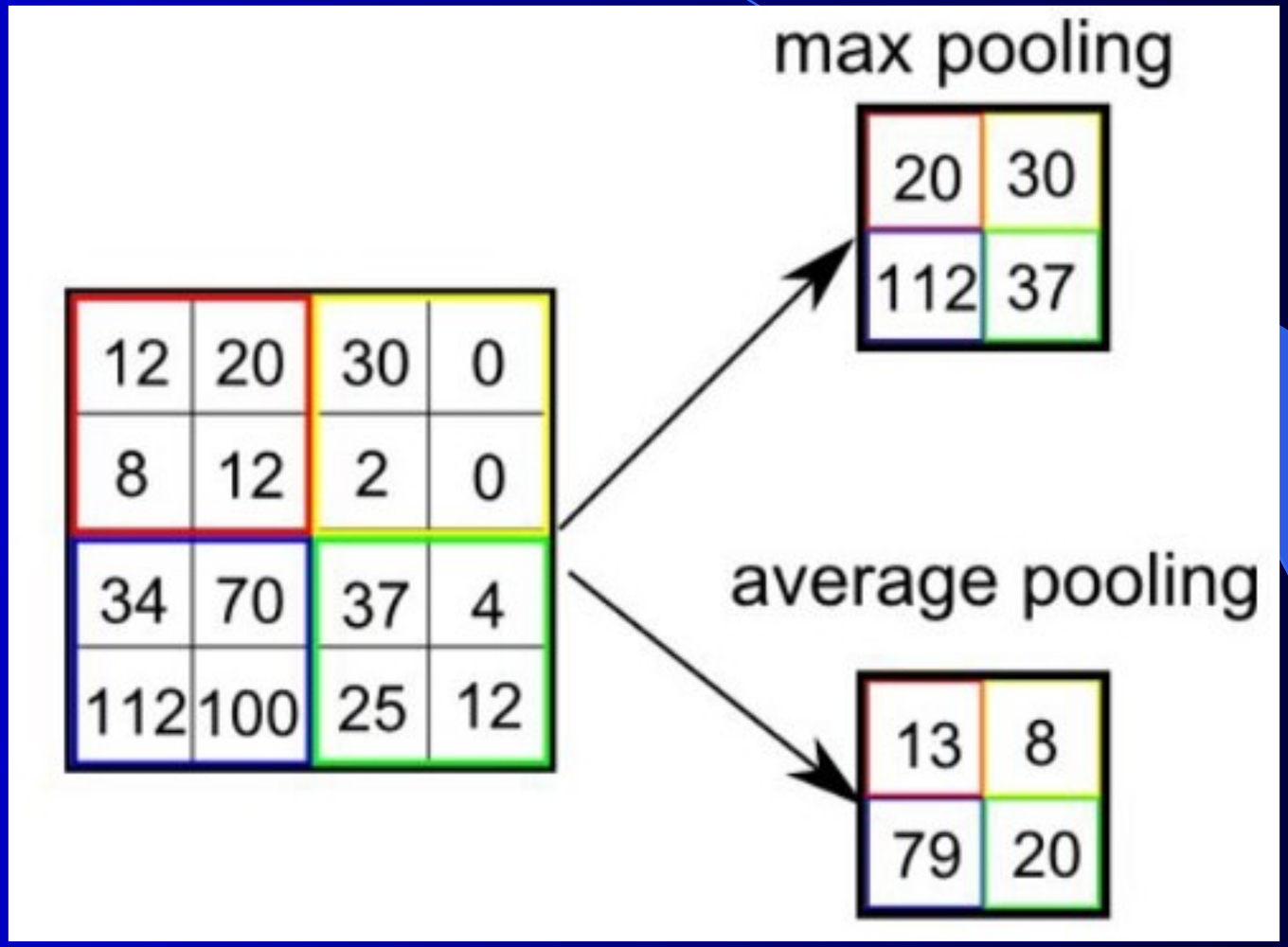
- Convolution and sub-sampling layers can be interleaved
- Sub/Down-sampling (Pooling) allows the number of features to be diminished, and to pool information
  - Pooling replaces the network output at a certain point with a summary statistic of nearby outputs
  - Max-Pooling common (Just as long as the feature is there, take the max, as exact position is not that critical), also averaging, etc.
  - Pooling smooths the data and reduces spatial resolution and thus naturally decreases importance of exactly where a feature was found, just keeping the rough location – translation invariance
  - 2x2 pooling would do 4:1 compression, 3x3 9:1, etc.
  - Convolution may increase number of feature maps per layer, pooling keeps same number of *reduced* maps (one-to-one correspondence of convolution map to pooled map) as the previous layer
  - Less pooling layers common in recent architectures to allow more depth



