Data Mining Techniques

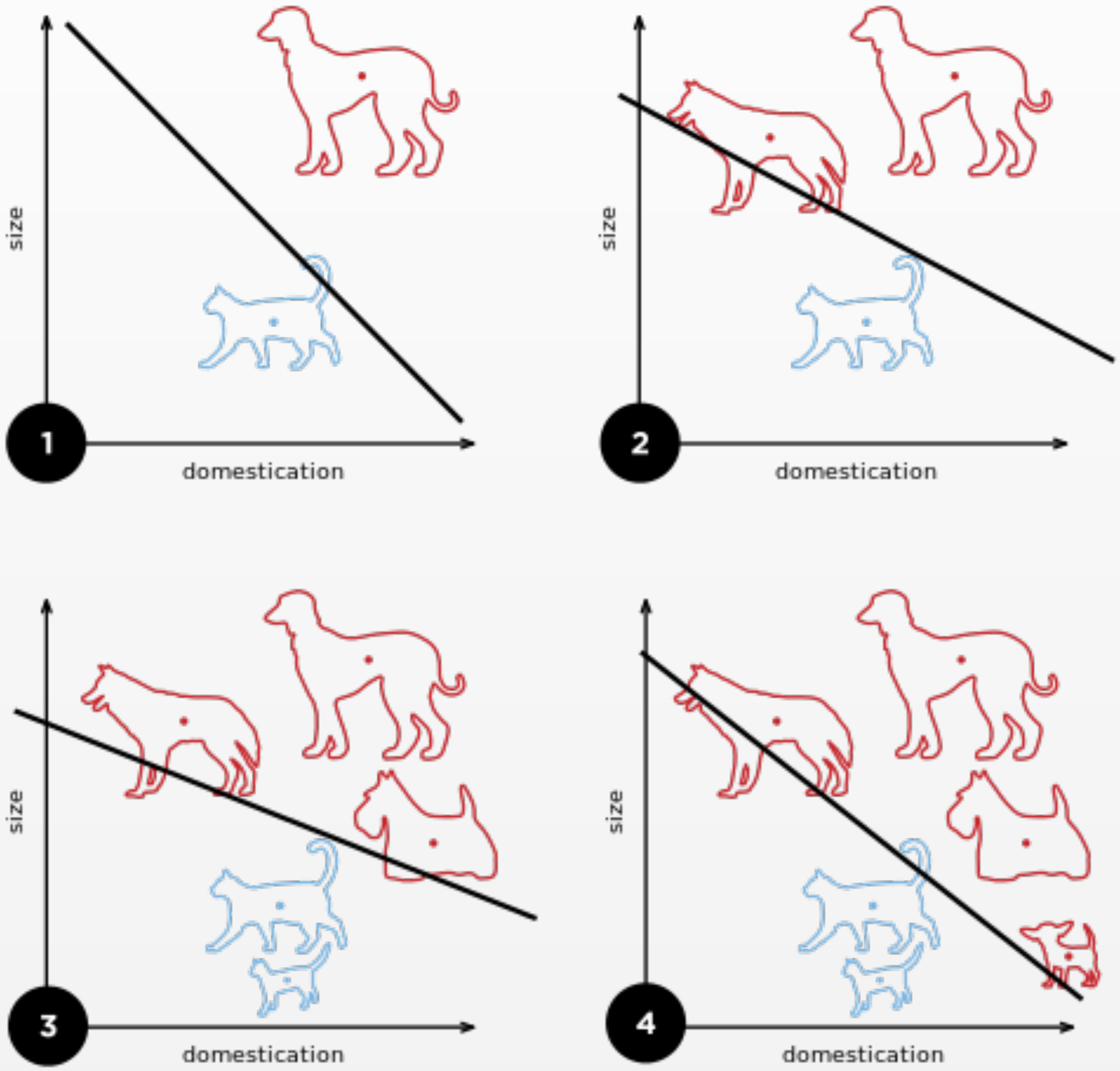
CS 6220 - Section 3 - Fall 2016

Lecture 20: Deep Learning

Jan-Willem van de Meent
(*credit:* CS 231n)

NORTHEASTERN UNIVERSITY
LVX VERITAS VIRTVS
FOUNDED A.D. 1898
BOSTON, MASSACHUSETTS

Perceptron

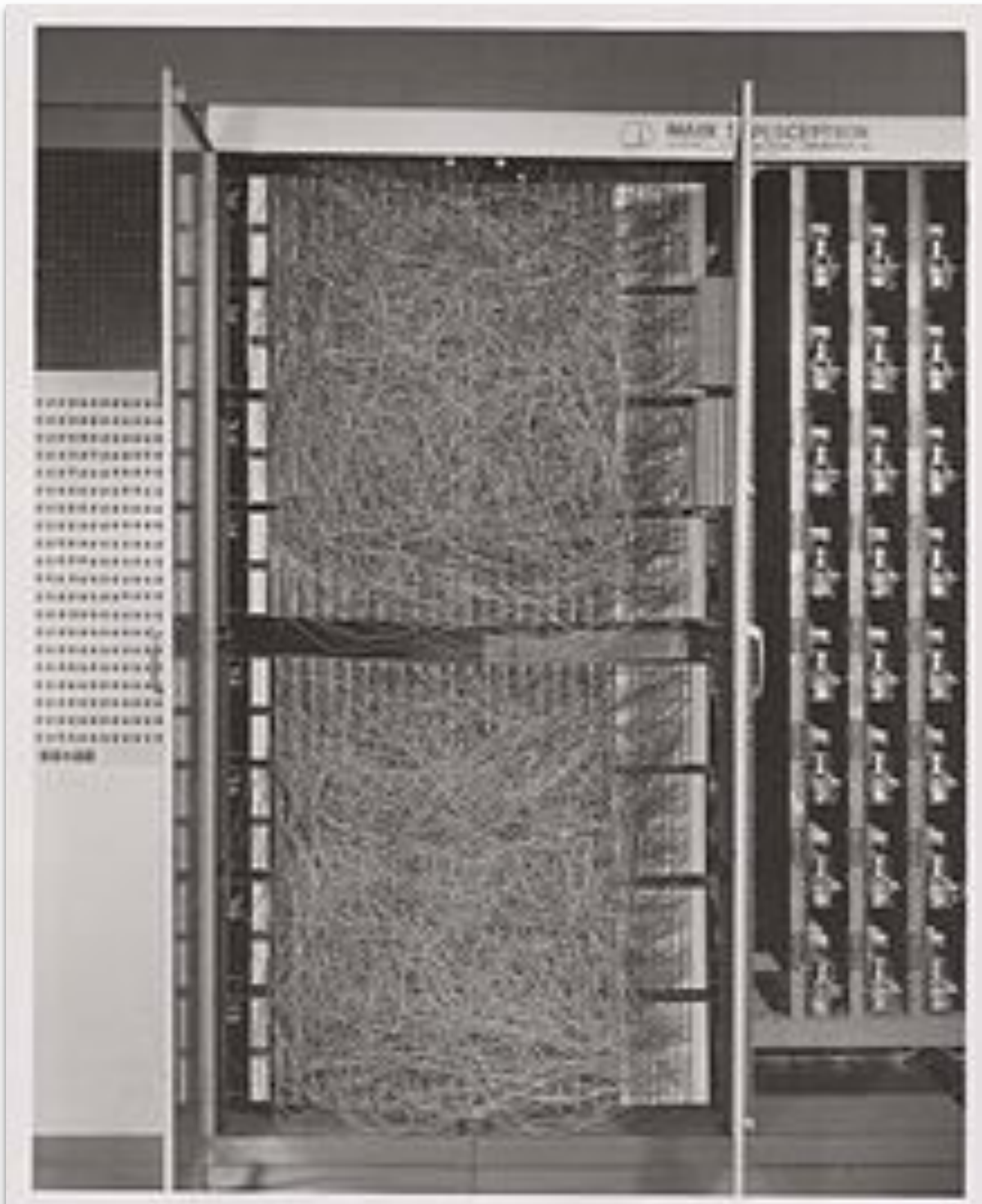


$$y = f\left(\sum_i w_i x_i + b\right)$$

$$f(z) = \begin{cases} 1 & z > 0 \\ 0 & z \leq 0 \end{cases}$$

Simple classifier:
"Linear regression + Sign"

Perceptron



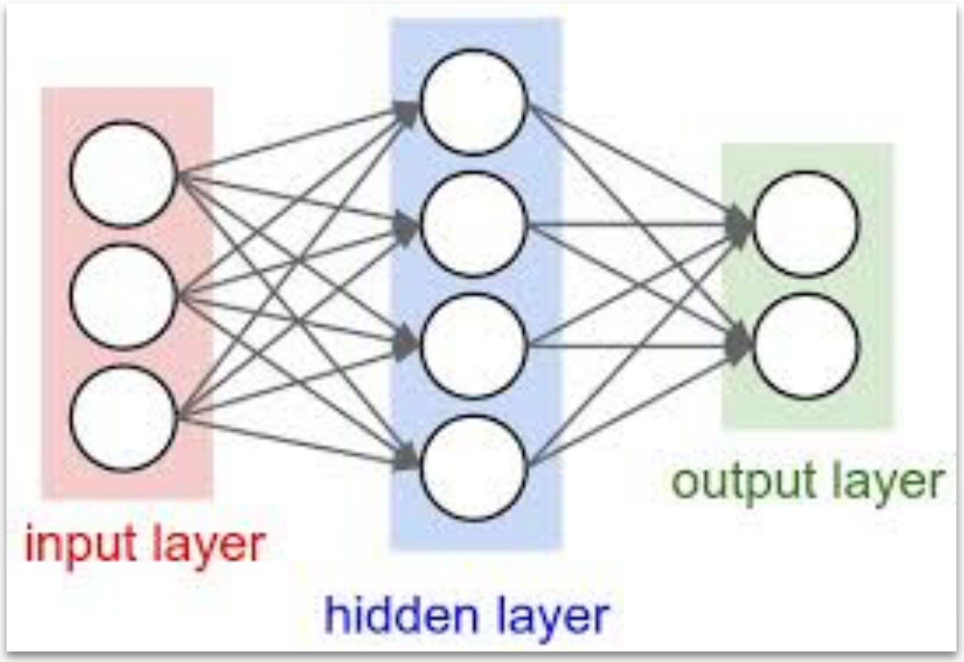
https://en.wikipedia.org/wiki/Perceptron

$$y = f\left(\sum_i w_i x_i + b\right)$$

$$f(z) = \begin{cases} 1 & z > 0 \\ 0 & z \leq 0 \end{cases}$$

Mark I Perceptron. Used 20×20 cadmium sulfide photocells to produce a 400-pixel image.

Multi-layer Perceptron



Add a “hidden” layer

$$y_j = f\left(\sum_i W_{ji}^{(2)} h_i + b_j^{(2)}\right)$$

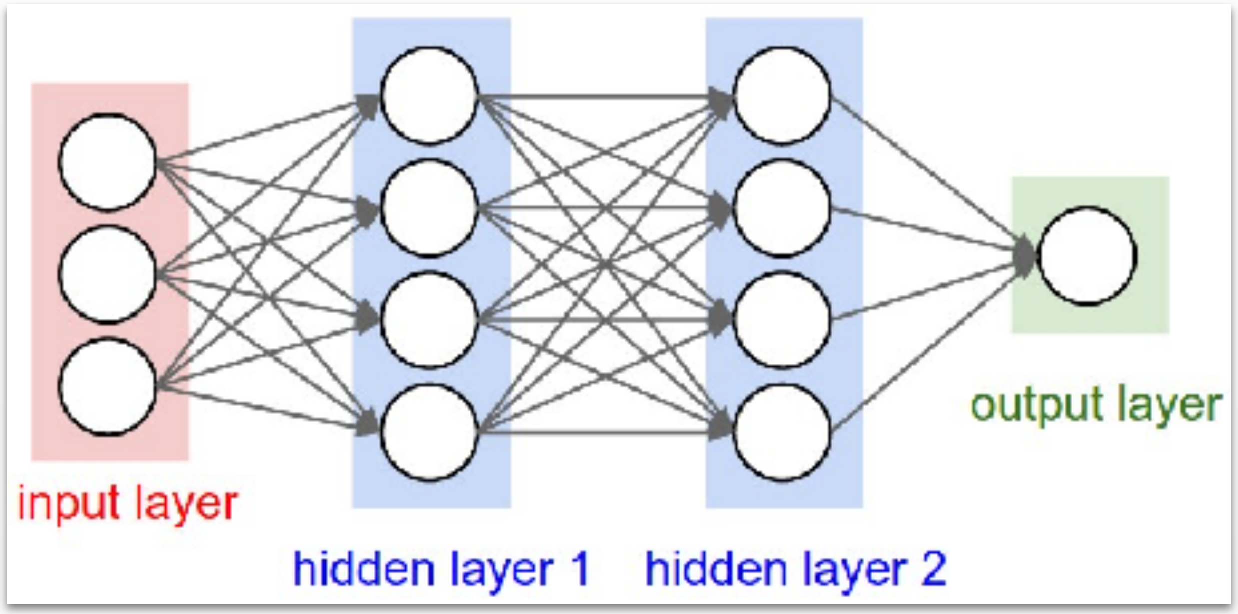
$$h_j = f\left(\sum_i W_{ji}^{(1)} x_i + b_j^{(1)}\right)$$

Basically like “stacked” logistic regression

Sigmoid Activation

$$\begin{aligned} f(z) &= \sigma(z) \\ &= 1/(1+e^{-z}) \end{aligned}$$

Multi-layer Perceptron



Multiple Hidden Layers

$$y_j = f\left(\sum_i W_{ji}^{(y)} h_i^{(N)} + b_j^{(y)}\right)$$

$$h_j^{(n)} = f\left(\sum_i W_{ji}^{(n)} h_i^{(n-1)} + b_j^{(n)}\right)$$

$$h_j^{(1)} = f\left(\sum_I W_{ji}^{(1)} x_i + b_j^{(1)}\right)$$

Number of parameters
in this example:

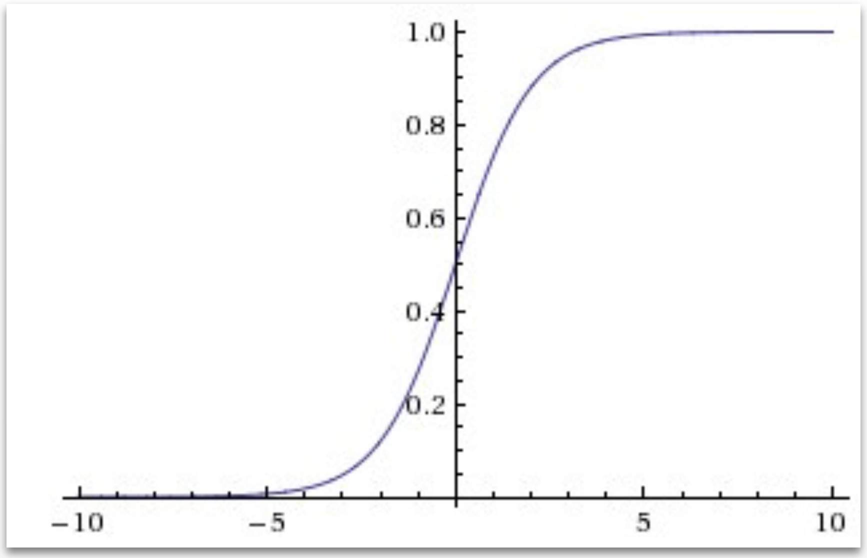
4*(3+1) + 4*(4+1) + 1*(4+1)
= 41

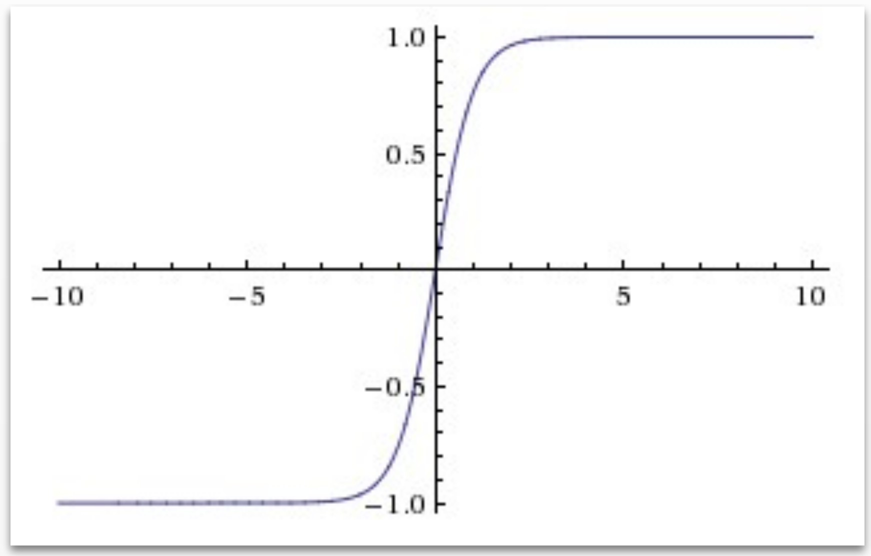
(quadratic in layer size)

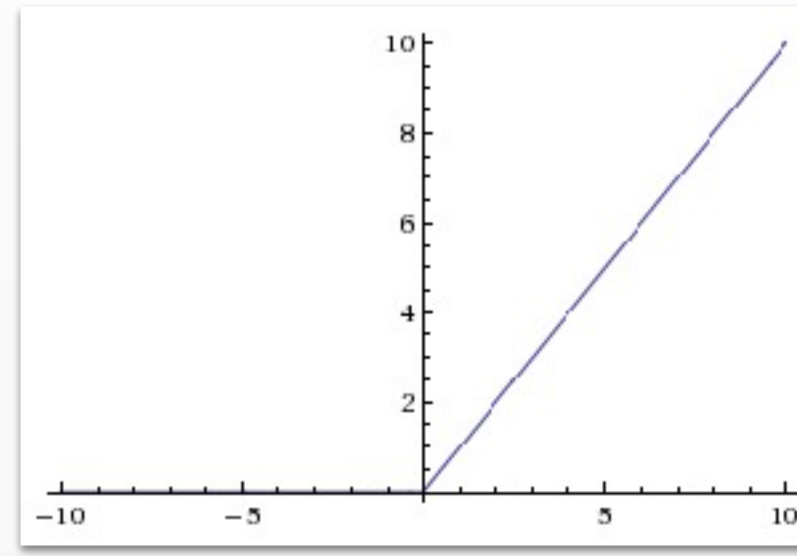
Sigmoid Activation

$$f(z) = \sigma(z)$$
$$= 1/(1+e^{-z})$$

Activation Functions







Sigmoid Activation

$$f(z) = \sigma(z)$$
$$= 1/(1+e^{-z})$$

Tanh Activation

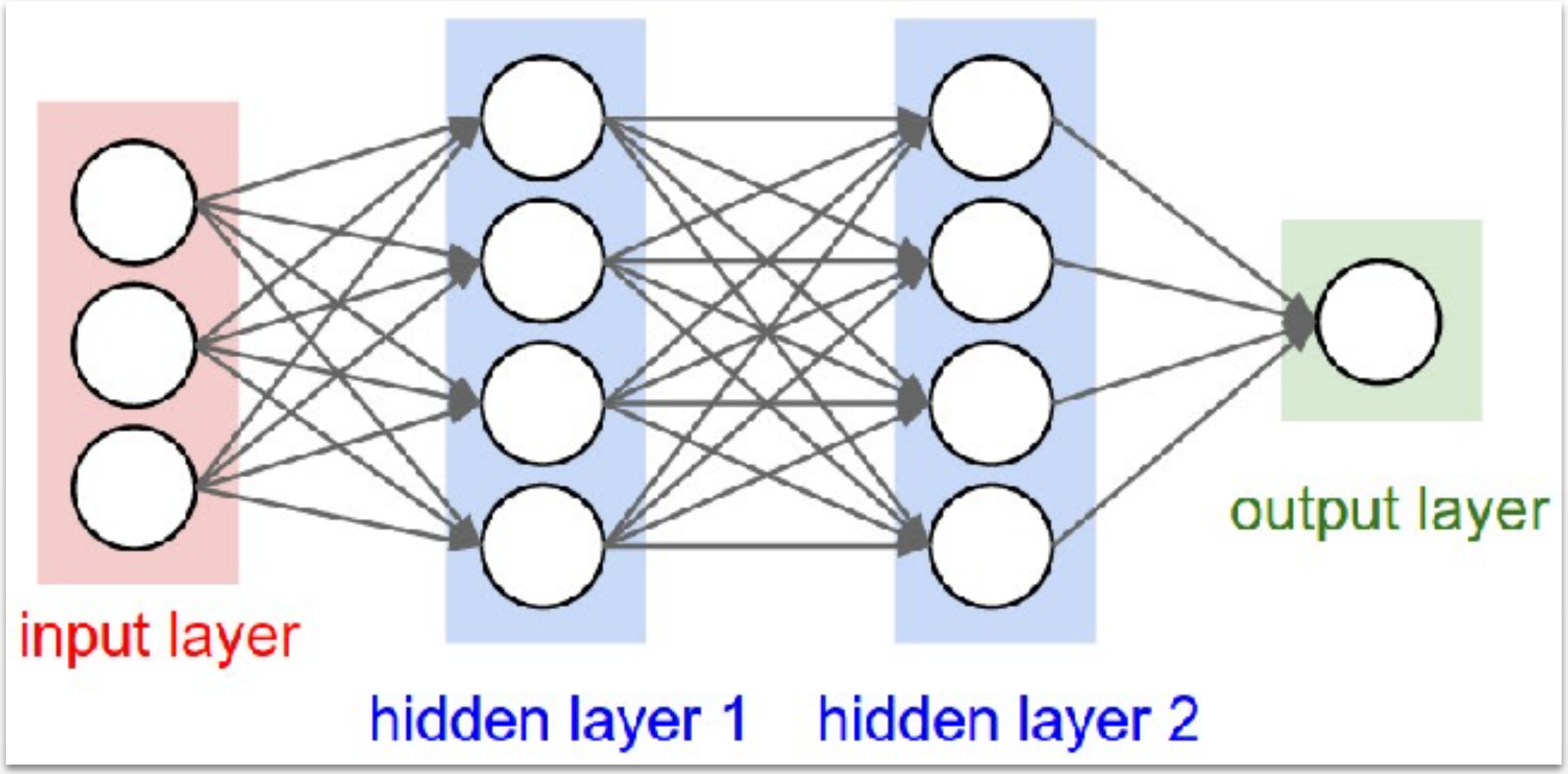
$$f(z) = \tanh(z)$$

ReLU Activation

$$f(z) = \max\{0, z\}$$

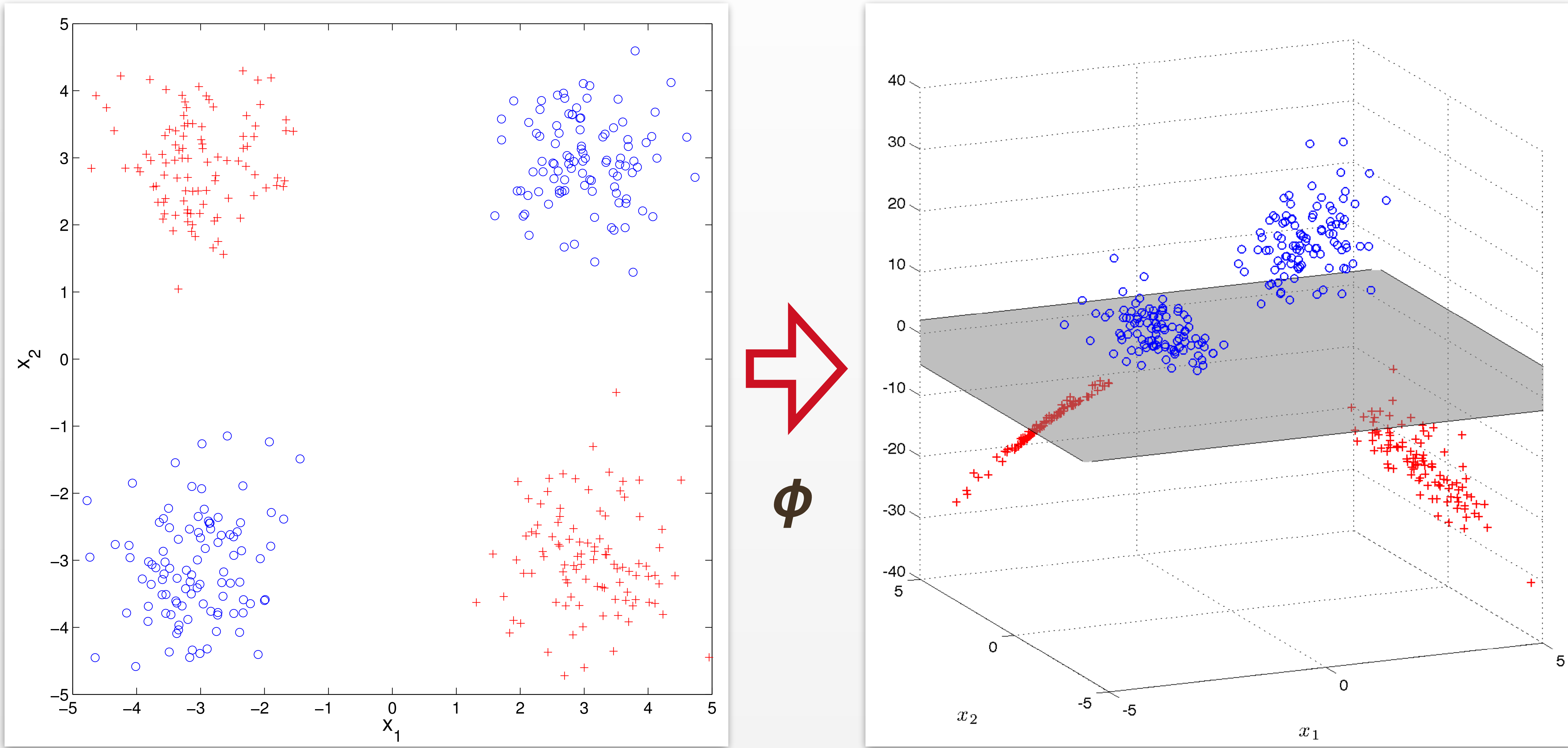
"Rectified linear units"

