Applying your Convolutional Neural Networks

DEEP LEARNING FUNDAMENTALS PART 3

databricks

Logistics

- We can’t hear you…
- Recording will be available…
- Slides will be available…
- Code samples and notebooks will be available…
- Queue up Questions…

databricks

VISION	Accelerate innovation by unifying data science, engineering and business
PRODUCT	Unified Analytics Platform powered by Apache Spark™
WHO WE ARE	• Founded by the original creators of Apache Spark • Contributes 75% of the open source code, 10x more than any other company • Trained 100k+ Spark users on the Databricks platform

About our speaker

Denny Lee

Developer Advocate, Databricks

Former:

- Senior Director of Data Sciences Engineering at SAP Concur
- Principal Program Manager at Microsoft
    - Azure Cosmos DB Engineering Spark and Graph Initiatives
    - Isotope Incubation Team (currently known as HDInsight)
    - Bing's Audience Insights Team
    - Yahoo!'s 24TB Analysis Services cube

Deep Learning Fundamentals Series

This is a three-part series:

- Introduction to Neural Networks
- Training Neural Networks
- Applying your Convolutional Neural Network

This series will be make use of Keras (TensorFlow backend) but as it is a fundamentals series, we are focusing primarily on the concepts.

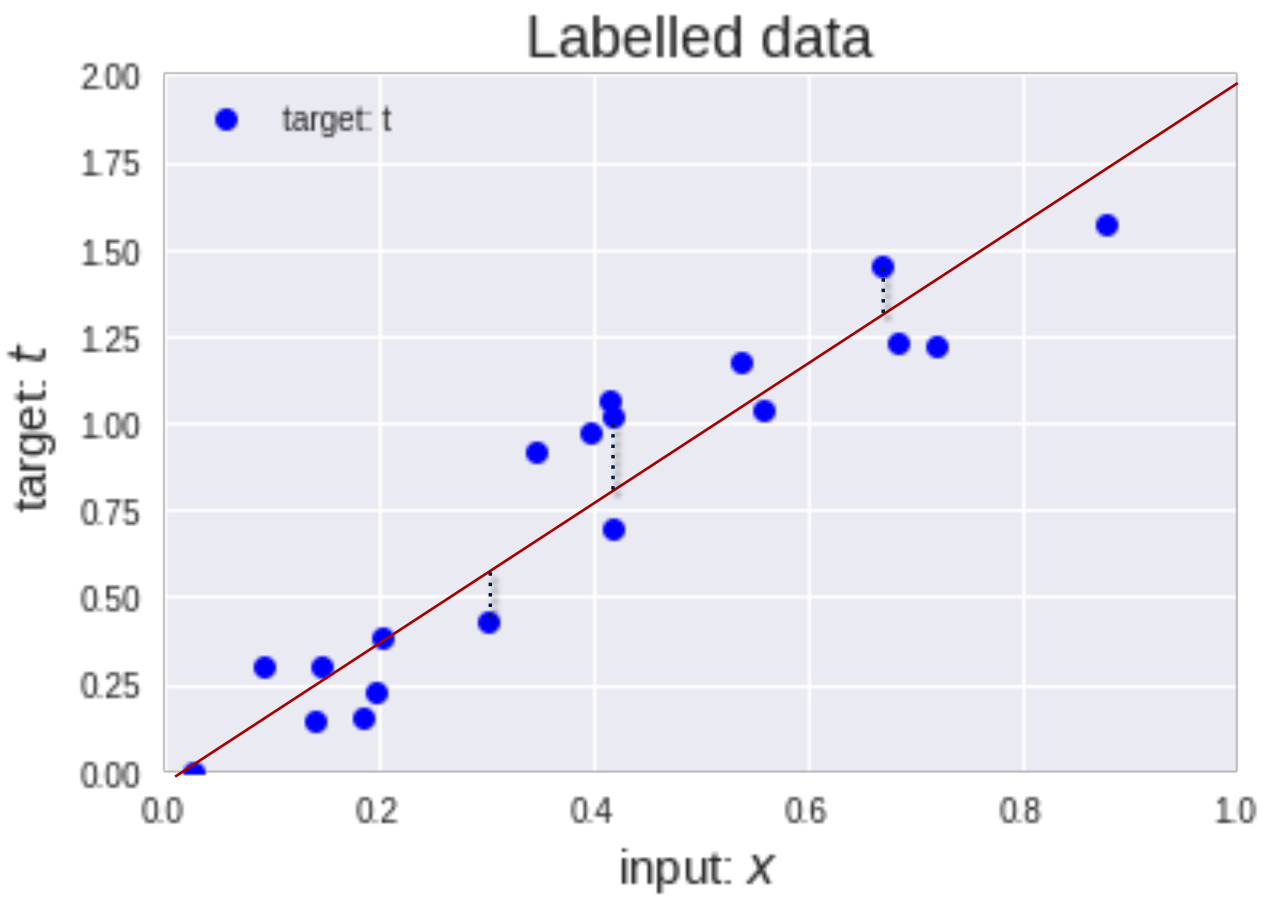
Current Session: Applying Neural Networks

- Diving further into CNNs
- CNN Architectures
- Convolutions at Work!

A quick review

Overfitting and underfitting

Cost function



Source: https://bit.ly/2IoAGzL

For this linear regression example, to determine the best p (slope of the line) for

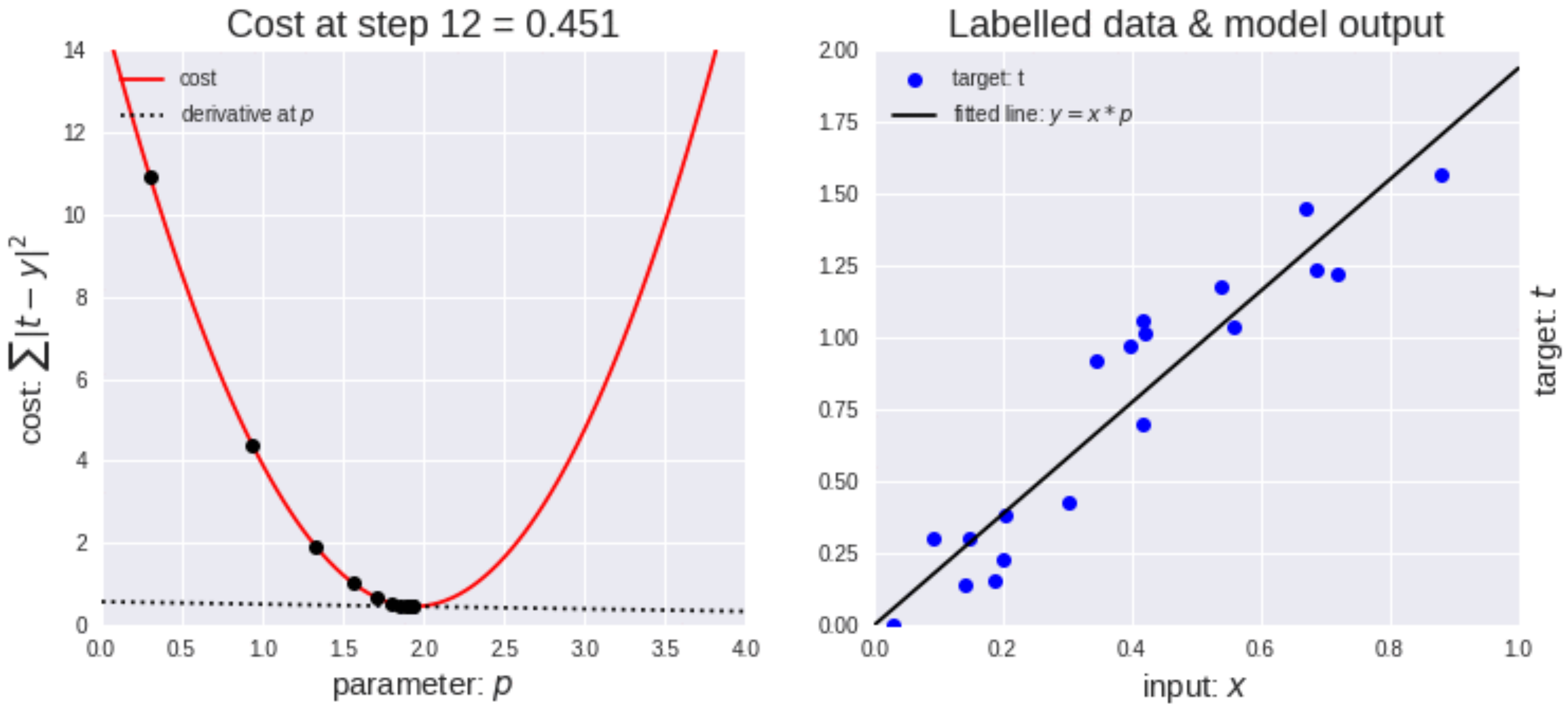
$$y = x \cdot p$$

we can calculate the cost function, such as Mean Square Error, Mean absolute error, Mean bias error, SVM Loss, etc.

