Convolutional Neural Nets II

EECS 442 – David Fouhey

Fall 2019, University of Michigan

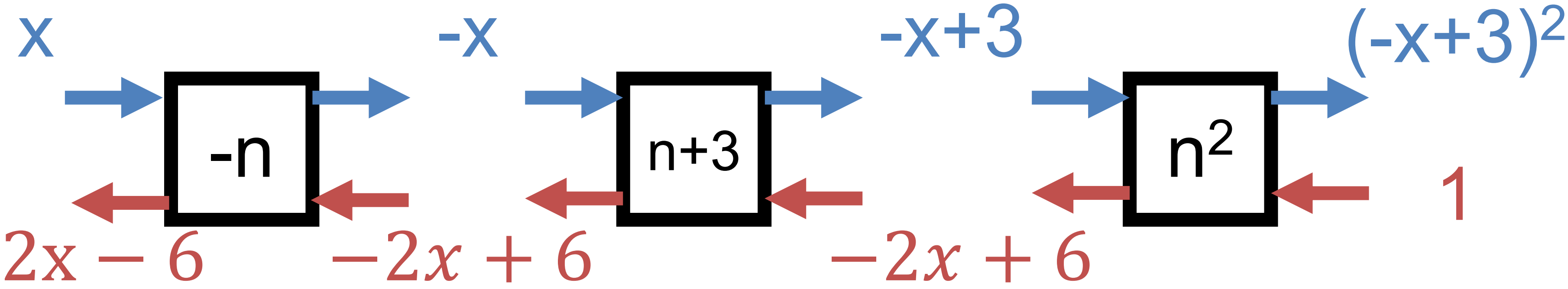
http://web.eecs.umich.edu/~fouhey/teaching/EECS442_F19/

Administrivia

- If you need a team-mate for your project, email me. I’m matching people.

Previously – Backpropagation

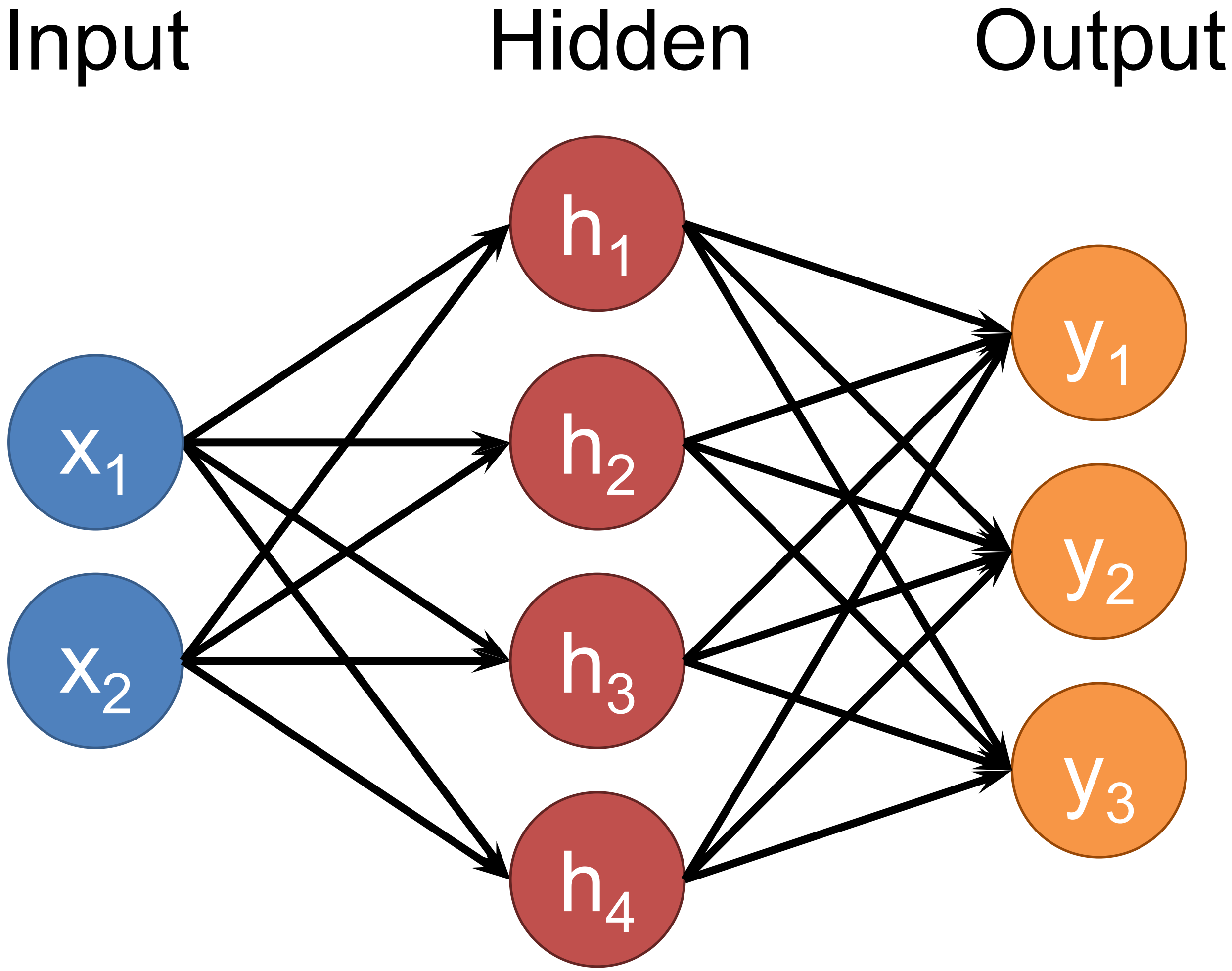
$$f(x) = (-x + 3)^2$$



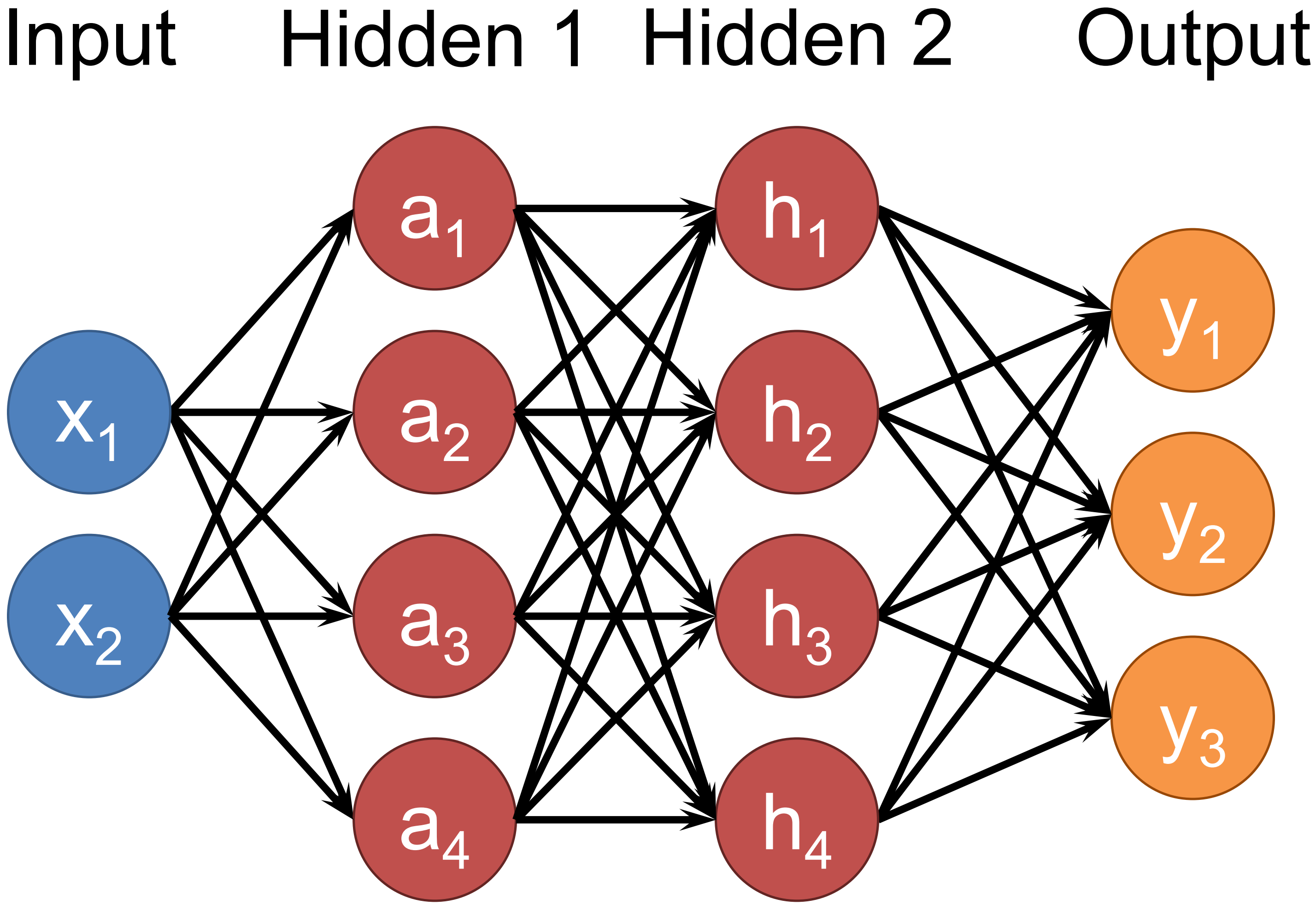
Forward pass: compute function

Backward pass: compute derivative of all parts of the function

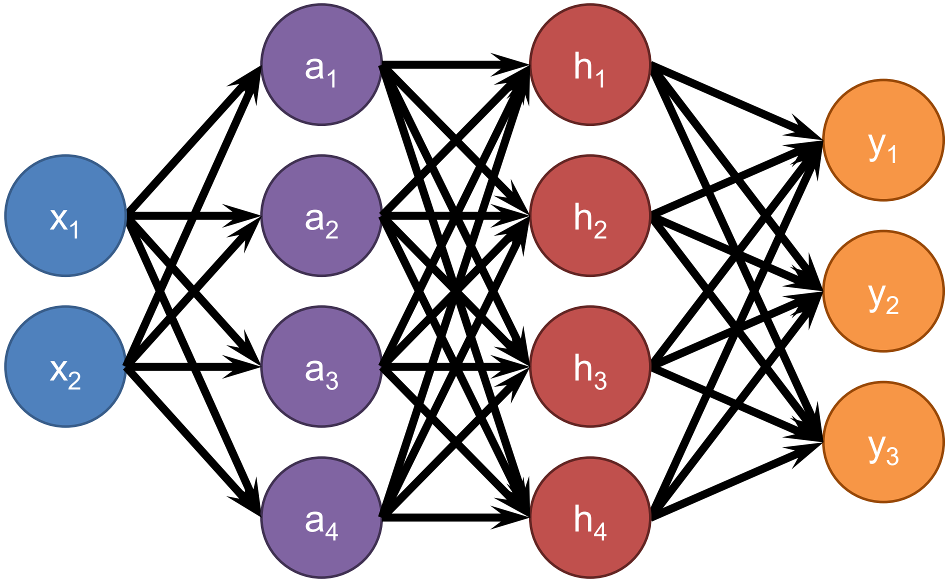
Setting Up A Neural Net



Setting Up A Neural Net



Fully Connected Network

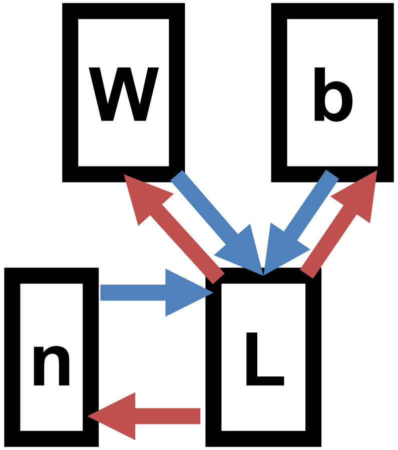


Each neuron connects to each neuron in the previous layer

Fully Connected Network

Define New Block: “Linear Layer”

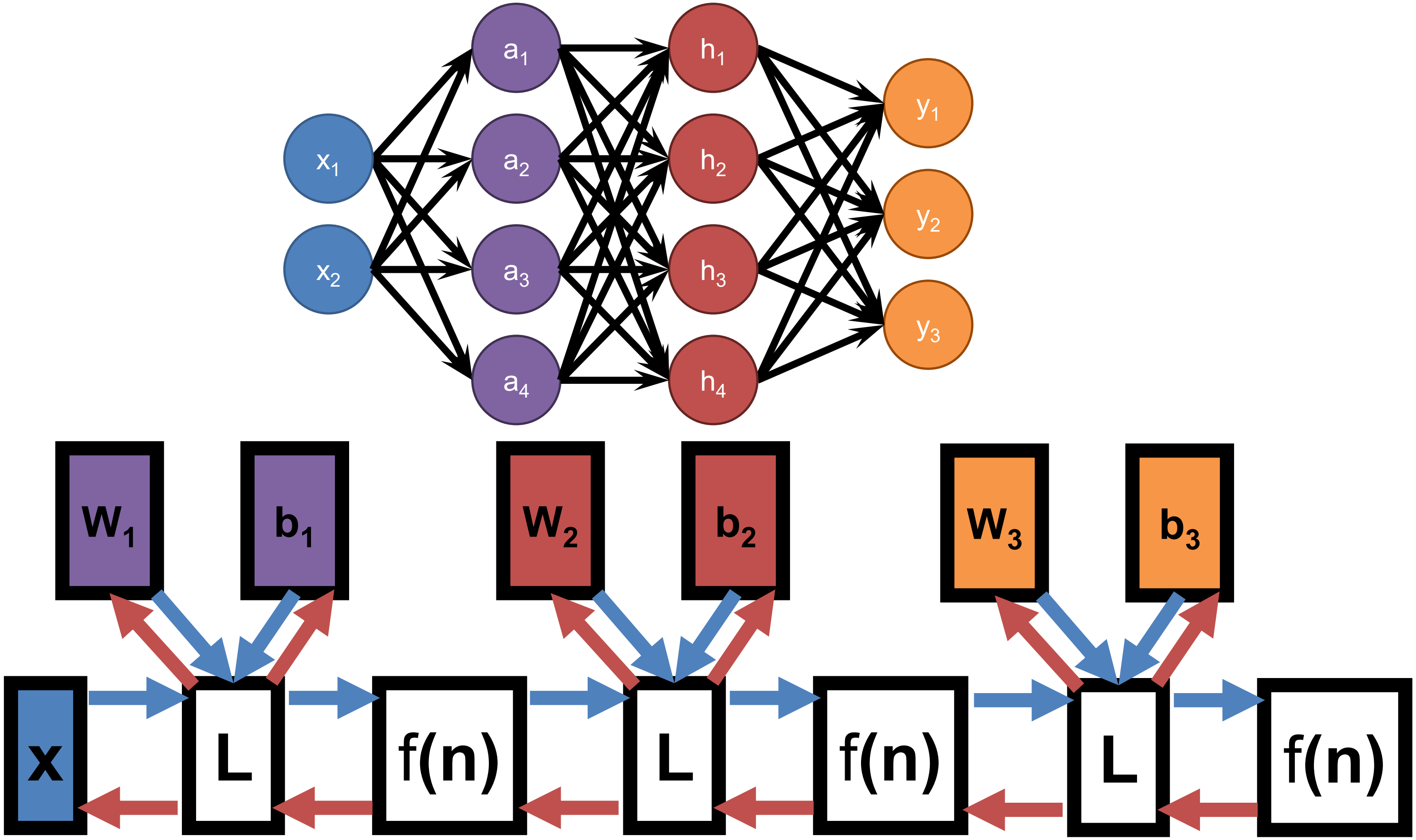
(Ok technically it’s Affine)



$$L(\boldsymbol{n}) = \boldsymbol{W}\boldsymbol{n} + \boldsymbol{b}$$

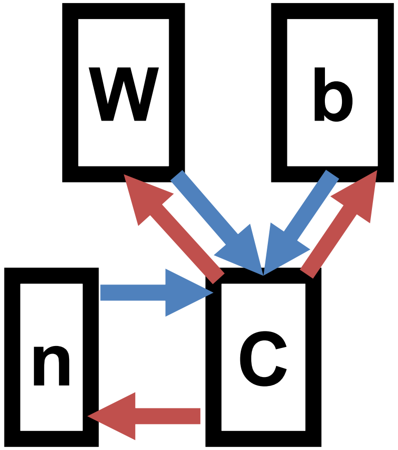
Can get gradient with respect to all the inputs

Fully Connected Network



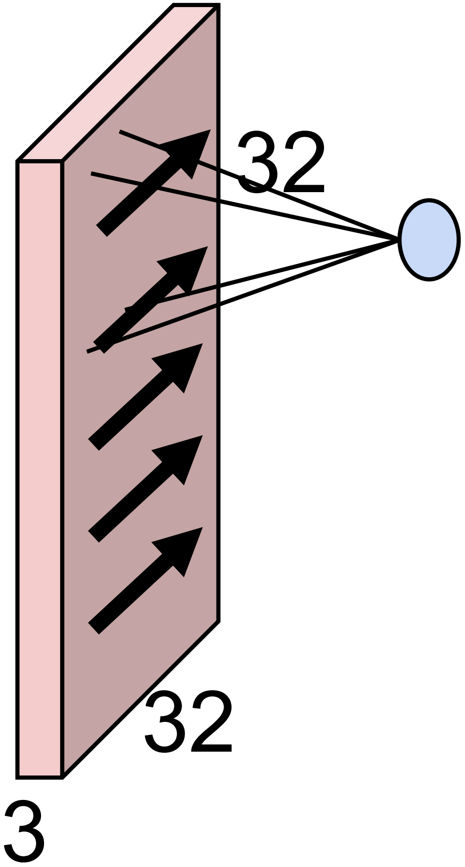
Convolutional Layer

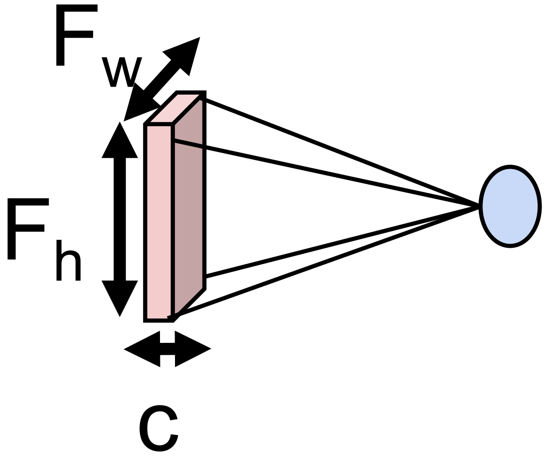
New Block: 2D Convoluiton



$$C(\boldsymbol{n}) = \boldsymbol{n} * \boldsymbol{W} + \boldsymbol{b}$$

Convolution Layer

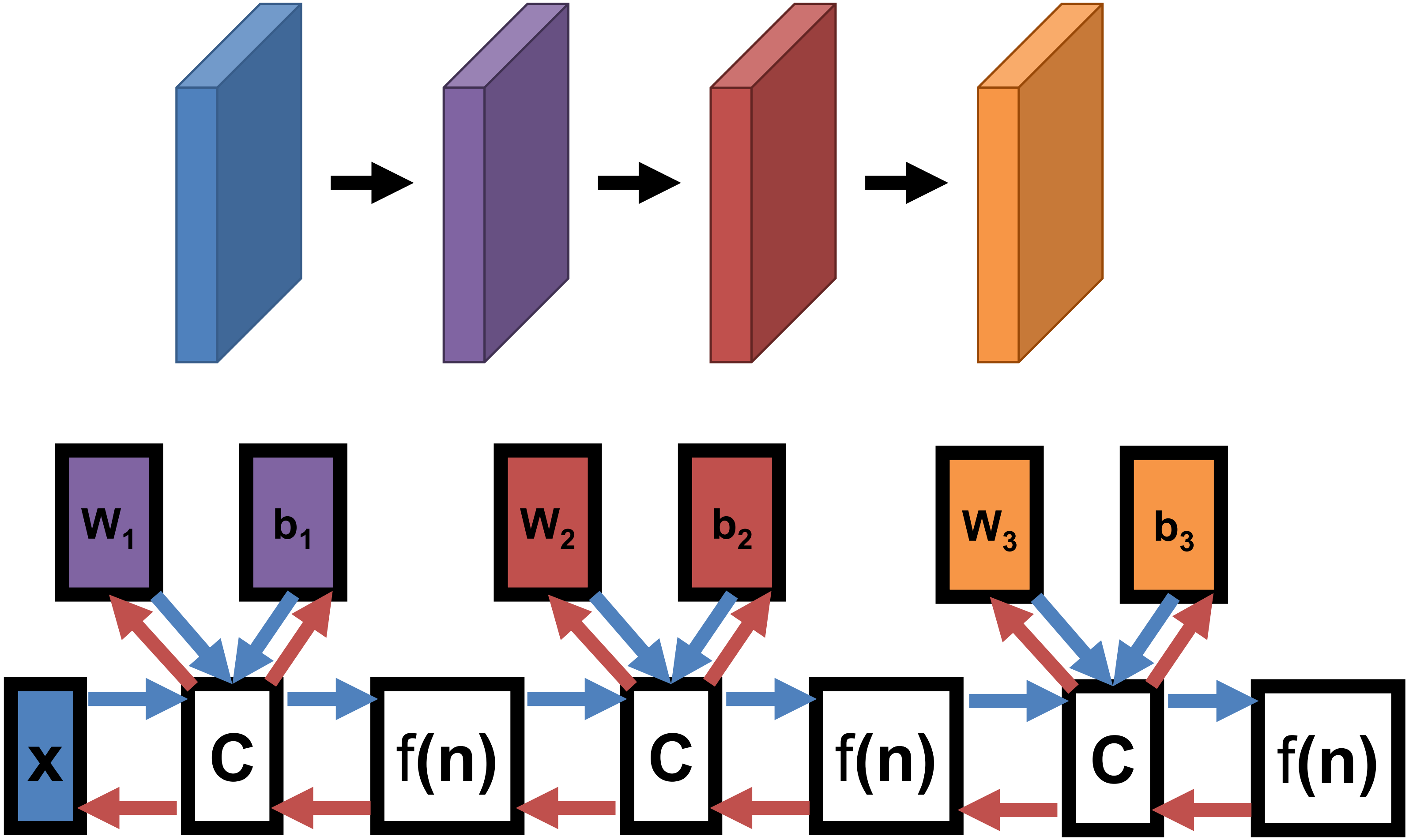




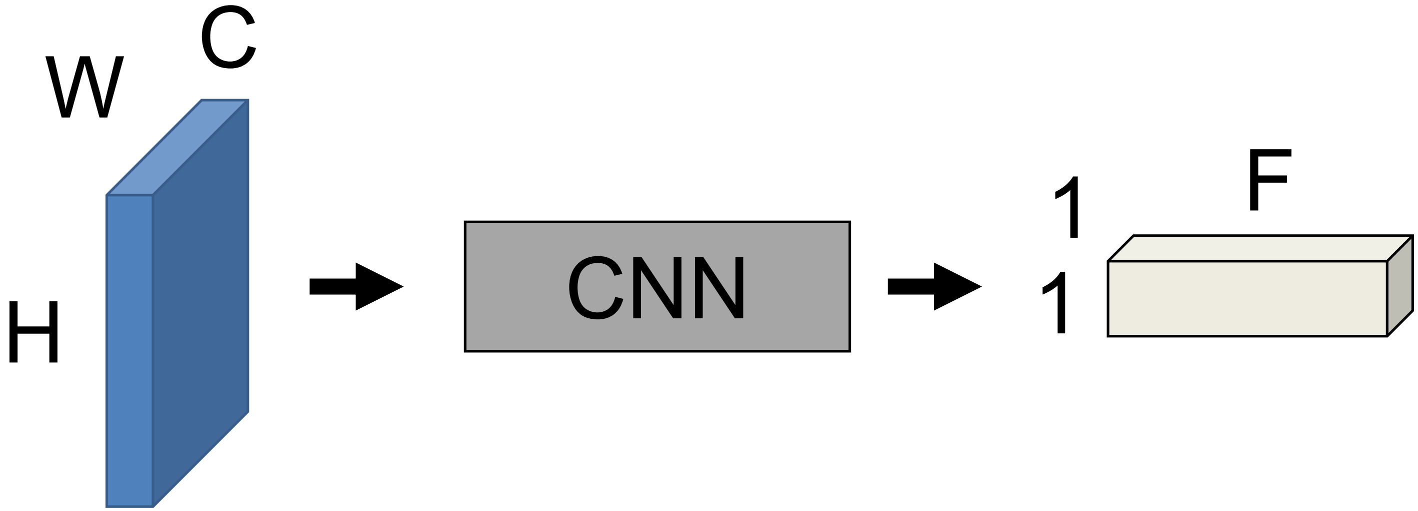
$$b + \sum_{i=1}^{F_h} \sum_{j=1}^{F_w} \sum_{k=1}^{c} F_{i,j,k} * I_{y+i,x+j,c}$$