Some Basic Questions



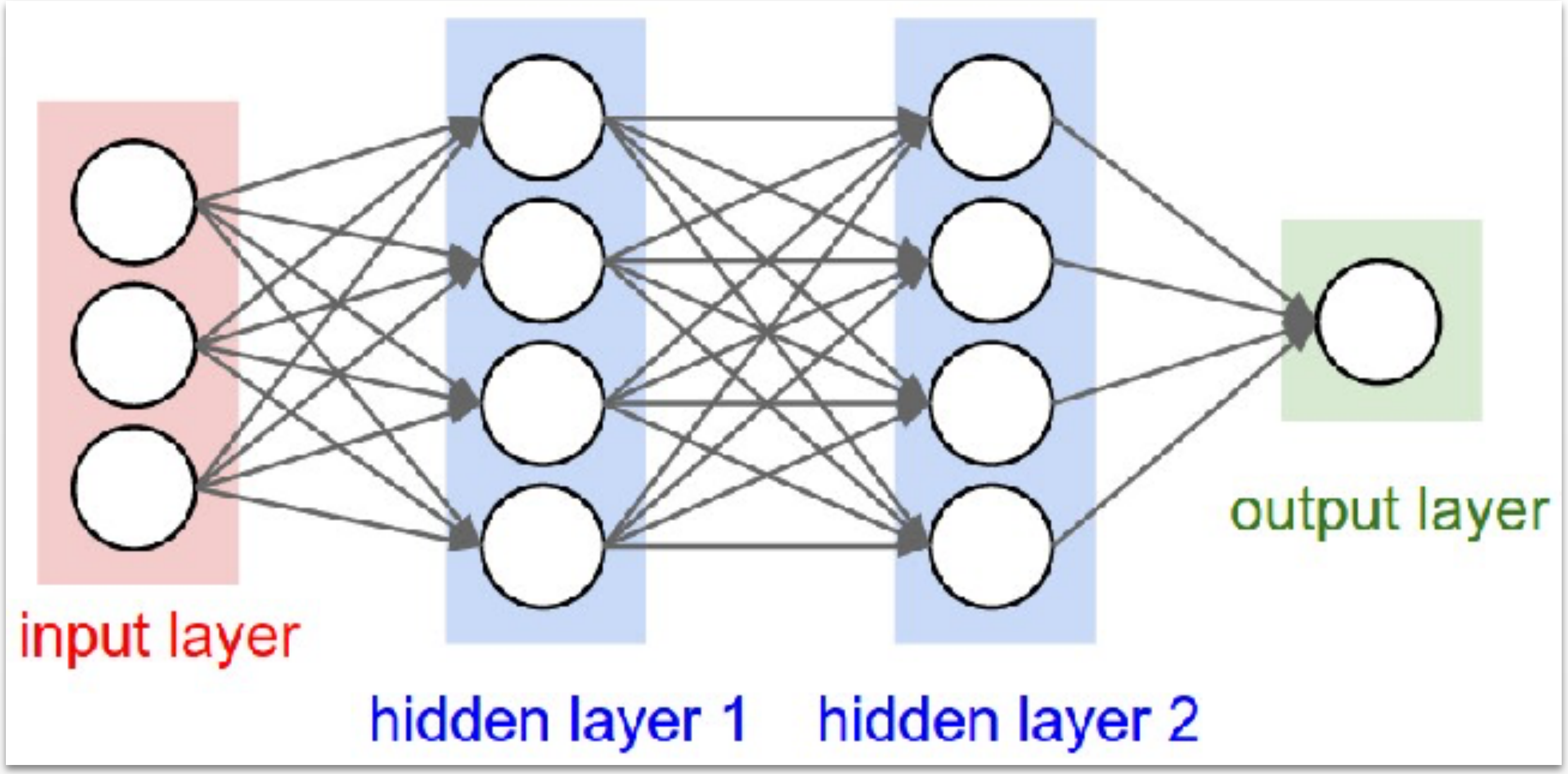
1. Why might this be a good idea?
2. How can we learn the parameters?

Reminder: Feature Maps



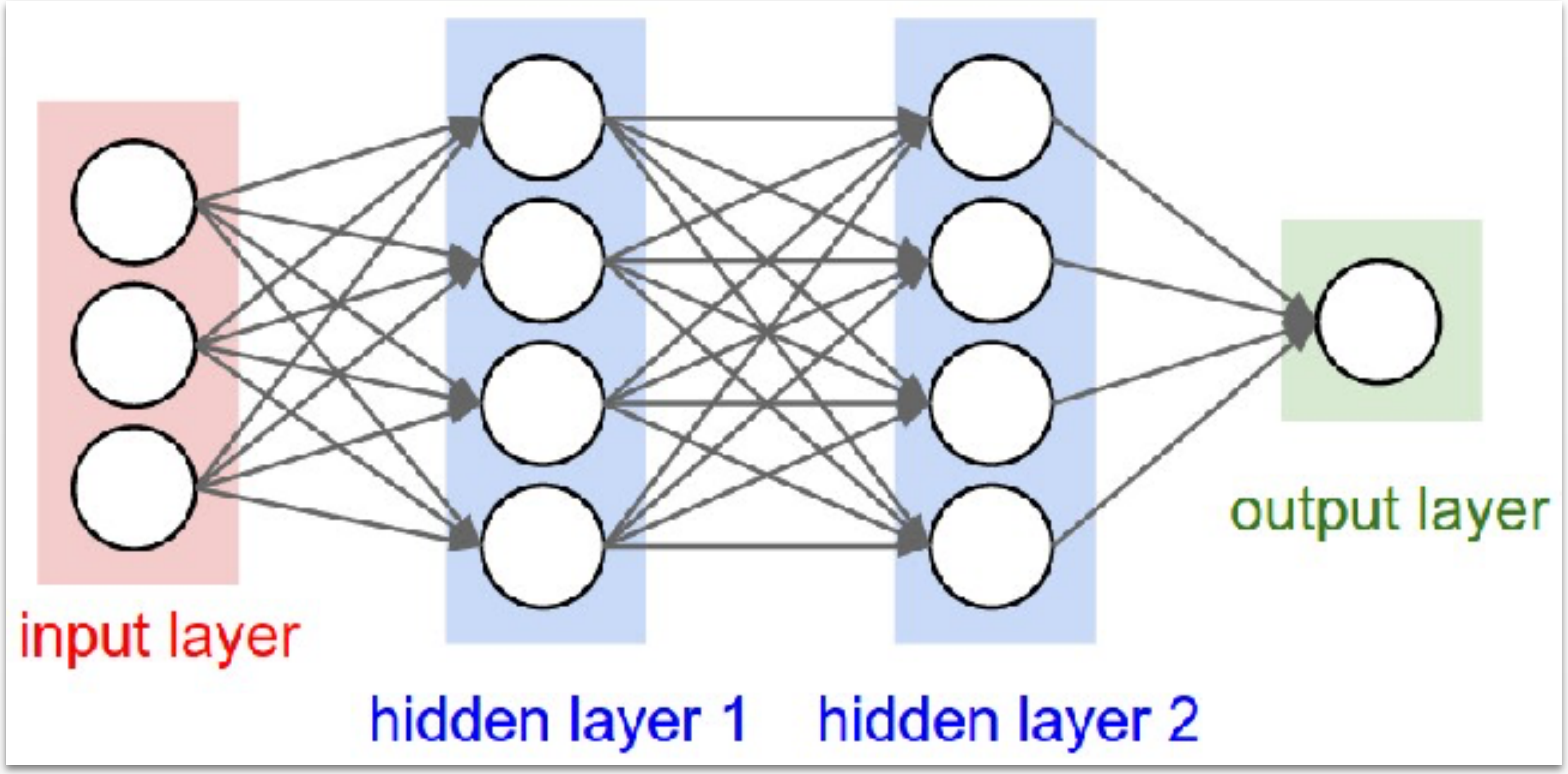
$$\phi(x) = \begin{bmatrix} x_1 & x_2 & x_1 x_2 \end{bmatrix} \in \mathbb{R}^3$$

Learning Feature Maps



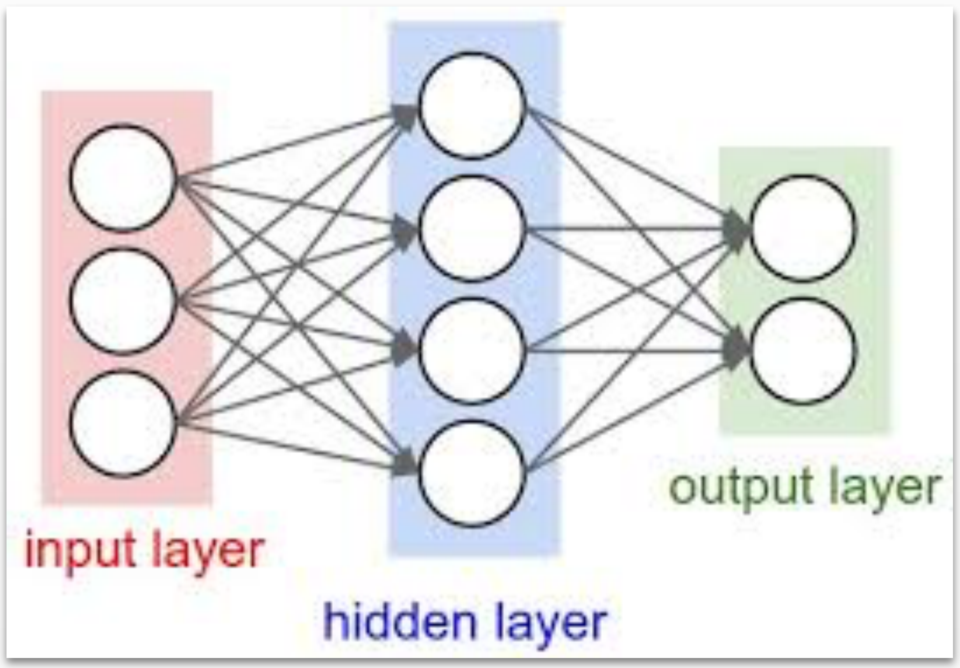
- Make hidden layer wider than inputs
- “Learn” feature representation
  (by training parameters)
- Use multiple layers to learn “abstractions”

Training Neural Nets



- Define a loss function on output
  (e.g. regularized squared/logistic/hinge loss)
- Calculate gradients of loss w.r.t. weights
- Perform stochastic gradient descent

Back-propagation
(a.k.a. applying the chain rule)



1-Layer Perceptron

$$y_j = f\left(\sum_i W_{ji}^{(2)} h_i + b_j^{(2)}\right)$$

$$h_j = f\left(\sum_i W_{ji}^{(1)} x_i + b_j^{(1)}\right)$$

$$\frac{\partial \mathcal{L}}{b_k^{(1)}} = \sum_i \frac{\partial \mathcal{L}}{y_i} \sum_j \frac{\partial y_i}{\partial h_k} \frac{\partial h_k}{b_k^{(1)}} + \frac{\partial \mathcal{L}}{\partial b_k^{(1)}}$$

Now minimize loss using Stochastic Gradient Descent

Reminder: Stochastic Gradient Descent

Batch gradient descent (evaluates all data)

$$\boldsymbol{w}_t = \boldsymbol{w}_{t-1} - \alpha_t \nabla_{\boldsymbol{w}} E(\boldsymbol{y}; \boldsymbol{w})|_{\boldsymbol{w}=\boldsymbol{w}_{t-1}}$$

Minibatch gradient descent (evaluates subset)

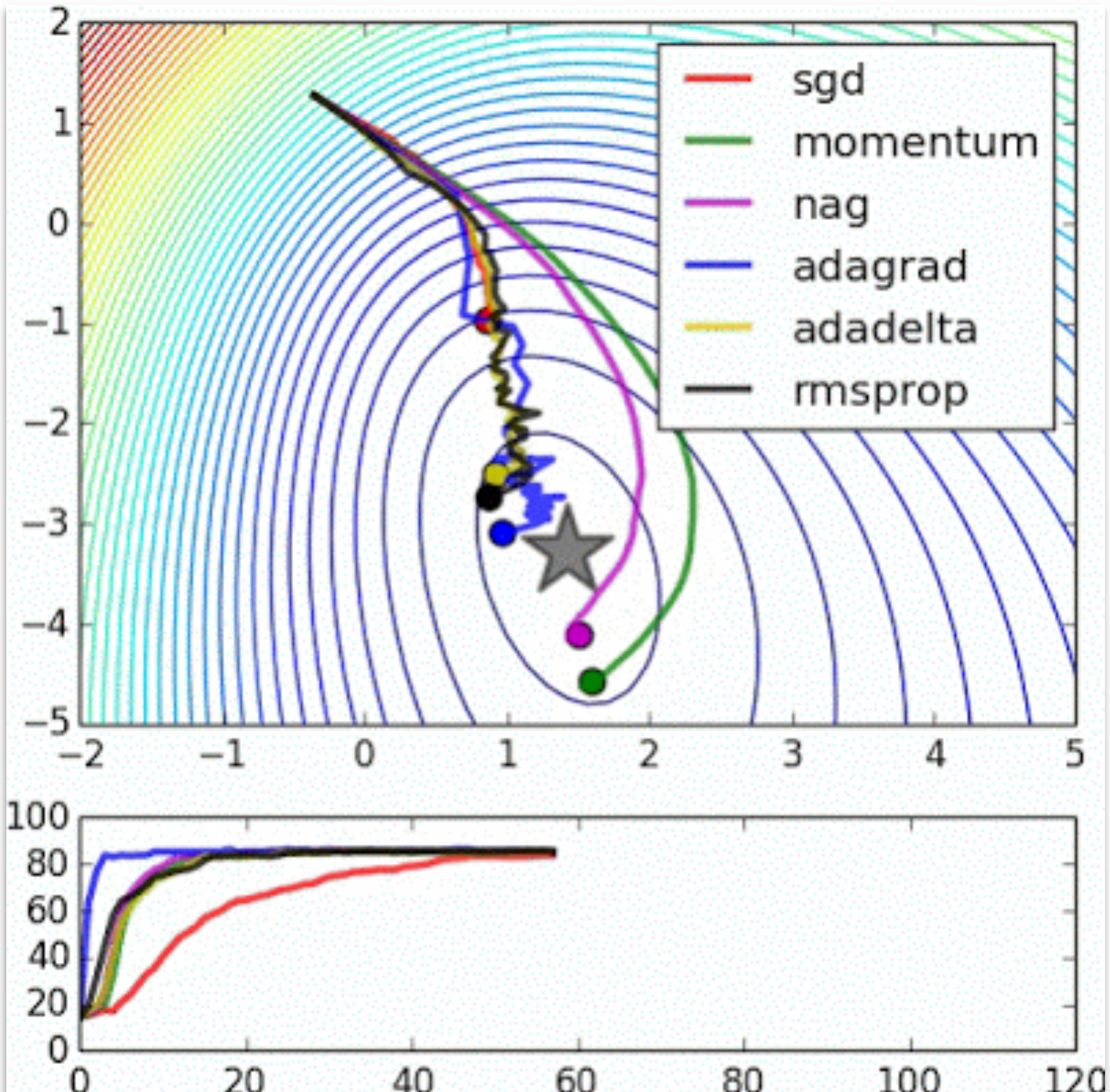
$$\boldsymbol{w}_t = \boldsymbol{w}_{t-1} - \alpha_t \nabla_{\boldsymbol{w}} E(\boldsymbol{y}_t; \boldsymbol{w})|_{\boldsymbol{w}=\boldsymbol{w}_{t-1}} \qquad \boldsymbol{y}_t \subset \boldsymbol{y}$$

Converges under Robbins-Monro conditions

$$\sum_{t=1}^{\infty} \alpha_t = \infty \qquad \sum_{t=1}^{\infty} \alpha_t^2 < \infty$$

$$\alpha_t = \frac{\alpha_0}{(\tau + t)^{\kappa}}$$

Improvements on SGD



credit: Alec Radford

Nesterov Momentum Update

$$v_t = \mu v_{t-1} - \alpha_t \nabla f(\theta_{t-1} + \mu v_{t-1})$$
$$\theta_t = \theta_{t-1} + v_t$$

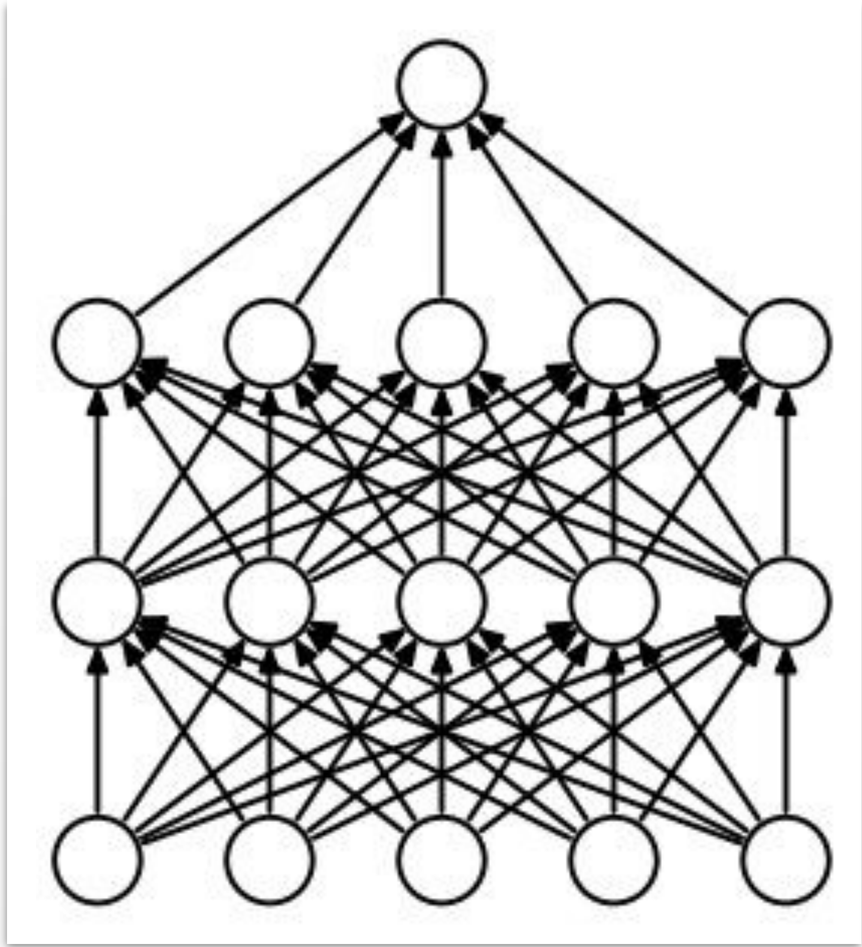
AdaGrad Update

$$g_t = \nabla f(\theta_t)$$
$$G_t = G_{t-1} + \mathrm{diag}\left[g_{t-1} g_{t-1}^\top\right]$$
$$\theta_t = \theta_{t-1} - \alpha_t \frac{g_{t-1}}{G_t^{1/2} + \epsilon}$$

- Momentum/NAG: Average gradients over multiple steps
- Adagrad/RMSprop: Approximate inverse of Hessian

(adapted from: http://cs231n.stanford.edu)

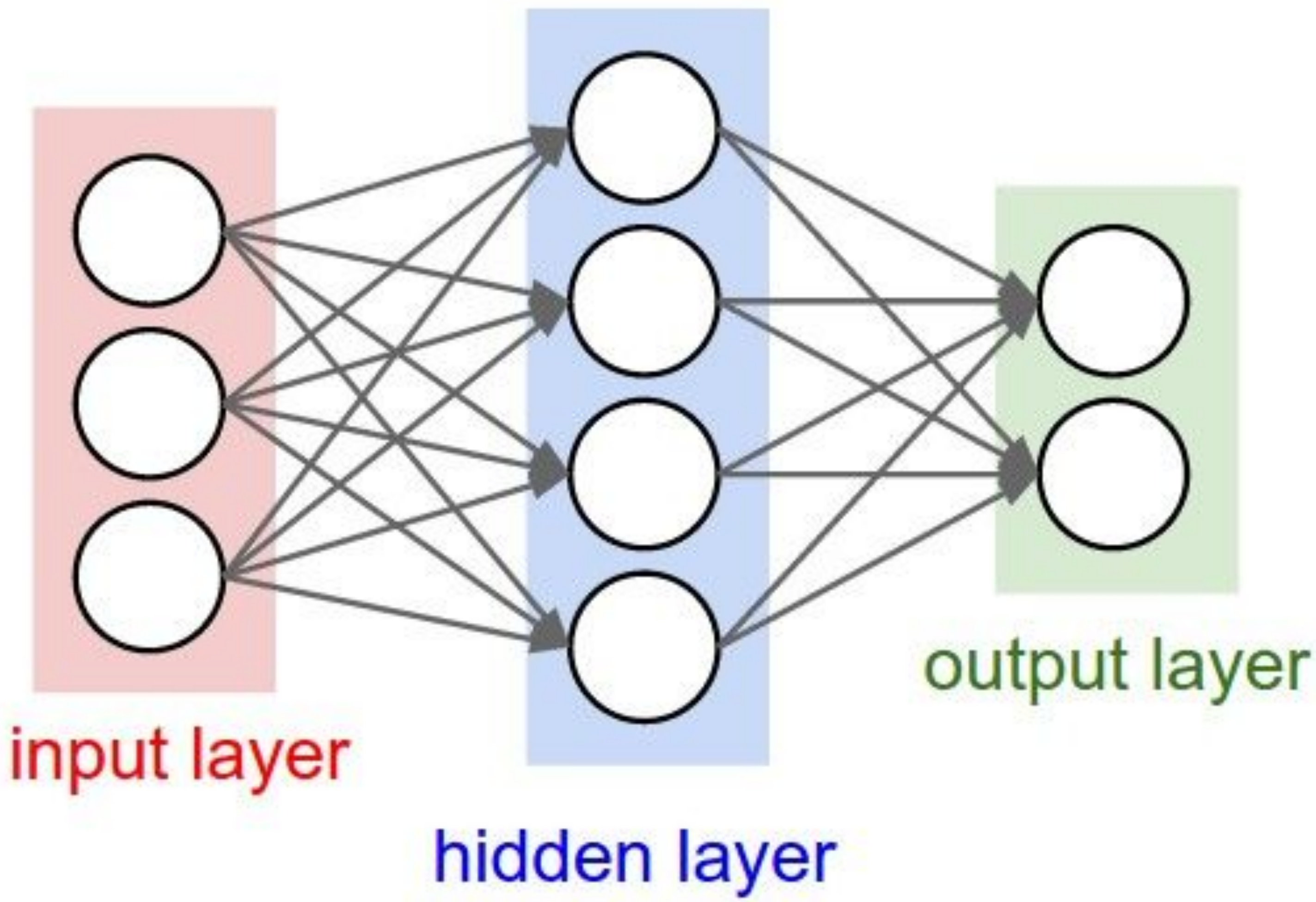
Challenges in Training



- SGD local optimum is sensitive to initialization method for weights
- The gradient signal may be too noisy to learn from for deeper layers

Weight initialization

- Q: what happens when W=0 init is used?



