

Deep Learning for Computer Vision

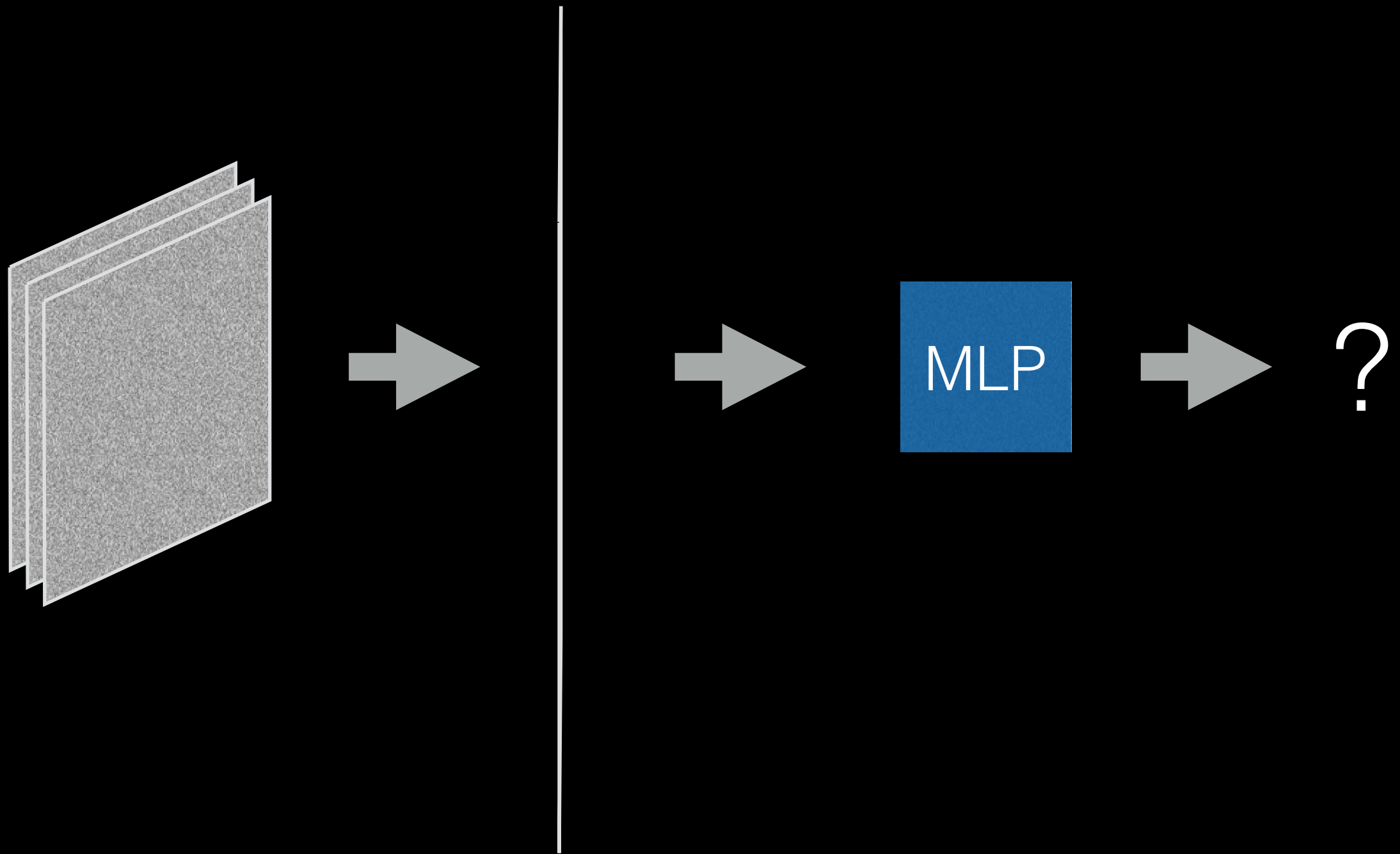
Lecture 9: Convolutional Neural Networks (CNNs)

Peter Belhumeur

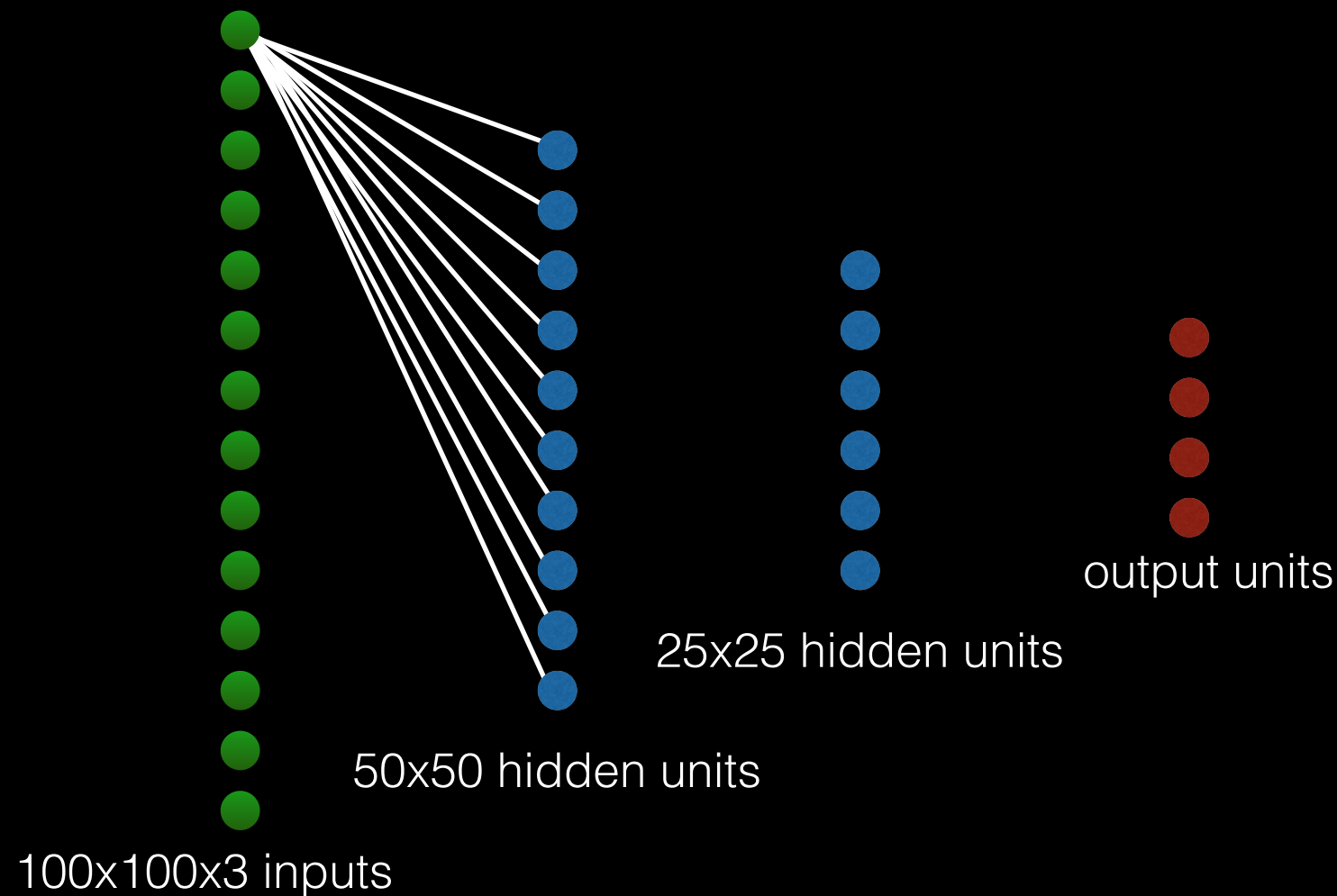
Computer Science
Columbia University

Finally, we get to images...

What if we just vectorized images and stuffed these into a MLP?

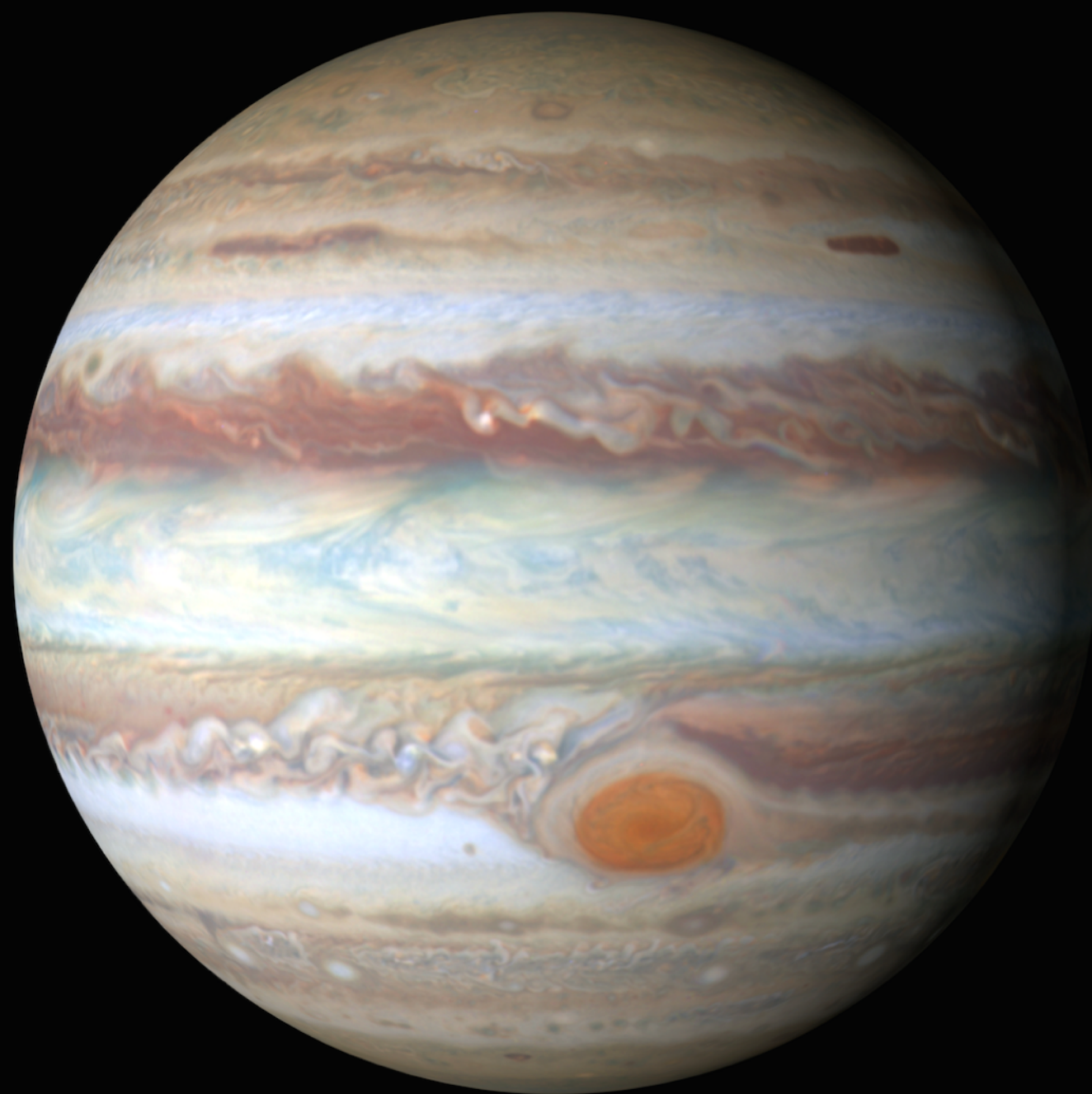


Too many weights and connections!