Slide credit: Karpathy and Fei-Fei

Convolutional Neural Network (CNN)



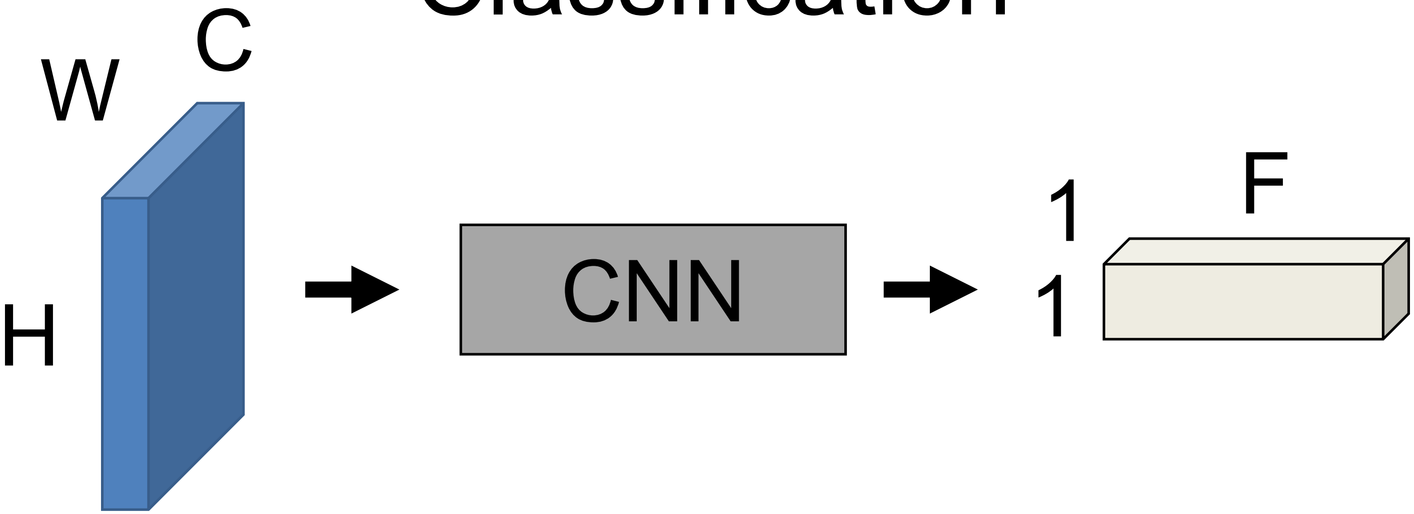
Today



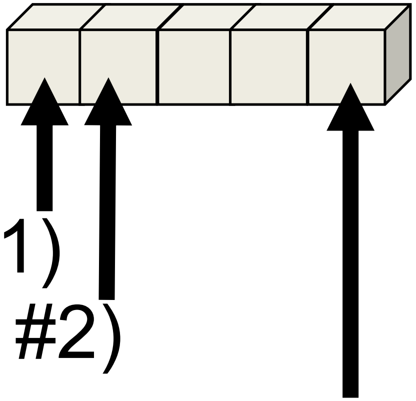
Convert HxW image into a F-dimensional vector

- What's the probability this image is a cat (F=1)
- Which of 1000 categories is this image? (F=1000)
- At what GPS coord was this image taken? (F=2)
- Identify the X,Y coordinates of 28 body joints of an image of a human (F=56)

Today's Running Example: Classification



Running example:
image classification

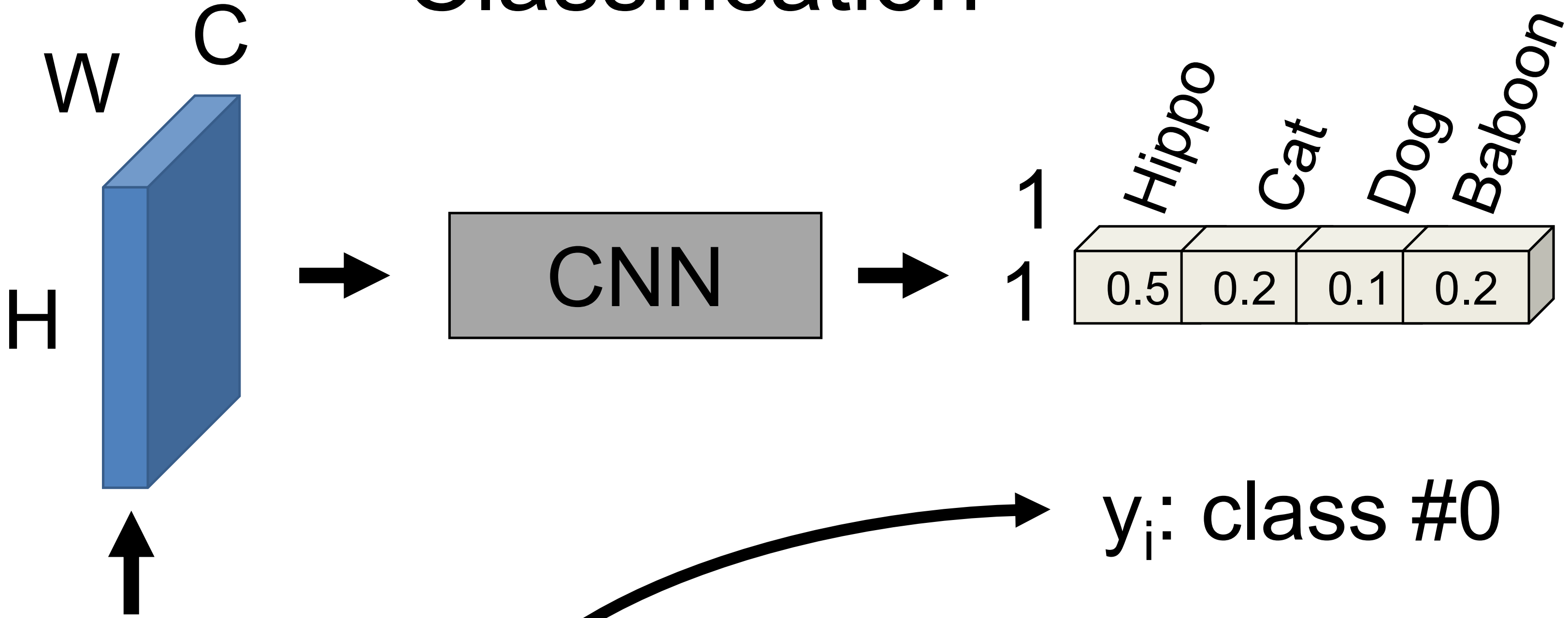


P(image is class #1)

P(image is class #2)

P(image is class #F)

Today's Running Example: Classification

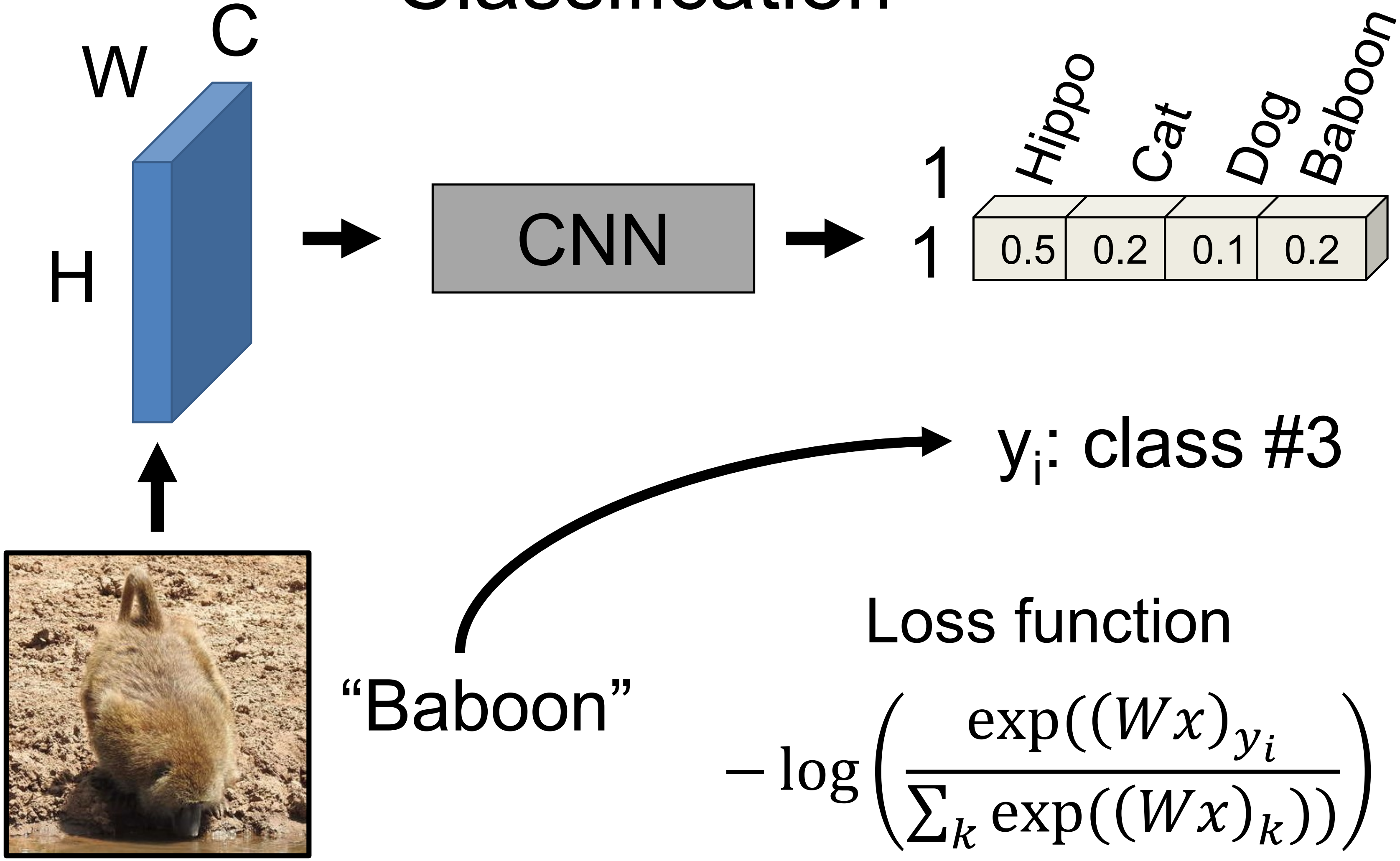


"Hippo"

Loss function

$$-\log\left(\frac{\exp((Wx)_{y_i})}{\sum_k \exp((Wx)_k)}\right)$$

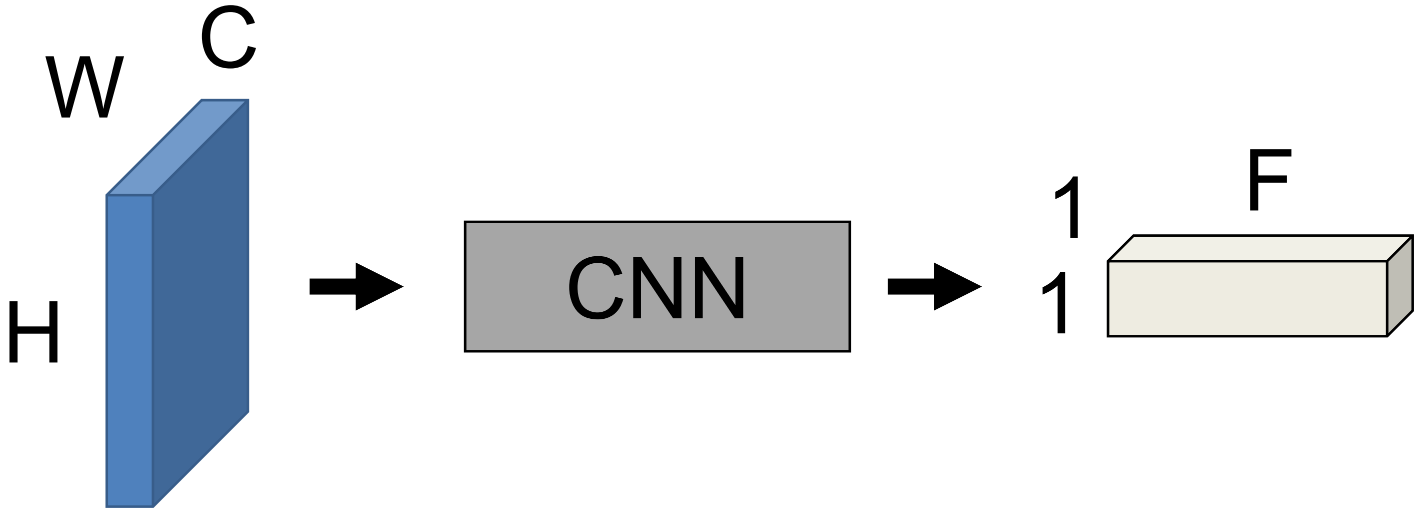
Today's Running Example: Classification



Loss function

$$-\log\left(\frac{\exp((Wx)_{y_i}}{\sum_k \exp((Wx)_k))}\right)$$

Model For Your Head



- Provide:
  - Examples of images and desired outputs
  - Sequence of layers producing a 1x1xF output
  - A *loss* function that measures success
- Train the network -> network figures out the parameters that makes this work

Layer Collection

You can construct functions out of layers. The only requirement is the layers “fit” together. Optimization figures out what the parameters of the layers are.

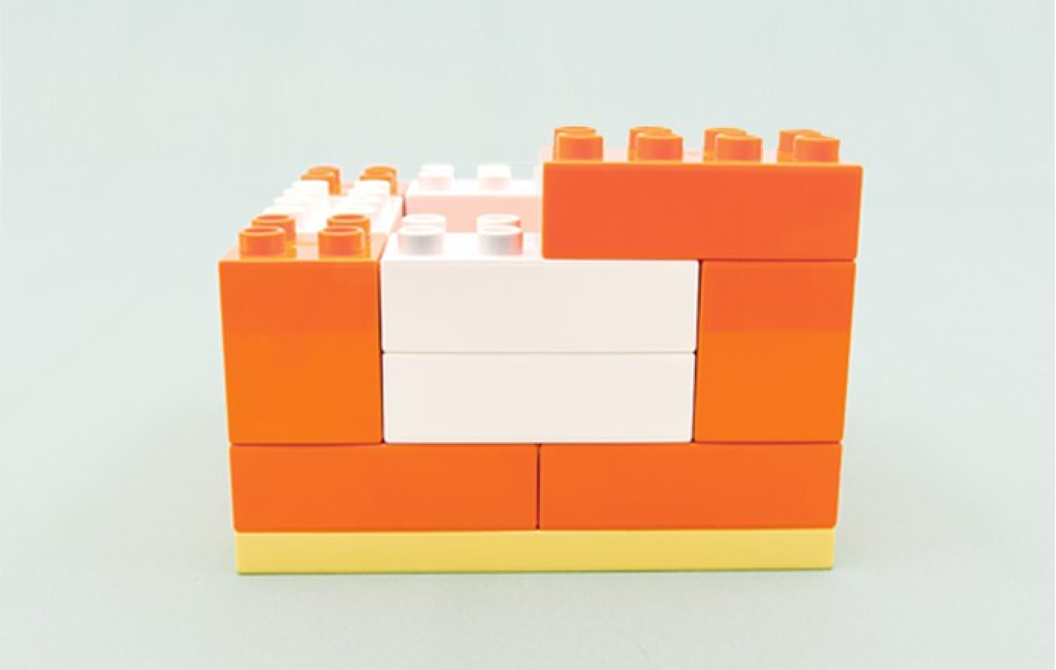
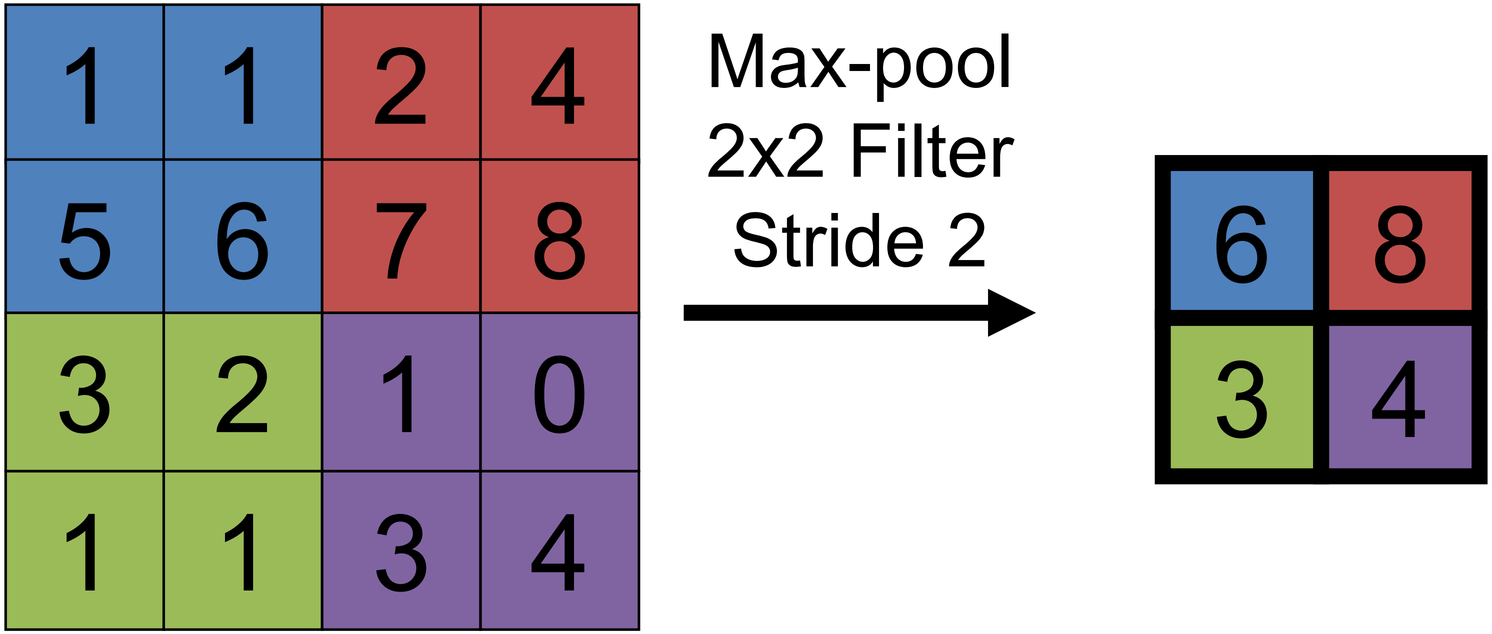


Image credit: lego.com

Review – Pooling

Idea: just want spatial resolution of activations / images smaller; applied per-channel



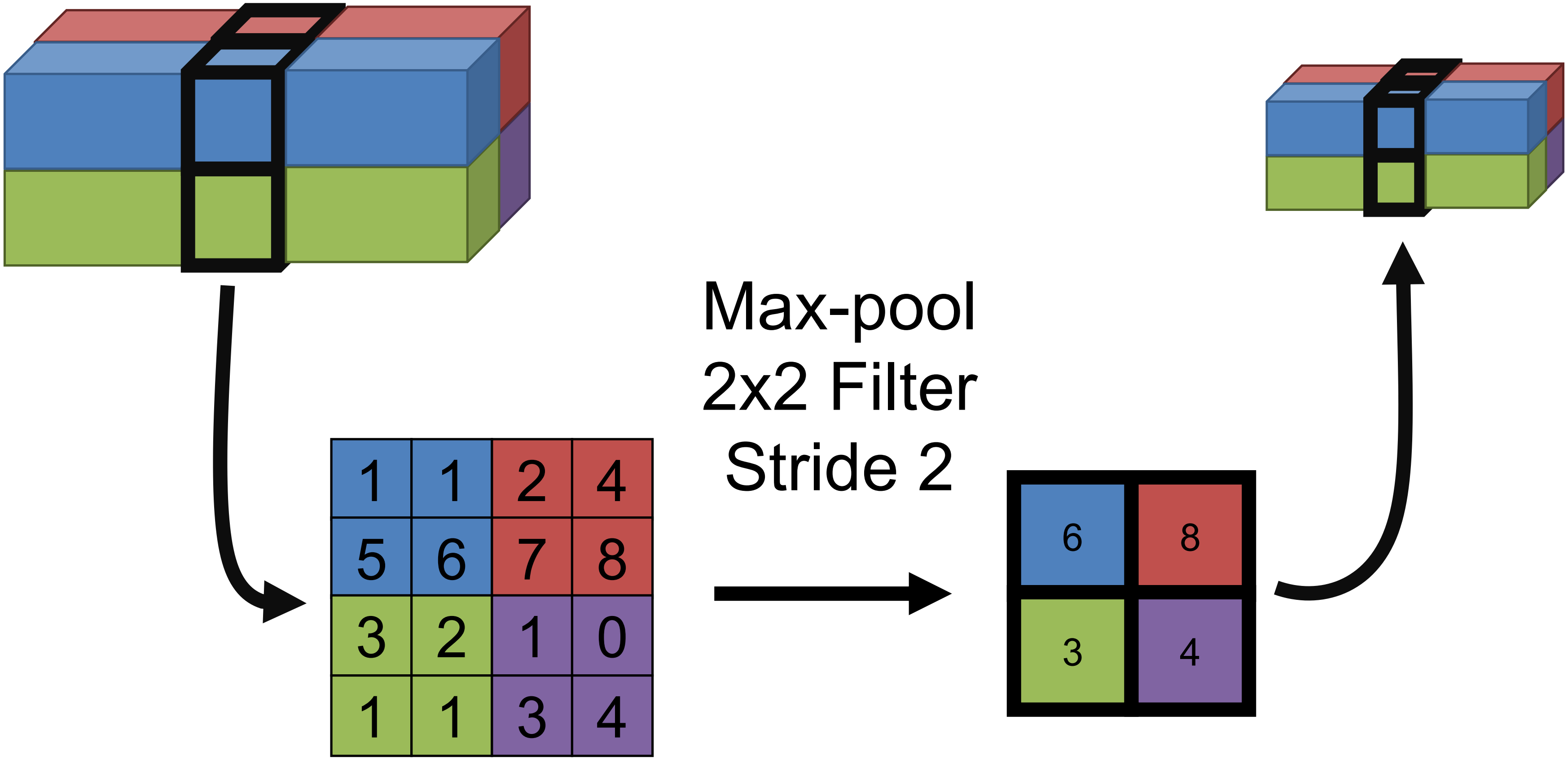
1	1	2	4
5	6	7	8
3	2	1	0
1	1	3	4

Max-pool
2x2 Filter
Stride 2

6	8
3	4

Slide credit: Karpathy and Fei-Fei

Review – Pooling

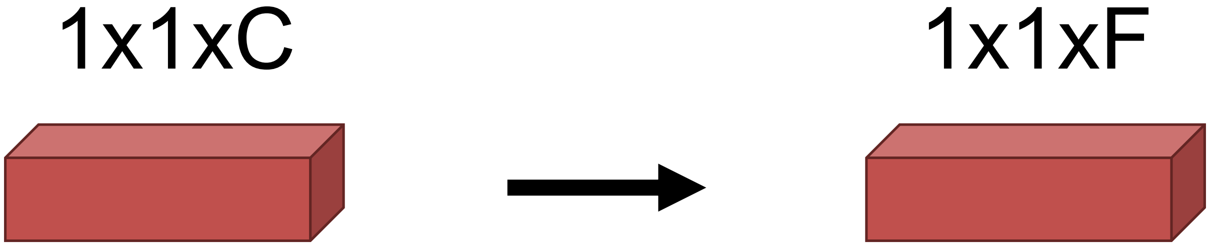


Max-pool
2x2 Filter
Stride 2

1	1	2	4
5	6	7	8
3	2	1	0
1	1	3	4

6	8
3	4

Other Layers – Fully Connected



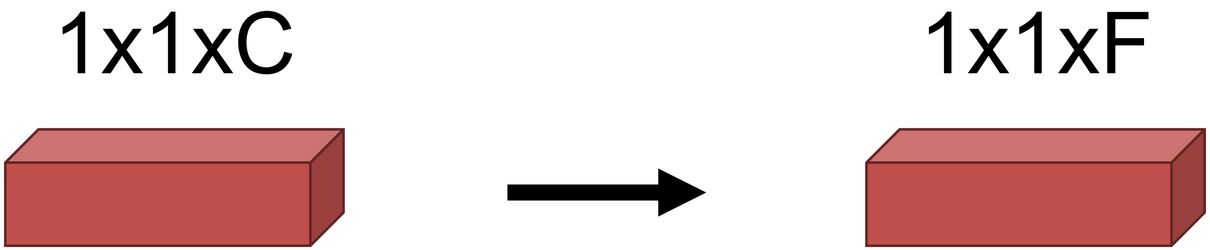
1x1xC

1x1xF

Map C-dimensional feature to F-dimensional feature using linear transformation
W (FxC matrix) + b (Fx1 vector)

How can we write this as a convolution?