(adapted from: http://cs231n.stanford.edu)

Weight initialization

- First idea: **Small random numbers**

(gaussian with zero mean and 1e-2 standard deviation)

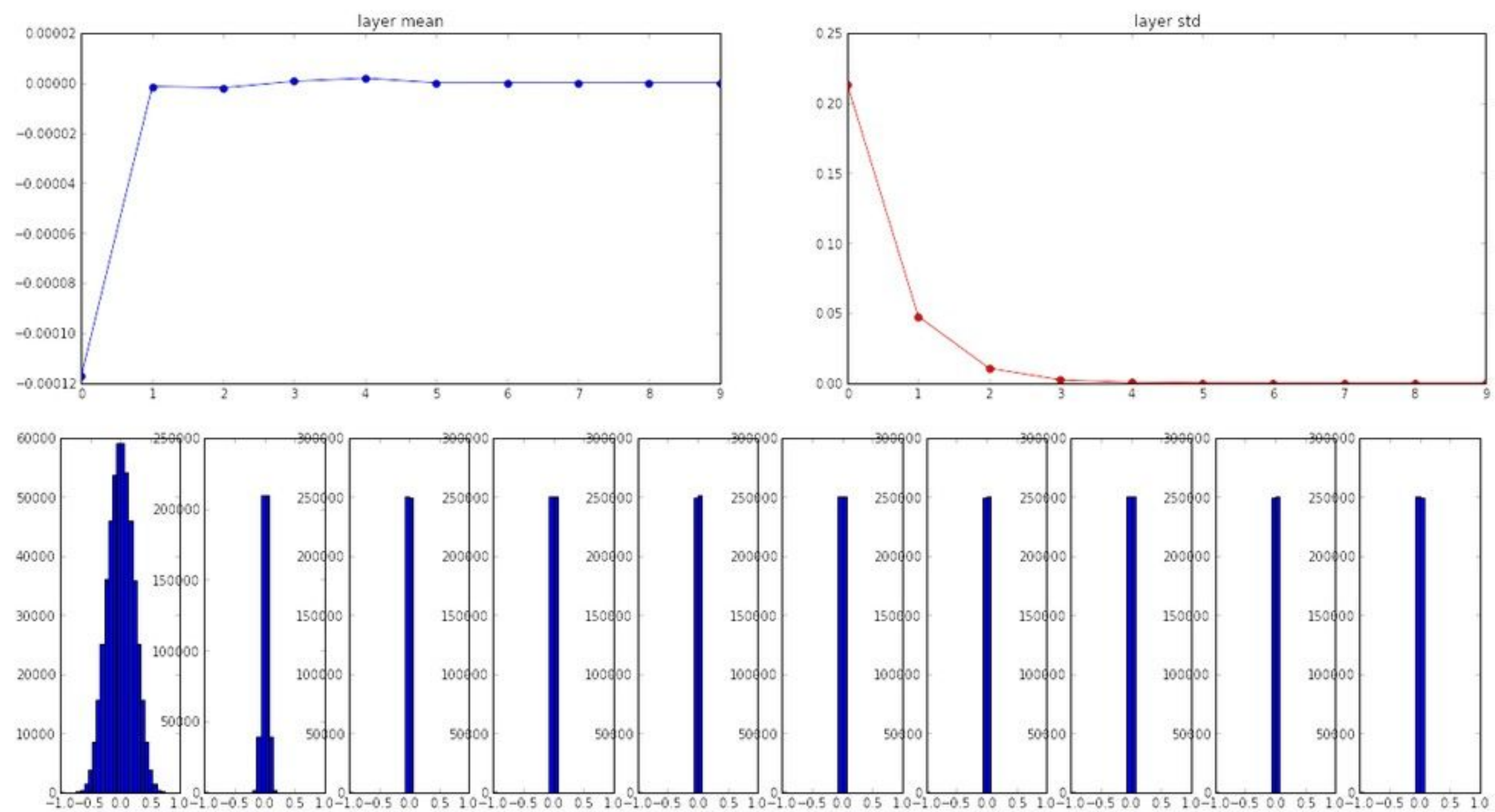
```
W = 0.01* np.random.randn(D,H)
```

Works ~okay for small networks, but can lead to non-homogeneous distributions of activations across the layers of a network.

(adapted from: http://cs231n.stanford.edu)

Weight initialization

```
input layer had mean 0.000927 and std 0.998388
hidden layer 1 had mean -0.000117 and std 0.213081
hidden layer 2 had mean -0.000001 and std 0.047551
hidden layer 3 had mean -0.000002 and std 0.010630
hidden layer 4 had mean 0.000001 and std 0.002378
hidden layer 5 had mean 0.000002 and std 0.000532
hidden layer 6 had mean -0.000000 and std 0.000119
hidden layer 7 had mean 0.000000 and std 0.000026
hidden layer 8 had mean -0.000000 and std 0.000006
hidden layer 9 had mean 0.000000 and std 0.000001
hidden layer 10 had mean -0.000000 and std 0.000000
```



All activations become zero!

Q: think about the backward pass. What do the gradients look like?

Hint: think about backward pass for a W*X gate.

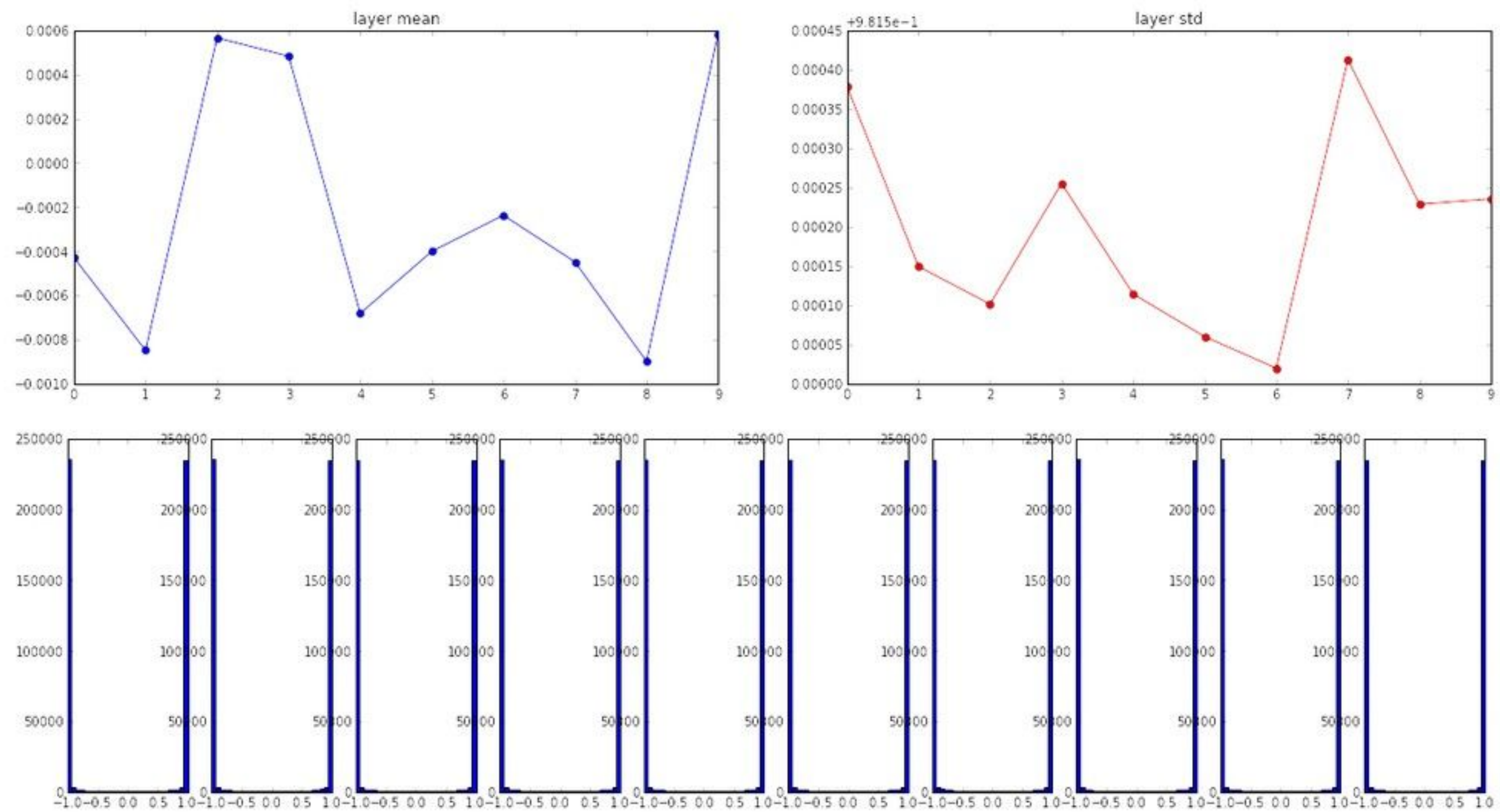
(adapted from: http://cs231n.stanford.edu)

Weight initialization

```
W = np.random.randn(fan_in, fan_out) * 1.0 # layer initialization
```

*1.0 instead of *0.01

```
input layer had mean 0.001800 and std 1.001311
hidden layer 1 had mean -0.000430 and std 0.981879
hidden layer 2 had mean -0.000849 and std 0.981649
hidden layer 3 had mean 0.000566 and std 0.981601
hidden layer 4 had mean 0.000483 and std 0.981755
hidden layer 5 had mean -0.000682 and std 0.981614
hidden layer 6 had mean -0.000401 and std 0.981560
hidden layer 7 had mean -0.000237 and std 0.981520
hidden layer 8 had mean -0.000448 and std 0.981913
hidden layer 9 had mean -0.000899 and std 0.981728
hidden layer 10 had mean 0.000584 and std 0.981736
```



Almost all neurons completely saturated, either -1 and 1. Gradients will be all zero.

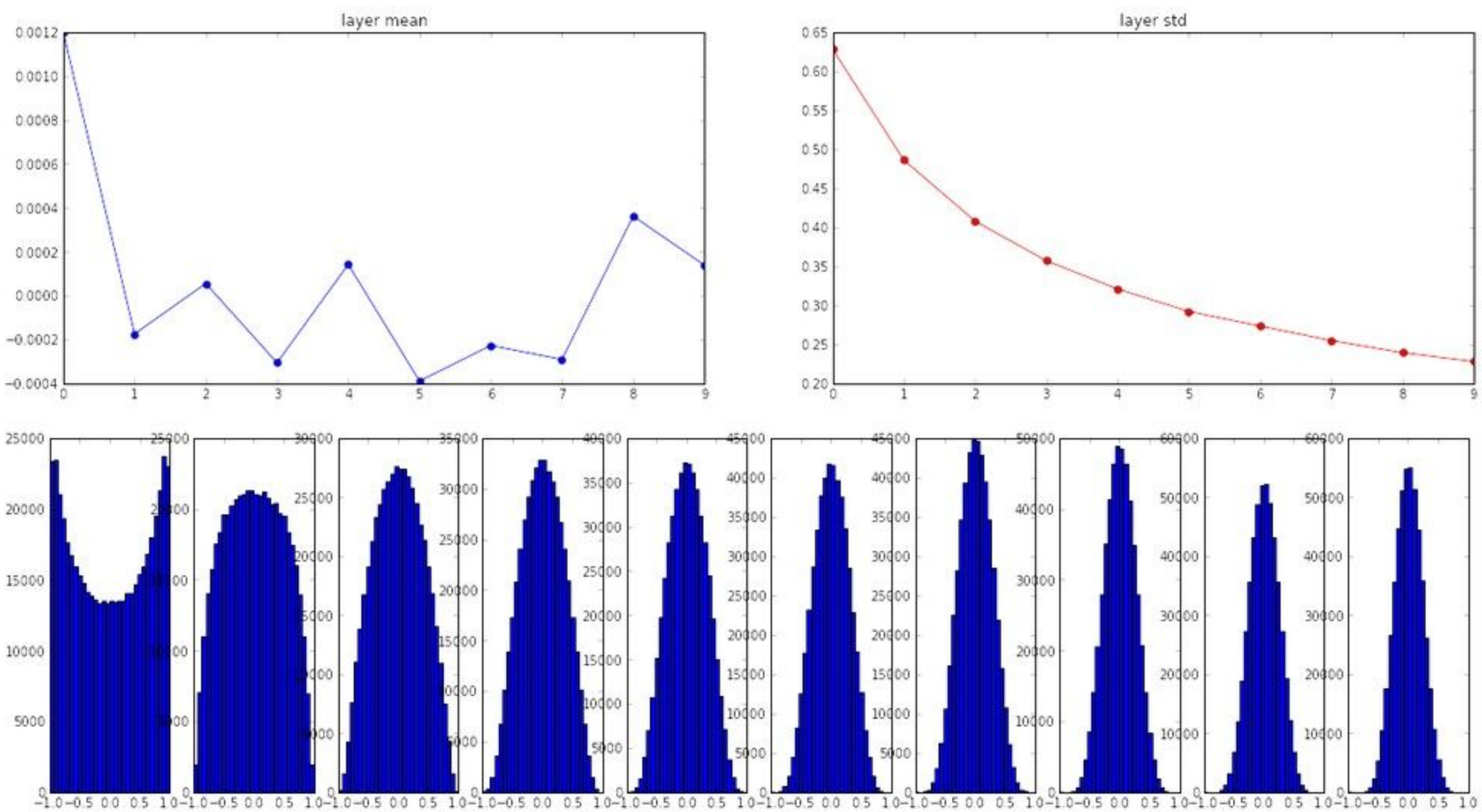
(adapted from: http://cs231n.stanford.edu)

Weight initialization

```
input layer had mean 0.001800 and std 1.001311
hidden layer 1 had mean 0.001198 and std 0.627953
hidden layer 2 had mean -0.000175 and std 0.486051
hidden layer 3 had mean 0.000055 and std 0.407723
hidden layer 4 had mean -0.000306 and std 0.357108
hidden layer 5 had mean 0.000142 and std 0.320917
hidden layer 6 had mean -0.000389 and std 0.292116
hidden layer 7 had mean -0.000228 and std 0.273387
hidden layer 8 had mean -0.000291 and std 0.254935
hidden layer 9 had mean 0.000361 and std 0.239266
hidden layer 10 had mean 0.000139 and std 0.228008
```

```
W = np.random.randn(fan_in, fan_out) / np.sqrt(fan_in) # layer initialization
```

“Xavier initialization”
[Glorot et al., 2010]



Reasonable initialization.
(Mathematical derivation assumes linear activations)

Size of weight proportional to square root of number inputs

(adapted from: http://cs231n.stanford.edu)

Batch Normalization [Ioffe & Szegedy 2015]

"you want unit gaussian activations? just make them so."

consider a batch of activations at some layer.
To make each dimension unit gaussian, apply:

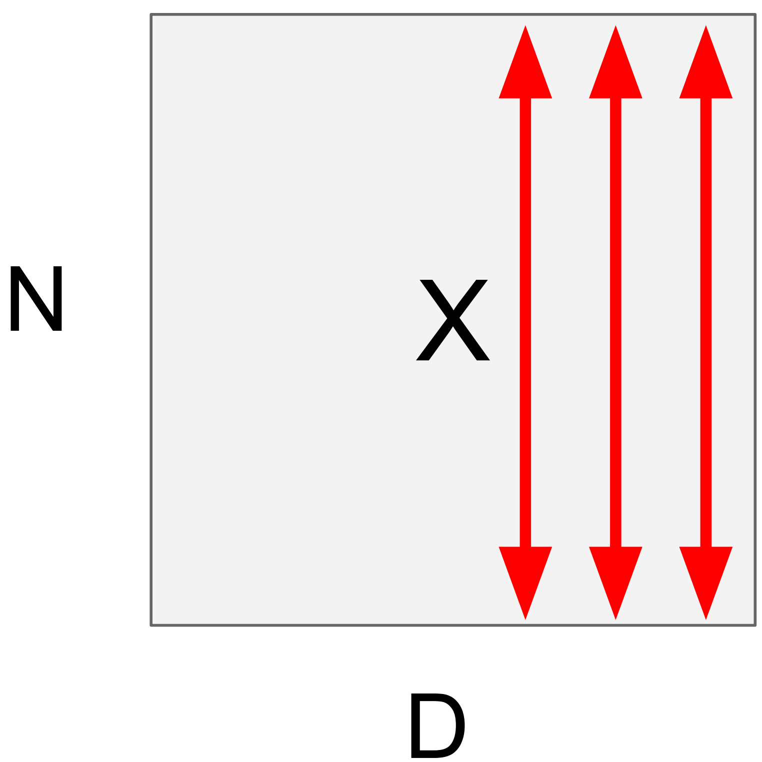
$$\widehat{x}^{(k)} = \frac{x^{(k)} - \mathrm{E}[x^{(k)}]}{\sqrt{\mathrm{Var}[x^{(k)}]}}$$

this is a vanilla differentiable function...

(adapted from: http://cs231n.stanford.edu)

Batch Normalization [Ioffe & Szegedy 2015]

"you want unit gaussian activations?
just make them so."



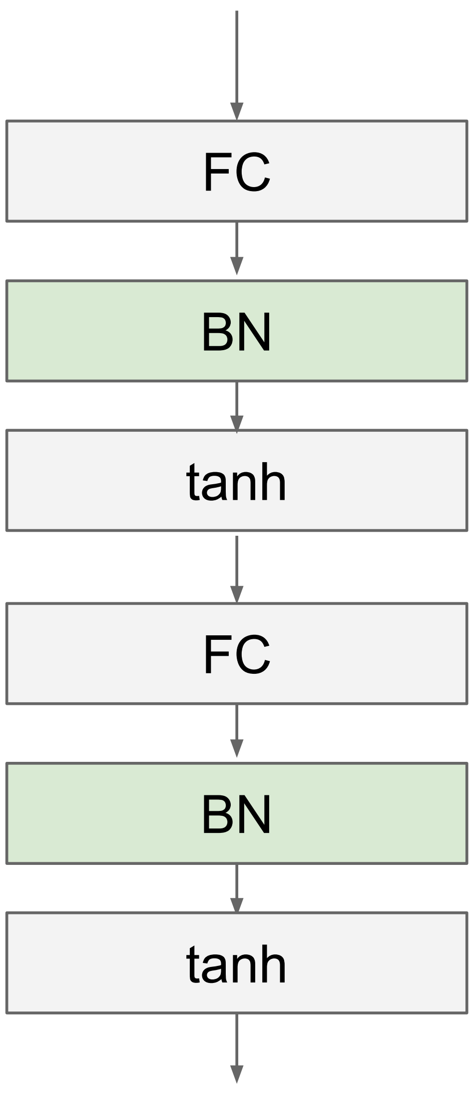
1. compute the empirical mean and variance independently for each dimension.

2. Normalize

$$\widehat{x}^{(k)} = \frac{x^{(k)} - \mathrm{E}[x^{(k)}]}{\sqrt{\mathrm{Var}[x^{(k)}]}}$$

(adapted from: http://cs231n.stanford.edu)

Batch Normalization [Ioffe & Szegedy 2015]



Usually inserted after Fully Connected / (or Convolutional, as we'll see soon) layers, and before nonlinearity.

Problem: do we necessarily want a unit gaussian input to a tanh layer?

$$\widehat{x}^{(k)} = \frac{x^{(k)} - \mathrm{E}[x^{(k)}]}{\sqrt{\mathrm{Var}[x^{(k)}]}}$$

(adapted from: http://cs231n.stanford.edu)

Batch Normalization [Ioffe & Szegedy 2015]

Normalize:

$$\widehat{x}^{(k)} = \frac{x^{(k)} - \mathrm{E}[x^{(k)}]}{\sqrt{\mathrm{Var}[x^{(k)}]}}$$

And then allow the network to squash the range if it wants to:

$$y^{(k)} = \gamma^{(k)} \widehat{x}^{(k)} + \beta^{(k)}$$

Note, the network can learn:

$$\gamma^{(k)} = \sqrt{\mathrm{Var}[x^{(k)}]}$$

$$\beta^{(k)} = \mathrm{E}[x^{(k)}]$$

to recover the identity mapping.