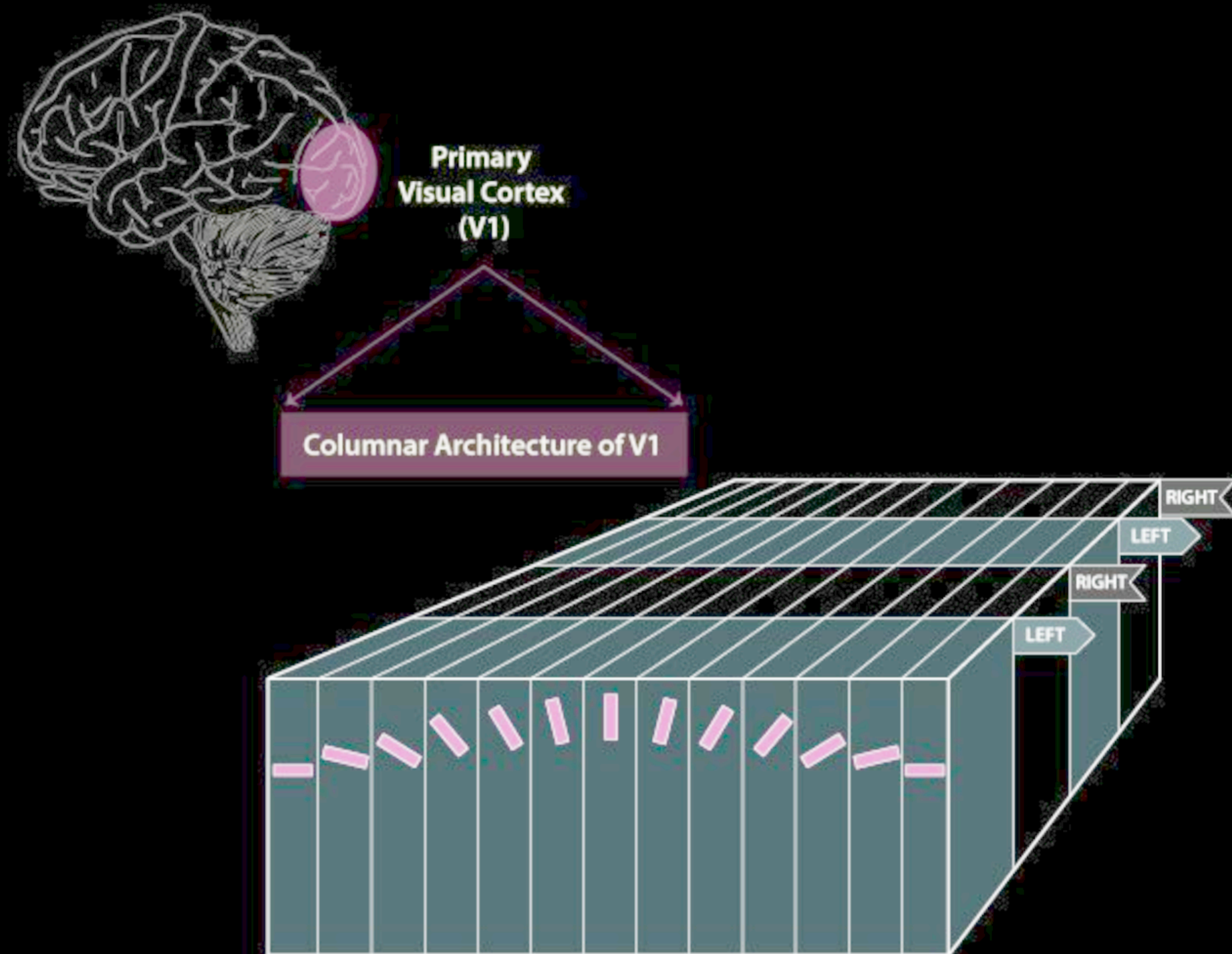
- This fully connected hidden layer might have 75 million weights!
- And this is just for a thumbnail image.





Remember Hubel and Wiesel's cat!

Early processing in cat visual cortex looks a lot like convolutions!



Edges and blobs

- Early stages of processing in cat visual cortex looks like it is performing ***convolutions*** that are looking for oriented edges and blobs
- Certain cells are looking for edges with a particular orientation at a particular spatial location in the visual field.

Convolution in 1D

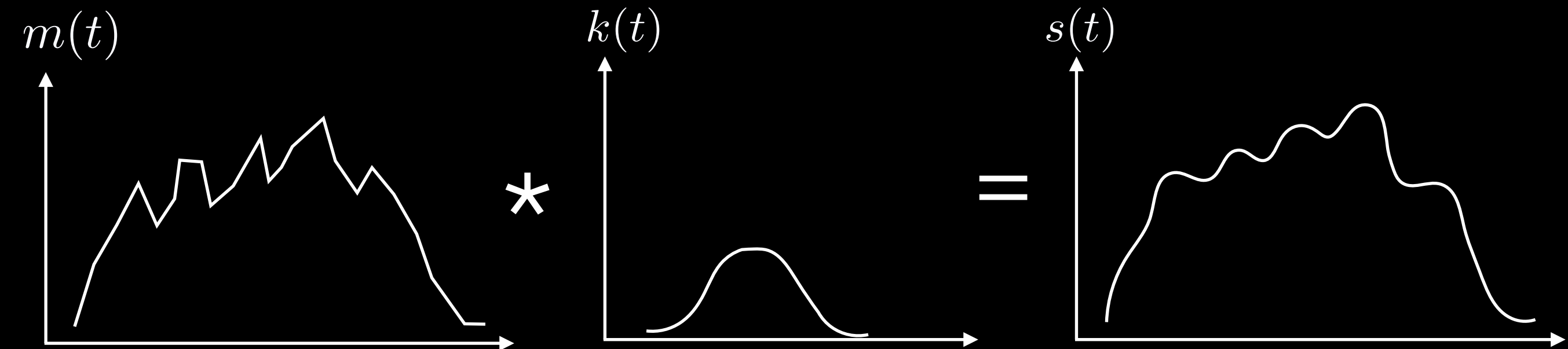
measurement kernel

The diagram illustrates the 1D convolution equation. At the top, the words "measurement" and "kernel" are positioned above the terms $m(r)$ and $k(t-r)$ respectively in the integral equation. Arrows point from "measurement" to $m(r)$ and from "kernel" to $k(t-r)$. Below the integral equation, the expression $= (m * k)(t)$ is shown. An arrow points from the text "convolution operator" at the bottom to the asterisk $*$ in this expression.

$$s(t) = \int_{-\infty}^{\infty} m(r)k(t-r)dr$$
$$= (m * k)(t)$$

convolution operator

Convolution in 1D



Discrete Convolution in 1D

measurement kernel (or filter)

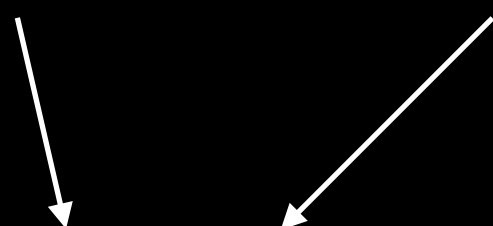
$$s(t) = \sum_{-\infty}^{\infty} m(r)k(t-r)$$

convolution operator

The diagram illustrates the 1D discrete convolution equation. At the top, the words 'measurement' and 'kernel (or filter)' are positioned above the terms $m(r)$ and $k(t-r)$ respectively in the summation formula. Arrows point from these labels to their corresponding terms. Below the summation formula, the equation is simplified to $= (m * k)(t)$. An arrow points from the text 'convolution operator' at the bottom to the asterisk symbol in this simplified equation.

Convolution is commutative

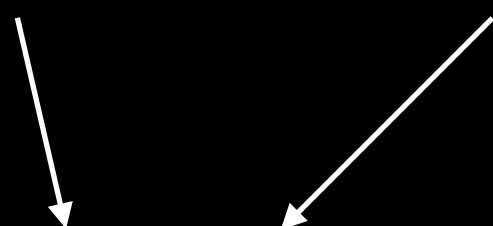

measurement kernel


$$\begin{aligned}s(t) &= (m * k)(t) \\ &= (k * m)(t) \\ &= \sum_{-\infty}^{\infty} k(r)m(t - r)\end{aligned}$$

Kernel is smaller than measurement so better to sum over it!

In practice, don't flip!

measurement kernel

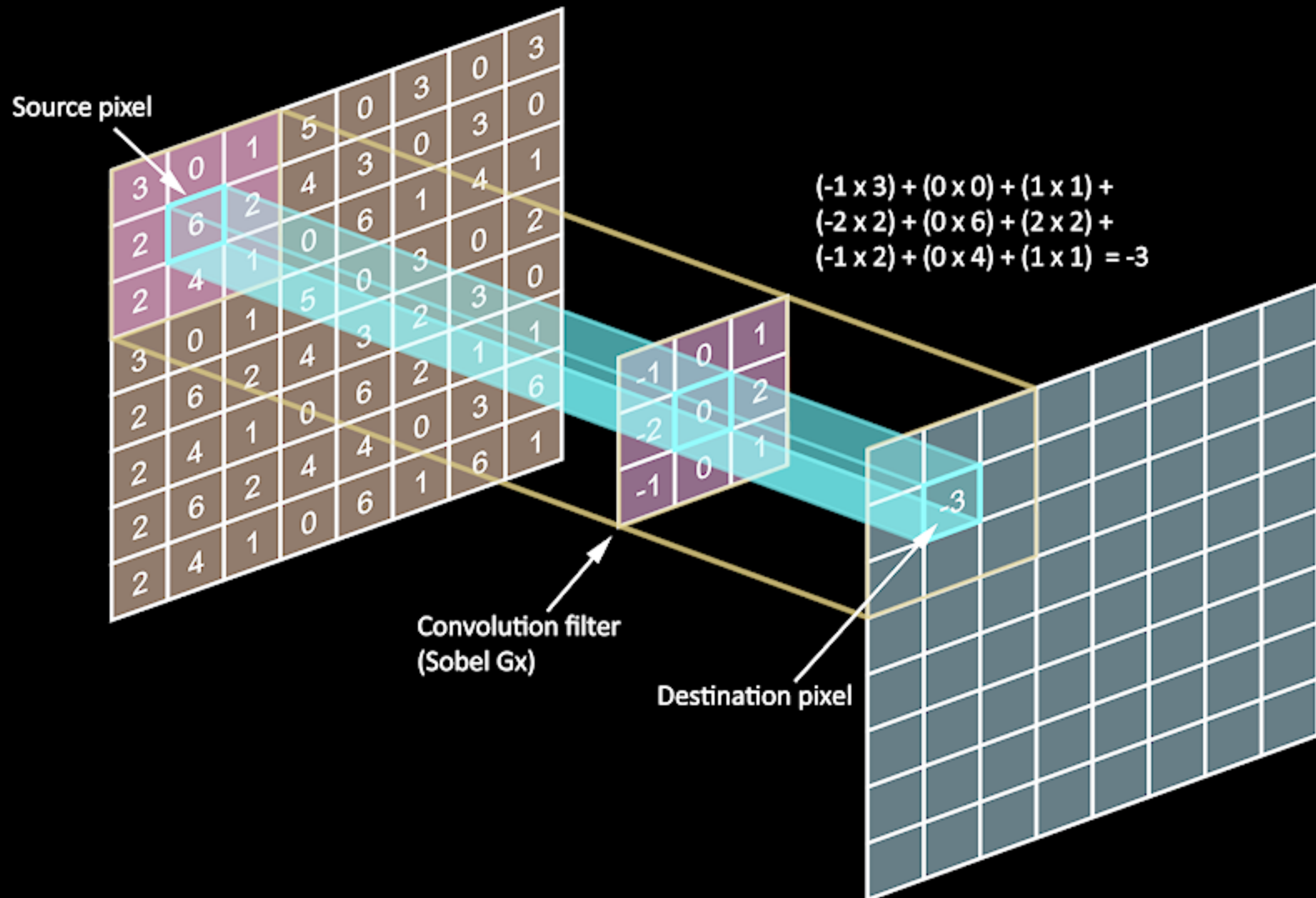

$$\begin{aligned}s(t) &= (m * k)(t) \\ &= \sum_r m(t + r)k(r)\end{aligned}$$


Only need to sum where kernel is non-zero!

