Convolutional neural networks

- Motivation
- Biological inspiration
- Convolution operation
- Convolutional layer
- Padding and stride
- CNN architecture

Motivation for convolutional neural networks

- For images, fully connected networks require many parameters. In CIFAR-10, the input size is $32 \times 32 \times 3 = 3072$, requiring 3072 weights for each neuron. For a more normal sized image that is 200×200, each neuron in the *first layer* would require 120000 parameters. This quickly gets out of hand. This network, with a huge number of parameters, would not only take long to train, but may also be prone to overfitting.

Biological inspiration

Principles of the convolutional neural network are inspired from neuroscience. Seminal work by Hubel and Wiesel in cat visual cortex in the 1960's (Nobel prize awarded in the 1980's) found that V1 neurons were tuned to the movement of oriented bars. Hubel and Wiesel also defined "simple" and "complex" cells, the computational properties of which were later well-described by linear models and rectifiers (e.g., Movshon and colleagues, 1978). Three key principles of neural coding in V1 are incorporated in convolutional neural networks (CNNs):

- V1 has a retinotopic map.
- V1 is composed of simple cells that can be adequately described by a linear model in a spatially localzed receptive field.
- V1 is composed of complex cells that respond to similar features as simple cells, but importantly are largely invariant to the position of the feature.

Biological inspiration (cont.)

From these discoveries, the following were incorporated into CNNs.

- Retinotopic map: CNNs are spatially organized, so that nearby cells act on nearby parts of the input image.
- Simple cells: CNNs use spatially localized linear filters, which are followed by thresholding.
- Complex cells: CNNs use pooling units to incorporate invariance to shifts of the position of the feature.

Some recent work has shown that artificial neurons in convolutional neural networks have similar hierarchical structure to the visual pathway. For example, deeper layers of a CNN can better predict firing rates from neurons in deeper layers of the visual processing pathway (see review by Yamins and DiCarlo, Nature Neurosci 2016).

Limitations of biological analogies

There are several limitations in these analogies. For example,

- CNNs have no feedback connections; neural populations are recurrently connected.
- The brain foveates a small region of the visual space, using saccades to focus on different areas.
- The output of the brain is obviously more than just image category classification.
- Pooling done by complex cells is an approximate way to incorporate invariance.

The convolution operation

The convolution of two functions, $f(t)$ and $g(t)$, is given by:

$$(f * g)(t) = \int_{-\infty}^{\infty} f(\tau)g(t-\tau)d\tau$$

In discrete time, this is given by:

$$(f * g)(n) = \sum_{m=-\infty}^{\infty} f(m)g(n-m)$$

Note, however, that in general CNNs don't use *convolution*, but instead use *cross-correlation*. Colloquially, instead of "flip-and-drag," CNNs just "drag." For real-valued functions, cross-correlation is defined by:

$$(f \star g)(n) = \sum_{m=-\infty}^{\infty} f(m)g(n+m)$$

We'll follow the field's convention and call this operation convolution.

Convolution in 2D

The 2D convolution (formally: cross-correlation) is given by:

$$(f * g)(i, j) = \sum_{m=-\infty}^{\infty} \sum_{n=-\infty}^{\infty} f(m, n) g(i + m, j + n)$$

This generalizes to higher dimensions as well. Note also: these "convolutions" are not commutative.

Example: Note, we assume that outside of each grid are zero values (that are not drawn). Now, dragging across the top row, we get:

1	1	1
1	1	1
1	1	1

$*$

w	x
y	z

$=$

z	$y+z$	$y+z$	y

Convolution in 2D (cont.)

Now, dragging across the second row:

1	1	1
1	1	1
1	1	1

$*$

w	x
y	z

$=$

z	$y+z$	$y+z$	y
$x+z$	$w+x+$ $y+z$	$w+x+$ $y+z$	$w+y$

Convolution in 2D (cont.)

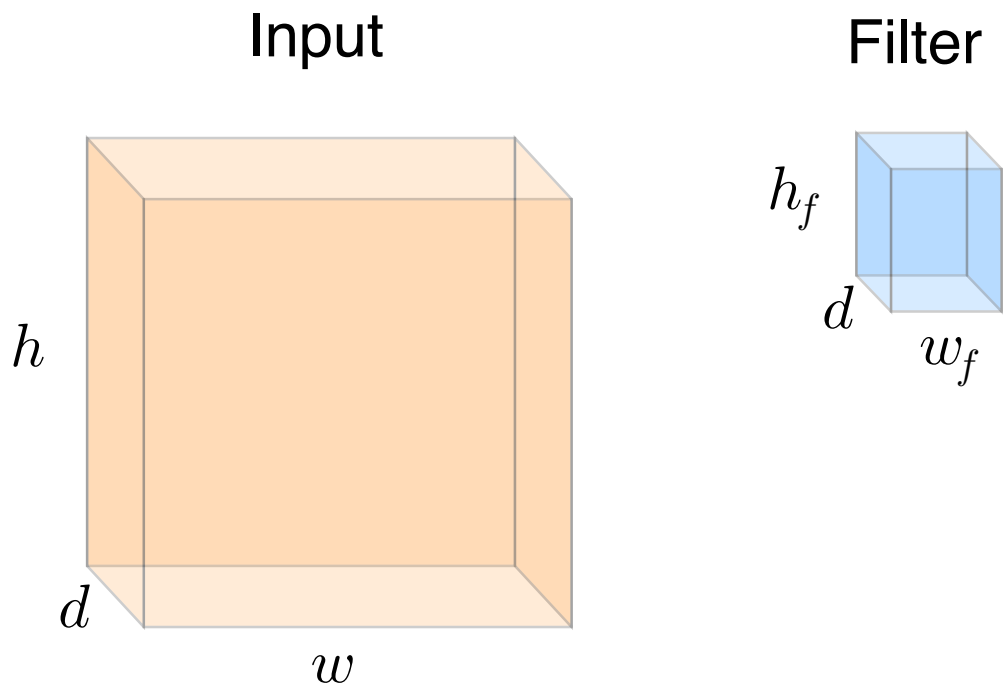
Finishing the convolution:

$$\begin{array}{|c|c|c|} \hline 1 & 1 & 1 \\ \hline 1 & 1 & 1 \\ \hline 1 & 1 & 1 \\ \hline \end{array} * \begin{array}{|c|c|} \hline w & x \\ \hline y & z \\ \hline \end{array} = \begin{array}{|c|c|c|c|} \hline z & y+z & y+z & y \\ \hline x+z & w+x+y+z & w+x+y+z & w+y \\ \hline x+z & w+x+y+z & w+x+y+z & w+y \\ \hline x & w+x & w+x & w \\ \hline \end{array}$$