(adapted from: http://cs231n.stanford.edu)

Batch Normalization [Ioffe & Szegedy 2015]

Input: Values of x over a mini-batch: $\mathcal{B} = \{x_{1\ldots m}\}$;
Parameters to be learned: γ, β

Output: $\{y_i = \mathrm{BN}_{\gamma,\beta}(x_i)\}$

$$\mu_\mathcal{B} \leftarrow \frac{1}{m}\sum_{i=1}^{m} x_i \qquad \text{// mini-batch mean}$$

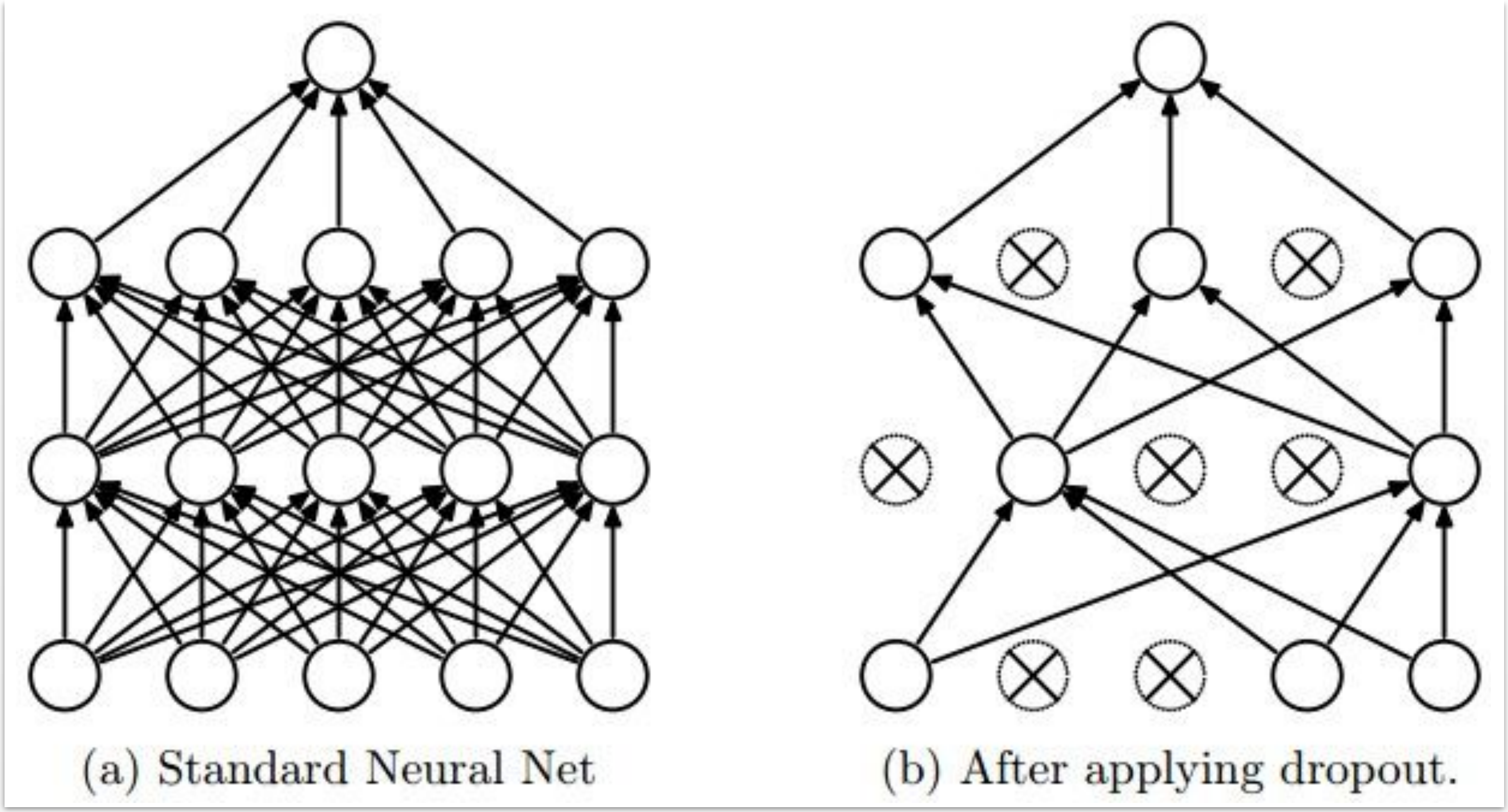
$$\sigma_\mathcal{B}^2 \leftarrow \frac{1}{m}\sum_{i=1}^{m} (x_i - \mu_\mathcal{B})^2 \qquad \text{// mini-batch variance}$$

$$\widehat{x}_i \leftarrow \frac{x_i - \mu_\mathcal{B}}{\sqrt{\sigma_\mathcal{B}^2 + \epsilon}} \qquad \text{// normalize}$$

$$y_i \leftarrow \gamma \widehat{x}_i + \beta \equiv \mathrm{BN}_{\gamma,\beta}(x_i) \qquad \text{// scale and shift}$$

- Improves gradient flow through the network
- Allows higher learning rates
- Reduces the strong dependence on initialization
- Acts as a form of regularization in a funny way, and slightly reduces the need for dropout, maybe

Dropout [Srivastava et al. 2014]

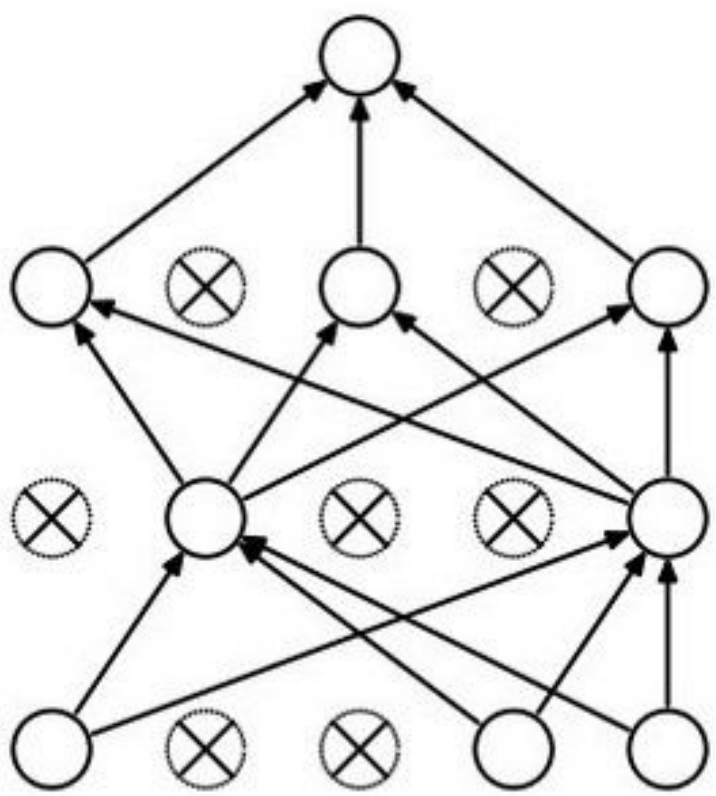


Idea: For each gradient step, “turn off” random subset of units in each layer (i.e. multiply by zero)

(adapted from: http://cs231n.stanford.edu)

Dropout [Srivastava et al. 2014]

Waaaait a second…
How could this possibly be a good idea?

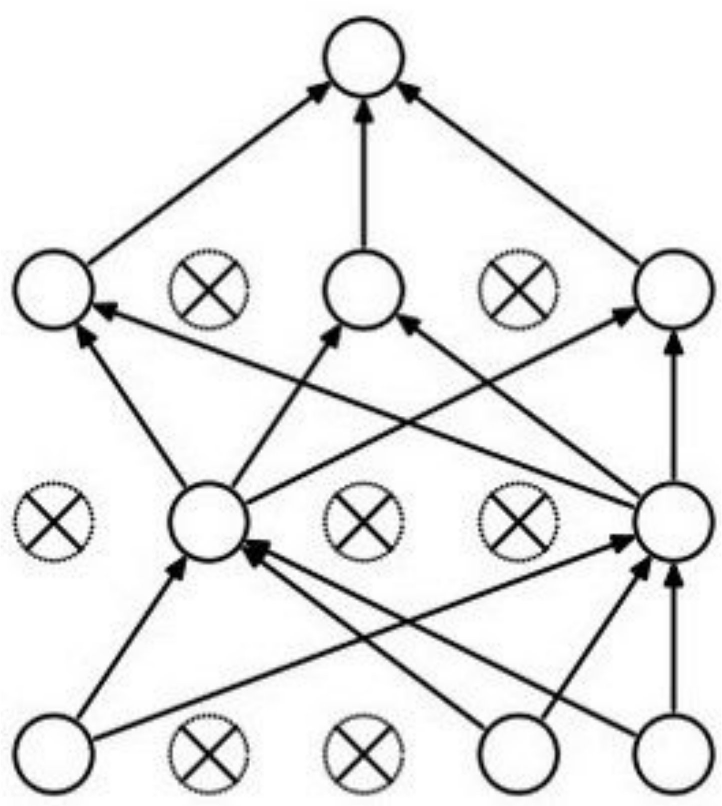


Forces the network to have a redundant representation.

(adapted from: http://cs231n.stanford.edu)

Dropout [Srivastava et al. 2014]

Waaaait a second…
How could this possibly be a good idea?



Another interpretation:

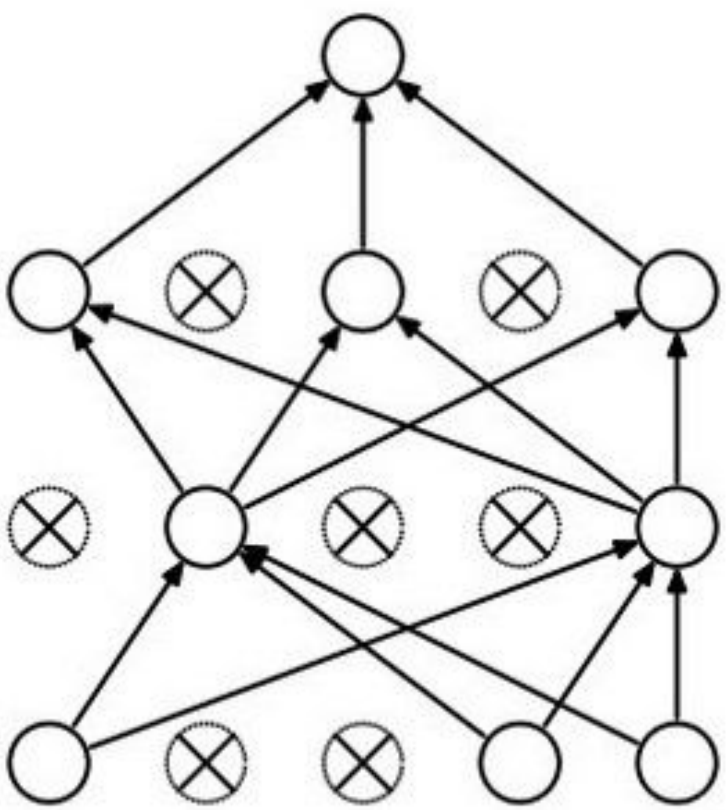
Dropout is training a large ensemble of models (that share parameters).

Each binary mask is one model, gets trained on only ~one datapoint.

(adapted from: http://cs231n.stanford.edu)

Dropout [Srivastava et al. 2014]

At test time….



Ideally:
want to integrate out all the noise

Monte Carlo approximation:
do many forward passes with different dropout masks, average all predictions

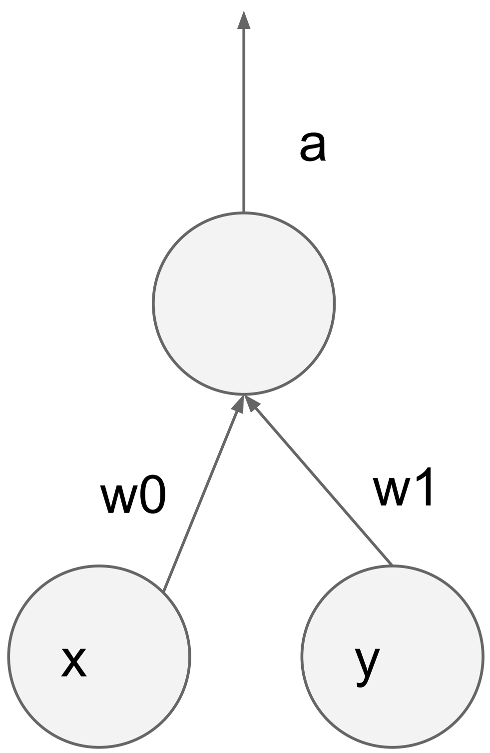
(adapted from: http://cs231n.stanford.edu)

Dropout [Srivastava et al. 2014]

At test time….

Can in fact do this with a single forward pass! (approximately)

Leave all input neurons turned on (no dropout).



during test: **a = w0*x + w1*y**

during train:

E[a] = ¼ * (w0*0 + w1*0
w0*0 + w1*y
w0*x + w1*0
w0*x + w1*y)
= ¼ * (2 w0*x + 2 w1*y)
= **½ * (w0*x + w1*y)**

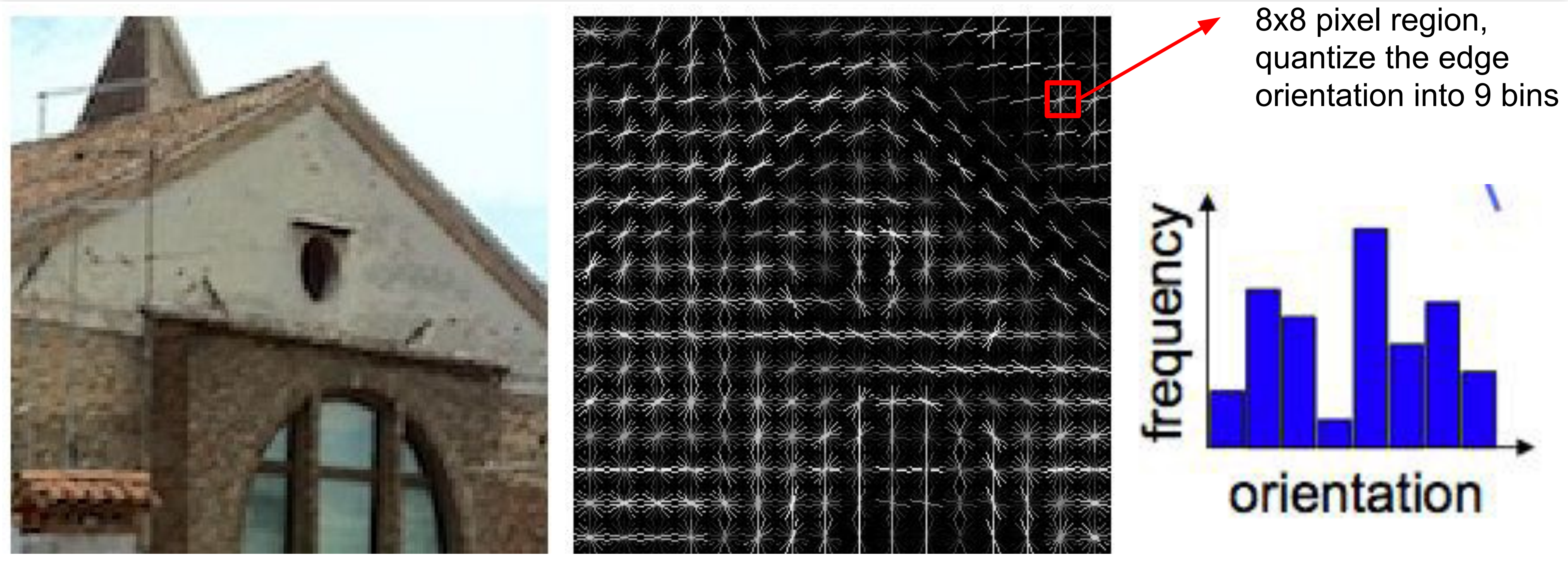
With p=0.5, using all inputs in the forward pass would inflate the activations by 2x from what the network was "used to" during training!
=> Have to compensate by scaling the activations back down by ½

Inverted Dropout: Scale up activations at train time

(adapted from: http://cs231n.stanford.edu)

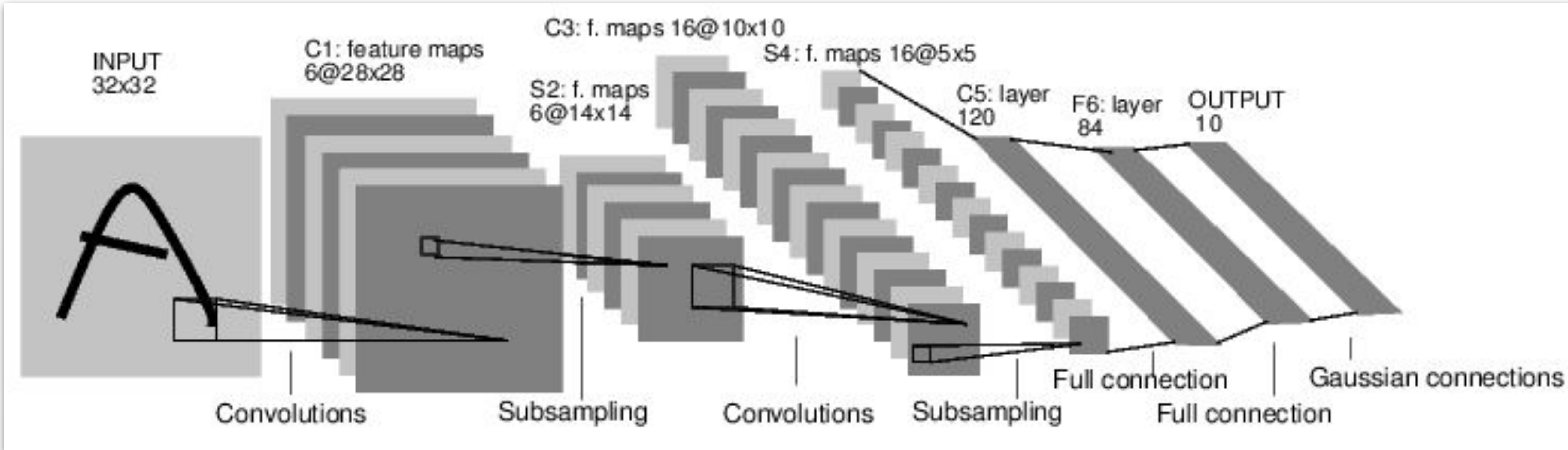
Convolutional Neural Nets

HoG/SIFT features in Computer Vision



State of the art before deep learning:
calculate histograms of gradients

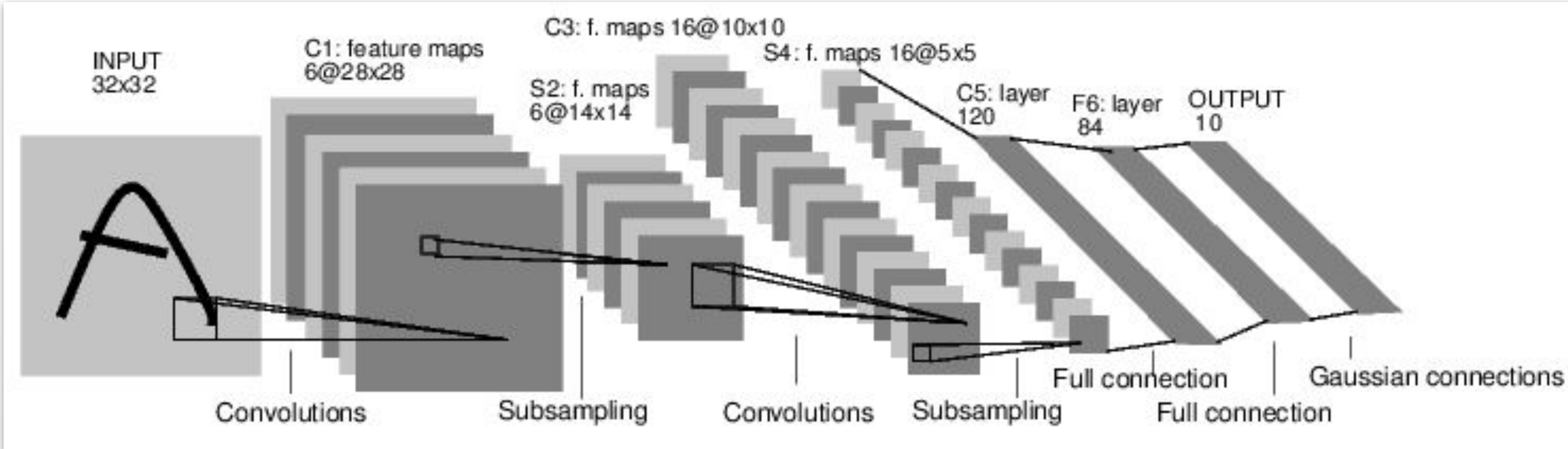
Convolutional Neural Nets



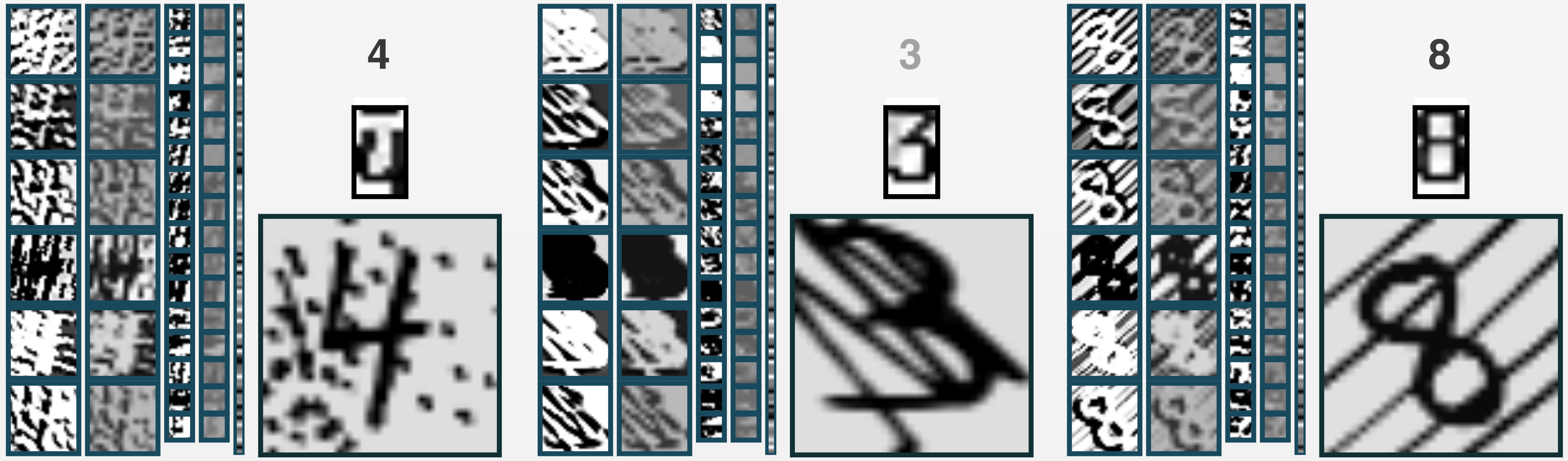
[Lecun, Bottou, Bengio & Haffner, 1998]

Gradient-based learning applied to document recognition
Y LeCun, L Bottou, Y Bengio… - Proceedings of the …, 1998 - ieeexplore.ieee.org
Multilayer neural networks trained with the back-propagation algorithm constitute the best example of a successful gradientbased learning technique. Given an appropriate network architecture, gradient-based learning algorithms can be used to synthesize a complex decision surface that can classify high-dimensional patterns, such as handwritten characters, with minimal preprocessing. This paper reviews various methods applied to ...
Cited by 6350 Related articles All 53 versions Cite Save

Convolutional Neural Nets

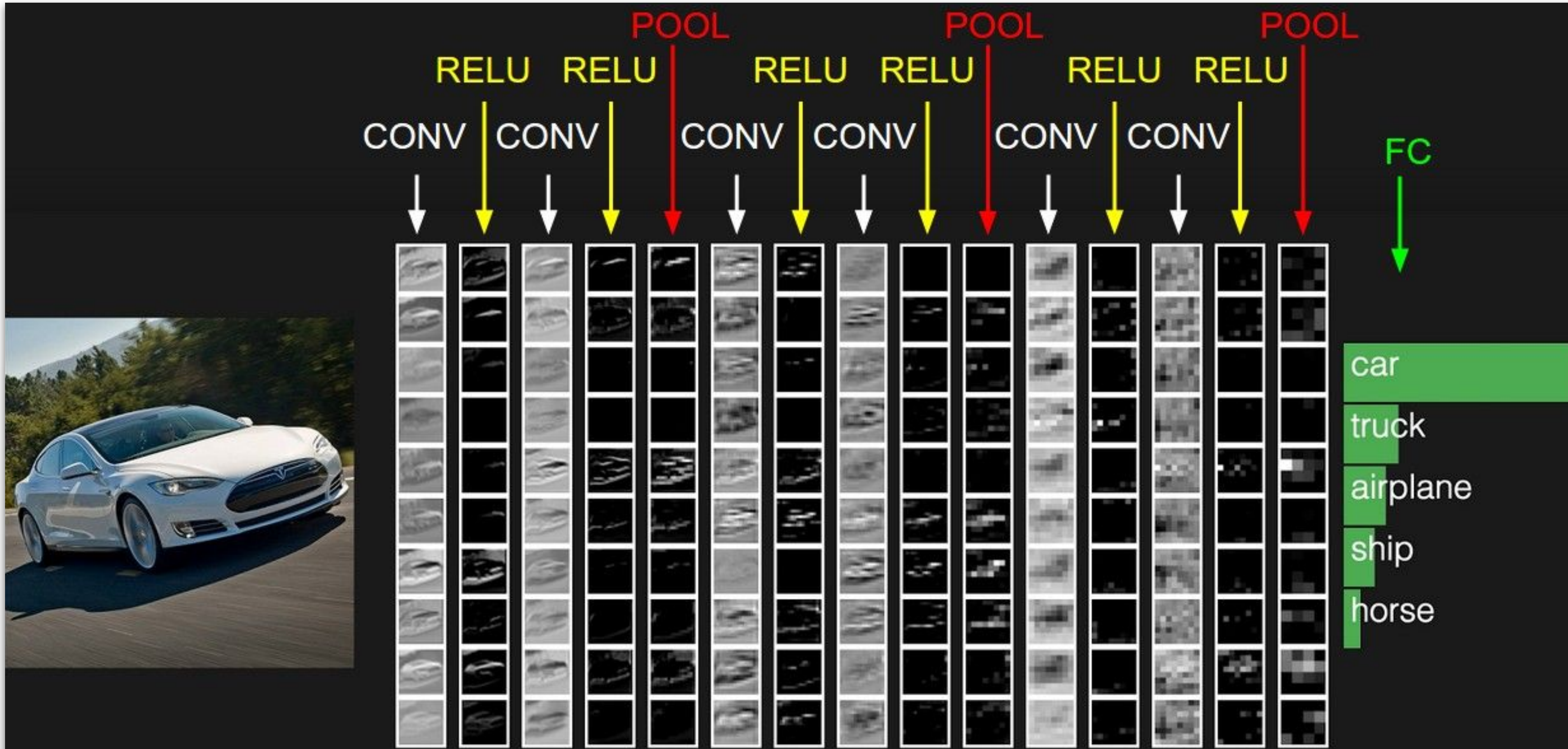


[Lecun, Bottou, Bengio & Haffner, 1998]