Convolution in 2D

$$\begin{aligned} S(i, j) &= (I * K)(i, j) \\ &= \sum_m \sum_n I(i + m, j + n) K(m, n) \end{aligned}$$

Convolution in 2D



Convolution with 2D Kernel



*

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

=



Convolution with 2D Kernel



*

-1	-1	-1
-1	8	-1
-1	-1	-1

=



Convolution with 2D Kernel



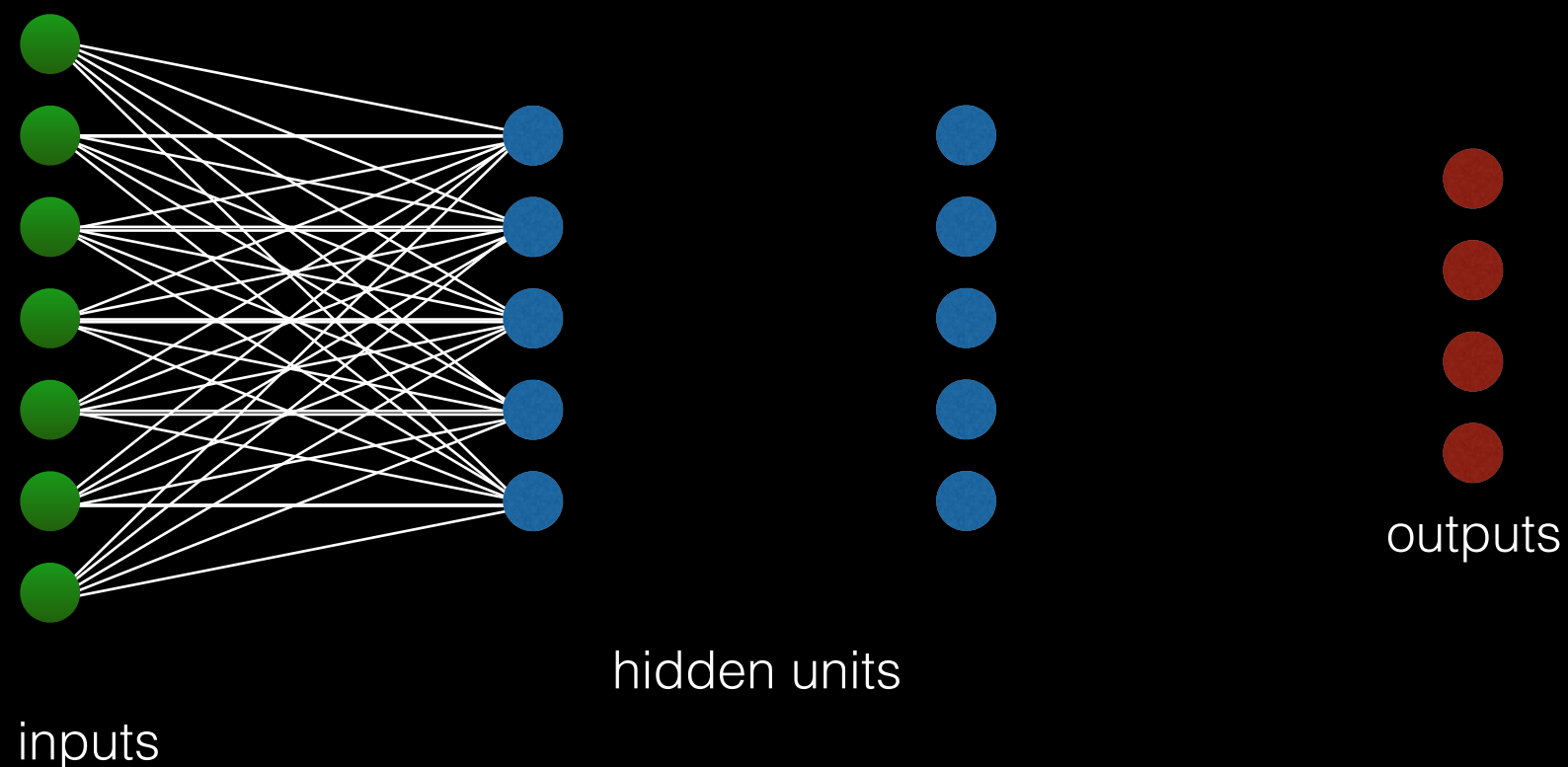
*

1	2	1
2	4	2
1	2	1

/ 16 =

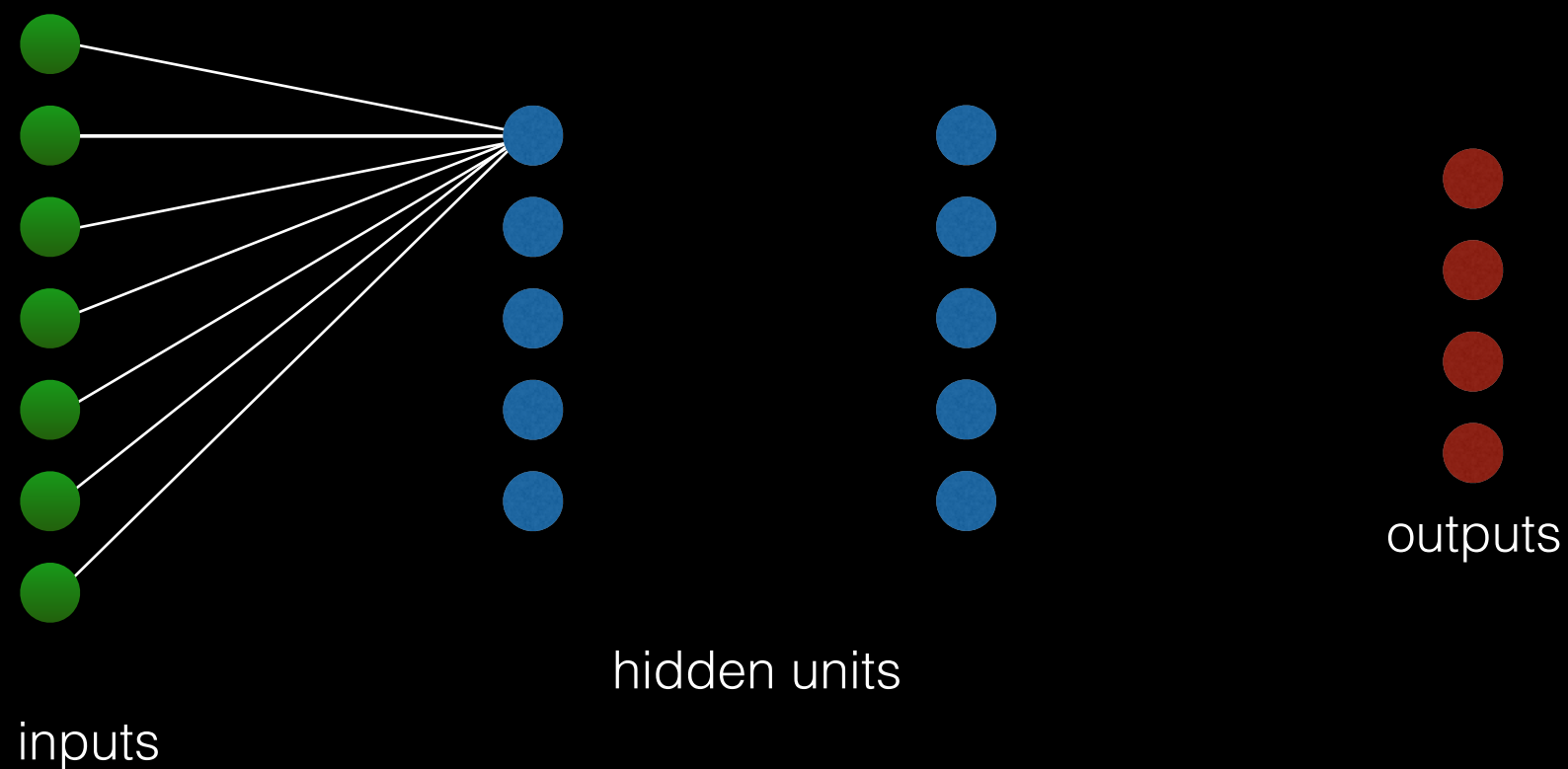


Fully Connected (FC) Layer



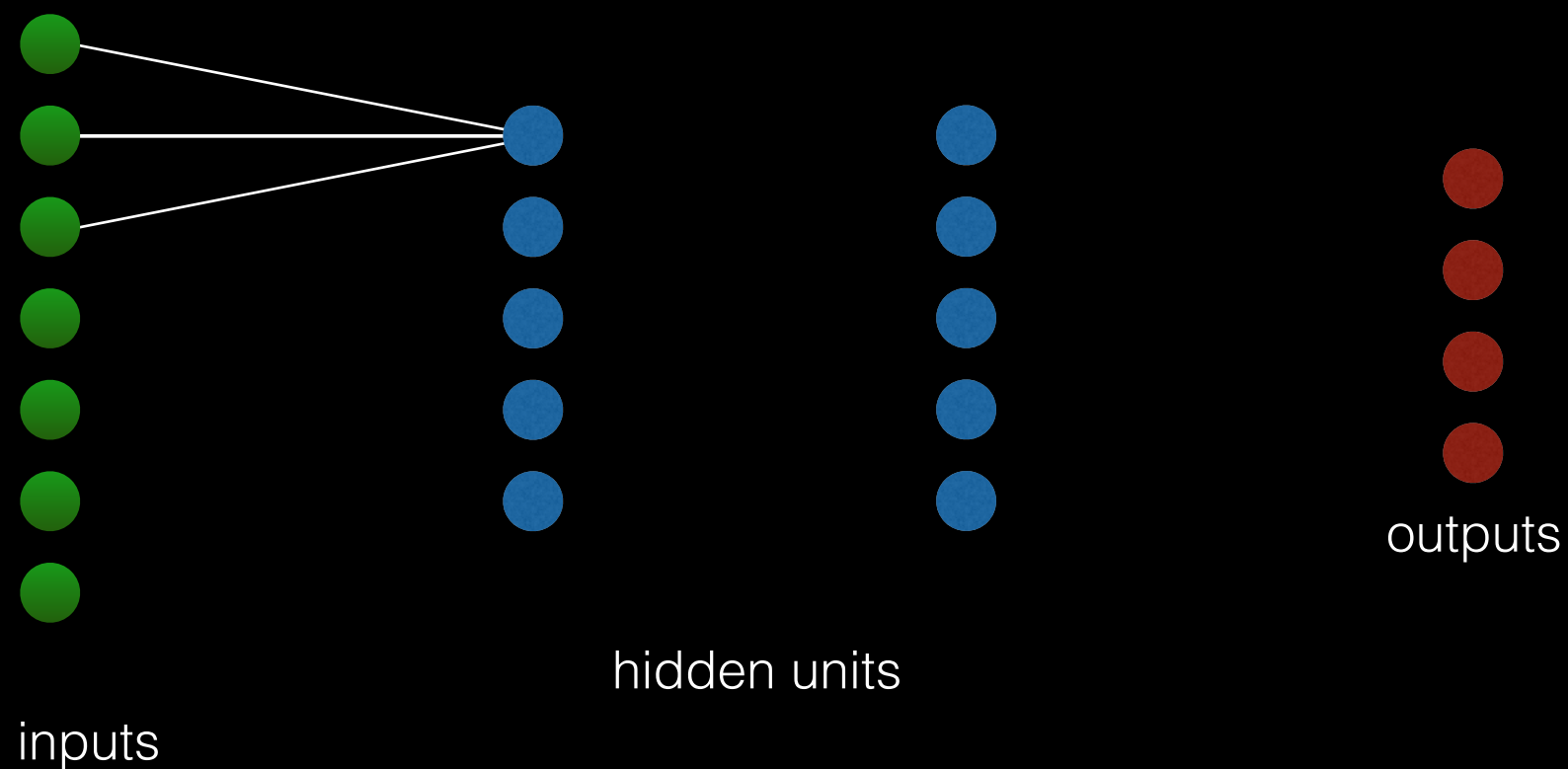
Every input unit is connected to every output unit.

Fully Connected (FC) Layer



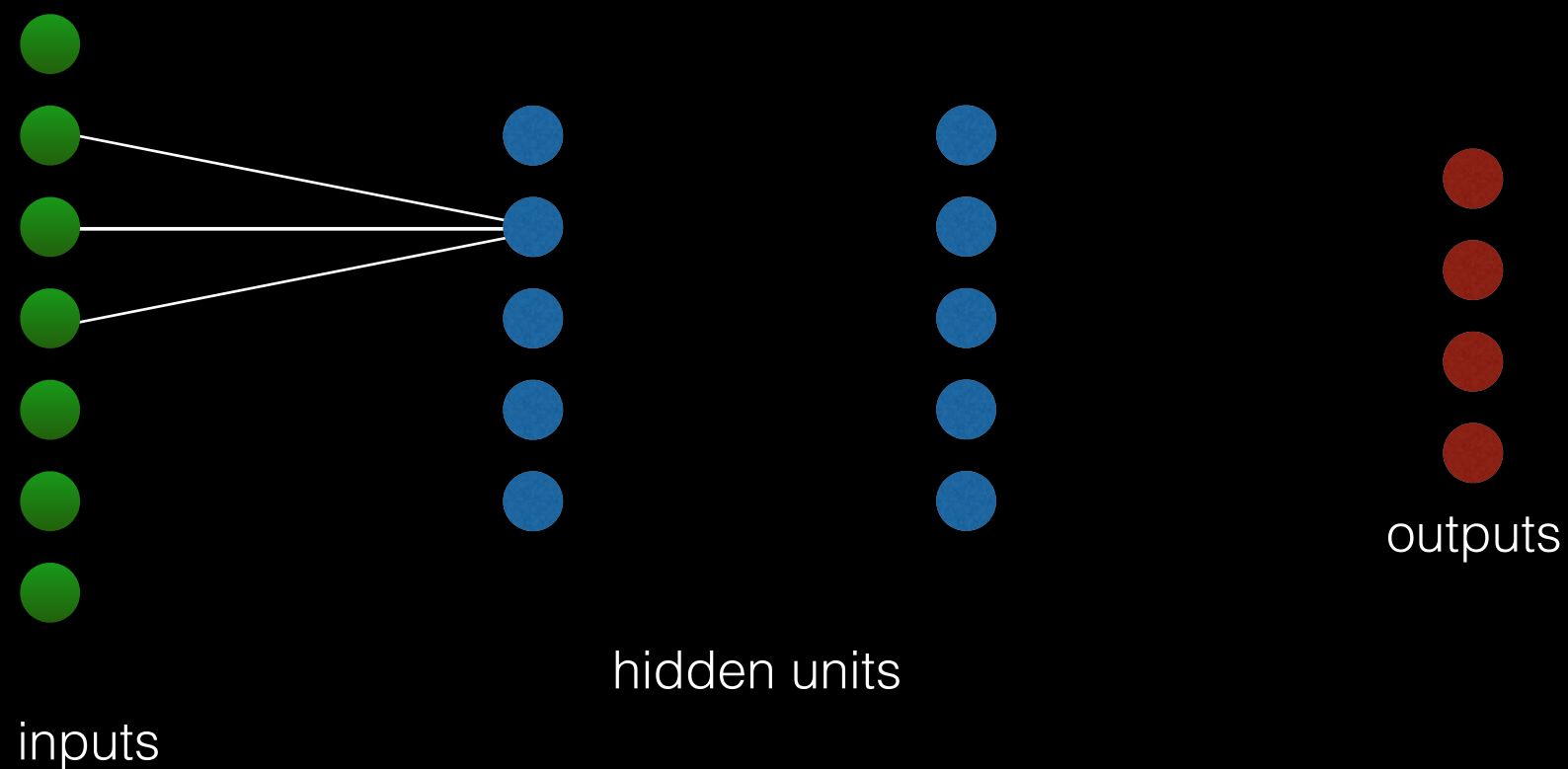
Consider a hidden unit: it connects to all units from the previous layer.

