JOHNS HOPKINS
WHITING SCHOOL of ENGINEERING

EN.601.482/682 Deep Learning

Convolutional Neural Networks

Mathias Unberath, PhD
Assistant Professor
Dept of Computer Science
Johns Hopkins University

Reminder

- Neural networks began as computational models of the brain / cognition

- Connectionism
  → Neurons connect to neurons
  → Knowledge is encoded in these connections

- Today’s neural networks are connectionist machines

Reminder

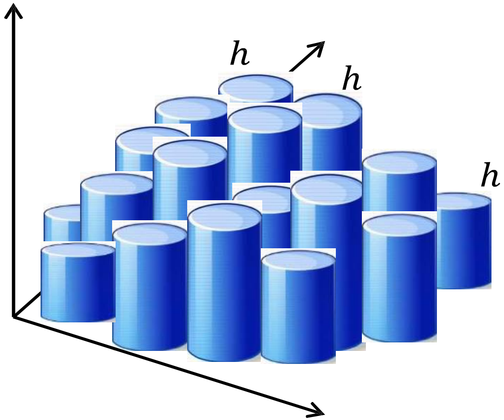
- McCullough and Pitts model
  - Excitatory and inhibitory synapses
  - No learning rule
- Hebbian Learning
  - Neurons that fire together, wire together!
  - Unstable!
- Rosenblatt's perceptron
  - Convergent learning rule for linearly separable problems
  - Single perceptrons are limited (Minsky and Papert)
- Multi-layer perceptrons model arbitrary Boolean functions

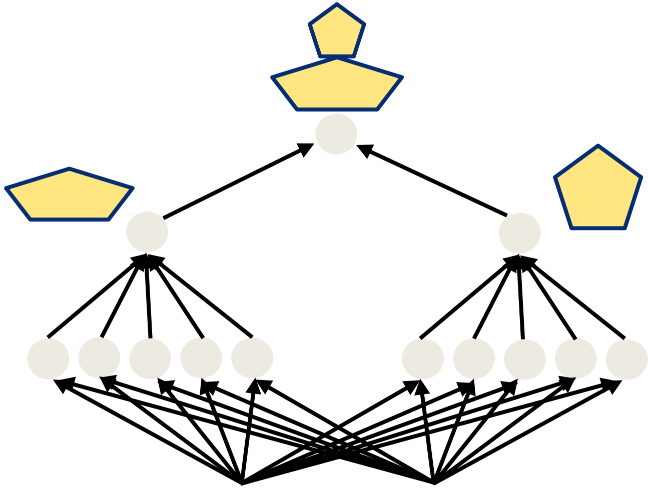
Reminder

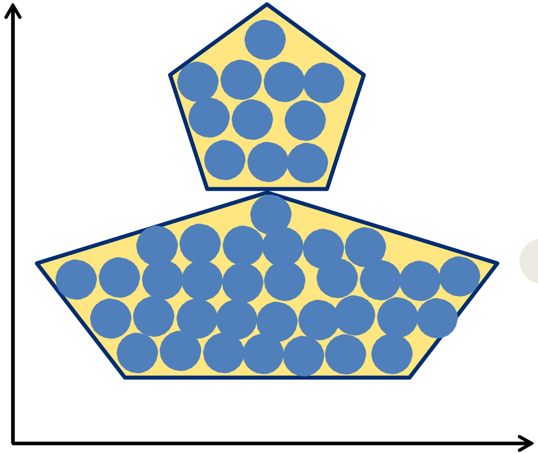
Multi-layer perceptrons are

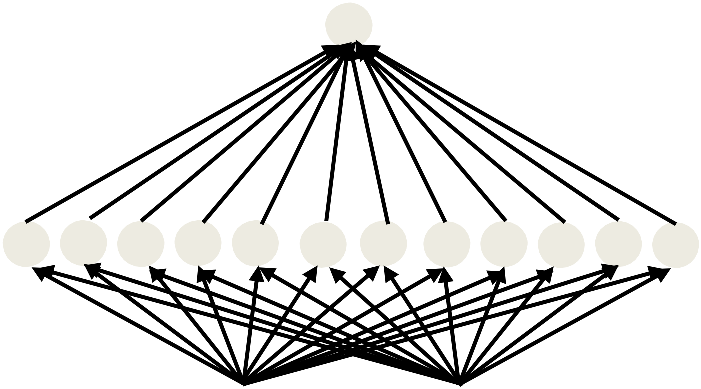
- Universal Boolean functions
- Universal classifiers
- Universal approximators

… but may require infinitely many neurons



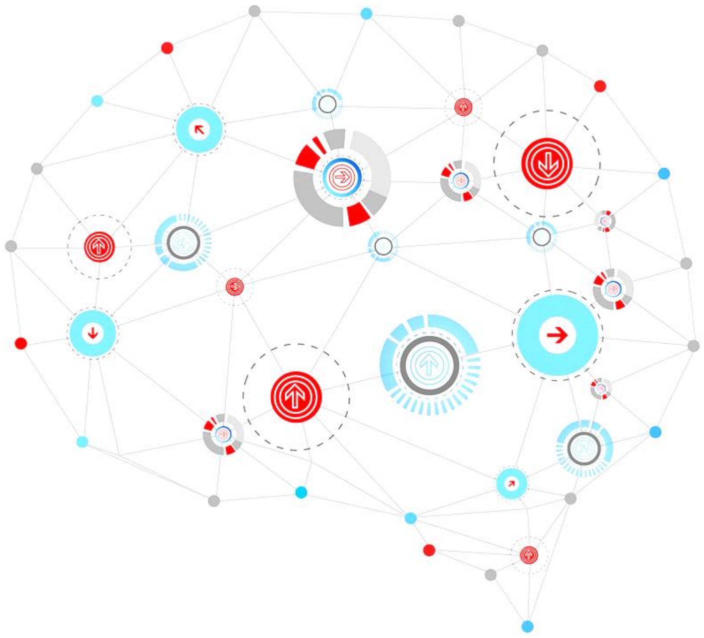




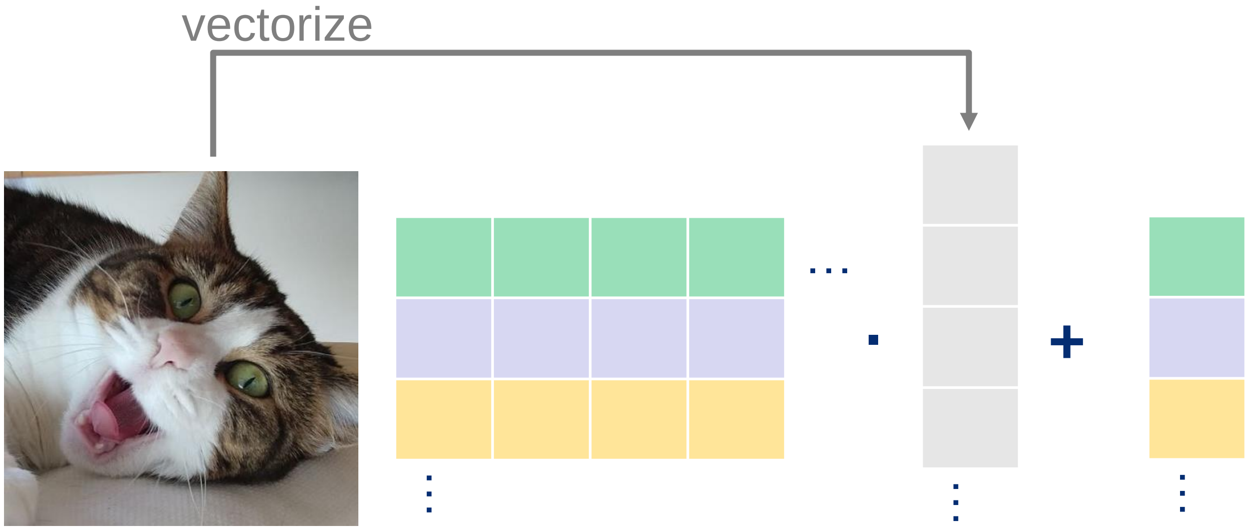


Today's Lecture

Convolutional Neural Networks



The Image Classification Problem: Revisited

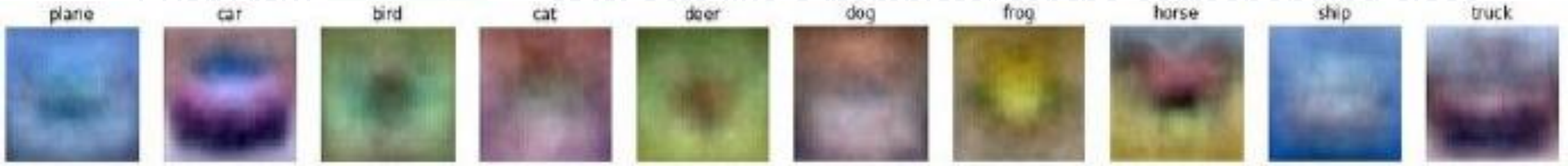


Input image

- Every row ... of **W** acts like a “template” for the corresponding class

The Image Classification Problem: Revisited

Visual Interpretation

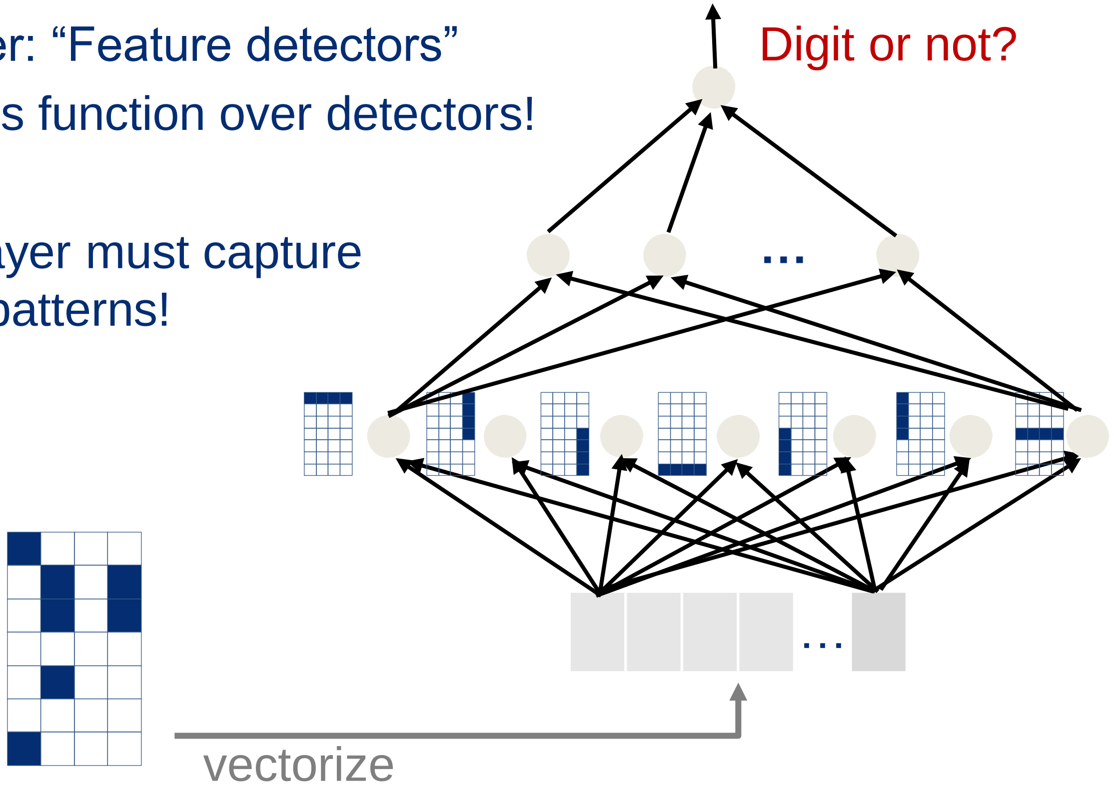


The Image Classification Problem: Revisited

Input layer: “Feature detectors”

Network is function over detectors!

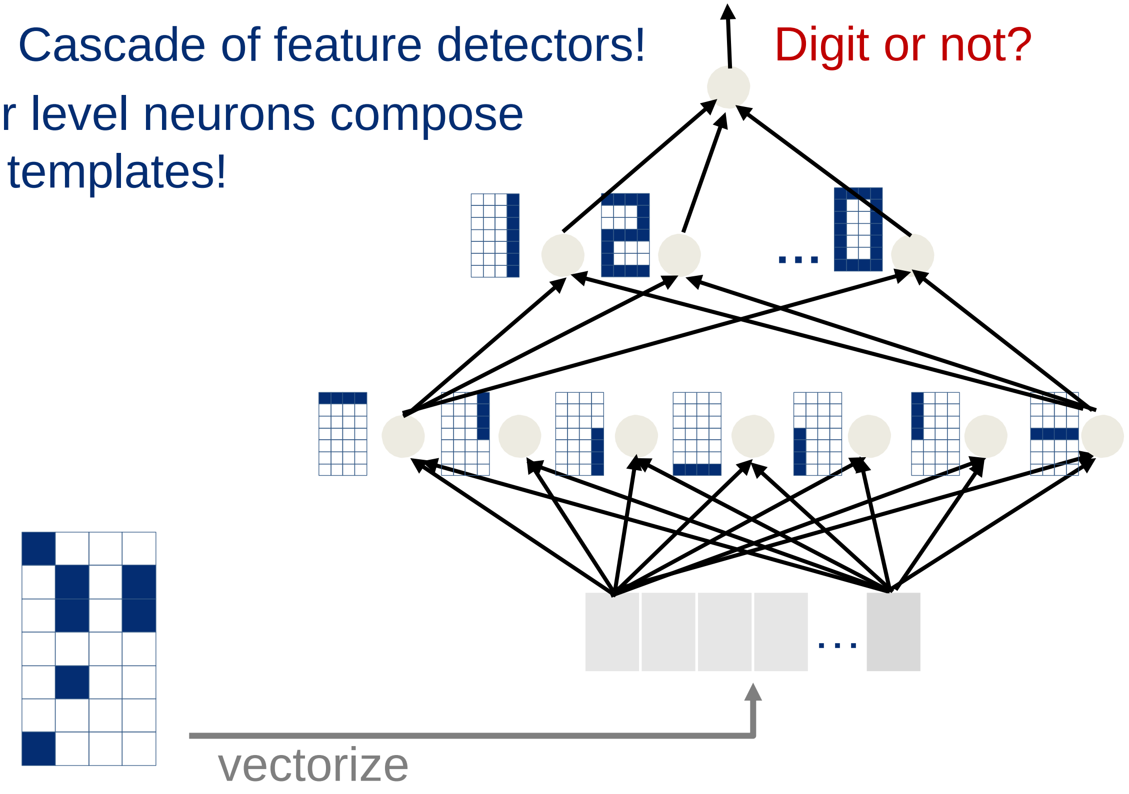
→ First layer must capture relevant patterns!



The Image Classification Problem: Revisited

Network: Cascade of feature detectors!

→ Higher level neurons compose complex templates!

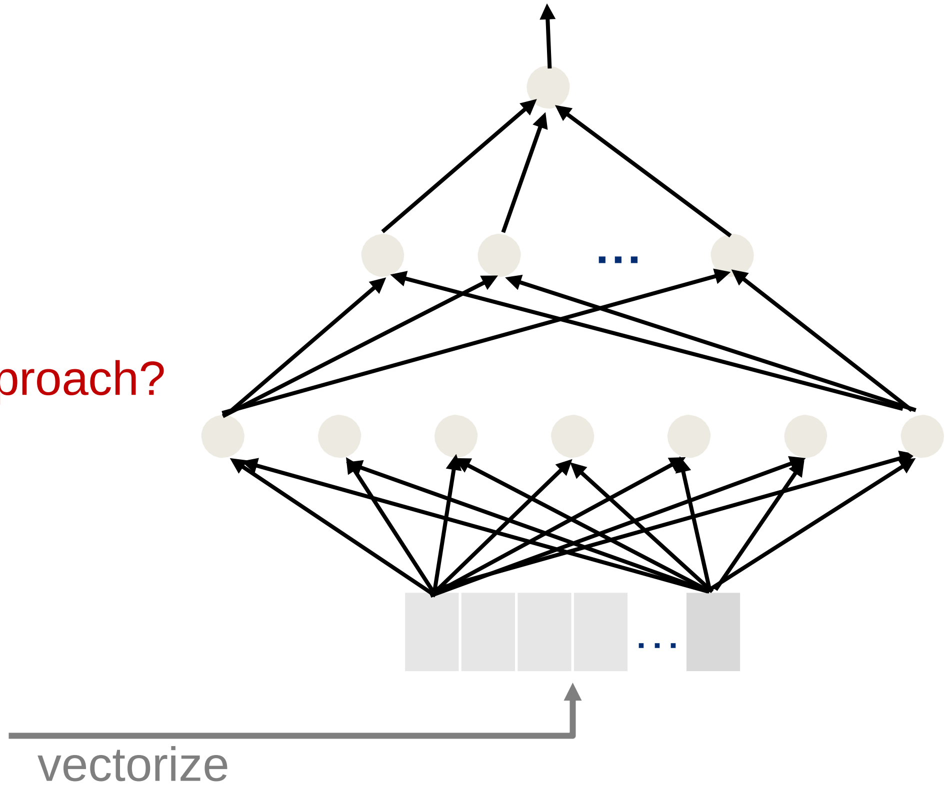


The Image Classification Problem: Revisited

Is there a deer in this image?

Trivial solution:
→ Train a MLP for the entire image

Q: What is the problem with this approach?

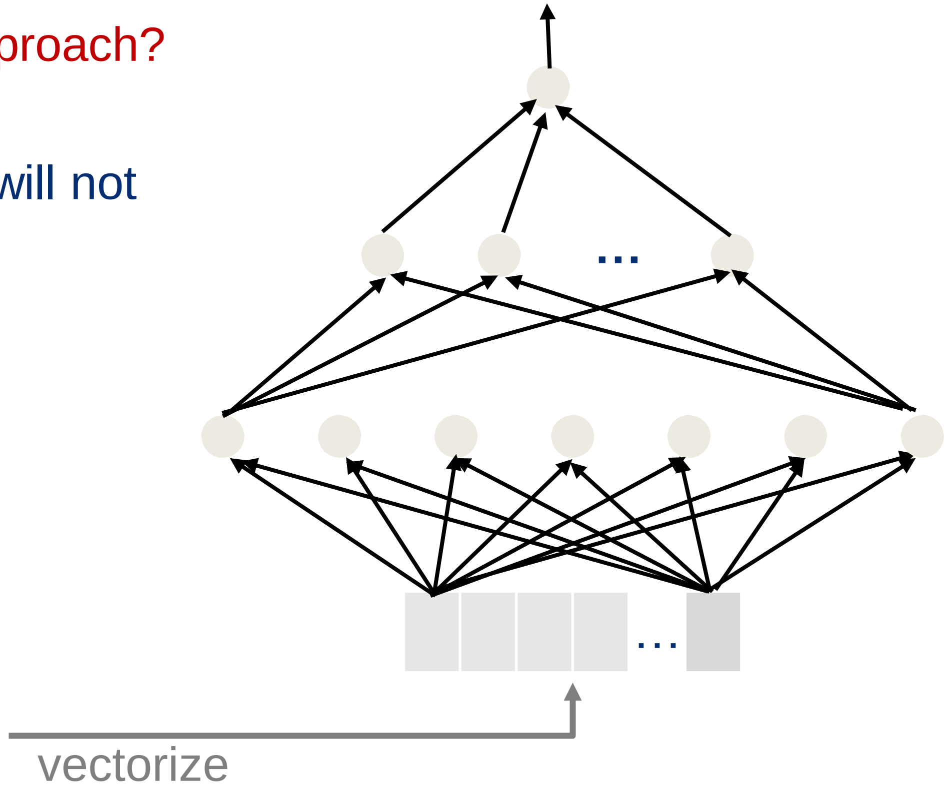


The Image Classification Problem: Revisited

Q: What is the problem with this approach?

An MLP trained on the right image will not find a deer in the left one (unless trained on both).

→ Large amount of training data!



The Image Classification Problem: Revisited

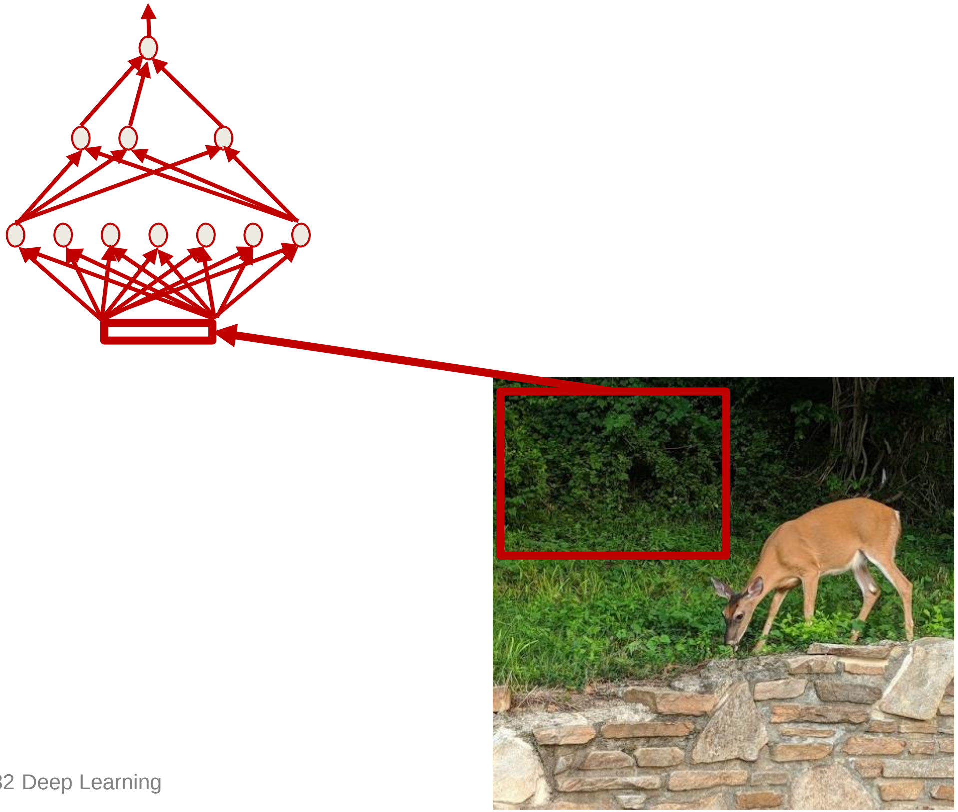
What we need / want:

→ Simple solution that works on both!

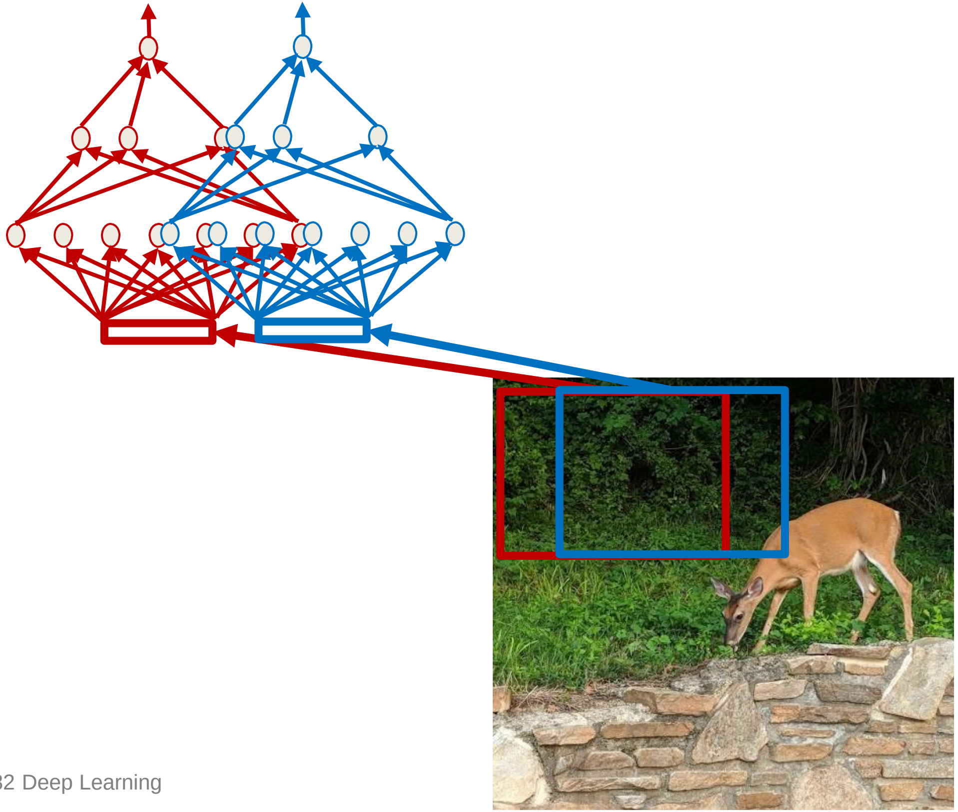
Conventional MLPs are sensitive to location, but often the **location** of a pattern **is not important**!