Everything's a Convolution

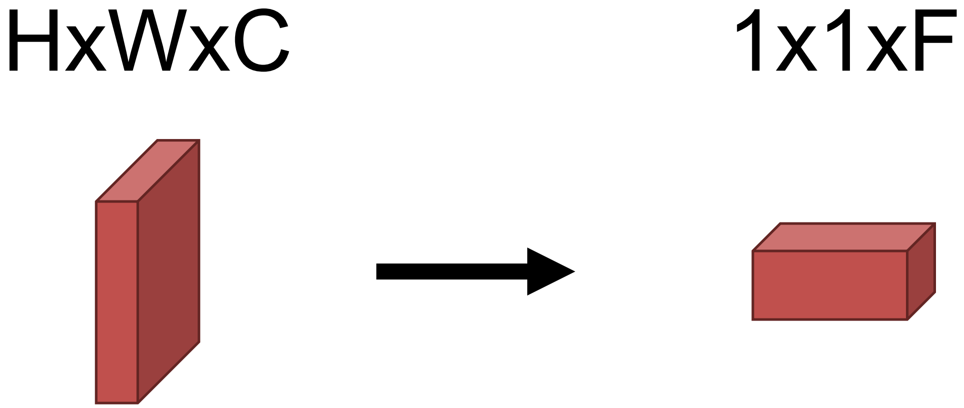


Set Fh=1, Fw=1

1x1 Convolution with F Filters

$$b + \sum_{i=1}^{F_h} \sum_{j=1}^{F_w} \sum_{k=1}^{c} F_{i,j,k} * I_{y+i,x+j,c} \longrightarrow b + \sum_{k=1}^{c} F_k * I_c$$

Converting to a Vector

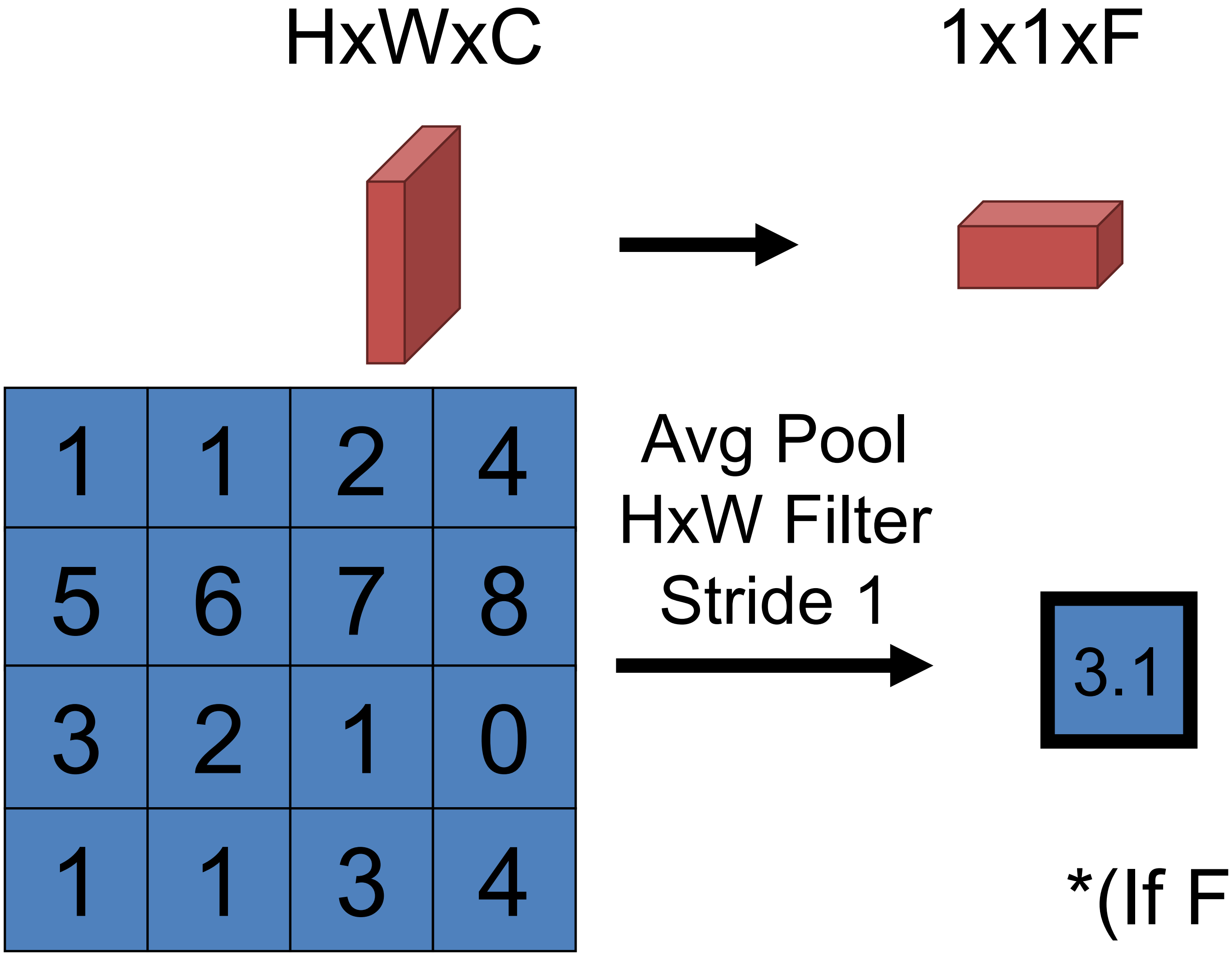


HxWxC

1x1xF

How can we do this?

Converting to a Vector* – Pool



HxWxC

1x1xF

1	1	2	4
5	6	7	8
3	2	1	0
1	1	3	4

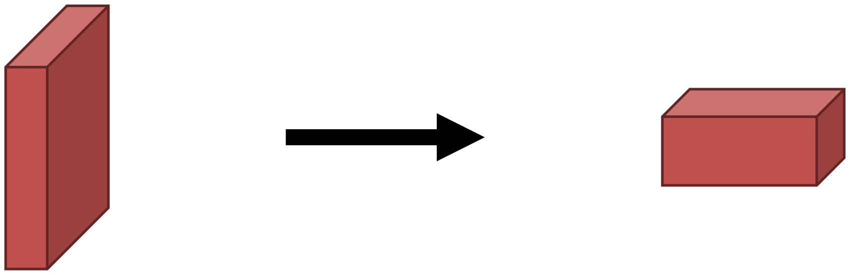
Avg Pool
HxW Filter
Stride 1

3.1

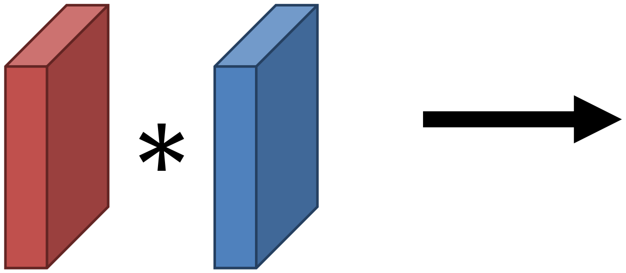
*(If F == C)

Converting to a Vector – Convolve

HxWxC 1x1xF



HxW Convolution with F Filters



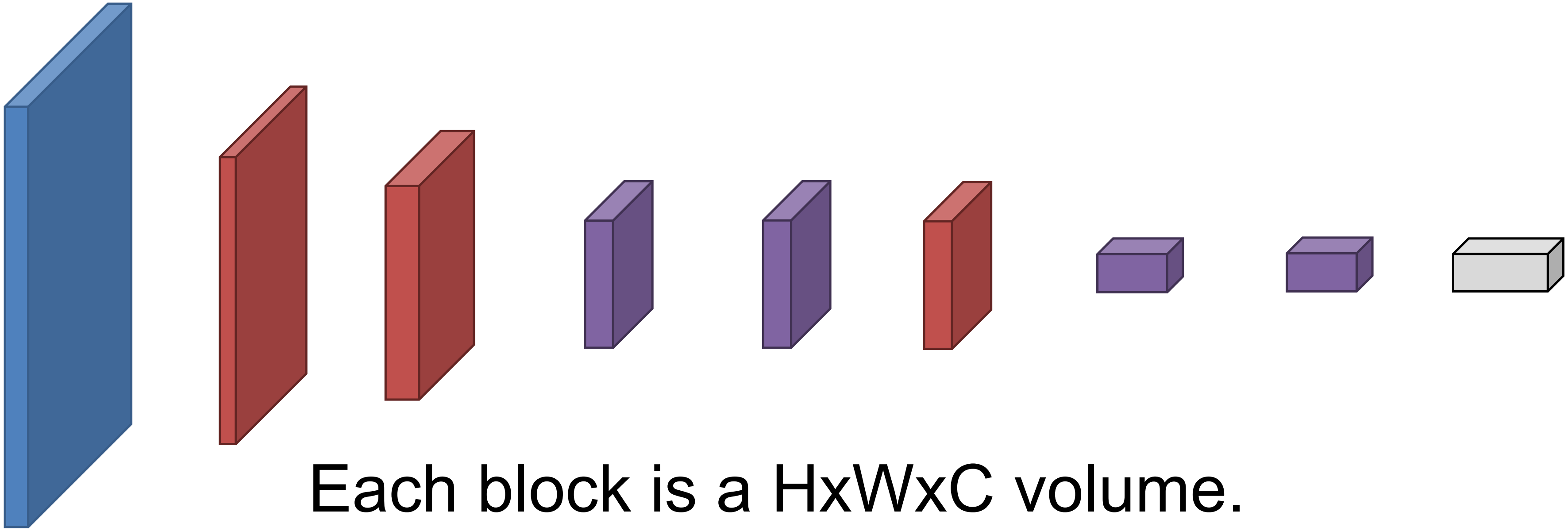
Single value
Per-filter

Looking At Networks

- We’ll look at 3 landmark networks, each trained to solve a 1000-way classification output (Imagenet)
  - Alexnet (2012)
  - VGG-16 (2014)
  - Resnet (2015)

AlexNet

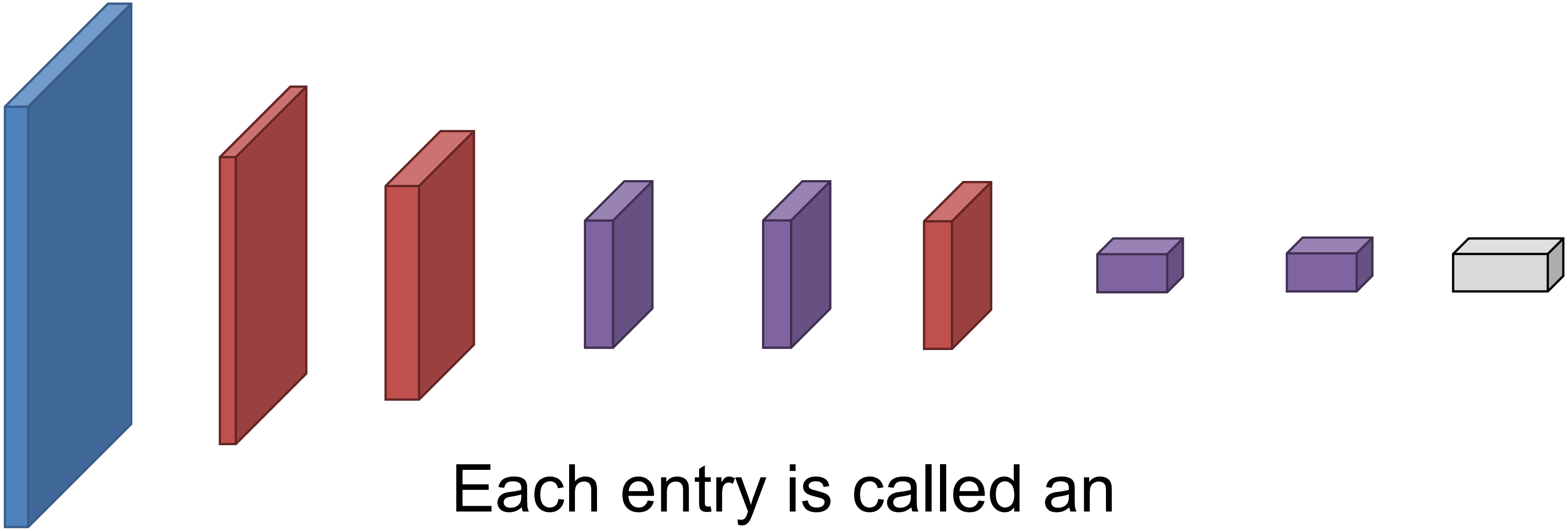
Input	Conv 1	Conv 2	Conv 3	Conv 4	Conv 5	FC 6	FC 7	Output
227x227	55x55	27x27	13x13	13x13	13x13	1x1	1x1	1x1
3	96	256	384	384	256	4096	4096	1000



Each block is a HxWxC volume.
You transform one volume to another with convolution

CNN Terminology

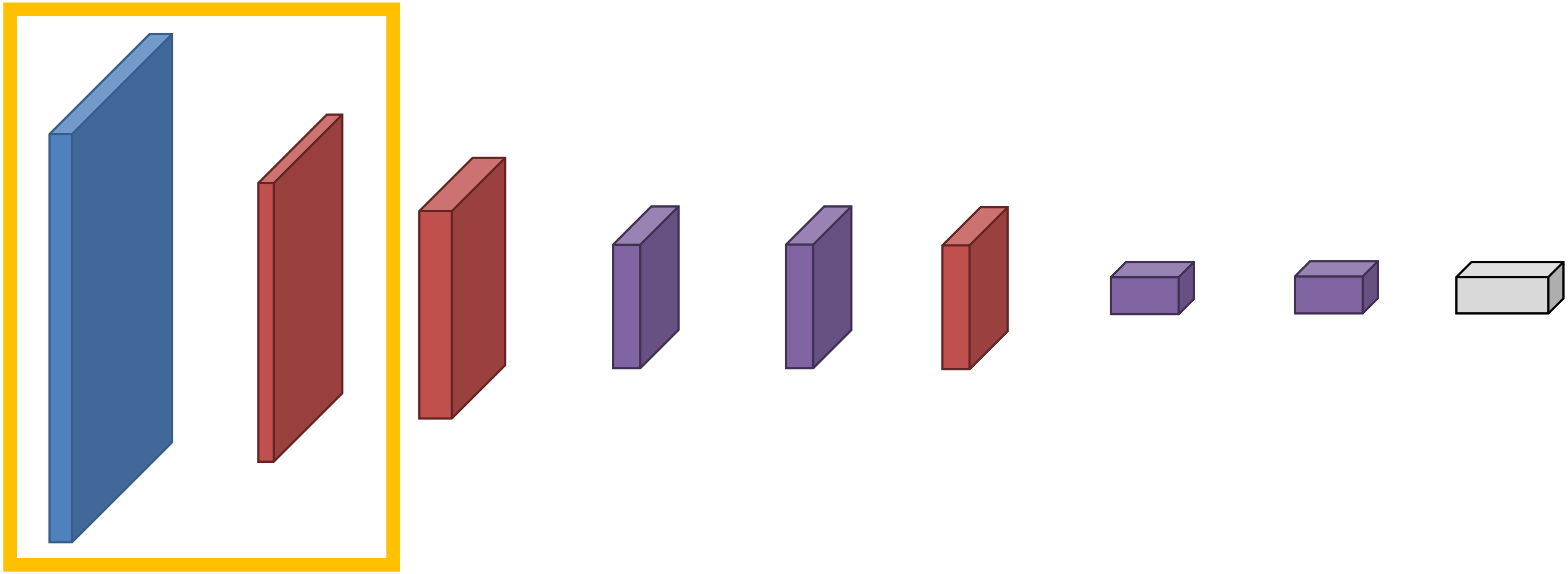
Input	Conv 1	Conv 2	Conv 3	Conv 4	Conv 5	FC 6	FC 7	Output
227x227	55x55	27x27	13x13	13x13	13x13	1x1	1x1	1x1
3	96	256	384	384	256	4096	4096	1000



Each entry is called an
“activation”/“neuron”/“feature”

AlexNet

Input	Conv 1	Conv 2	Conv 3	Conv 4	Conv 5	FC 6	FC 7	Output
227x227 3	55x55 96	27x27 256	13x13 384	13x13 384	13x13 256	1x1 4096	1x1 4096	1x1 1000



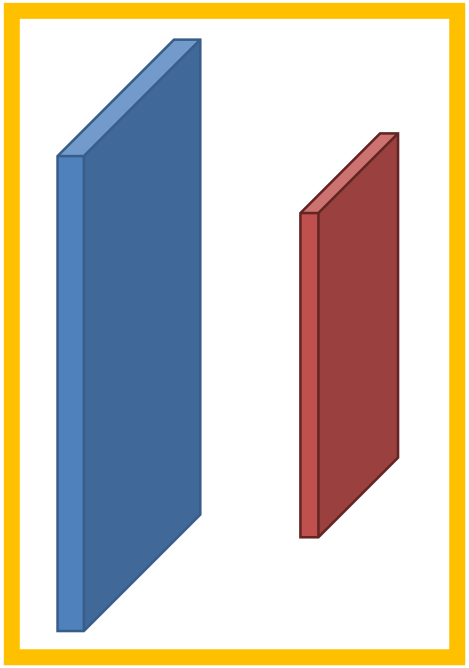
AlexNet

Input

227x227
3

Conv
1

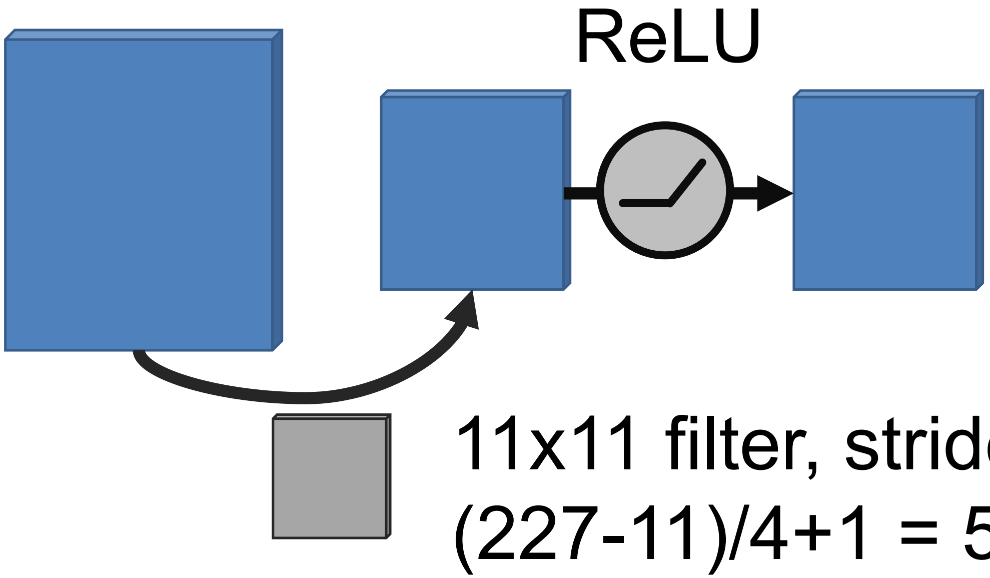
55x55
96



227x227
3

55x55
96

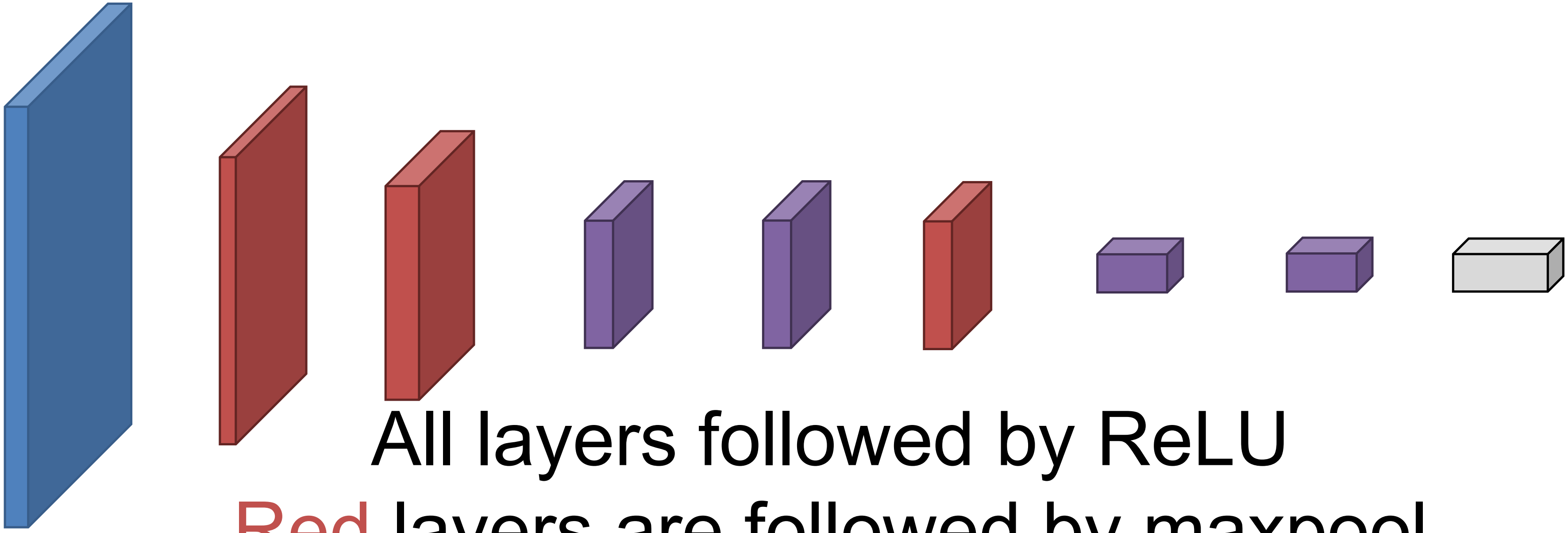
55x55
96



11x11 filter, stride of 4
(227-11)/4+1 = 55

AlexNet

Input	Conv 1	Conv 2	Conv 3	Conv 4	Conv 5	FC 6	FC 7	Output
227x227	55x55	27x27	13x13	13x13	13x13	1x1	1x1	1x1
3	96	256	384	384	256	4096	4096	1000