Convolutional Layer: Local Connections



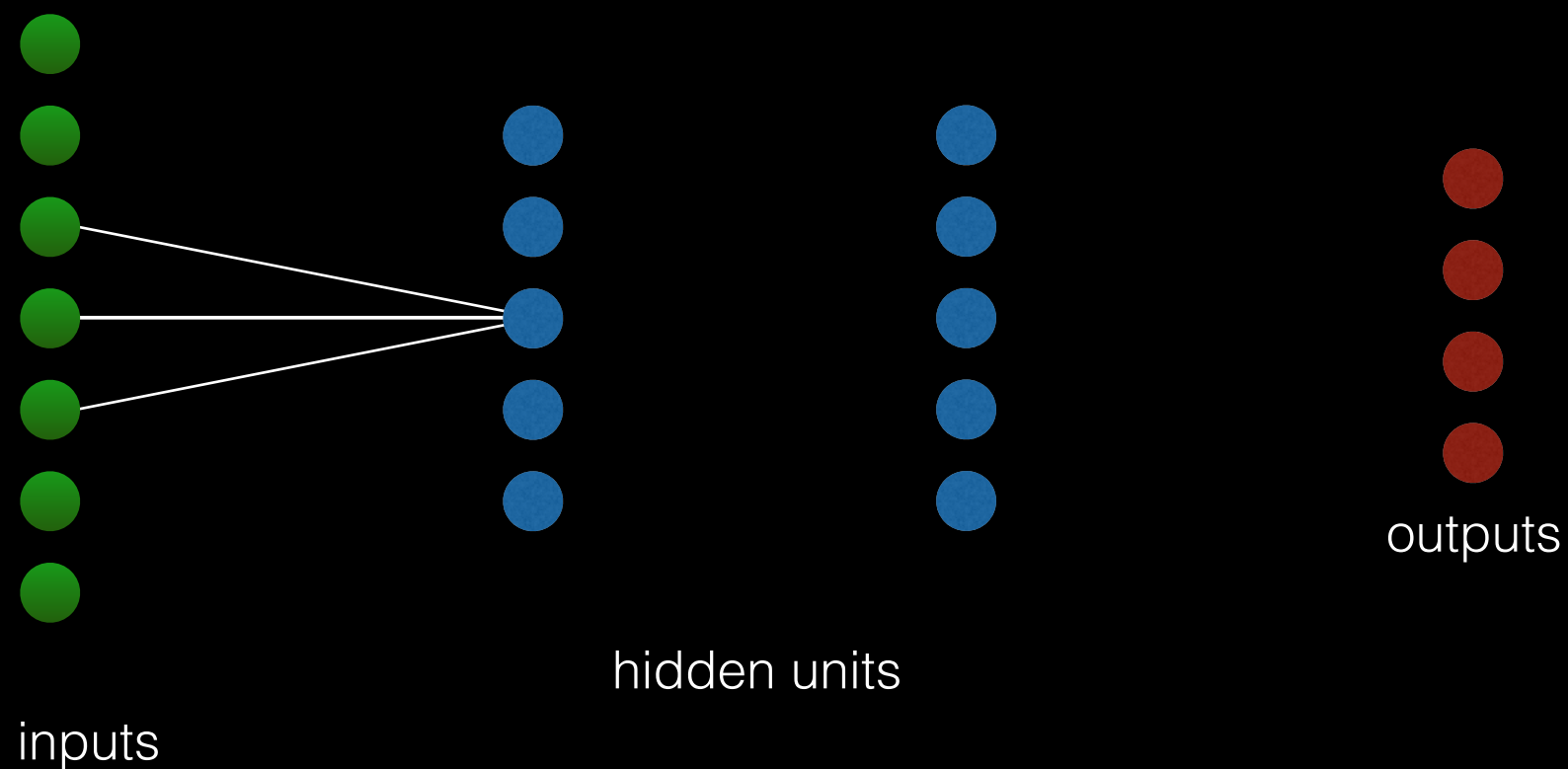
Here the connections are spatially local and governed by the kernel size.

Convolutional Layer: Local Connections



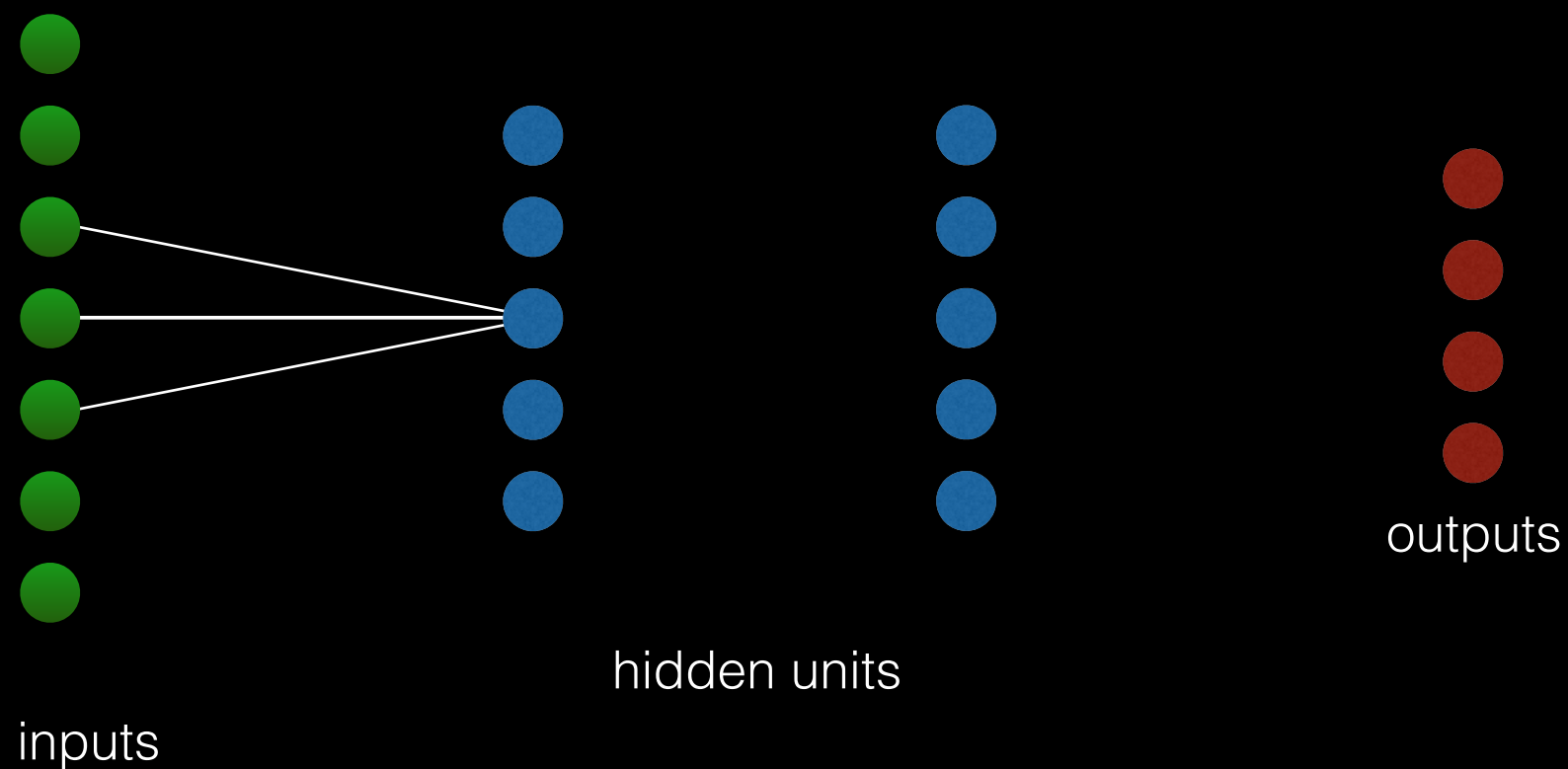
Here the connections are spatially local and governed by the kernel size.

Convolutional Layer: Local Connections



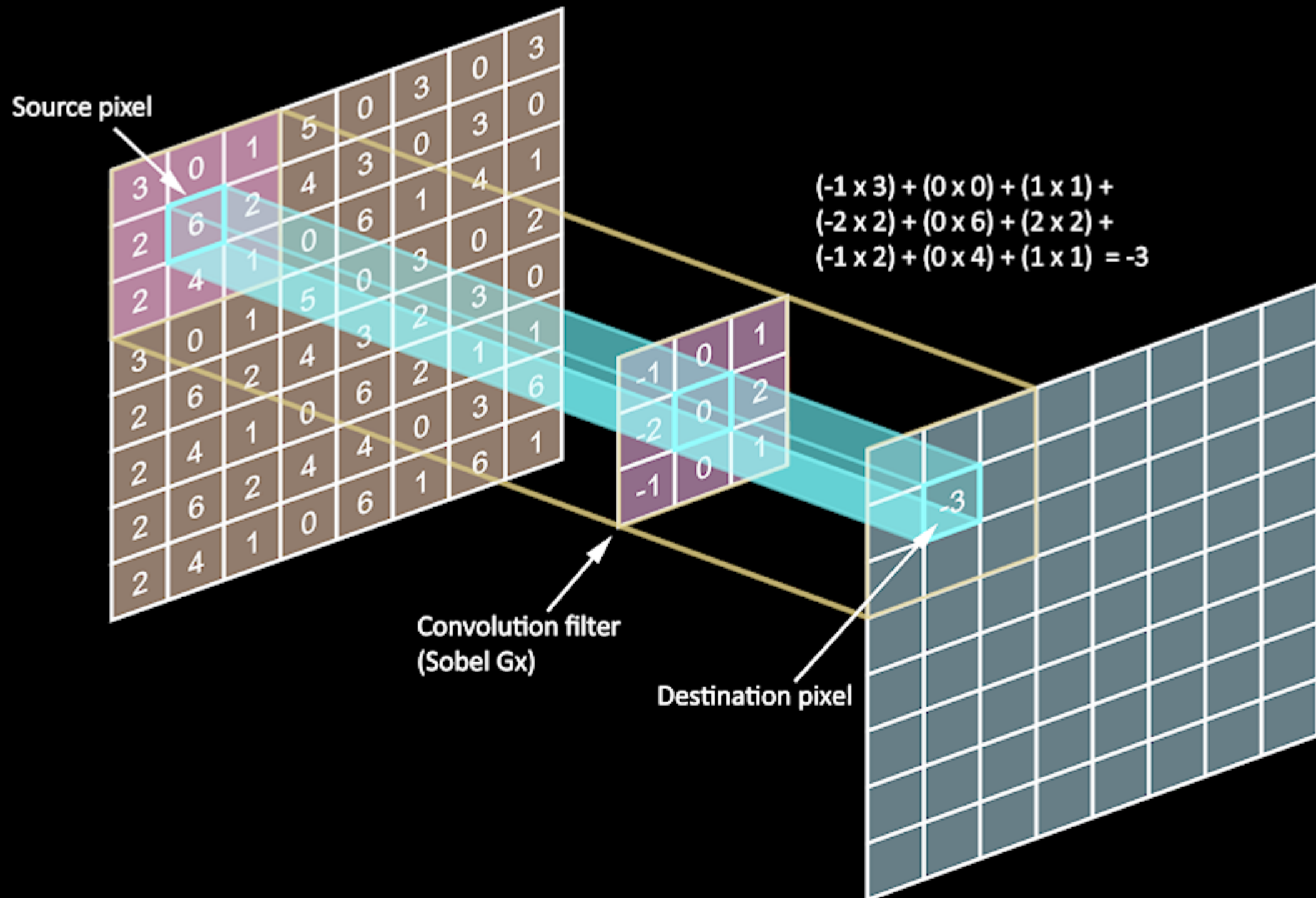
Here the connections are spatially local and governed by the kernel size.