→ Shift invariance!

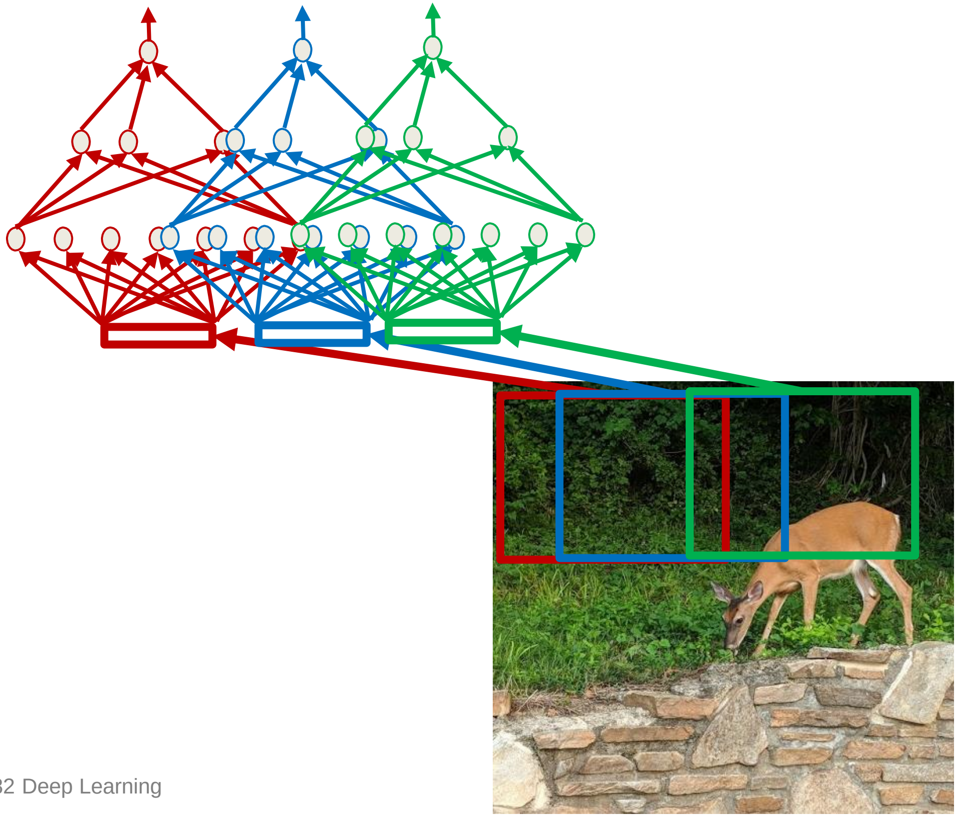
Shift Invariance: Scanning for Patterns



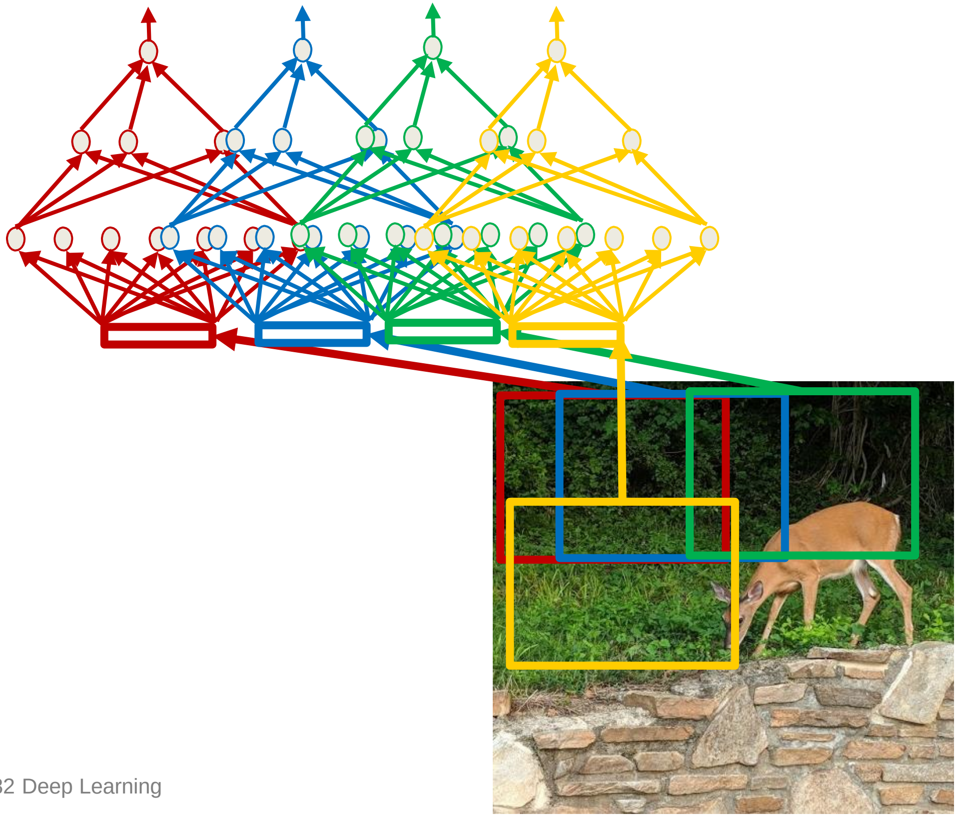
Shift Invariance: Scanning for Patterns



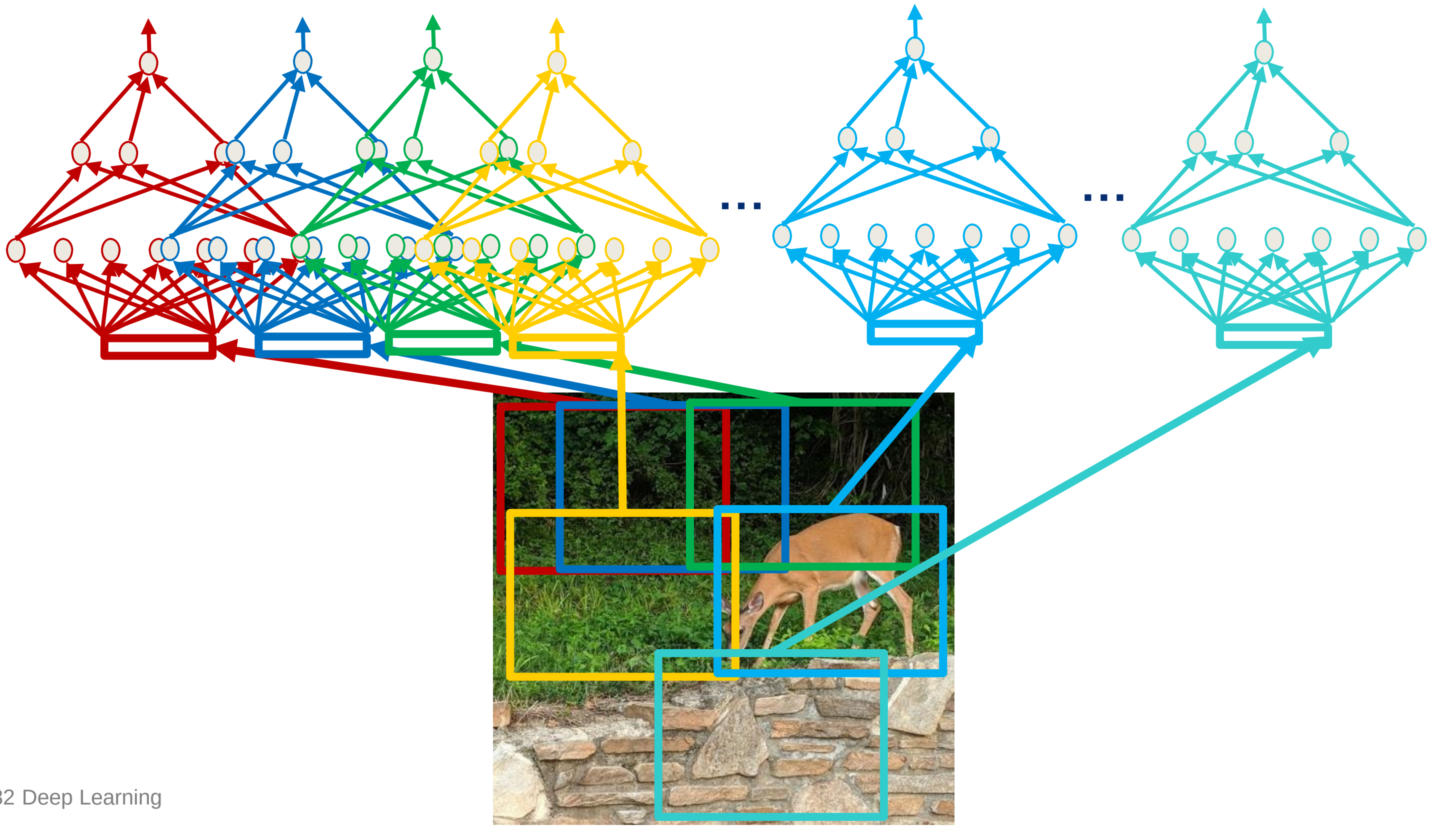
Shift Invariance: Scanning for Patterns



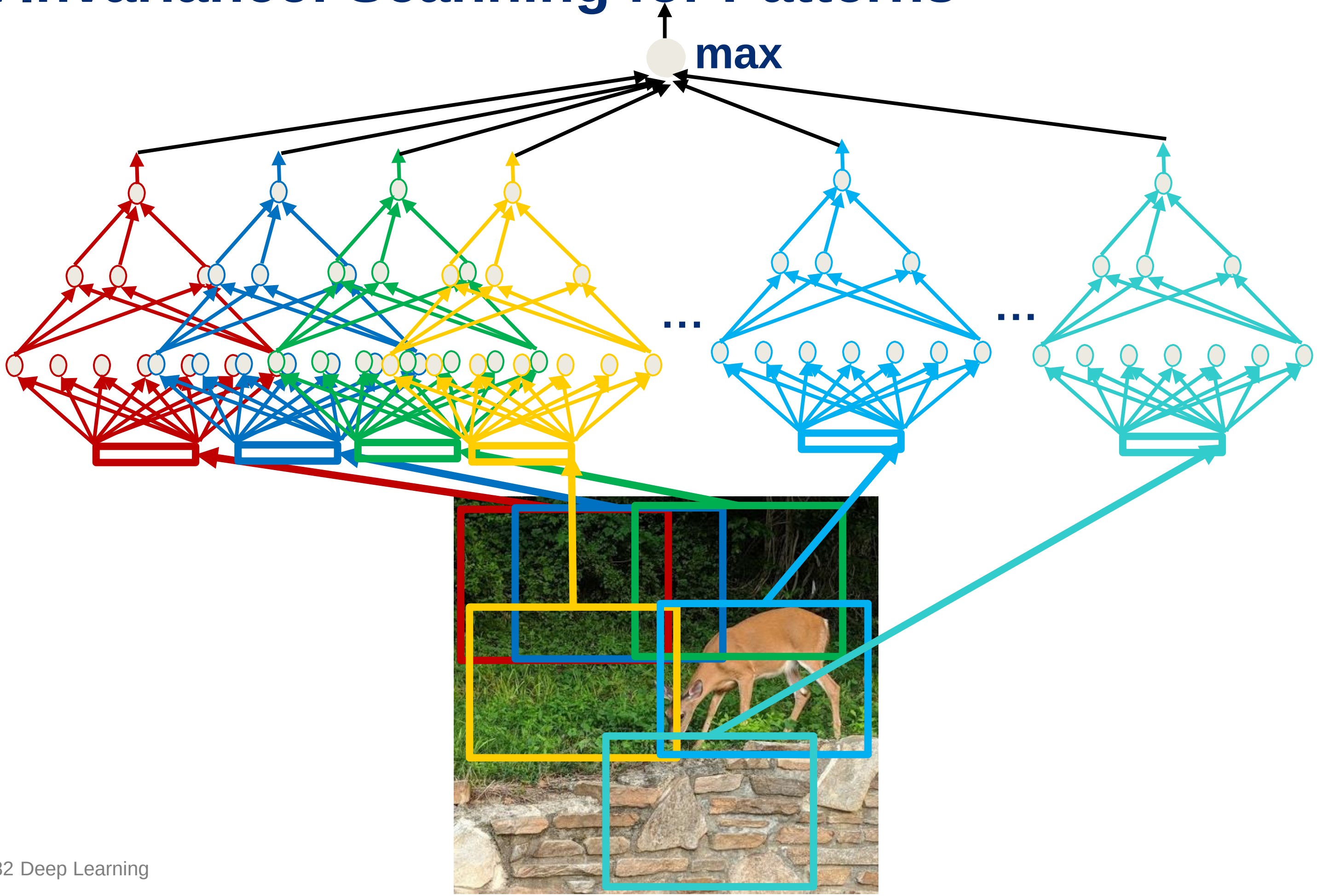
Shift Invariance: Scanning for Patterns



Shift Invariance: Scanning for Patterns



Shift Invariance: Scanning for Patterns



Summary

- Is this particular pattern in this image?
  - Maximum of all outputs in sub-windows (→ Boolean OR)
  - Or can be softmax
  - Or even another MLP

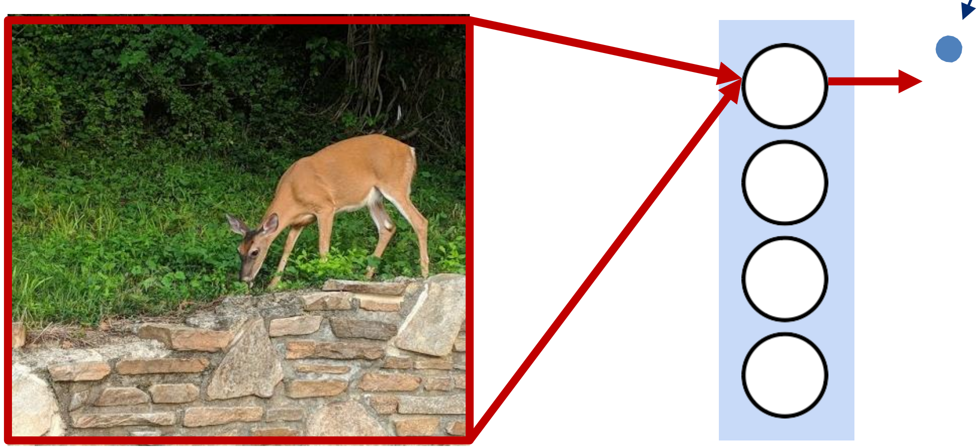
- Important observation: Entire operation can be viewed as one network!
  - Many subnetworks: One per window!
  - But, shift invariance: All subnetworks are identical