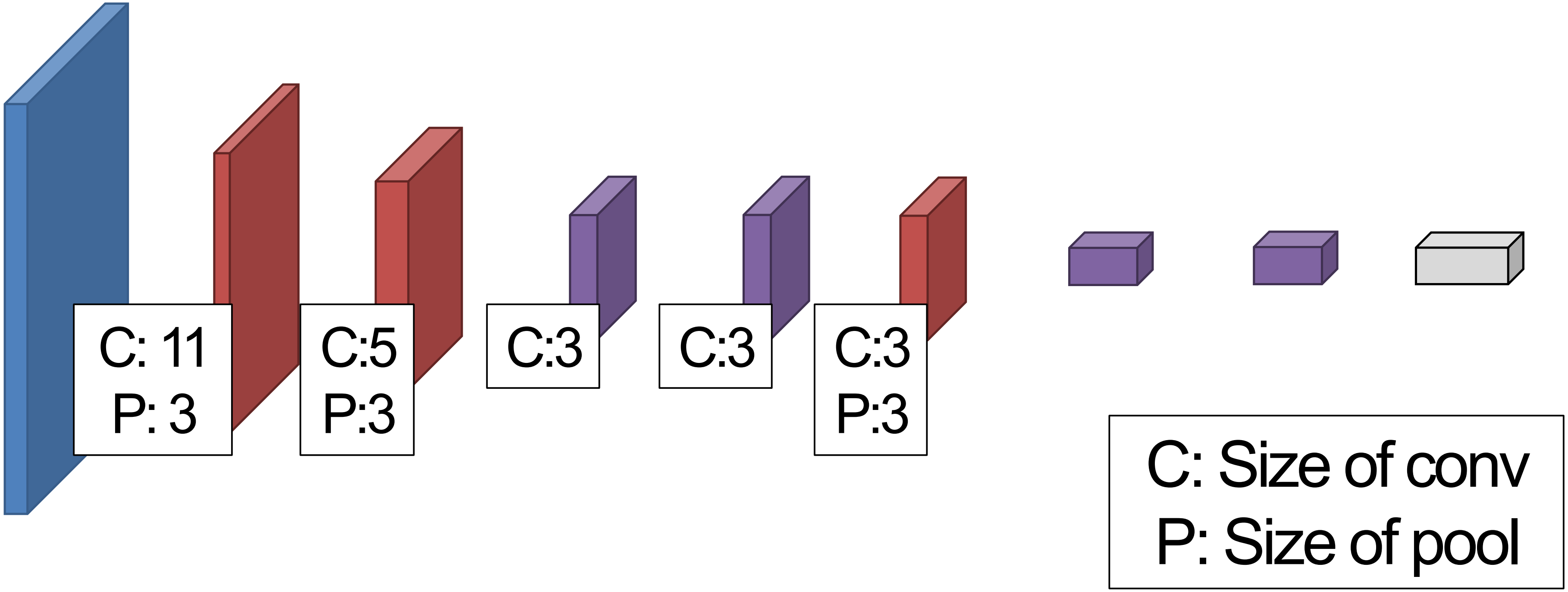
All layers followed by ReLU

Red layers are followed by maxpool

Early layers have “normalization”

AlexNet – Details

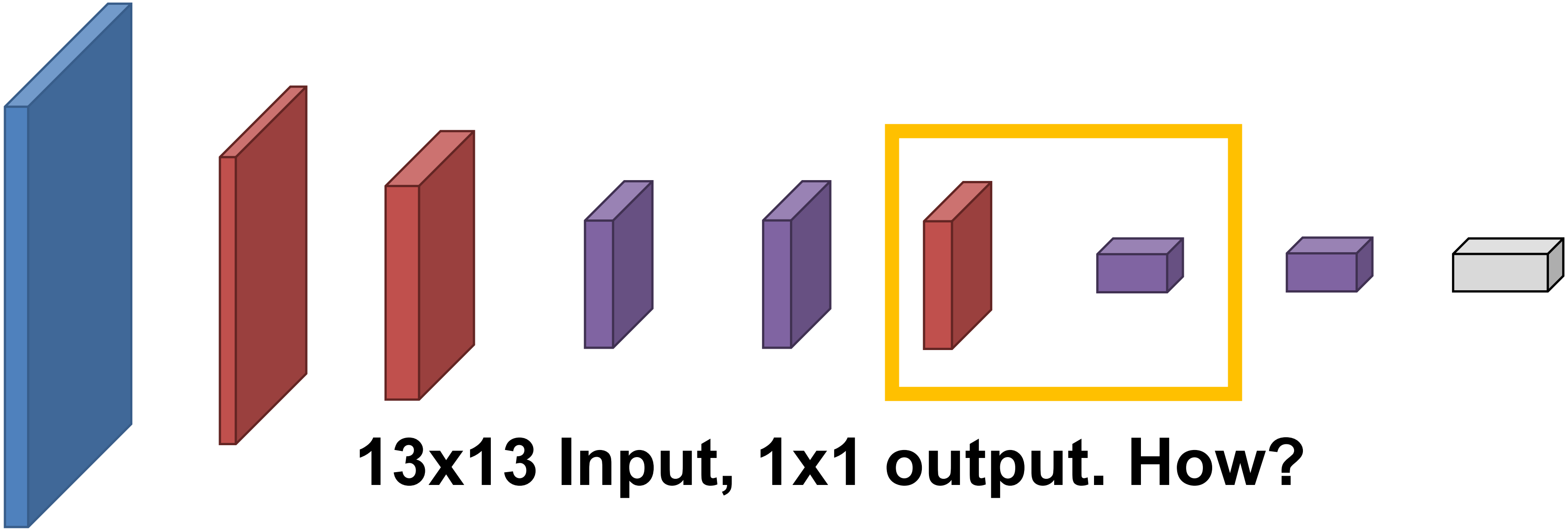
Input	Conv 1	Conv 2	Conv 3	Conv 4	Conv 5	FC 6	FC 7	Output
227x227	55x55	27x27	13x13	13x13	13x13	1x1	1x1	1x1
3	96	256	384	384	256	4096	4096	1000



C: Size of conv
P: Size of pool

AlexNet

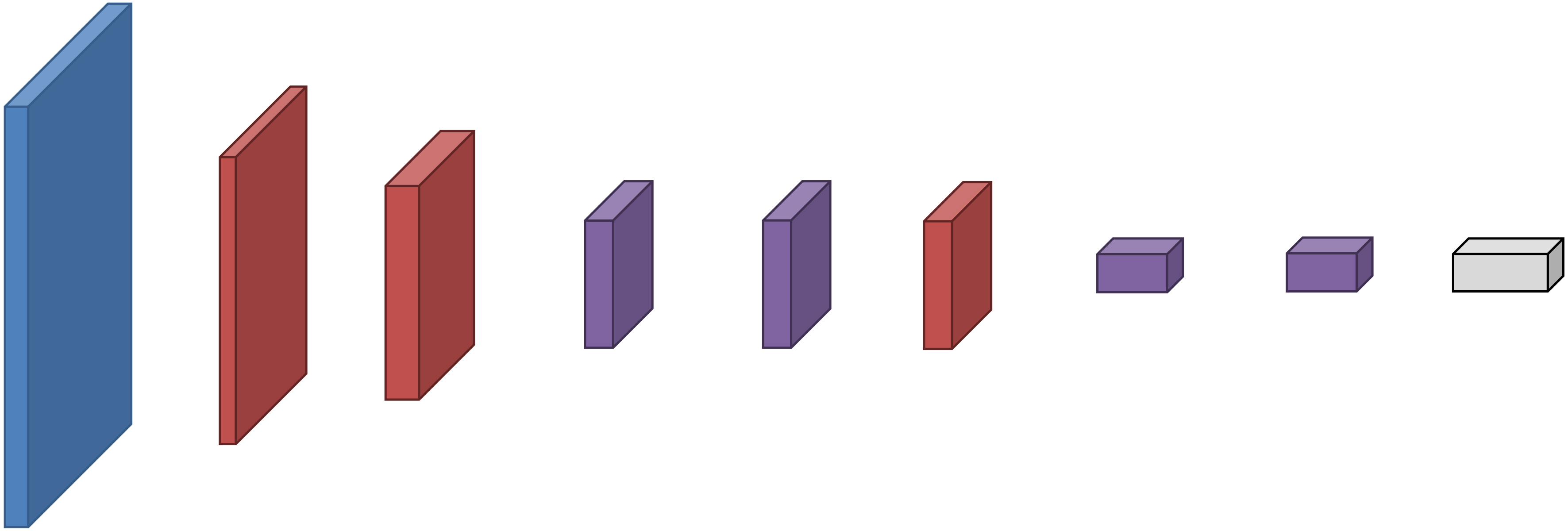
Input	Conv 1	Conv 2	Conv 3	Conv 4	Conv 5	FC 6	FC 7	Output
227x227 3	55x55 96	27x27 256	13x13 384	13x13 384	13x13 256	1x1 4096	1x1 4096	1x1 1000



13x13 Input, 1x1 output. How?

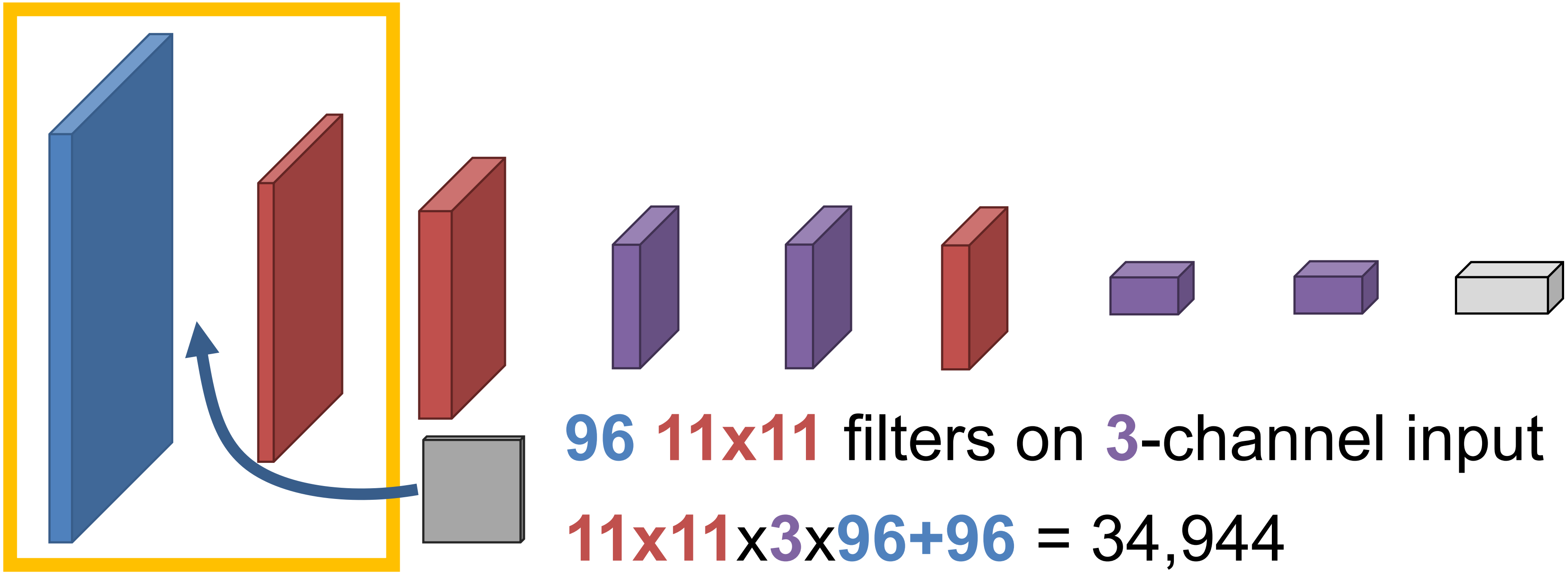
Alexnet – How Many Parameters?

Input	Conv 1	Conv 2	Conv 3	Conv 4	Conv 5	FC 6	FC 7	Output
227x227	55x55	27x27	13x13	13x13	13x13	1x1	1x1	1x1
3	96	256	384	384	256	4096	4096	1000



Alexnet – How Many Parameters?

Input	Conv 1	Conv 2	Conv 3	Conv 4	Conv 5	FC 6	FC 7	Output
227x227	55x55	27x27	13x13	13x13	13x13	1x1	1x1	1x1
3	96	256	384	384	256	4096	4096	1000

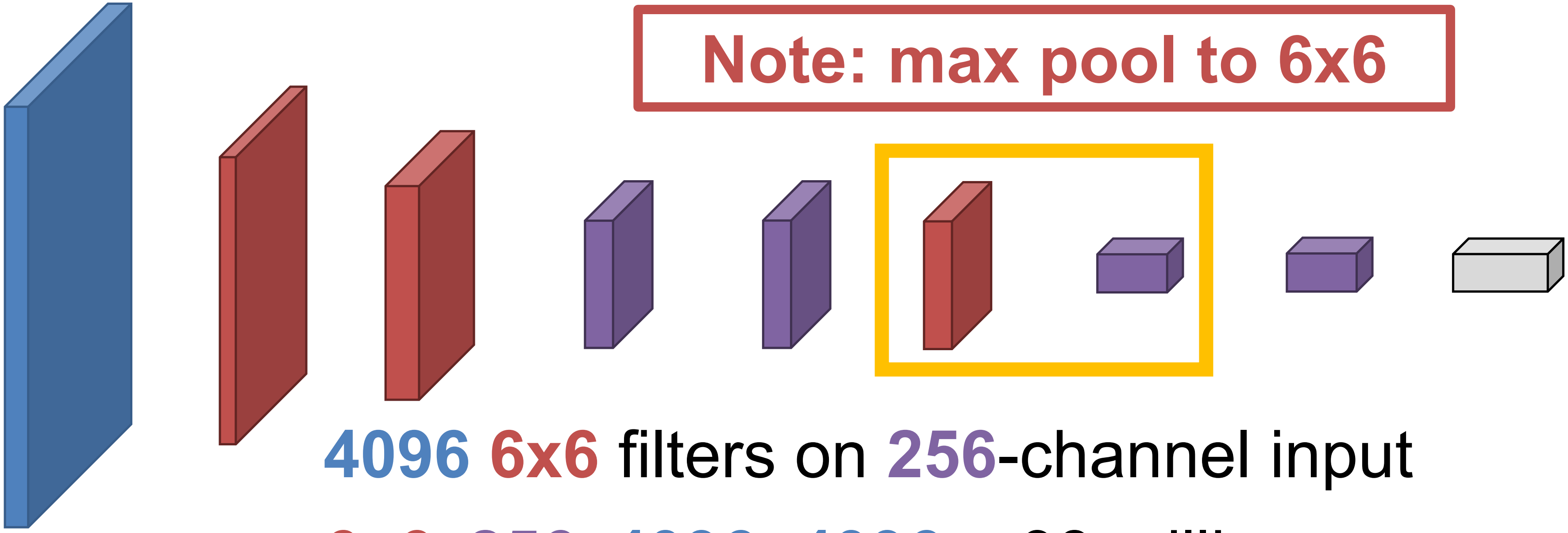


96 11x11 filters on 3-channel input

11x11x3x96+96 = 34,944

Alexnet – How Many Parameters?

Input	Conv 1	Conv 2	Conv 3	Conv 4	Conv 5	FC 6	FC 7	Output
227x227	55x55	27x27	13x13	13x13	13x13	1x1	1x1	1x1
3	96	256	384	384	256	4096	4096	1000



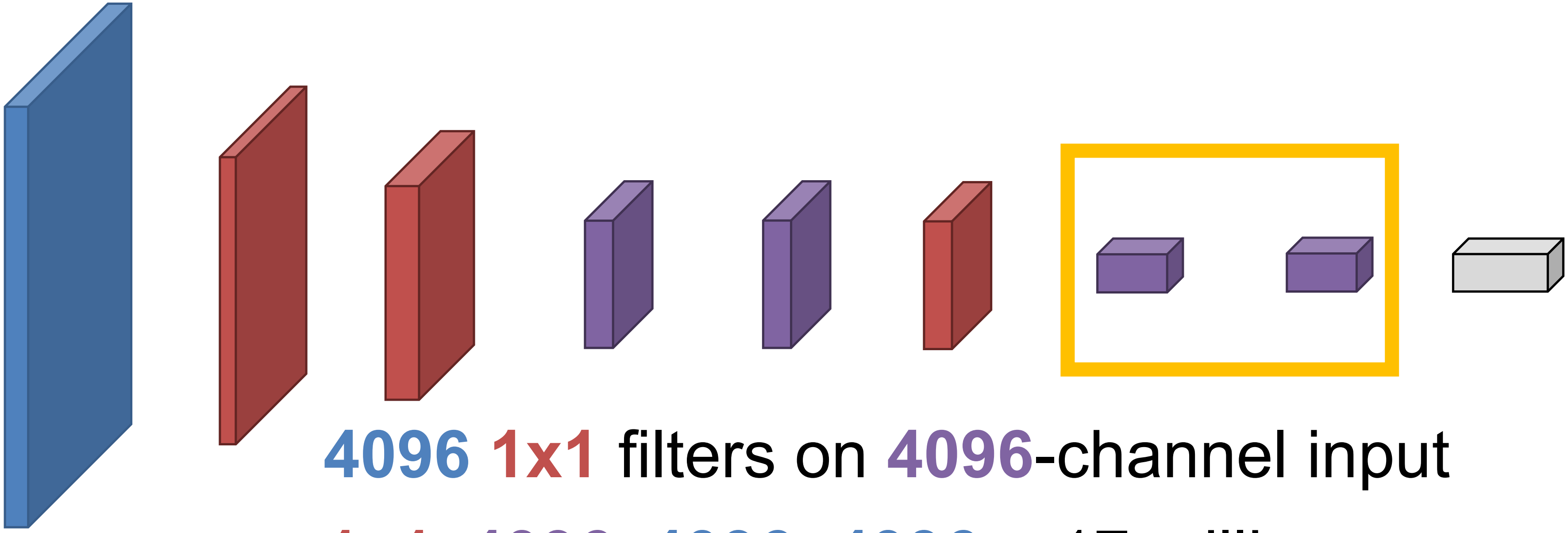
Note: max pool to 6x6

4096 **6x6** filters on **256**-channel input

6x6x**256**x**4096+4096** = 38 million

Alexnet – How Many Parameters?

Input	Conv 1	Conv 2	Conv 3	Conv 4	Conv 5	FC 6	FC 7	Output
227x227 3	55x55 96	27x27 256	13x13 384	13x13 384	13x13 256	1x1 4096	1x1 4096	1x1 1000



4096 1x1 filters on **4096**-channel input

1x1x**4096**x**4096+4096** = 17 million

Alexnet – How Many Parameters

How long would it take you to list the parameters of Alexnet at 4s / parameter?

1 year? 4 years? 8 years? 16 years?

- 62.4 million parameters
- *Vast majority in fully connected layers*
- But... paper notes that removing the convolutions is disastrous for performance.

Dataset – ILSVRC

- Imagenet Largescale Visual Recognition Challenge
- 1.4M images
- 1000 Categories, often ridiculously precise

Dataset – ILSVRC

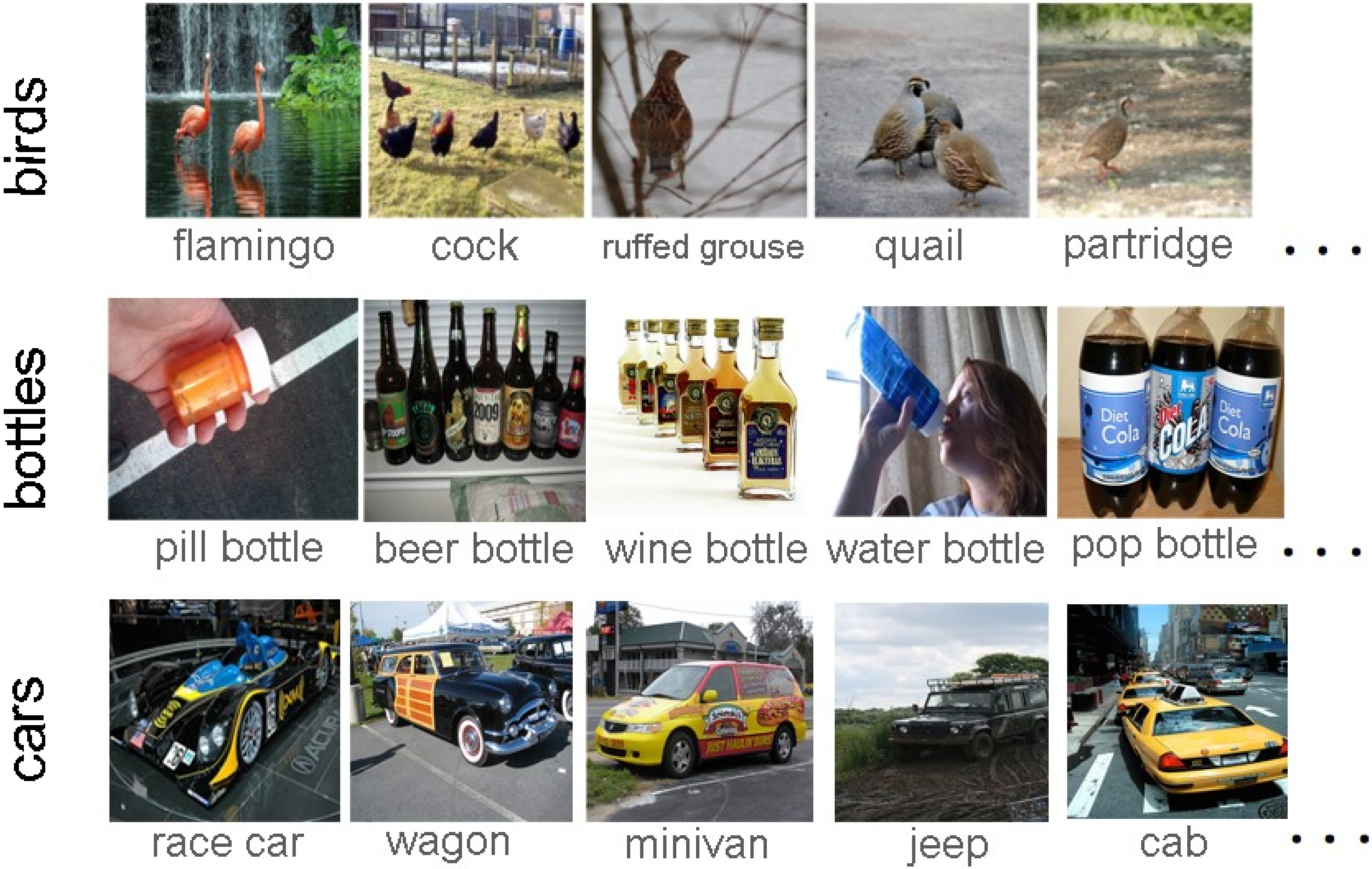
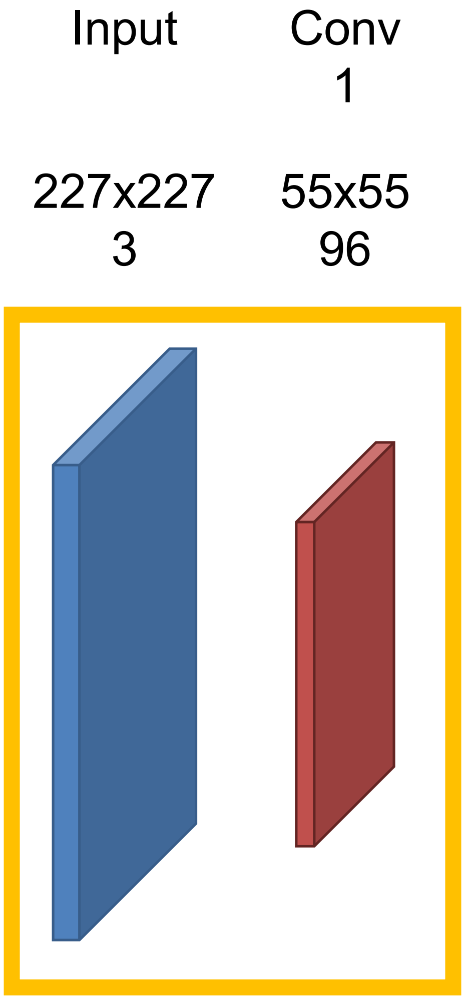


Figure Credit: O. Russakovsky

Visualizing Filters



Input

Conv 1

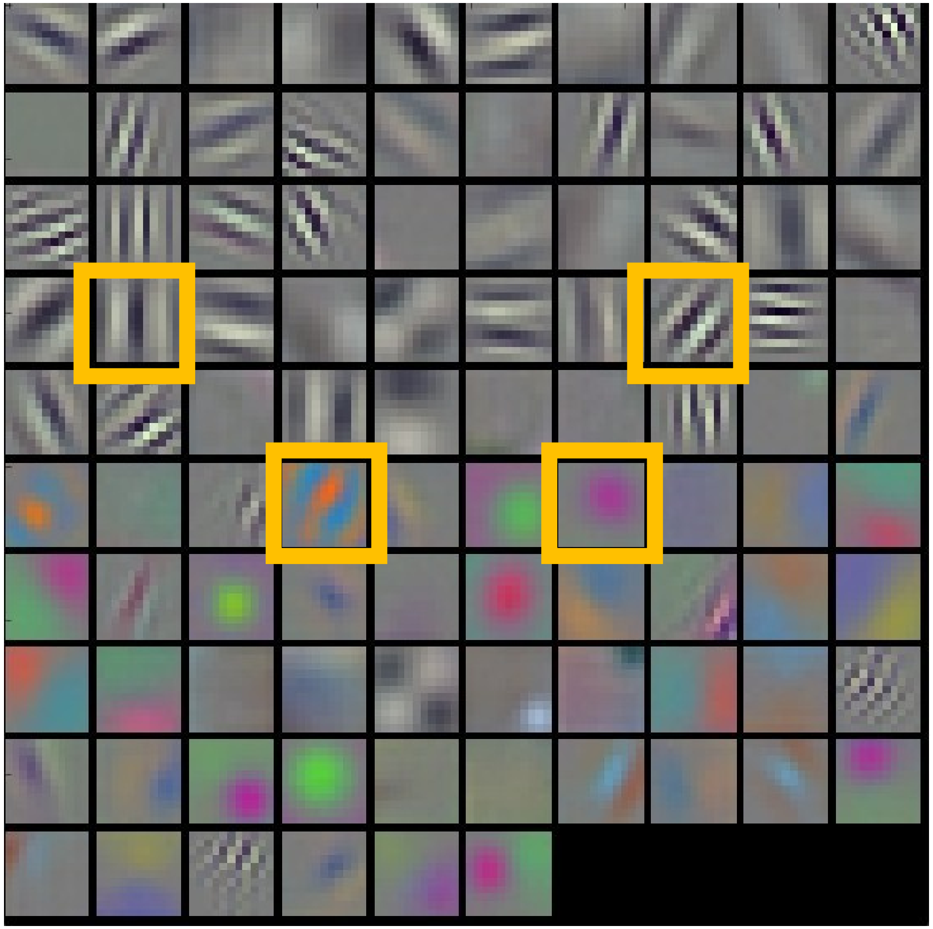
227x227
3

55x55
96

<u>Conv 1 Filters</u>

- **Q. How many input dimensions?**
  - A: 3
- **What does the input mean?**
  - R, G, B, duh.