For this example, we'll use sum of squared absolute differences

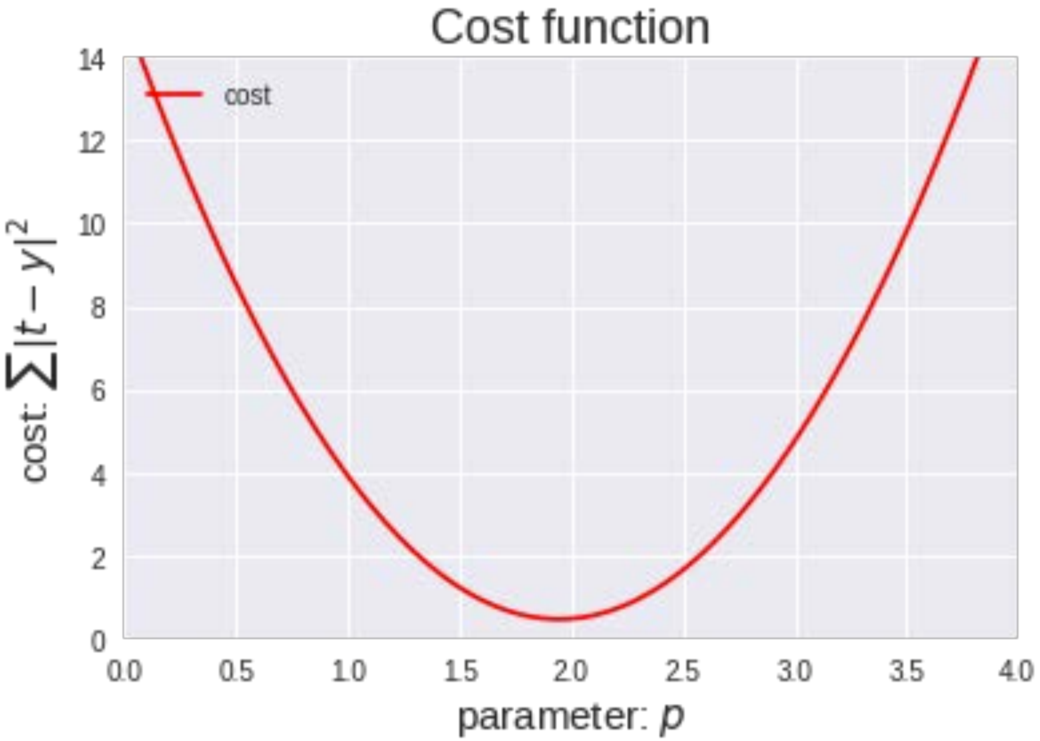
$$cost = \sum |t - y|^2$$

Gradient Descent Optimization



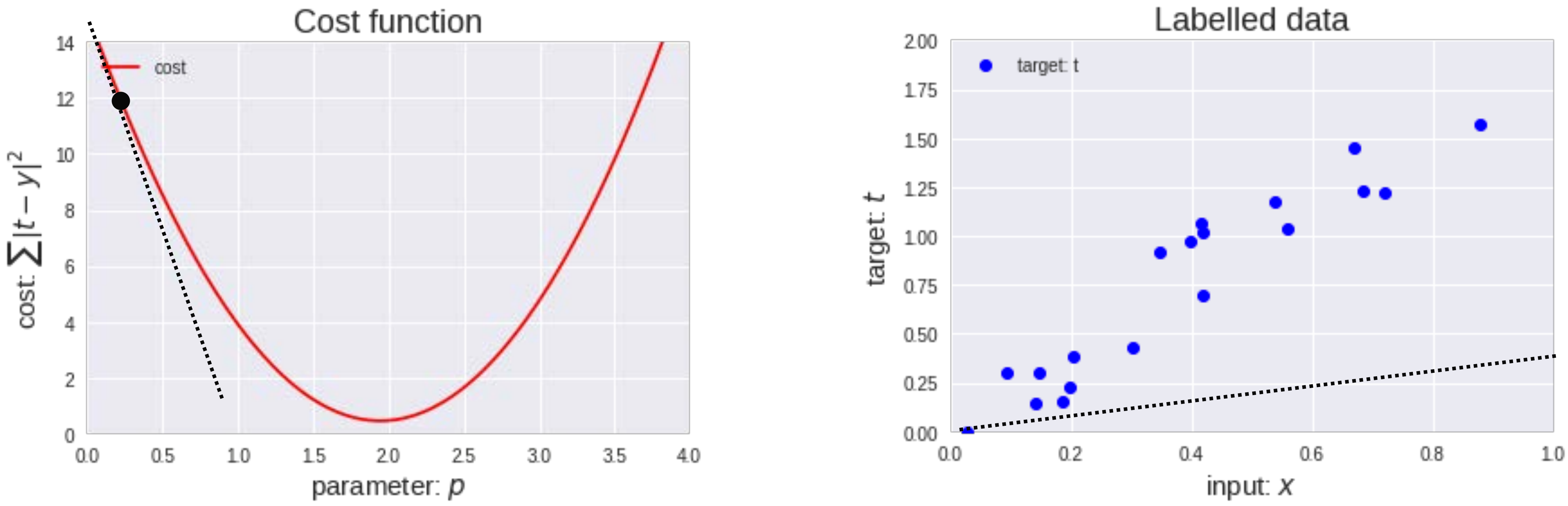
Source: https://bit.ly/2IoAGzL

Small Learning Rate



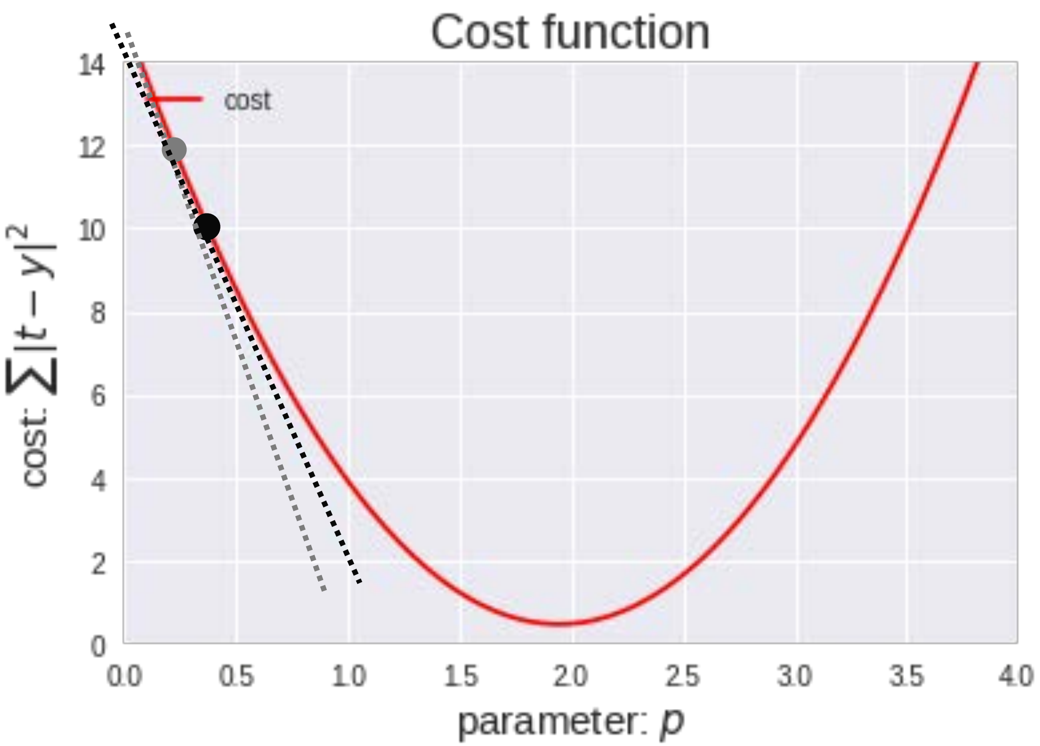
Source: https://bit.ly/2IoAGzL

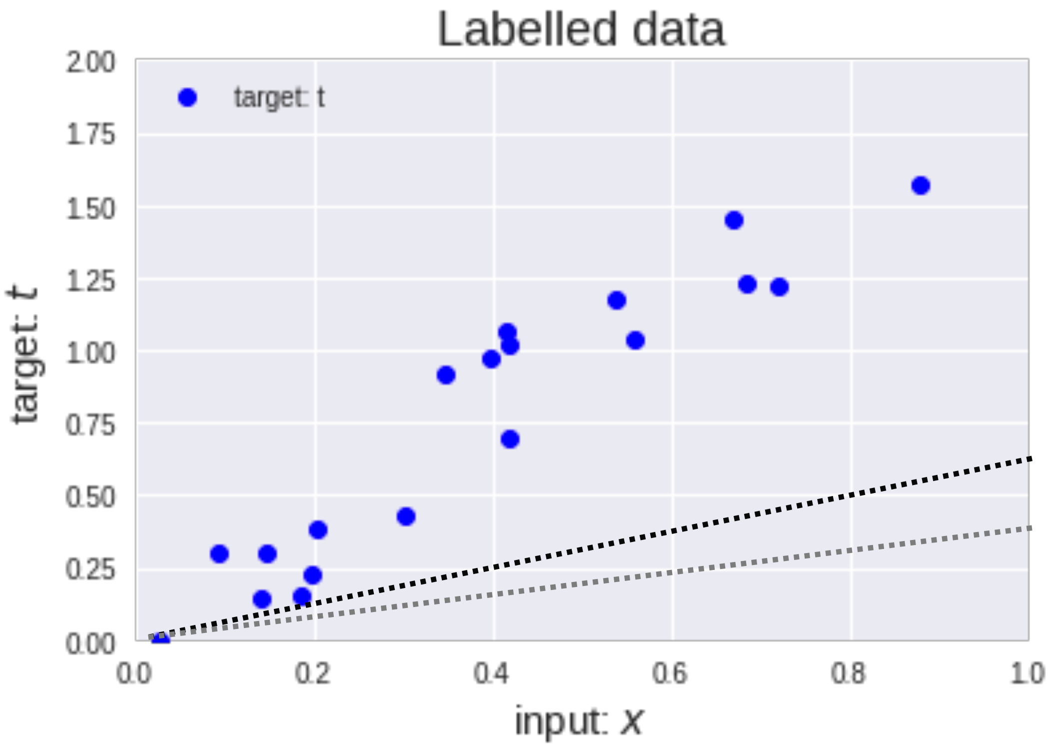
Small Learning Rate



Source: https://bit.ly/2IoAGzL

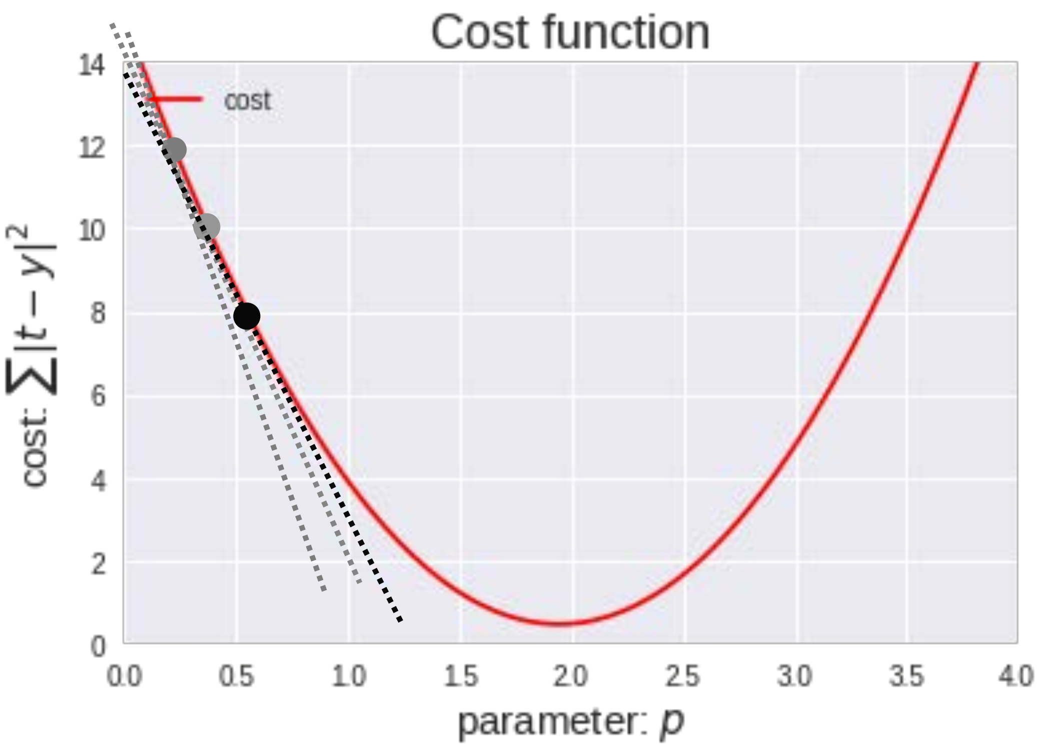
Small Learning Rate

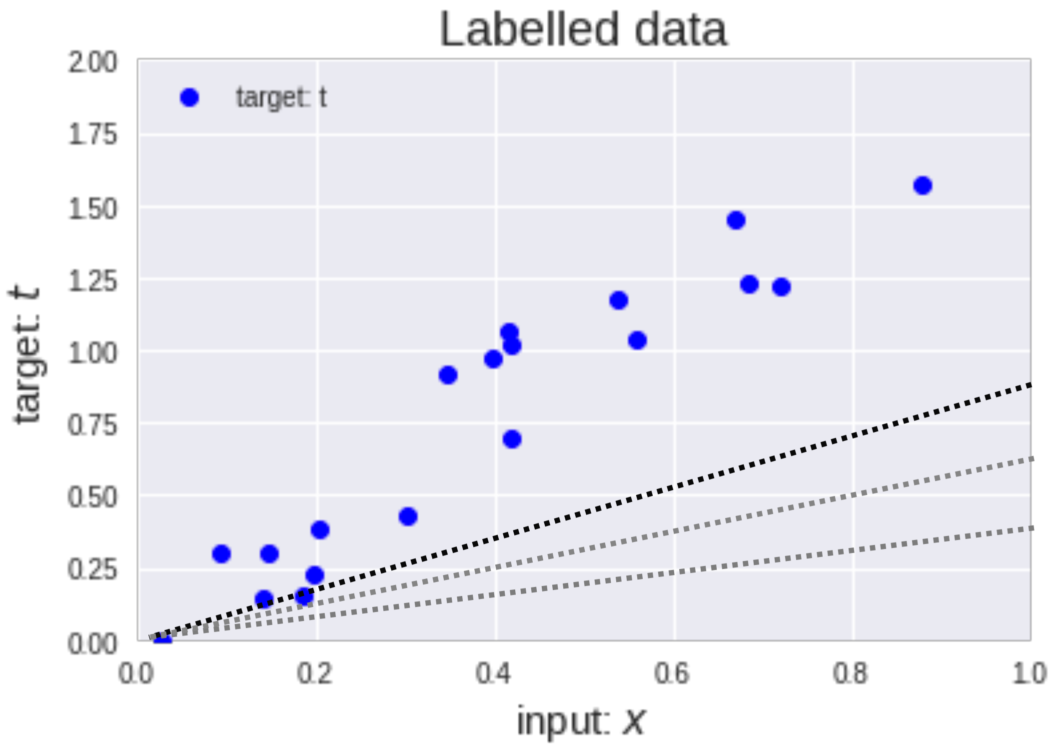




Source: https://bit.ly/2IoAGzL

Small Learning Rate





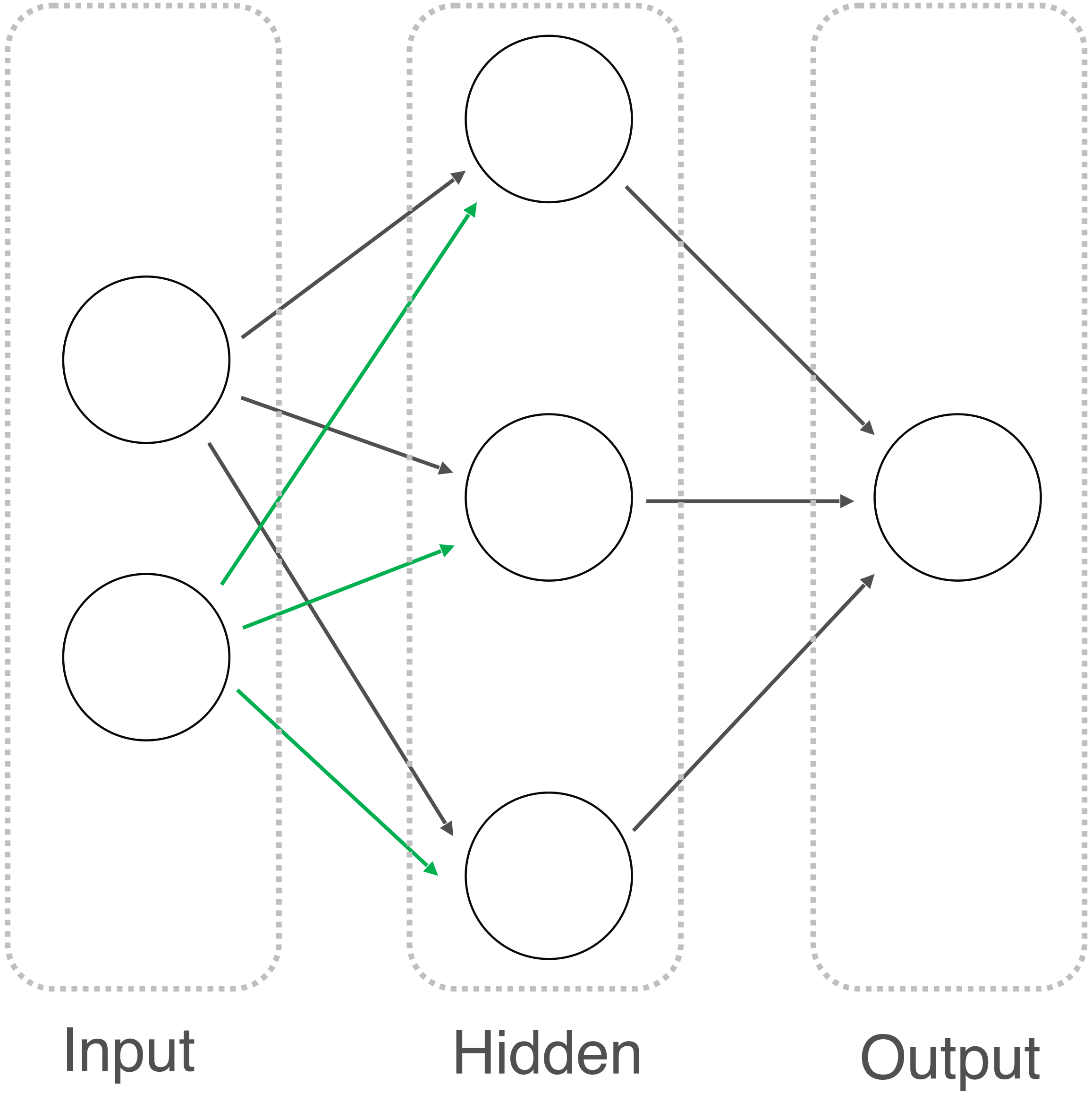
Source: https://bit.ly/2IoAGzL

Hyperparameters: Activation Functions?

- Good starting point: ReLU
  - Note many neural networks samples: Keras MNIST, TensorFlow CIFAR10 Pruning, etc.
  - Note that each activation function has its own strengths and weaknesses. A good quote on activation functions from CS231N summarizes the choice well:

"What neuron type should I use?" Use the ReLU non-linearity, be careful with your learning rates and possibly monitor the fraction of "dead" units in a network. If this concerns you, give Leaky ReLU or Maxout a try. Never use sigmoid. Try tanh, but expect it to work worse than ReLU/Maxout.

Simplified Two-Layer ANN



Do I snowboard this weekend?

$x_1 \rightarrow$ *Apres Ski'er*

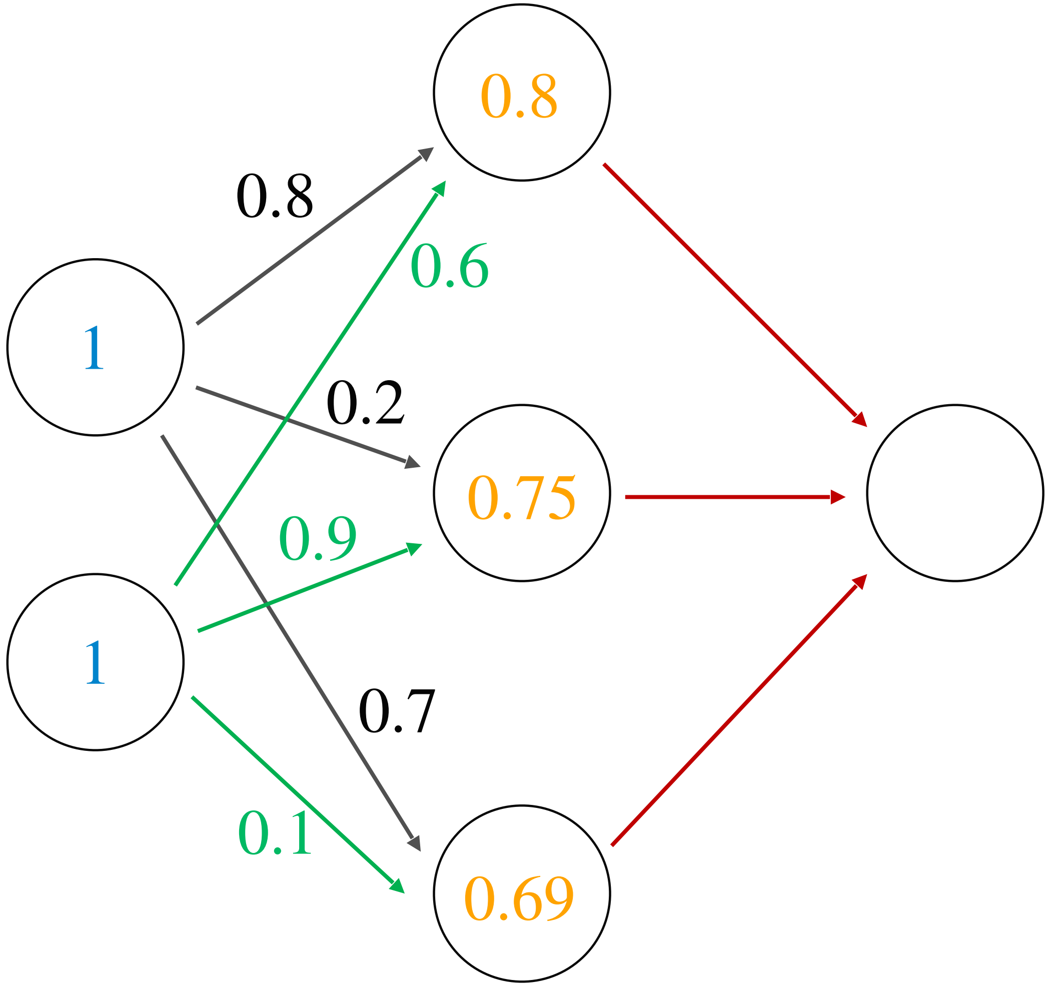
$x_2 \rightarrow$ *Shredder*

$h_1 \rightarrow$ *weather*

$h_2 \rightarrow$ *powder*

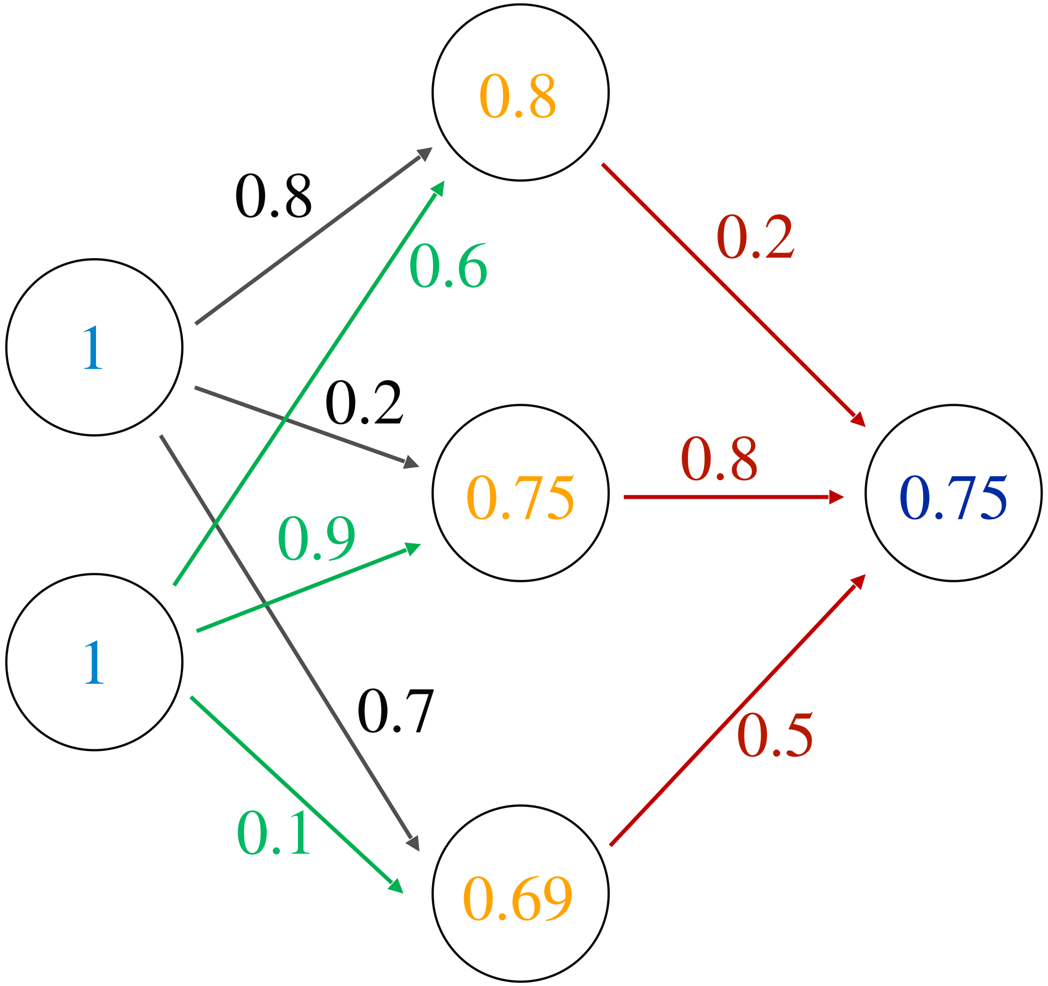
$h_3 \rightarrow$ *driving*

Simplified Two-Layer ANN



$$h_1 = \sigma(1x0.8 + 1x0.6) = 0.80$$
$$h_2 = \sigma(1x0.2 + 1x0.9) = 0.75$$
$$h_3 = \sigma(1x0.7 + 1x0.1) = 0.69$$