Convolutional Layer: Shared Weights

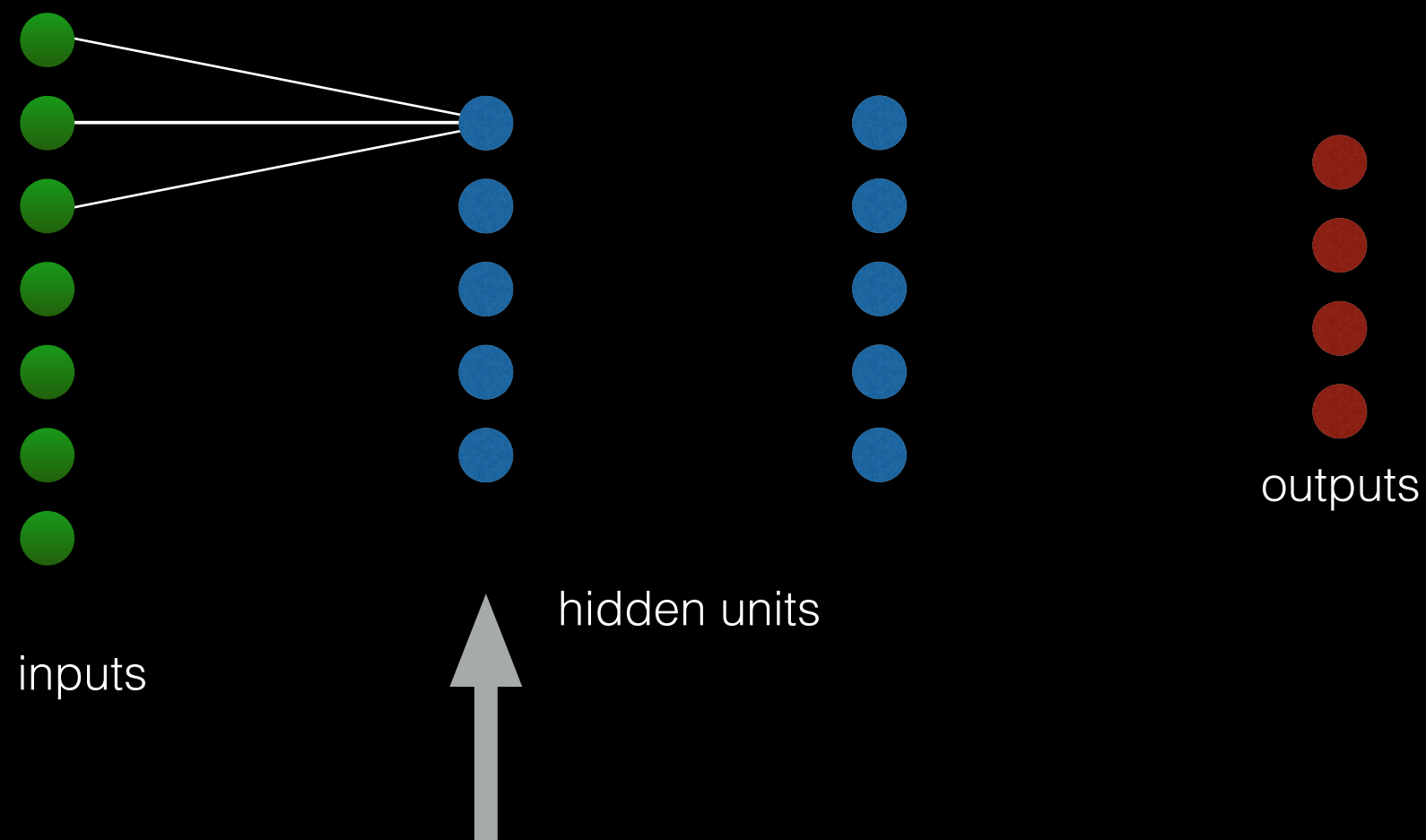


The weights for the kernel are shared. They are the same for each position.

Convolution in 2D

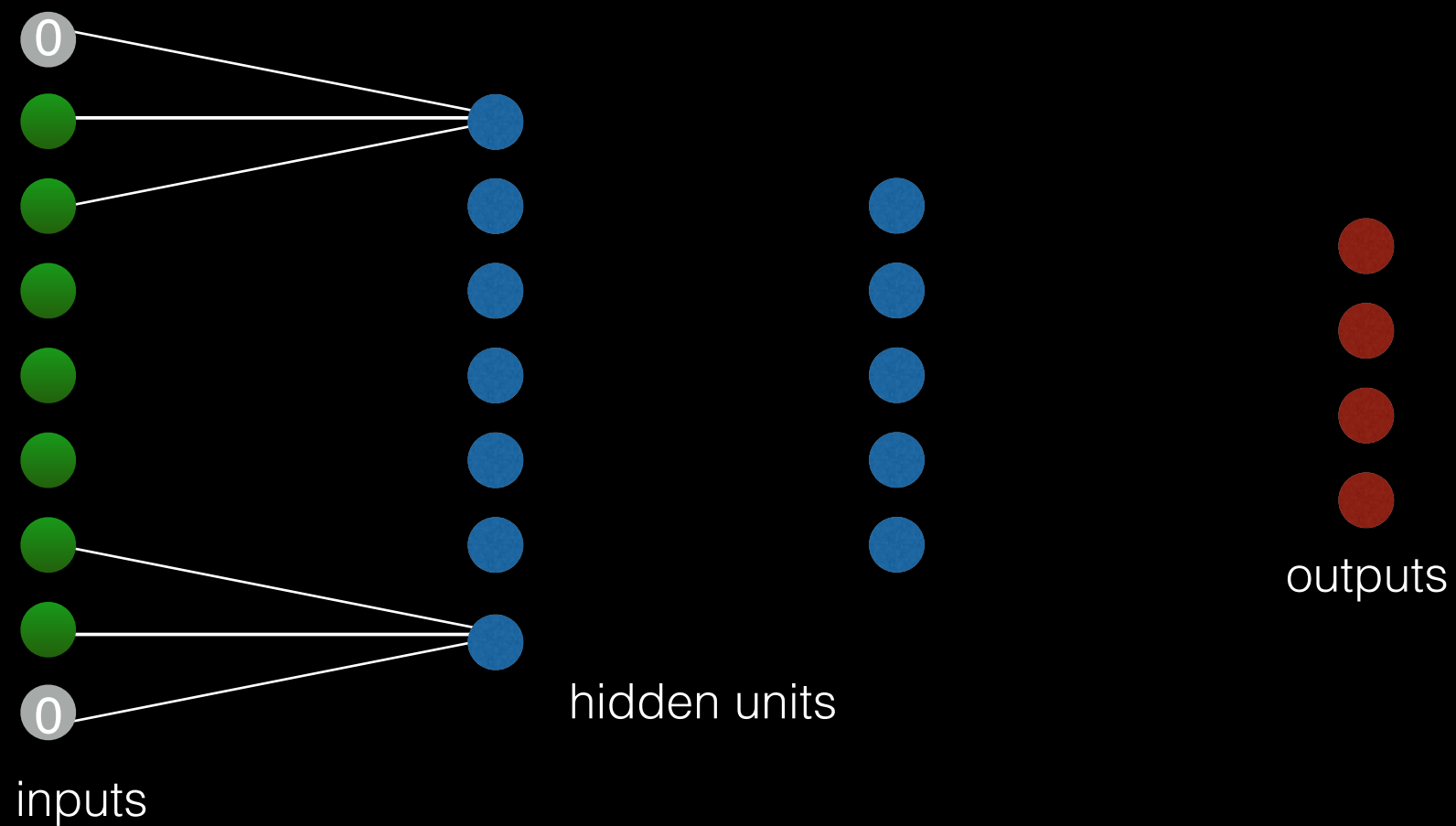


Convolutional Layer: No Padding



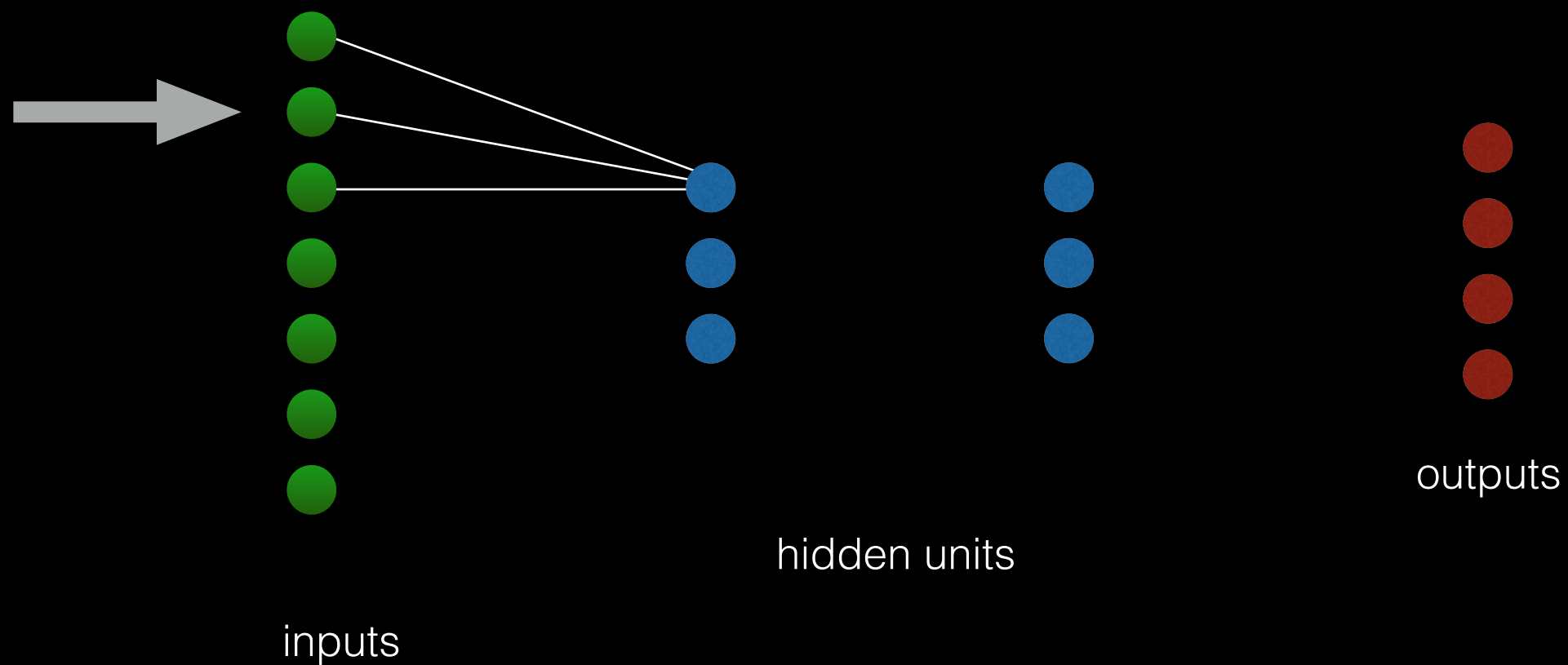
If we don't pad the inputs then the output dim = input dim - kernel size + 1.