Convolutional layer

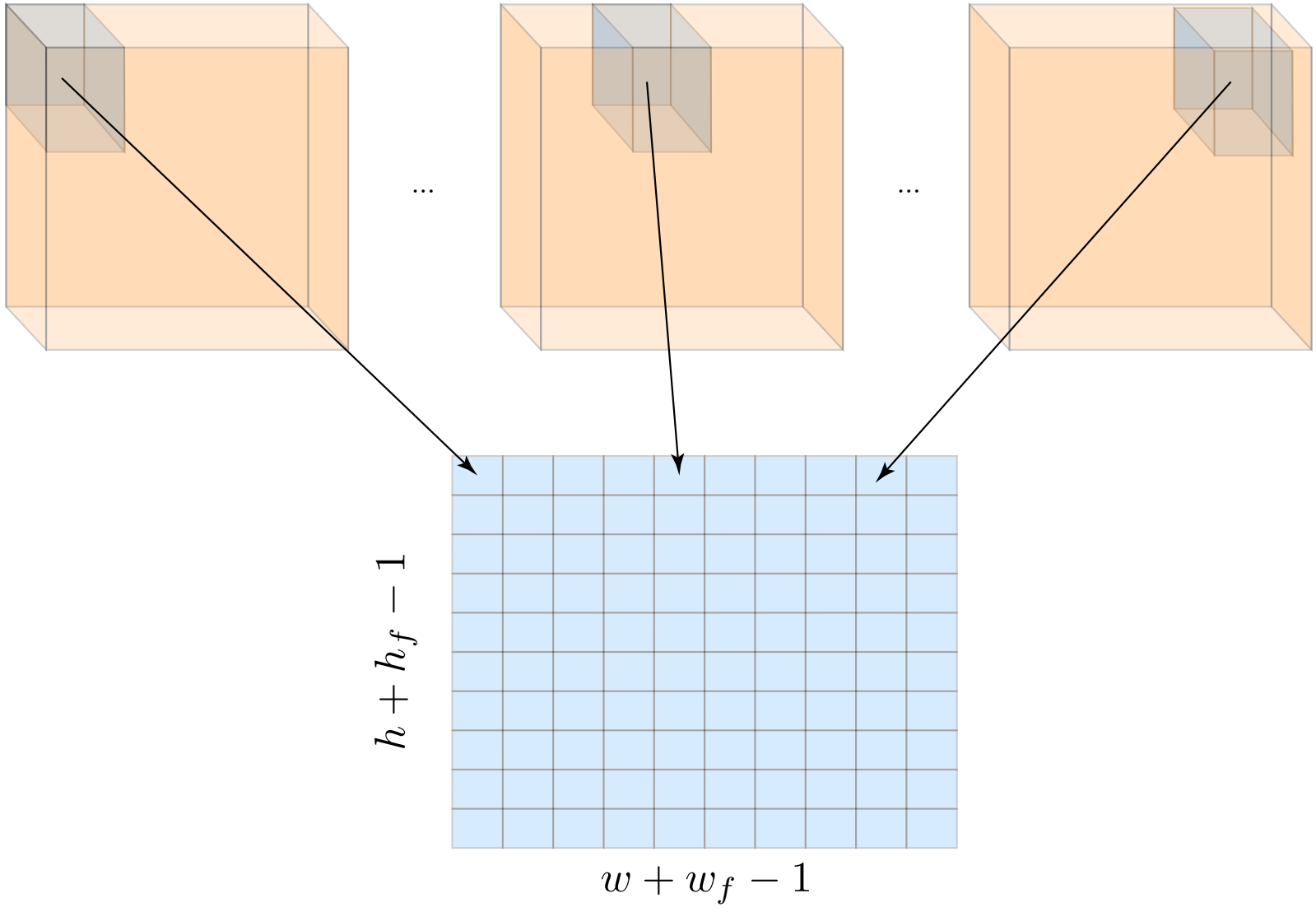
This convolution operation (typically in 3D, since images often come with a width and height as well as *depth* for the R, G, B channels) defines the "convolutional layer" of a CNN. The convolutional layer defines a collection of filters (or activation maps), each with the same dimension as the input.

- Say the input was of dimensionality (w, h, d).
- Say the filter dimensionality is (w_f, h_f, d). So that the filter operates on a small region of the input, typically $w_f < w$.
- The depths being equal means that the output of this convolution operation is 2D.