Simplified Two-Layer ANN

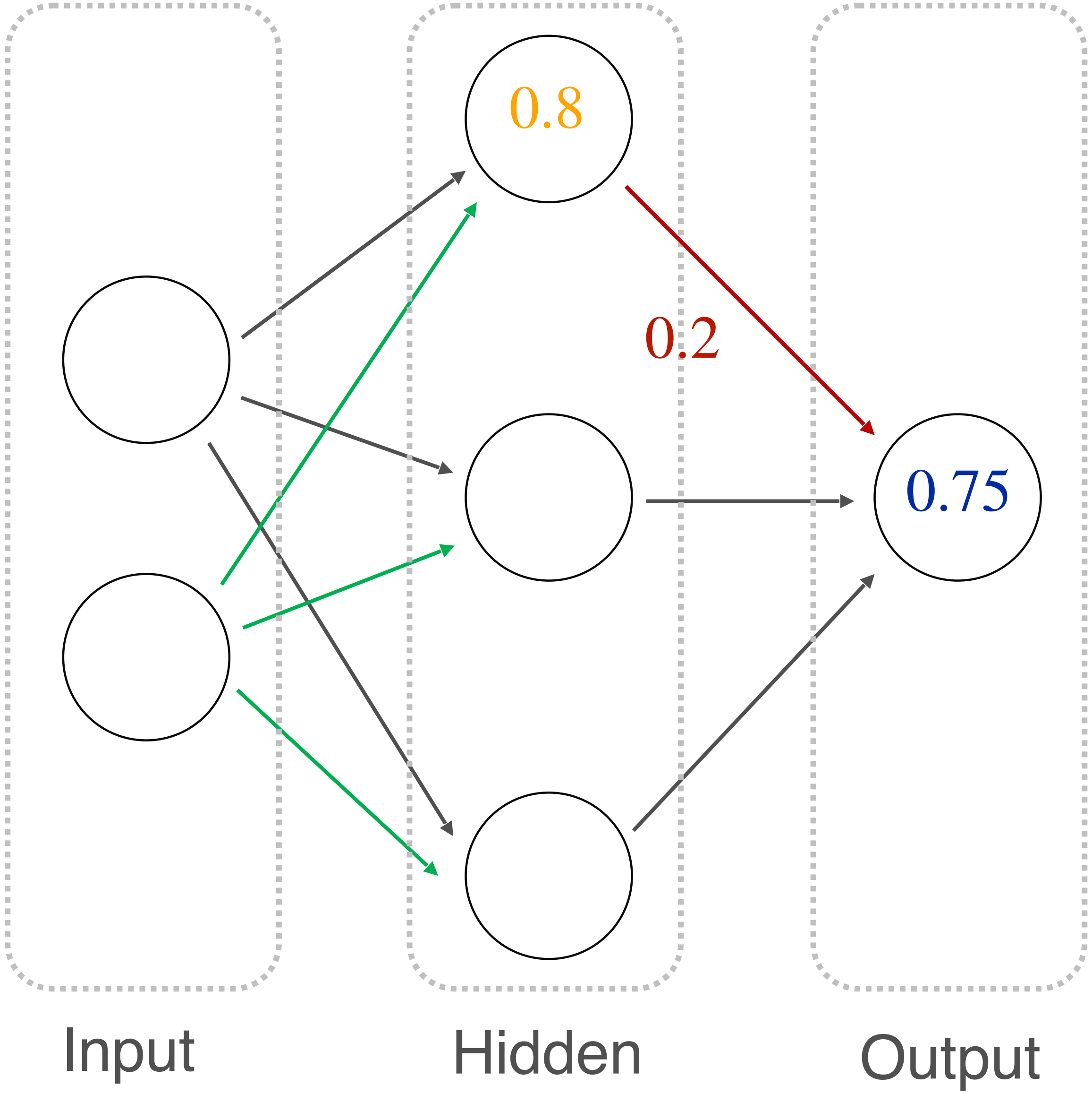


$$out = \sigma(0.2x0.8 + 0.8x0.75 + 0.5x0.69)$$

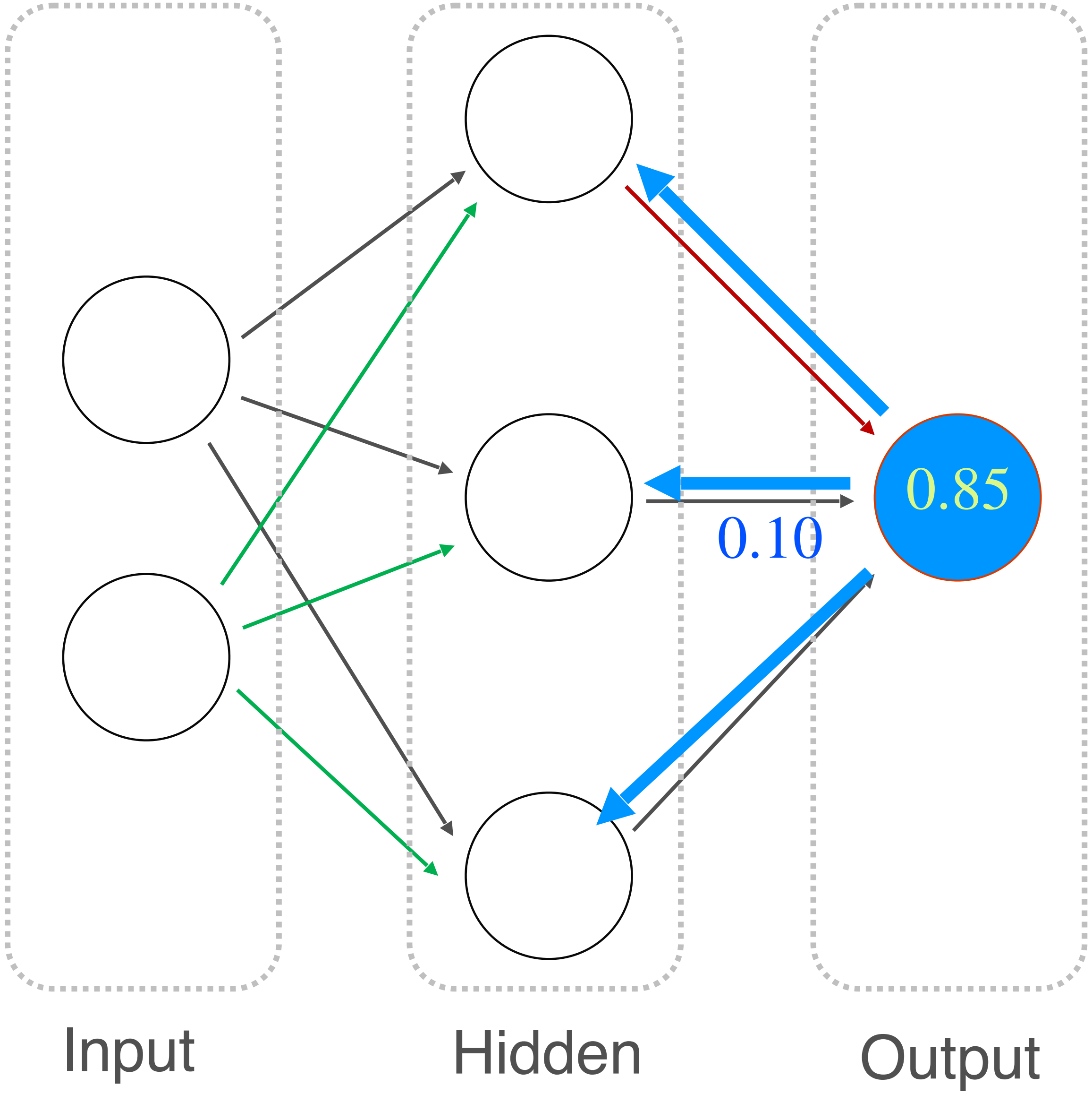
$$= \sigma(1.105)$$

$$= 0.75$$

Backpropagation

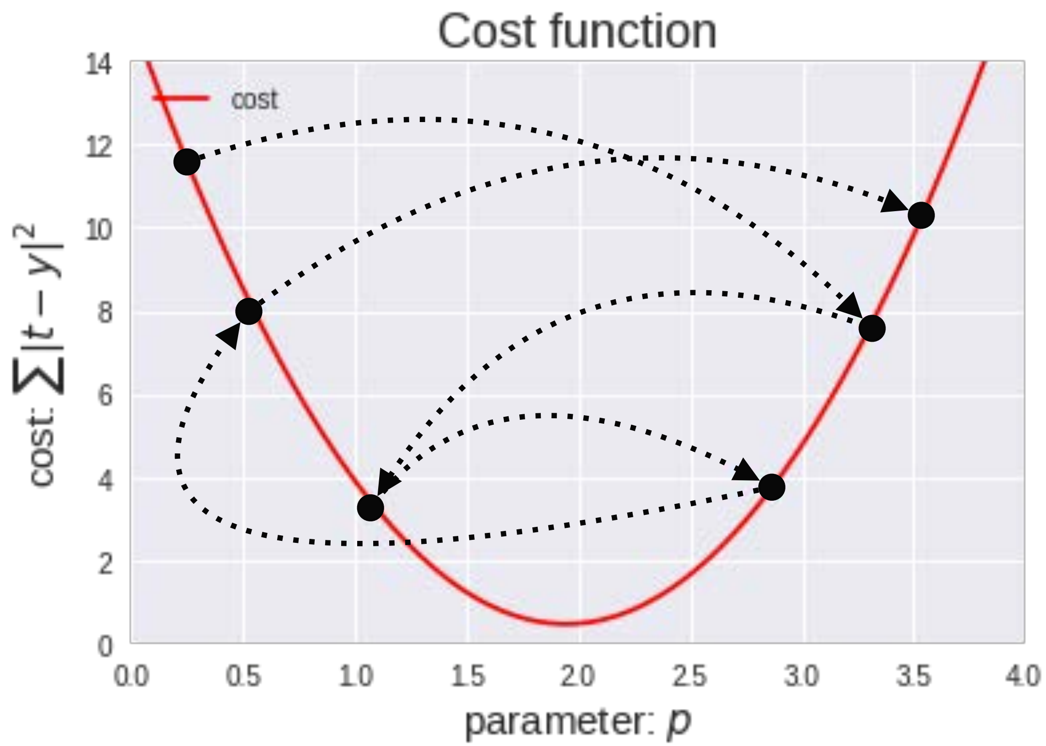


Backpropagation



- Backpropagation: calculate the gradient of the cost function in a neural network
- Used by gradient descent optimization algorithm to adjust weight of neurons
- Also known as backward propagation of errors as the error is calculated and distributed back through the network of layers

Sigmoid function (continued)



Source: https://bit.ly/2IoAGzL

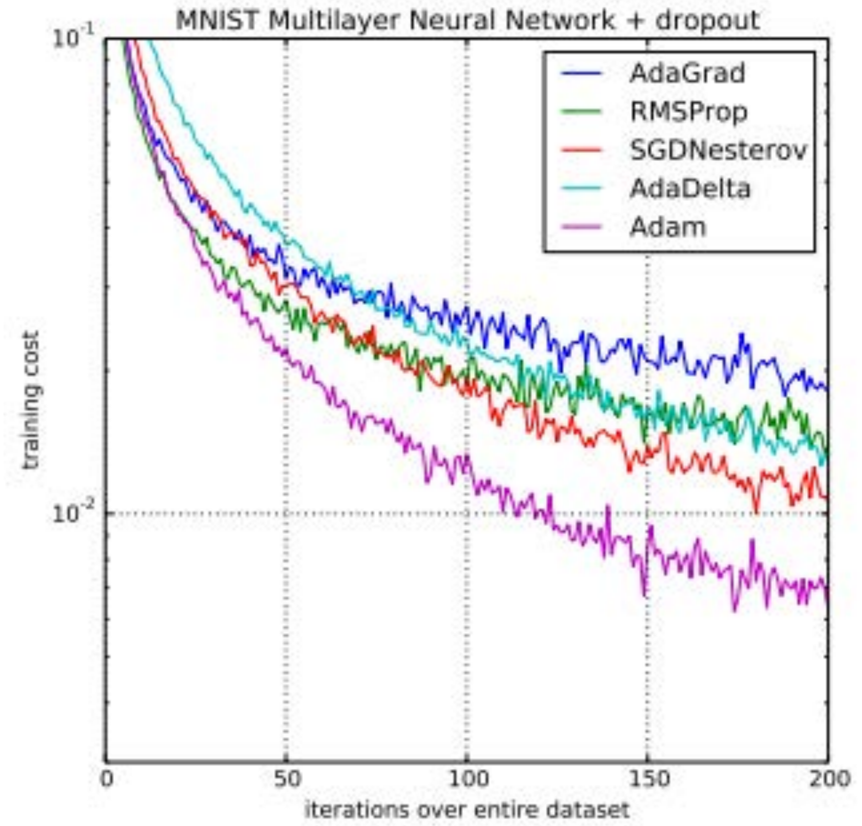
Output is not zero-centered: During gradient descent, if all values are positive then during backpropagation the weights will become all positive or all negative creating zig zagging dynamics.

Learning Rate Callouts

- Too small, it may take too long to get minima
- Too large, it may skip the minima altogether

Which Optimizer?

Source: https://goo.gl/2da4WY



"In practice Adam is currently recommended as the default algorithm to use, and often works slightly better than RMSProp. However, it is often also worth trying SGD+Nesterov Momentum as an alternative.."

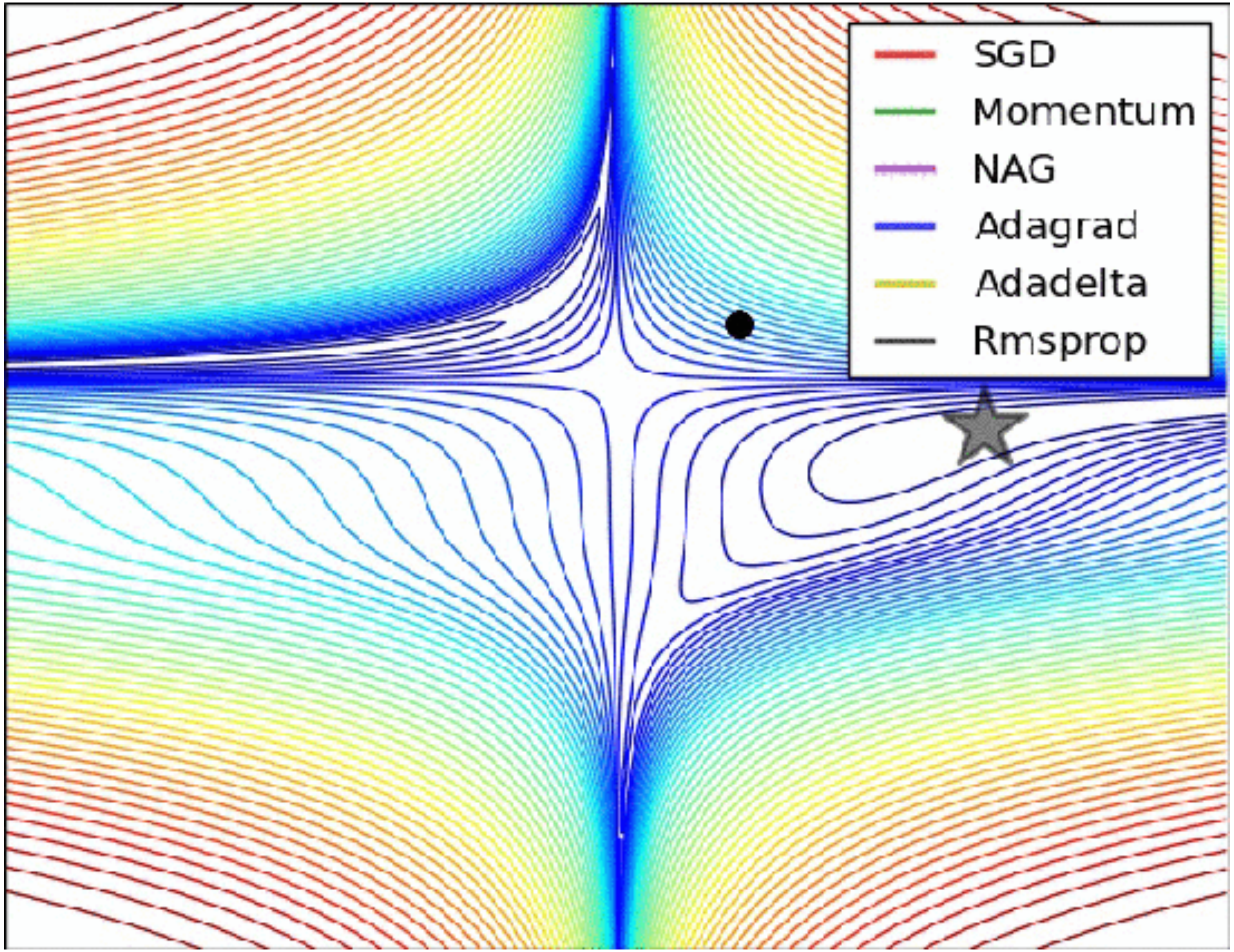
Andrej Karpathy, et al, CS231n

Comparison of Adam to Other Optimization Algorithms Training a Multilayer Perceptron
Taken from Adam: A Method for Stochastic Optimization, 2015.