Used for handwriting recognition

Convolutional Neural Nets



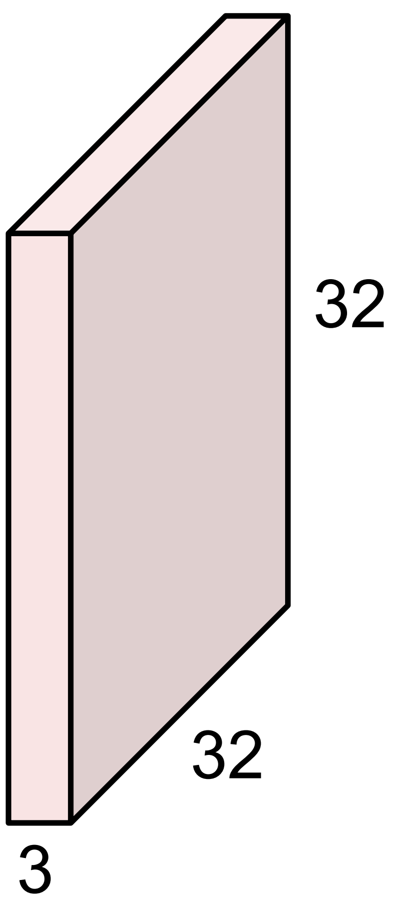
(adapted from: http://cs231n.stanford.edu)

Convolutional Neural Nets

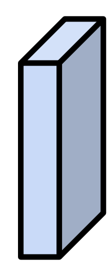
Convolution Layer

Filters always extend the full depth of the input volume

32x32x3 image



5x5x3 filter

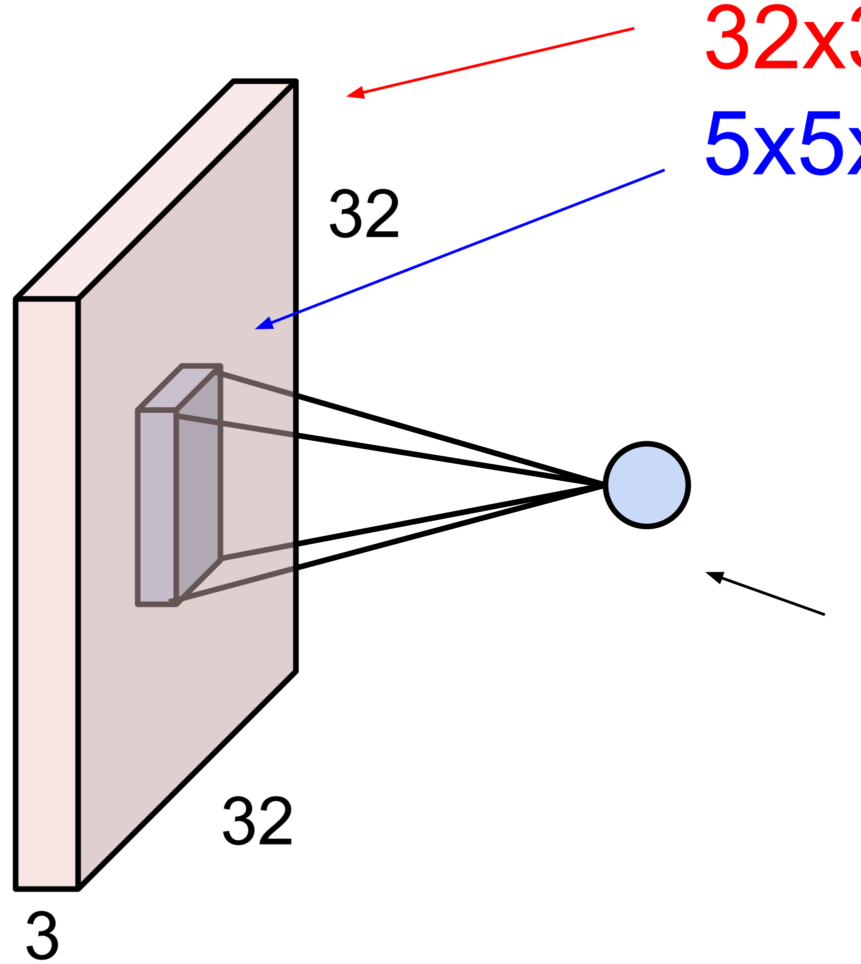


Convolve the filter with the image
i.e. “slide over the image spatially, computing dot products”

(adapted from: http://cs231n.stanford.edu)

Convolutional Neural Nets

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)

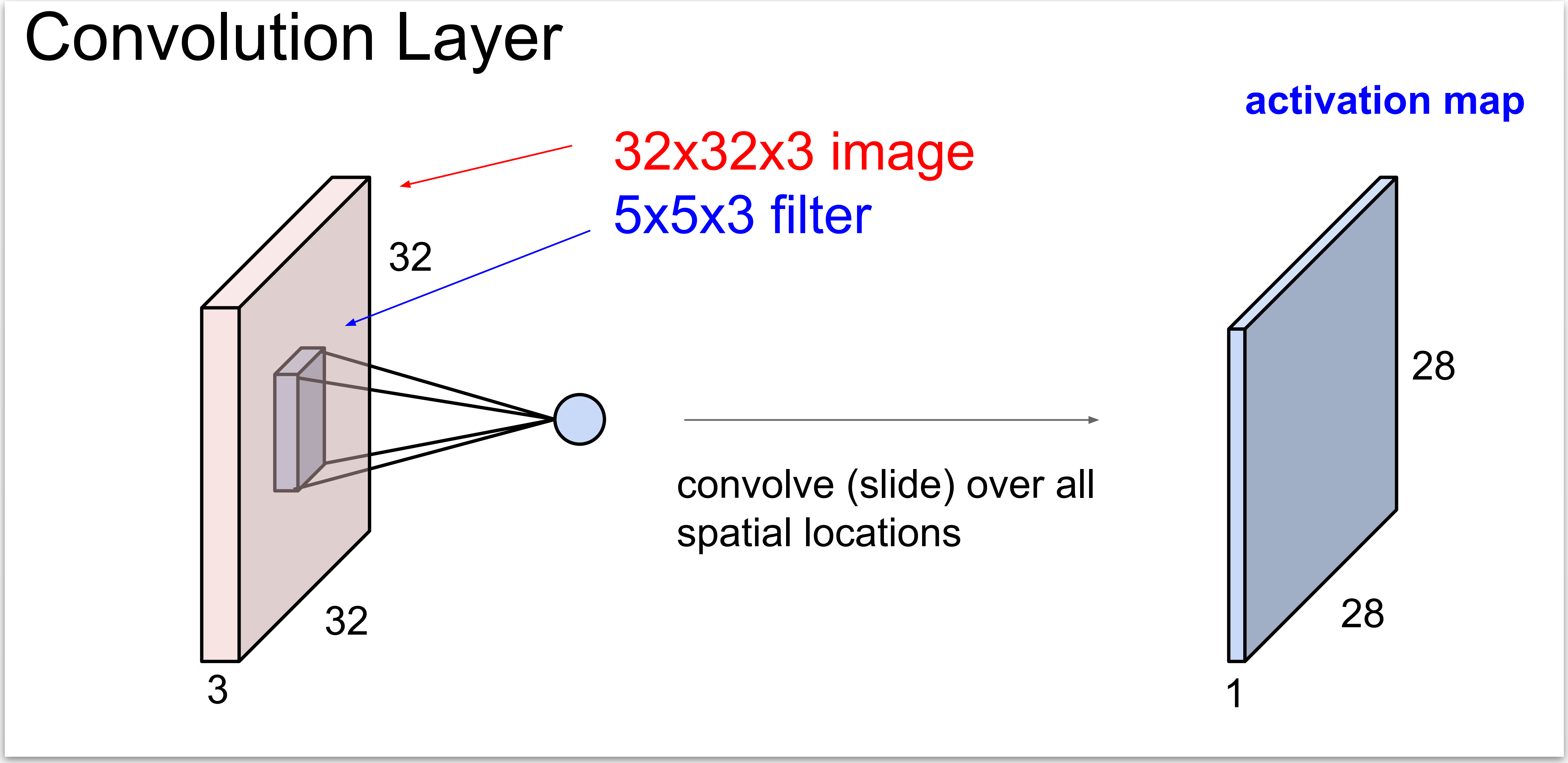
$$w^T x + b$$

Convolution:

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

(adapted from: http://cs231n.stanford.edu)

Convolutional Neural Nets

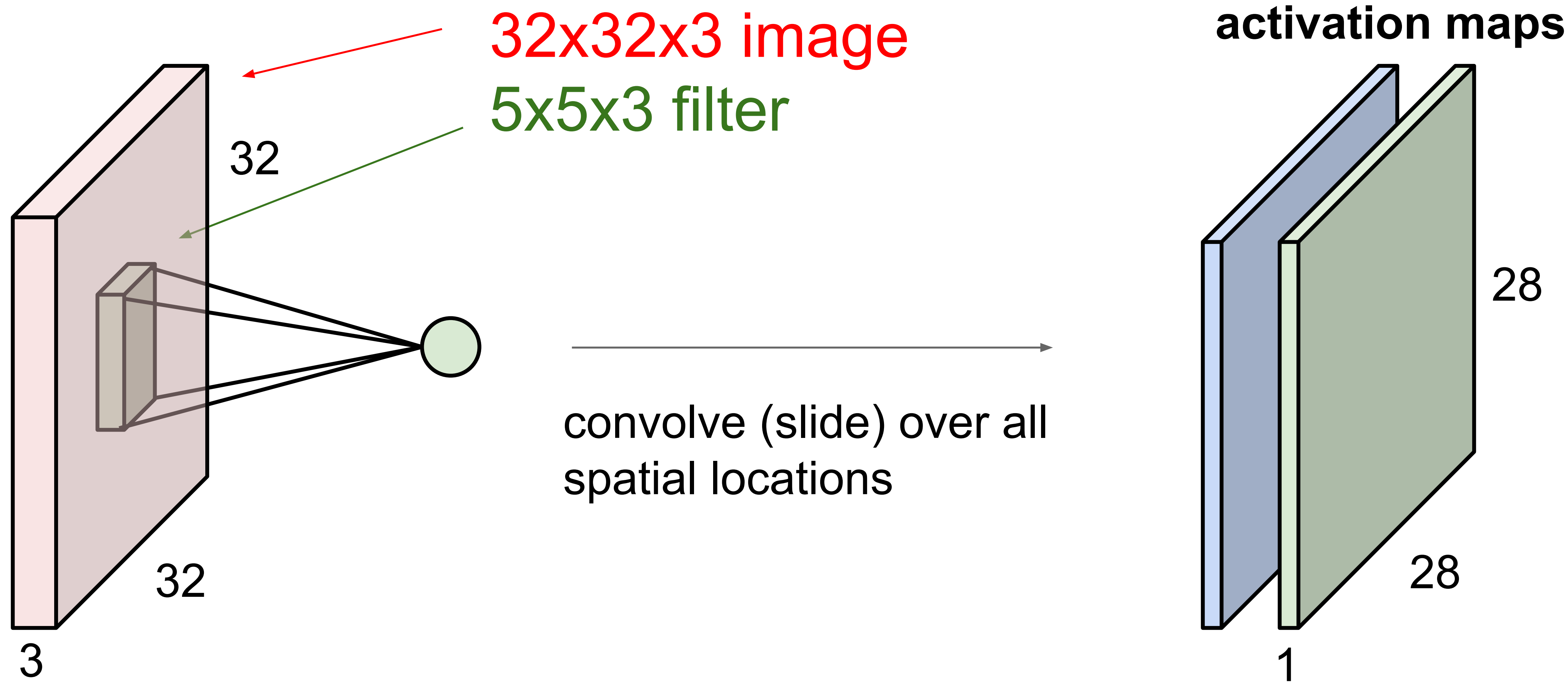


(adapted from: http://cs231n.stanford.edu)

Convolutional Neural Nets

Convolution Layer

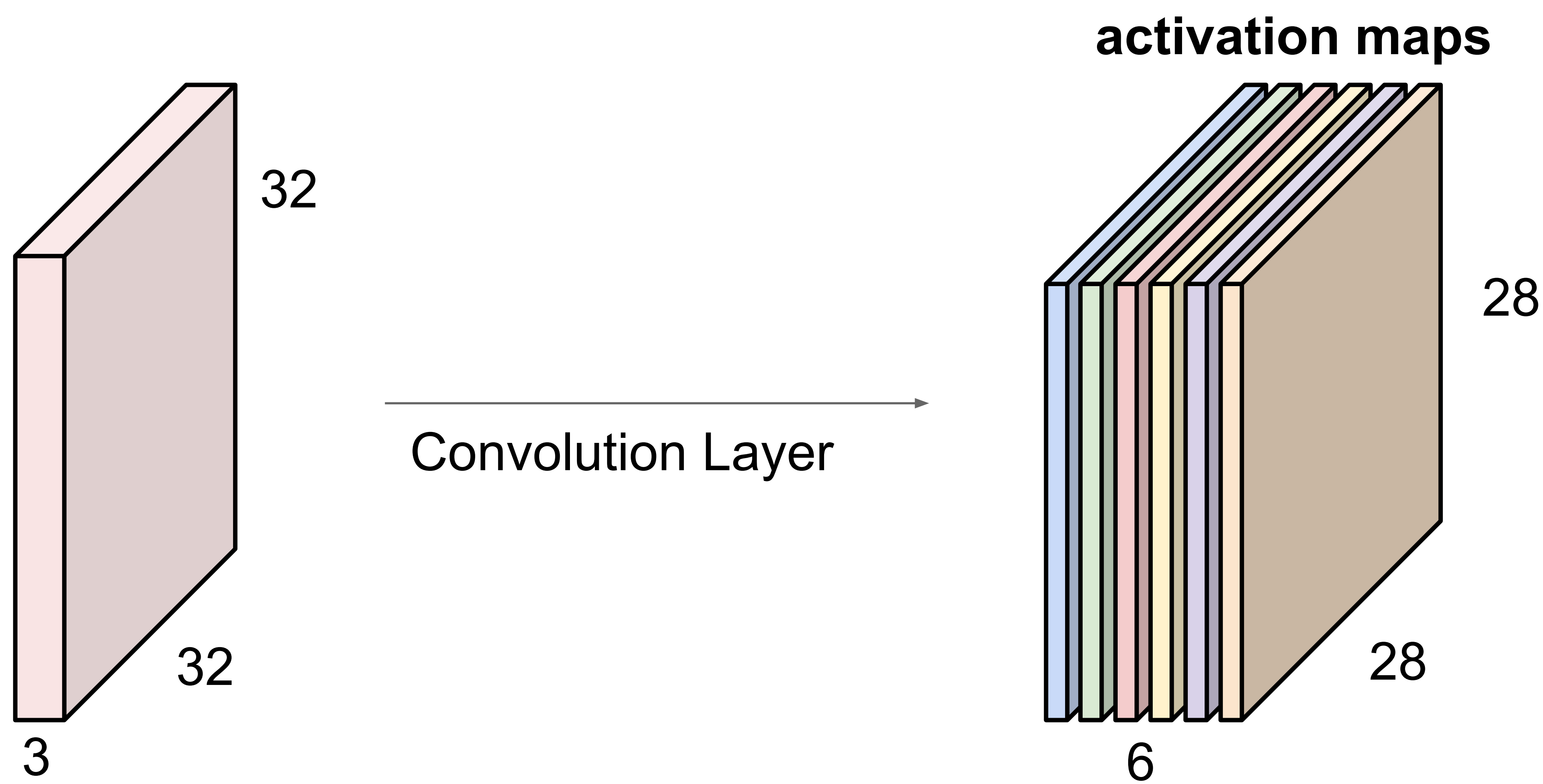
consider a second, green filter



(adapted from: http://cs231n.stanford.edu)

Convolutional Neural Nets

6 filters of size 5x5x3 yields a new 28x28x6 "image"



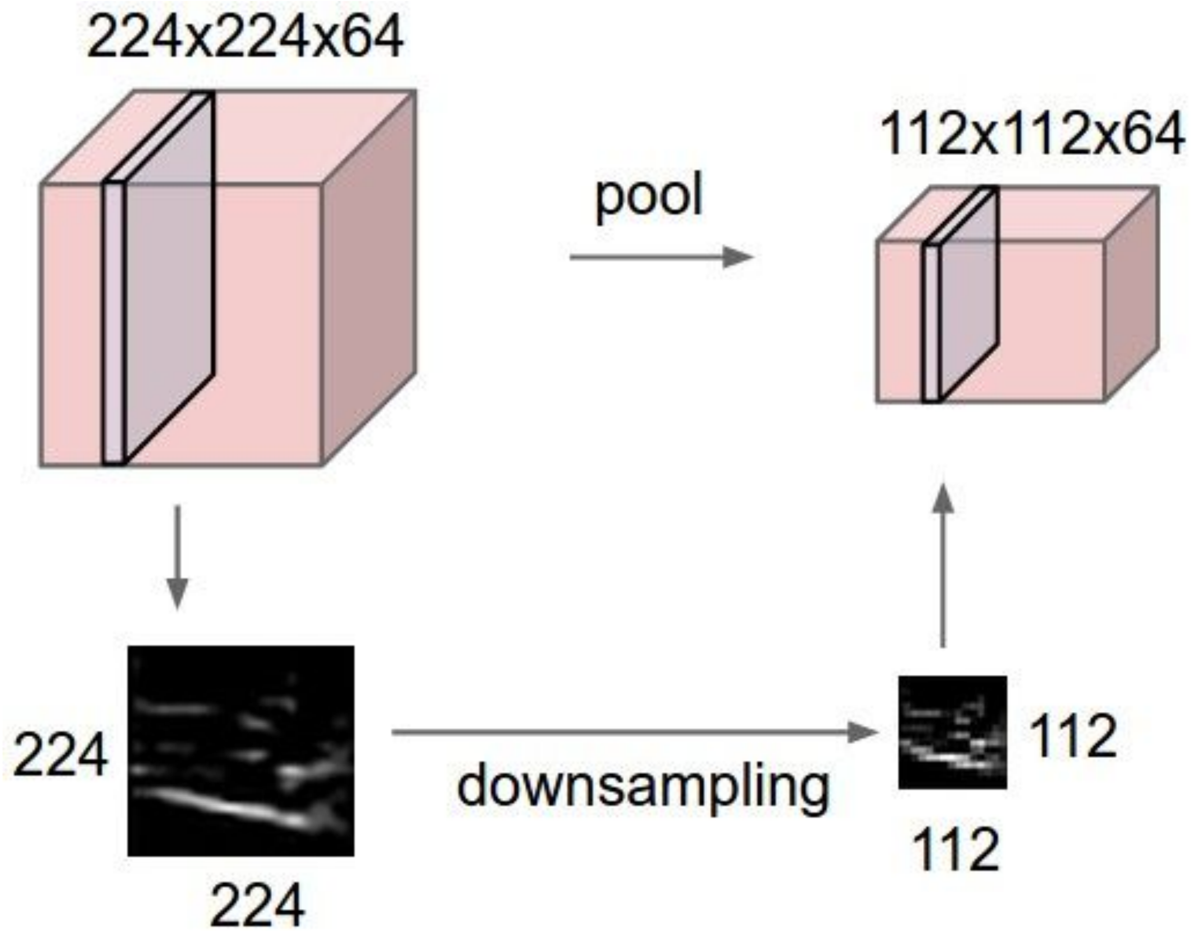
We stack these up to get a "new image" of size 28x28x6!

(adapted from: http://cs231n.stanford.edu)

Convolutional Neural Nets

Pooling layer

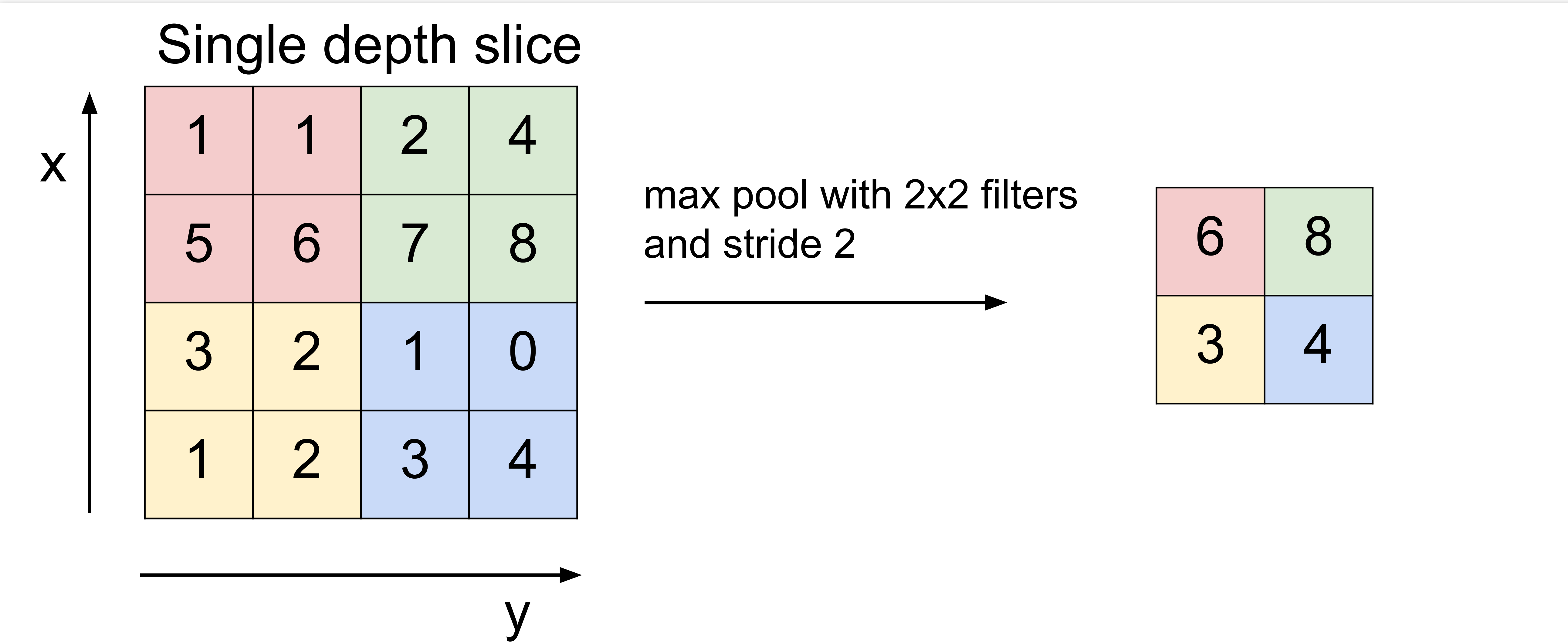
- makes the representations smaller and more manageable
- operates over each activation map independently:



(adapted from: http://cs231n.stanford.edu)