Convolutional Neural Networks

Scanning for Patterns

Scanning: A Closer Look

$h(\sum_i w_i x_i + b)$

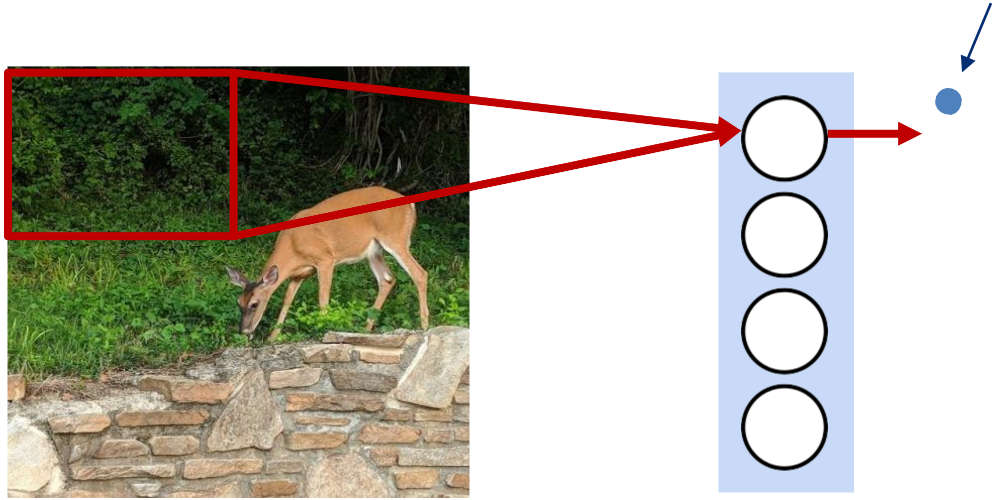


Previously

→ Evaluate perceptron on complete image

Scanning: A Closer Look

$$h(\sum_i w_i x_i + b)$$

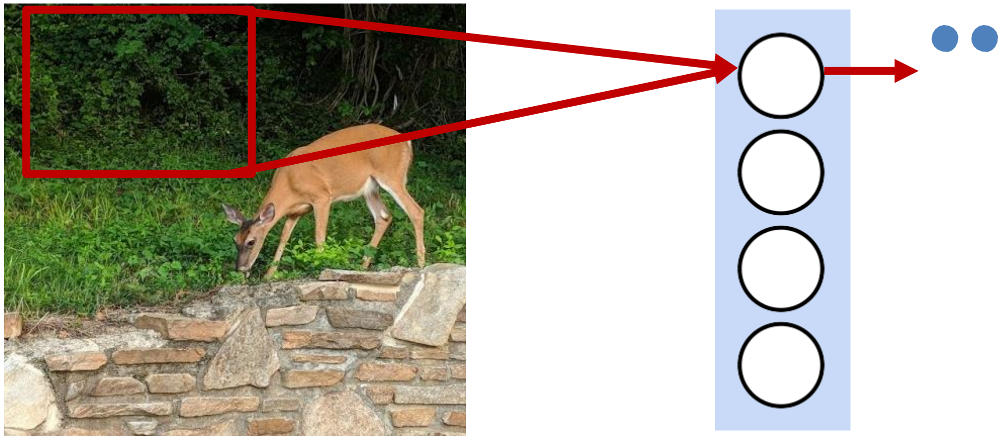


Now

→ Perceptron evaluates this region of the image

→ We can arrange these outputs in form of the original picture

Scanning: A Closer Look

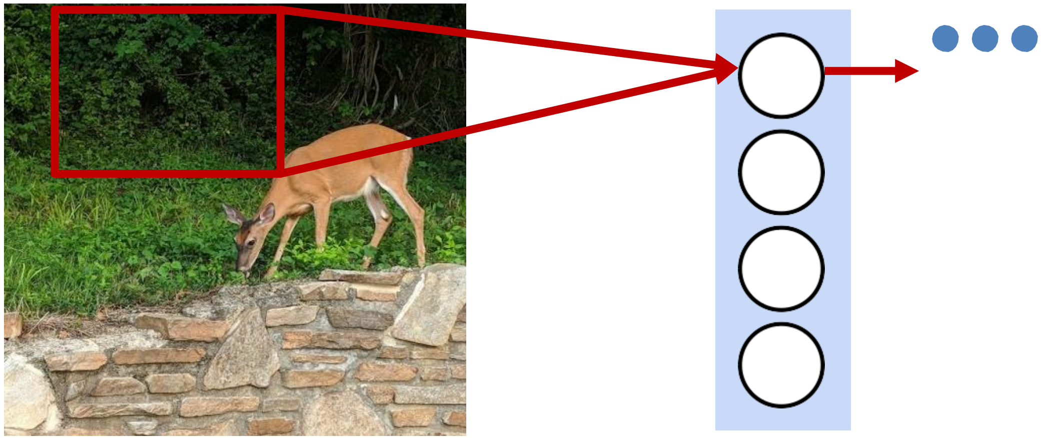


Now

→ Perceptron evaluates this region of the image

→ We can arrange these outputs in form of the original picture

Scanning: A Closer Look

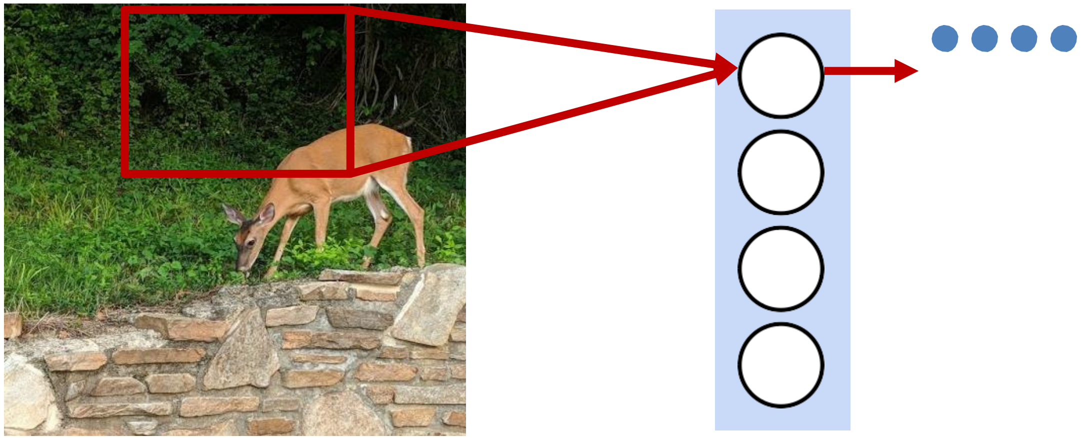


Now

→ Perceptron evaluates this region of the image

→ We can arrange these outputs in form of the original picture

Scanning: A Closer Look

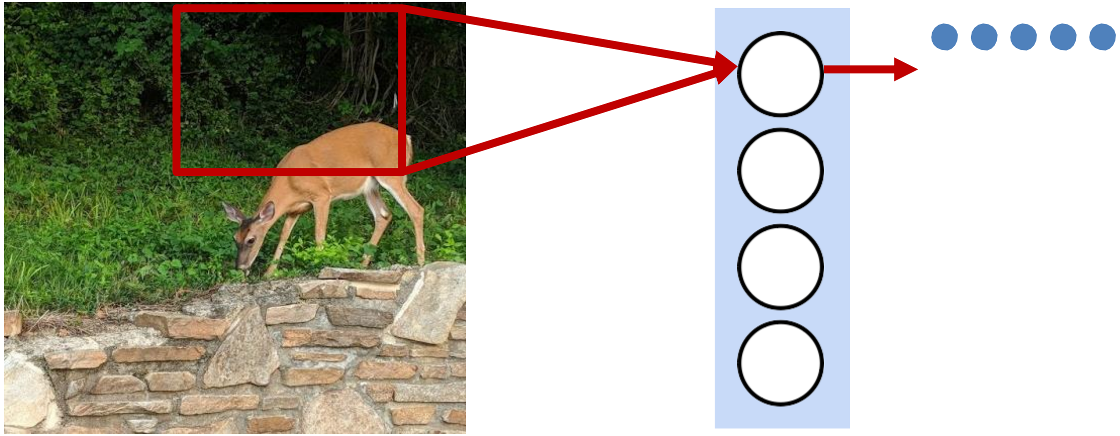


Now

→ Perceptron evaluates this region of the image

→ We can arrange these outputs in form of the original picture

Scanning: A Closer Look

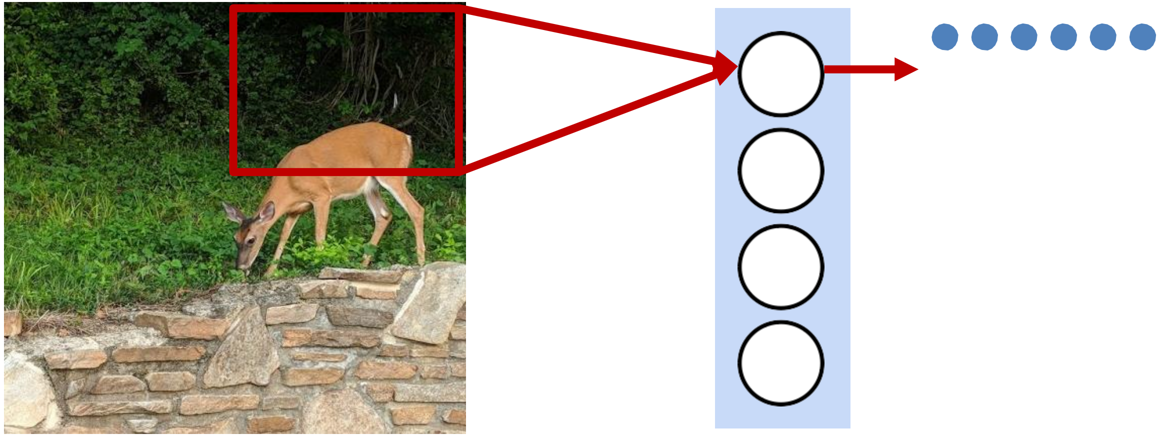


Now

→ Perceptron evaluates this region of the image

→ We can arrange these outputs in form of the original picture

Scanning: A Closer Look

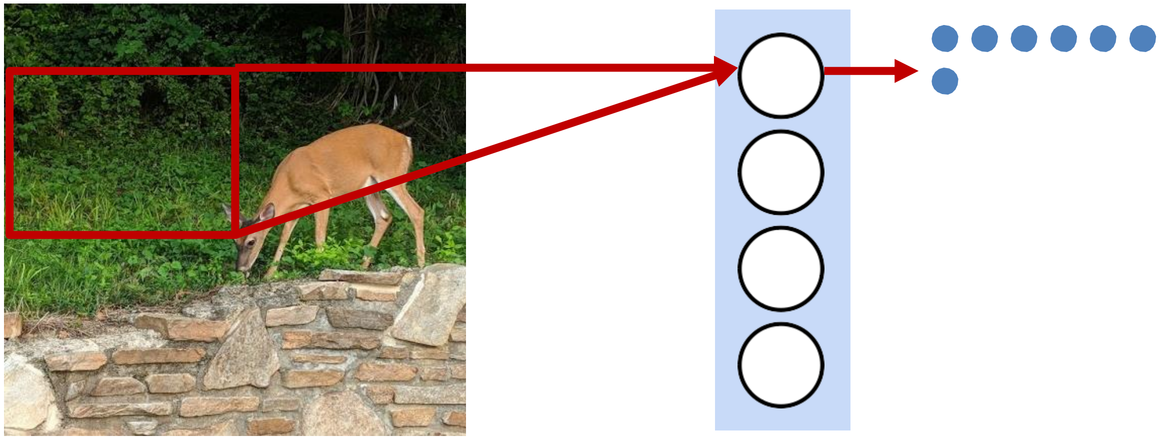


Now

→ Perceptron evaluates this region of the image

→ We can arrange these outputs in form of the original picture

Scanning: A Closer Look

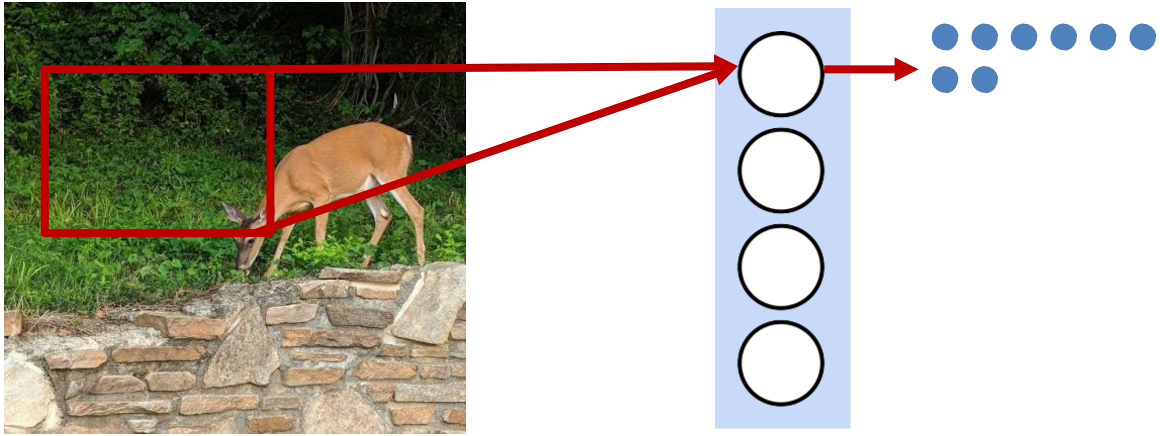


Now

→ Perceptron evaluates this region of the image

→ We can arrange these outputs in form of the original picture

Scanning: A Closer Look

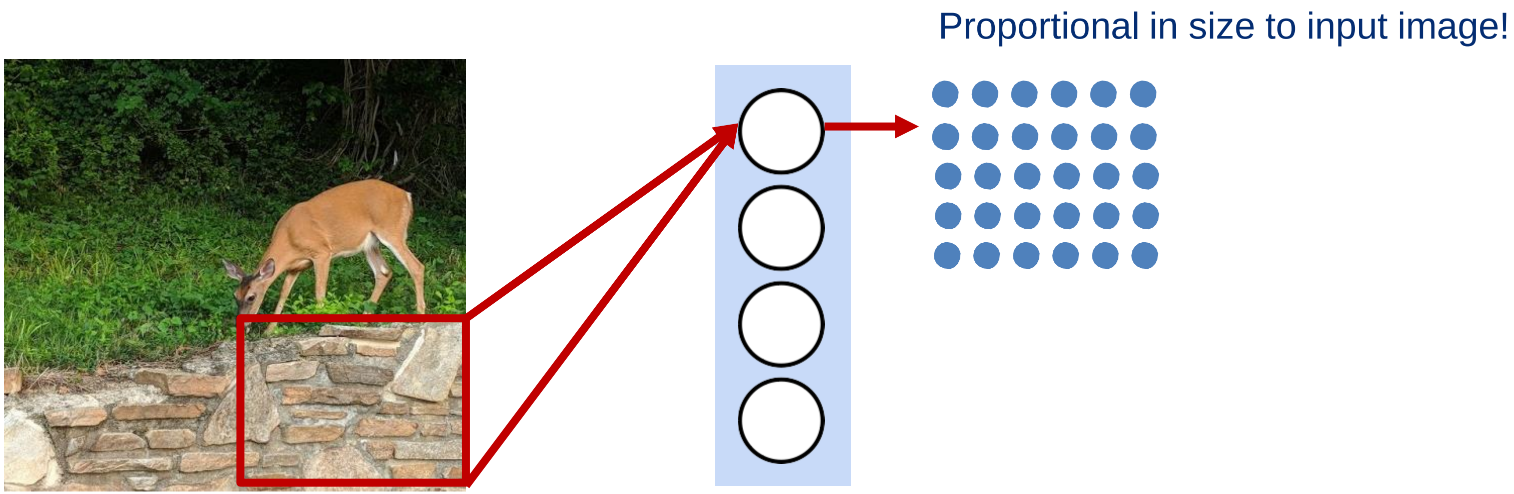


Now

→ Perceptron evaluates this region of the image

→ We can arrange these outputs in form of the original picture

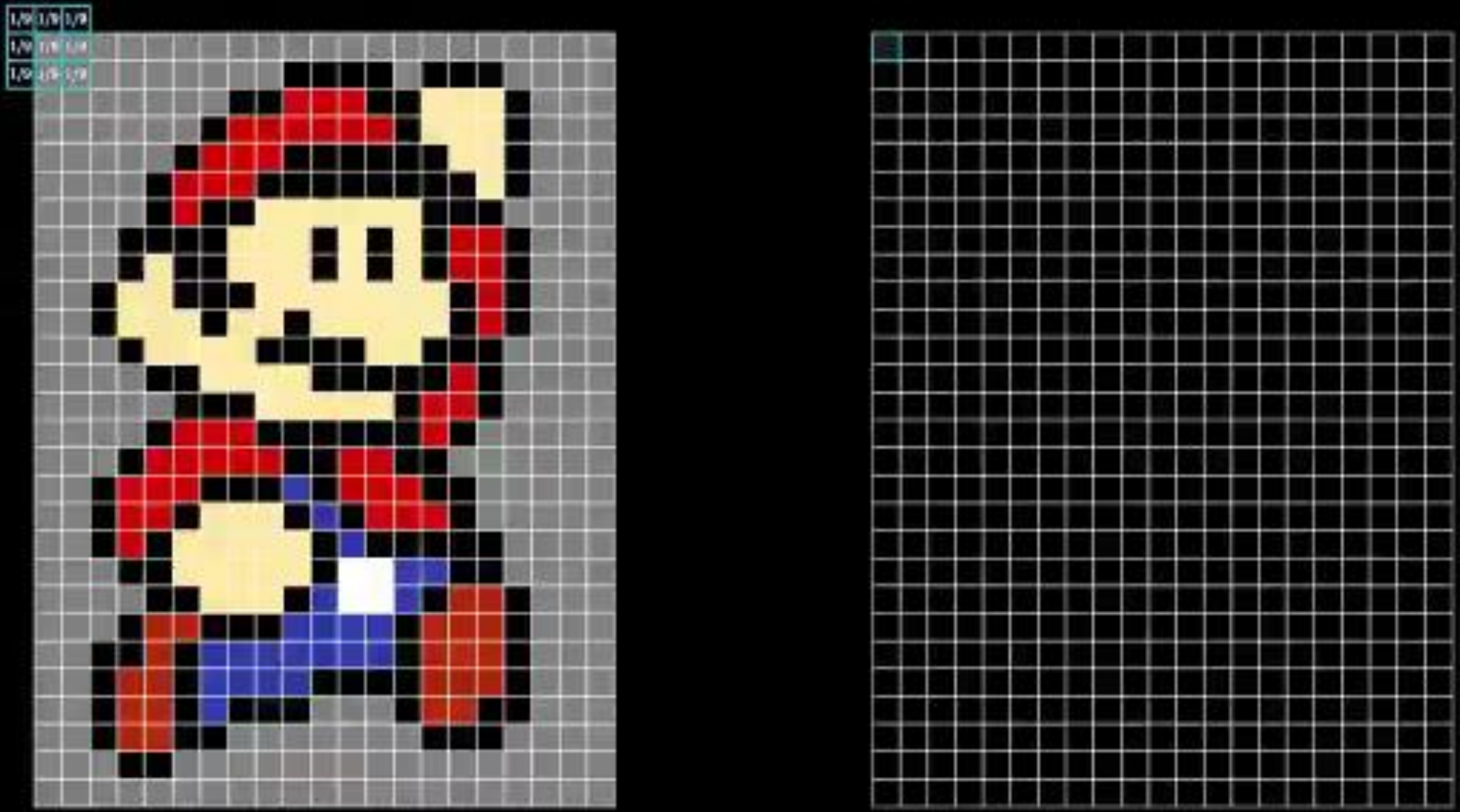
Scanning: A Closer Look



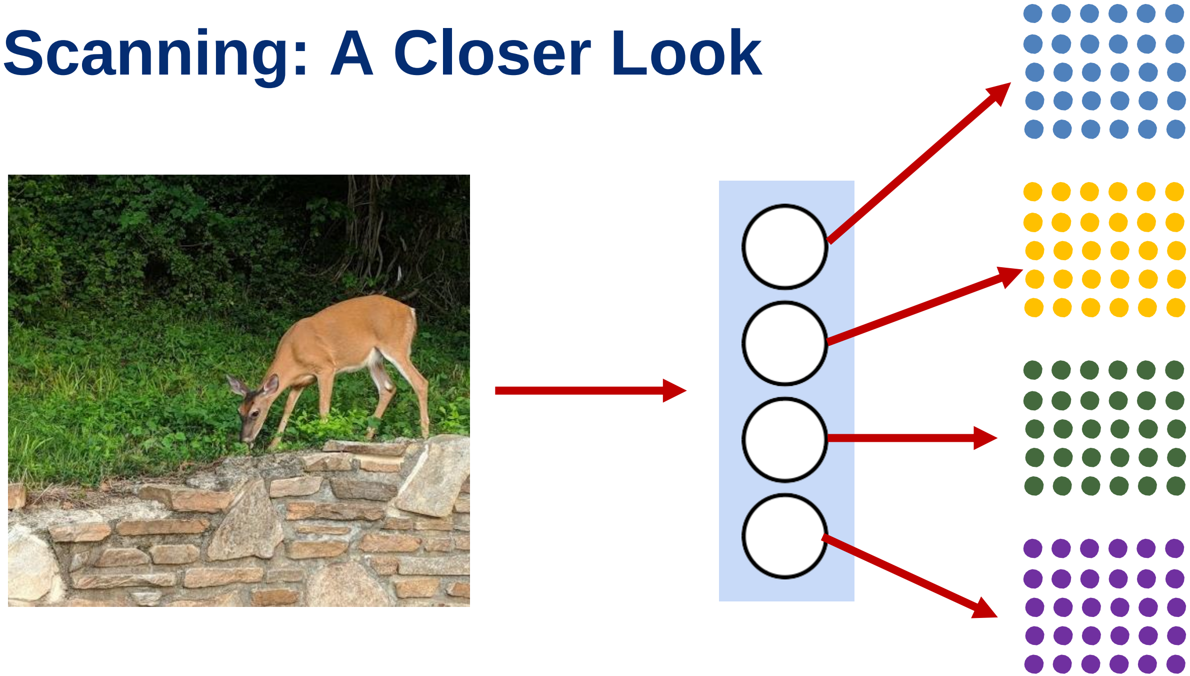
Now

→ Perceptron evaluates this region of the image

→ We can arrange these outputs in form of the original picture

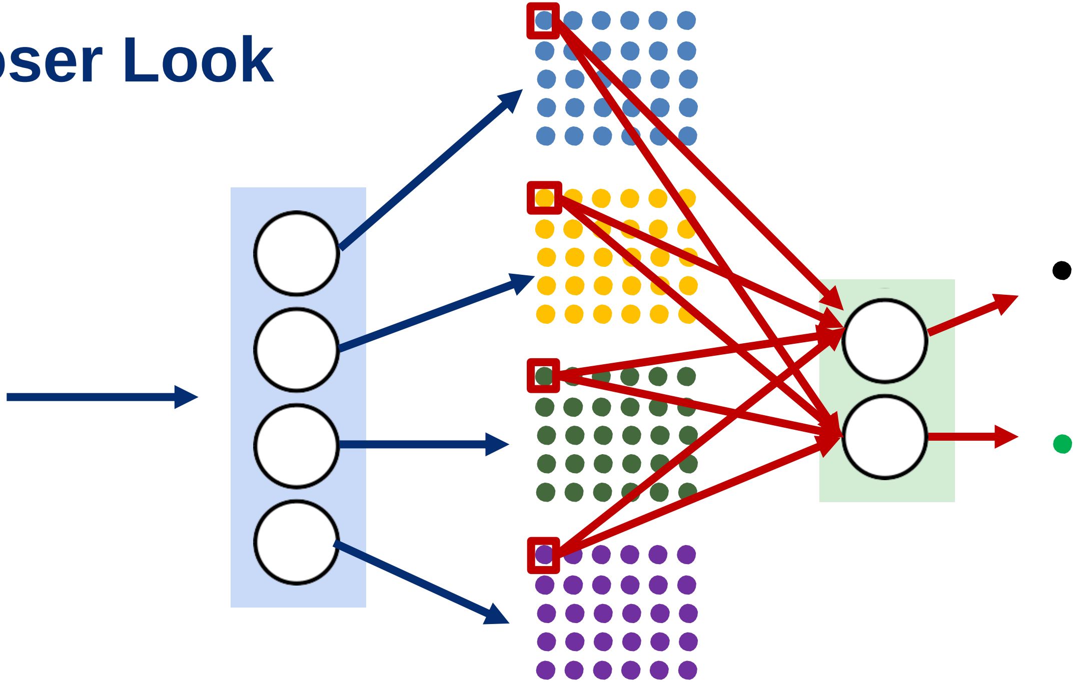

1/9 1/9 1/9
1/9 1/9 1/9
1/9 1/9 1/9

Scanning: A Closer Look



This can be repeated for every perceptron's output

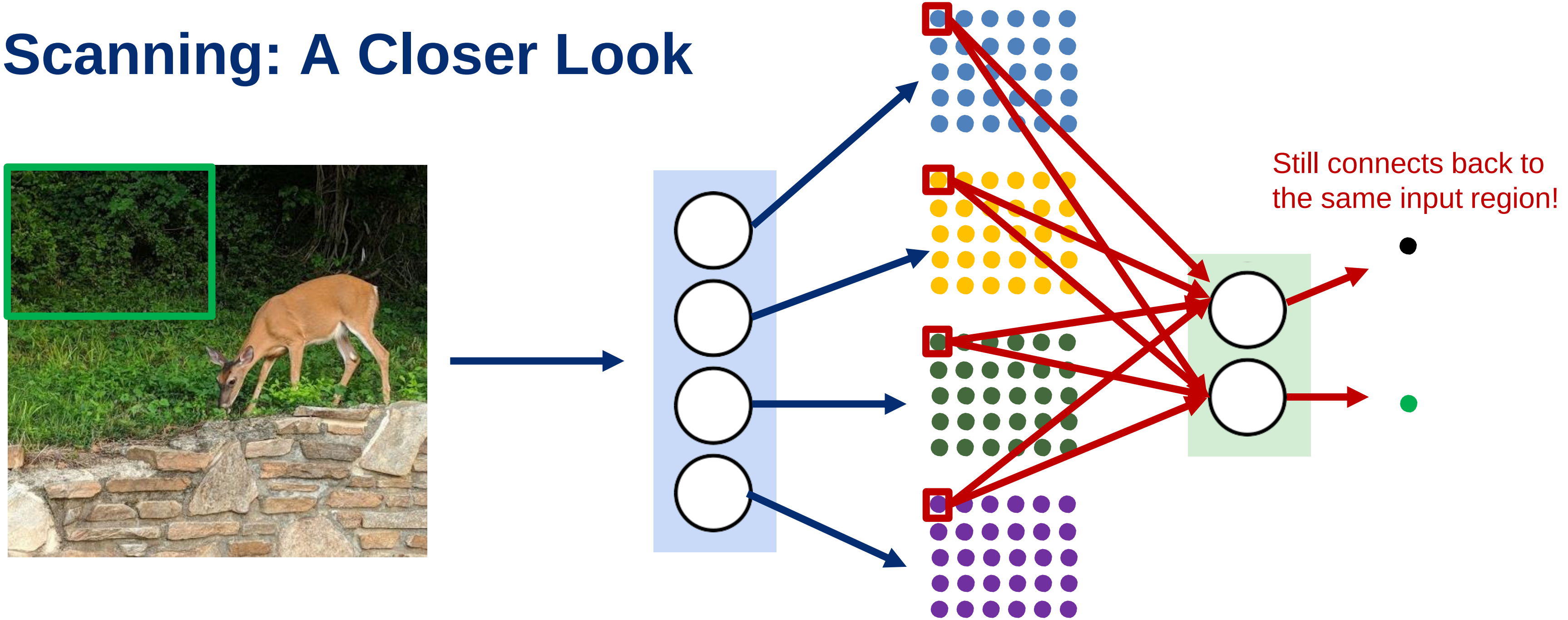
Scanning: A Closer Look



We can recurse the logic!

Now, perceptrons are jointly scanning multiple “images”

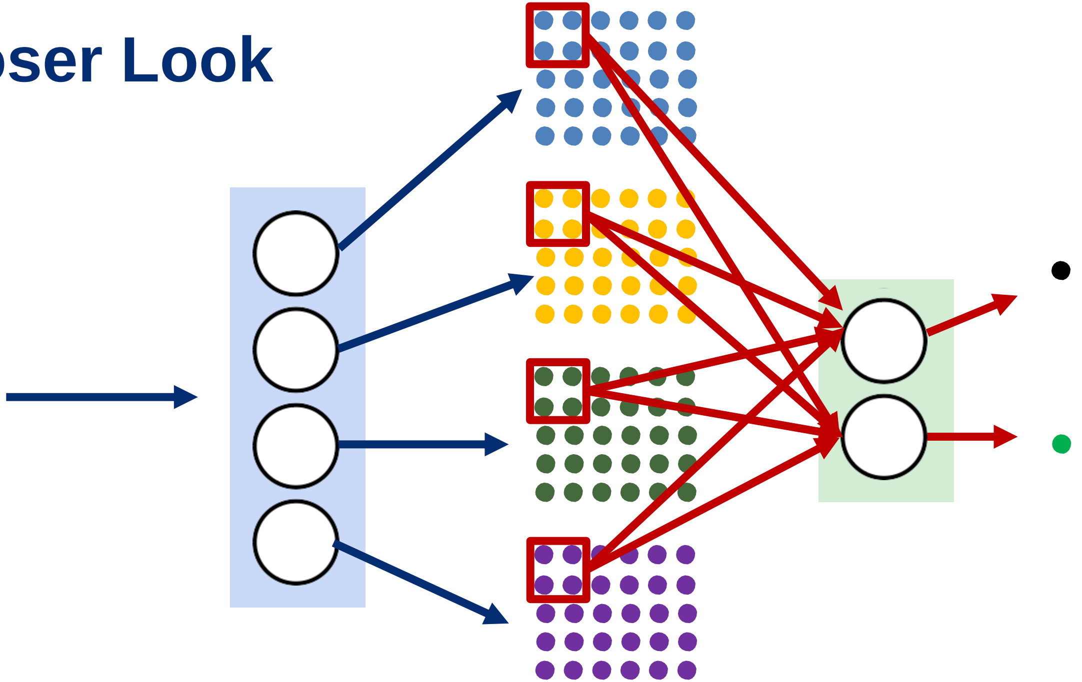
Scanning: A Closer Look



We can recurse the logic!

Now, perceptrons are jointly scanning multiple “images”

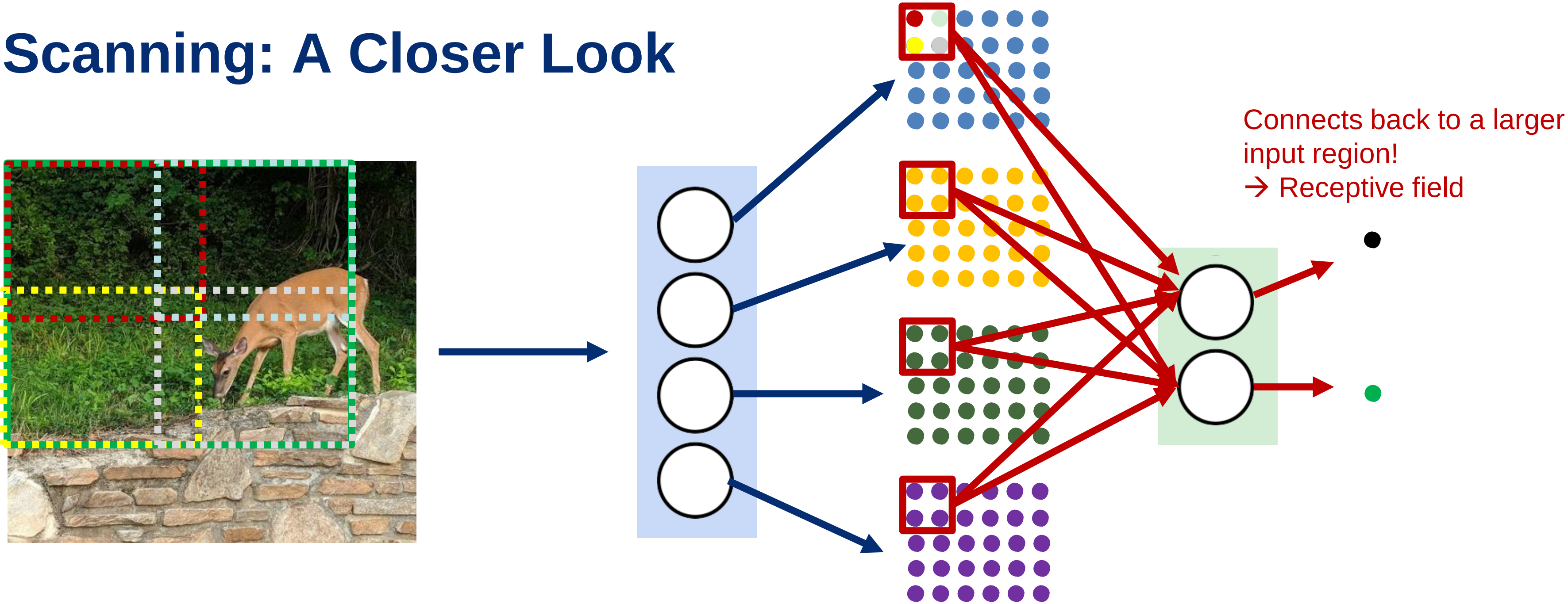
Scanning: A Closer Look



We can recurse the logic!