Optimization on loss surface contours

Source: http://cs231n.github.io/neural-networks-3/#hyper

Image credit: [Alec Radford](#)



Adaptive algorithms converge quickly and find the right direction for the parameters.

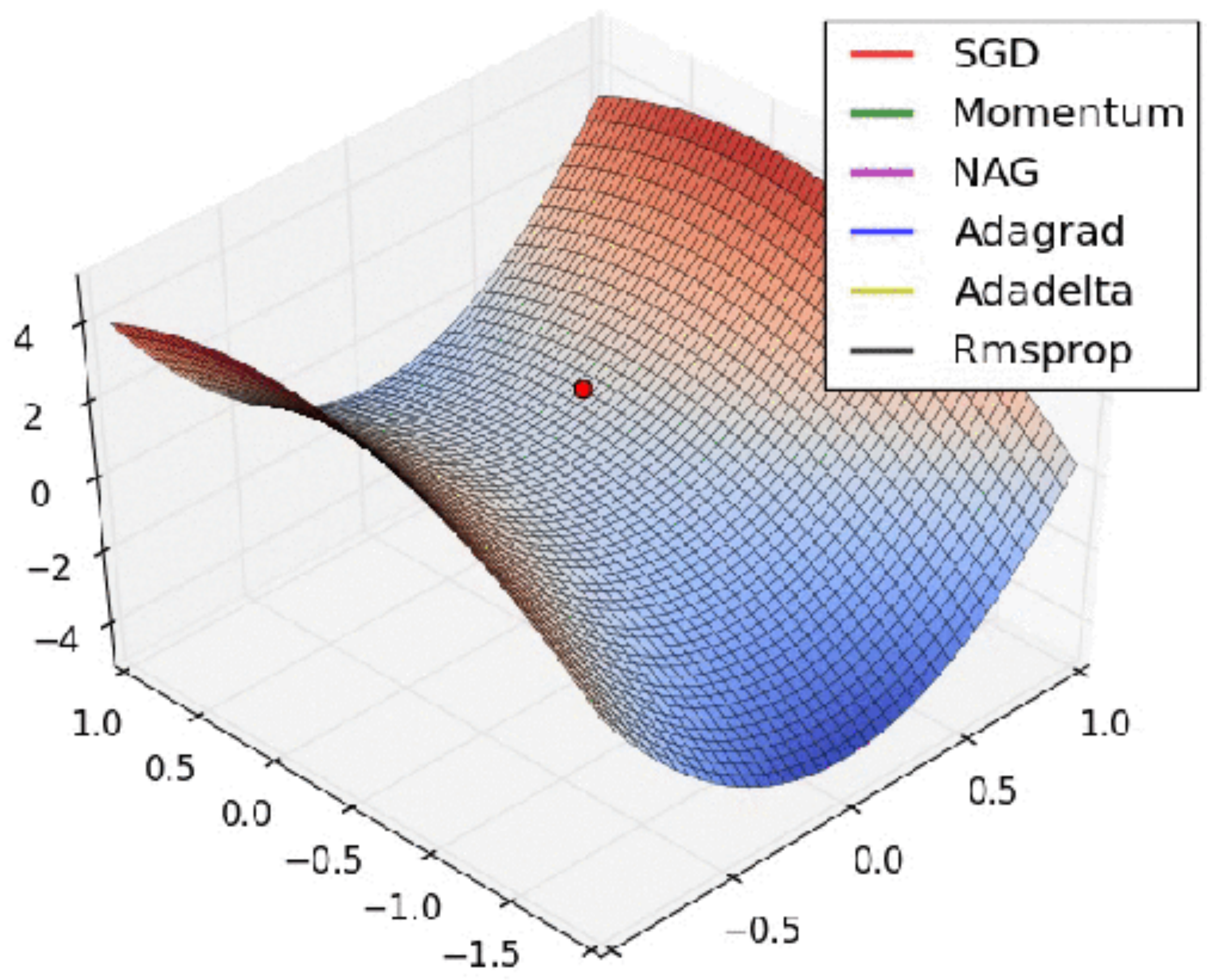
In comparison, SGD is slow

Momentum-based methods overshoot

Optimization on saddle point

Source: http://cs231n.github.io/neural-networks-3/#hyper

Image credit: [Alec Radford](https://twitter.com/alecrad)



Notice how SGD gets stuck near the top

Meanwhile adaptive techniques optimize the fastest

Good References

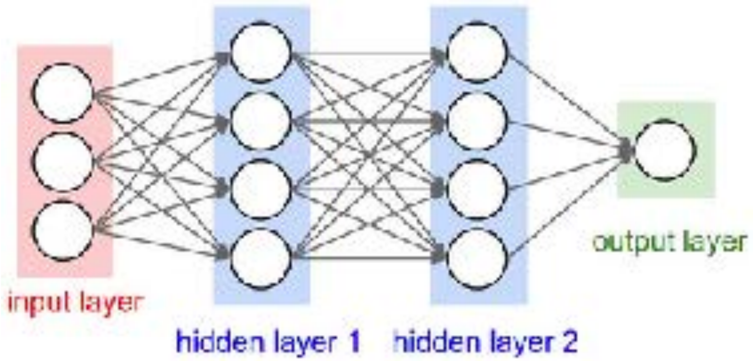
- Suki Lau's Learning Rate Schedules and Adaptive Learning Rate Methods for Deep Learning
- CS23n Convolutional Neural Networks for Visual Recognition
- Fundamentals of Deep Learning
- ADADELTA: An Adaptive Learning Rate Method
- Gentle Introduction to the Adam Optimization Algorithm for Deep Learning

Convolutional Networks

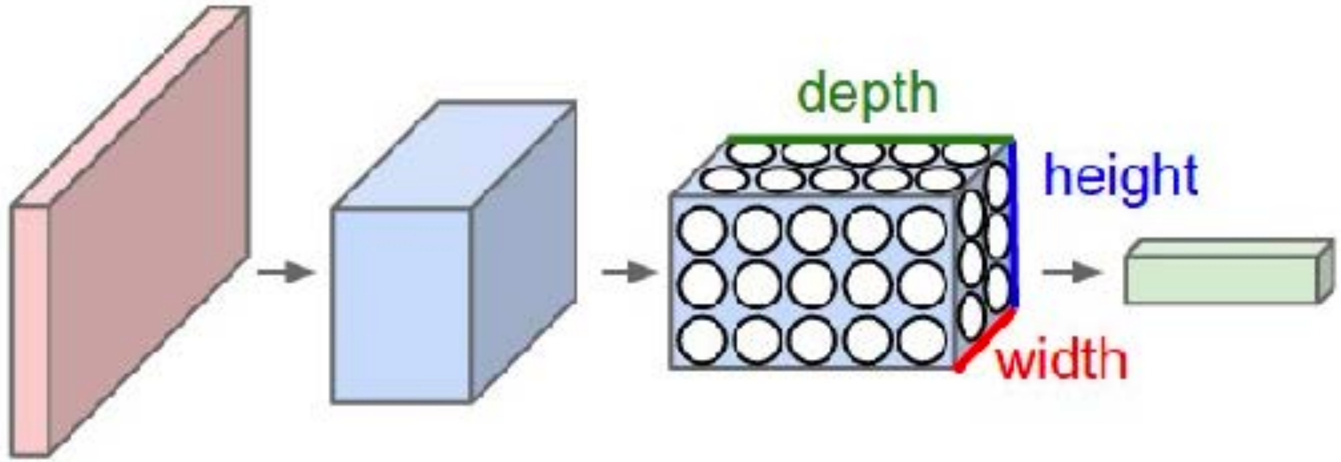
Convolutional Neural Networks

- Similar to Artificial Neural Networks but CNNs (or ConvNets) make explicit assumptions that the input are images
- Regular neural networks do not scale well against images
  - E.g. CIFAR-10 images are 32x32x3 (32 width, 32 height, 3 color channels) = 3072 weights – somewhat manageable
  - A larger image of 200x200x3 = 120,000 weights
- CNNs have neurons arranged in 3D: width, height, depth.
  - Neurons in a layer will only be connected to a small region of the layer before it, i.e. NOT all of the neurons in a fully-connected manner.
  - Final output layer for CIFAR-10 is 1x1x10 as we will reduce the full image into a single vector of class scores, arranged along the depth dimension

CNNs / ConvNets



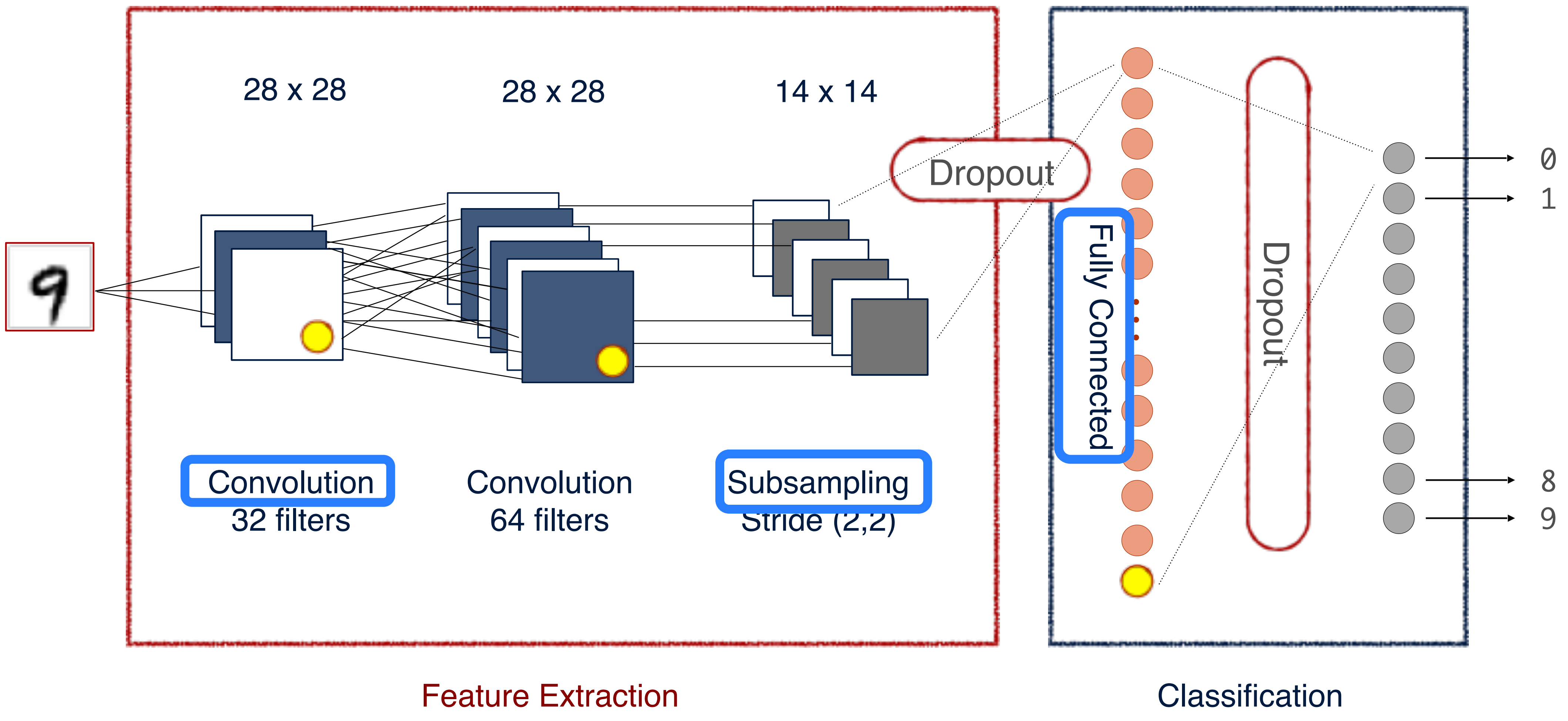
Regular 3-layer neural network



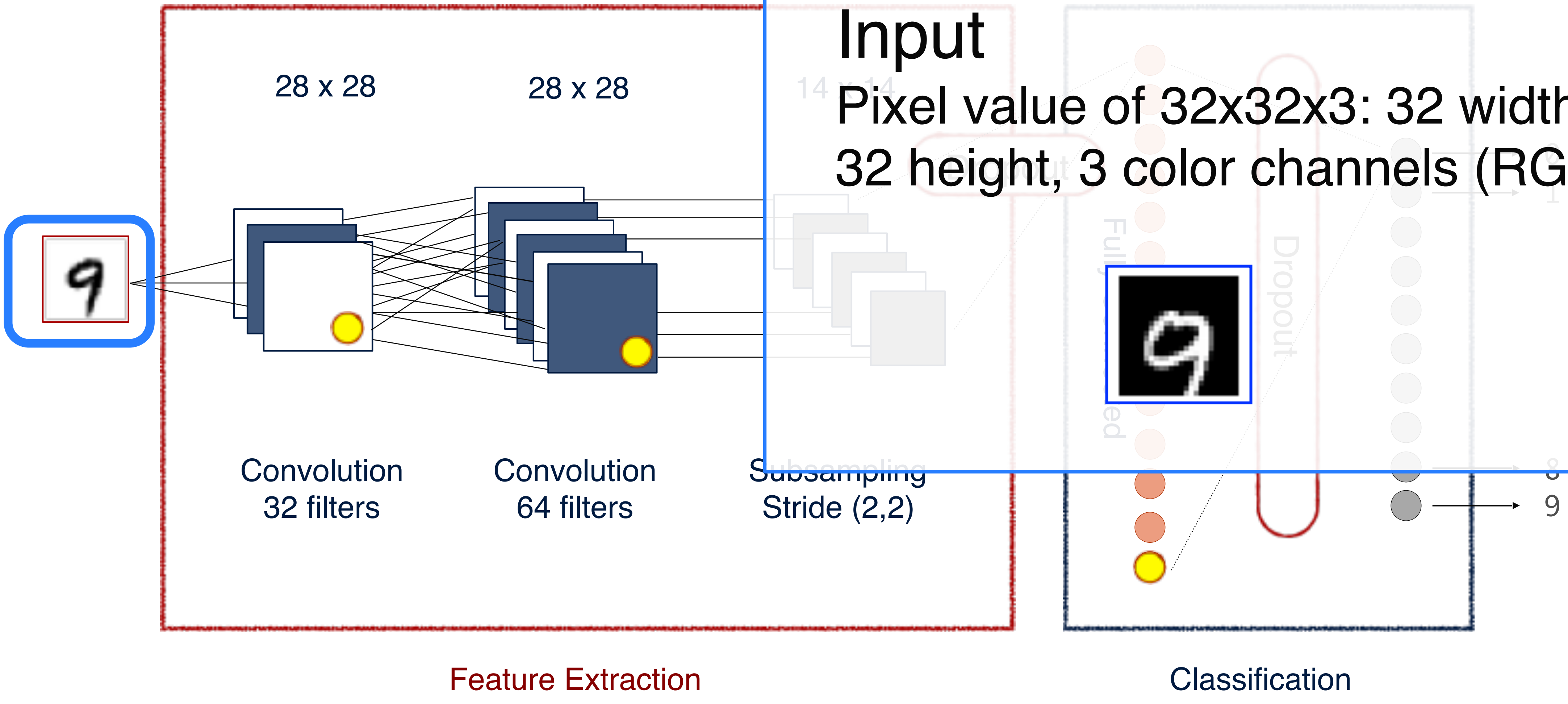
- ConvNet arranges neurons in 3 dimensions
- 3D input results in 3D output