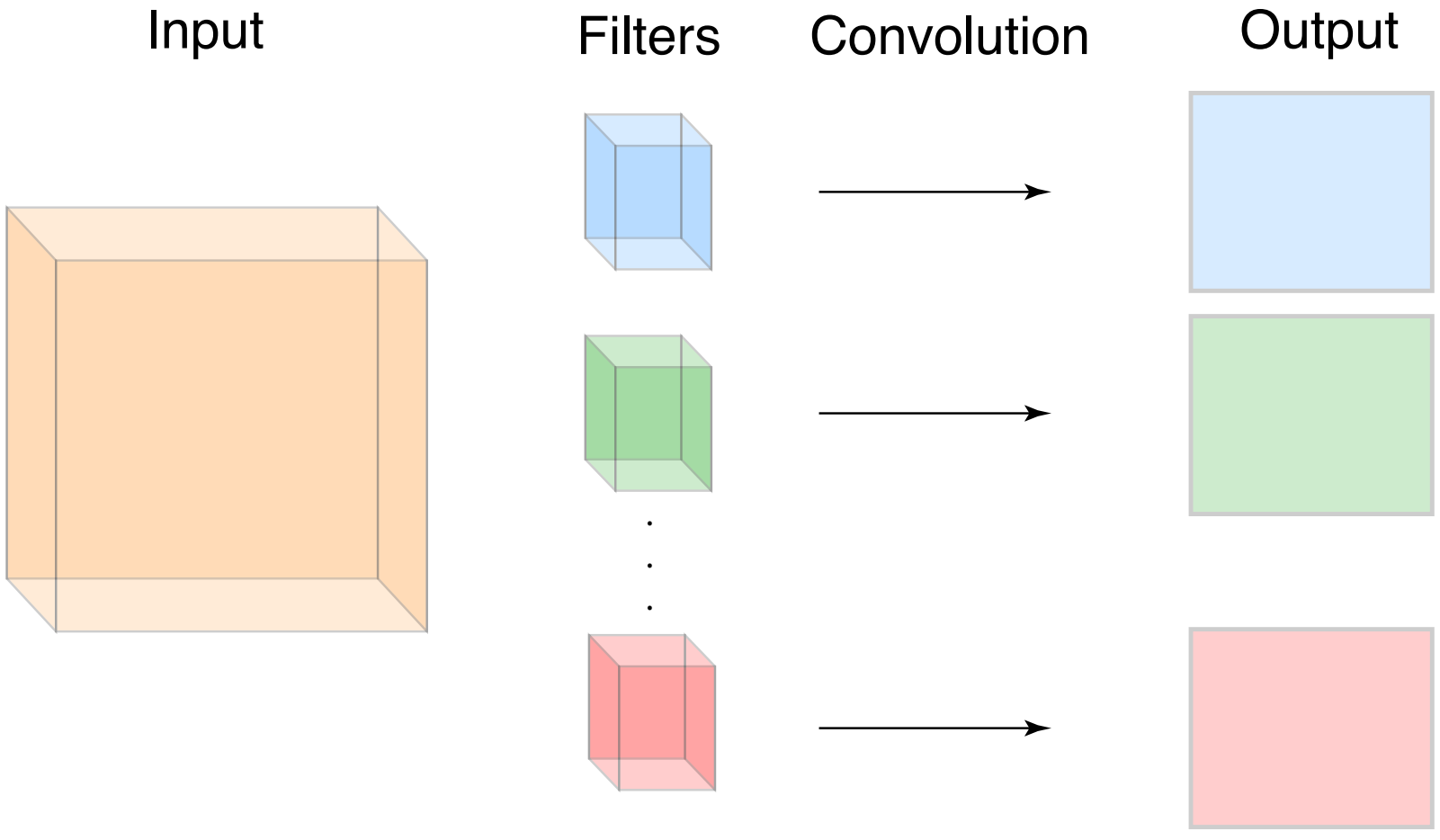
Convolutional layer (cont.)

After performing the convolution, the output is $(w + w_f - 1, h + h_f - 1)$.



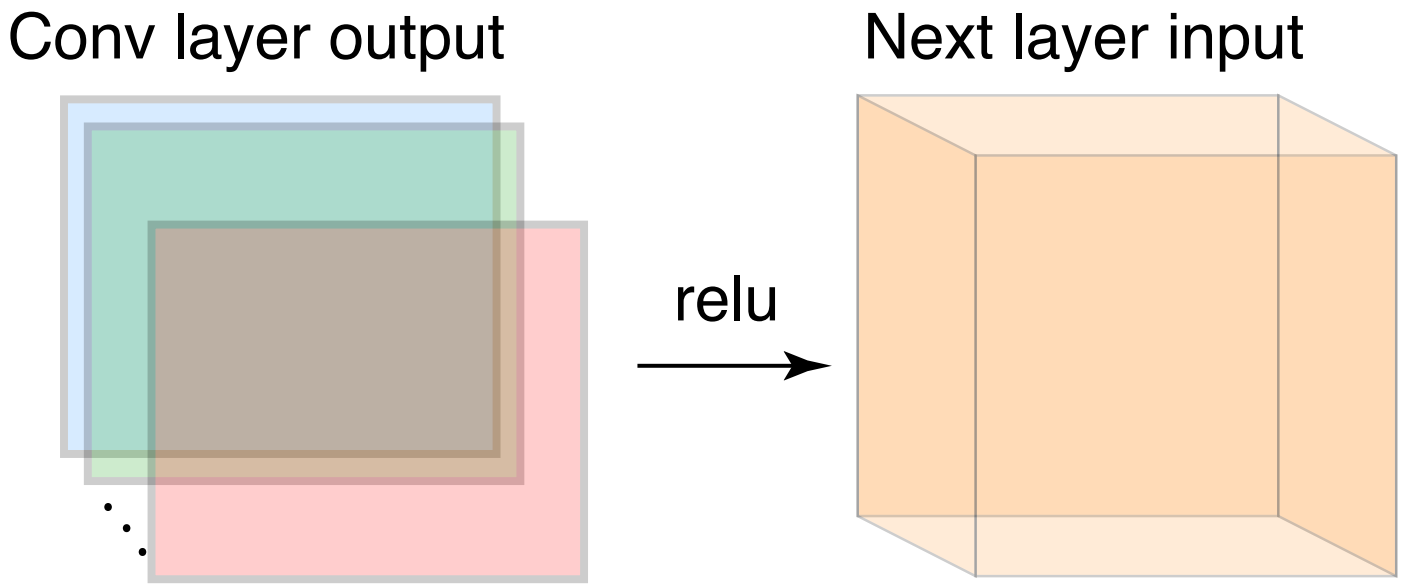
Convolutional layer (cont.)

Now, we don't have just one filter in a convolutional layer, but multiple filters. We call each output (matrix) a slice.



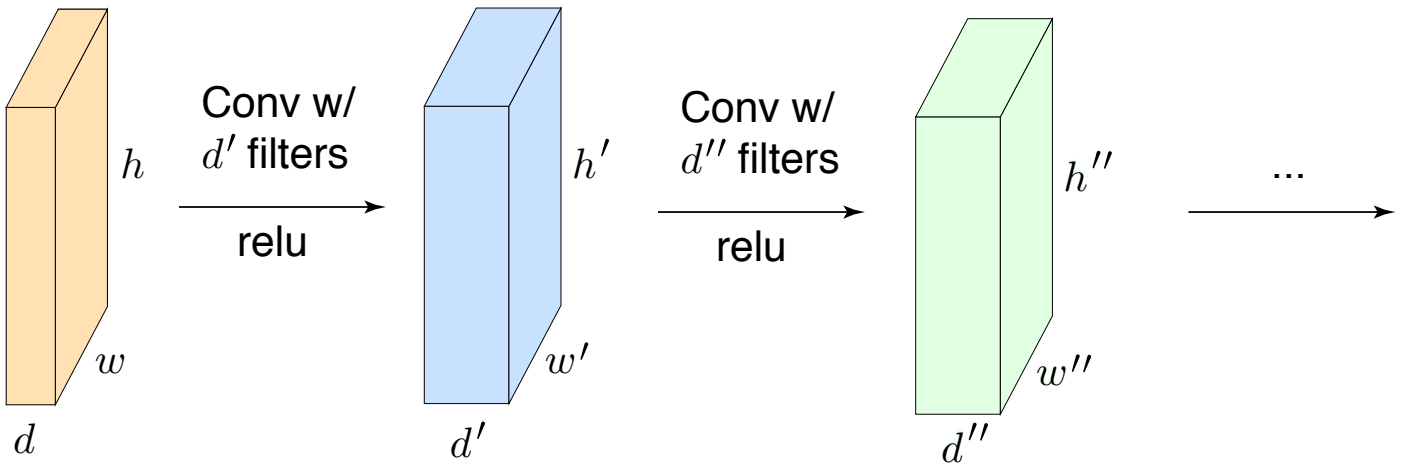
Convolutional layer (cont.)