What's Learned



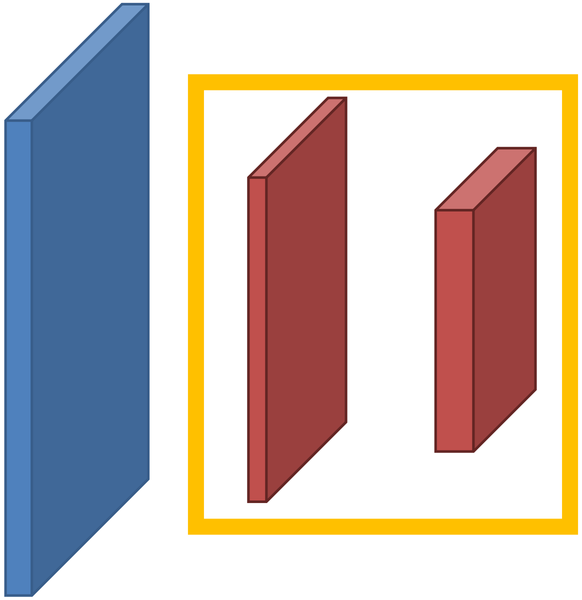
First layer filters of a network trained to distinguish 1000 categories of objects

Remember these filters go over color.

Figure Credit: Karpathy and Fei-Fei

Visualizing Later Filters

Input	Conv 1	Conv 2
227x227 3	55x55 96	27x27 256



Conv 2 Filters

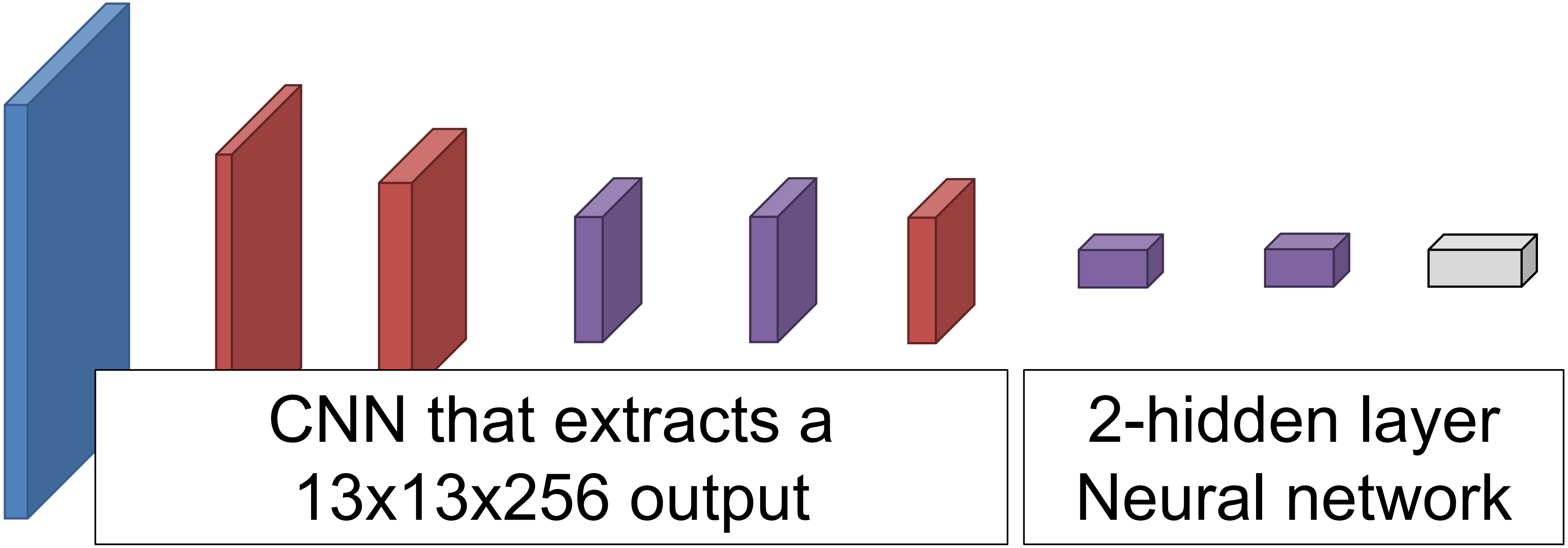
- **Q. How many input dimensions?**
  - A: 96…. hmmm
- **What does the input mean?**
  - Uh, the uh, previous slide

Visualizing Later Filters

- Understanding the meaning of the later filters *from their values* is typically impossible: too many input dimensions, not even clear what the input means.

Understanding Later Filters

Input	Conv 1	Conv 2	Conv 3	Conv 4	Conv 5	FC 6	FC 7	Output
227x227	55x55	27x27	13x13	13x13	13x13	1x1	1x1	1x1
3	96	256	384	384	256	4096	4096	1000

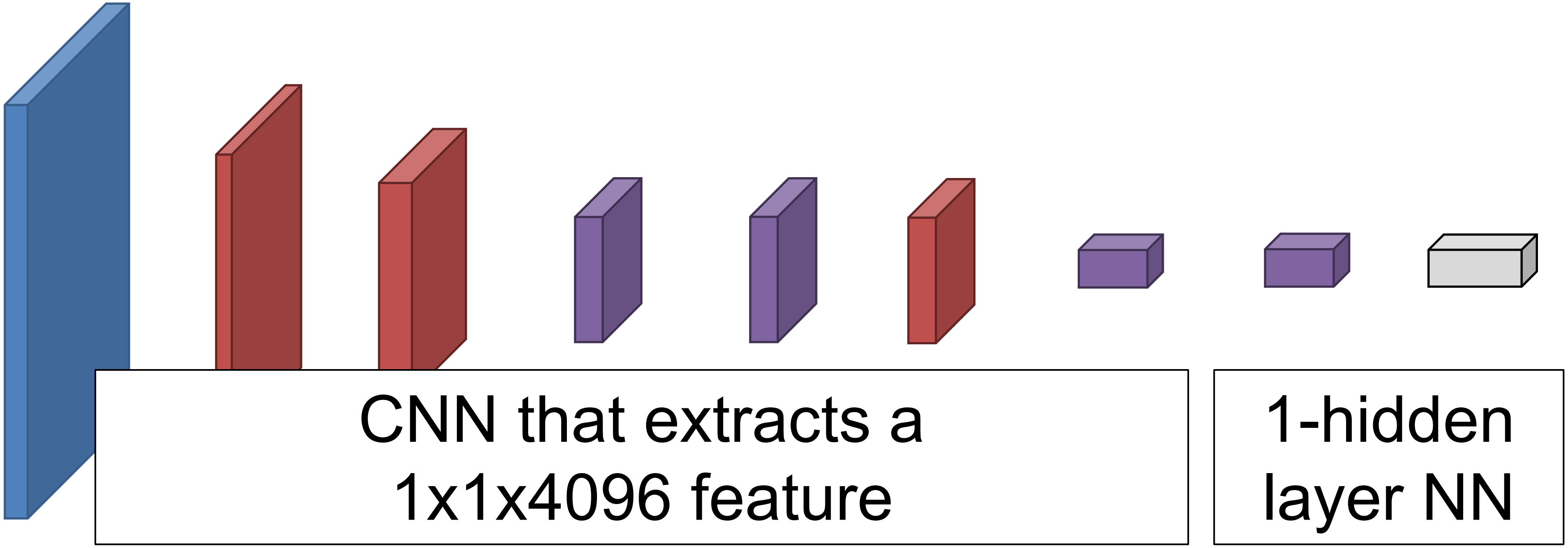


CNN that extracts a 13x13x256 output

2-hidden layer Neural network

Understanding Later Filters

Input	Conv 1	Conv 2	Conv 3	Conv 4	Conv 5	FC 6	FC 7	Output
227x227	55x55	27x27	13x13	13x13	13x13	1x1	1x1	1x1
3	96	256	384	384	256	4096	4096	1000

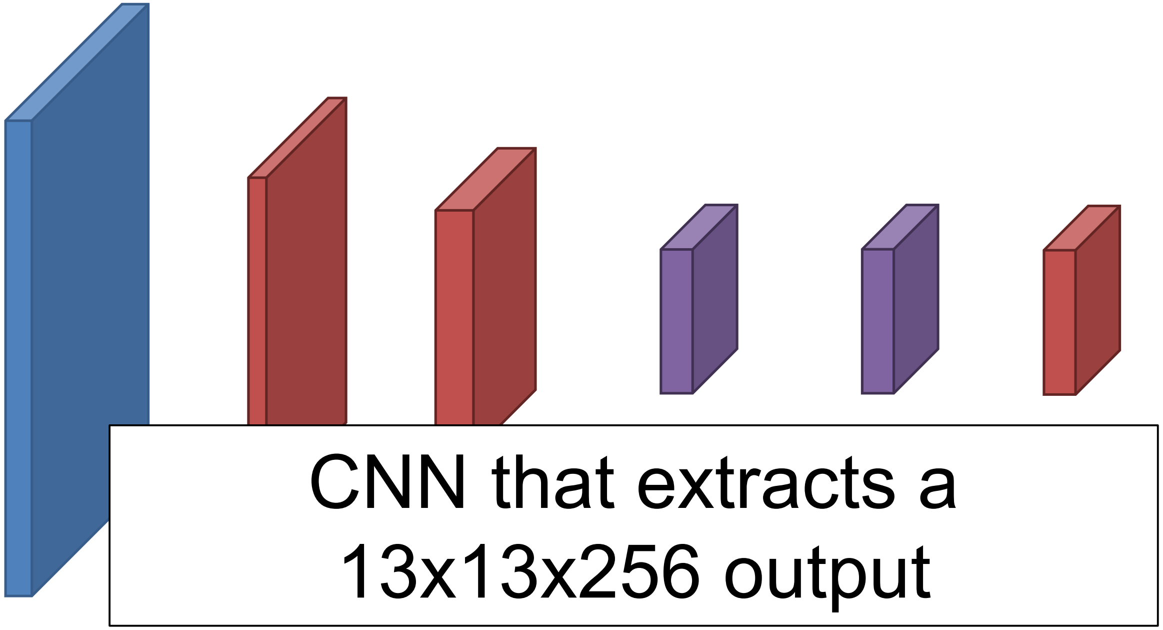


CNN that extracts a 1x1x4096 feature

1-hidden layer NN

Understanding Later Filters

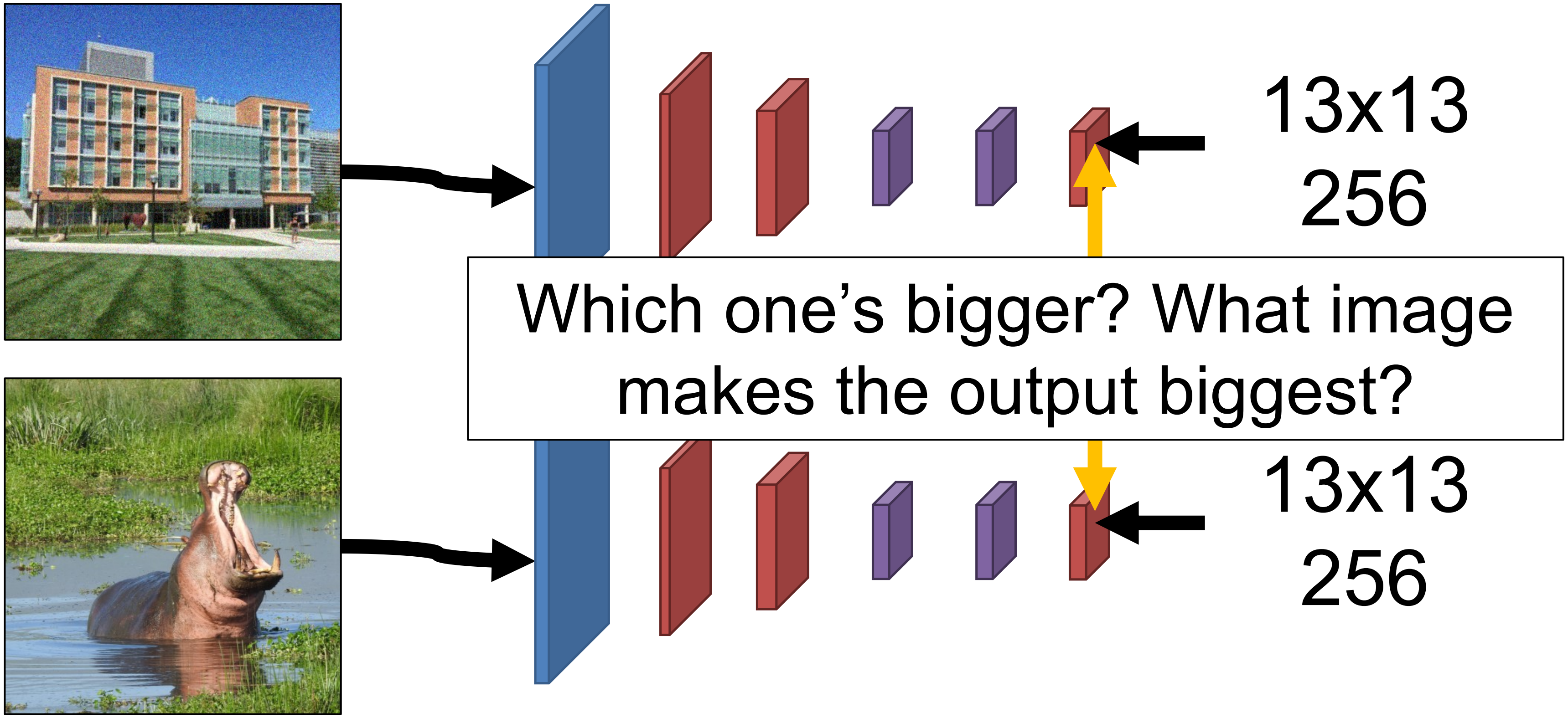
Input	Conv 1	Conv 2	Conv 3	Conv 4	Conv 5
227x227	55x55	27x27	13x13	13x13	13x13
3	96	256	384	384	256



CNN that extracts a 13x13x256 output

Understanding Later Filters

Feed an image in, see what score the filter gives it. A more pleasant version of a real neuroscience procedure.



Which one's bigger? What image makes the output biggest?

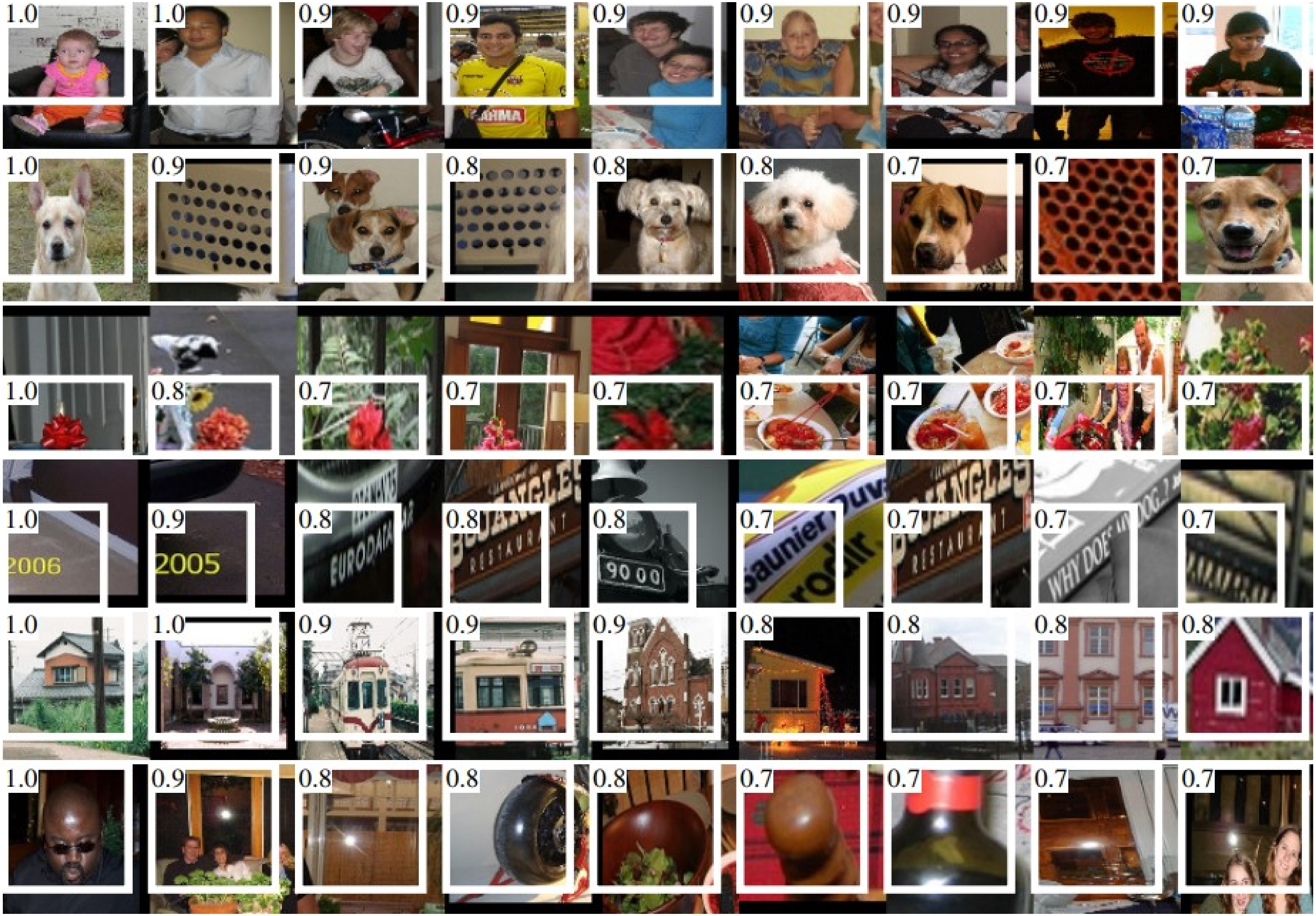
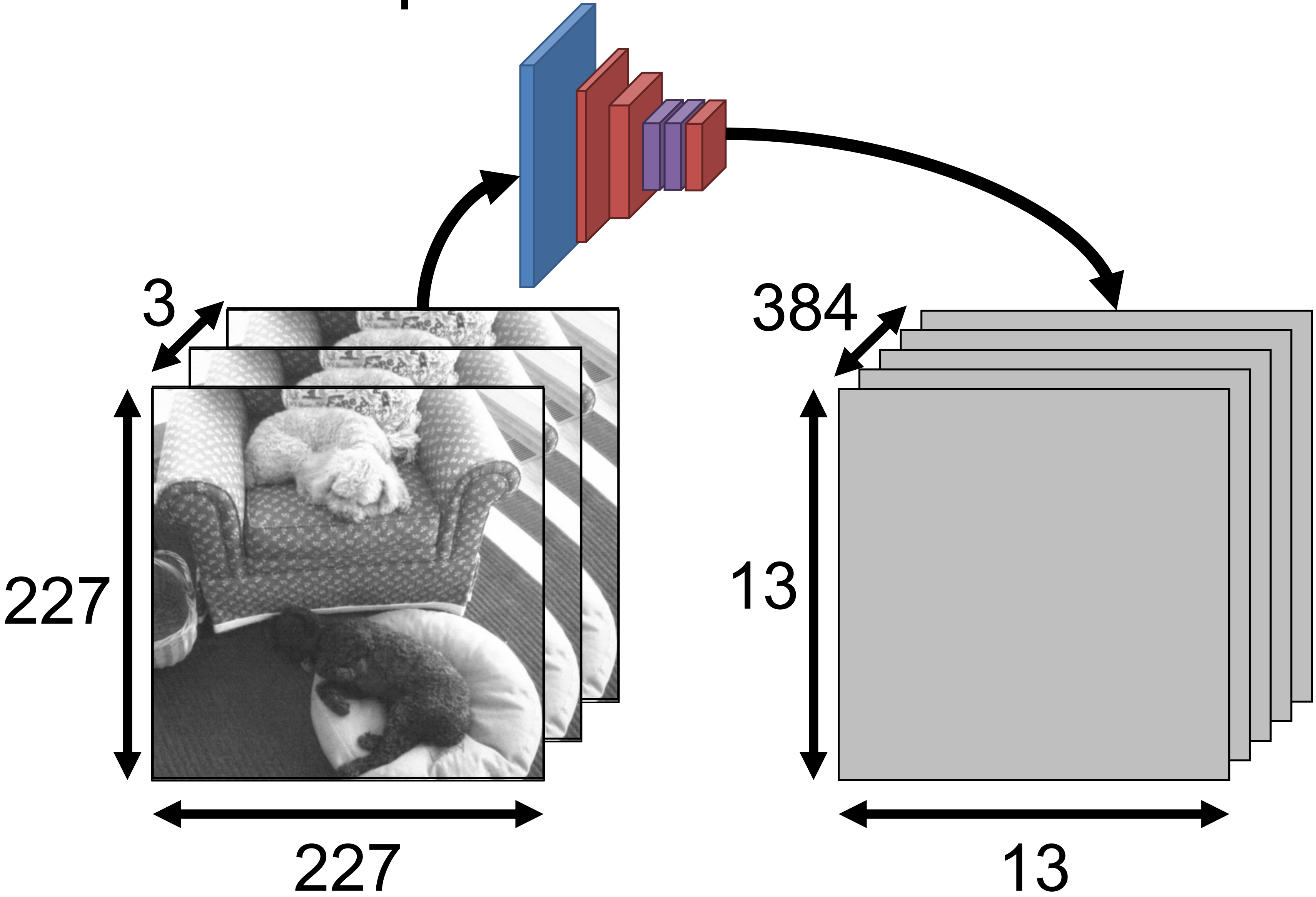
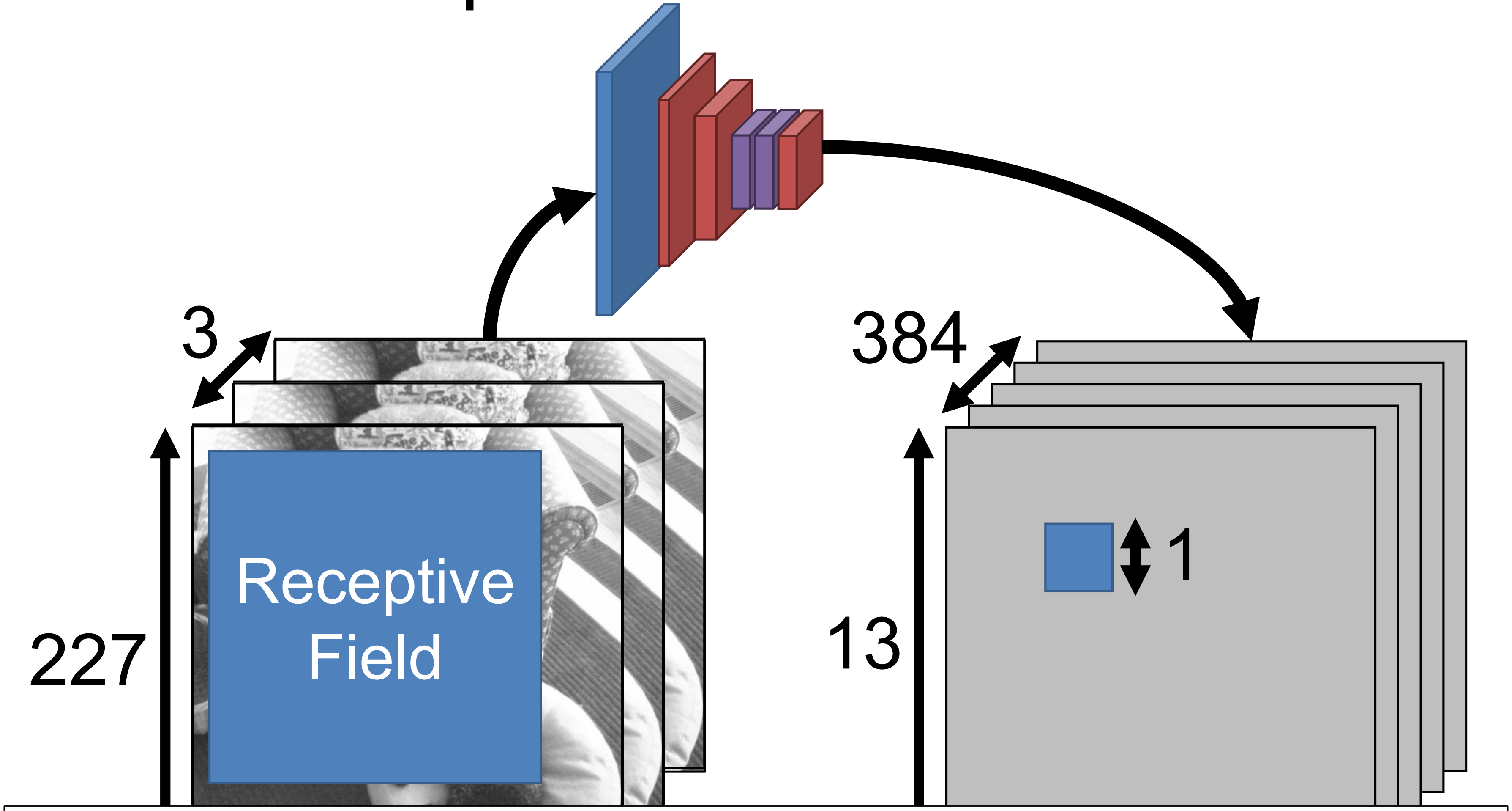