Max Pooling Example
(Sum or Average sometimes used)

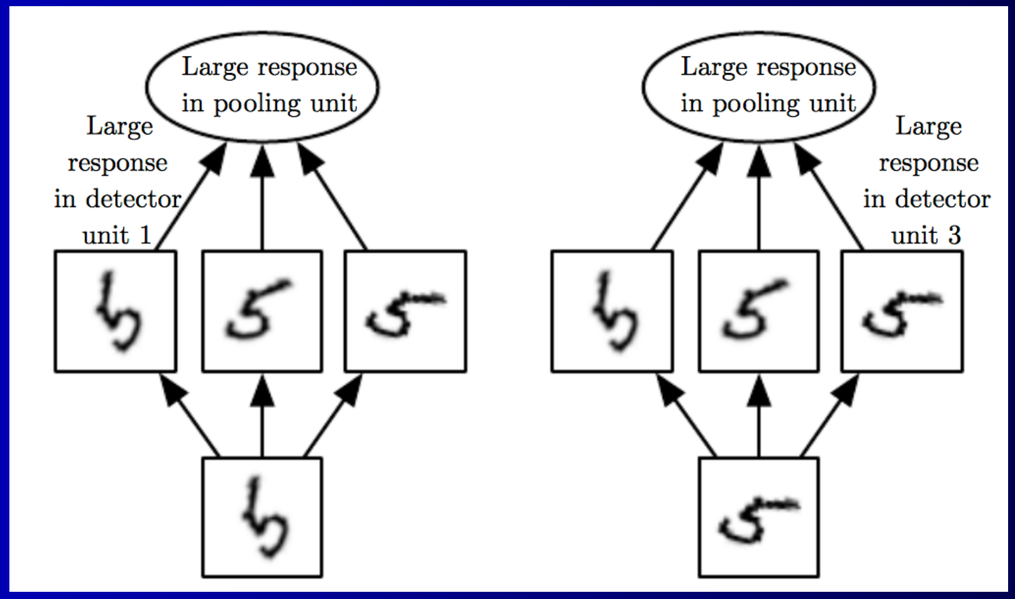


Max and Average Pooling
Non overlapping: Stride = 2



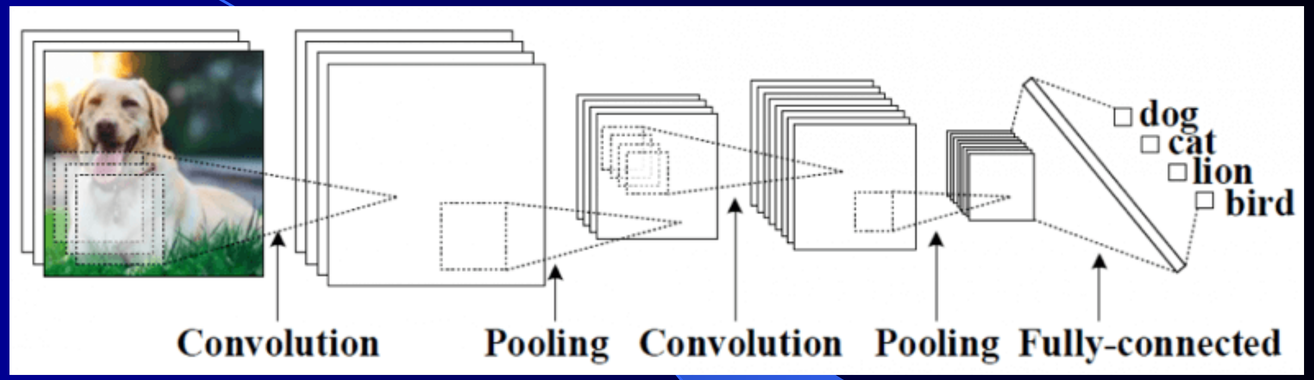
Pooling (cont.)