Convolutional Layer: Zero Padding



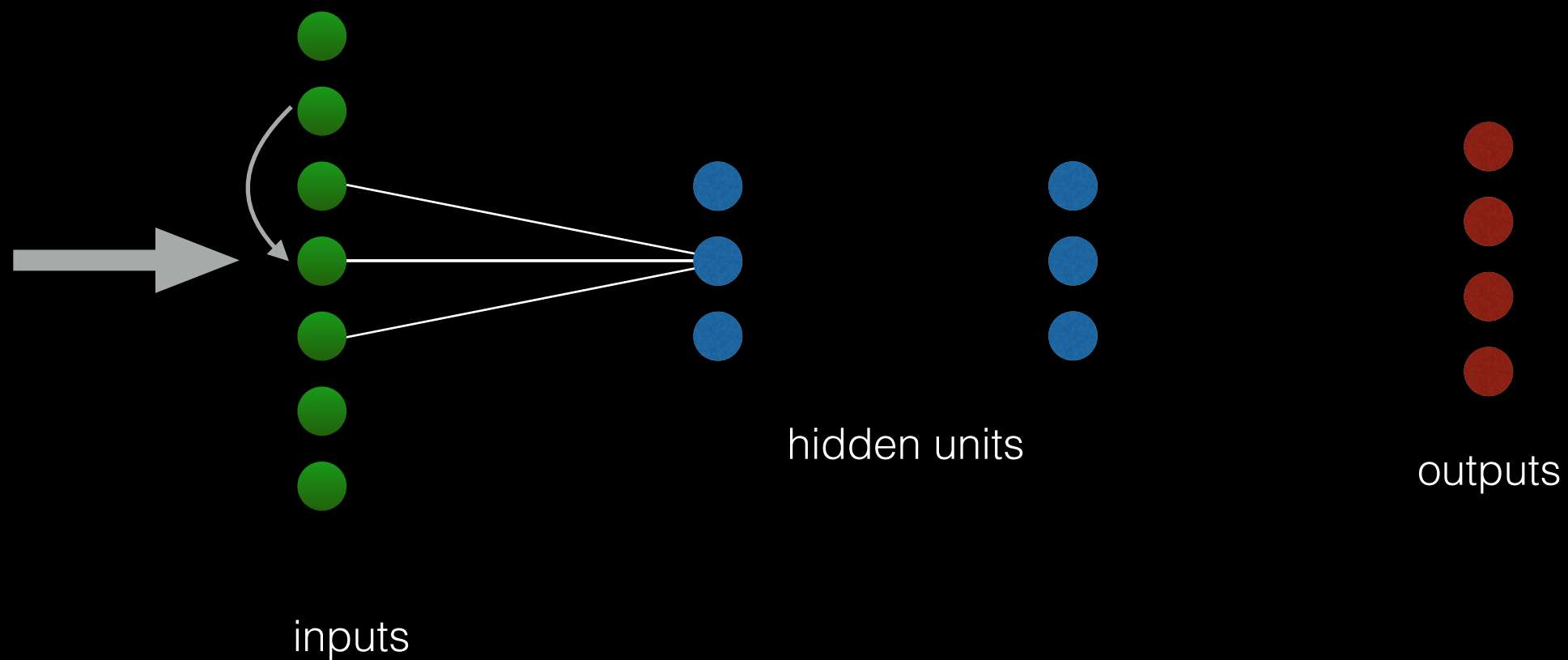
But we can pad the input with zeros to control output size.

Convolutional Layer: Stride



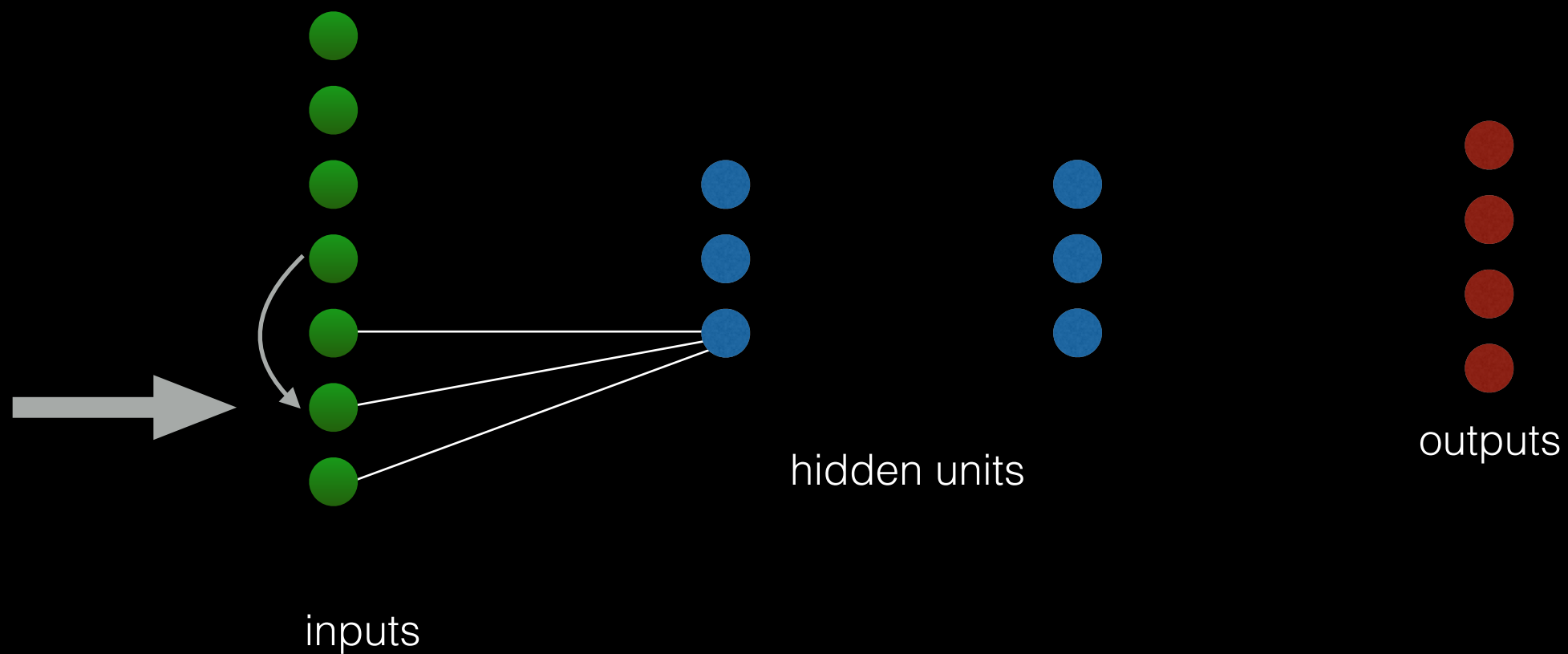
We can skip input pixels by choosing a *stride* > 1 .

Convolutional Layer: Stride



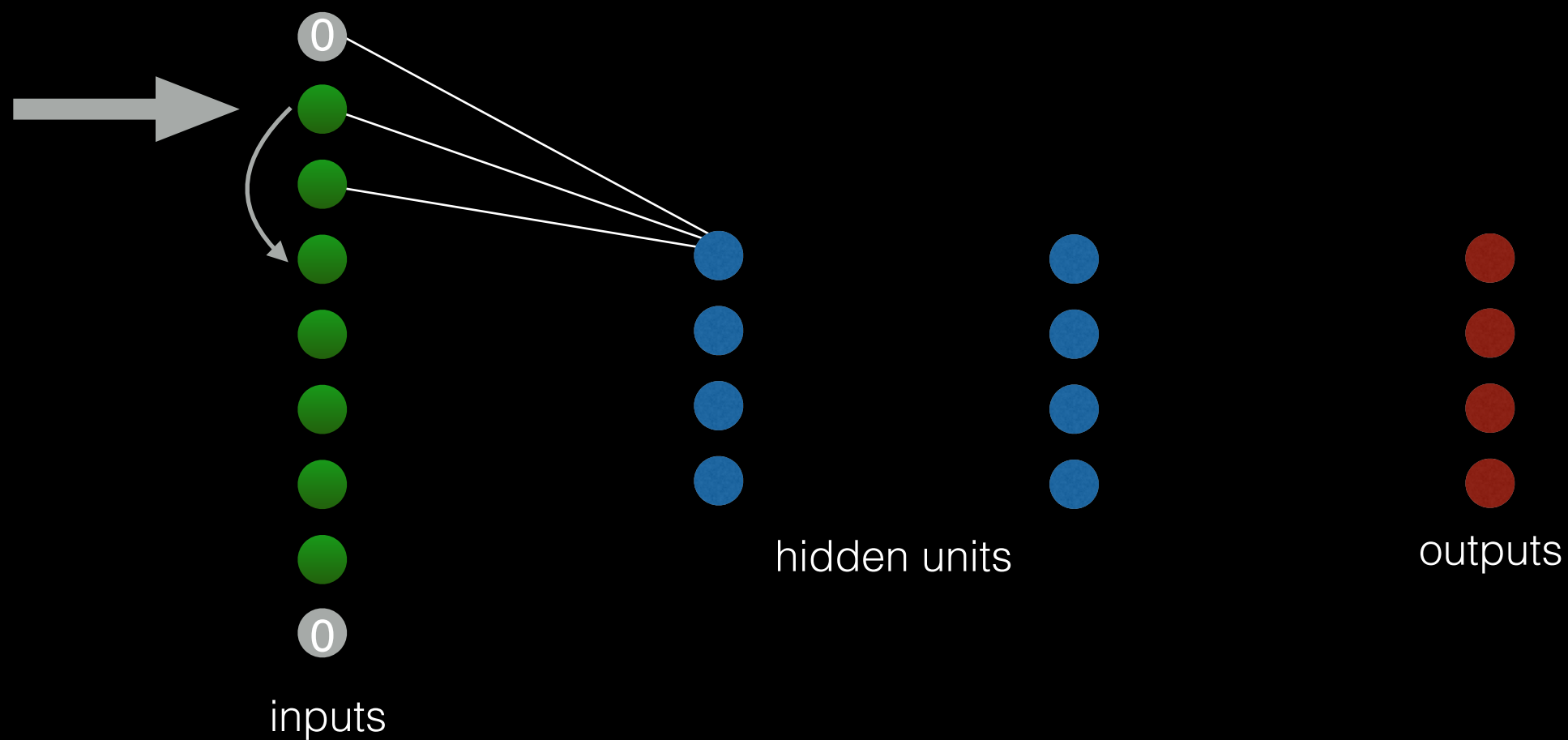
We can skip input pixels by choosing a *stride* > 1 .

Convolutional Layer: Stride



The output dim = (input dim - kernel size) / stride + 1.

Convolutional Layer: Padding + Stride



Output dimension = (input dimension - kernel size + 2 * padding) / stride + 1.

In general, when you design a network you have to choose these numbers so that everything comes together without dangling connections.

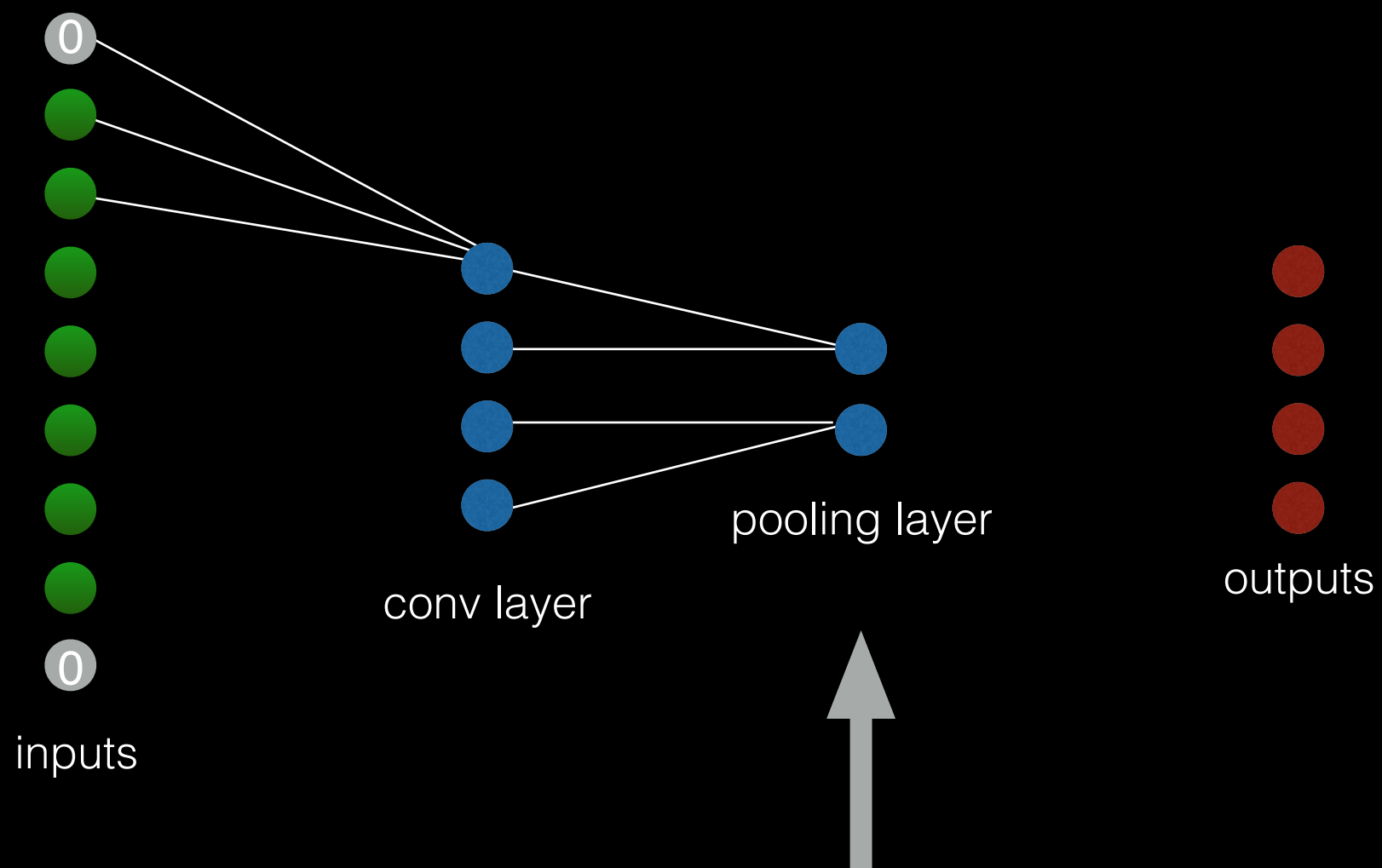
ReLU used with ConvNets

- Just like with our fully connected layers, for our convolutional layers we will follow the linear operation (convolution) with with a non-linear squashing function.
- Again the fcn to use for now is ReLU.
- But we are not done...there's one more thing!