Figure Credit: Girschick et al. CVPR 2014.

What’s Up With the White Boxes?

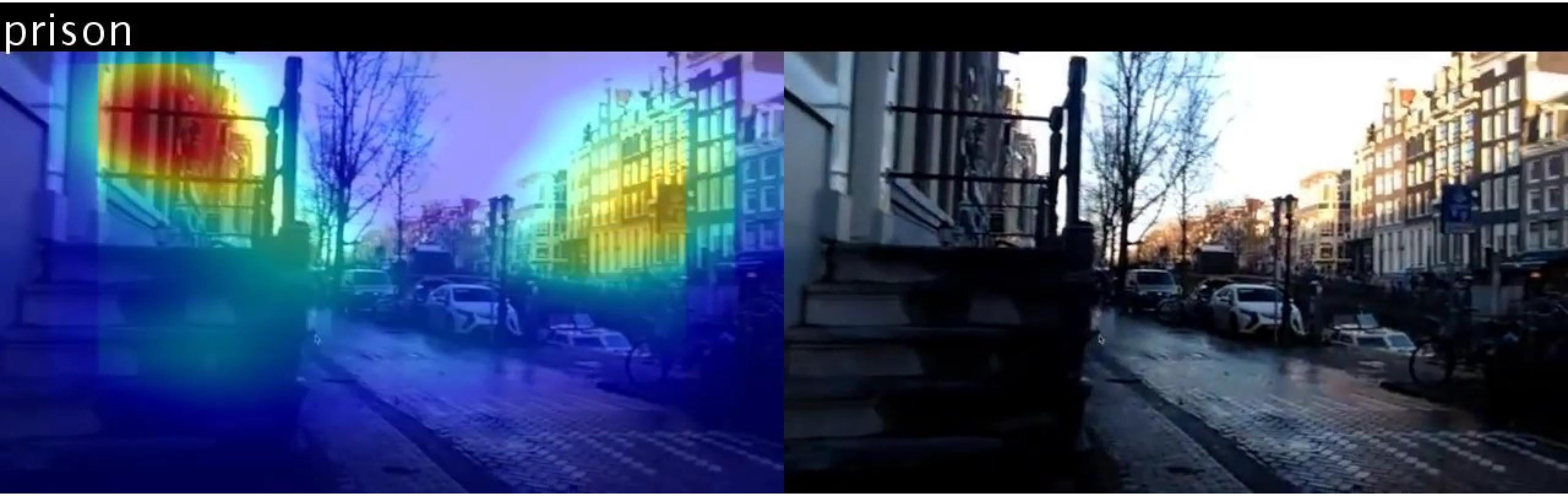


What's Up With the White Boxes?



Due to convolution, each later layer's value depends on / "sees" only a fraction of the input image.

Can use receptive fields to see where the network is “looking” to make its decisions



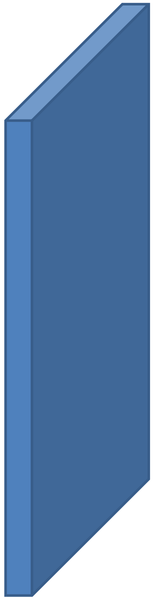
A very active area of research
(lots of great work done by Bolei Zhou, MIT → CUHK)

B. Zhou et al. Learning Deep Features for Discriminative Localization. CVPR 2016.

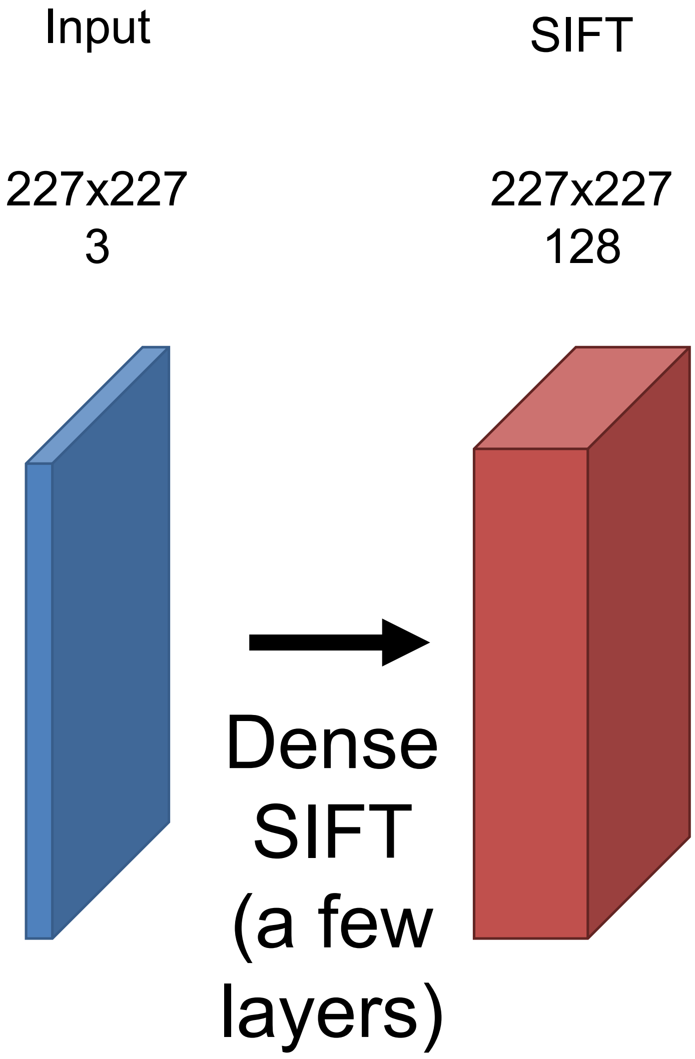
Classic Recognition

Input

227x227
3



Classic Recognition



Input

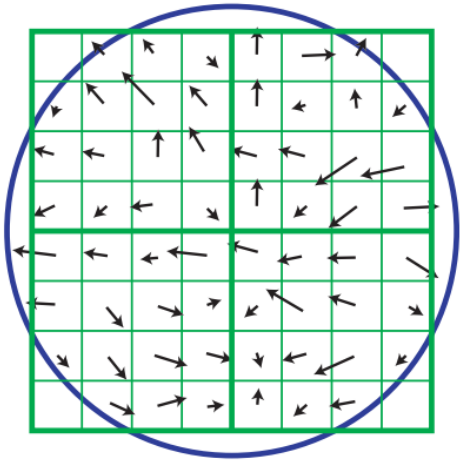
227x227
3

SIFT

227x227
128

Dense
SIFT
(a few
layers)

Recall: can compute a descriptor based on histograms of image gradients. Do it densely (at each pixel).



Classic Recognition

Input

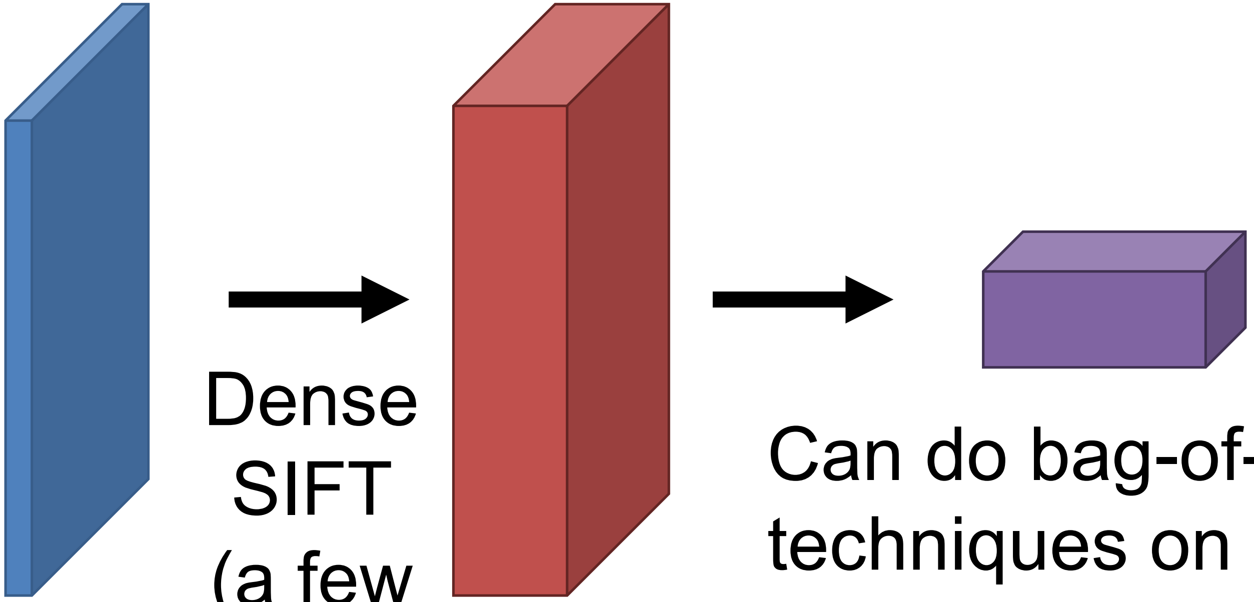
227x227
3

SIFT

227x227
128

Bag of Words

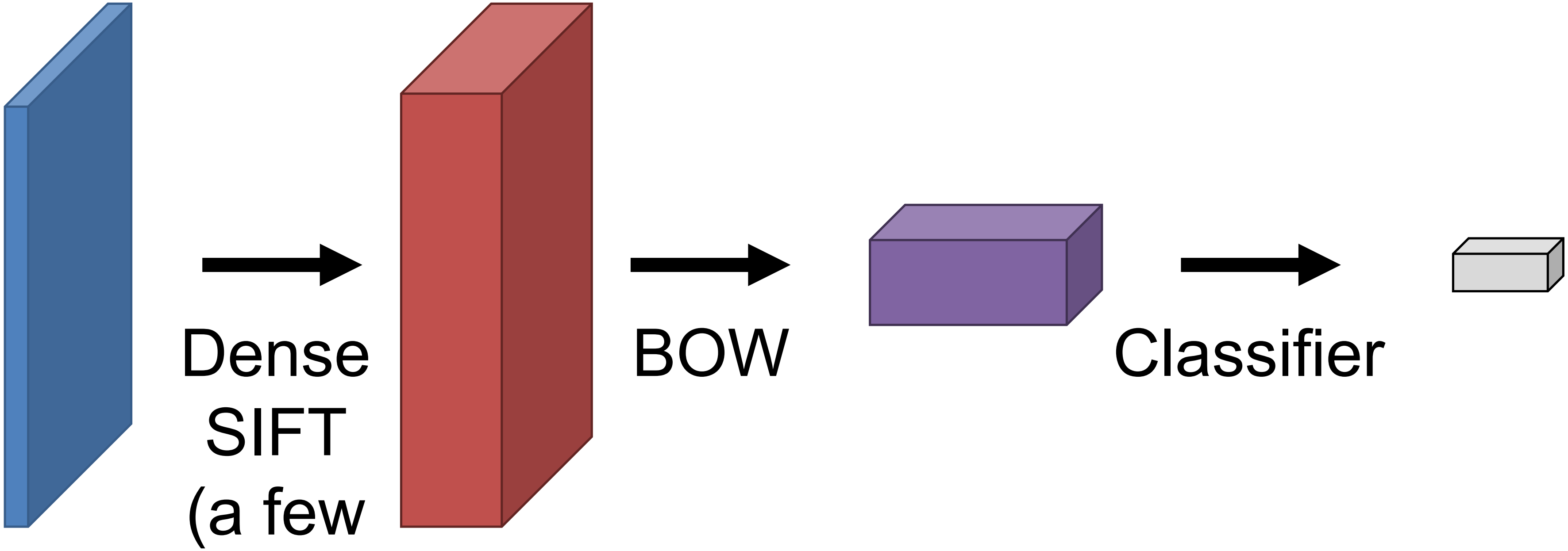
HxW
#codewords



Dense
SIFT
(a few
layers)

Can do bag-of-words-like techniques on SIFT, taking into consideration spatial location.

Classic Recognition



Input

227x227
3

SIFT

227x227
128

Bag of Words

HxW
#codewords

Output

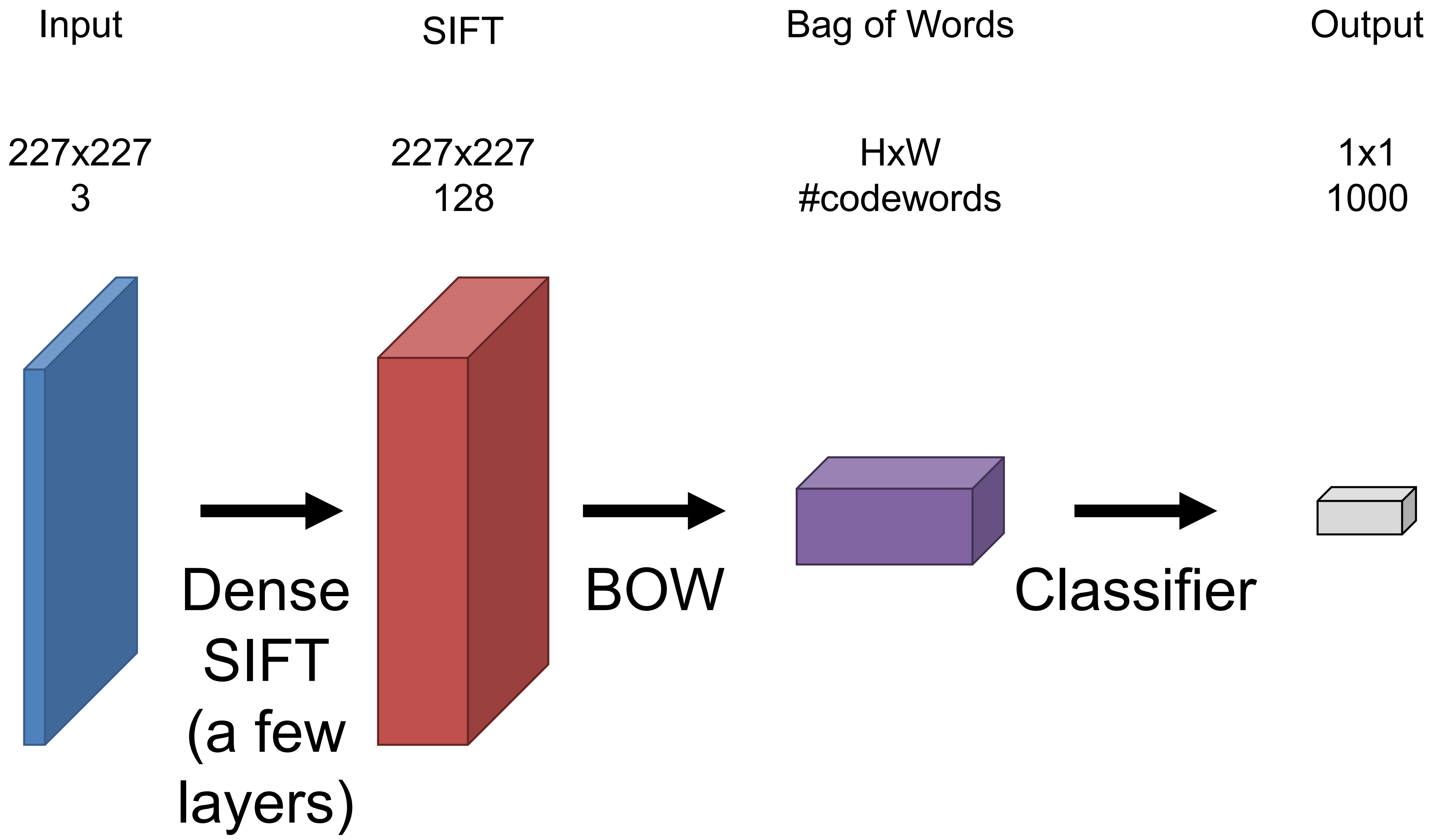
1x1
1000

Dense SIFT (a few layers)

BOW

Classifier

Classic Recognition



Input

227x227
3

SIFT

227x227
128

Bag of Words

HxW
#codewords

Output

1x1
1000

Dense SIFT (a few layers)

BOW

Classifier

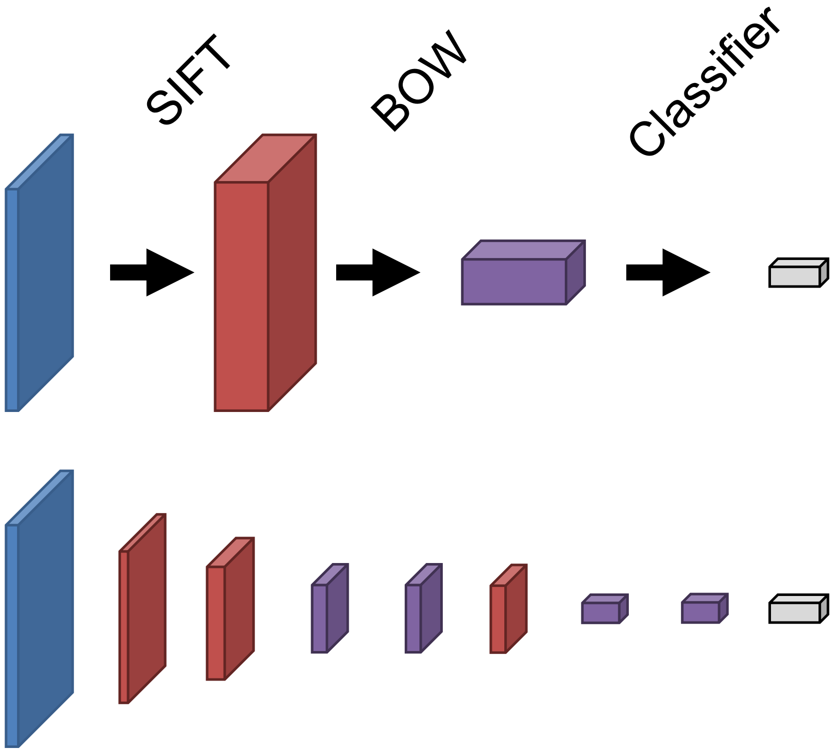
Classic vs Deep Recognition

Classic

Pipeline of hand-engineered steps

Deep

Pipeline of learned convolutions + simple operations

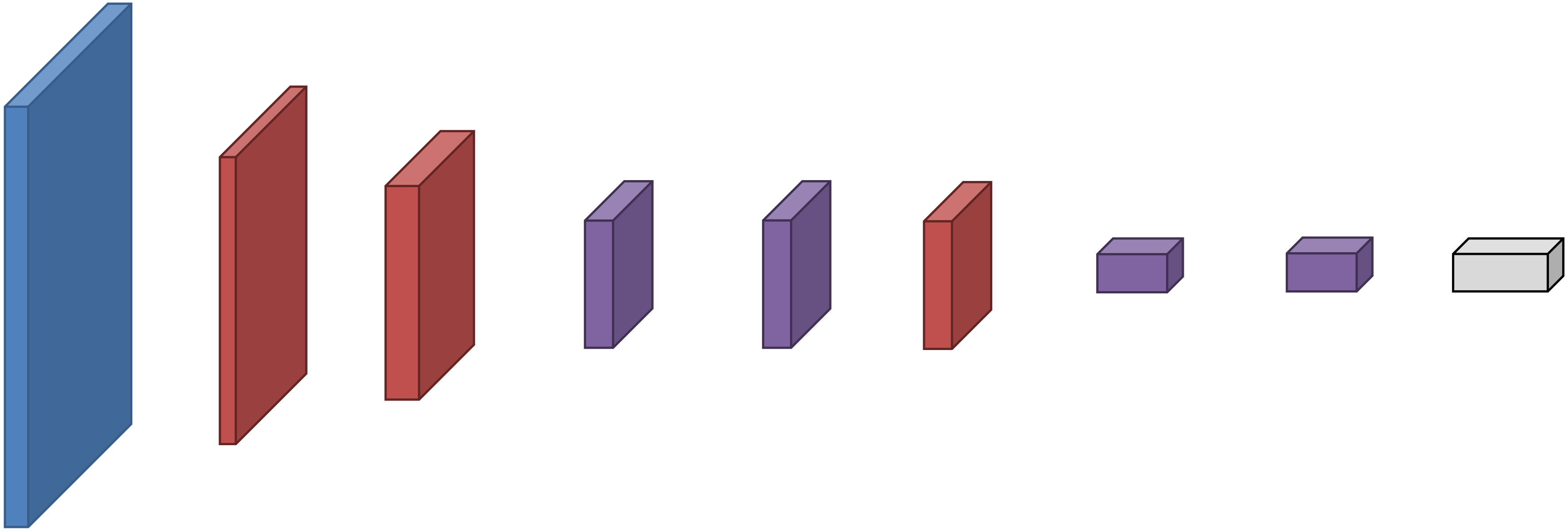


What are some differences?