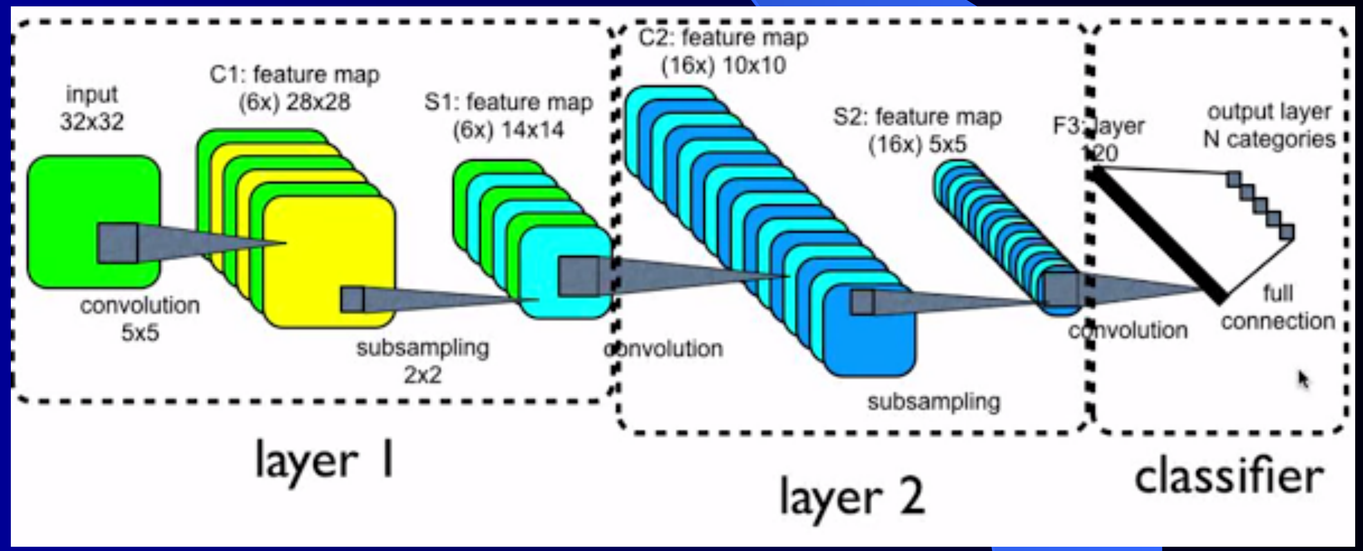
- Common layers are convolution, non-linearity, then pool (repeat)
- Note that pooling/down-sampling decreases map sizes (unless pool stride = 1, highly overlapped), making real deep nets more difficult. Pooling is sometimes used only after multiple convolved layers and sometimes not at all.
- At later layers pooling can make network invariant to more than just translation – *learned invariances*



Convolutional Neural Networks

- Big Picture: Each feature map learns a different feature. Each node in feature map has same translated kernel (and weights)
- Brute force search to see if and where certain features exist
- Pooling/sub-sampling – Does the feature exist in a general area
- Final standard supervised layer with improved feature space