The output slices of the convolution operations with each filter are composed together to form a $(w + w_f - 1, h + h_f - 1, n_f)$ tensor, where n_f is the number of filters. The output is then passed through an activation nonlinearity, such as $\mathrm{ReLU}(\cdot)$. This then acts as the input to the next layer.



Composition of convolutional layers

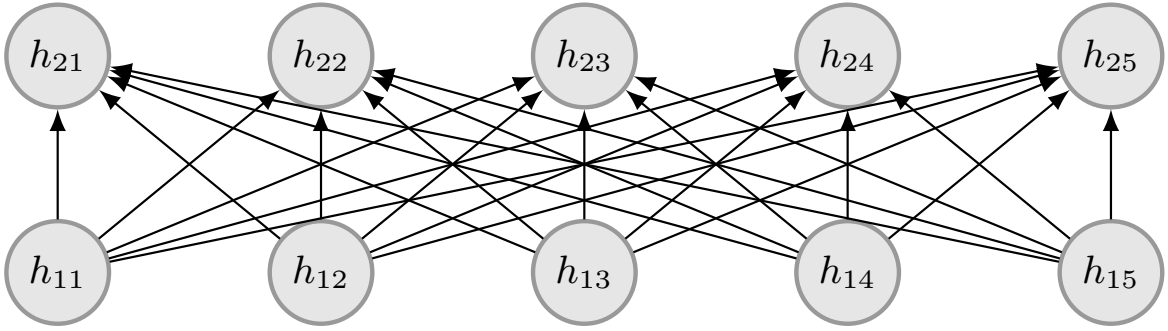
These layers can then be composed, which comprise a large component of the CNN.



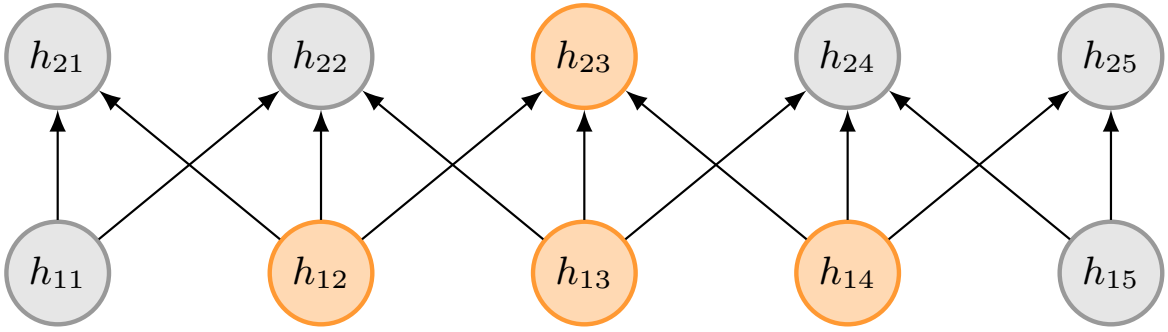
Convolutional layers have sparse interactions

Convolutional layers have *sparse interactions* or *sparse connectivity*. The idea of sparse connectivity is illustrated below, where each output is connected to a small number of inputs.

Fully connected layers:

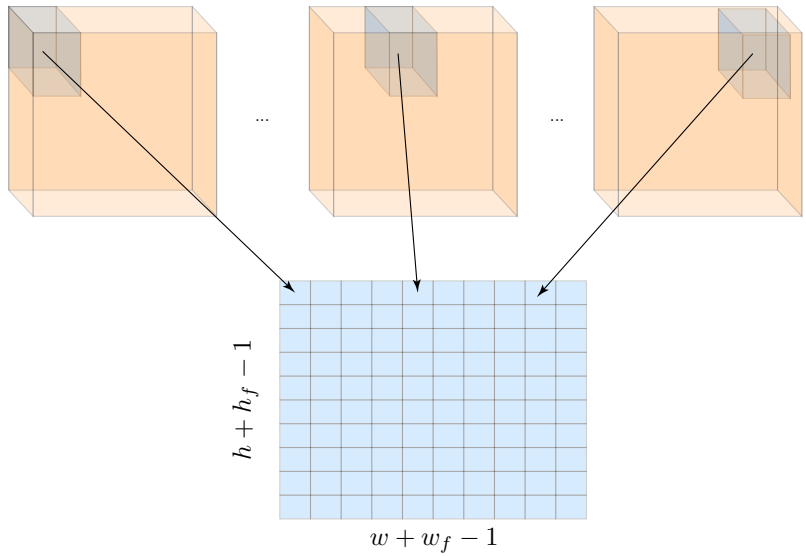


Sparsely connected layers:



Convolutional layers have sparse interactions

- The convolutional layer has sparse connectivity, since each output is only connected to a small region of inputs. This is apparent from the prior image:



  Equivalently, many of the input-to-output weights are zero.
- Decreasing the filter size will make the connections more sparse.
- This is relevant for image processing because images have small but meaningful features such as edges.

Convolutional layers have sparse interactions (cont.)