The classic steps don’t: talk to each other or have many parameters that are learned from data.

Last Time on EECS 442

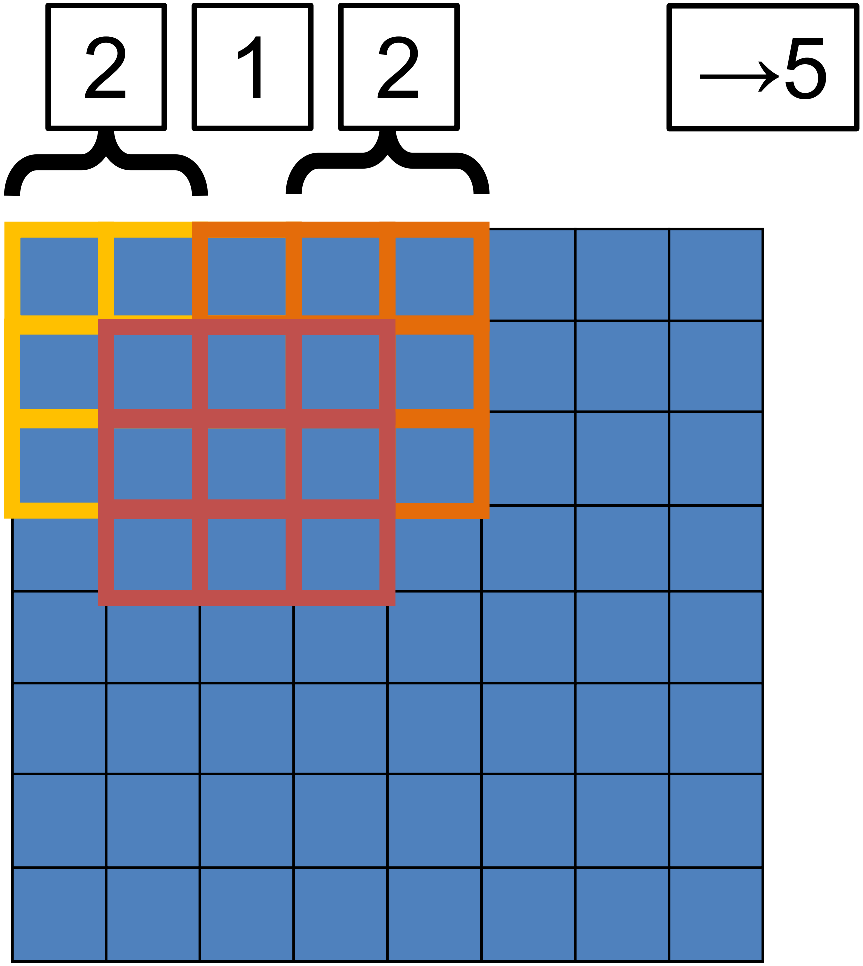
Input	Conv 1	Conv 2	Conv 3	Conv 4	Conv 5	FC 6	FC 7	Output
227x227 3	55x55 96	27x27 256	13x13 384	13x13 384	13x13 256	1x1 4096	1x1 4096	1x1 1000



3 Key Developments Since Alexnet

- 3x3 Filters
- Batch Normalization
- Residual Learning

Key Idea – 3x3 Filters

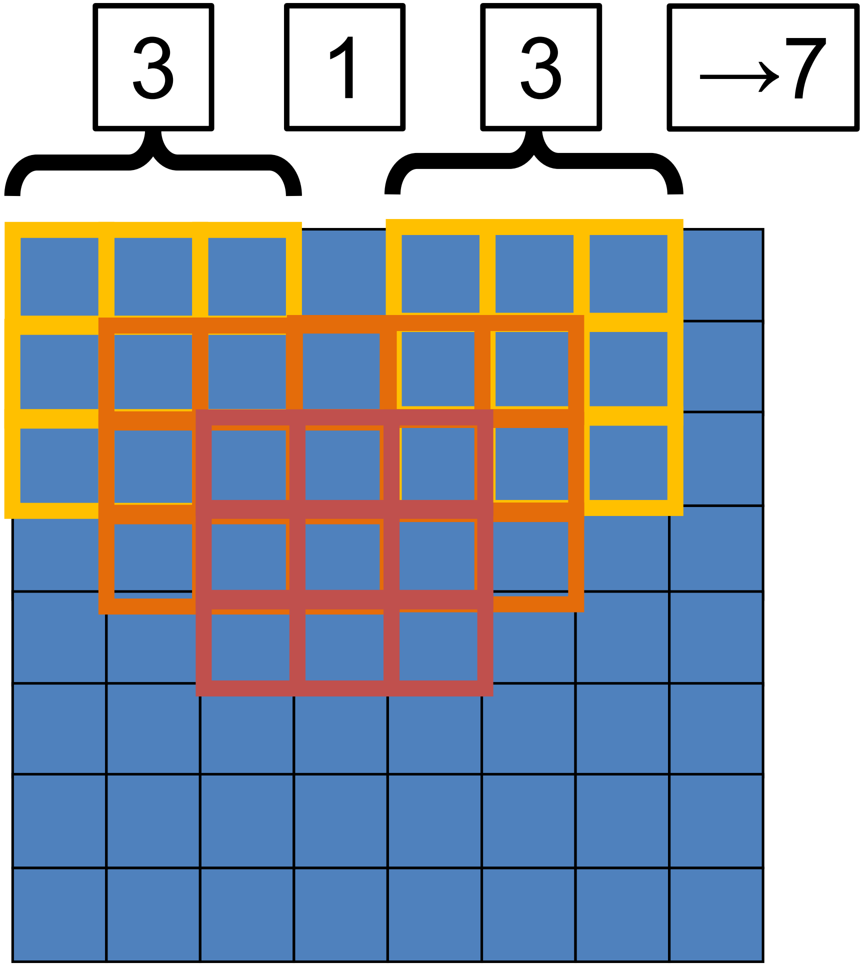


3x3 filter followed by 3x3 filter

→

Filter with 5x5 receptive field

Key Idea – 3x3 Filters

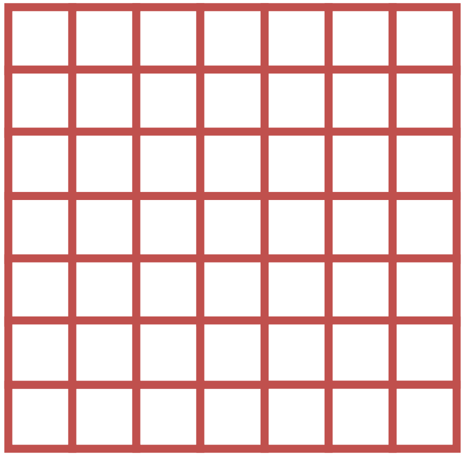


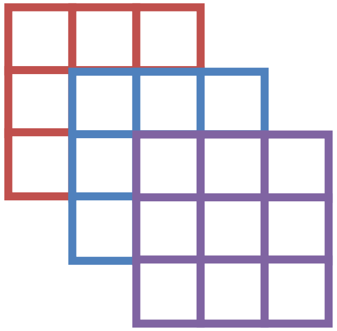
3x3 filter followed by
3x3 filter followed by
3x3 filter

→

Filter with 7x7
receptive field

Why Does This Make A Difference?

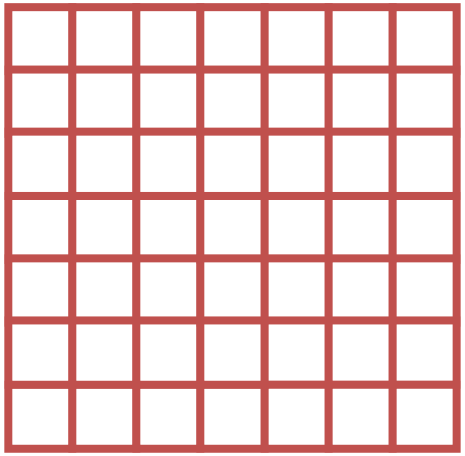




Empirically, repeated 3x3 filters do better compared to a 7x7 filter.

Why?

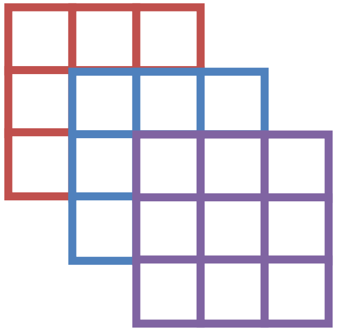
Key Idea – 3x3 Filters



Receptive Field: 7x7 pixels

Parameters/channel: 49

Number of ReLUs: 1



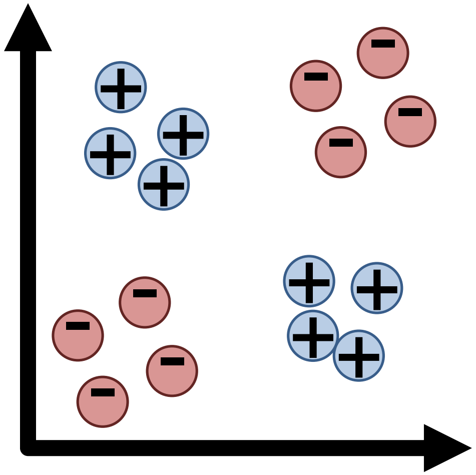
Receptive Field: 7x7 pixels

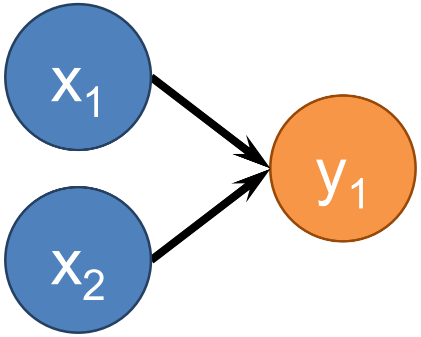
Parameters/channel: 3x3x3=**27**

Number of ReLUs: **3**

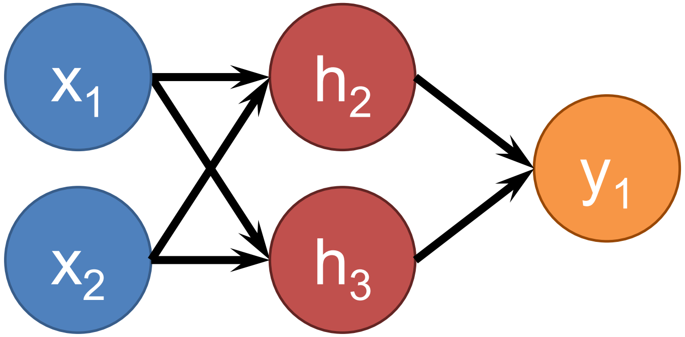
We Want More Non-linearity!

Can they implement xor?





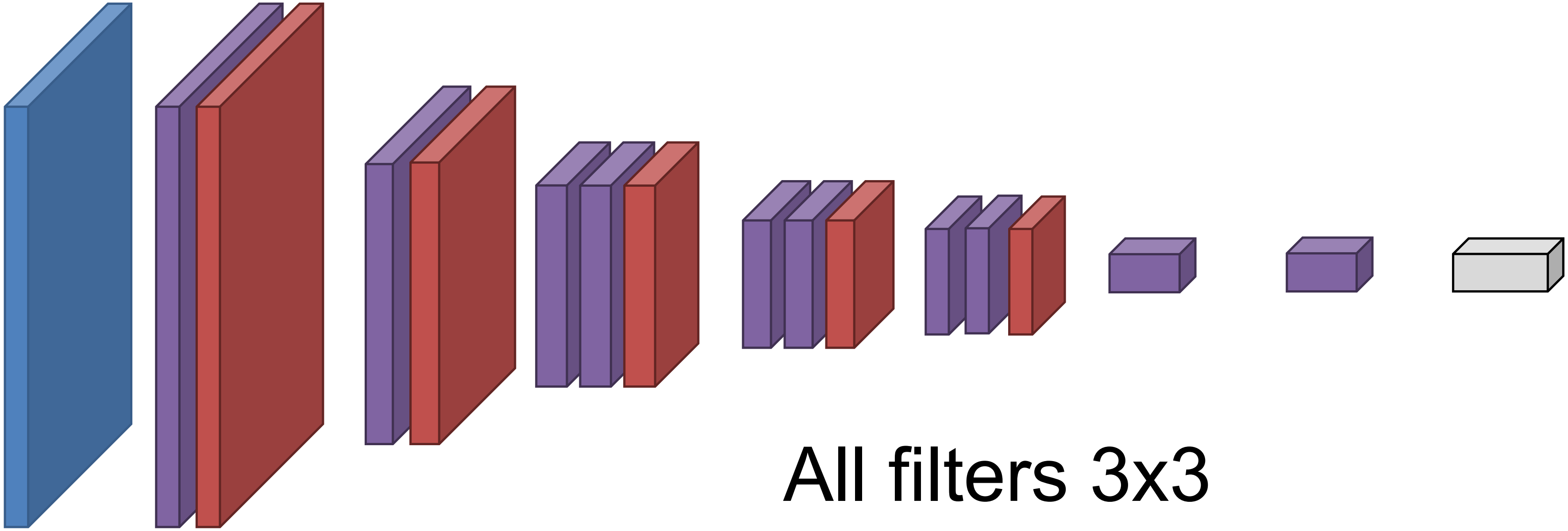
No



Yes

VGG16

Input	Conv 1	Conv 2	Conv 3	Conv 4	Conv 5	FC 6	FC 7	Output
224x224 3	224x224 64	112x112 128	56x56 256	28x28 512	14x14 512	1x1 4096	1x1 4096	1x1 1000



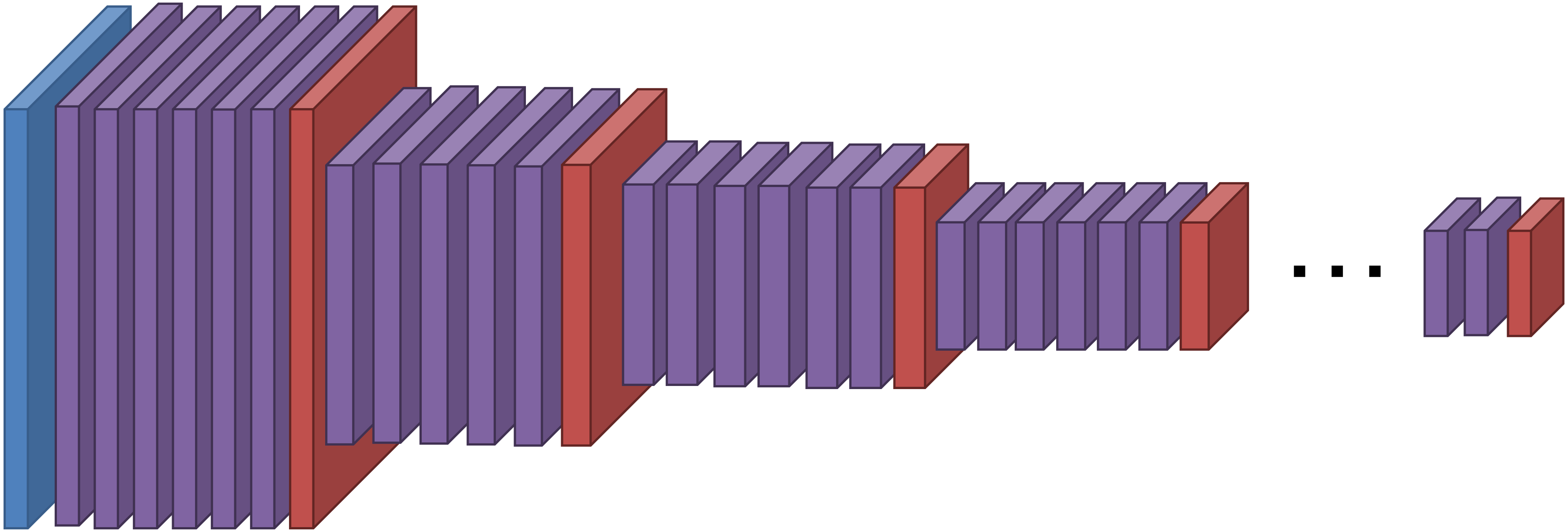
All filters 3x3

All filters followed by ReLU

Training Deeper Networks

Why not just stack continuously?

What will happen to gradient going back?



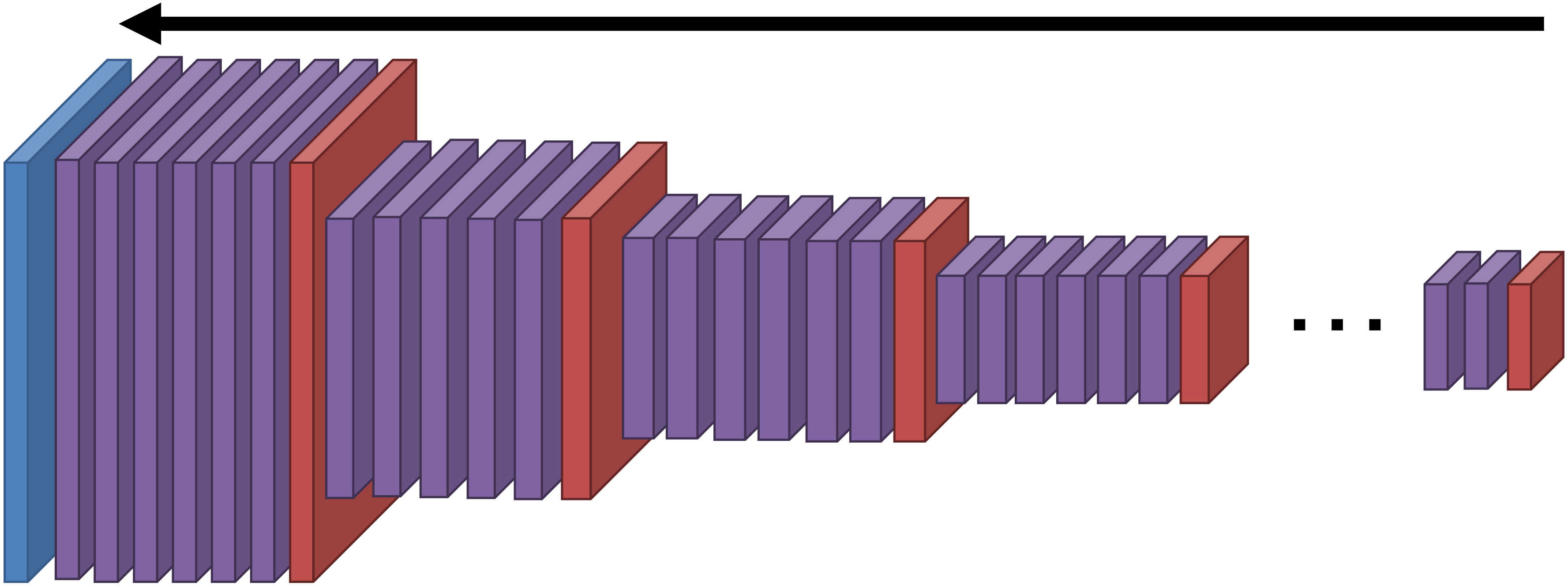
Backprop

Every backpropagation step multiplies the gradient by the local gradient

$1 * d * d * d \dots * d = d^{n-1}$

What if d << 1, n big?

Vanishing Gradients



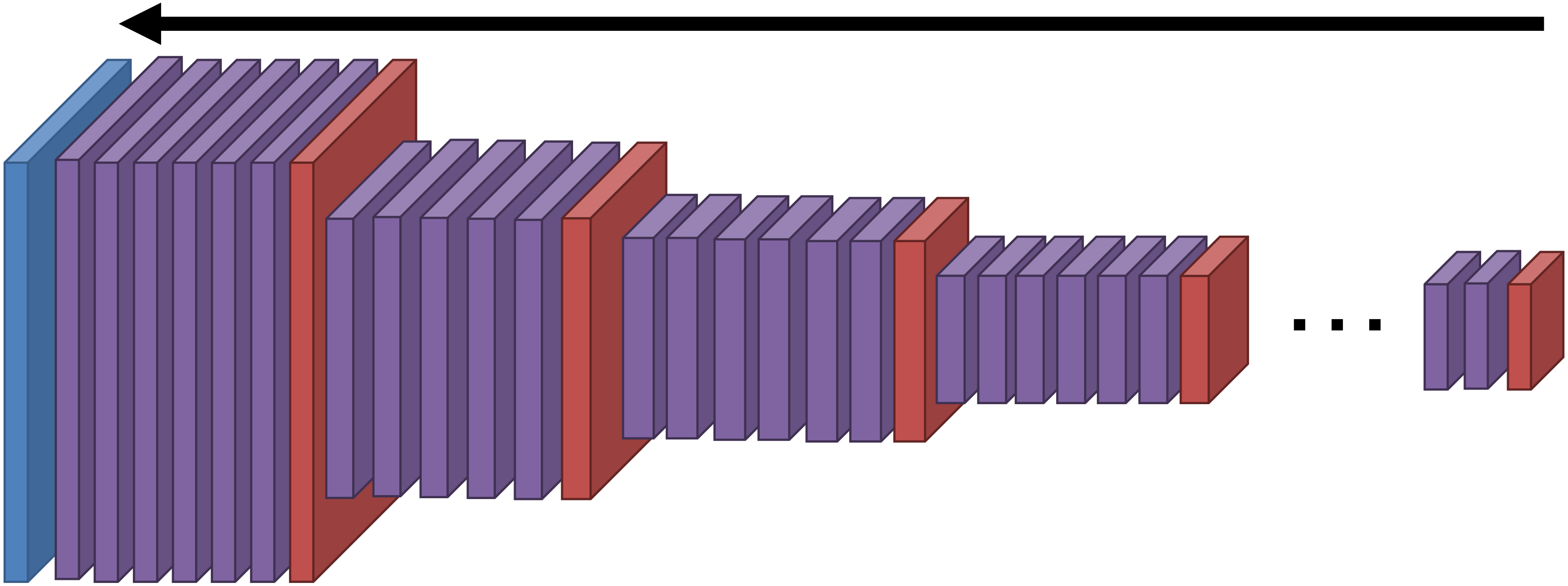
Backprop

Every backpropagation step multiplies the gradient by the local gradient

$1 *d * d * d \ldots * d = d^{n-1}$

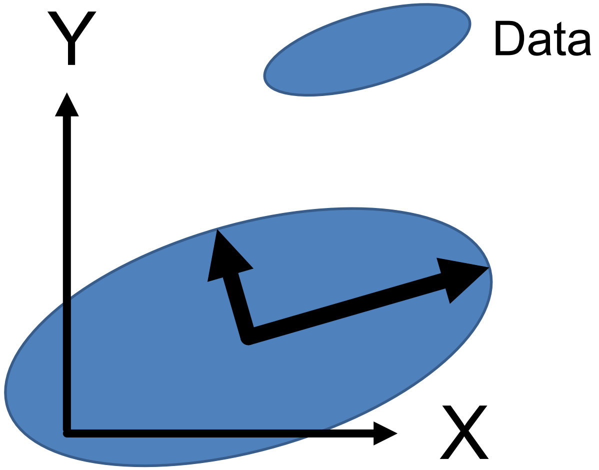
What if d >> 1, n big?