Pooling

- We can spatially pool the output from the ReLU to reduce the size of subsequent layers in our network.
- This reduces both computation and the number of parameters that need to be fit and helps prevent overfitting.
- The pooling operation is often the **max** value in the region, but it could be **average**, or **median**, etc.
- The pooling has a stride associated with it that determines the downsampling of the input.

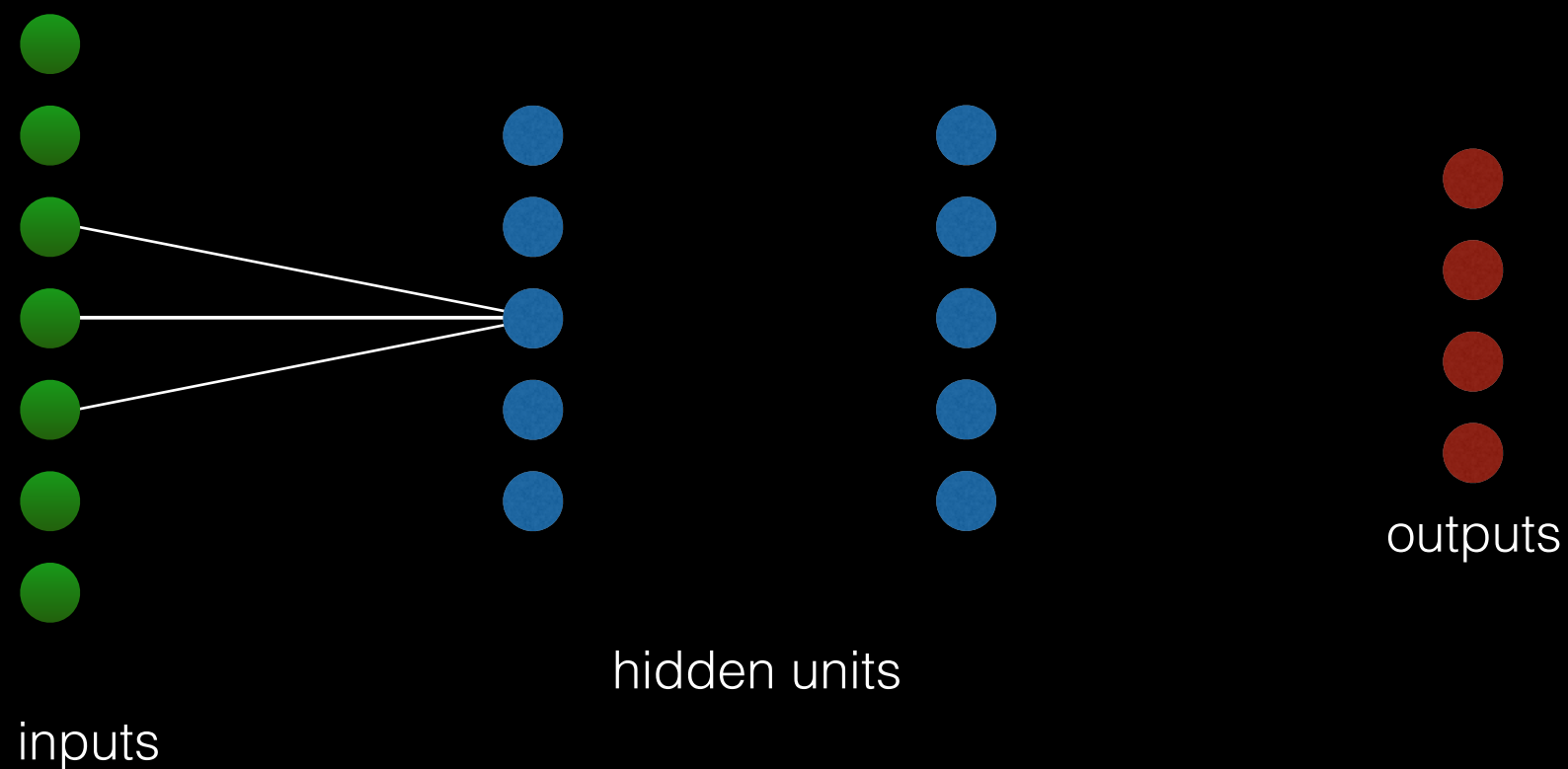
Pooling Layer



The pooling layer pools values in regions of the conv layer.

Oops. Just one more thing...Recall

Convolutional Layer: Shared Weights



The weights for the kernel are shared. They are the same for each position.

Each kernel finds just one type of feature.



*

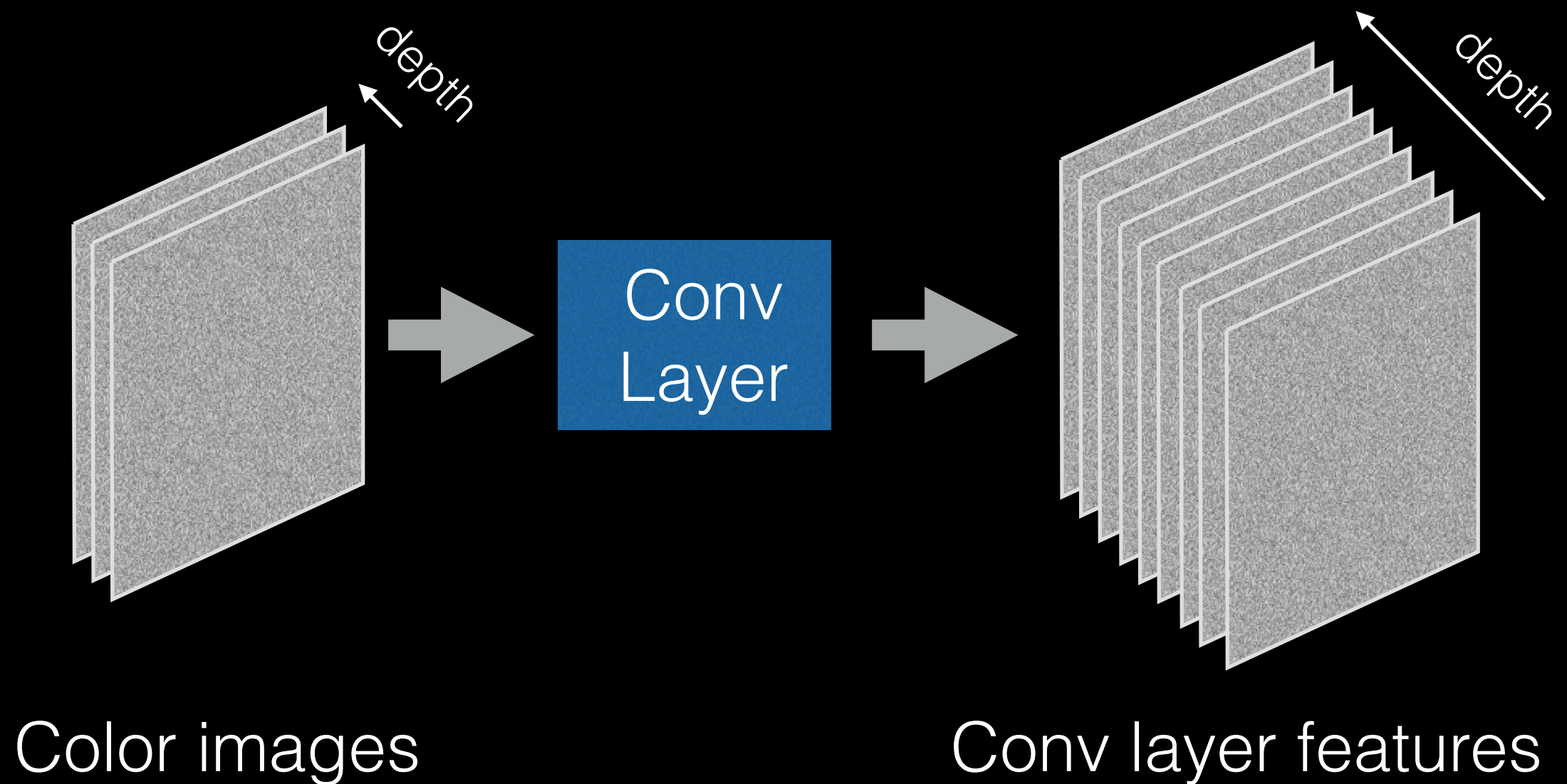
-1	-1	-1
-1	8	-1
-1	-1	-1

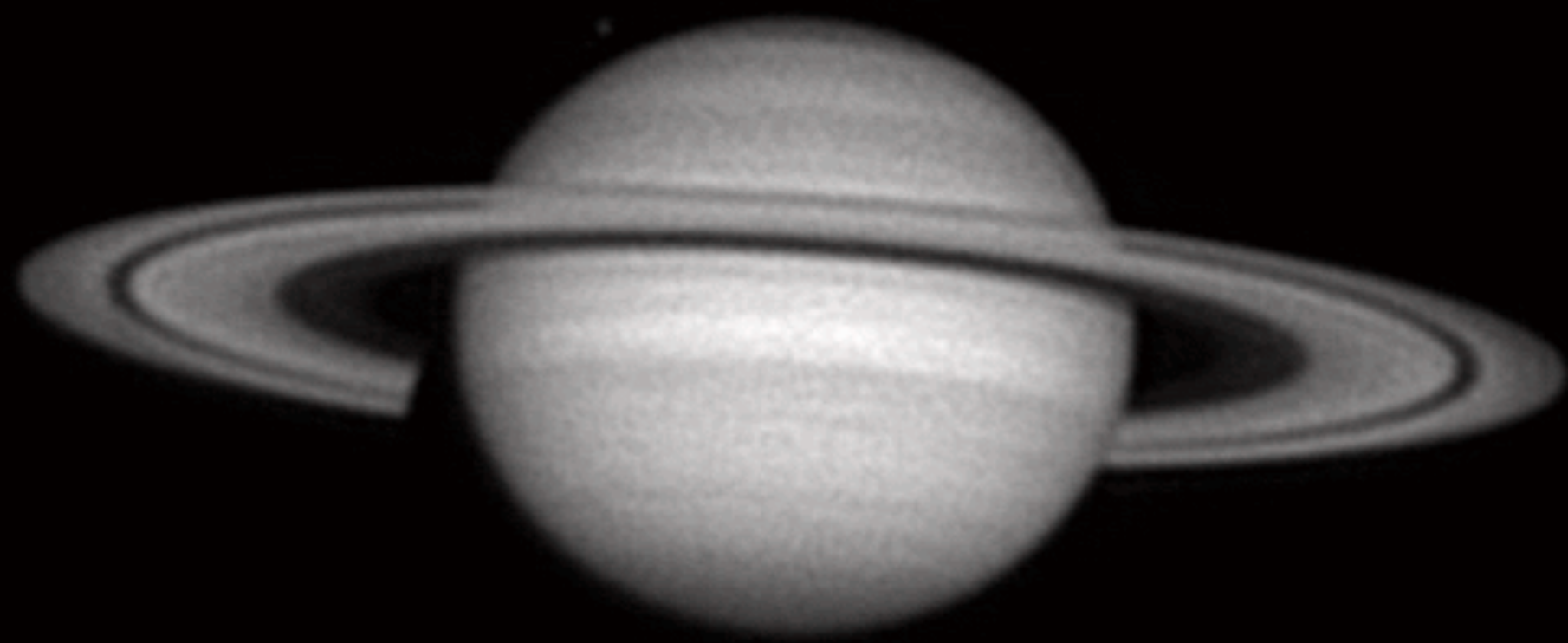
=



If a kernel shares weights then it can only extract one type of feature.

Why not allow for many kernels and many features!