Source: https://cs231n.github.io/convolutional-networks/

Convolutional Neural Networks



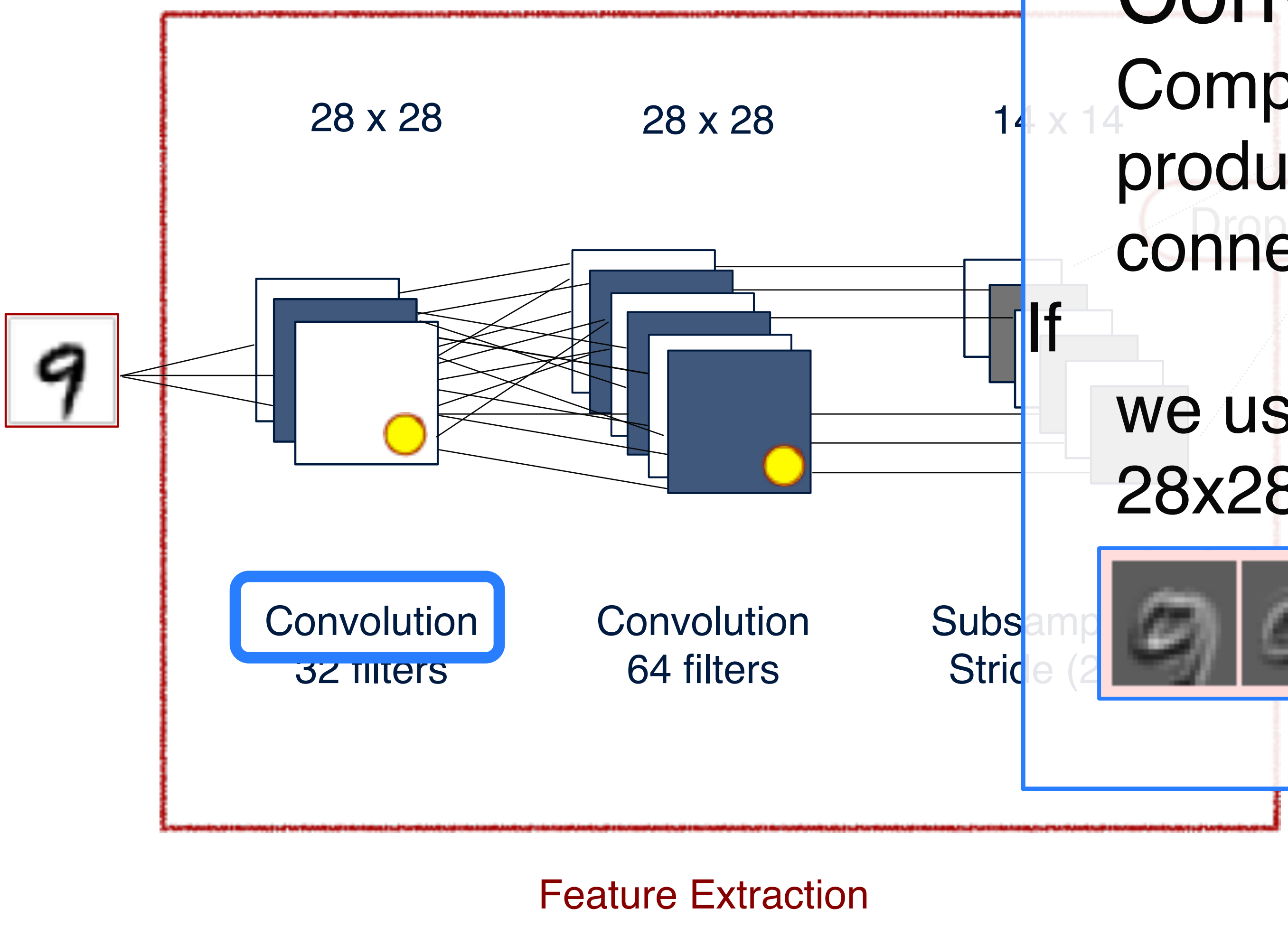
Convolutional Neural Networks



Input

Pixel value of 32x32x3: 32 width, 32 height, 3 color channels (RGB)

Convolutional Neural Networks



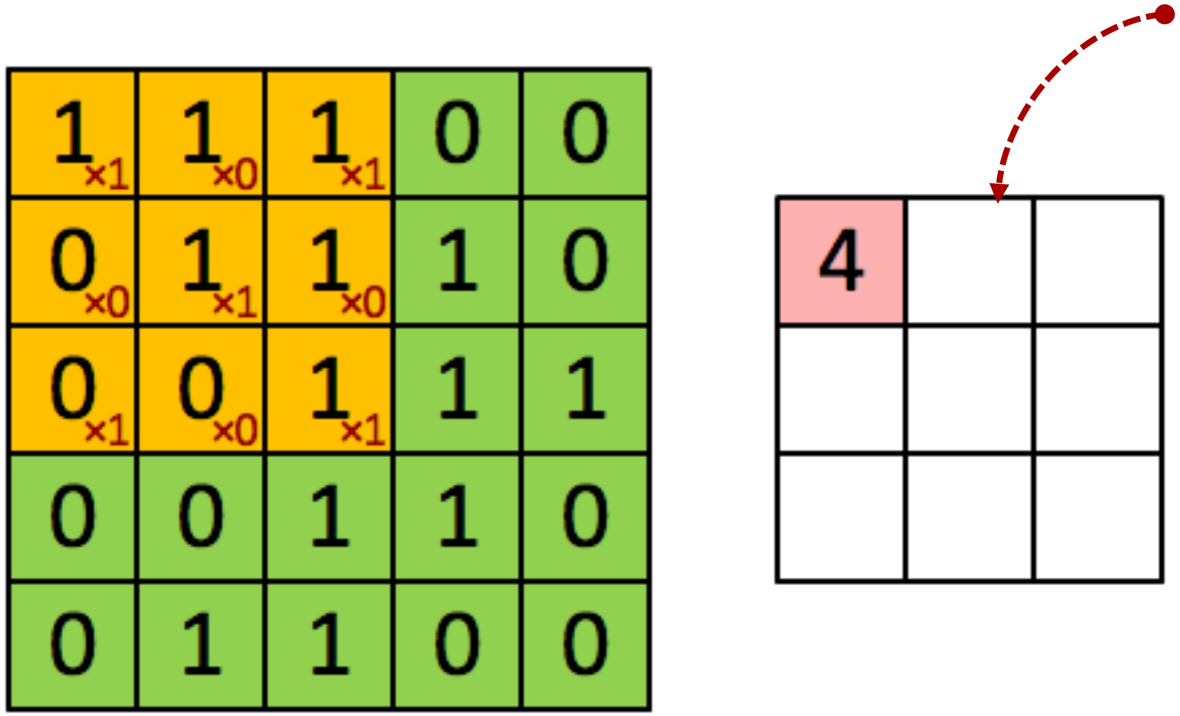
Convolutions

Compute output of neurons (dot product between their weights) connected to a small local region.

If we use 32 filters, then the output is 28x28x32 (using 5x5 filter)

Convolution: Kernel = Filter

kernel = filter = feature detector



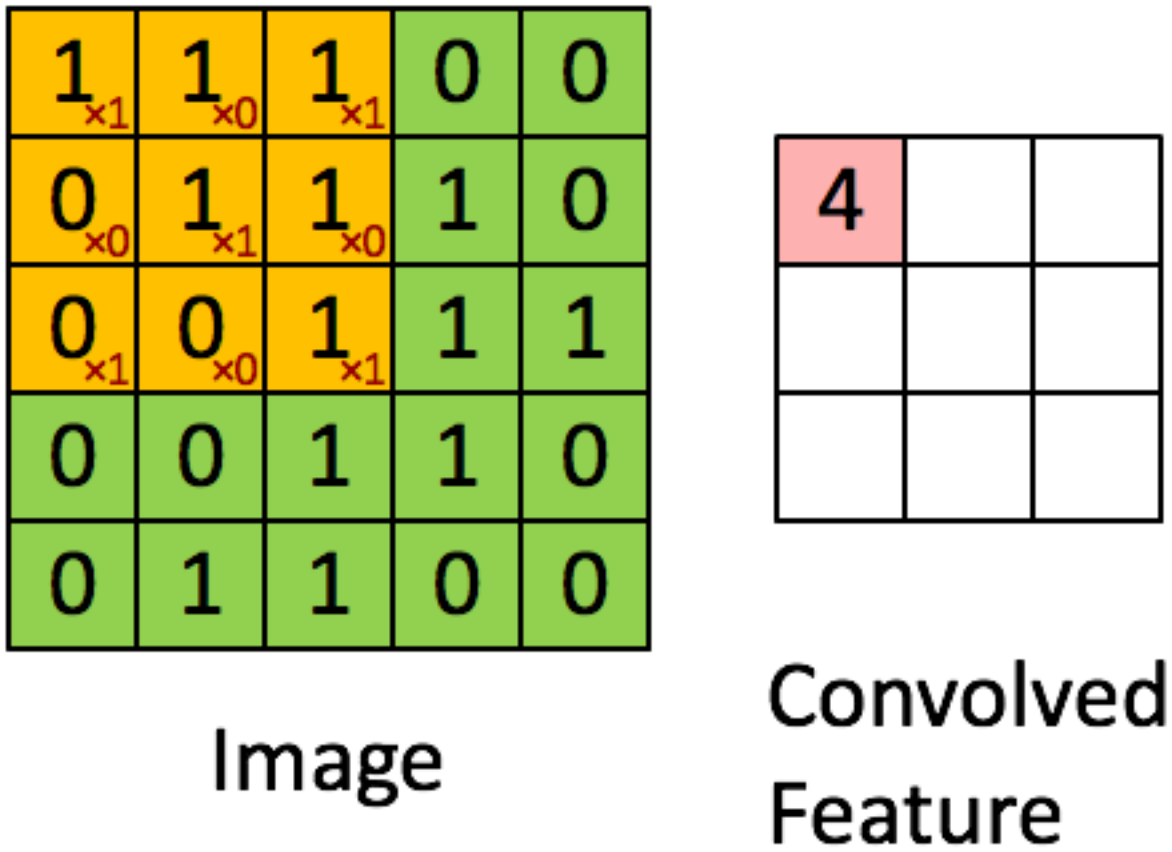
Image

Convolved Feature

During forward pass, we slide over the image spatially (i.e. convolve) each filter across the width and height, computing dot products.

Source: http://deeplearning.stanford.edu/wiki/index.php/Feature_extraction_using_convolution

Convolution: Local Connectivity

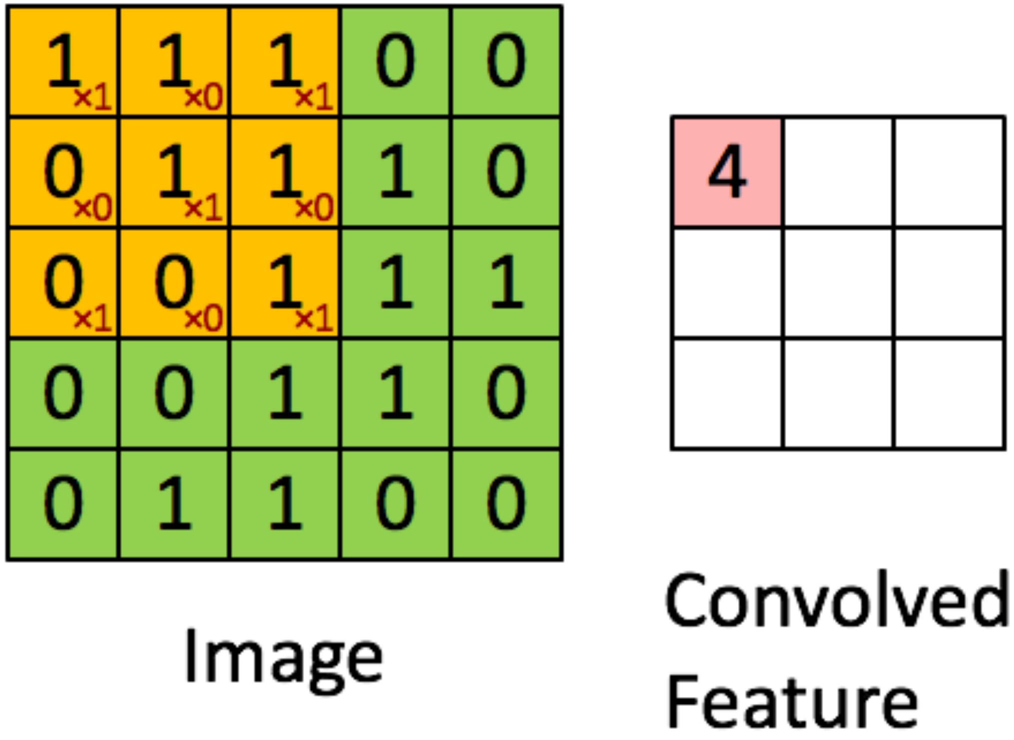


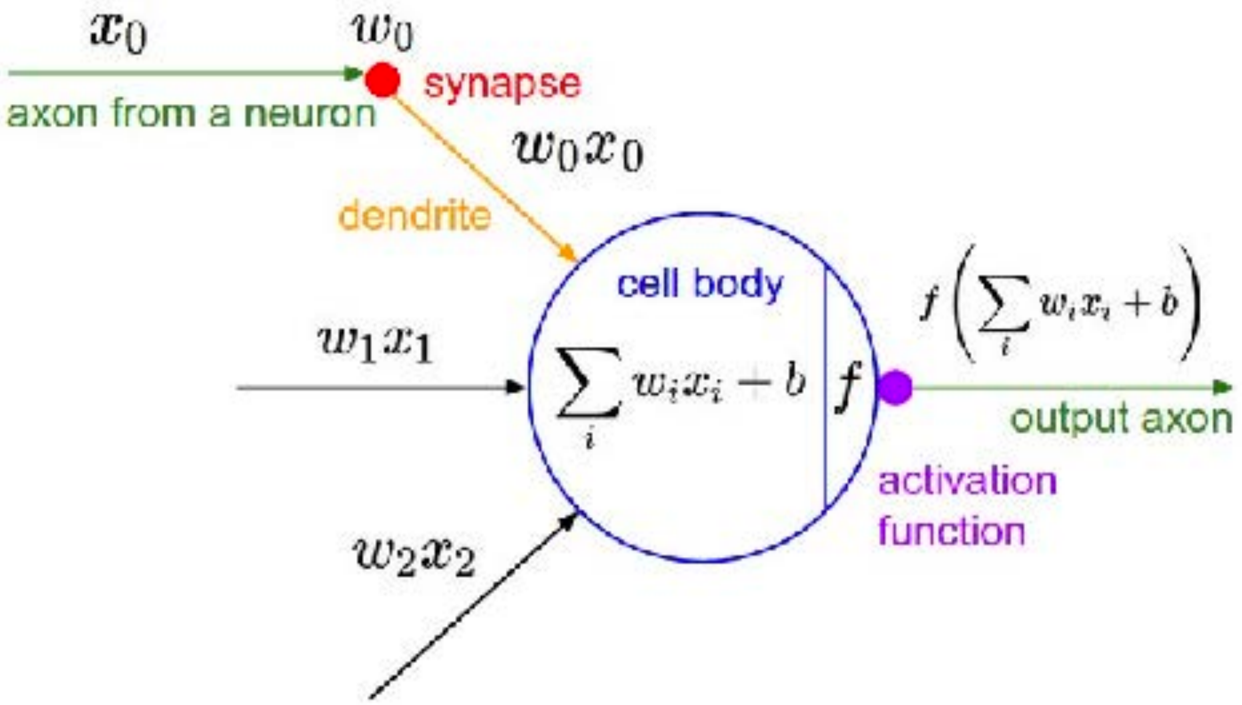
Connect neurons to only a small local region as it is impractical to connect to all neurons. Depth of filter = depth of input volume.

Source: http://deeplearning.stanford.edu/wiki/index.php/Feature_extraction_using_convolution

Convolution: Local Connectivity

The neurons still compute a dot product of their weights with the input followed by a non-linearity, but their connectivity is now restricted to be local spatially.



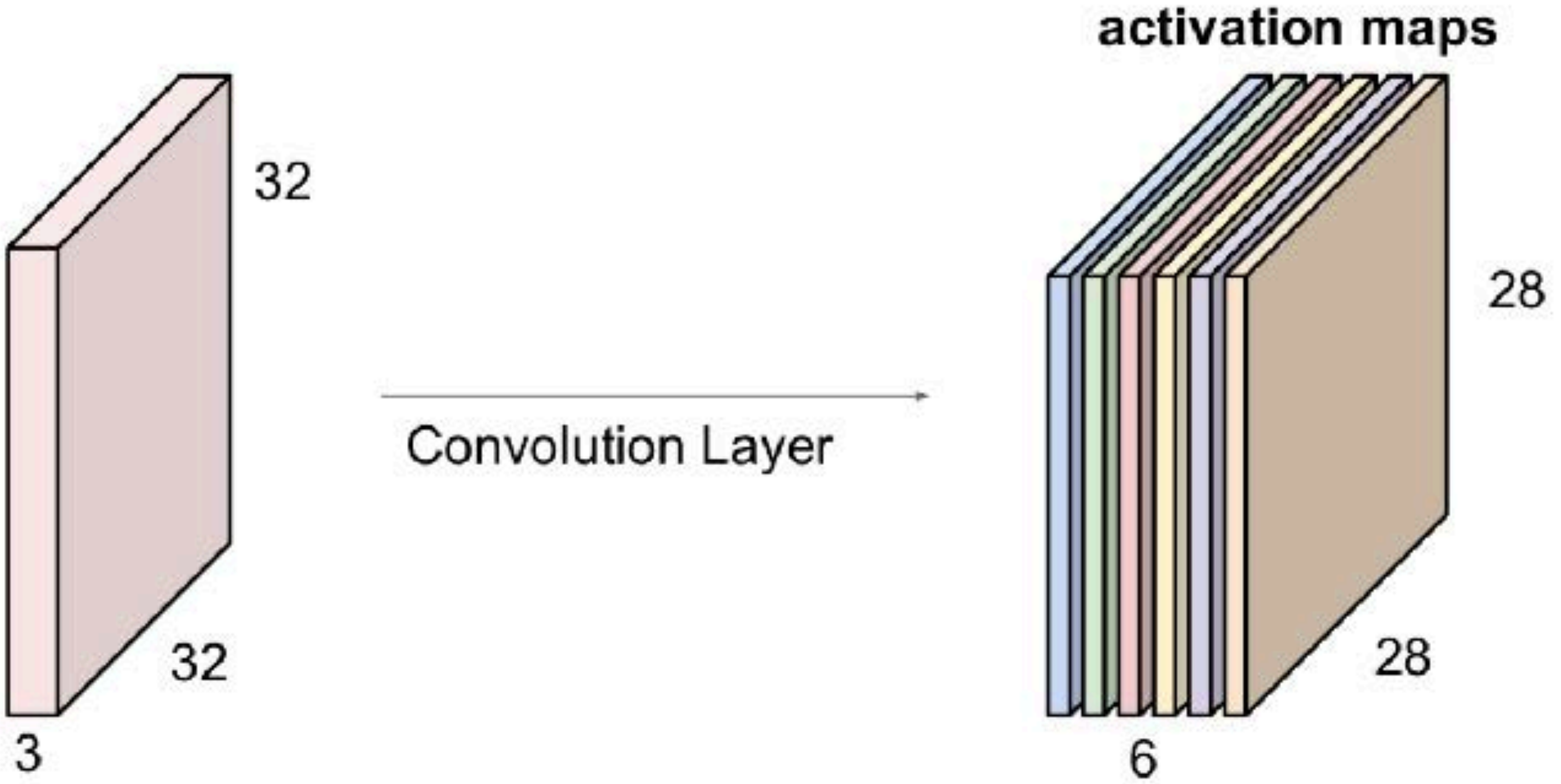


Source: http://deeplearning.stanford.edu/wiki/index.php/Feature_extraction_using_convolution