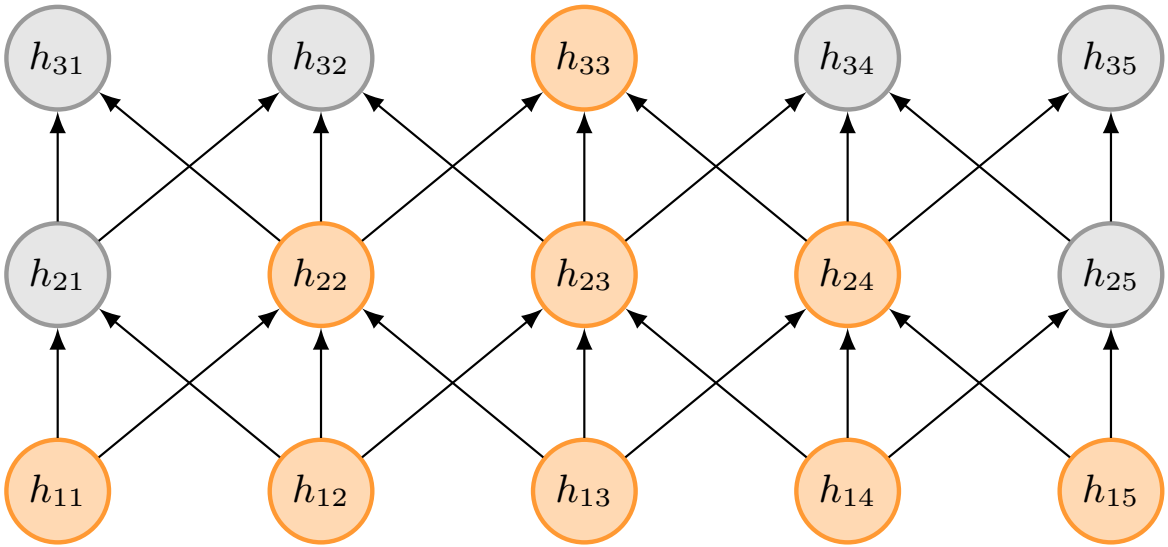
Sparse interactions aid with computation.

- **Sparse interactions reduce computational memory.** In a fully connected layer, each neuron has $w \cdot h \cdot d$ weights corresponding to the input. In a convolutional layer, each neuron has $w_f \cdot h_f \cdot d$ weights corresponding to the input. Moreover, in a convolutional layer, every output neuron in a slice has the *same* $w_f \cdot h_f \cdot d$ weights (more on this later). As there are far fewer parameters, this reduces the memory requirements of the model.
- **Sparse interactions reduce computation time.** If there are m inputs and n outputs in a hidden layer, a fully connected layer would require $\mathcal{O}(mn)$ operations to compute the output. If each output is connected to only k inputs (e.g., where $k = w_f \cdot h_f \cdot d$) then the layer would require $\mathcal{O}(kn)$ operations to compute the output.

Convolutional layers have sparse interactions (cont.)

A concern of sparse interactions is that information information from different parts of the input may not interact. For example, a self-driving car should know where obstacles are from all over the image to avoid them.

This argues that networks should be composed of more layers, since units in deeper layers indirectly interact with larger portions of the input.



Convolutional layers share parameters

Convolutional layers have shared parameters (or *tied weights*), in every output neuron in a given slice uses the same set of parameters in the filter.

Example: Consider an input that is $(32 \times 32 \times 3)$. We have two architectures; in the fully connected architecture, there are 500 output neurons at the first layer. In the convolutional neural net there are 4 filters that are all $4 \times 4 \times 3$. (1) How many output neurons are there in the convolutional neural network, assuming that the convolution is only applied in regions where the filter fully overlaps the kernel? (2) How many parameters are in each model?

(1) Each output slice is $(32 - 4 + 1, 32 - 4 + 1)$, which is 841 neurons. With 4 slices, there are $3,364$ output neurons in the CNN. What is a con of having many more neurons?

(2) The fully connected network has $(32 \times 32 \times 3) \times 500 = 1,536,000$ parameters. The convolutional neural network has $4 \times (4 \times 4 \times 3) = 192$ parameters.

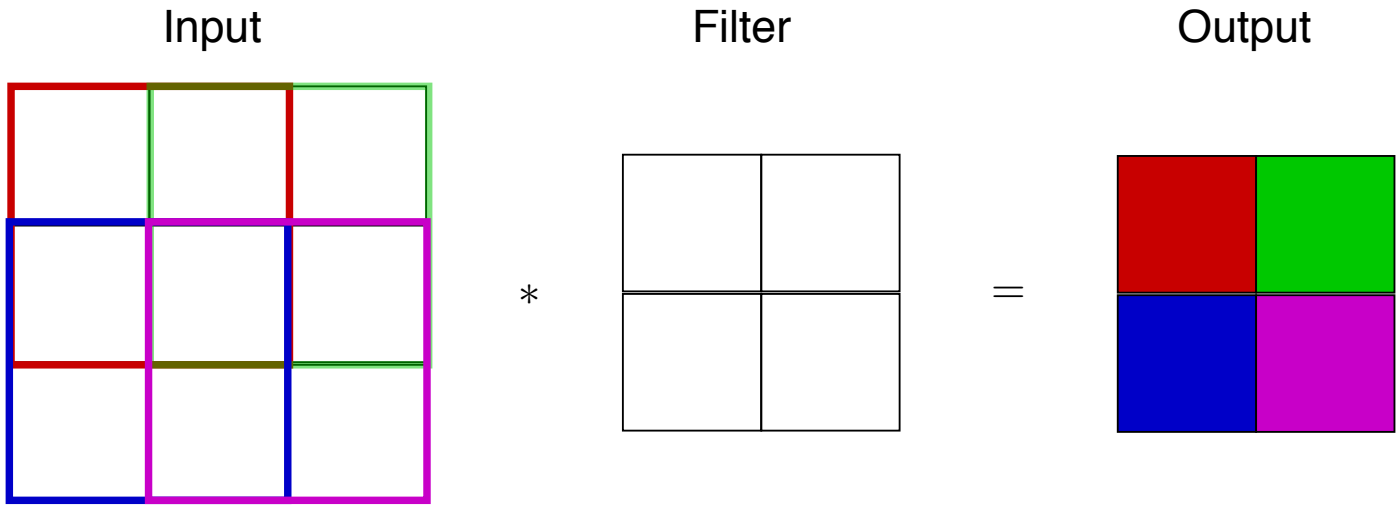
Convolutional layers can handle variable sized inputs

Convolutional neural networks can process inputs with varying sizes or spatial extents. In a fully conected layer, each layer specifies parameters $\mathbf{W}, \mathbf{b}$ such that $\mathbf{z} = \mathbf{W}\mathbf{x} + \mathbf{b}$, and thus the inputs and output scales are defined. In a convolutional layer, the parameters are convolutional filters, which are then convolved with an input.