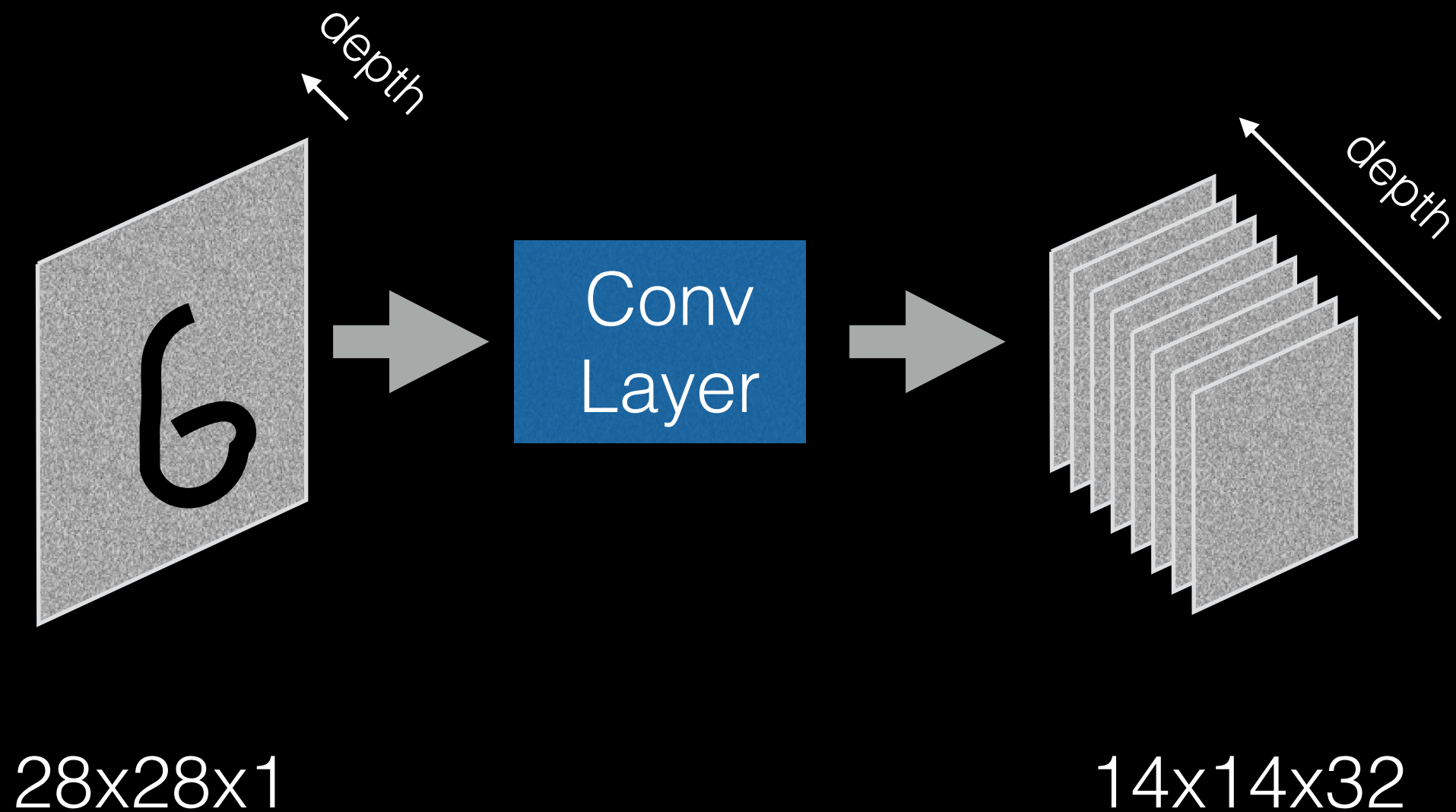




A Convolutional Net

- Let's assume we have 28x28 grayscale images as input to our conv net. So we will input 28x28x1 samples into the net.
- Let's fix our kernel size at 5x5 and, to make this simple, pad our images with zeros and use a stride = 1.
- Let's use max pooling on the output, with a 2x2 pooling region and a stride of 2.
- Let's extract 32 features after the first layer.
- So the output from this layer will be 14x14x32.

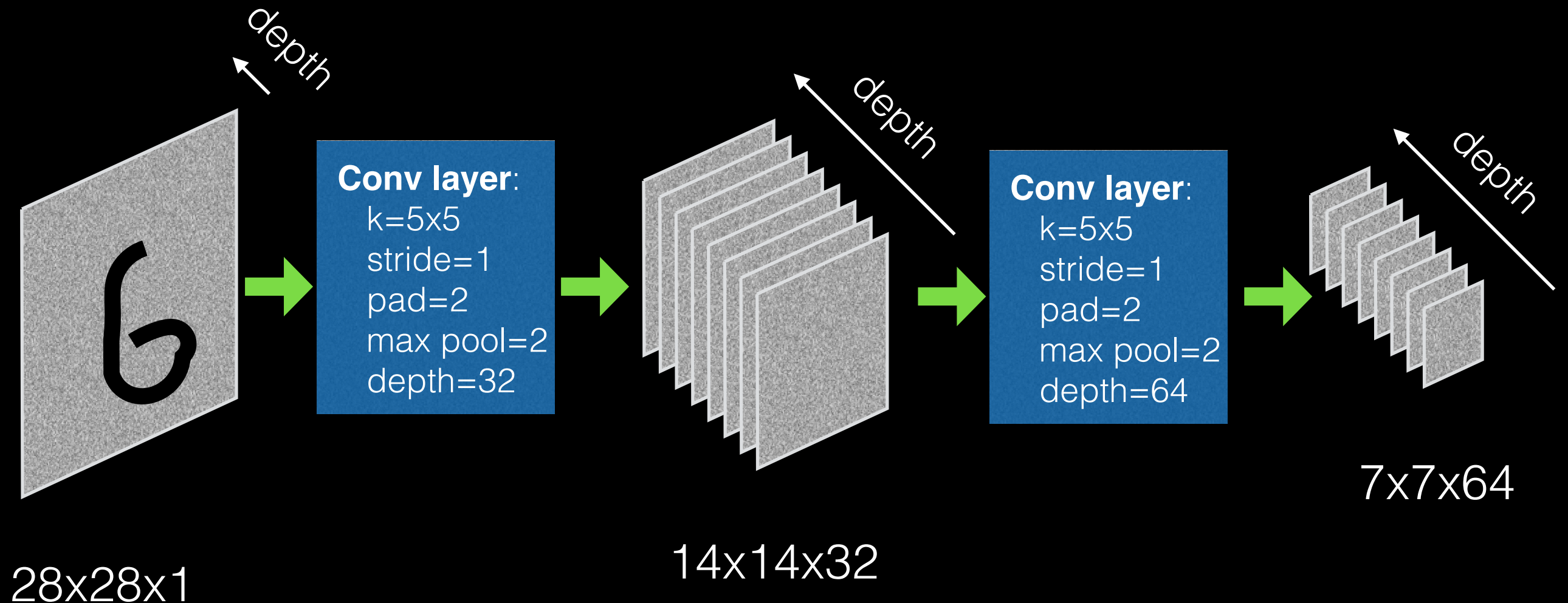
A Convolutional Net



A Convolutional Net

- Now let's make a second layer, also convolutional.
- Let's fix our kernel size at 5×5 , pad our images with zeros and use a stride = 1.
- Let's use max pooling on the output again, with a 2×2 pooling region and a stride of 2.
- Let's extract 64 features after the second layer.
- So the output from this layer will be $7 \times 7 \times 64$.

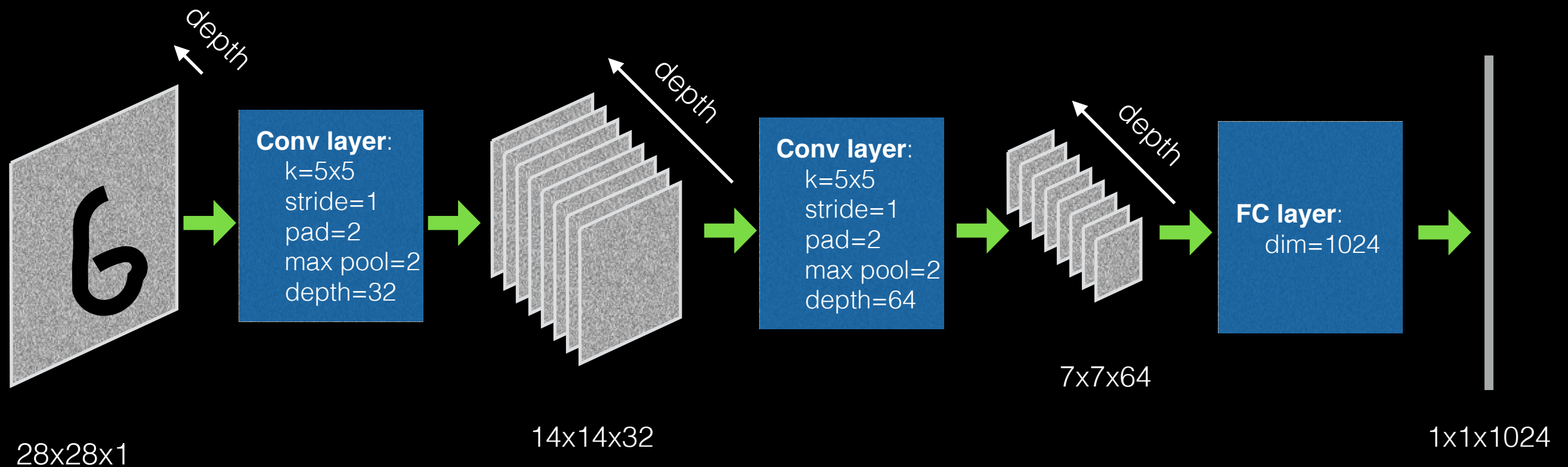
A Convolutional Net



A Convolutional Net

- Our third layer will be a fully connected layer mapping our convolutional features to a 1024 dimensional feature space.
- This layer is just like any of the hidden layers you have created before. It is a linear transformation followed by ReLU.
- So the output from this layer will be $1 \times 1 \times 1024$.

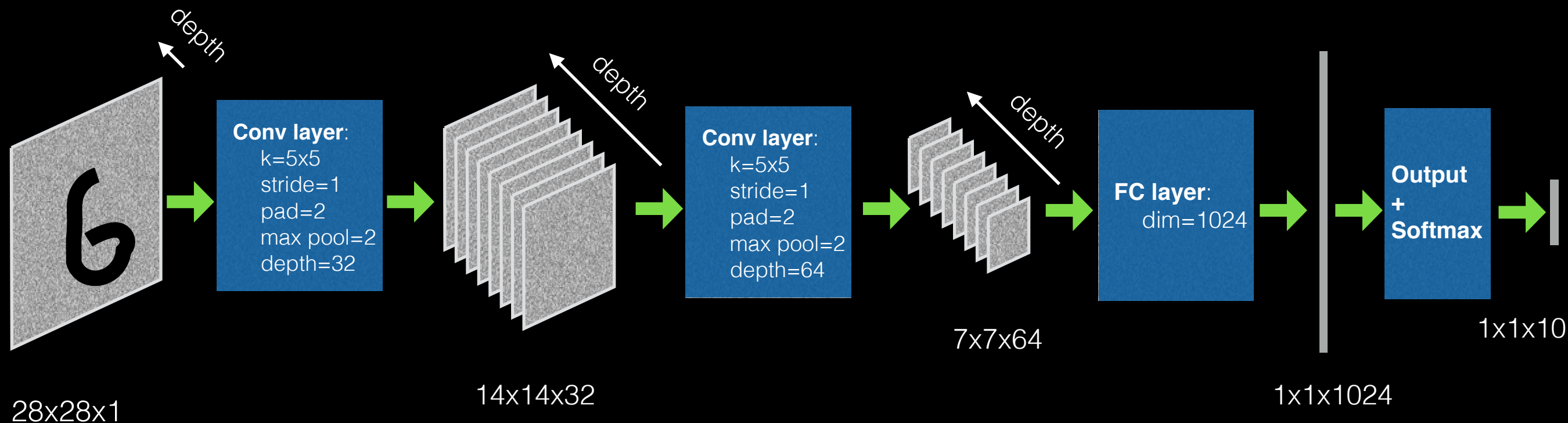
A Convolutional Net



A Convolutional Net

- Finally, will map this feature space to a 10 class output space and use a softmax with a MLE/cross entropy loss function.
- And...we're done!

A Convolutional Net



$$\text{Parameters} = (5 \times 5 \times 1 \times 32 + 32) + (5 \times 5 \times 32 \times 64 + 64) + (7 \times 7 \times 64 \times 1024 + 1024) + (1024 \times 10 + 10)$$