Source: https://cs231n.github.io/convolutional-networks/

Convolution Goal

Goal is to create an entire set of filters in each CONV layer and produce 2D activation maps (e.g. for 6 filters)



Source: http://cs231n.stanford.edu/slides/2017/cs231n_2017_lecture5.pdf

Convolution Example with 8 filters

Activations:

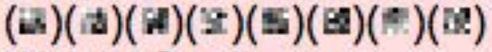
Activation Gradients:

conv (24x24x8)
filter size 5x5x1, stride 1
max activation: 2.72095, min: -2.44127
max gradient: 0.01954, min: -0.02194
parameters: 8x5x5x1+8 = 208

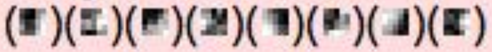
Activations:

Activation Gradients:

Weights:

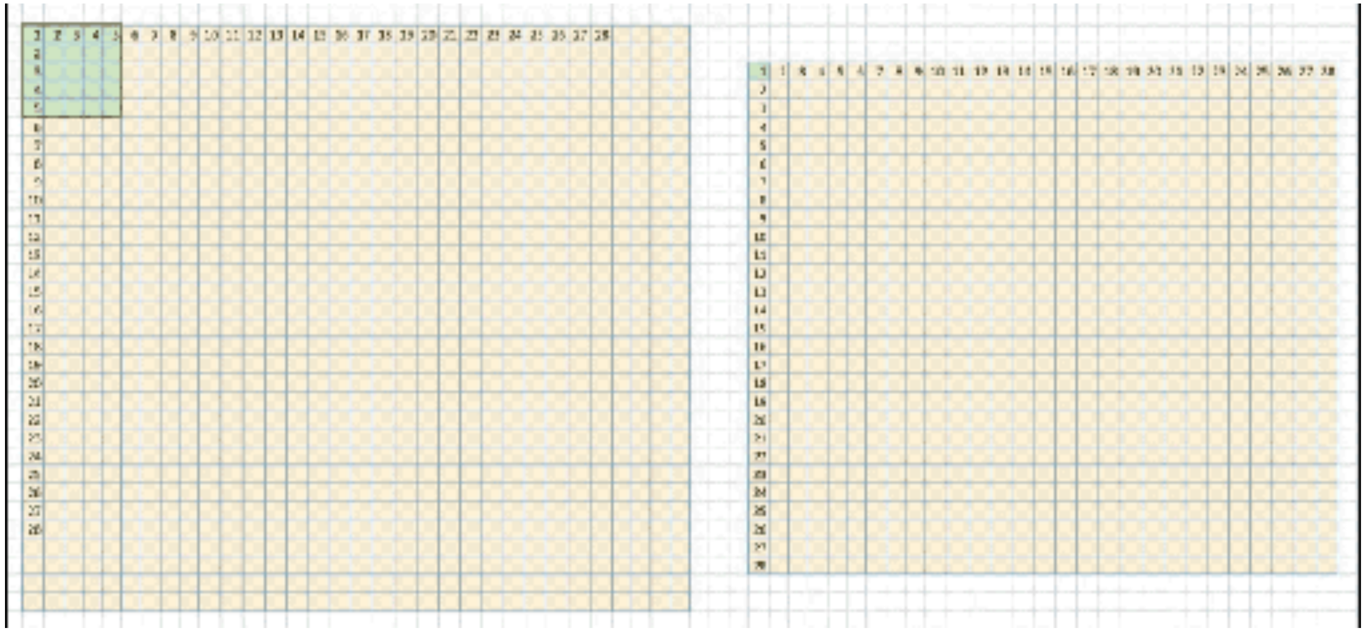


Weight Gradients:



Source: https://cs.stanford.edu/people/karpathy/convnetjs/demo/mnist.html

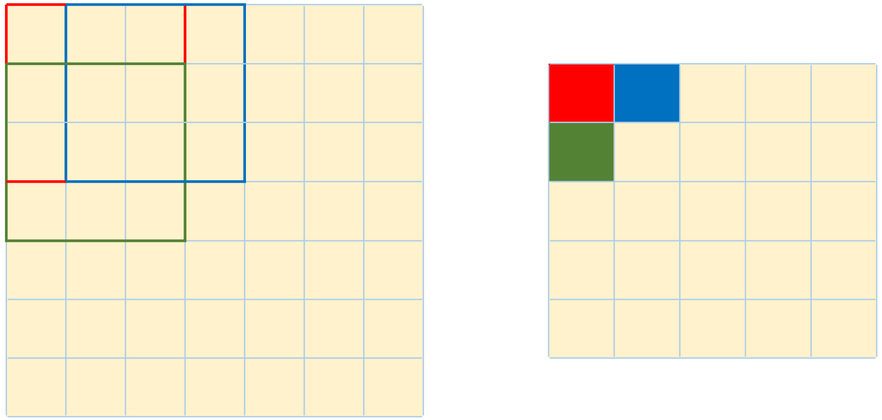
Convolution: 32x32 to 28x28 using 5x5 filter



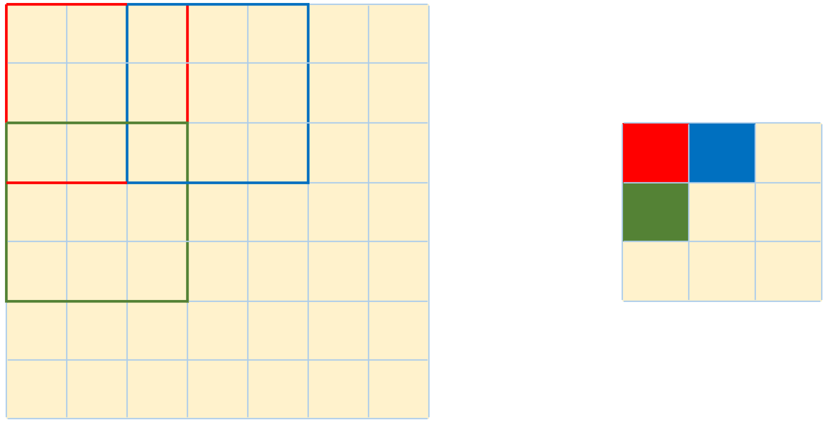
Taking a stride…

Source: http://cs231n.stanford.edu/slides/2017/cs231n_2017_lecture5.p

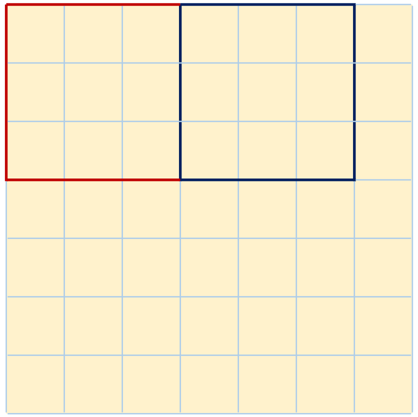
Stride = 1



Stride = 2



Stride = 3



Does not fit!

Output Size:

(N – F) / stride + 1

e.g. N = 7, F = 3:

Stride 1 = 5

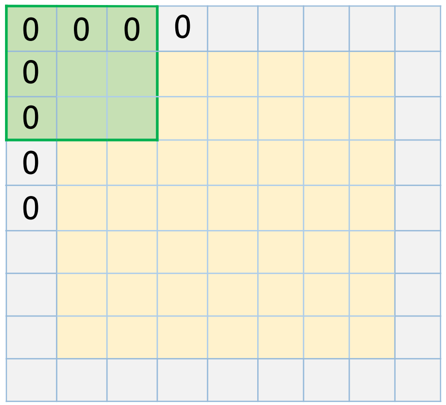
Stride 2 = 3

Stride 3 = 2.33 (doh!)

Padding

Source: http://cs231n.stanford.edu/slides/2017/cs231n_2017_lecture5.p

Zero pad image border to preserve size spatially



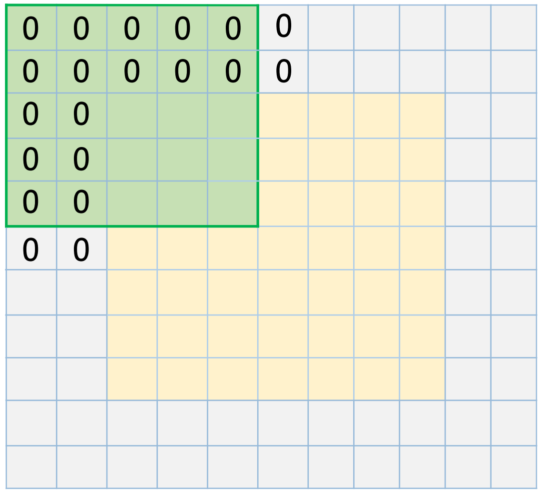
Input: 7 x 7

Filter: 3 x 3

Stride = 1

Zero Pad = 1

Output: 7 x 7



Input: 7 x 7

Filter:5 x 5

Stride = 1

Zero Pad = 2

Output: 7 x 7

Common Practice:

- Stride = 1
- Filter size: F x F
- Zero Padding: (F – 1)/2

Calculate Output Size

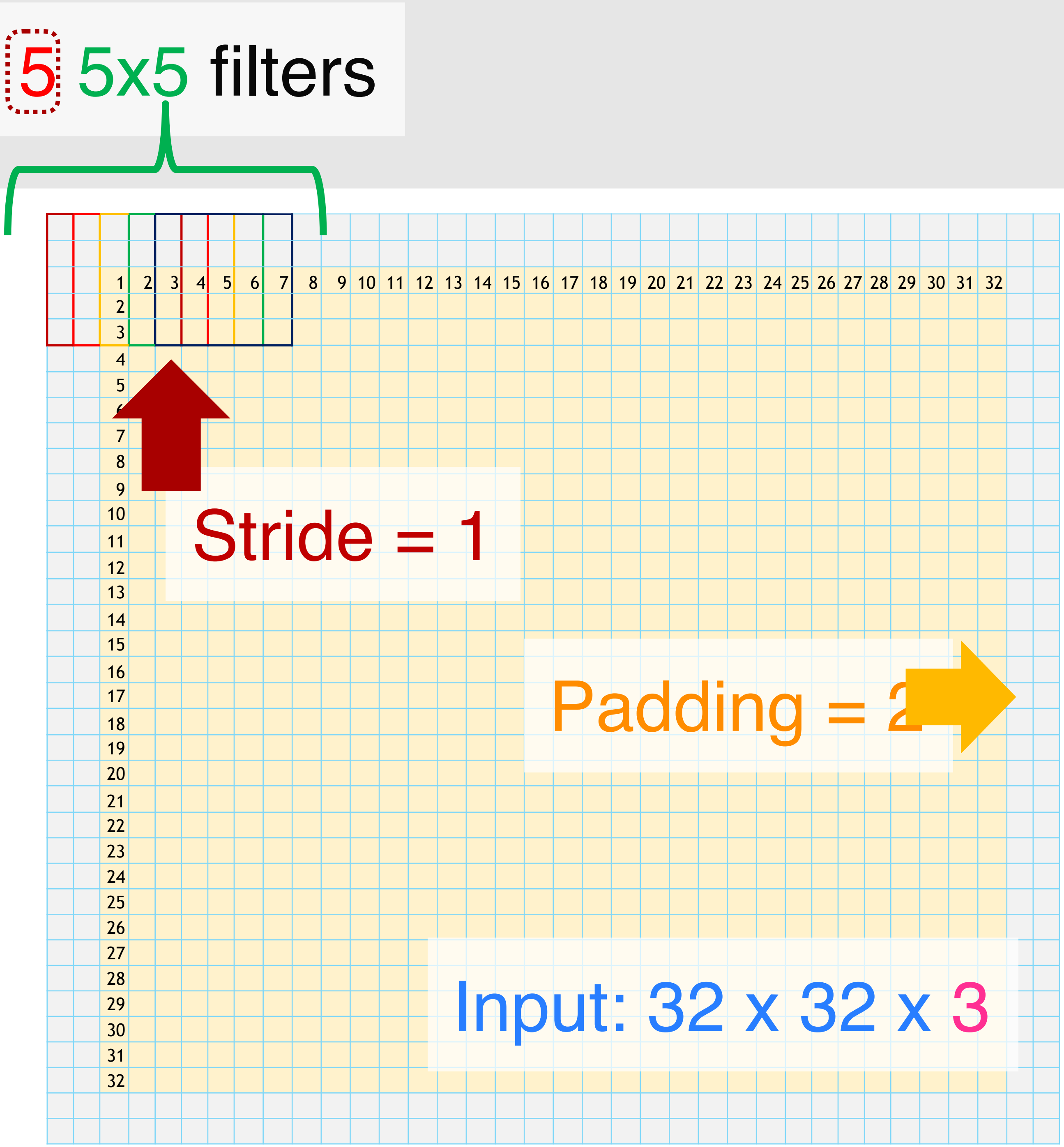
Output Volume Size:

$$\frac{(Input + 2)\ x\ Padding - Filter}{Stride + 1} =$$

$$\frac{(32 + 2)\ x\ 2 - 5}{1 + 1} = 32$$

$$32\ x\ 32\ x\ \boxed{5}$$

Source: http://cs231n.stanford.edu/slides/2017/cs231n_2017_lecture5.pdf



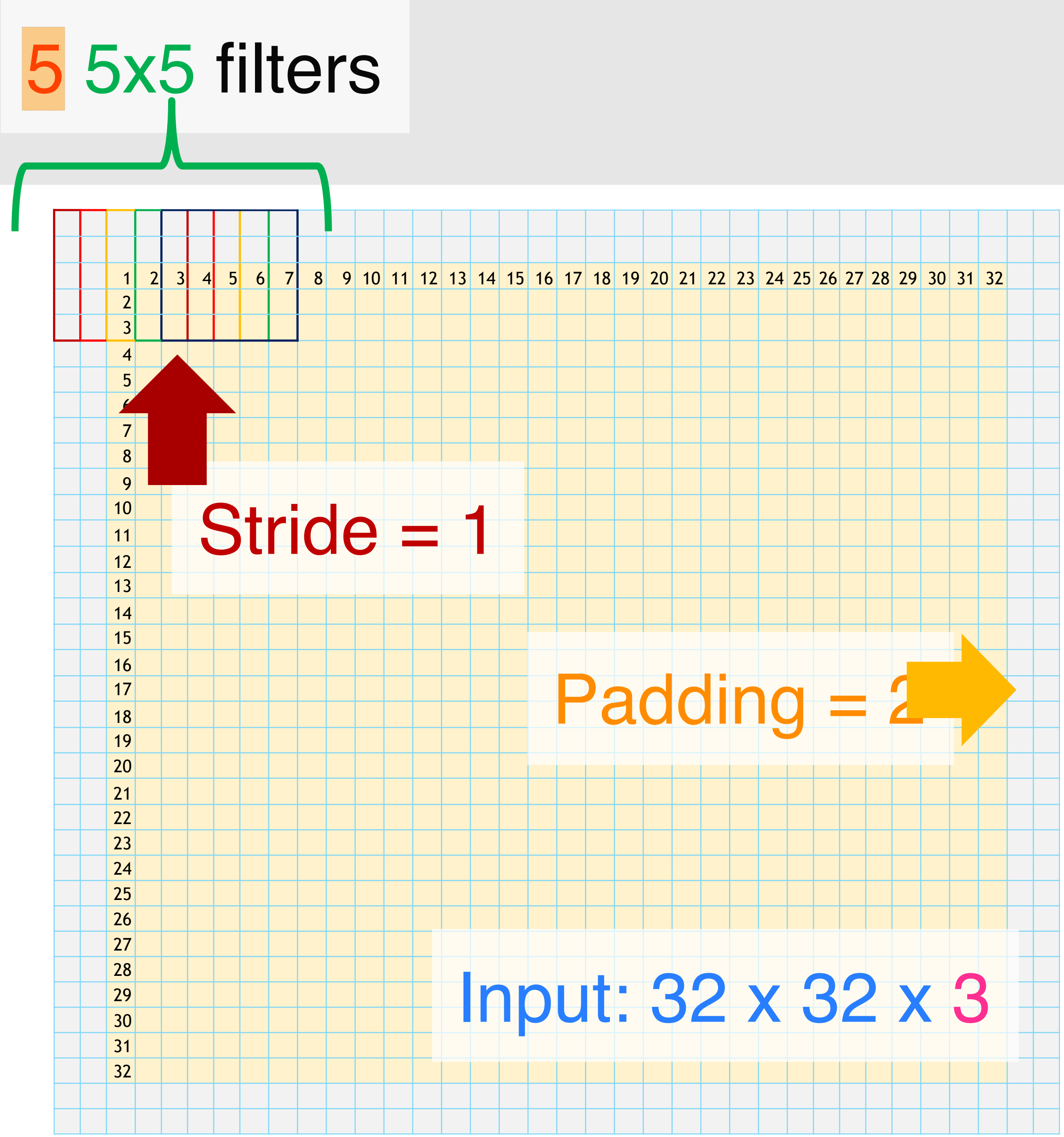
Calculate Output Size

Number of Parameters

$$(Filter \ x \ Filter \ x \ Depth + 1)(\# \ of \ Filters) =$$

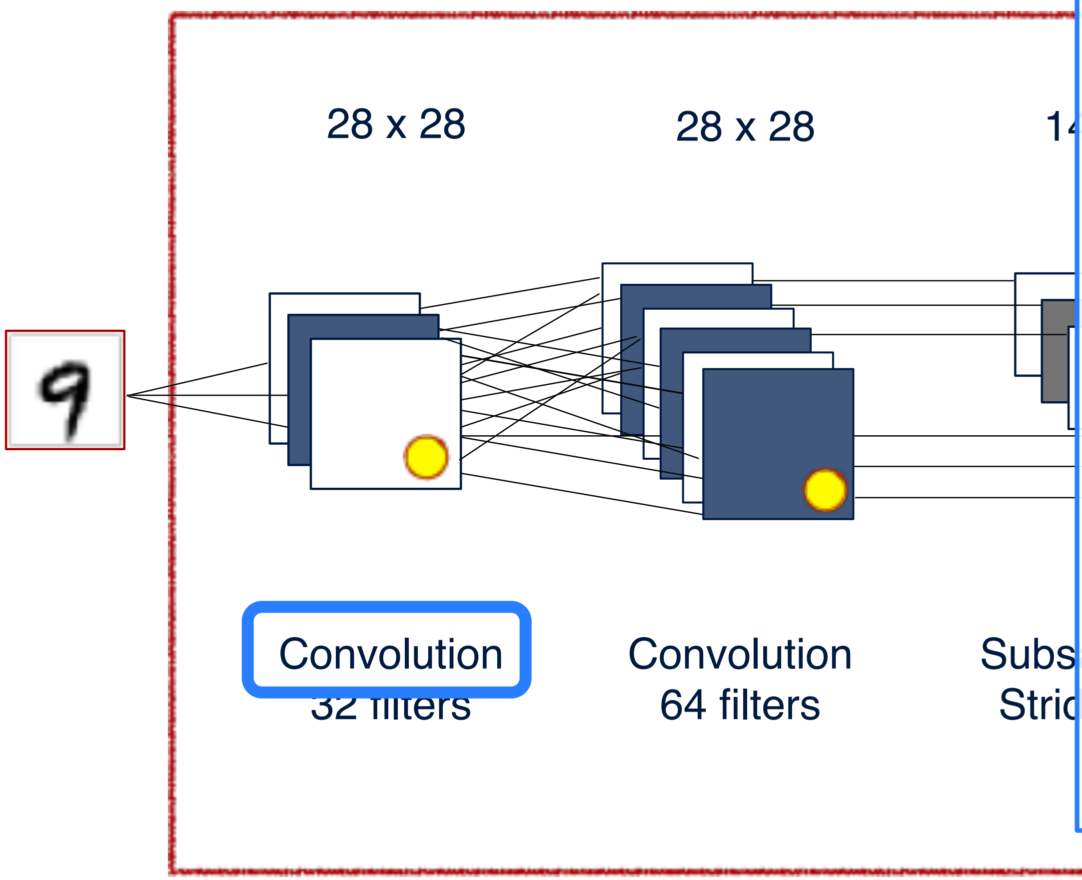
$$(5 \ x \ 5 \ x \ 3 + 1) \ x \ (5) =$$

380



Source: http://cs231n.stanford.edu/slides/2017/cs231n_2017_lecture5.pdf

Convolutional Neural Networks



Feature Extraction

Classification

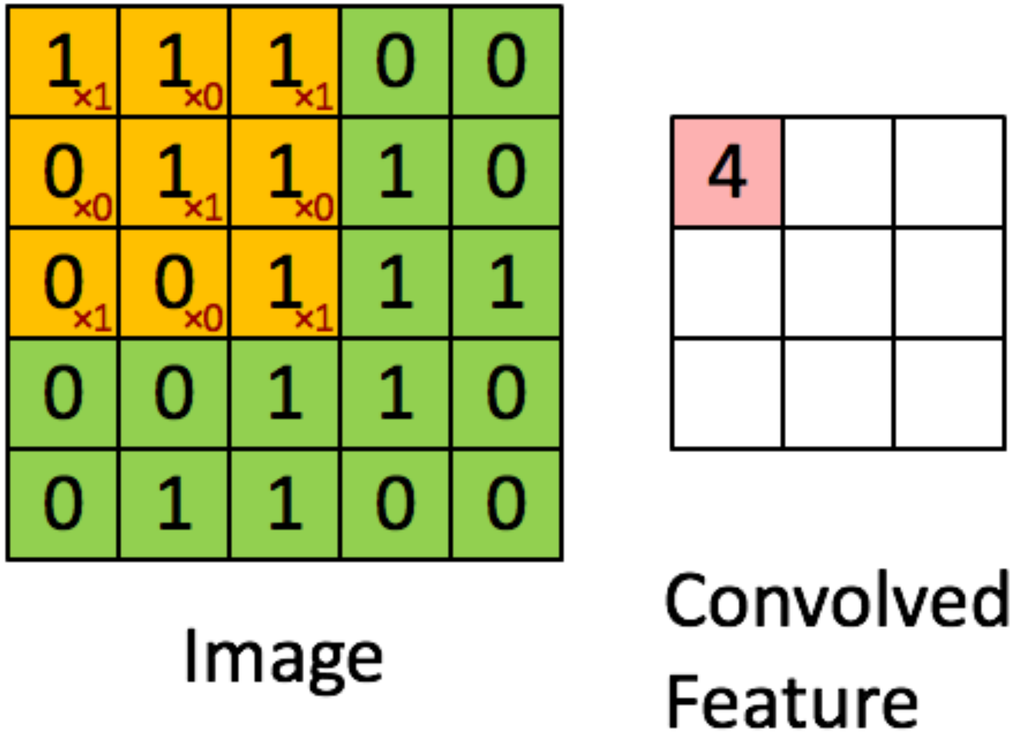
Activation Function (ReLU)

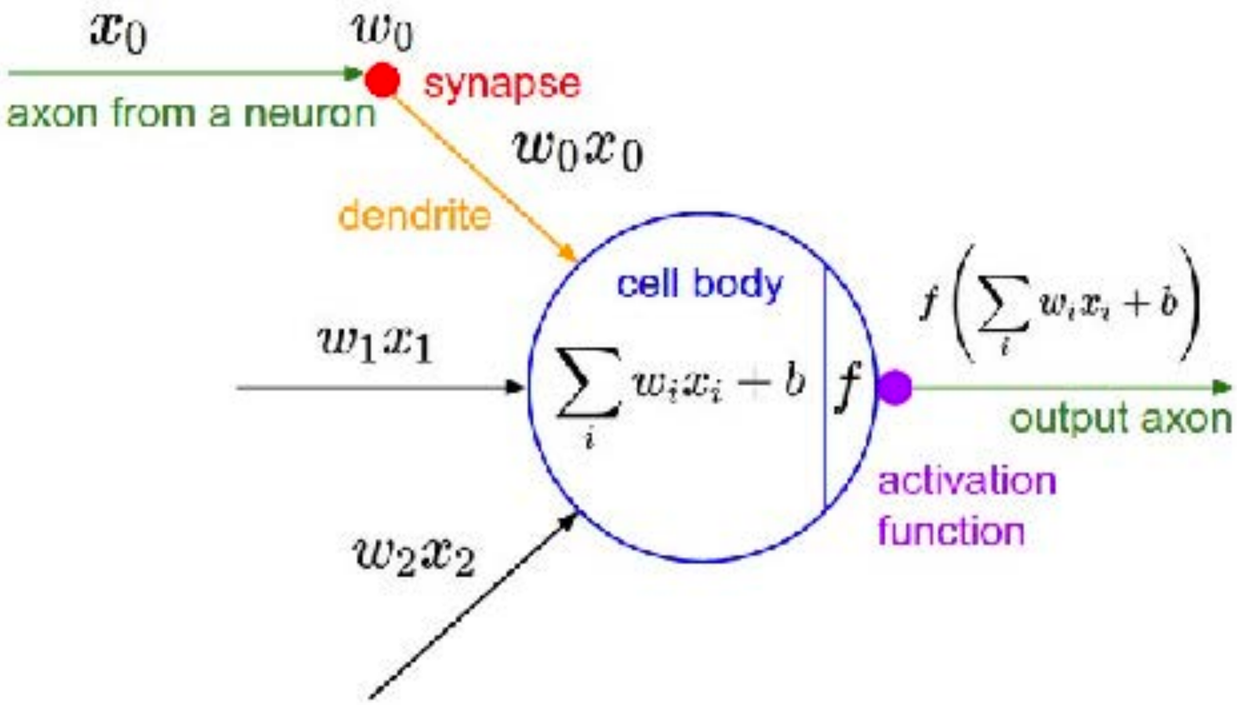
Compute output of neurons (dot product between their weights) connected to a small local region.

If we use 32 filters, then the output is 28x28x32 (using 5x5 filter)

ReLU Step (Local Connectivity)

The neurons still compute a dot product of their weights with the input followed by a non-linearity, but their connectivity is now restricted to be local spatially.

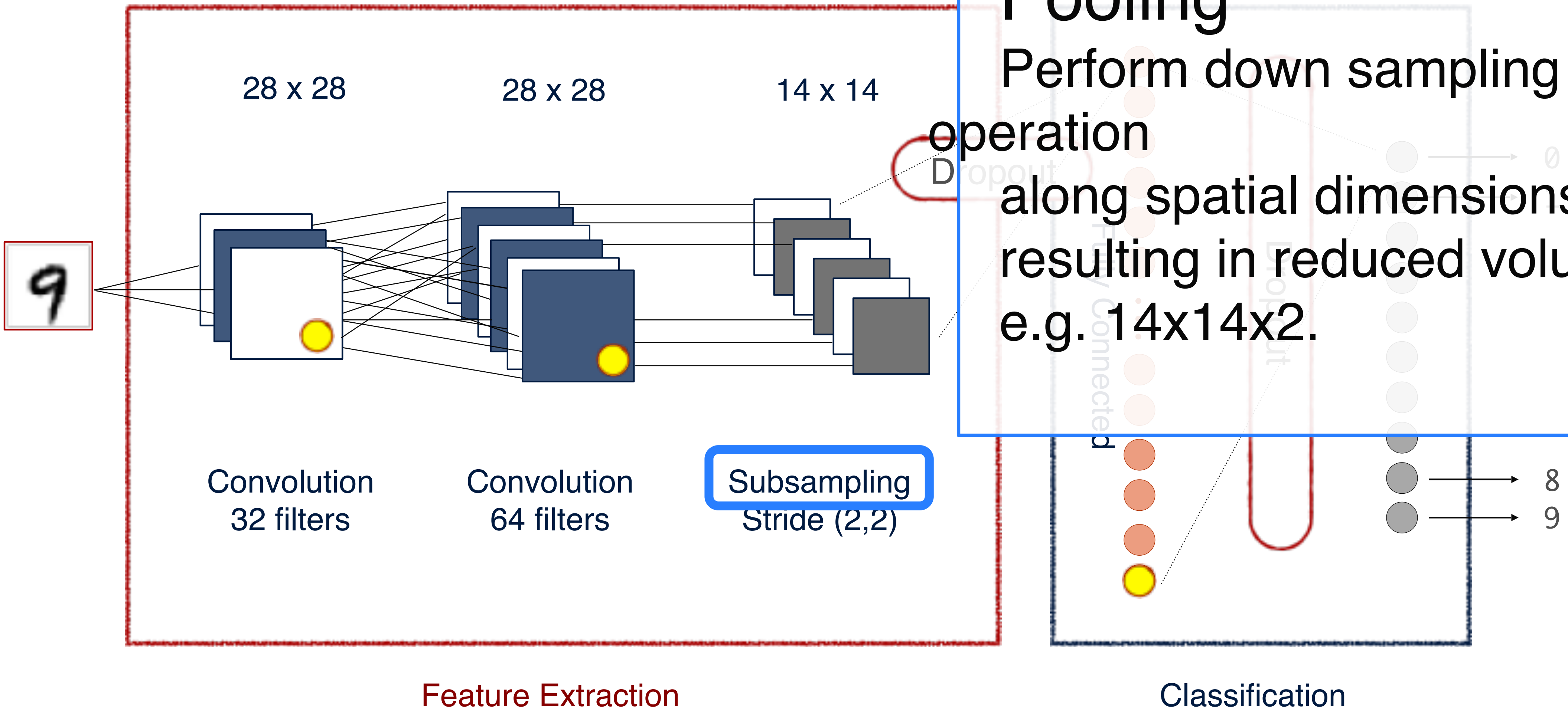




Source: http://deeplearning.stanford.edu/wiki/index.php/Feature_extraction_using_convolution

Source: https://cs231n.github.io/convolutional-networks/

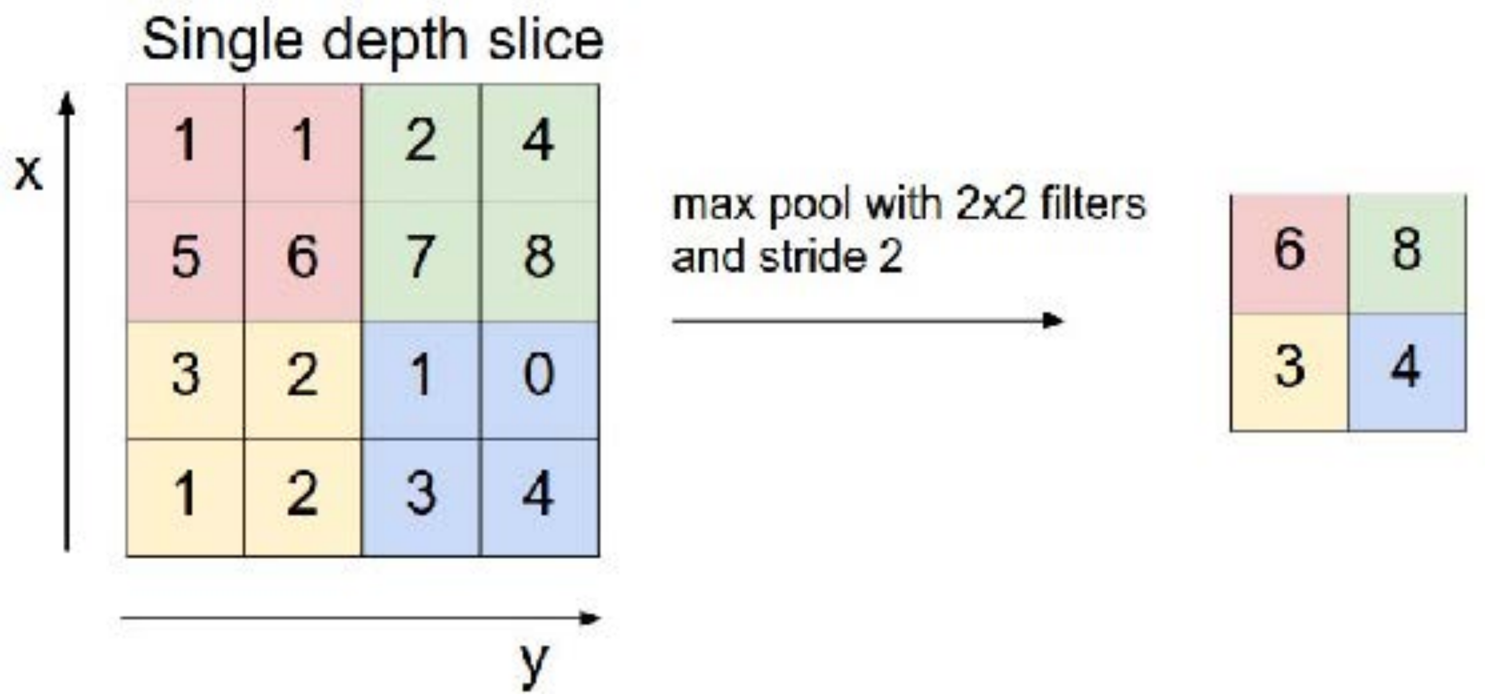
Convolutional Neural Networks



Pooling

Perform down sampling operation along spatial dimensions (w, h) resulting in reduced volume, e.g. 14x14x2.