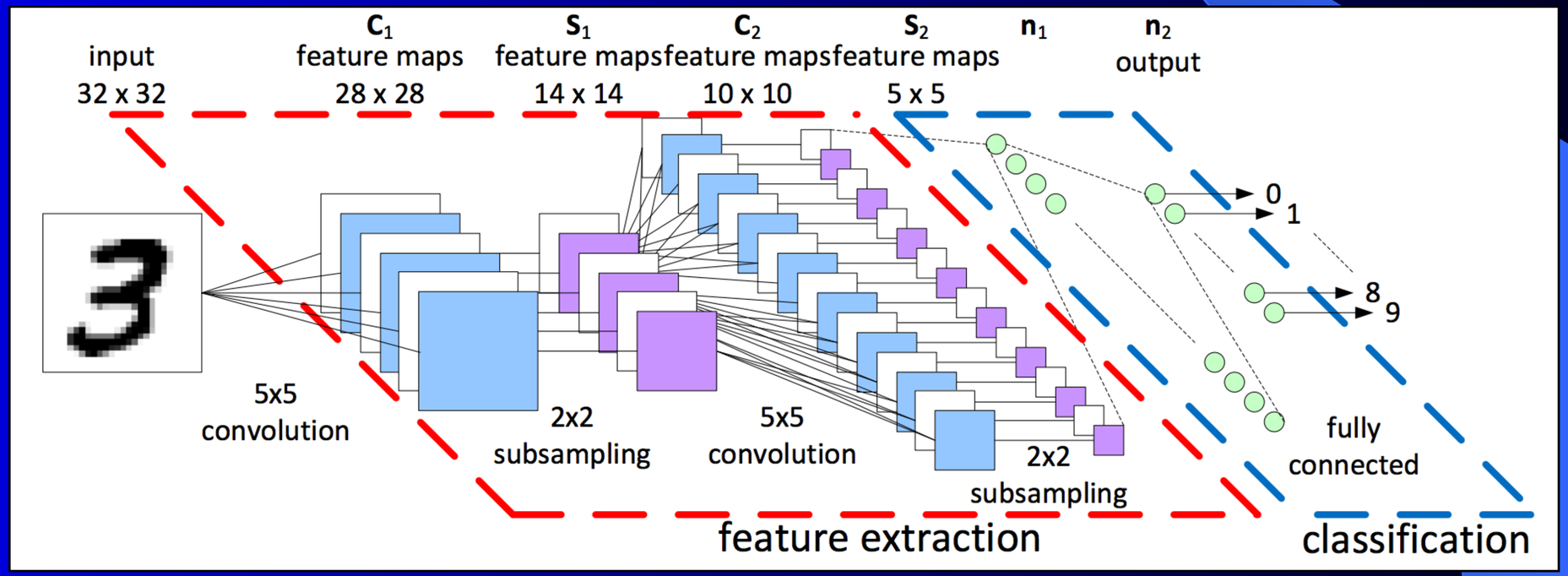
CNN Training

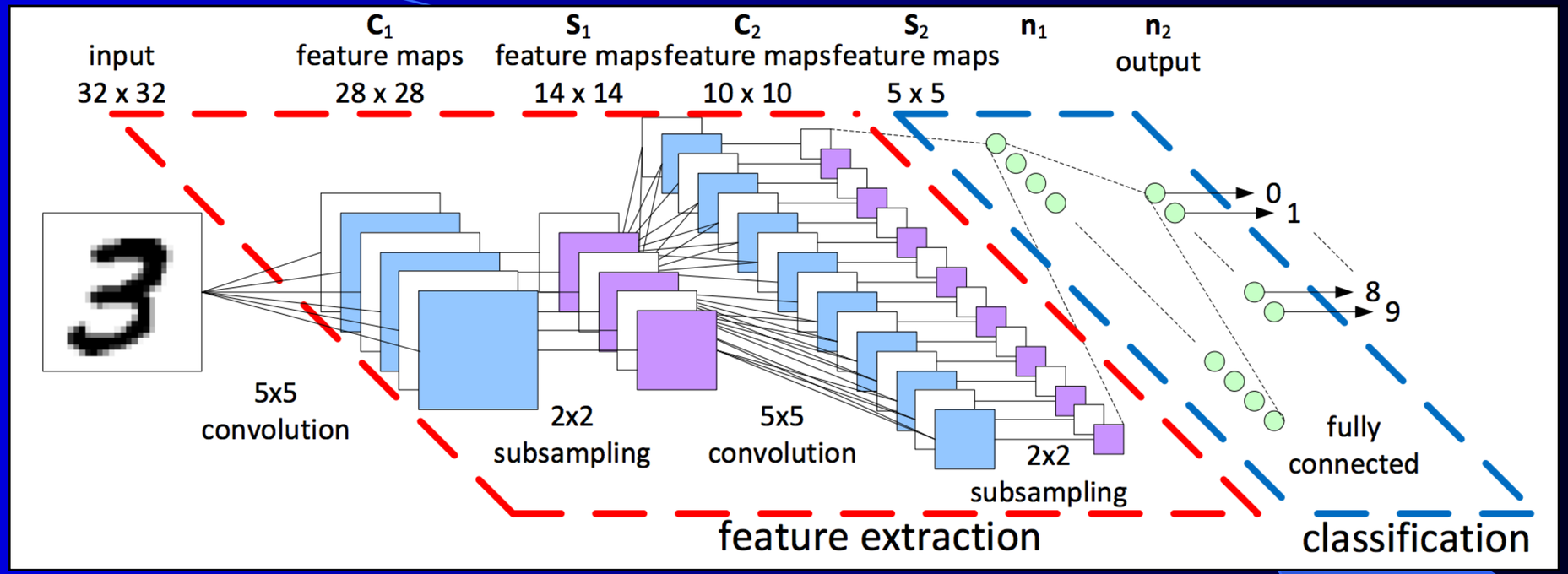
- Trained with BP, with weight tying in each feature map
  - Randomized initial weights throughout entire network, standard training on final fully connected network
- Feature Maps
  - Each feature map has one weight for each input and one bias
  - Thus a feature map with a 5x5 kernel (filter) would have a total of 26 weights, which are the same coming into each node of the feature map
  - If a convolution layer had 10 feature maps with 5x5 kernels, then a total of 260 unique weights would be trained in that layer (much less than an arbitrary deep net layer without sharing)
  - Calculate weight updates independently into each node but don't update yet. Average the weight updates over the tied weights and update each the same.
- Sub-Sampling (Pooling) Layer
  - All elements of kernel max'd, averaged, summed, etc. No trainable weights necessary
- While all weights are trained, the structure of the CNN is currently usually hand crafted with trial and error, including number of total layers, number of kernels, size of kernels, size of sub-sampling (pooling) fields, etc.