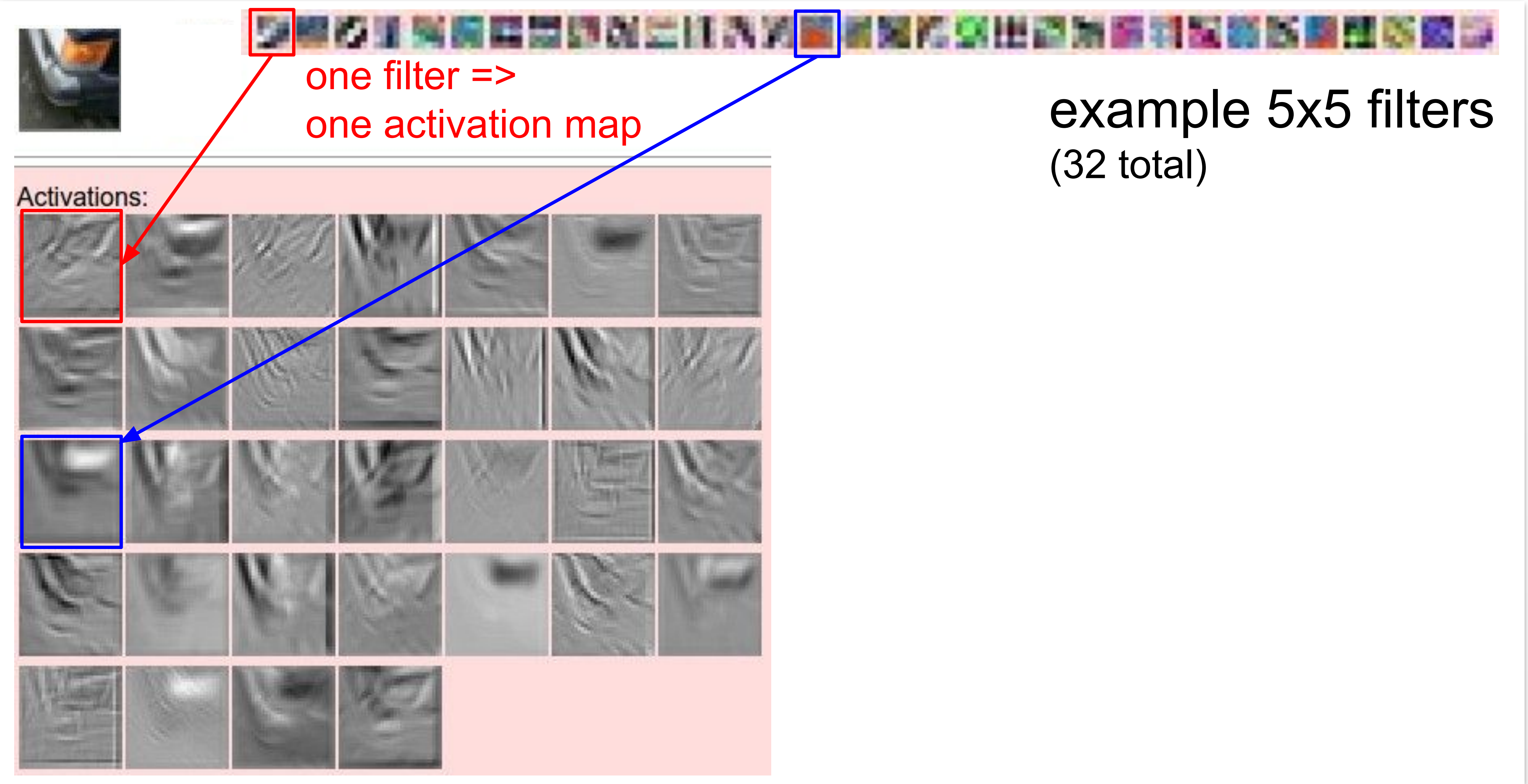
Convolutional Neural Nets

Max-pooling: Subsample by taking maximum in window



(adapted from: http://cs231n.stanford.edu)

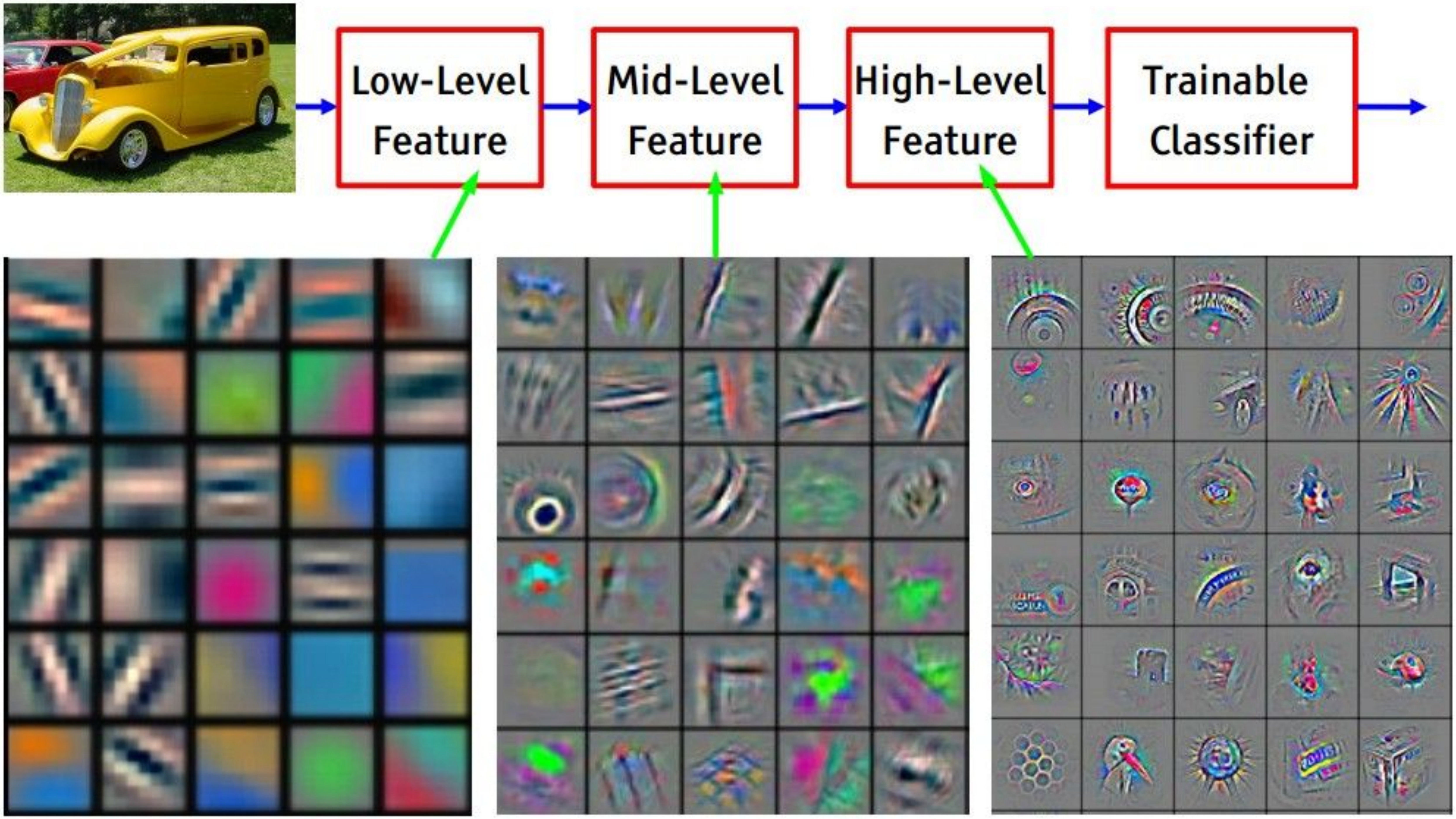
Feature Maps



example 5x5 filters
(32 total)

(adapted from a slide by Yann LeCun)

Feature Abstractions



Feature visualization of convolutional net trained on ImageNet from [Zeiler & Fergus 2013]

(adapted from a slide by Yann LeCun)

Neural Nets Keep Getting Bigger

IMAGENET Large Scale Visual Recognition Challenge

Year 2010

NEC-UIUC

Dense grid descriptor: HOG, LBP

Coding: local coordinate, super-vector

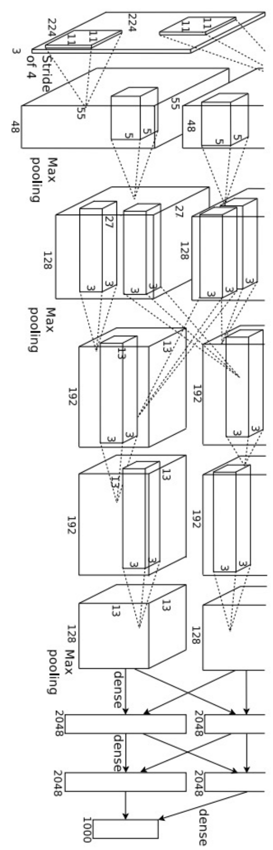
Pooling, SPM

Linear SVM

[Lin CVPR 2011]

Year 2012

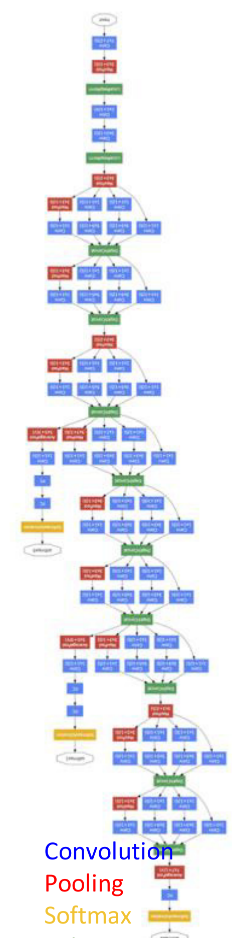
SuperVision



[Krizhevsky NIPS 2012]

Year 2014

GoogLeNet



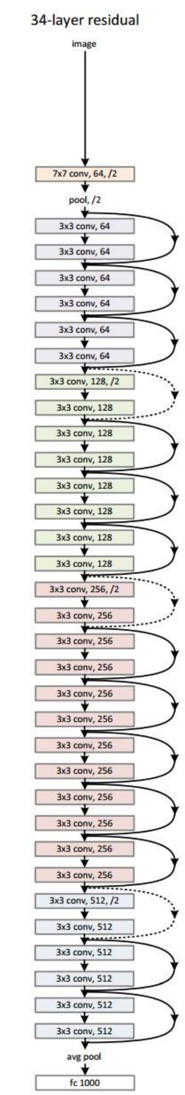
[Szegedy arxiv 2014]

VGG

[Simonyan arxiv 2014]

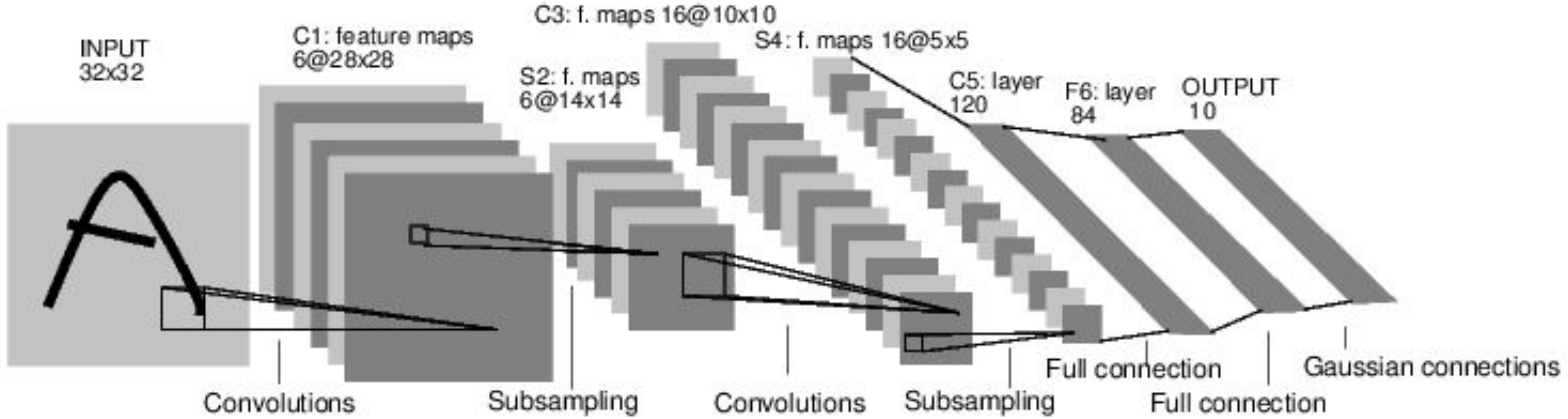
Year 2015

MSRA



(adapted from: http://cs231n.stanford.edu)

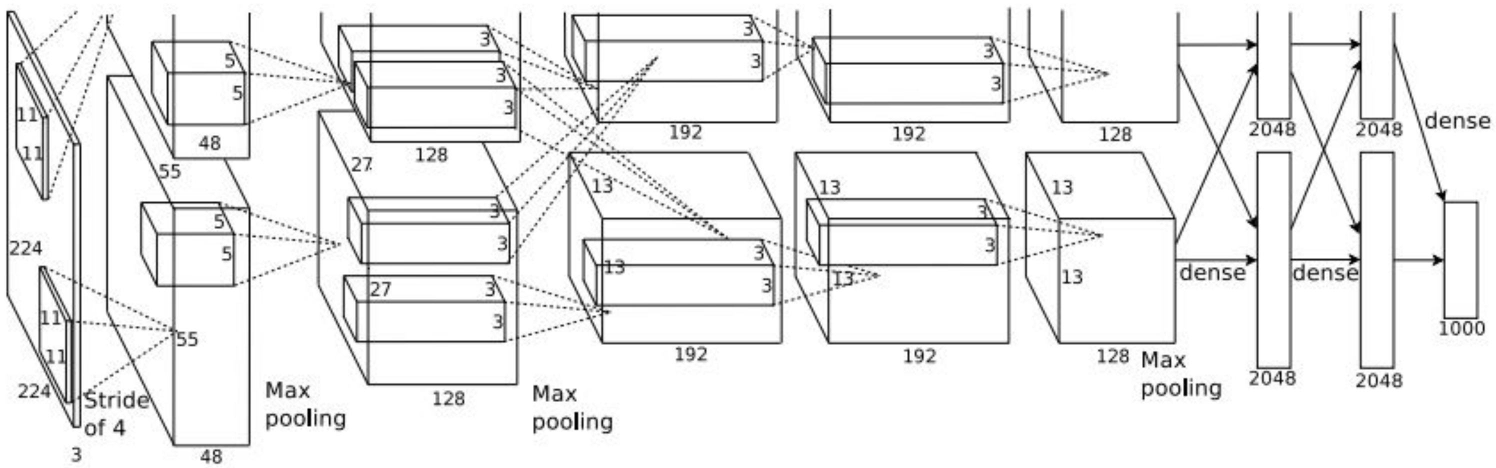
Case Study: LeNet-5 [Lecun et al 1998]



Conv filters were 5x5, applied at stride 1
Subsampling (Pooling) layers were 2x2 applied at stride 2
i.e. architecture is [CONV-POOL-CONV-POOL-CONV-FC]

(adapted from: http://cs231n.stanford.edu)

Case Study: AlexNet [Krizhevsky et al 2012]



Full (simplified) AlexNet architecture:
[227x227x3] INPUT
[55x55x96] CONV1: 96 11x11 filters at stride 4, pad 0
[27x27x96] MAX POOL1: 3x3 filters at stride 2
[27x27x96] NORM1: Normalization layer
[27x27x256] CONV2: 256 5x5 filters at stride 1, pad 2
[13x13x256] MAX POOL2: 3x3 filters at stride 2
[13x13x256] NORM2: Normalization layer
[13x13x384] CONV3: 384 3x3 filters at stride 1, pad 1
[13x13x384] CONV4: 384 3x3 filters at stride 1, pad 1
[13x13x256] CONV5: 256 3x3 filters at stride 1, pad 1
[6x6x256] MAX POOL3: 3x3 filters at stride 2
[4096] FC6: 4096 neurons
[4096] FC7: 4096 neurons
[1000] FC8: 1000 neurons (class scores)

Details/Retrospectives:
- first use of ReLU
- used Norm layers (not common anymore)
- heavy data augmentation
- dropout 0.5
- batch size 128
- SGD Momentum 0.9
- Learning rate 1e-2, reduced by 10 manually when val accuracy plateaus
- L2 weight decay 5e-4
- 7 CNN ensemble: 18.2% -> 15.4%

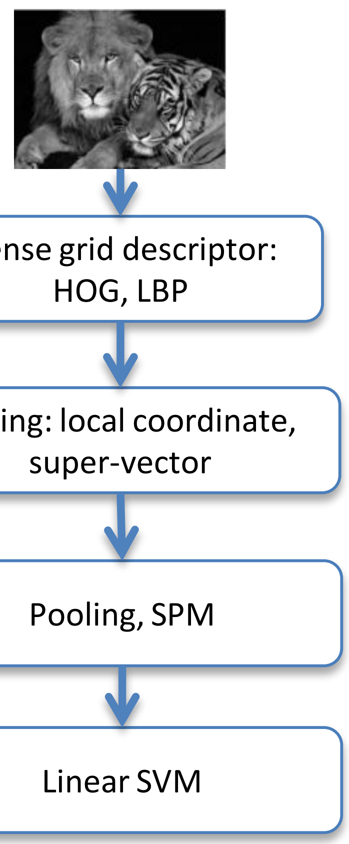
60M parameters

(adapted from: http://cs231n.stanford.edu)

ImageNet Winners Keep Getting Deeper

Year 2010

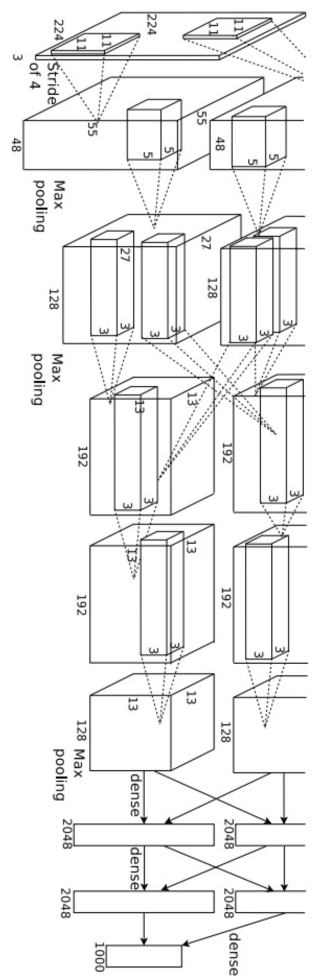
NEC-UIUC



[Lin CVPR 2011]

Year 2012

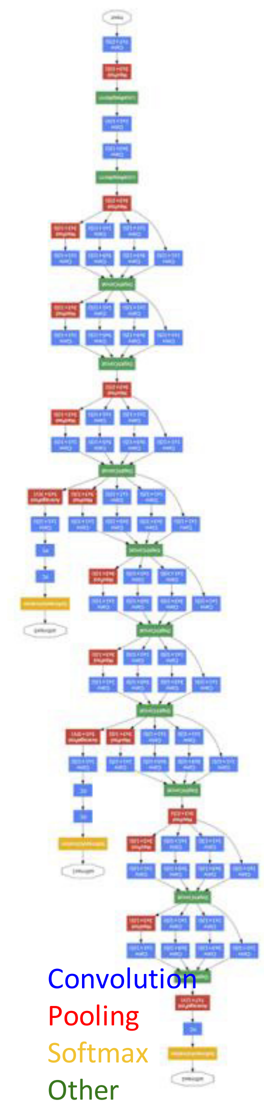
SuperVision



[Krizhevsky NIPS 2012]

Year 2014

GoogLeNet



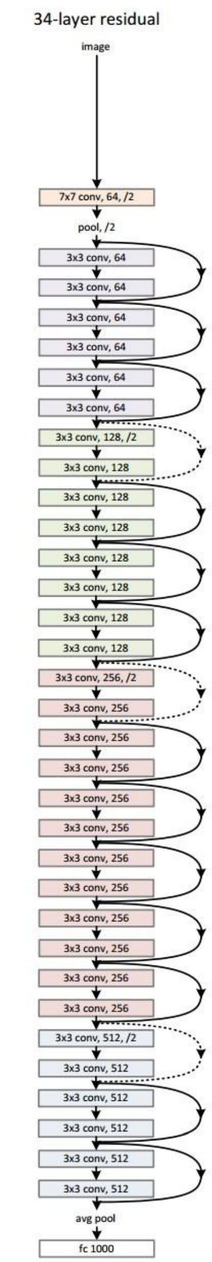
[Szegedy arxiv 2014]

VGG

[Simonyan arxiv 2014]

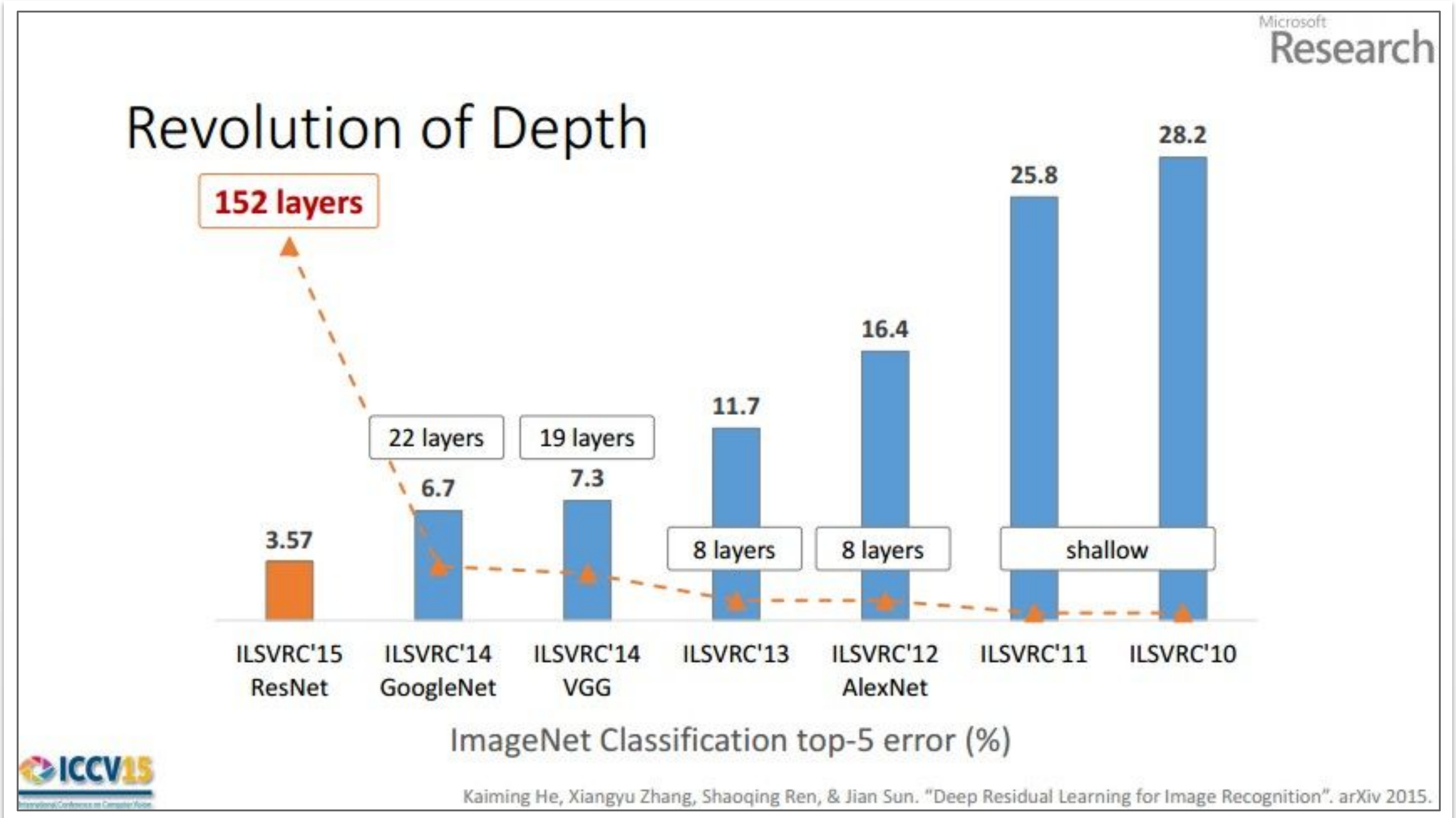
Year 2015

MSRA



(adapted from: http://cs231n.stanford.edu)