Now, perceptrons are jointly scanning multiple “images”

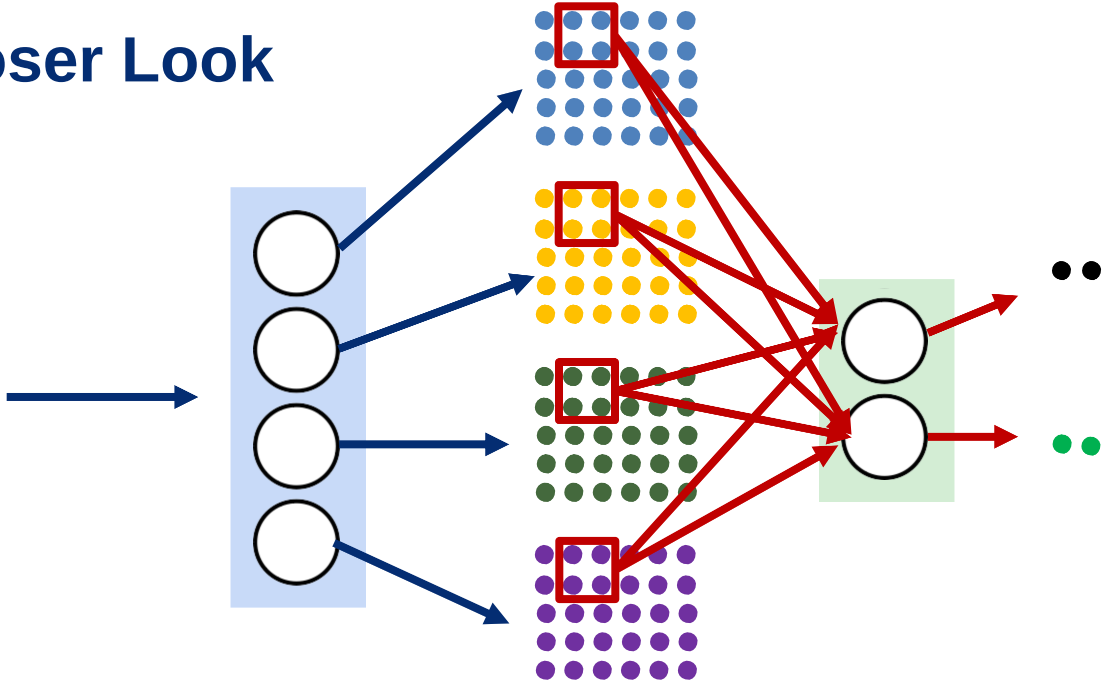
Scanning: A Closer Look



We can recurse the logic!

Now, perceptrons are jointly scanning multiple “images”

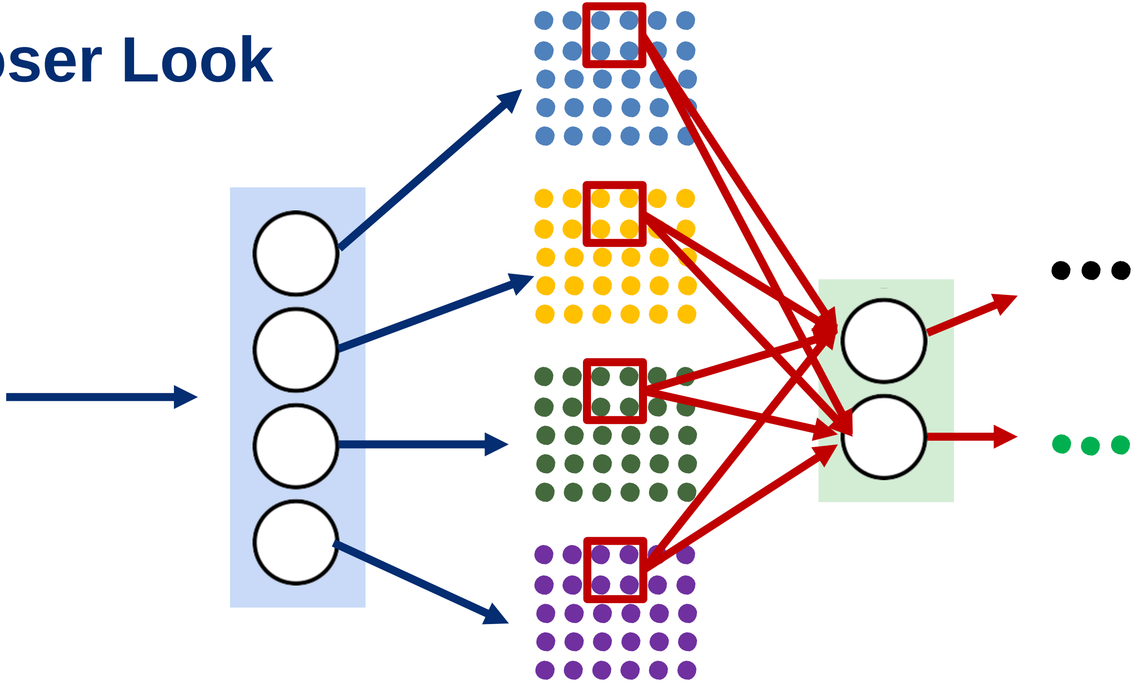
Scanning: A Closer Look



We can recurse the logic!

Now, perceptrons are jointly scanning multiple “images”

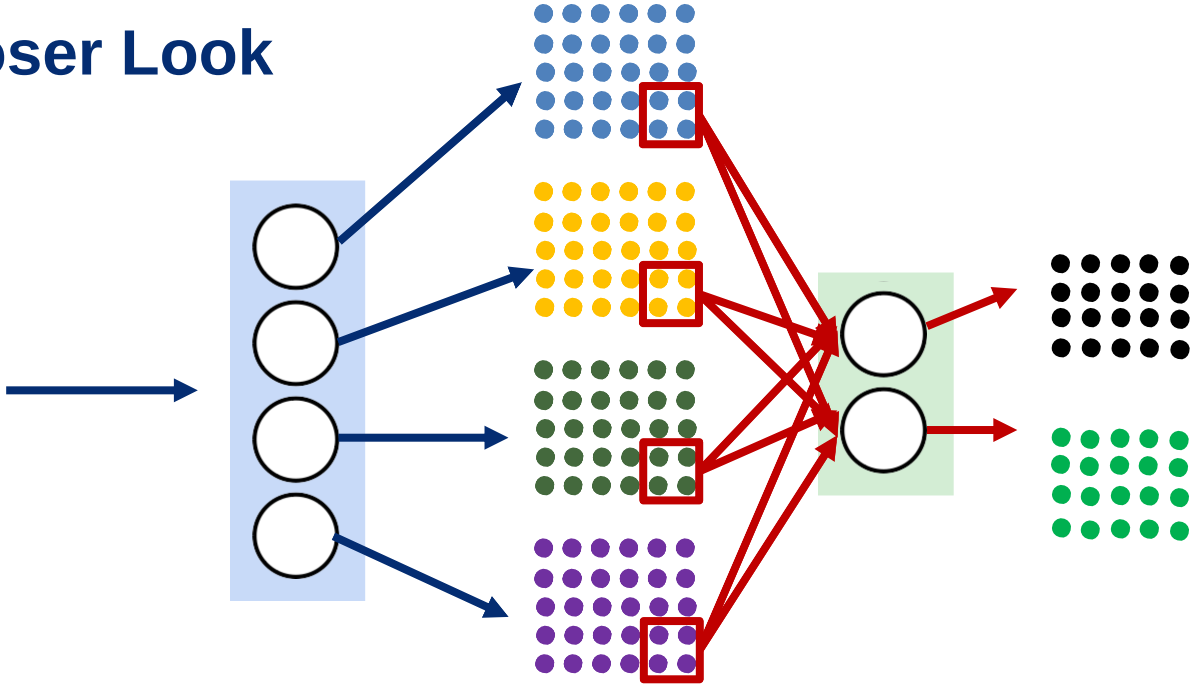
Scanning: A Closer Look



We can recurse the logic!

Now, perceptrons are jointly scanning multiple “images”

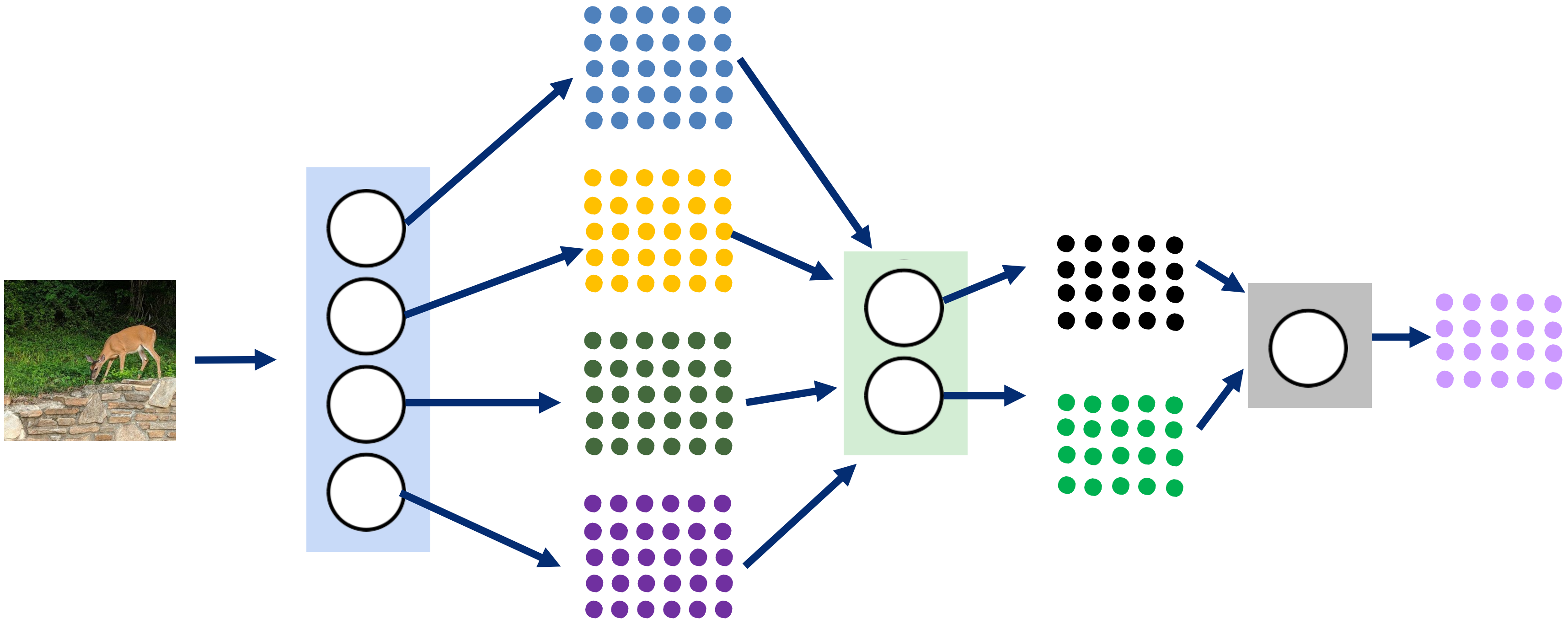
Scanning: A Closer Look



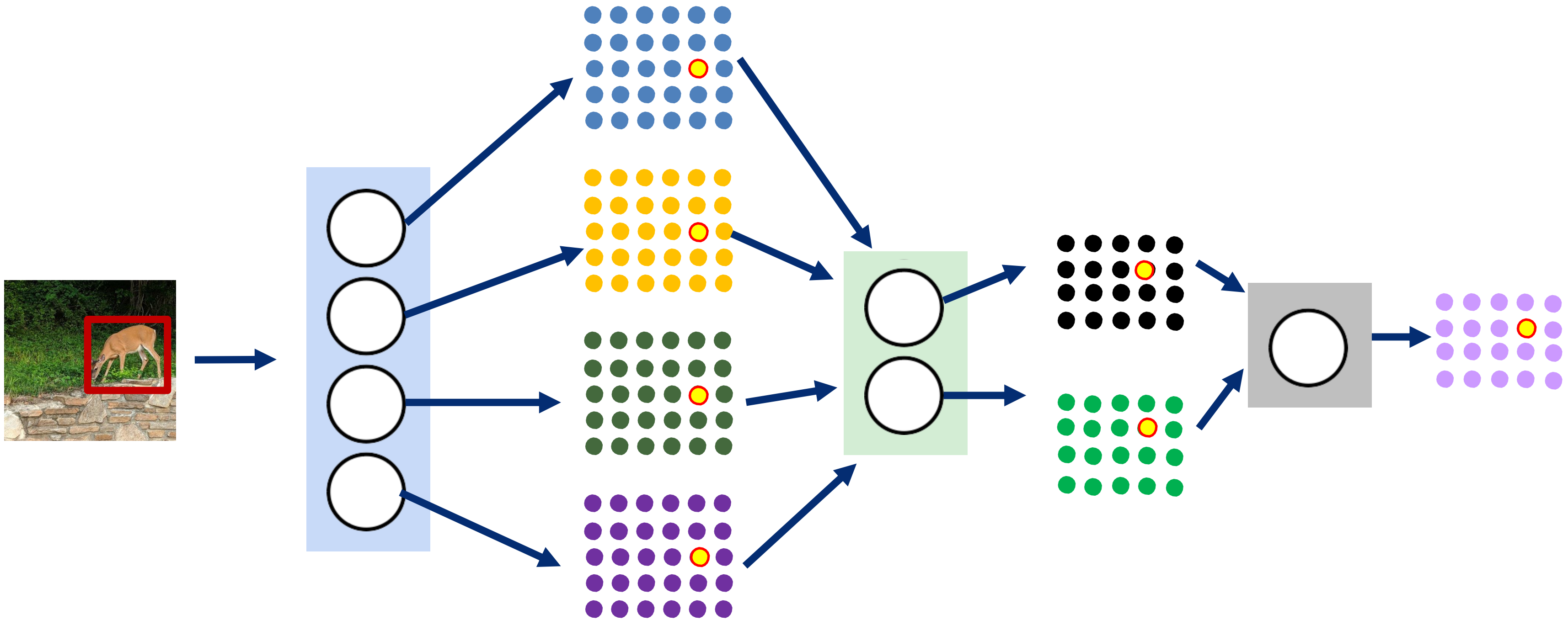
We can recurse the logic!

Now, perceptrons are jointly scanning multiple “images”

Scanning: A Closer Look



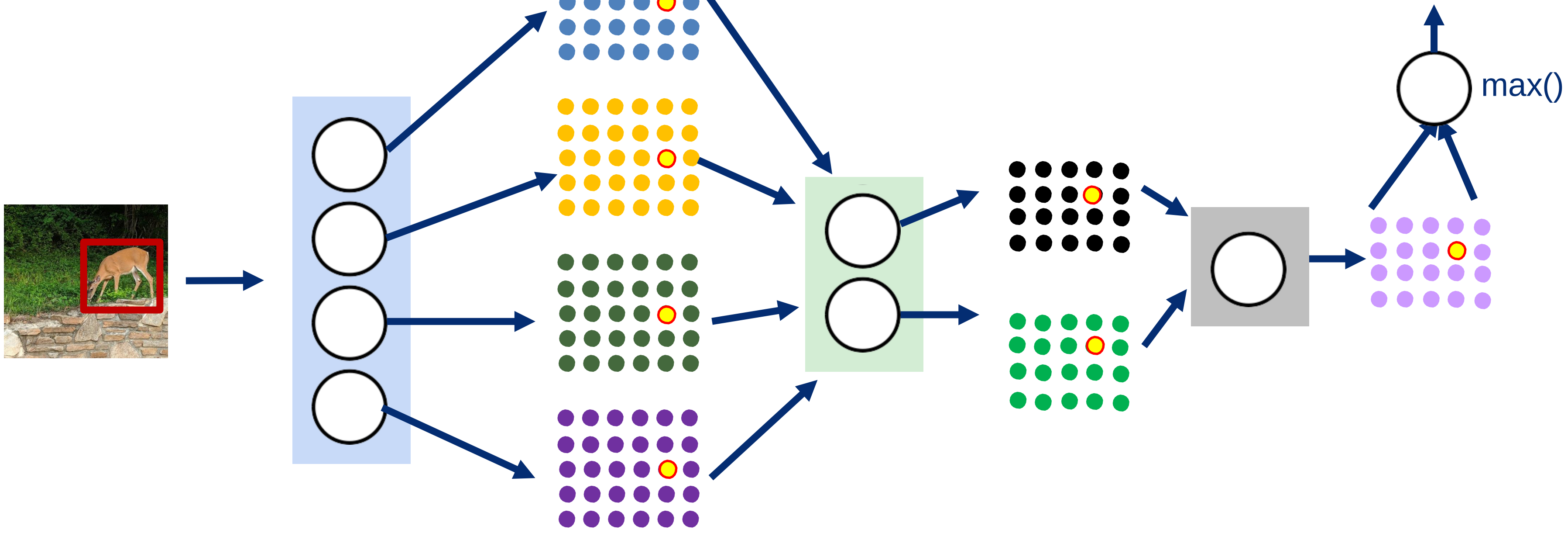
Scanning: A Closer Look



Output layer must consider last hidden layer!

→ “Detect location of object in image”

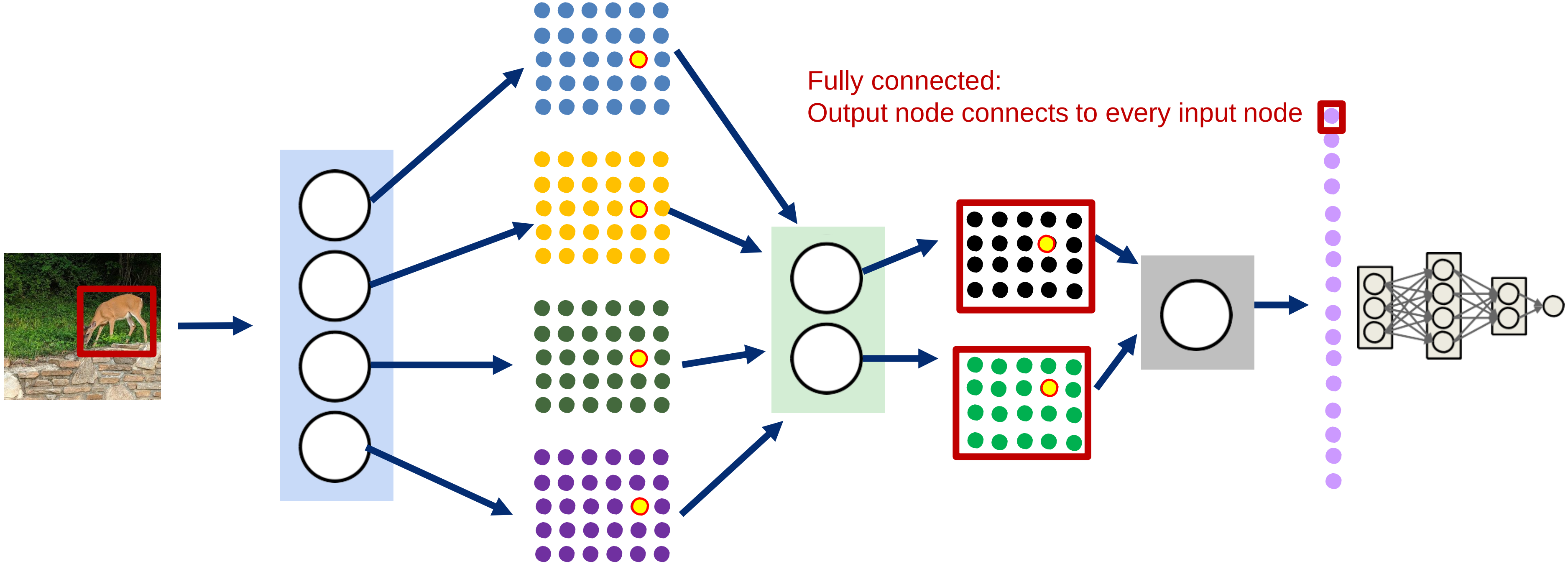
Scanning: A Closer Look



Output layer must consider last hidden layer!

→ **"Is there such object in image?"**

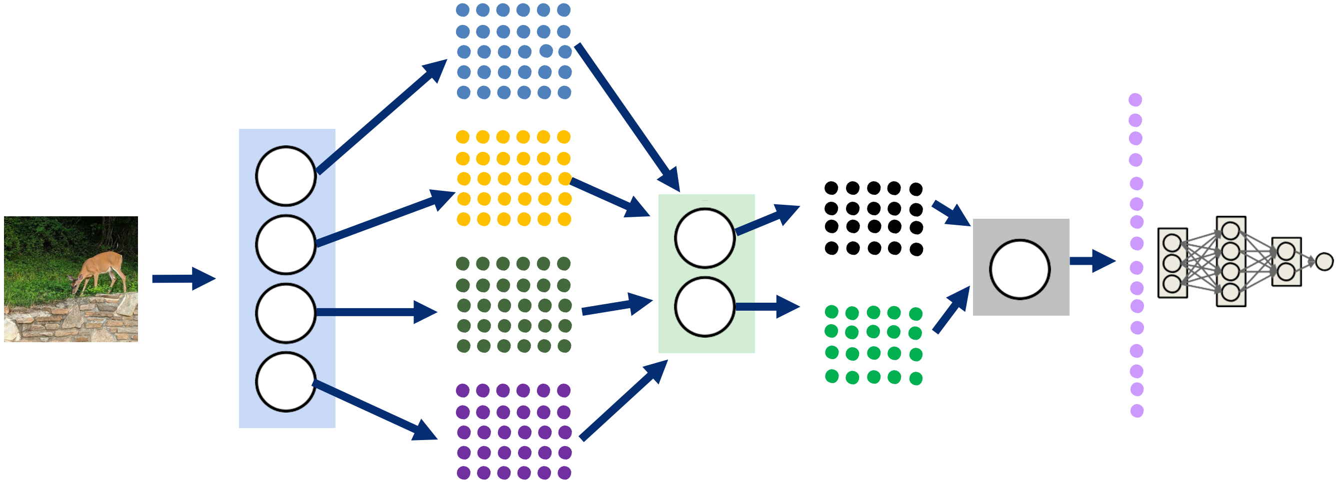
Scanning: A Closer Look



“Is there such object in image?”

→ Flatten output since spatial configuration is no longer important

Hierarchical Build-up of Features



Sharing parameters

→ Distribution forces localized patterns in lower layers (generalizability)

→ Reduction of parameters (because of sharing)

Recap

- Instead of passing a whole image into a MLP, we can slide a smaller MLP over the image
- This allowed us to recreate an image-like structure in the hidden layer

- We can use different MLPs (with different parameters) for every location
- Or, we can share parameters across the spatial domain
  → Translation invariance!

- Localized features in lower layers
- More abstract, complex features in deeper layers

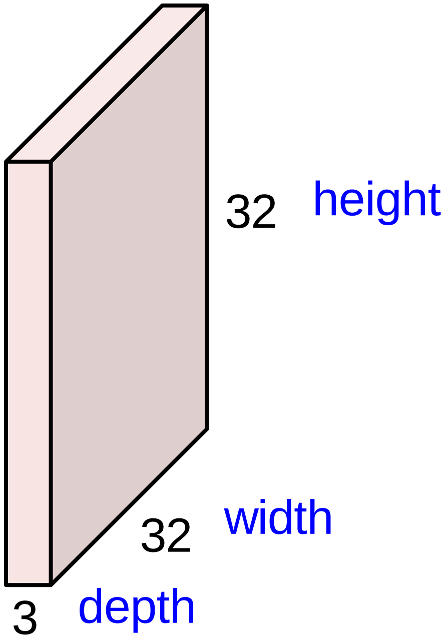
Q: How do we achieve this in practice?