- If the input changes in size, the convolution operation can still be performed.
- The dimensions of the output will be different, which may be okay depending on your application.
- If the output dimensions are a fixed size, some modifications may be made to the network (e.g., if the output size needs to be decreased, pooling layers, described later, can scale with the network input size to create the correct output size).

Convolution padding

Absent of any modifications, each convolutional layer will grow the width and height of the data at each layer by the filter length minus 1. In practice, sometimes the convolution is restrained to regions where the filter fully overlaps the input.



However, now we run into another problem: if we only constrain the kernel to areas where it fully overlaps, the resulting output will be of dimension:

$$(w - w_f + 1, h - h_f + 1)$$

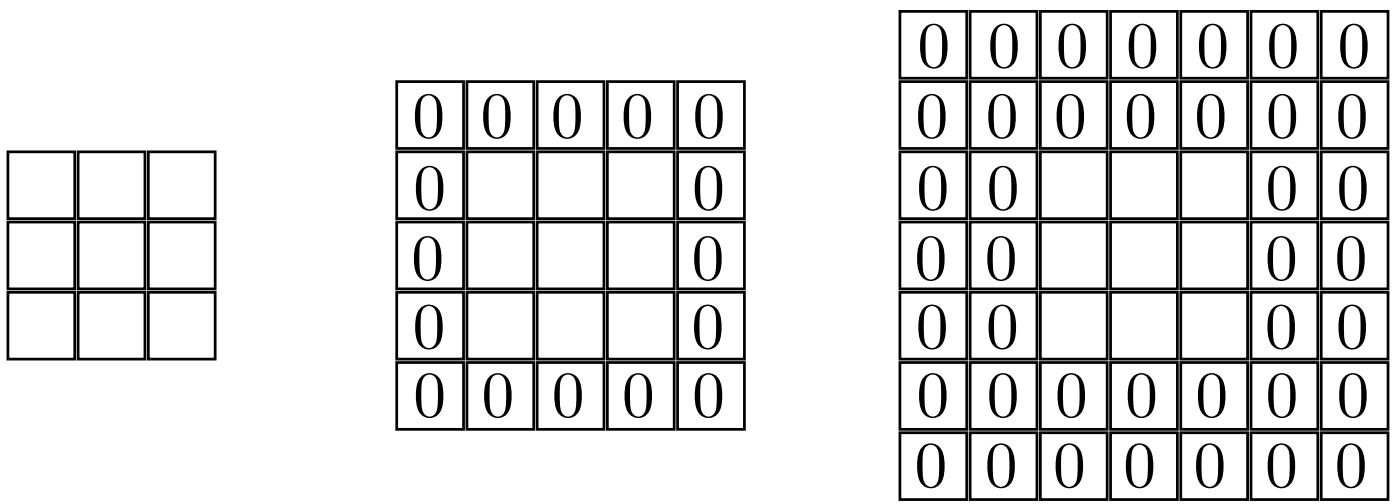
And thus, the input data constrains the number of convolutional layers that could be used.

Convolution padding (cont.)

For the rest of the class, we'll assume convolution is only applied when the filter completely overlaps the input. Now, the input and output will remain the same size is the input is zero-padded with $w_f - 1$ total zeros (i.e., $(w_f - 1)/2$ zeros on the left and right of the matrix) and the height is zero-padded with $h_f - 1$ total zeros (i.e., $(h_f - 1)/2$ zeros on the top and bottom of the matrix).

Usually, we specify a variable pad which is the amount of zeros to pad on each border.

$\texttt{pad} = 0$ $\texttt{pad} = 1$ $\texttt{pad} = 2$



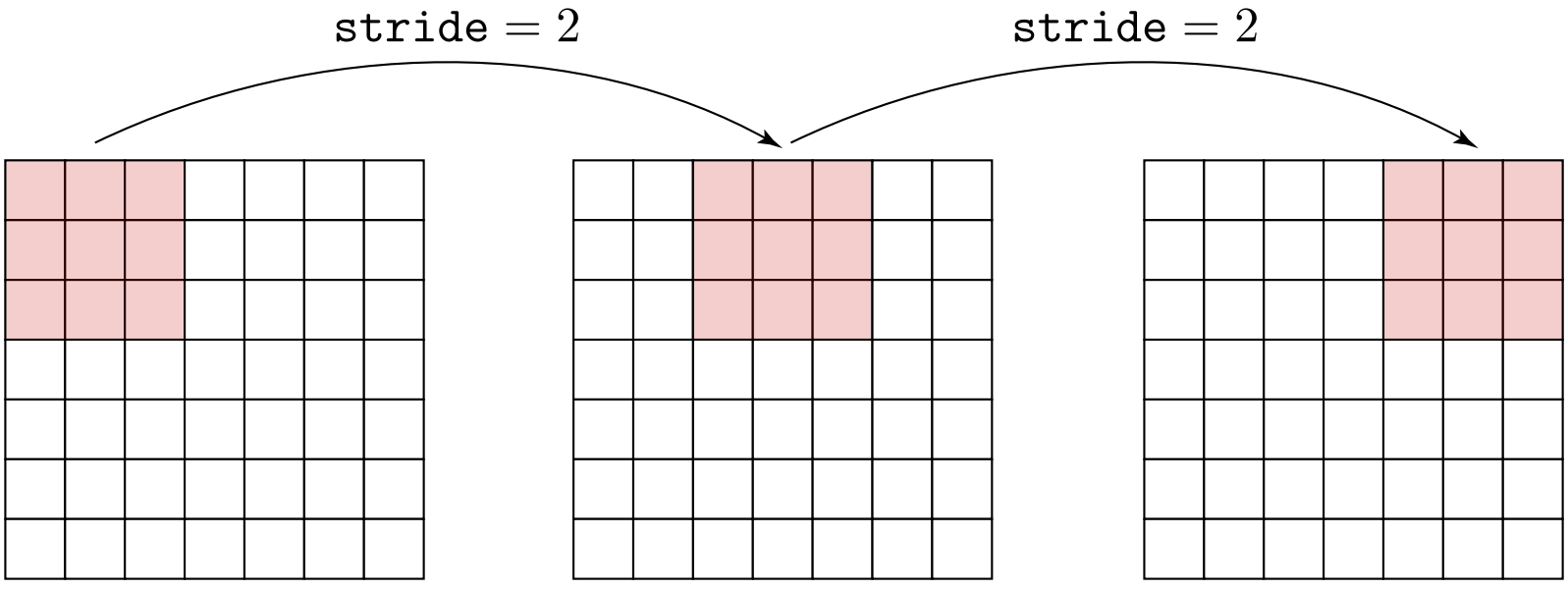
The output of the convolution is now $(w - w_f + 1 + 2\texttt{pad}, h - h_f + 1 + 2\texttt{pad})$.