Example – CNN MNIST Classification

- Roughly based on LeCun's original model. To help it all sink in:

- How many connections and trainable weights at each layer?
  - MNIST data input is 32x32 pixels
  - Stride = 1 for convolutions, and pooling is non-overlapping





Layer	Trainable Weights (per layer)	Total Connections
C1	(25+1)*6 = 156	156*28*28 = 122,304
S1	0 (LeCun had (1+1)*6 = 12)	4 (2x2 links) *6*14*14 = 4704
C2	(5*5*6+1)*16 = 2416	2416*10*10 = 241,600
S2	0	4 (2x2 links) *16*5*5 = 1600
N1	(5*5*16+1)*120 = 48,120	Same since fully connected MLP at this point
Output	(120+1)*10 = 1210	Same

Example Cont.

Layer	Trainable Weights per layer	Total Connections
C1	(25+1)*6 = 156	156*28*28 = 122,304
S1	0 (LeCun had (1+1)*6 = 12)	4 (2x2 links) *6*14*14 = 4704
C2	(5*5*6+1)*16 = 2416	2416*10*10 = 241,600
S2	0	4 (2x2 links) *16*5*5 = 1600
N1	(5*5*16+1)*120 = 48,120	Same since fully connected MLP at this point
Output	(120+1)*10 = 1210	Same

- Why 32x32 to start with? Actual characters never bigger than 28x28. Just padding the edges so for example the top corner of the feature map can have a pad of two up and left for its feature map (since kernel is 5x5). Same things happens with 14x14 to 10x10 drop from S1 to C2.
- S1 and S2 non-overlapping and pool (max most common) – We include here the 4 unweighted connections
  - LeCun had a trainable weight and a bias in pool layer followed by a non-linearity, not really necessary and not used these days
- C2: Connects to all preceding maps, but no zero-padding so maps decrease in size
  - LeCun had each map connect to a subset of the preceding maps
- S2: Final number of extracted features to go to the MLP: (5*5)*16 = 400
- 419,538 total connections, with 51,902 trainable parameters 95% of which are in the final MLP. Only 2572 trainable weights in CNN.

CNN Homework

- Assume a traditional CNN with an initial input image of 16x16, followed by a convolutional layer with 8 feature maps using 5x5 kernels, followed by a max pooling layer with 2x2 kernels, followed by a convolution layer with 10 feature maps using 3x3 kernels. Those outputs go straight into (no additional pooling layer) a fully connected MLP with 20 hidden nodes followed by 3 output nodes for 3 possible output classes. Assume no padding and stride=1 for convolution layers, no overlap and no trainable weights for the one pooling layer, and convolutional maps connect to all maps in the previous layer. Sketch the network. For each layer state a) What is the size of the maps in the layer (e.g. the input layer is 16x16), b) how many unique trainable weights are there per layer, and c) total connections in the layer. Show your work and explain your numbers in each case (similar to the previous slide).

CNN Summary

- High accuracy for image applications – Breaking all records and doing it using just using raw pixel features!
- Special purpose net –Works for images or problems with strong grid-like local spatial/temporal correlation (speech, games, etc.)
- Once trained on one problem (e.g. vision) could use same net (often fine-tuned) for a new similar problem – general creator of vision features, pre-trained nets
- Unlike traditional nets, handles variable sized inputs
  - Same filters and weights, just convolve across different sized image and dynamically scale size of sub-sampling (pooling or increase stride with convolutions), or # of nodes, to normalize
  - Different sized images, different length speech segments, etc.
- Lots of hand crafting and CV tuning to find the right recipe of kernels, layer interconnections, etc.
  - Lots more hyperparameters than standard nets, since the structures of CNNs are more handcrafted
  - CNNs getting wider and deeper with speed-up techniques (e.g. GPU, ReLU, etc.) and lots of current research, excitement, and success