Convolutional Neural Networks

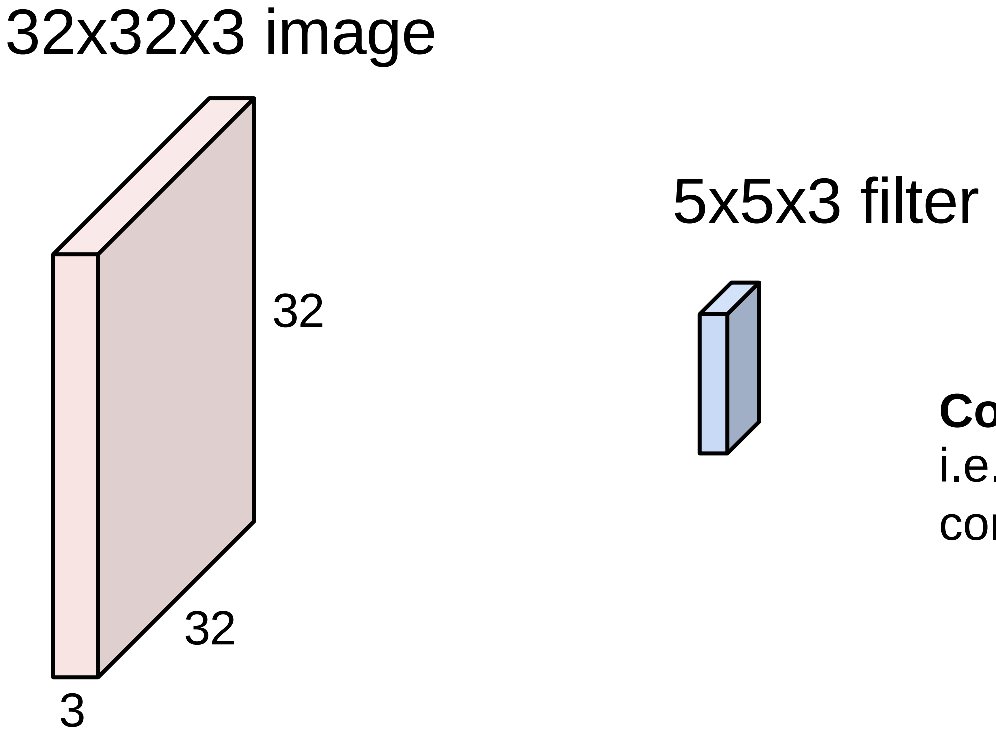
Convolutions

Convolution Layer

32x32x3 image -> preserve spatial structure



Convolution Layer

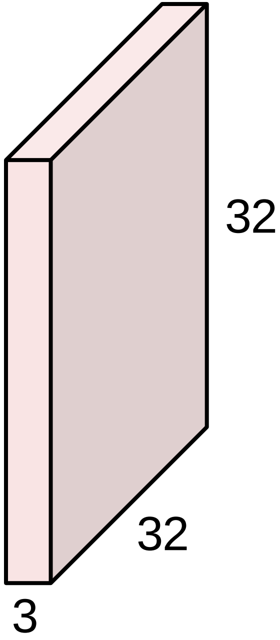


Convolve the filter with the image
i.e. “slide over the image spatially, computing dot products”

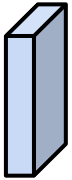
Convolution Layer

Filters always extend the full depth of the input volume

32x32x3 image

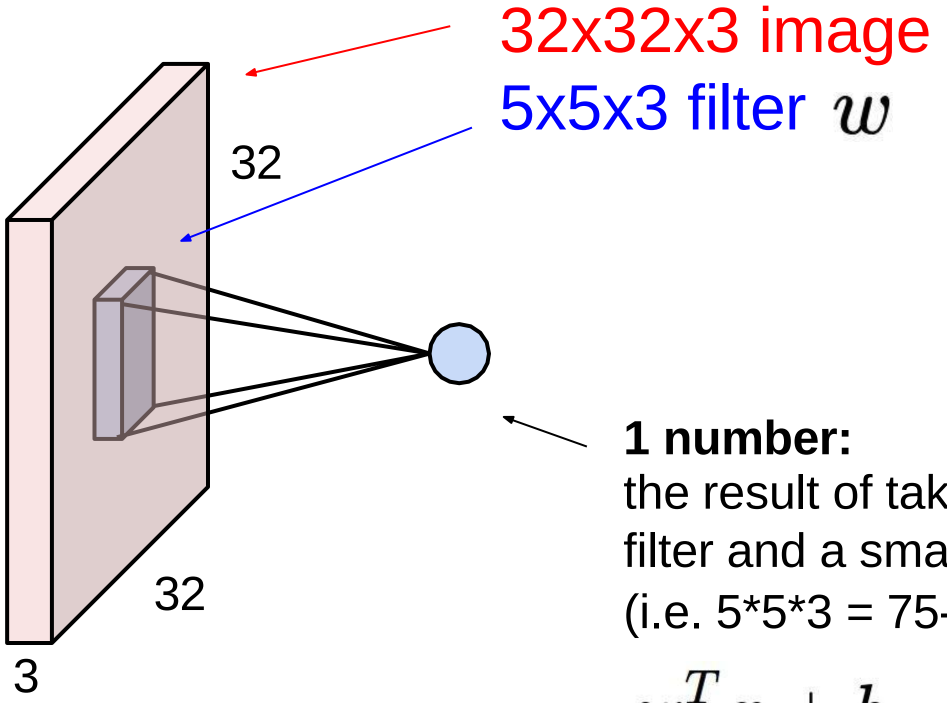


5x5x3 filter



Convolve the filter with the image i.e. “slide over the image spatially, computing dot products”

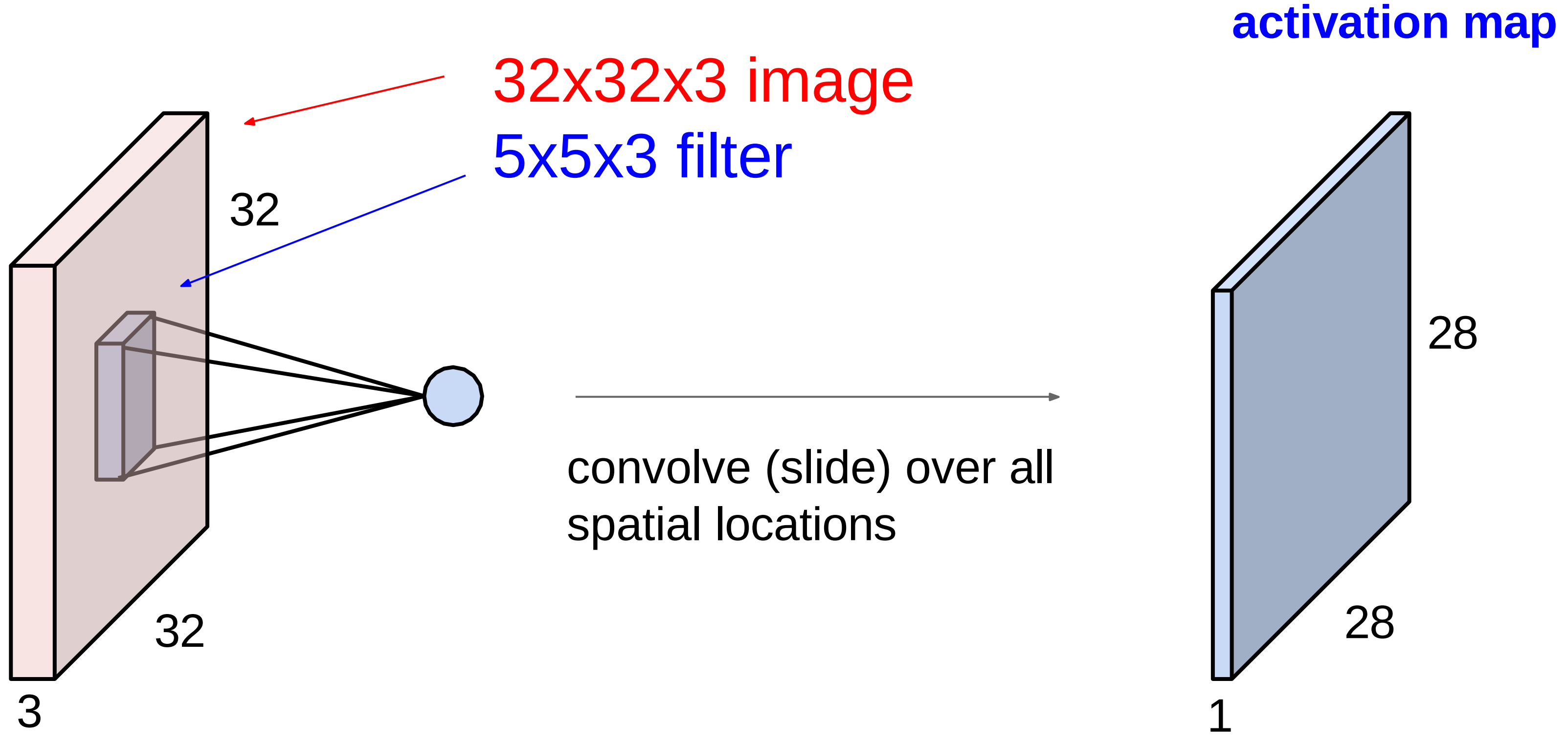
Convolution Layer



1 number:
the result of taking a dot product between the filter and a small 5x5x3 chunk of the image
(i.e. 5*5*3 = 75-dimensional dot product + bias)

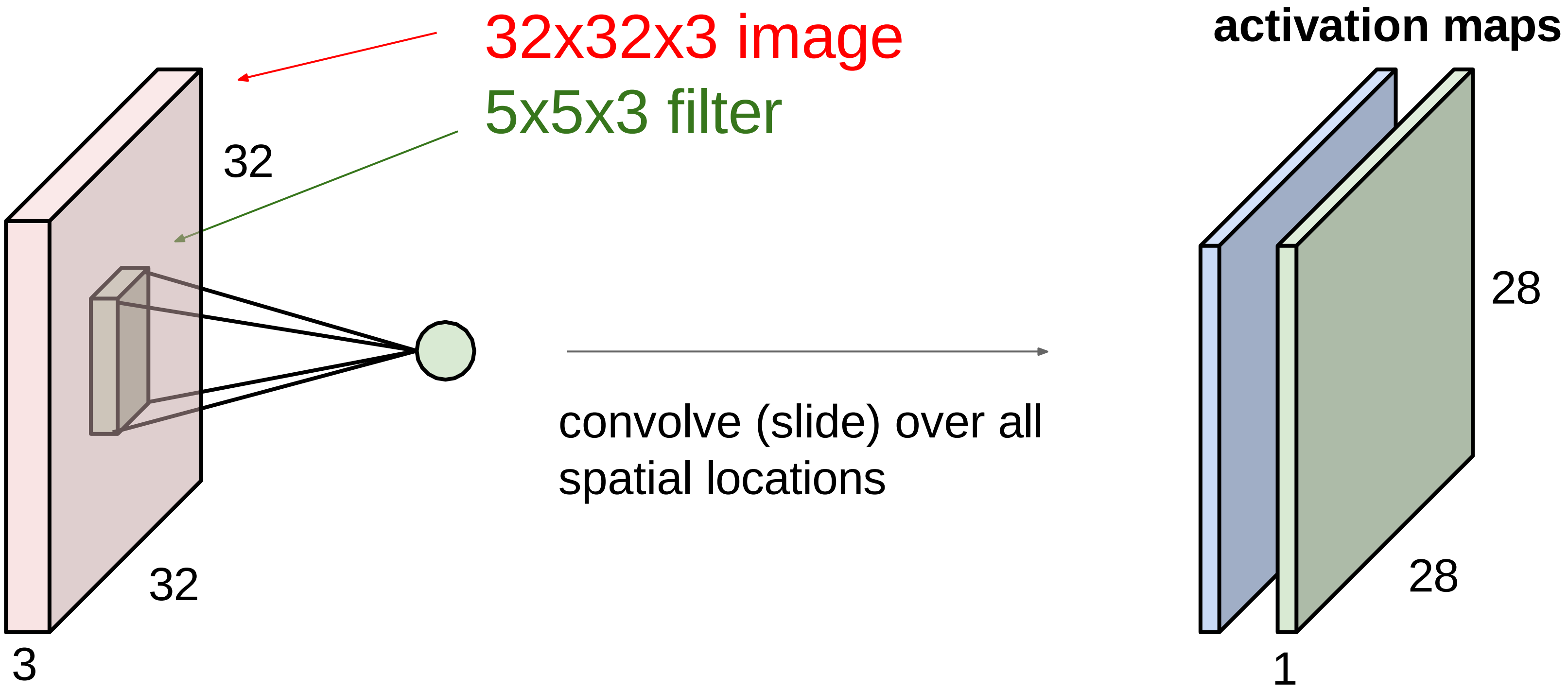
$$w^T x + b$$

Convolution Layer

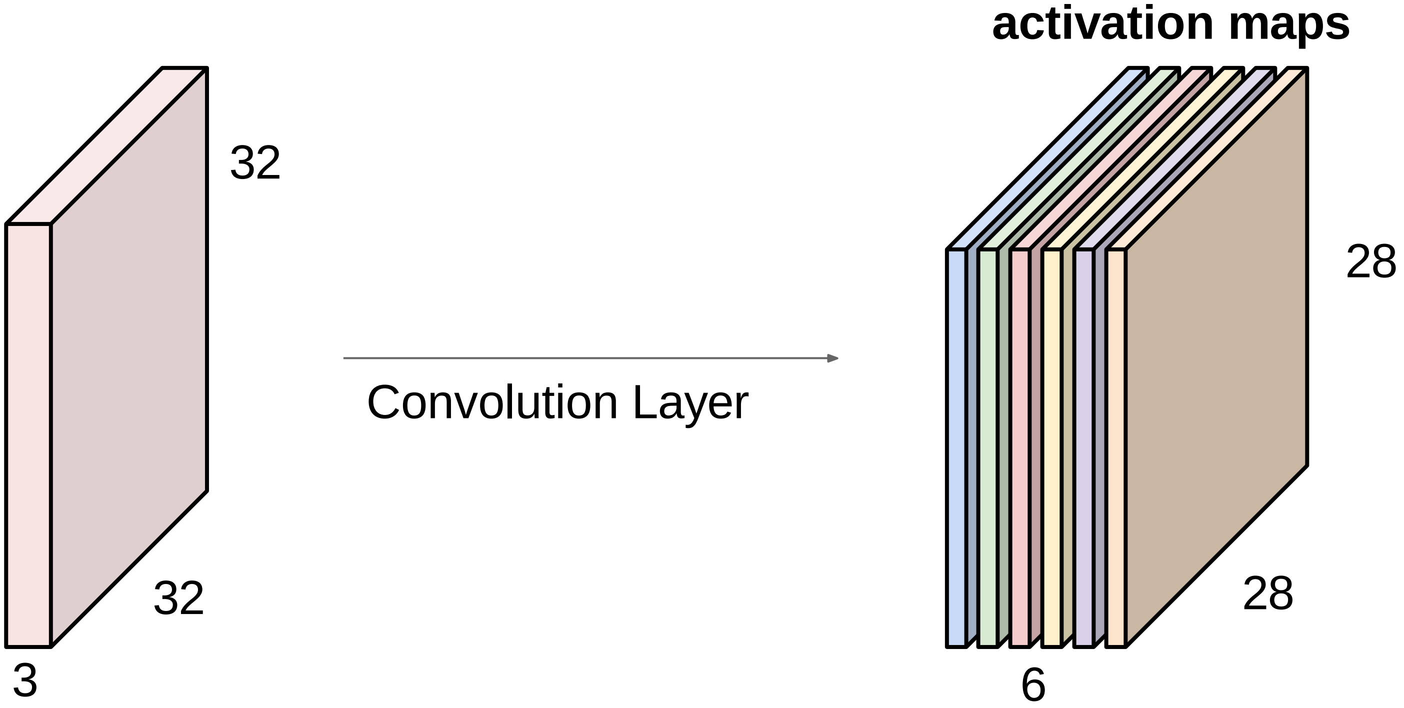


Convolution Layer

consider a second, green filter

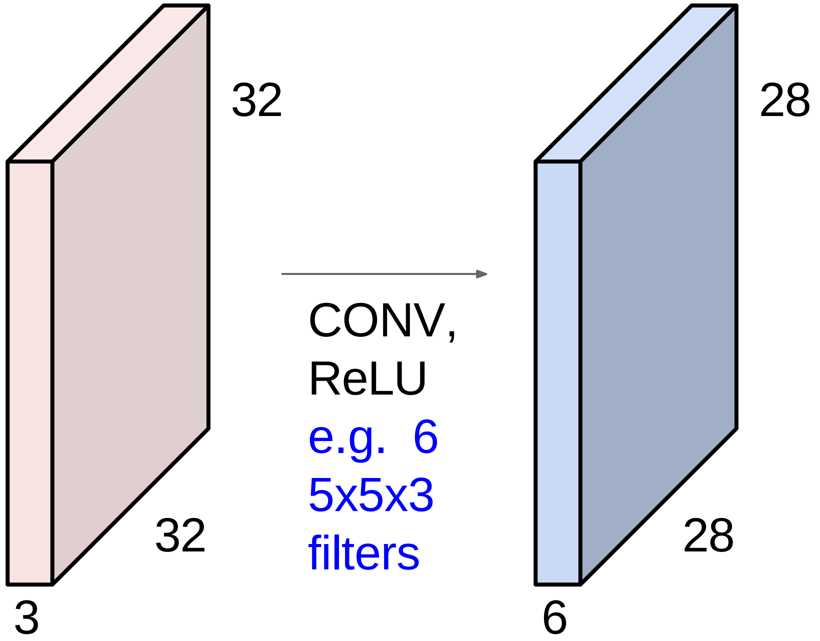


For example, if we had 6 5x5 filters, we'll get 6 separate activation maps:

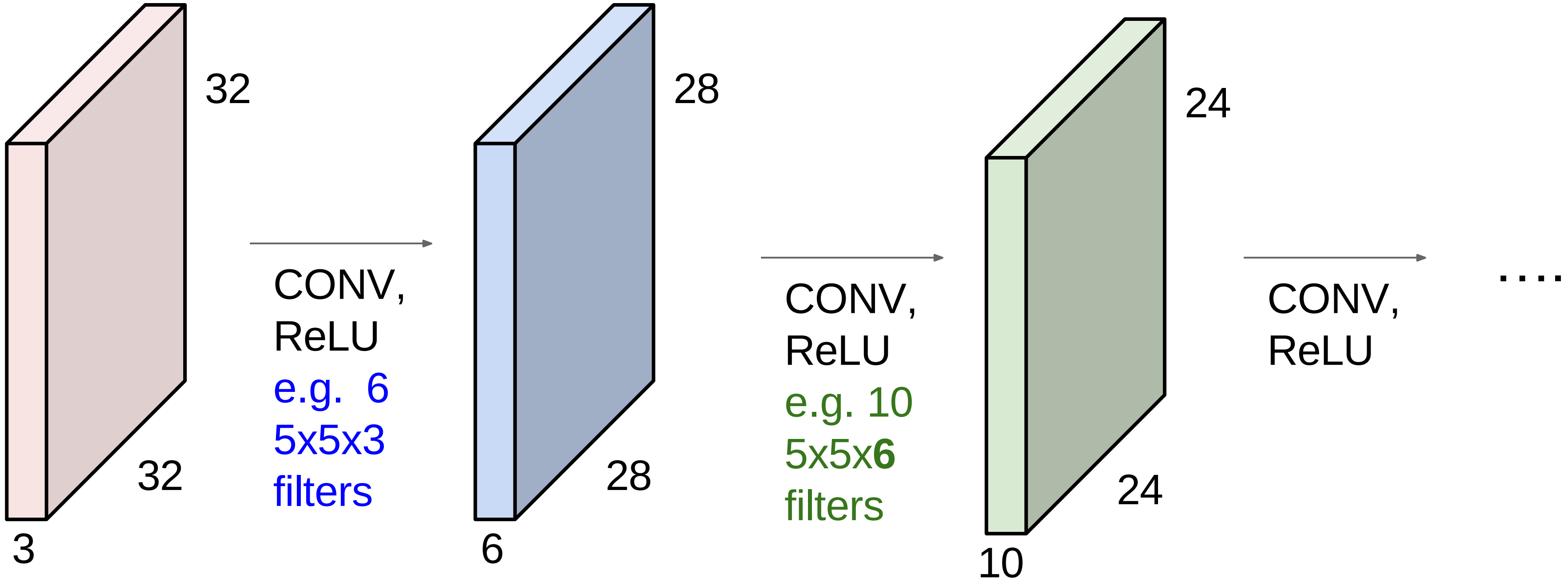


We stack these up to get a "new image" of size 28x28x6!

Preview: ConvNet is a sequence of Convolution Layers, interspersed with activation functions



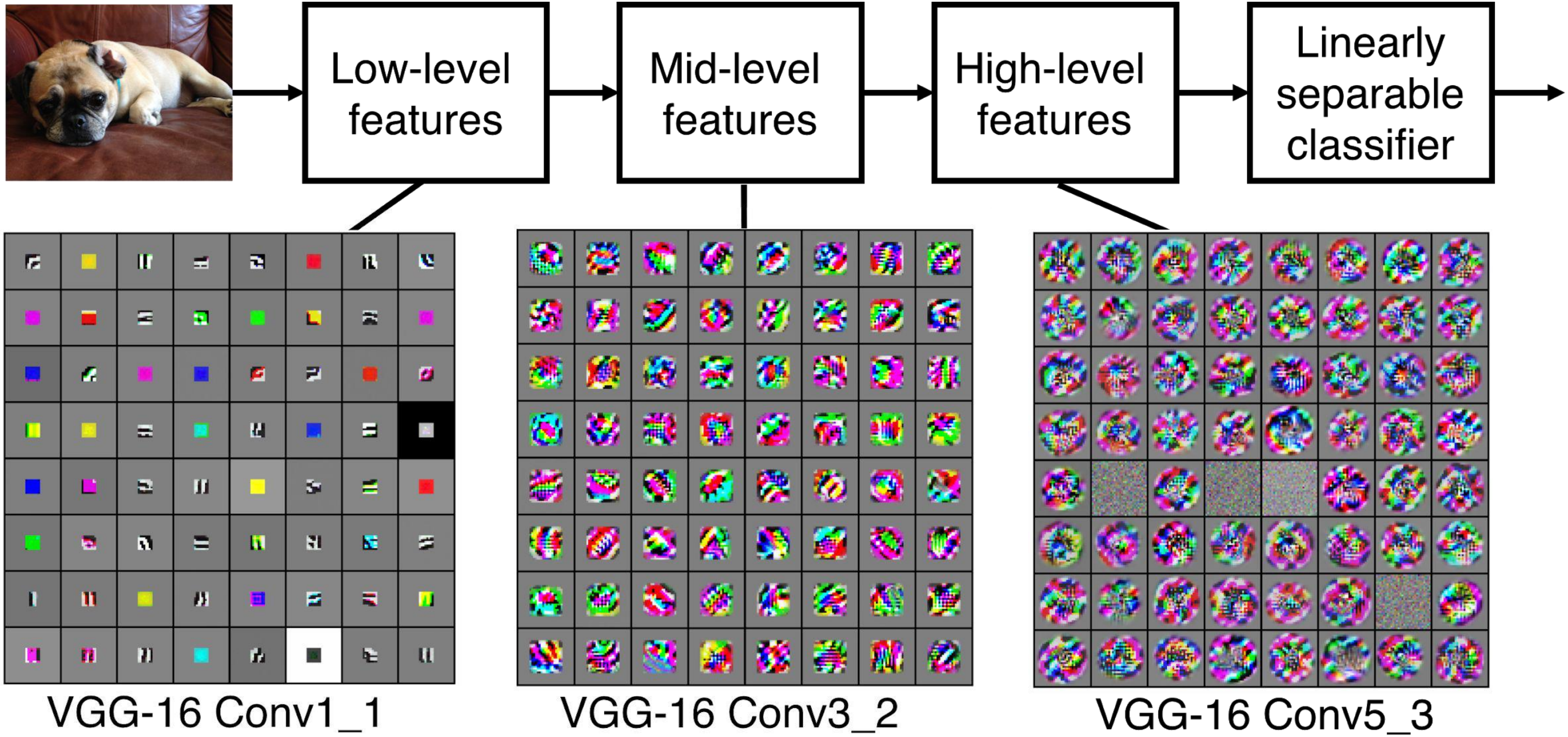
Preview: ConvNet is a sequence of Convolution Layers, interspersed with activation functions



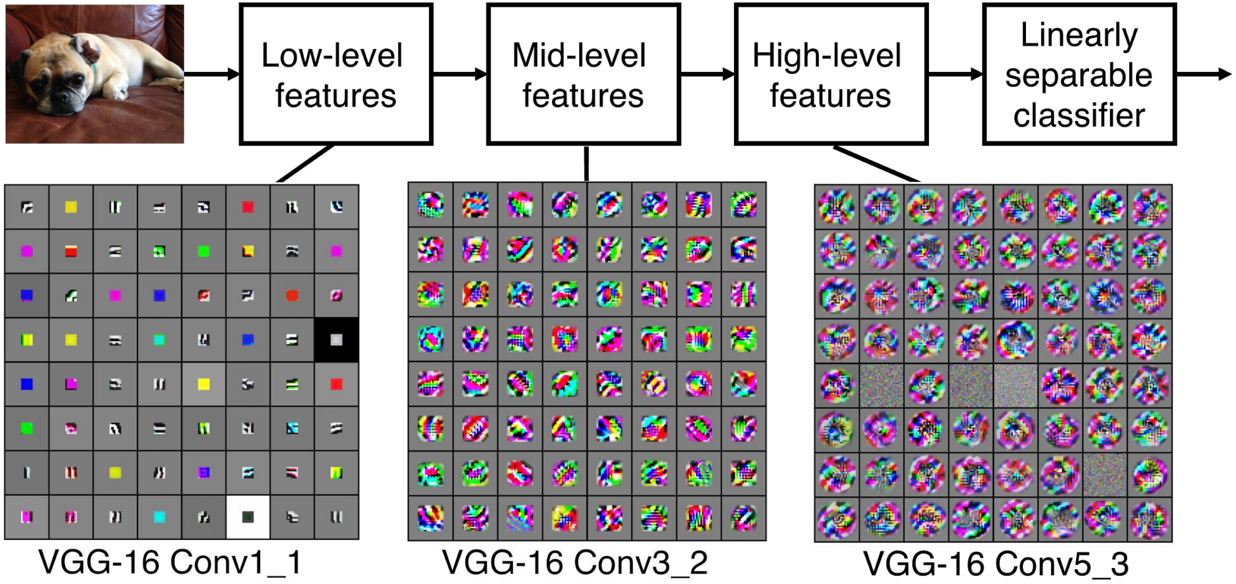
Preview

[Zeiler and Fergus 2013]

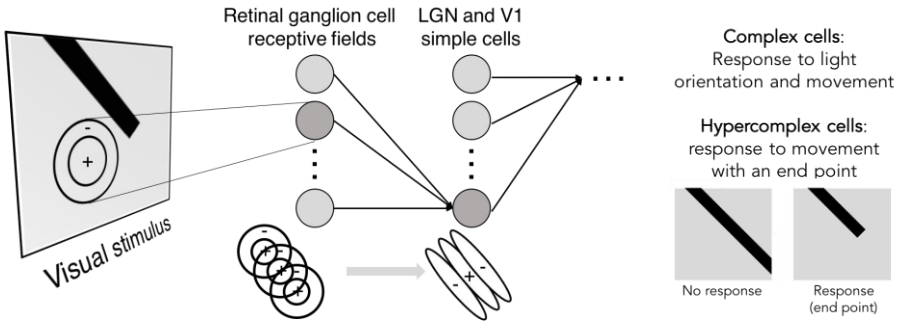
Visualization of VGG-16 by Lane McIntosh. VGG-16 architecture from [Simonyan and Zisserman 2014].