Exploding Gradients

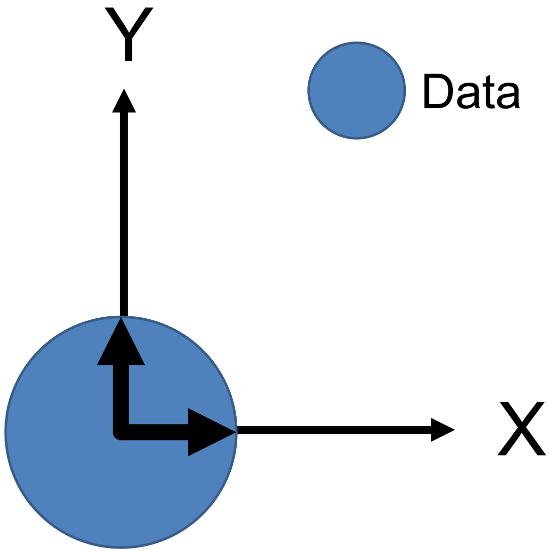


Solution 1 – Batch Normalization

Learning algorithms work far better when data looks like the right as opposed to the left

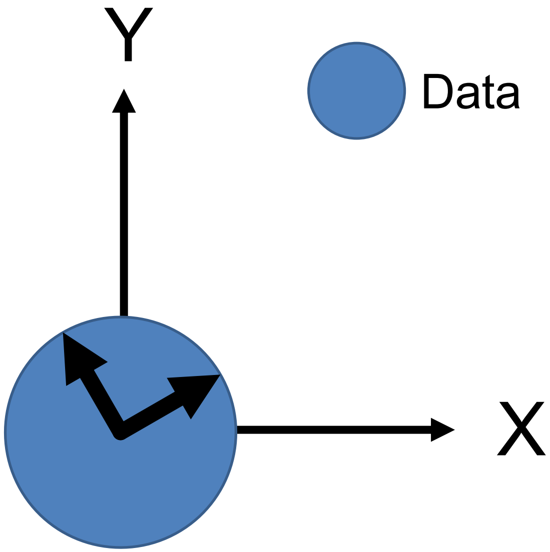


Mean(x) != Mean(Y) != 0
Var(x) != Var(y) != 0
Cov(x,y) != 0



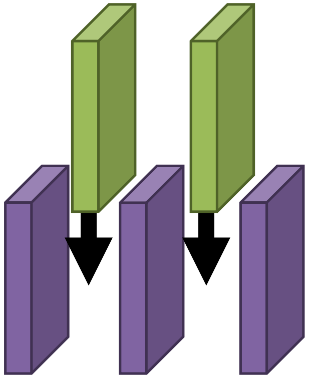
Mean(x) = Mean(Y) = 0
Var(x) = Var(y) = 1
Cov(x,y) = 0

Solution 1 – Batch Normalization



Mean(x) = Mean(Y) = 0
Var(x) = Var(y) = 1

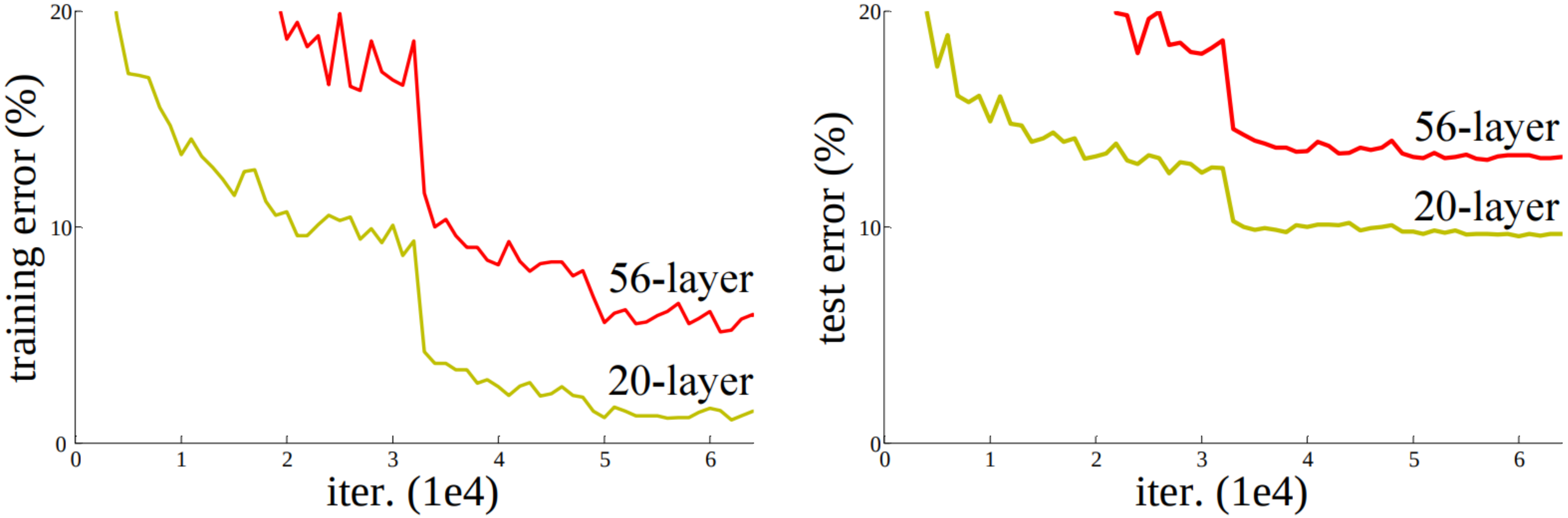
Idea: make layer (**Batch Norm**) that normalizes things going through it based on estimates of Var(x_i) in each batch.
Stick in between **other layers**



S. Ioffe and C. Szegedy. Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift.

There exists vs. We Can Find

- Still can’t **fit** models to the data: **Deeper model** fits worse than **shallower model** on the training data.
- **There exists a deeper model that’s identical to the shallow model. Why?**

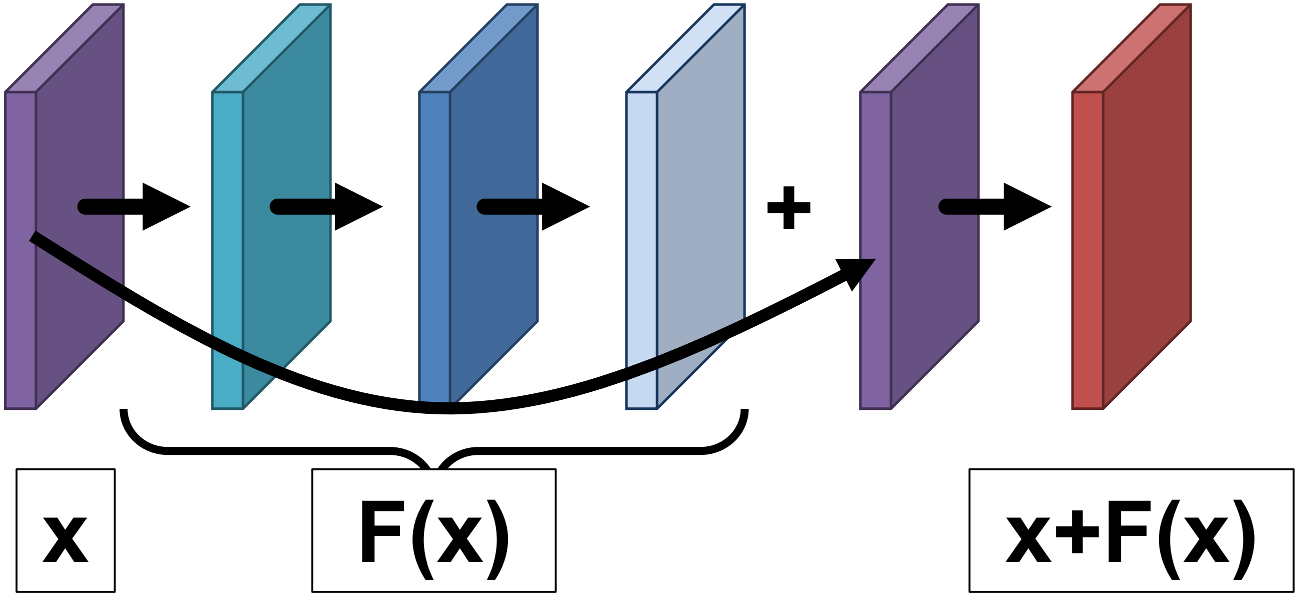


K. He et al. *Deep Residual Learning for Image Recognition.* CVPR 2016

Residual Learning

New Building Block: $x + F(x)$

Lets you train networks with 100s of layers.



Evaluating Results

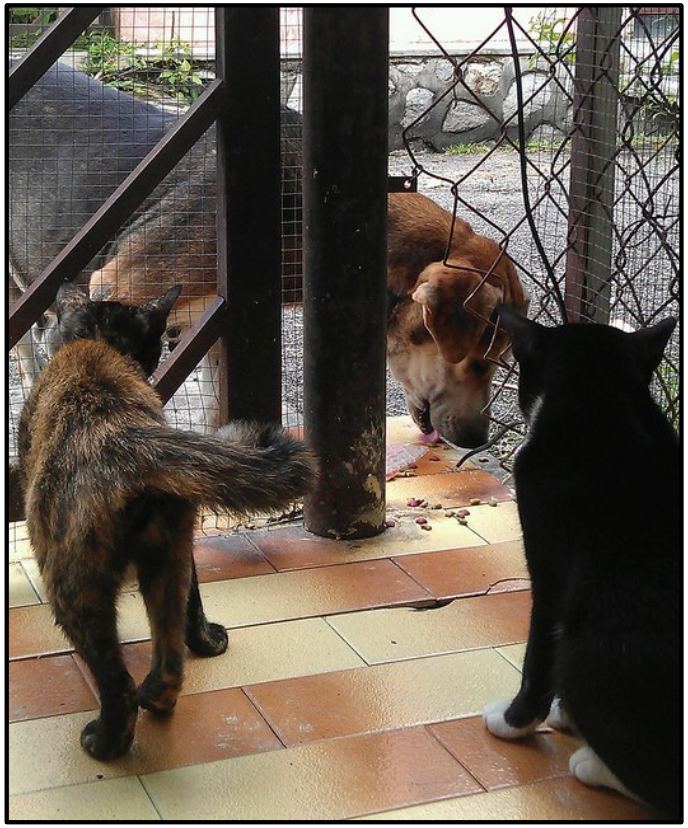
At training time, we minimize: $-\log\left(\frac{\exp((Wx)_{y_i}}{\sum_k \exp((Wx)_k))}\right)$

At test time, we evaluate, given predicted class $\widehat{y_i}$:

Accuracy: $$\frac{1}{n}\sum_{i=1}^{n} 1(y_i = \widehat{y_i})$$

Evaluating Many Categories

Does this image depict a cat or a dog?



To avoid penalizing ambiguous images, many challenges let you make five guesses (top-5 accuracy):

Your prediction is correct if one of the guesses is right.

Image credit: Coco dataset

Accuracy over the Years

	Top 1 Error	Top 5 Error
Best Pre-Deep	-	26.2%*
Alexnet	43.5%	20.9%
VGG-16	28.4%	9.6%
+Batch Norm	26.6%	8.5%
Resnet-152	21.7%	5.9%
Human*	-	5.1%

A Practical Aside

- People usually use hardware specialized for matrix multiplies (the card below does 13.4T flops if it's matrix multiplies).
- The real answer to why we love homogeneous coordinates?
  - Makes rendering matrix multiplies →
  - leads to matrix multiplication hardware →
  - deep learning.

Training a CNN

- Download a big dataset
- Initialize network weights randomly
- for epoch in range(epochs):
    - Shuffle dataset
    - for each minibatch in datsaet.:
        - Put data on GPU
        - Compute gradient
        - Update gradient with SGD

Training a CNN from Scratch

Need to start **w** somewhere

- AlexNet: weights ~ Normal(0,0.01), bias = 1
- “Xavier” initialization: Uniform($\frac{-1}{\sqrt{n}}, \frac{1}{\sqrt{n}}$) where n is the number of neurons
- “Kaiming” initialization: Normal(0,$\sqrt{2/n}$)

Take-home: important, but use defaults

Training a ConvNet

- Convnets typically have millions of parameters:
    - AlexNet: 62 million
    - VGG16: 138 million
- Convnets typically fit on ~1.2 million images
- Remember least squares: if we have fewer data points than parameters, we’re in trouble
- Solution: need regularization / more data

Training a CNN – Weight Decay

SGD Update

$$\boldsymbol{w_{t+1}} = \boldsymbol{w_t} - \epsilon \frac{\partial L}{\partial \boldsymbol{w_t}}$$

+Weight Decay

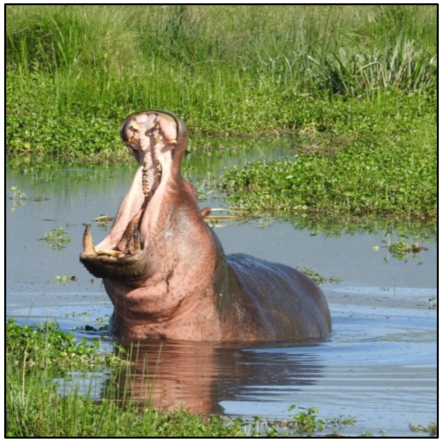
$$\boldsymbol{w_{t+1}} = \boldsymbol{w_t} - \eta\epsilon\boldsymbol{w_t} + \epsilon \frac{\partial L}{\partial \boldsymbol{w_t}}$$

What does this remind you of?

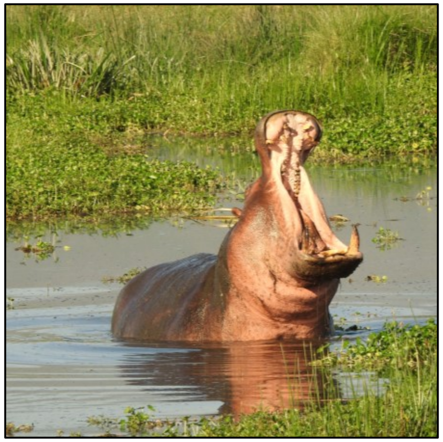
Weight decay is very similar to regularization but might not be the same for more complex optimization techniques.

Quick Quiz

Raise your hand if it’s a hippo



Horizontal Flip



Color Jitter

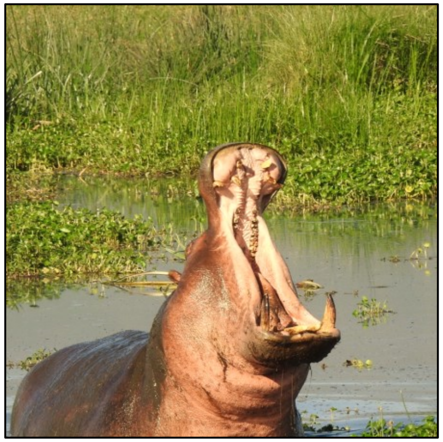
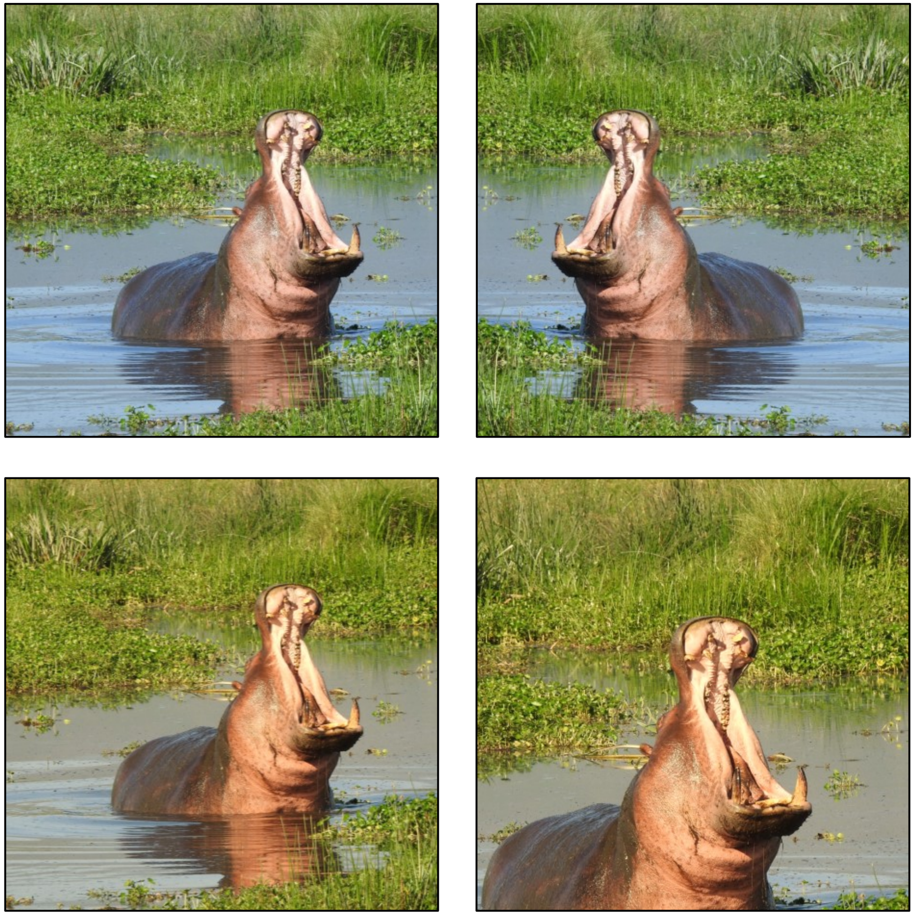


Image Cropping

Training a CNN –Augmentation

- Apply transformations that don’t affect the output
- Produces more data but you have to be careful that it doesn’t change the meaning of the output



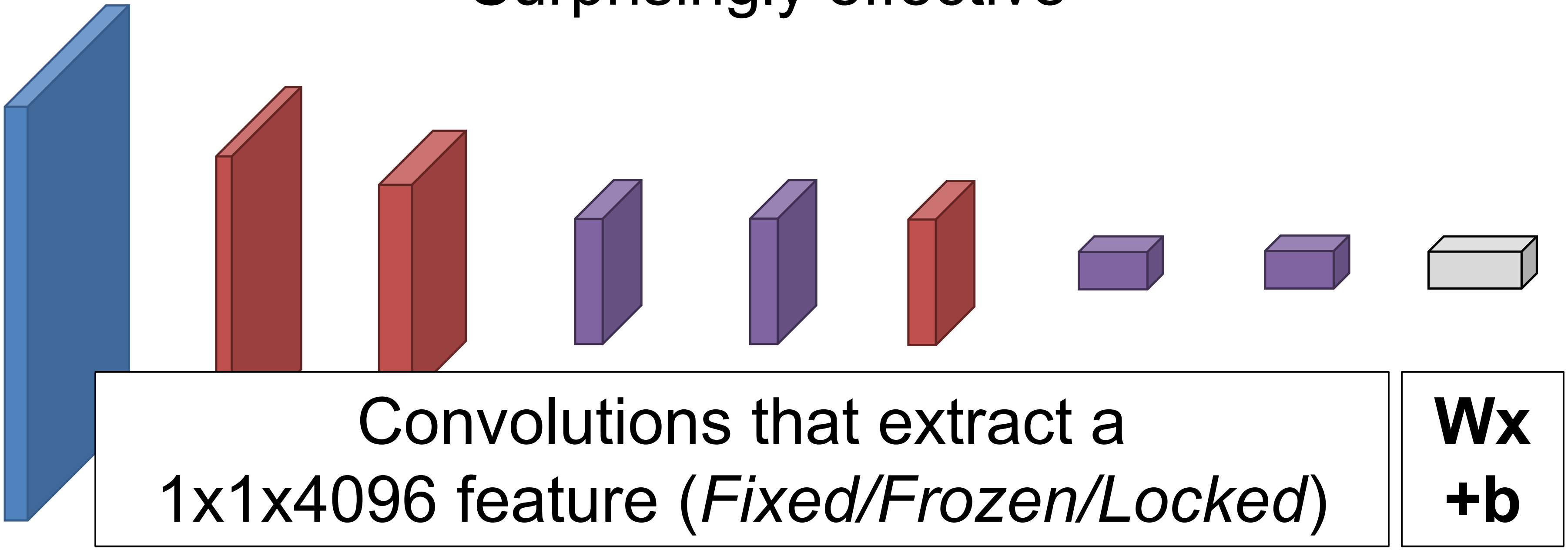
Training a CNN – Fine-tuning

- What if you don’t have data?

Fine-Tuning: Pre-trained Features

1. Extract some layer from an existing network
2. Use as your new feature.
3. Learn a linear model.

Surprisingly effective



Convolutions that extract a 1x1x4096 feature (*Fixed/Frozen/Locked*)

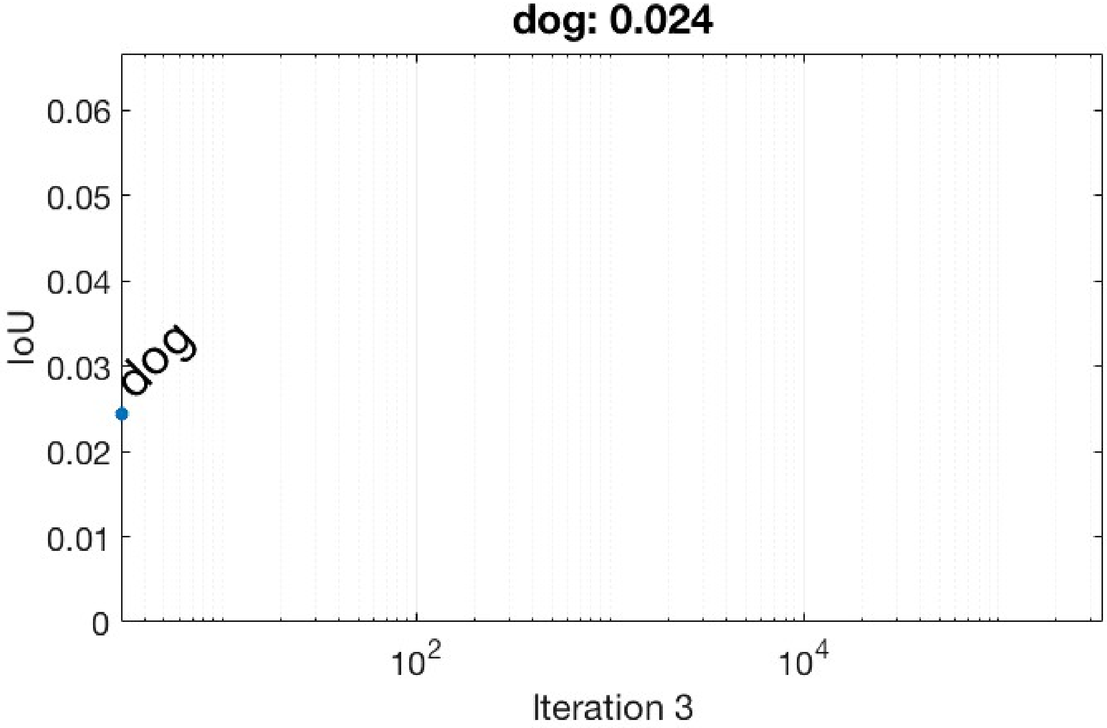
Wx +b

Fine-Tuning: Transfer Learning

- Rather than initialize from random weights, initialize from some “pre-trained” model that does something else.
- Most common model is trained on ImageNet.
- Other pretraining tasks exist but are less popular.

Fine-Tuning: Transfer Learning

Why should this work?
Transferring from objects (dog) to scenes (waterfall)



Bau and Zhou et al. Network Dissection: Quantifying Interpretability of Deep Visual Representations. CVPR 2017.

Recommendations

- <10K images: features
- **Always** try fine-tuning
- >100K images: consider trying from scratch

Summary

- We learned about converting an image into a vector output (e.g., which of K classes is this image, or predict K continuous outputs)
- We learned about some building blocks for doing this