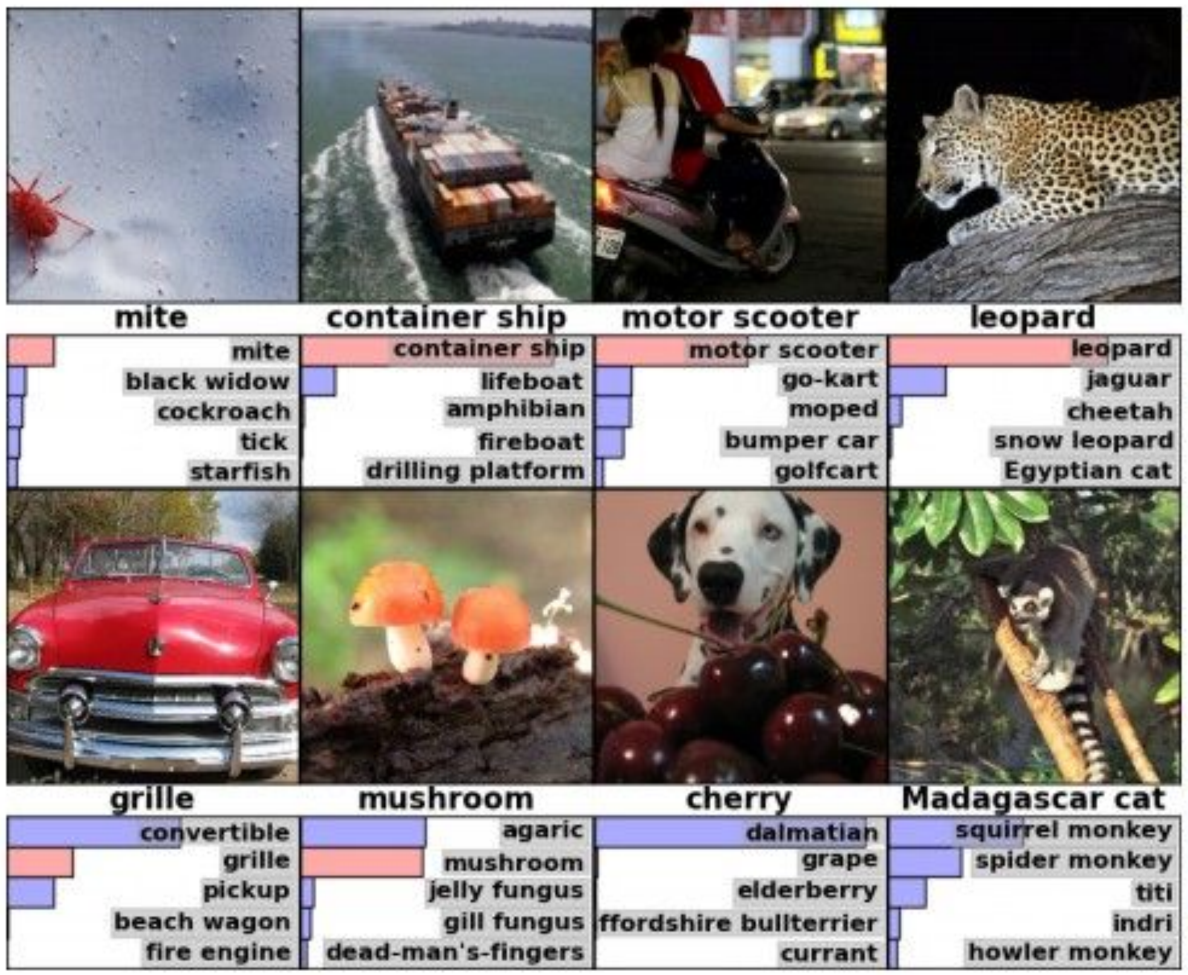
ImageNet Winners Keep Getting Deeper



(adapted from: http://cs231n.stanford.edu)

Convolutional Nets are Everywhere

Classification



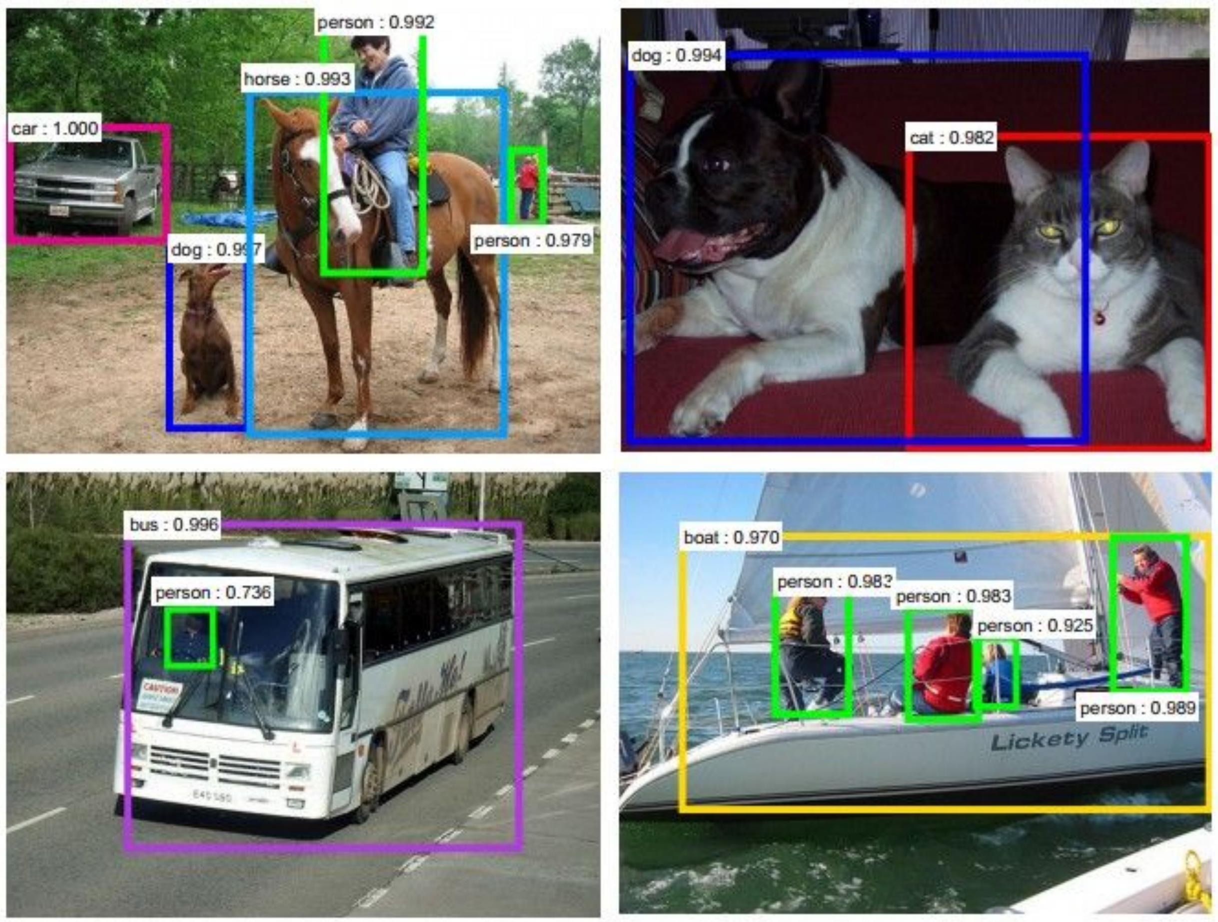
Retrieval

[Krizhevsky 2012]

(adapted from: http://cs231n.stanford.edu)

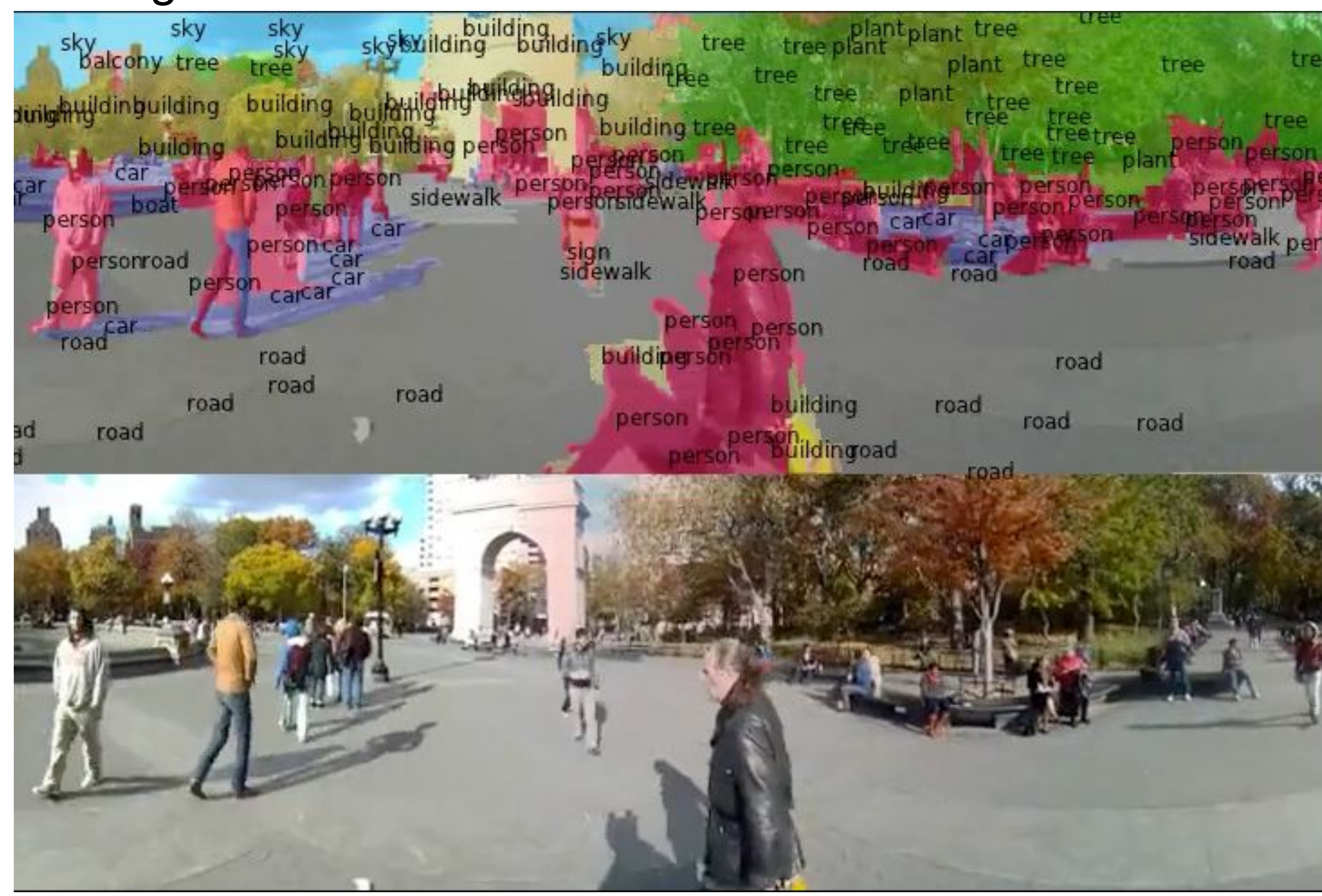
Convolutional Nets are Everywhere

Detection



[Faster R-CNN: Ren, He, Girshick, Sun 2015]

Segmentation

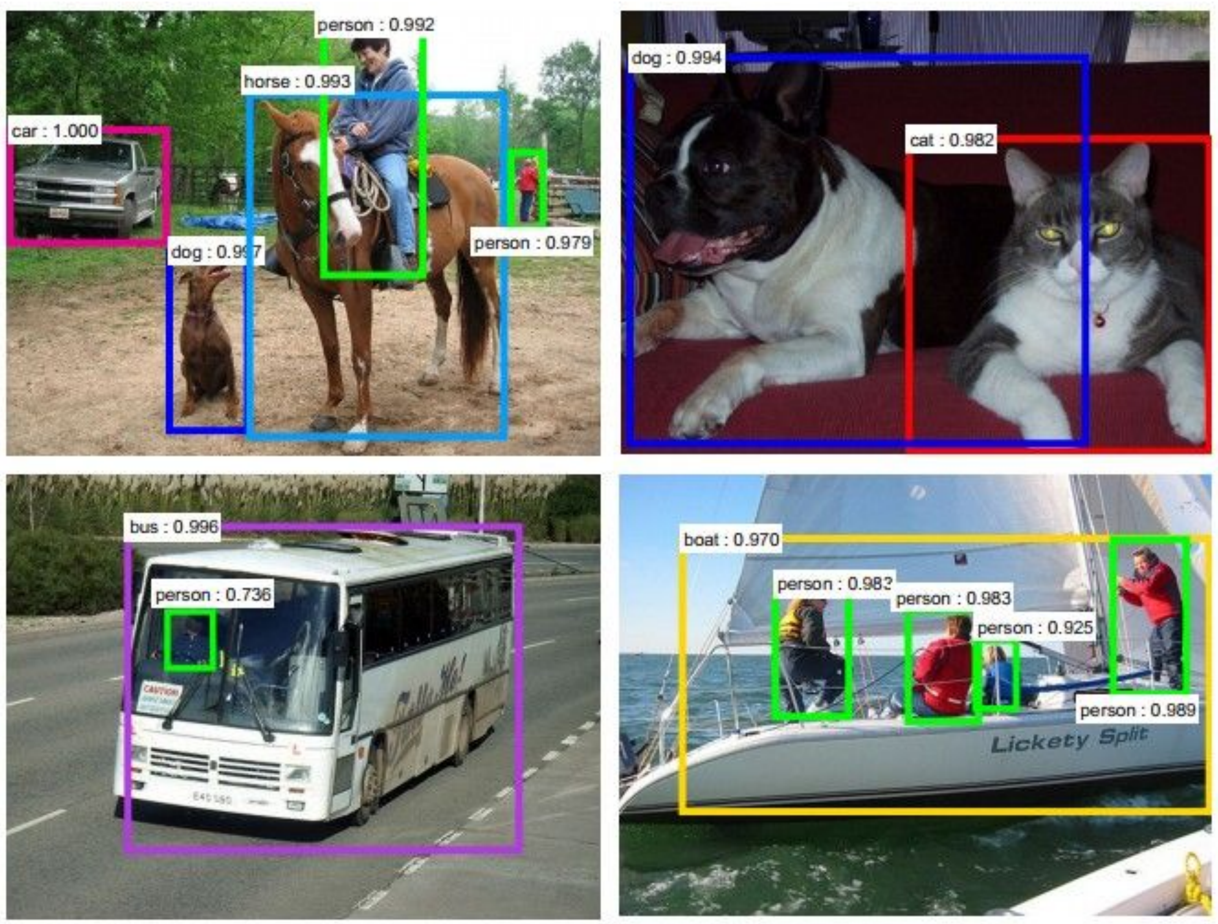


[Farabet et al., 2012]

(adapted from: http://cs231n.stanford.edu)

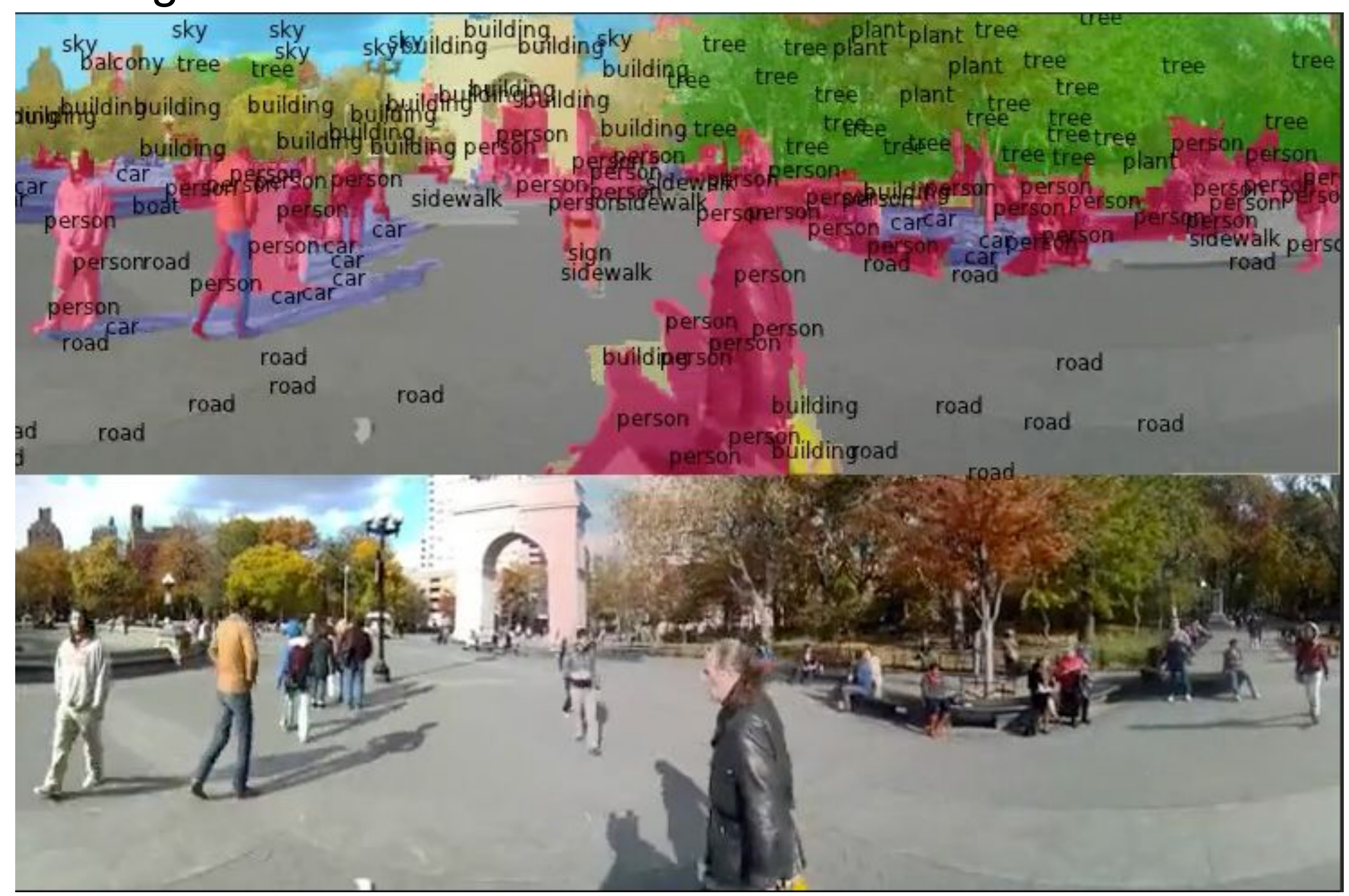
Convolutional Nets are Everywhere

Detection



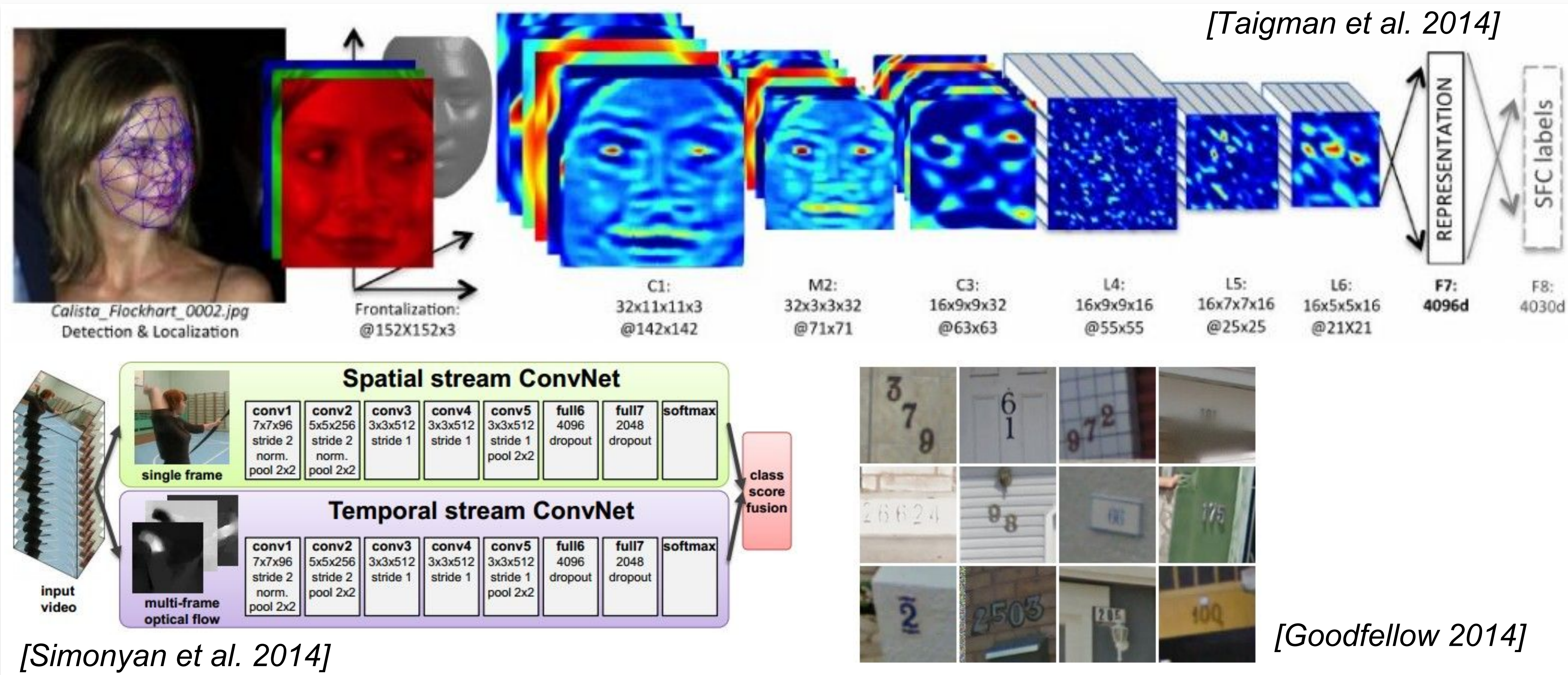
[Faster R-CNN: Ren, He, Girshick, Sun 2015]

Segmentation



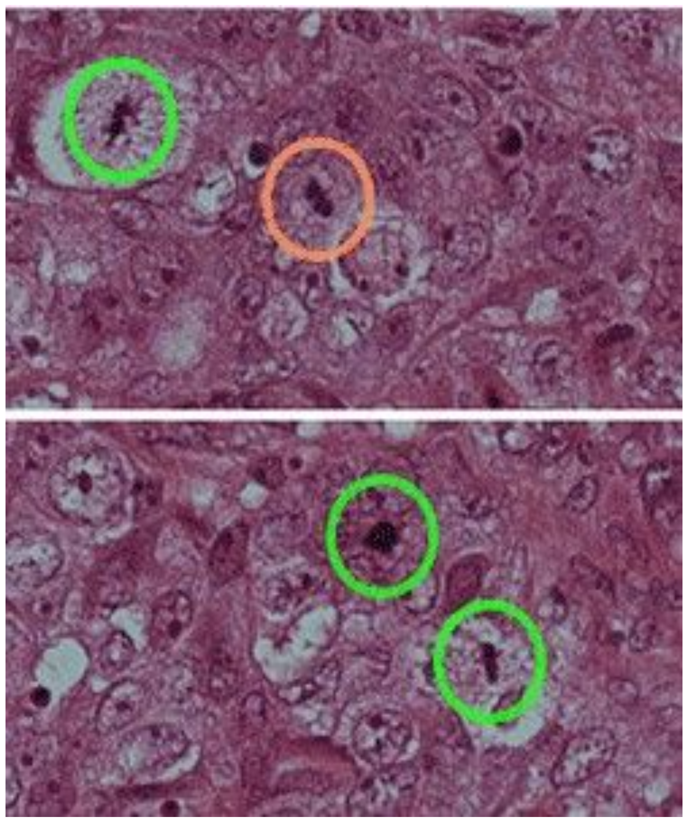
[Farabet et al., 2012]

Convolutional Nets are Everywhere



(adapted from: http://cs231n.stanford.edu)