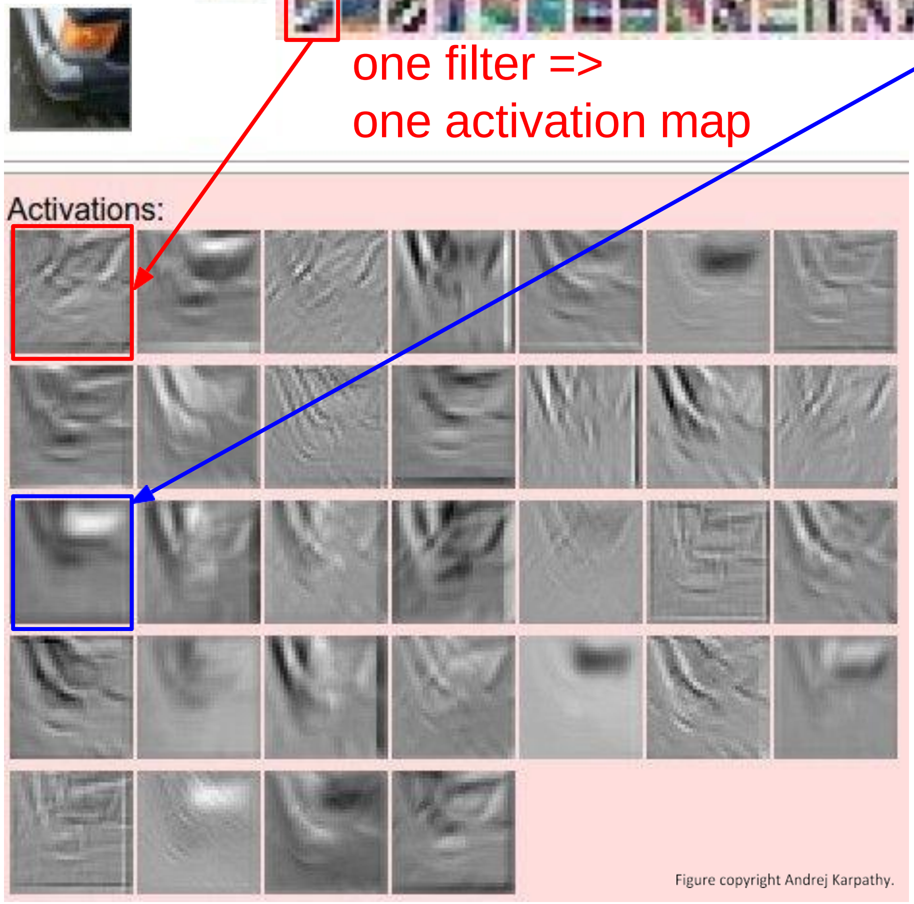
Preview



Low-level features
Mid-level features
High-level features
Linearly separable classifier
VGG-16 Conv1_1
VGG-16 Conv3_2
VGG-16 Conv5_3



Retinal ganglion cell receptive fields
LGN and V1 simple cells
Visual stimulus
Complex cells:
Response to light orientation and movement
Hypercomplex cells:
response to movement with an end point
No response
Response (end point)

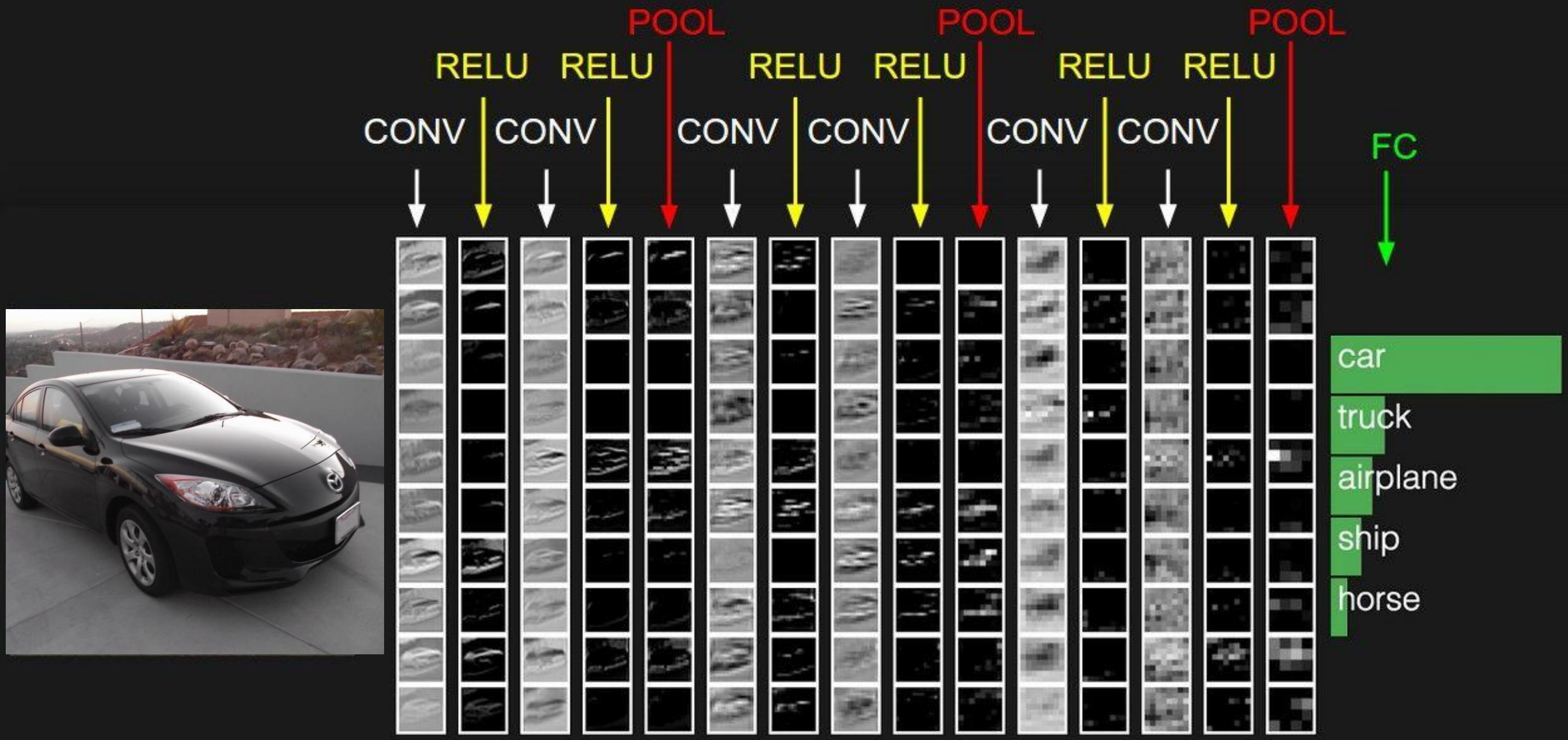


example 5x5 filters
(32 total)

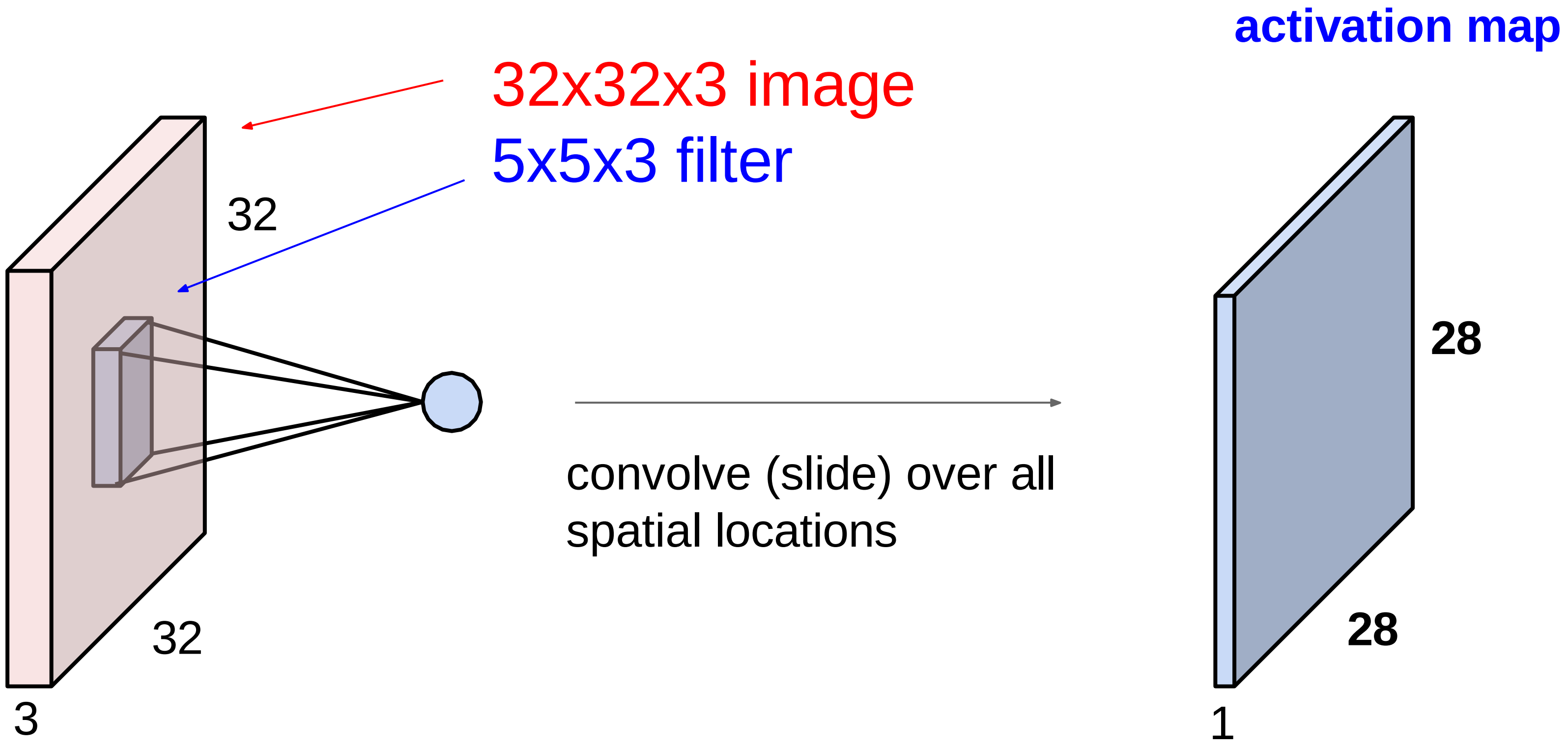
We call the layer convolutional because it is related to convolution of two signals:

$$f[x,y] * g[x,y] = \sum_{n_1=-\infty}^{\infty} \sum_{n_2=-\infty}^{\infty} f[n_1,n_2] \cdot g[x-n_1, y-n_2]$$

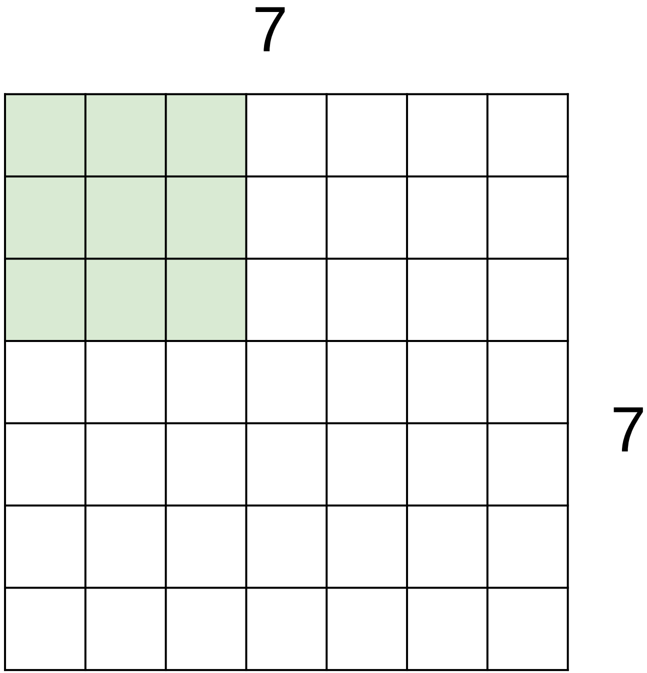
elementwise multiplication and sum of a filter and the signal (image)


POOL
POOL
POOL
RELU
RELU
RELU
RELU
RELU
RELU
CONV
CONV
CONV
CONV
CONV
CONV
FC
car
truck
airplane
ship
horse

A closer look at spatial dimensions:

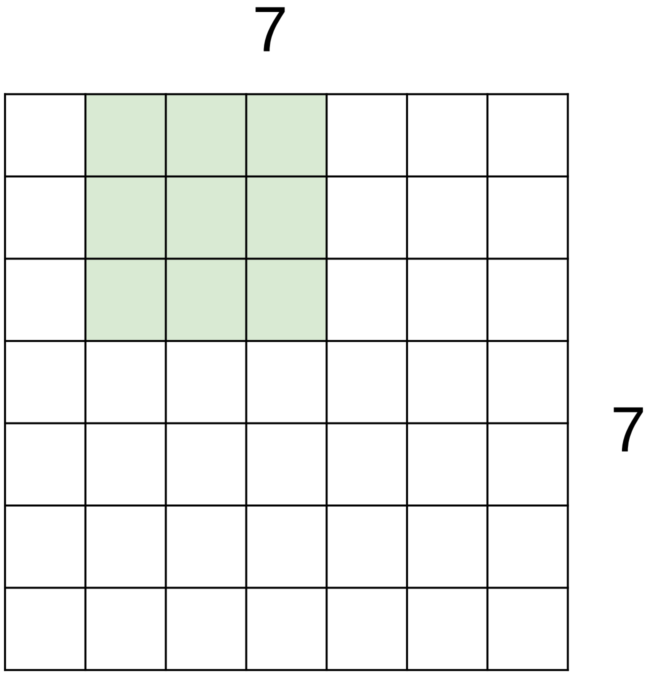


A closer look at spatial dimensions:



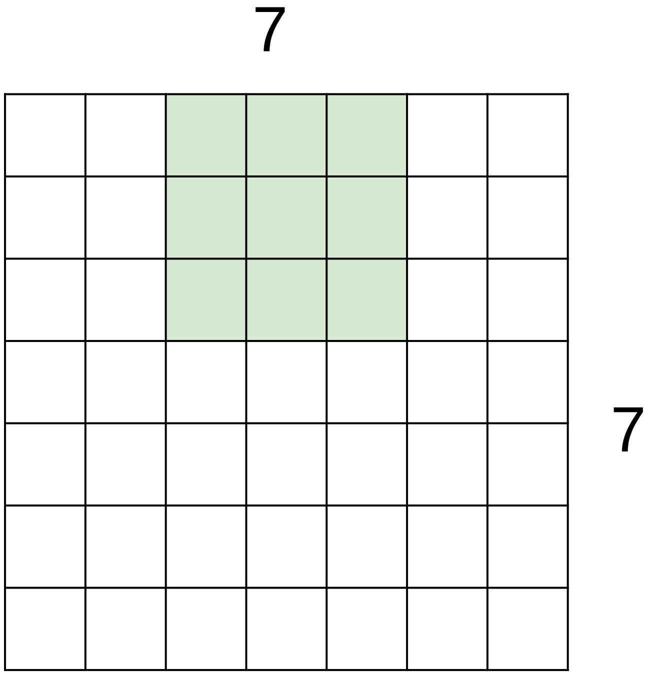
7x7 input (spatially)
assume 3x3 filter

A closer look at spatial dimensions:



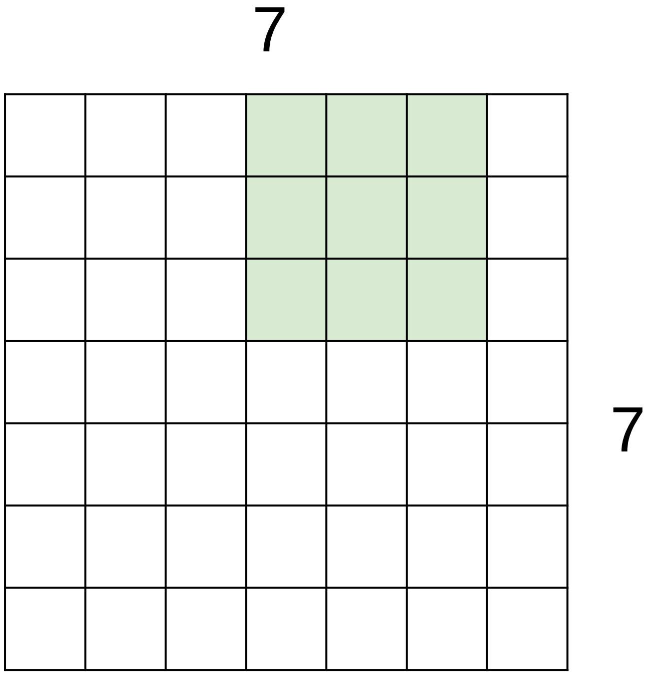
7x7 input (spatially)
assume 3x3 filter

A closer look at spatial dimensions:



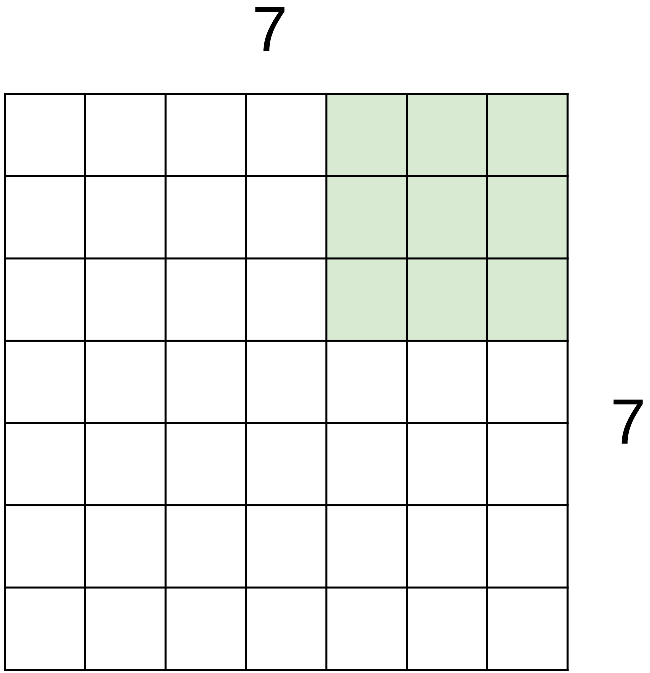
7x7 input (spatially)
assume 3x3 filter

A closer look at spatial dimensions:



7x7 input (spatially)
assume 3x3 filter

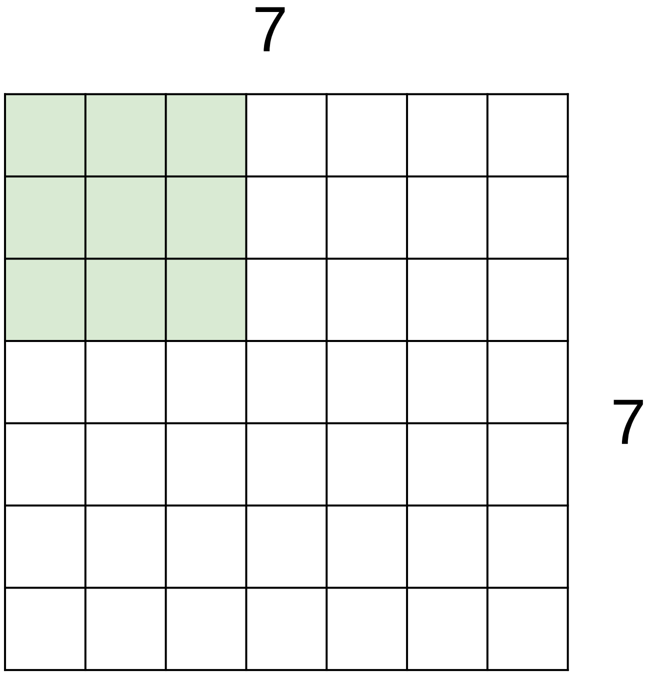
A closer look at spatial dimensions:



7x7 input (spatially)
assume 3x3 filter

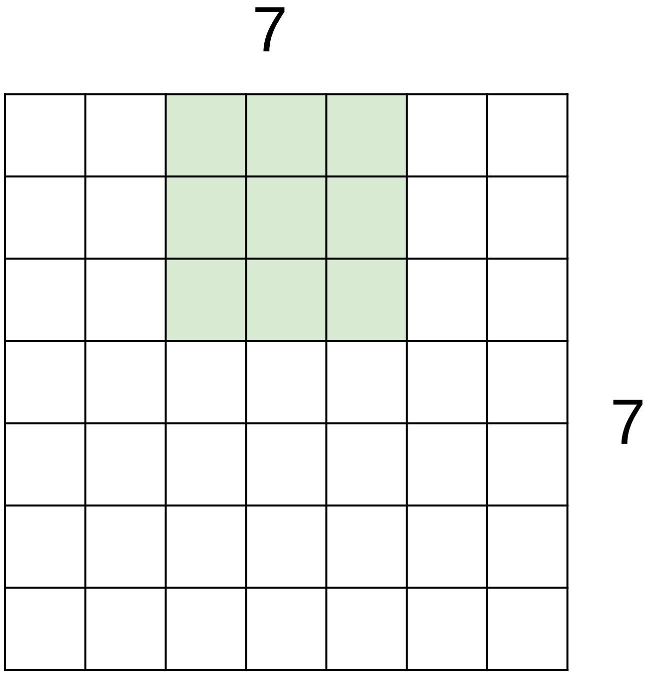
=> 5x5 output

A closer look at spatial dimensions:



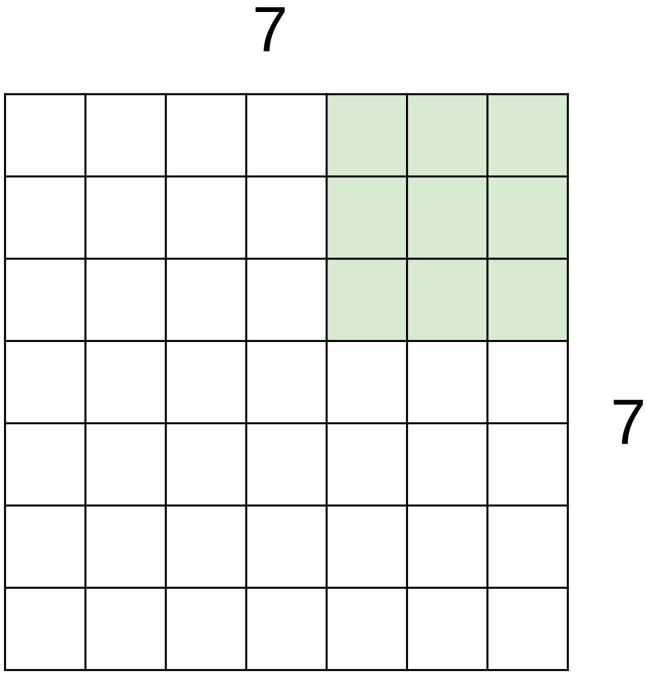
7x7 input (spatially)
assume 3x3 filter
applied **with stride 2**

A closer look at spatial dimensions:



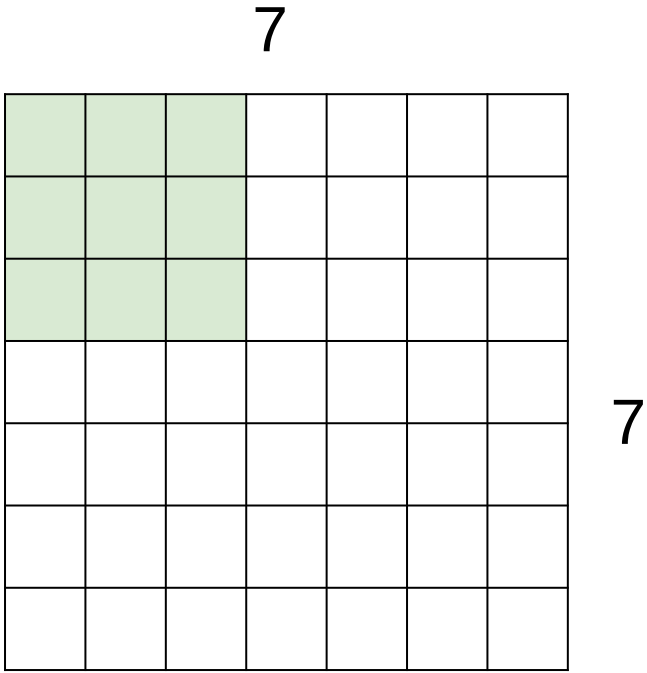
7x7 input (spatially)
assume 3x3 filter
applied **with stride 2**

A closer look at spatial dimensions:



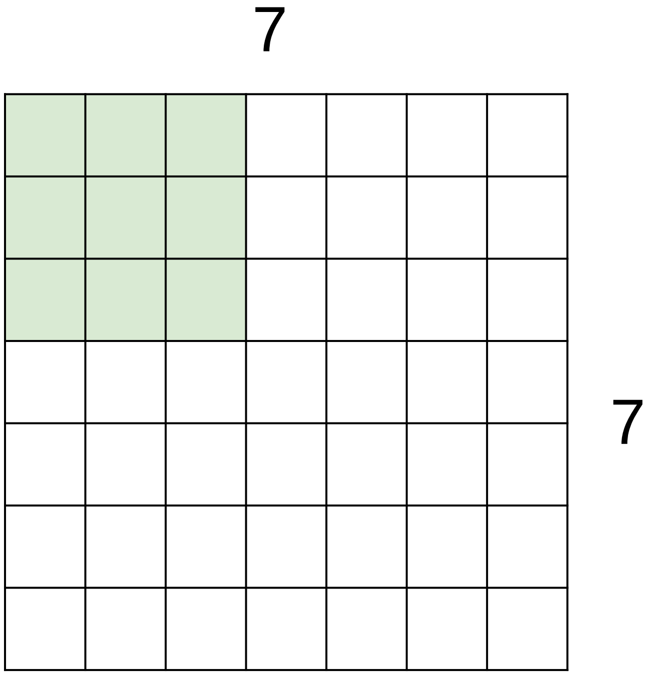
7x7 input (spatially)
assume 3x3 filter
applied **with stride 2**
=> 3x3 output!

A closer look at spatial dimensions:



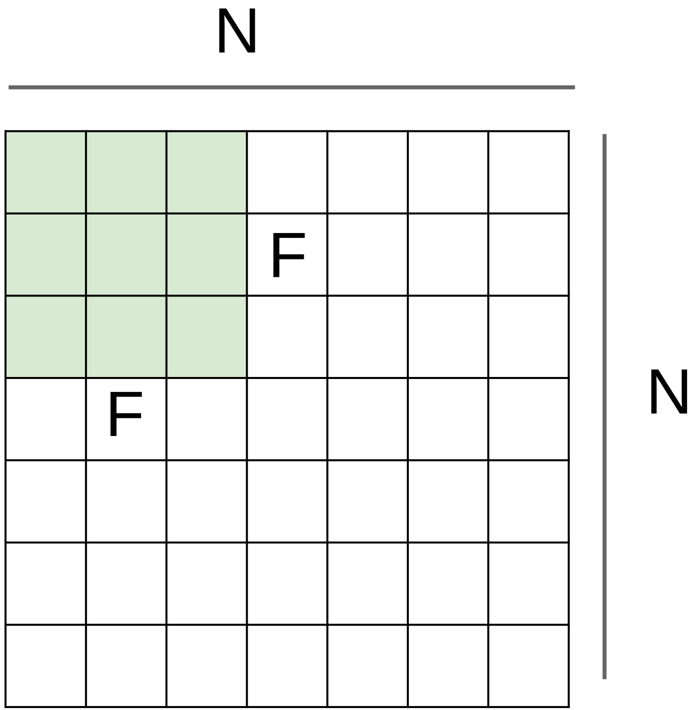
7x7 input (spatially)
assume 3x3 filter
applied **with stride 3?**

A closer look at spatial dimensions:



7x7 input (spatially)
assume 3x3 filter
applied **with stride 3?**

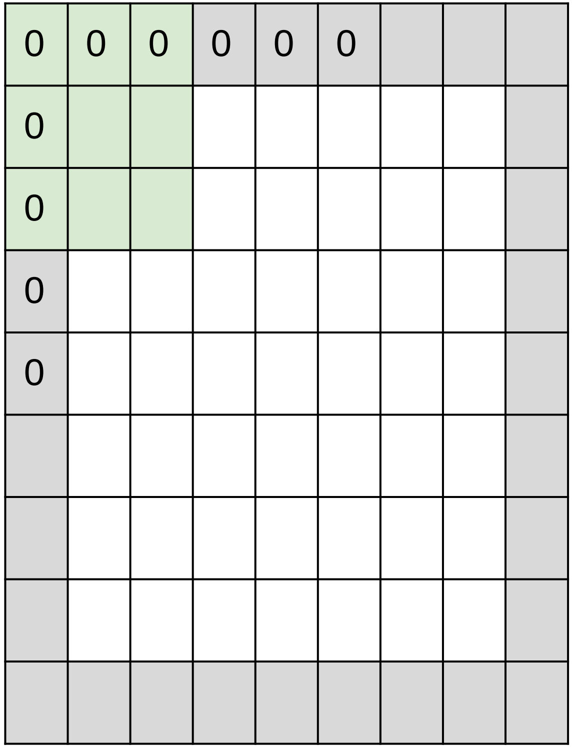
doesn't fit!
cannot apply 3x3 filter on
7x7 input with stride 3.



Output size:
(N - F) / stride + 1

e.g. N = 7, F = 3:
stride 1 => (7 - 3)/1 + 1 = 5
stride 2 => (7 - 3)/2 + 1 = 3
stride 3 => (7 - 3)/3 + 1 = 2.33 :\

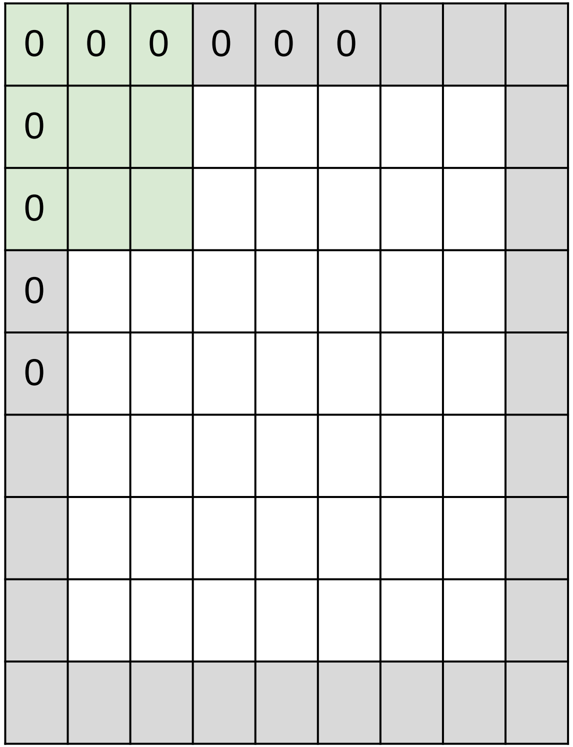
In practice: Common to zero pad the border



e.g. input 7x7
3x3 filter, applied with **stride 1**
pad with 1 pixel border => what is the output?

(recall:)
(N - F) / stride + 1

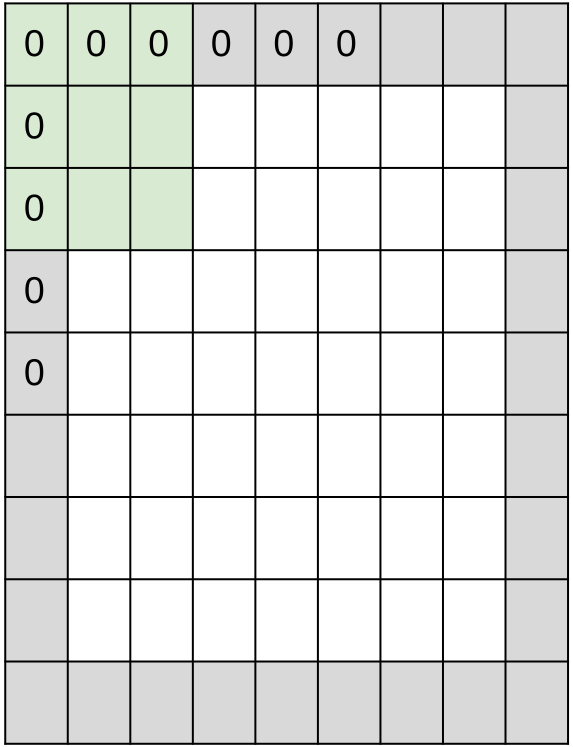
In practice: Common to zero pad the border



e.g. input 7x7
3x3 filter, applied with **stride 1**
pad with 1 pixel border => what is the output?

7x7 output!

In practice: Common to zero pad the border

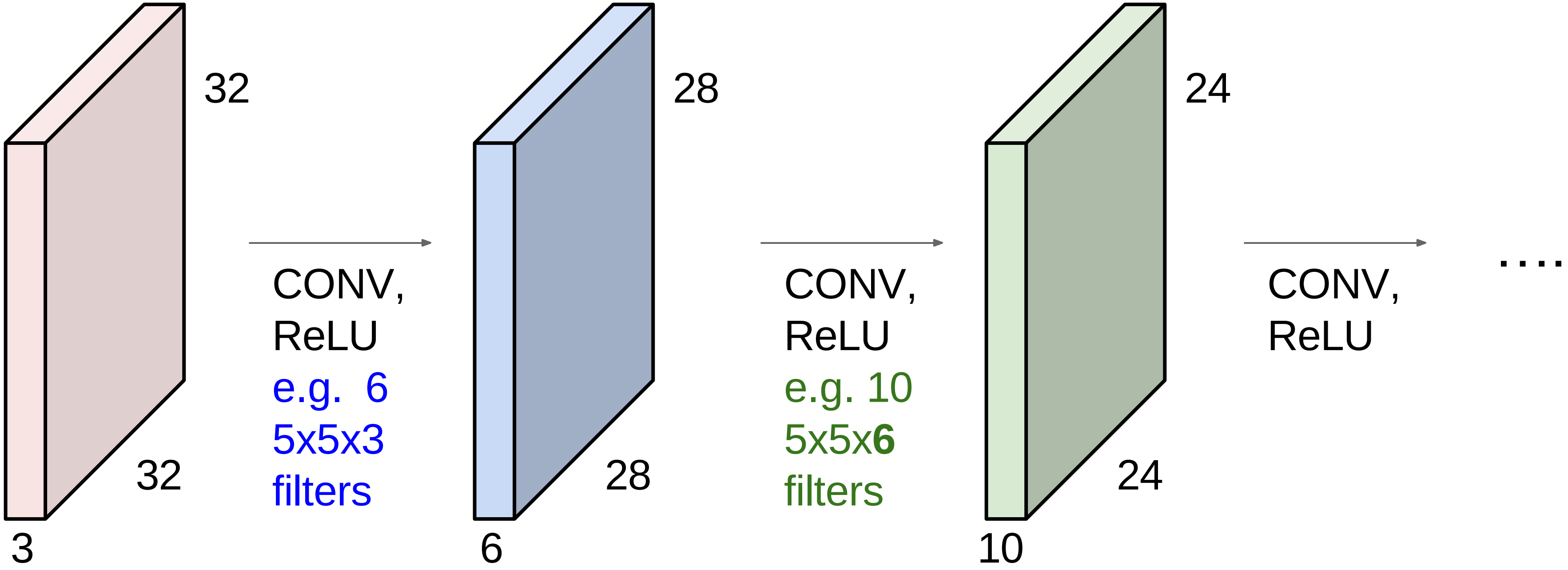


e.g. input 7x7
3x3 filter, applied with **stride 1**
pad with 1 pixel border => what is the output?

7x7 output!
in general, common to see CONV layers with stride 1, filters of size FxF, and zero-padding with (F-1)/2. (will preserve size spatially)
e.g. F = 3 => zero pad with 1
F = 5 => zero pad with 2
F = 7 => zero pad with 3

Remember back to…

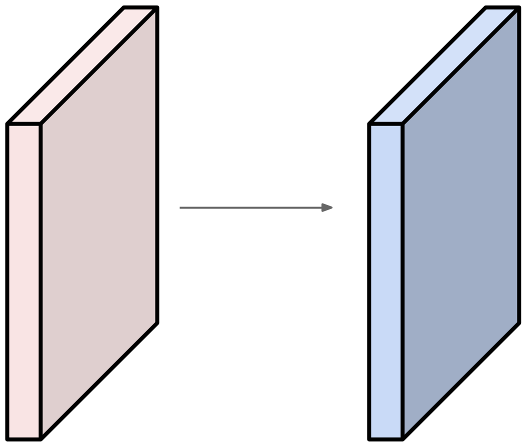
E.g. 32x32 input convolved repeatedly with 5x5 filters shrinks volumes spatially!
(32 -> 28 -> 24 ...). Shrinking too fast is not good, doesn’t work well.



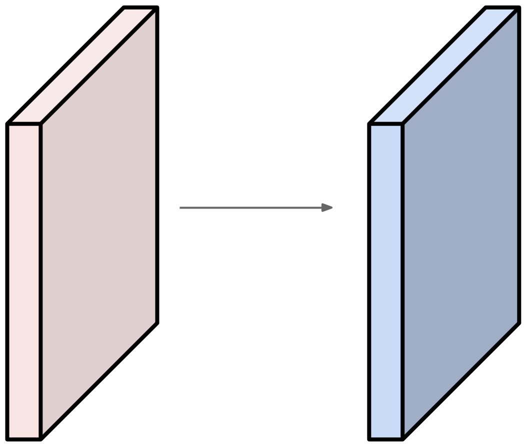
Examples time:

Input volume: **32x32x3**
10 5x5 filters with stride 1, pad 2

Output volume size: ?



Examples time:



Input volume: **32x32x3**

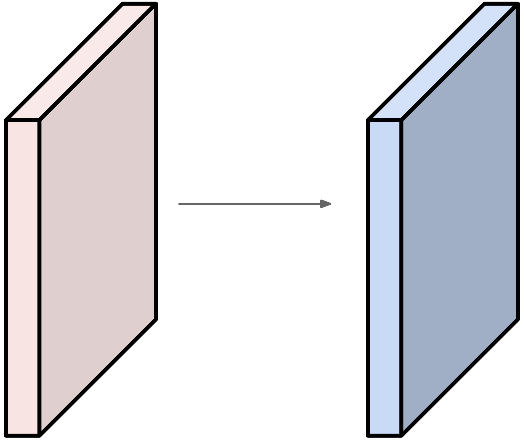
10 5x5 filters with stride 1, pad 2

Output volume size:

(32+2*2-5)/1+1 = 32 spatially, so

32x32x10

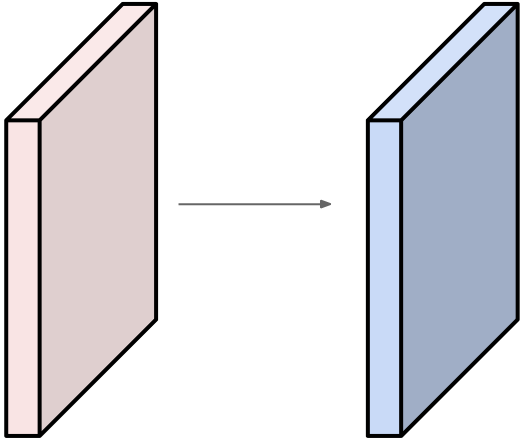
Examples time:



Input volume: **32x32x3**
10 5x5 filters with stride 1, pad 2

Number of parameters in this layer?

Examples time:



Input volume: **32x32x3**
10 5x5 filters with stride 1, pad 2

Number of parameters in this layer?
each filter has 5*5*3 + 1 = 76 params (+1 for bias)
=> 76*10 = **760**

Summary. To summarize, the Conv Layer:

- Accepts a volume of size $W_1 \times H_1 \times D_1$
- Requires four hyperparameters:
  - Number of filters K,
  - their spatial extent F,
  - the stride S,
  - the amount of zero padding P.
- Produces a volume of size $W_2 \times H_2 \times D_2$ where:
  - $W_2 = (W_1 - F + 2P)/S + 1$
  - $H_2 = (H_1 - F + 2P)/S + 1$ (i.e. width and height are computed equally by symmetry)
  - $D_2 = K$
- With parameter sharing, it introduces $F \cdot F \cdot D_1$ weights per filter, for a total of $(F \cdot F \cdot D_1) \cdot K$ weights and K biases.
- In the output volume, the d-th depth slice (of size $W_2 \times H_2$) is the result of performing a valid convolution of the d-th filter over the input volume with a stride of S, and then offset by d-th bias.

Common settings:

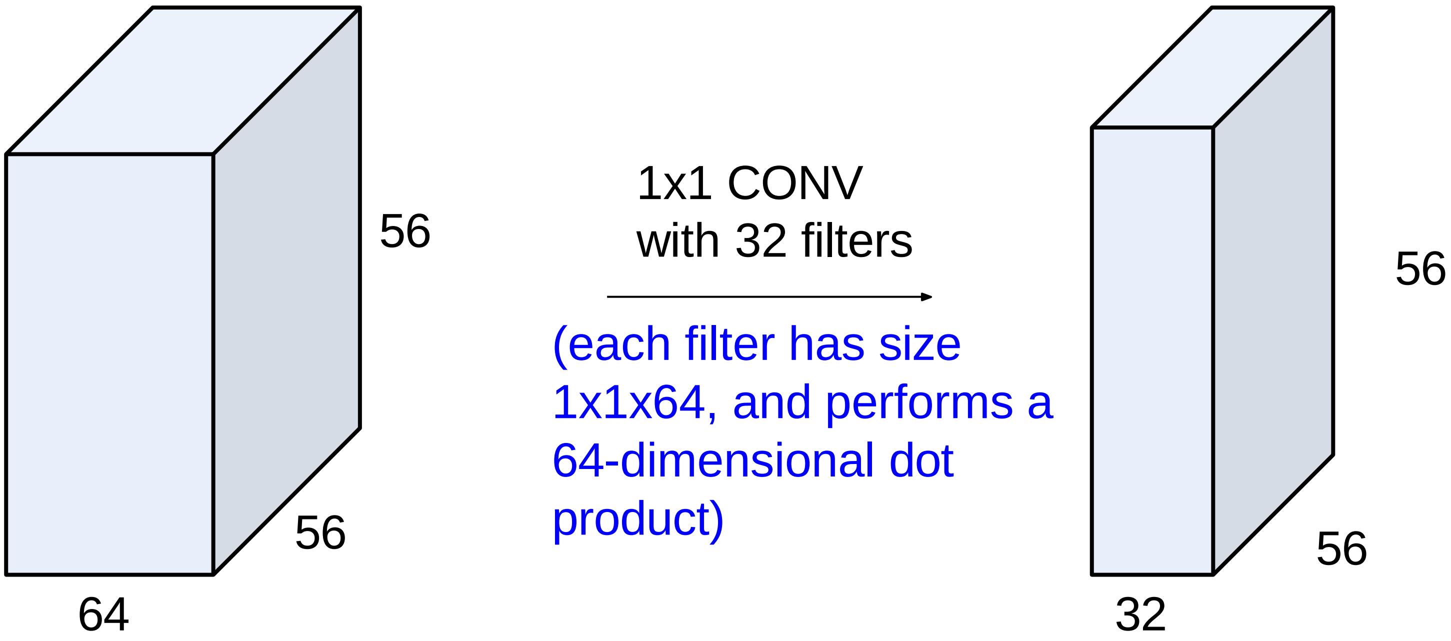
K = (powers of 2, e.g. 32, 64, 128, 512)

- F = 3, S = 1, P = 1
- F = 5, S = 1, P = 2
- F = 5, S = 2, P = ? (whatever fits)
- F = 1, S = 1, P = 0

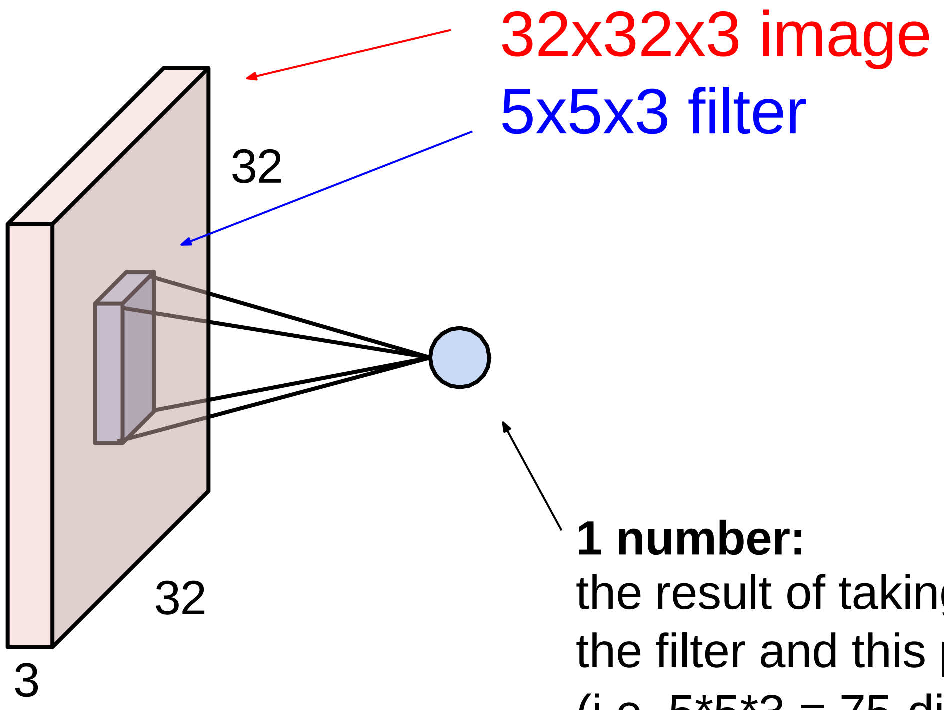
Summary. To summarize, the Conv Layer:

- Accepts a volume of size $W_1 \times H_1 \times D_1$
- Requires four hyperparameters:
  - Number of filters K,
  - their spatial extent F,
  - the stride S,
  - the amount of zero padding P.
- Produces a volume of size $W_2 \times H_2 \times D_2$ where:
  - $W_2 = (W_1 - F + 2P)/S + 1$
  - $H_2 = (H_1 - F + 2P)/S + 1$ (i.e. width and height are computed equally by symmetry)
  - $D_2 = K$
- With parameter sharing, it introduces $F \cdot F \cdot D_1$ weights per filter, for a total of $(F \cdot F \cdot D_1) \cdot K$ weights and K biases.
- In the output volume, the d-th depth slice (of size $W_2 \times H_2$) is the result of performing a valid convolution of the d-th filter over the input volume with a stride of S, and then offset by d-th bias.