Convolution padding (cont.)

It is worth noting that Goodfellow et al. report that the optimal amount of zero padding (in terms of test accuracy) is somewhere between $\texttt{pad} = 0$ and the pad that causes the output and input to have the same width and height.

Convolution stride

Another variable to control is the stride, which defines how much the filter moves in the convolution. For normal convolution, $\texttt{stride} = 1$, which denotes that the filter is dragged across every part of the input. Consider a 7×7 input with a 3×3 filter. If convolution is only applied where the kernel overlaps the input, the output is 5×5. With $\texttt{stride} = 2$, the output is 3×3.



Convolution stride (cont.)

The stride must be sensible. In the 7×7 input with 3×3 filter example, it is not correct to use $\texttt{stride} = 3$ as it is not consistent with input. However, one can zero-pad the input so that the stride is appropriate. The output size after accounting for stride and padding is:

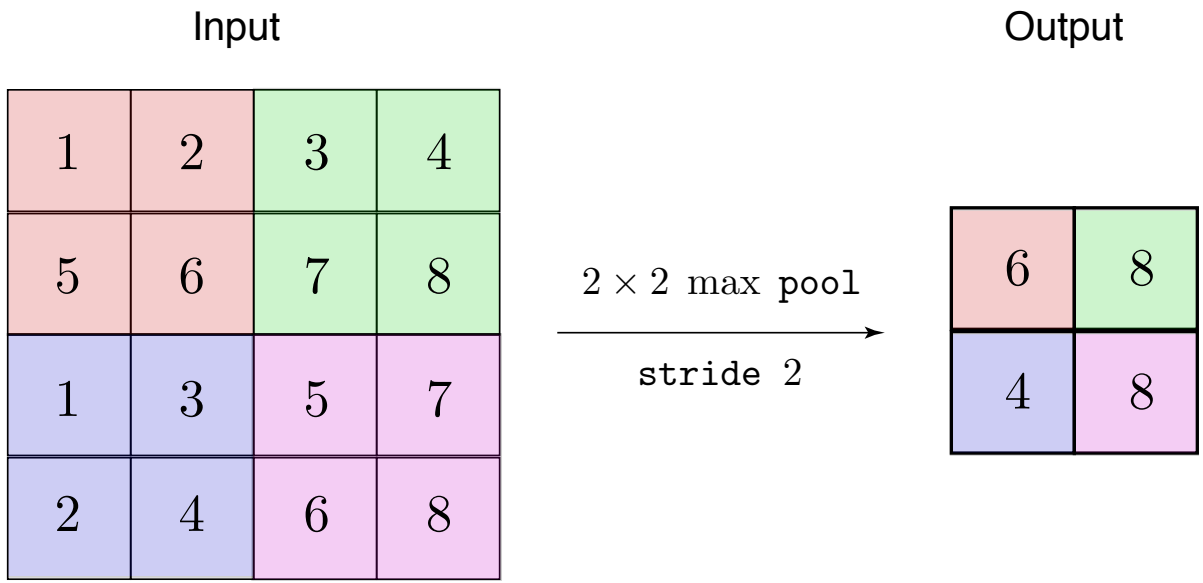
$$\left(\frac{w - w_f + 2\text{pad}}{\texttt{stride}} + 1, \frac{h - h_f + 2\text{pad}}{\texttt{stride}} + 1 \right)$$

Be careful when incorporating stride, as the data representation size will shrink in a divisive manner.

Pooling layer

CNNs also incorporate pooling layers, where an operation is applied to all elements within the filtering extent. This corresponds, effectively, to downsampling.

The pooling filter has width and height (w_p, h_p) and is applied with a given stride. It is most common to use the $\max()$ operation as the pooling operation.



Pooling layer (cont.)

A few notes on pooling layers.

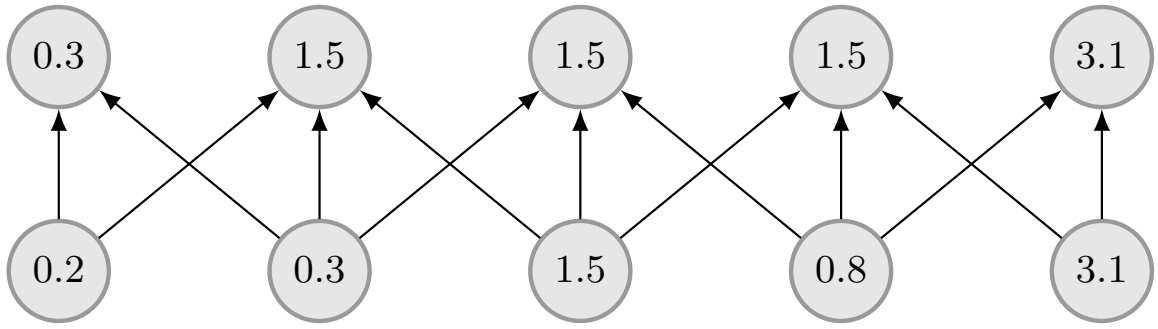
- The resulting output has dimensions

$$\left(\frac{w - w_p}{\texttt{stride}} + 1, \frac{h - h_p}{\texttt{stride}} + 1\right)$$

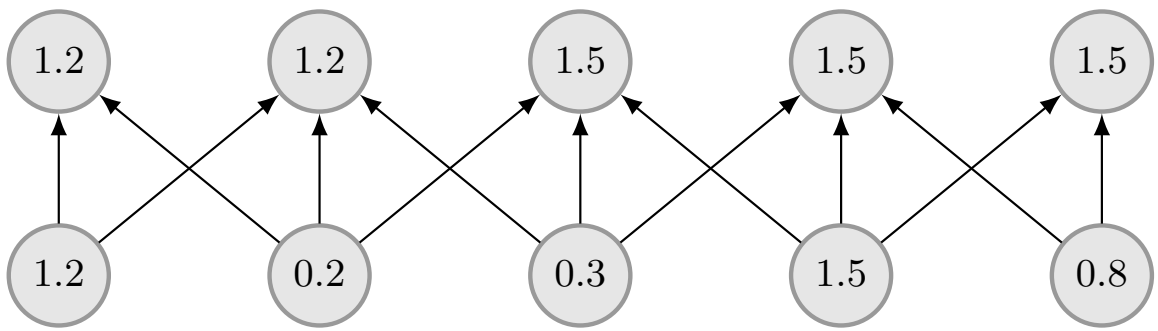
- Pooling layers introduce *no* additional parameters to the model.
- Pooling layers are intended to introduce small spatial invariances.