Convolutional Nets are Everywhere

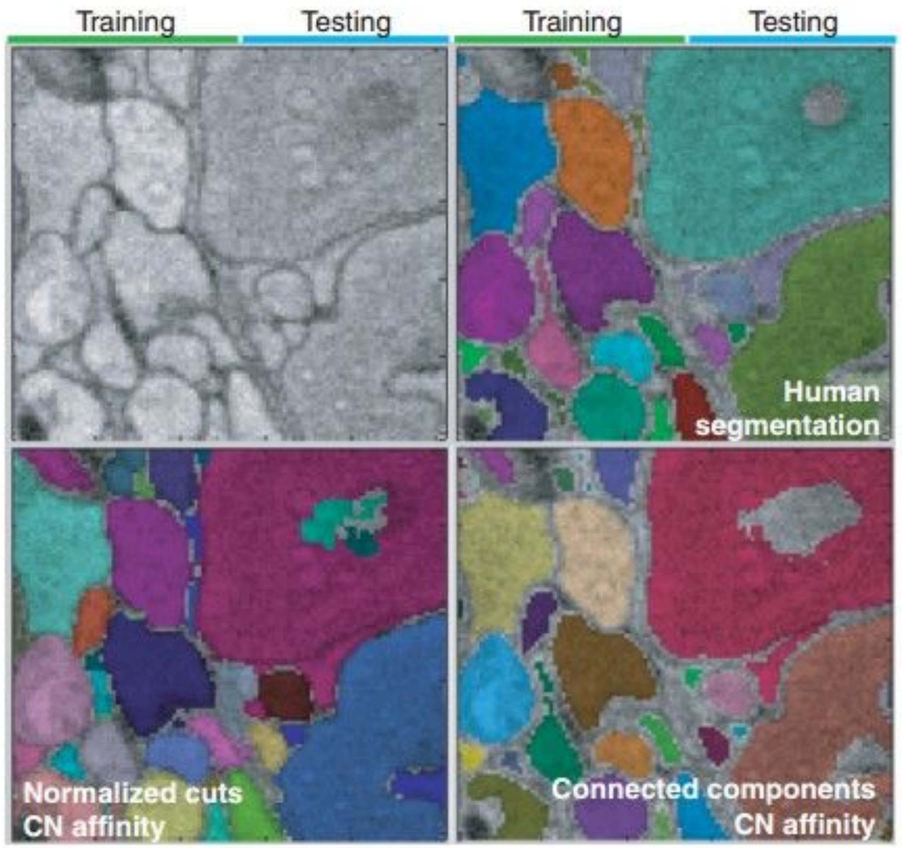


[Ciresan et al. 2013]

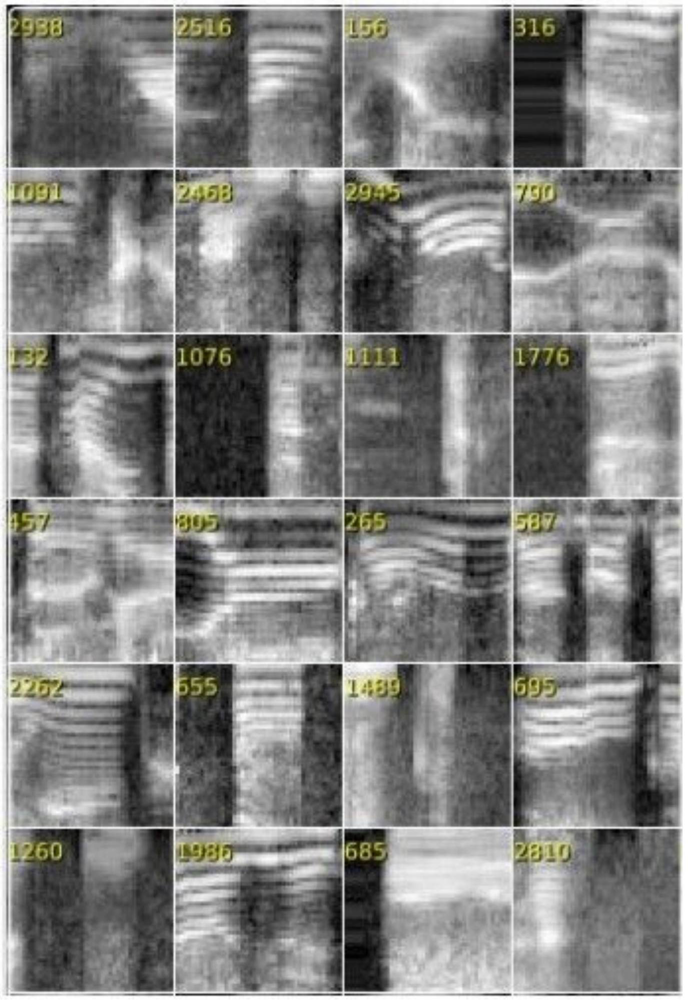
[Sermanet et al. 2011]
[Ciresan et al.]

(adapted from: http://cs231n.stanford.edu)

Convolutional Nets are Everywhere



[Turaga et al., 2010]



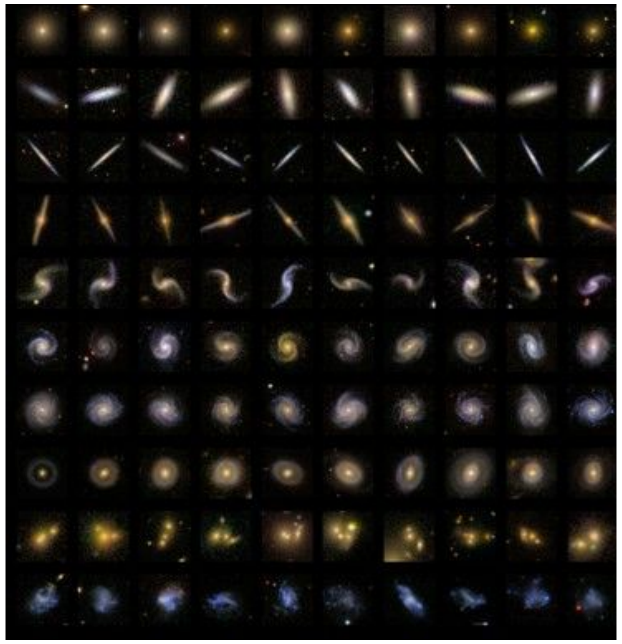
I caught this movie on the Sci-Fi channel recently. It actually turned out to be pretty decent as far as B-list horror/suspense films go. **Two guys (one naive and one loud mouthed a **) take a road trip to stop a wedding but have the worst possible luck when a maniac in a freaky, make-shift tank/truck hybrid decides to play cat-and-mouse with them.** Things are further complicated when they pick up a ridiculously whorish hitchhiker. What makes this film unique is that the combination of comedy and terror actually work in this movie, unlike so many others. The two guys are likable enough and there are some good chase/suspense scenes. Nice pacing and comic timing make this movie more than passable for the horror/slasher buff. **Definitely worth checking out.**

I just saw this on a local independent station in the New York City area. **The cast showed promise but when I saw the director, George Cosmotos, I became suspicious. And sure enough, it was every bit as bad, every bit as pointless and stupid as every George Cosmotos movie I ever saw.** He's like a stupid man's Michael Bey – with all the awfulness that accolade promises. There's no point to the conspiracy, no burning issues that urge the conspirators on. We are left to ourselves to connect the dots from one bit of graffiti on various walls in the film to the next. Thus, the current budget crisis, the war in Iraq, Islamic extremism, the fate of social security, 47 million Americans without health care, stagnating wages, and the death of the middle class are all subsumed by the sheer terror of graffiti. A truly, stunningly idiotic film.

Graphics is far from the best part of the game. **This is the number one best TH game in the series.** Next to Underground. **It deserves strong love. It is an insane game.** There are massive levels, massive unlockable characters... it's just a massive game. **Waste your money on this game. This is the kind of money that is wasted properly.** And even though graphics suck, thats doesn't make a game good. Actually, the graphics were good at the time. Today the graphics are crap. WHO CARES? As they say in Canada, This is the fun game, aye. (You get to go to Canada in THPS3) Well, I don't know if they say that, but they might. who knows. Well, Canadian people do. Wait a minute, I'm getting off topic. This game rocks. Buy it, play it, enjoy it, love it. It's PURE BRILLIANCE.

The first was good and original. I was a not bad horror/comedy movie. So I heard a second one was made and I had to watch it . What really makes this movie work is Judd Nelson's character and the sometimes clever script. **A pretty good script for a person who wrote the Final Destination films and the direction was okay.** Sometimes there's scenes where it looks like it was filmed using a home video camera with a grainy - look. Great made - for - TV movie. **It was worth the rental and probably worth buying just to get that nice eerie feeling and watch Judd Nelson's Stanley doing what he does best.** I suggest newcomers to watch the first one before watching the sequel, just so you'll have an idea what Stanley is like and get a little history background.

[Denil et al. 2014]



(adapted from: http://cs231n.stanford.edu)

CNNs for Reinforcement Learning

[Mnih et al., Nature 2015]

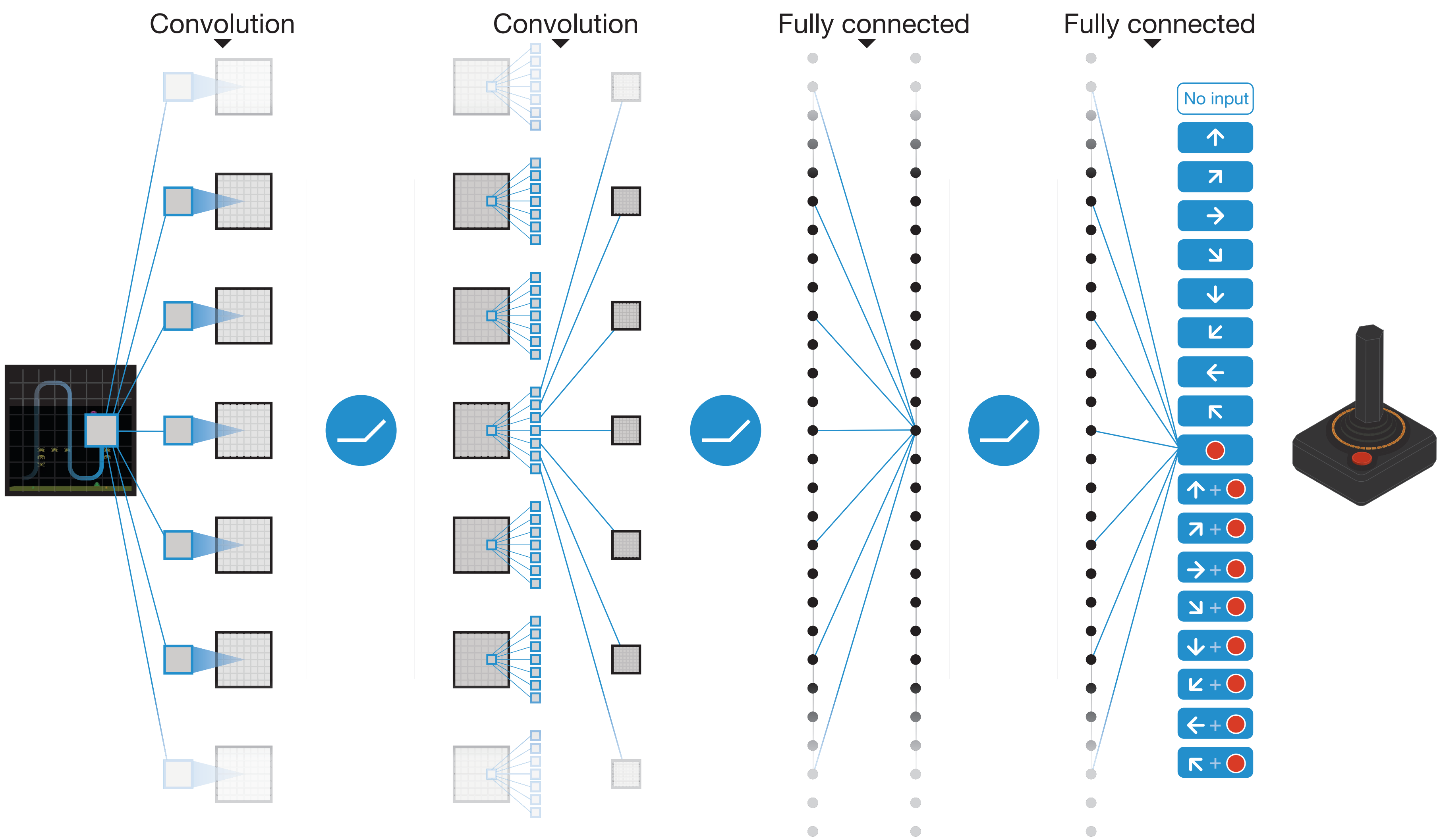
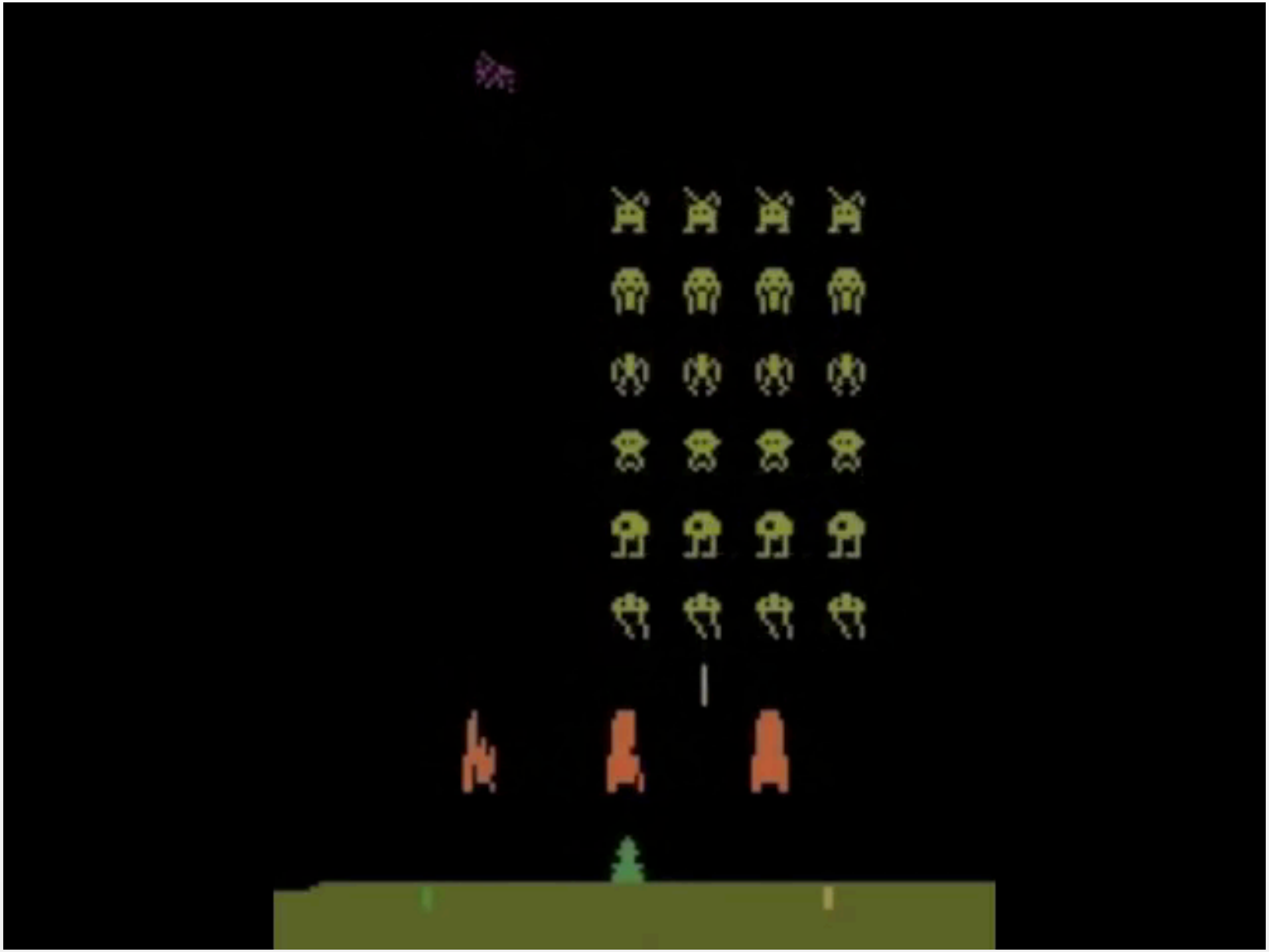


Figure 1 | Schematic illustration of the convolutional neural network. The details of the architecture are explained in the Methods. The input to the neural network consists of an $84 \times 84 \times 4$ image produced by the preprocessing map ϕ, followed by three convolutional layers (note: snaking blue line symbolizes sliding of each filter across input image) and two fully connected layers with a single output for each valid action. Each hidden layer is followed by a rectifier nonlinearity (that is, $\max(0,x)$).

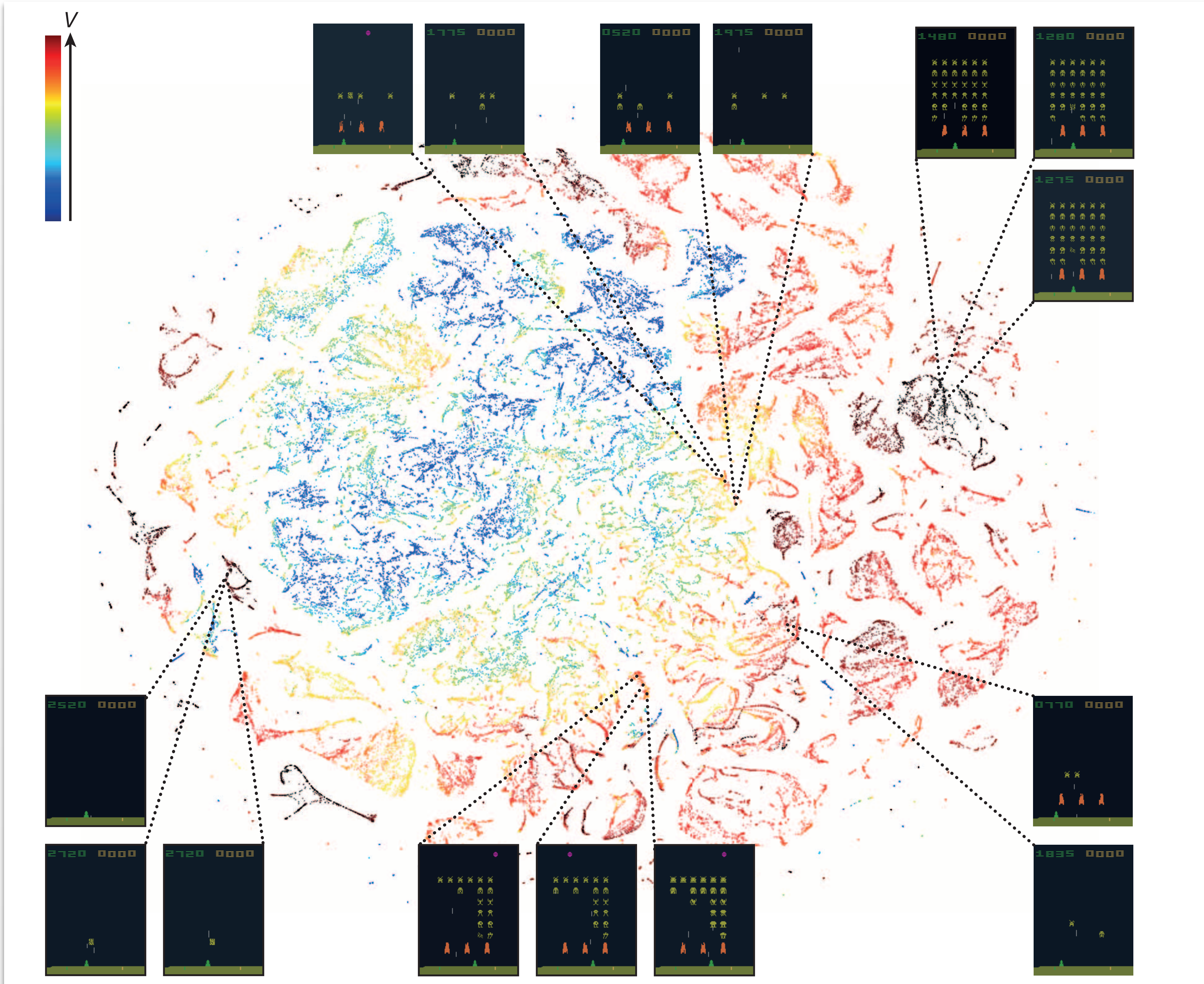
Inputs: Time series of Atari images, *Loss:* Game score

CNNs for Reinforcement Learning

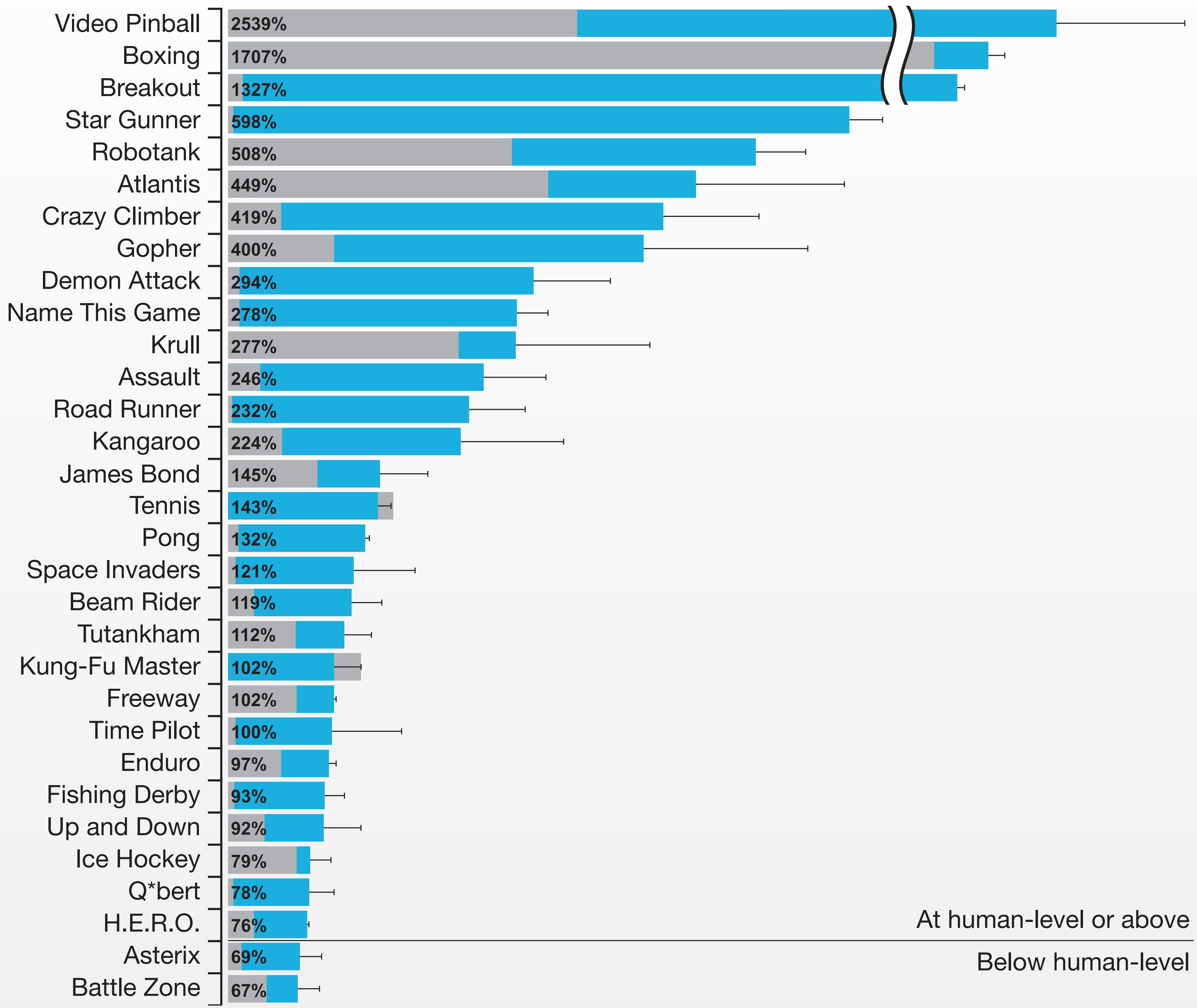


CNNs for Reinforcement Learning

CNNs for Reinforcement Learning



CNNs for Reinforcement Learning



CNNs for Reinforcement Learning