(btw, 1x1 convolution layers make perfect sense)



The brain/neuron view of CONV Layer

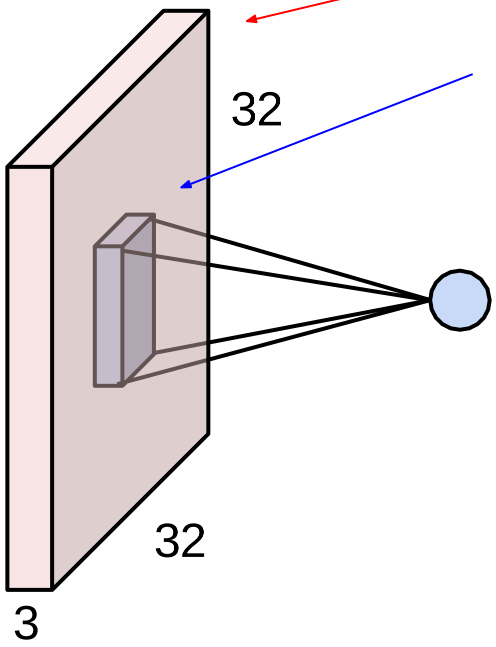


1 number:
the result of taking a dot product between the filter and this part of the image
(i.e. 5*5*3 = 75-dimensional dot product)

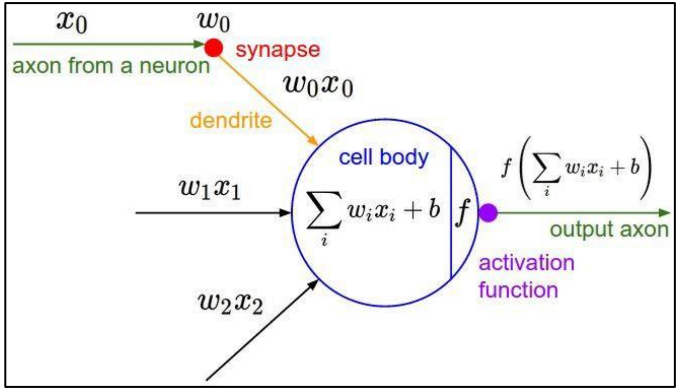
The brain/neuron view of CONV Layer

32x32x3 image

5x5x3 filter

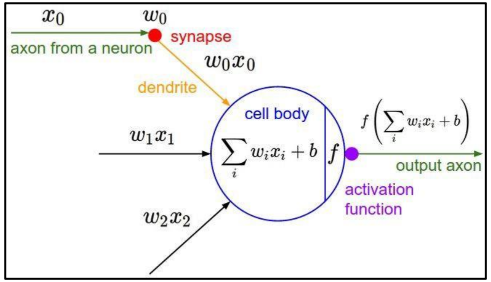


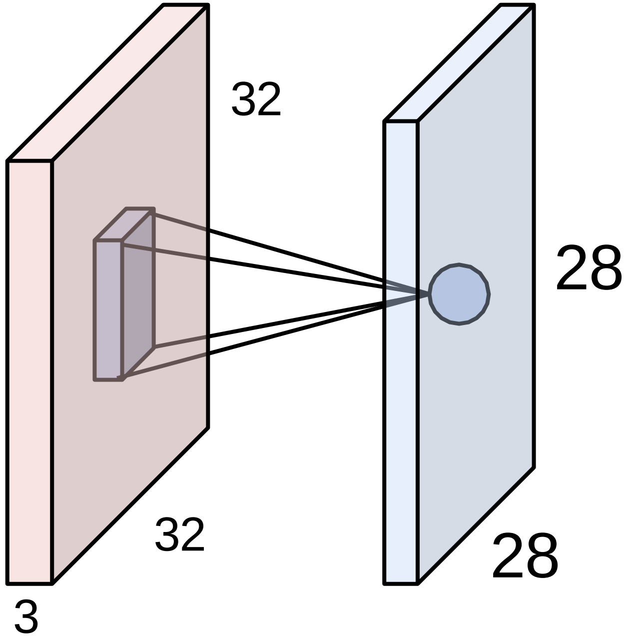
1 number:
the result of taking a dot product between the filter and this part of the image
(i.e. 5*5*3 = 75-dimensional dot product)



It's just a neuron with local connectivity...

The brain/neuron view of CONV Layer



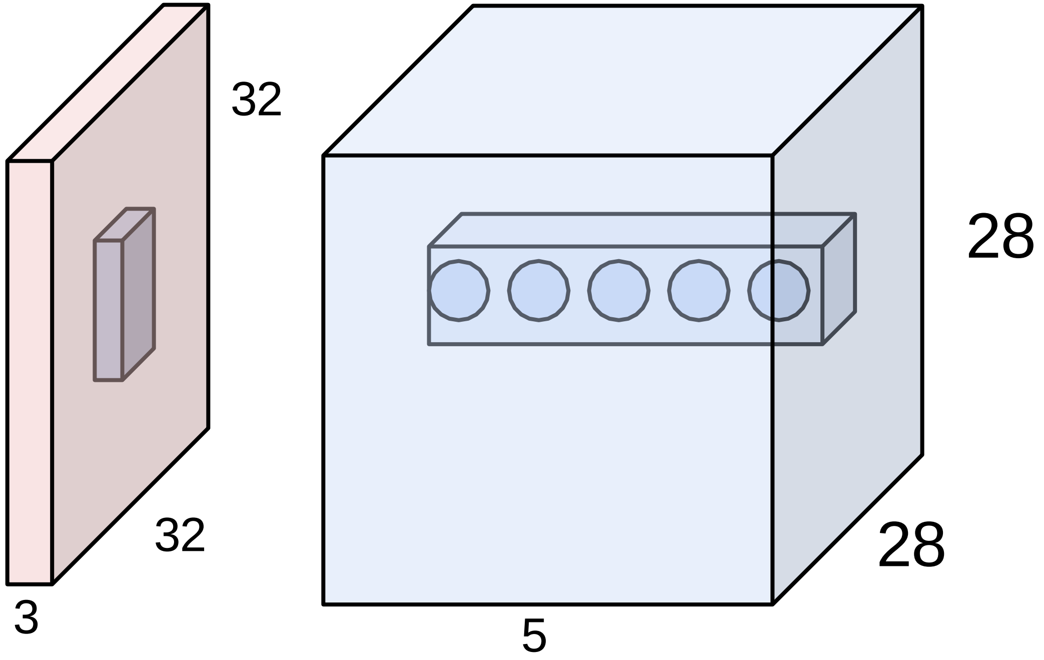


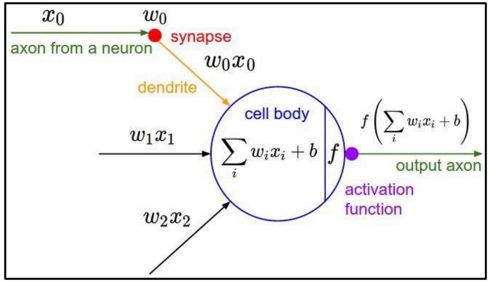
An activation map is a 28x28 sheet of neuron outputs:

1. Each is connected to a small region in the input
2. All of them share parameters

"5x5 filter" -> "5x5 receptive field for each neuron"

The brain/neuron view of CONV Layer





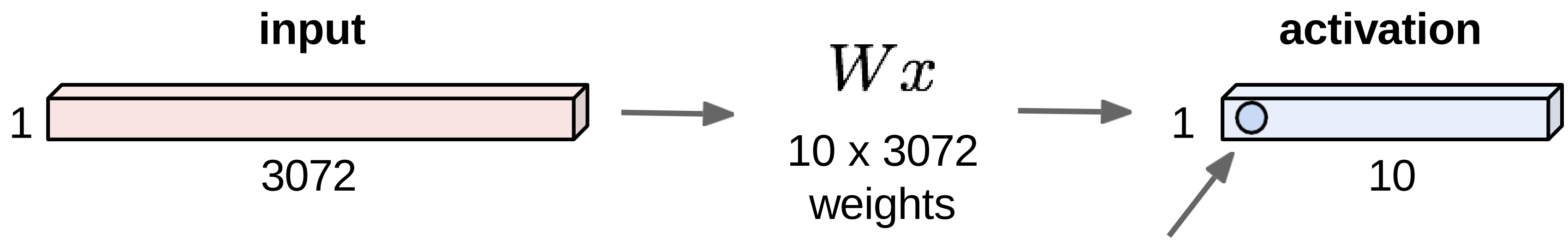
E.g. with 5 filters,
CONV layer consists of neurons arranged in a 3D grid (28x28x5)

There will be 5 different neurons all looking at the same region in the input volume

Reminder: Fully Connected Layer

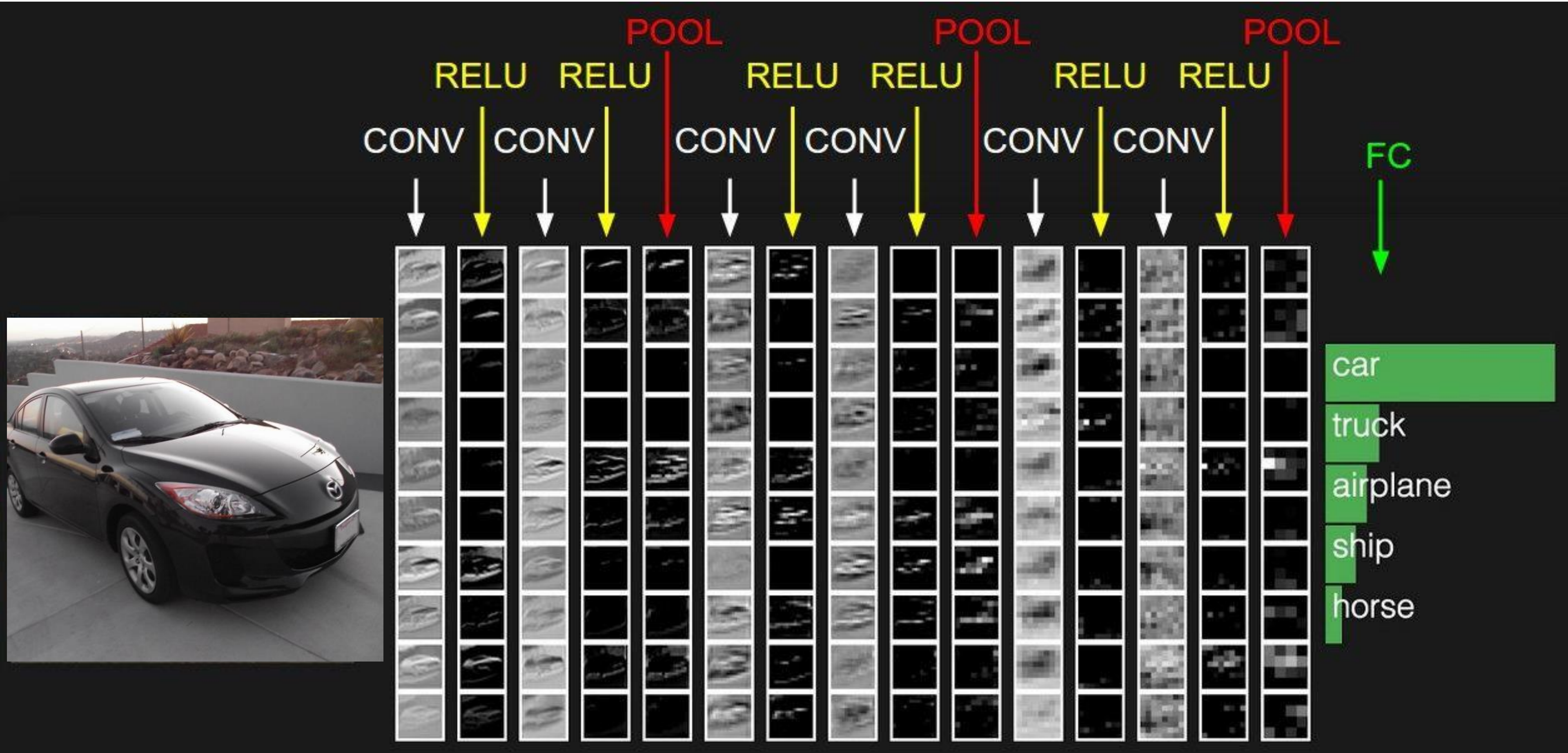
32x32x3 image -> stretch to 3072 x 1

Each neuron looks at the full input volume



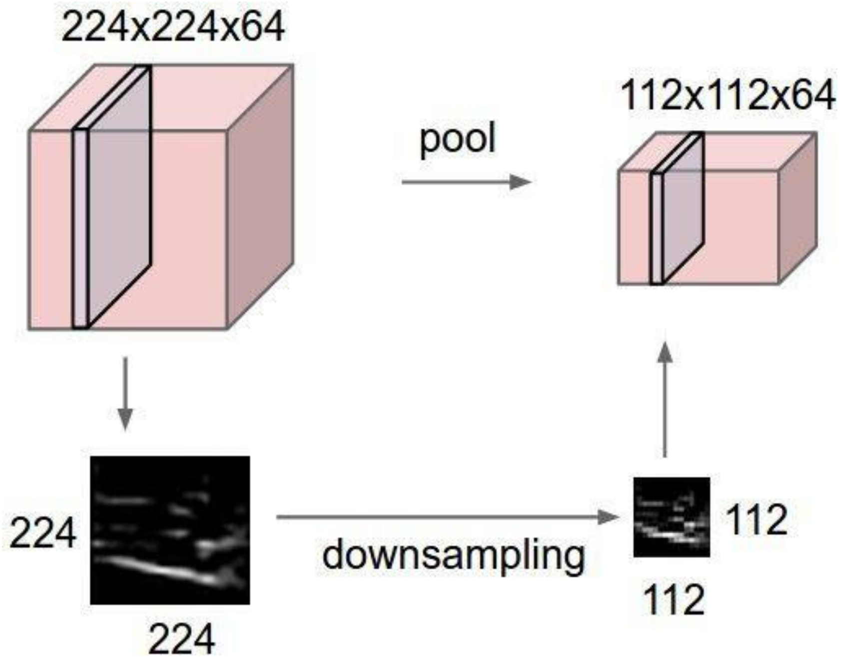
1 number:
the result of taking a dot product between a row of W and the input (a 3072-dimensional dot product)

two more layers to go: POOL/FC

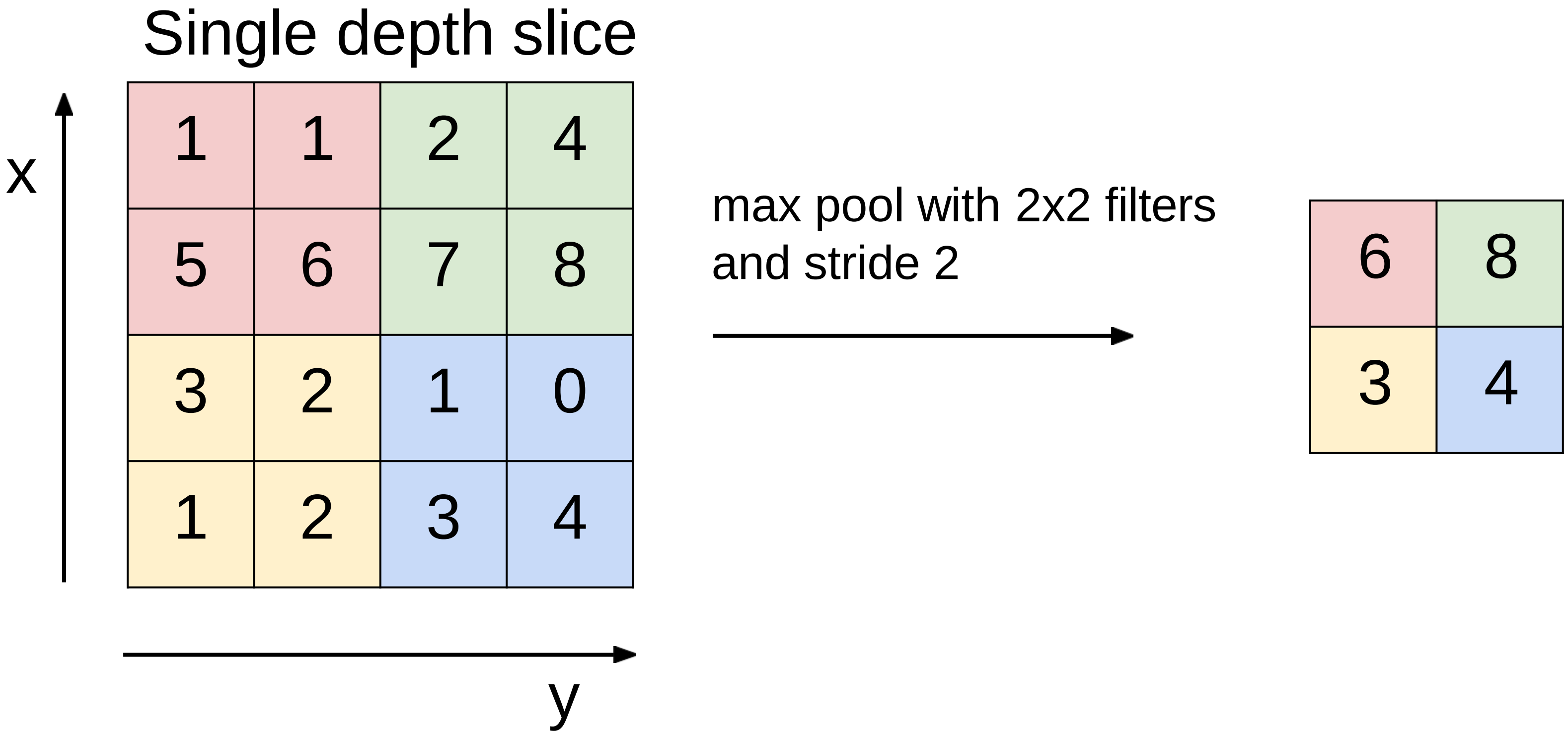


Pooling layer

- makes the representations smaller and more manageable
- operates over each activation map independently:



MAX POOLING


Single depth slice
x
1 1 2 4
5 6 7 8
3 2 1 0
1 2 3 4
y
max pool with 2x2 filters
and stride 2
6 8
3 4

- Accepts a volume of size $W_1 \times H_1 \times D_1$
- Requires three hyperparameters:
  - their spatial extent F,
  - the stride S,
- Produces a volume of size $W_2 \times H_2 \times D_2$ where:
  - $W_2 = (W_1 - F)/S + 1$
  - $H_2 = (H_1 - F)/S + 1$
  - $D_2 = D_1$
- Introduces zero parameters since it computes a fixed function of the input
- Note that it is not common to use zero-padding for Pooling layers

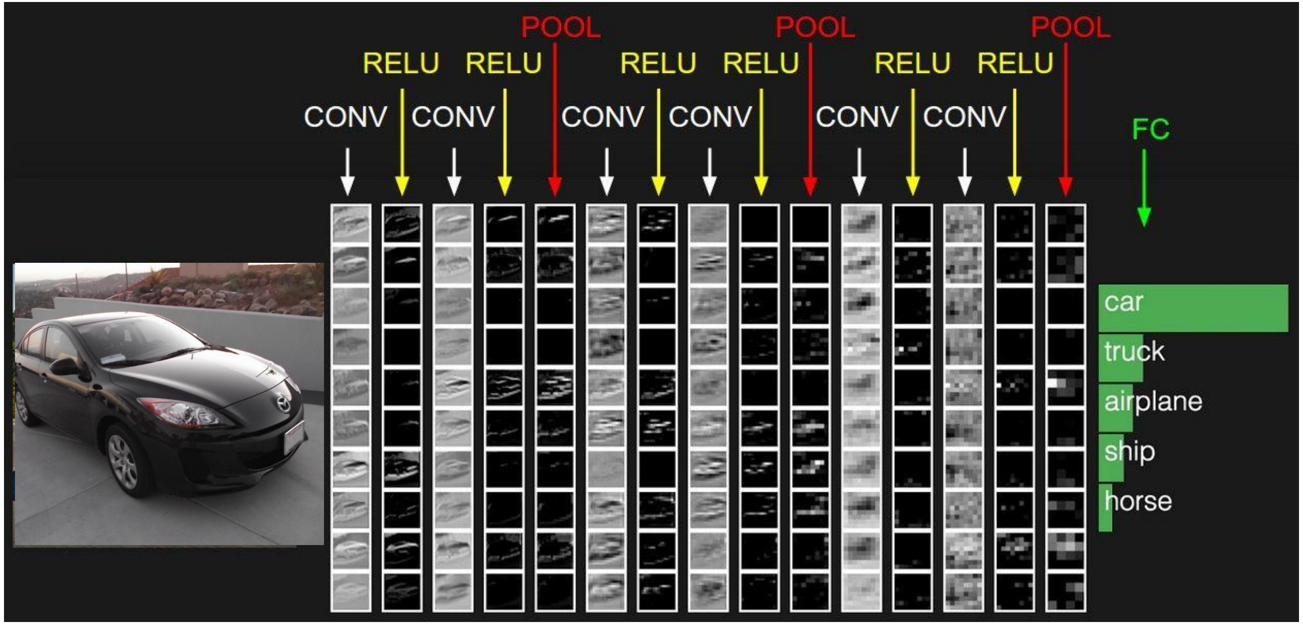
- Accepts a volume of size $W_1 \times H_1 \times D_1$
- Requires three hyperparameters:
  - their spatial extent F,
  - the stride S,
- Produces a volume of size $W_2 \times H_2 \times D_2$ where:
  - $W_2 = (W_1 - F)/S + 1$
  - $H_2 = (H_1 - F)/S + 1$
  - $D_2 = D_1$
- Introduces zero parameters since it computes a fixed function of the input
- Note that it is not common to use zero-padding for Pooling layers

Common settings:

F = 2, S = 2
F = 3, S = 2

Fully Connected Layer (FC layer)

- Contains neurons that connect to the entire input volume, as in ordinary Neural Networks



[ConvNetJS demo: training on CIFAR-10]

ConvNetJS CIFAR-10 demo

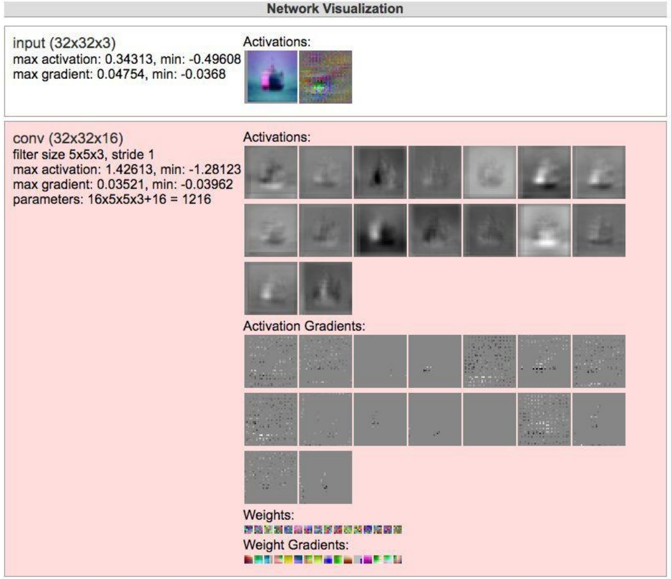
Description

This demo trains a Convolutional Neural Network on the CIFAR-10 dataset in your browser, with nothing but Javascript. The state of the art on this dataset is about 90% accuracy and human performance is at about 94% (not perfect as the dataset can be a bit ambiguous). I used this python script to parse the original files (python version) into batches of images that can be easily loaded into page DOM with img tags.

This dataset is more difficult and it takes longer to train a network. Data augmentation includes random flipping and random image shifts by up to 2px horizontally and verically.

By default, in this demo we're using Adadelta which is one of per-parameter adaptive step size methods, so we don't have to worry about changing learning rates or momentum over time. However, I still included the text fields for changing these if you'd like to play around with SGD+Momentum trainer.

Report questions/bugs/suggestions to @karpathy.



http://cs.stanford.edu/people/karpathy/convnetjs/demo/cifar10.html

Recap and Summary

- Convolutional Neural Networks (ConvNets) stack
  - Convolutional layers
  - Pooling layers
  - Fully connected (FC) layers

- Trend towards smaller filters and deeper architectures

- Typical architecture:
  [(Conv→Activation)*N → Pool]*M → (FC→Activation)*F, Softmax

Convolutional Neural Networks

Questions?