Pooling layer (cont.)

Invariance example:



Inputs shifted by 1. Five inputs change but only three outputs change.

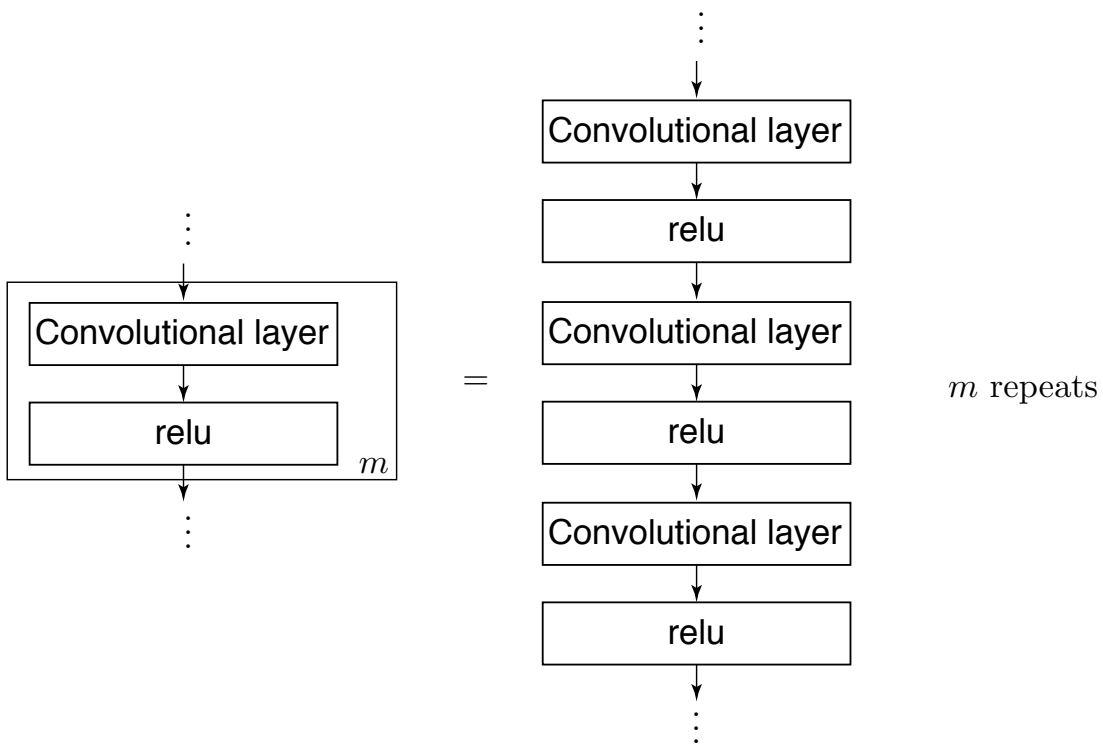


Convolutional neural network architectures

Typically, convolutional neural networks are comprised of units that are composed of:

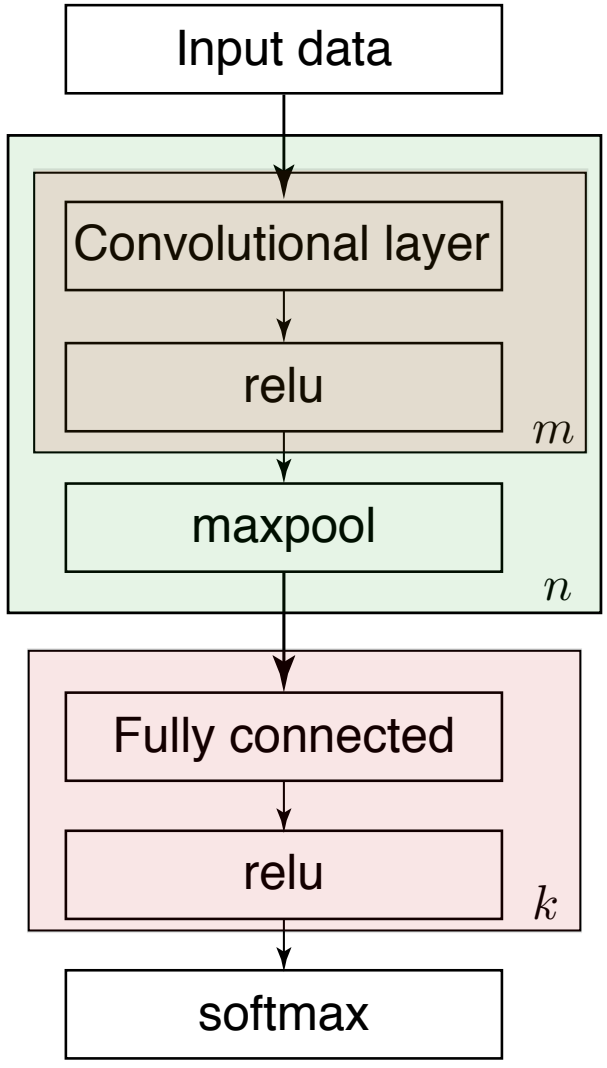
- Several paired convolutional-relu layers.
- Potentially one max pooling layer.

These units are cascaded. Finally, a CNN may end with a fully connected-relu layer, followed by a softmax classifier. To illustrate this, we borrow the plate notation from graphical models, where:



Convolutional neural network architectures (cont.)

With this in mind, the following describes a fairly typical CNN architecture.



Convolutional neural network architectures (cont.)

Beyond this, there are many architectural considerations. Indeed, much progress (as measured by ImageNet performance) has not been due to new optimization techniques or regularizations, but rather has largely been a result of architecture. We'll talk about case studies in class. These include:

- AlexNet (Krizhevsky et al., 2012) and ZFNet (Zeiler & Fergus, 2013)
- VGGNet and small filters (Simonyan & Zisserman, 2014)
- GoogLeNet and inception layers (Szegedy et al., 2014)
- ResNet and residual layers (He et al., 2015)
- FractalNet and the importance of reducing effective depth (Larsson et al., 2017)