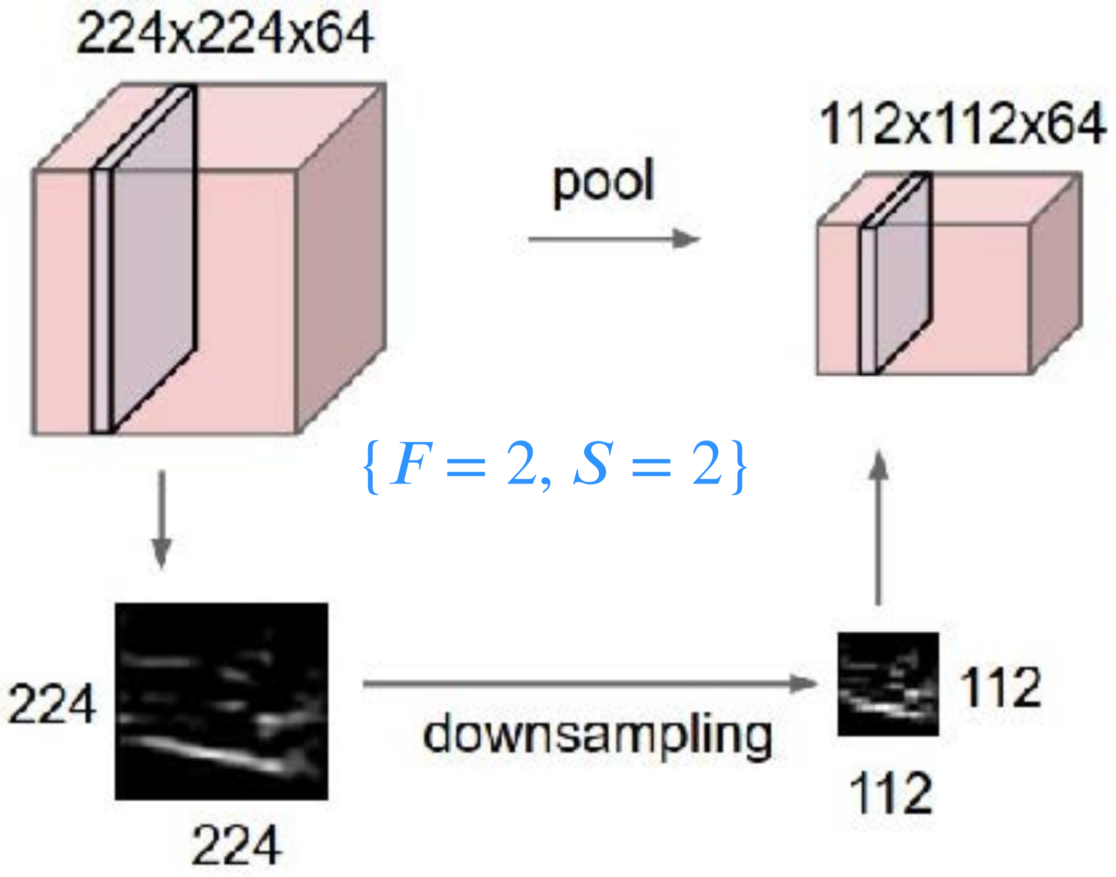
Pooling = subsampling = reduce image size



Source: https://cs231n.github.io/convolutional-networks/#pool

- Commonly insert pooling layers between successive convolution layers
- Reduces size spatially
- Reduces amount of parameters
- Minimizes likelihood of overfitting

Pooling common practices



$\{F = 2,\ S = 2\}$

Source: https://cs231n.github.io/convolutional-networks/#pool

- Input: $W_1\ x\ H_1\ x\ D_1$
- Output: $W_2\ x\ H_2\ x\ D_2\ where$

$$W_2 = \frac{(W_1 - F)}{S} + 1,\ D_2 = D_1$$

$$H_2 = \frac{(H_1 - F)}{S} + 1$$

- Typically $\{F = 2,\ S = 2\}$ or $\{F = 3,\ S = 2\}$ (overlap pooling).
- Pooling with larger receptive fields ($\{F\}$) too destructive.

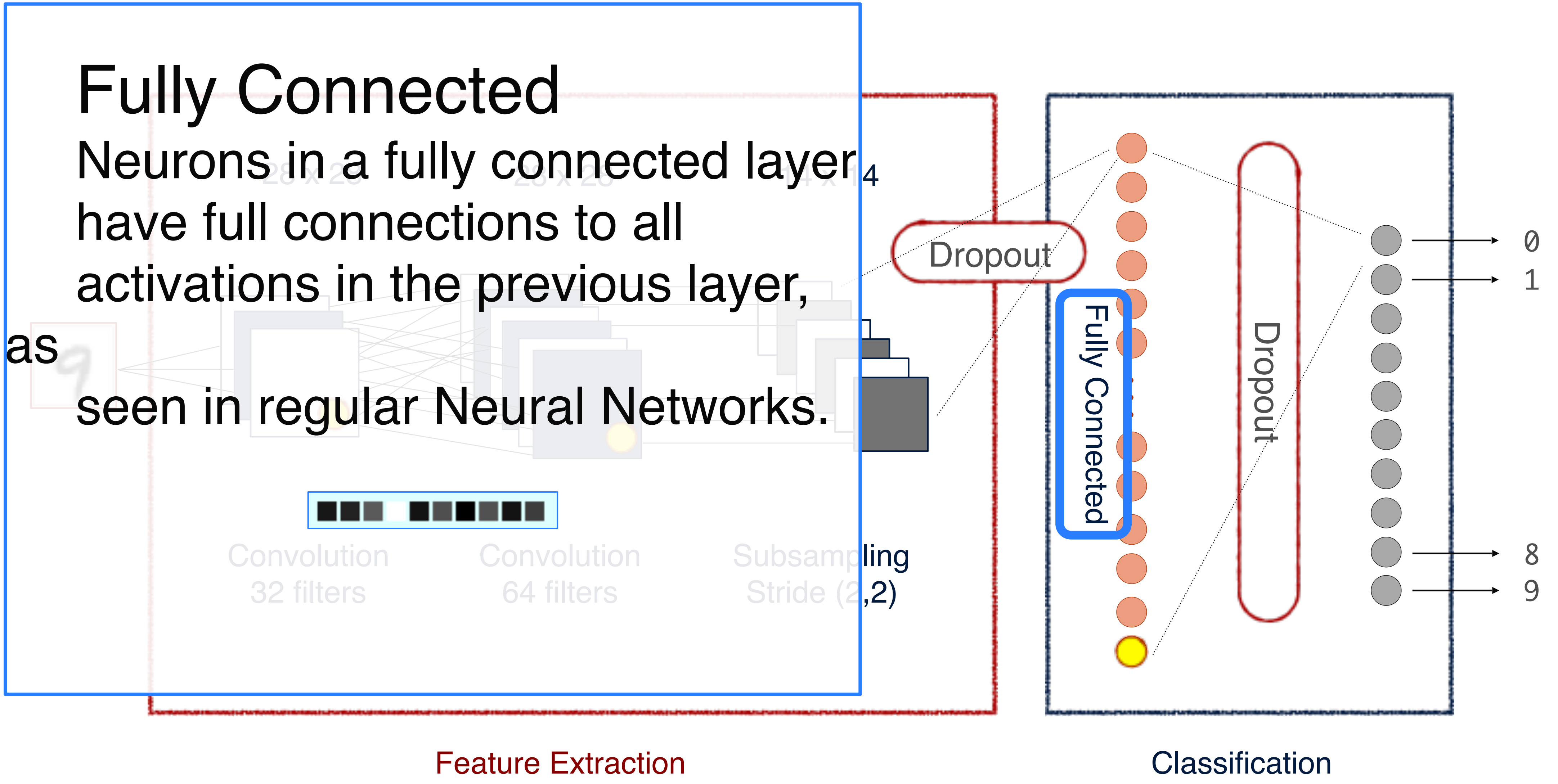
Stride, Pooling, Oh my!

- Smaller strides = Larger Output
- Larger strides = Smaller Output (less overlaps)
  - Less memory required (i.e. smaller volume)
  - Minimizes overfitting
- Potentially use larger strides in convolution layer to reduce number of pooling layers
  - Larger strides = smaller output reduce in spatial size ala pooling
  - ImageNet 2015 winner ResNet has only two pooling layers

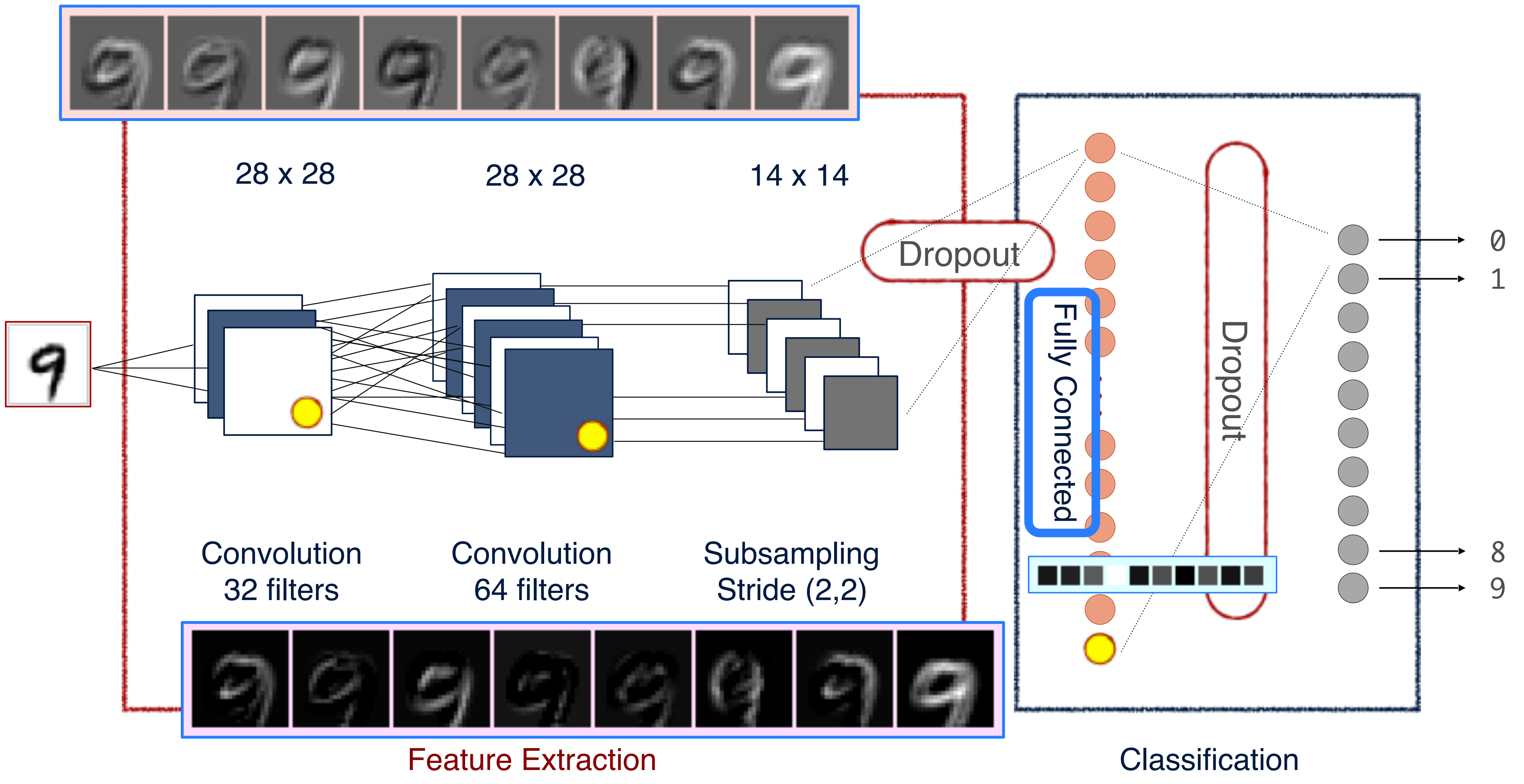
Convolutional Neural Networks

Fully Connected

Neurons in a fully connected layer have full connections to all activations in the previous layer, as seen in regular Neural Networks.



Convolutional Neural Networks Layers



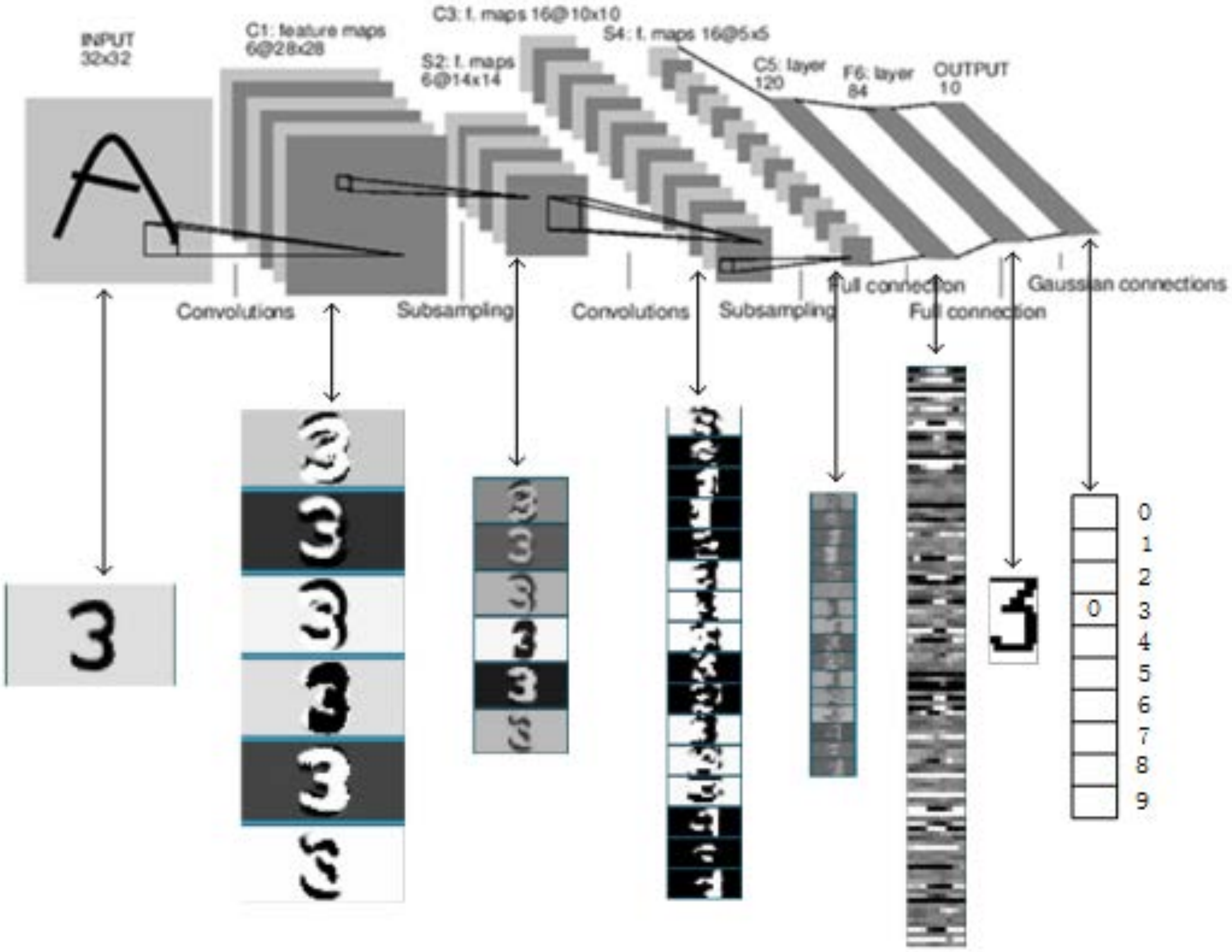
ConvNetJS MNIST Demo

https://cs.stanford.edu/people/karpathy/convnetjs/demo/mnist.html

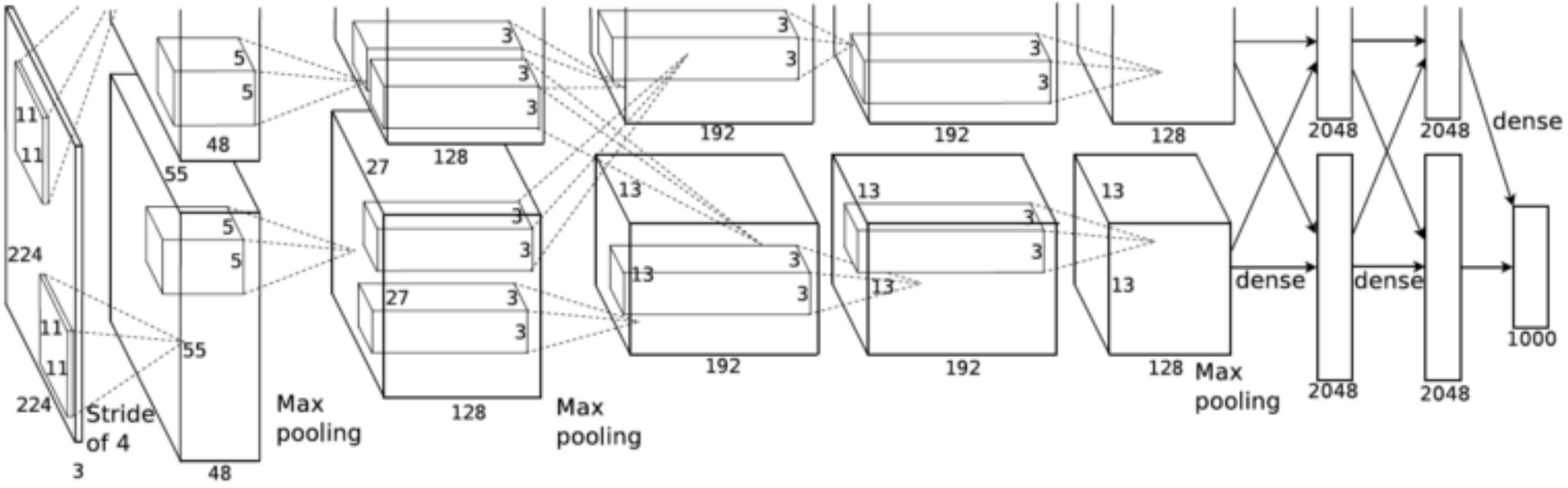
CNN Architectures

LeNet-5

- Introduced by Yann LeCun: http://yann.lecun.com/exdb/publis/pdf/lecun-01a.pdf
- Useful for recognizing single object images
- Used for handwritten digits recognition



AlexNet

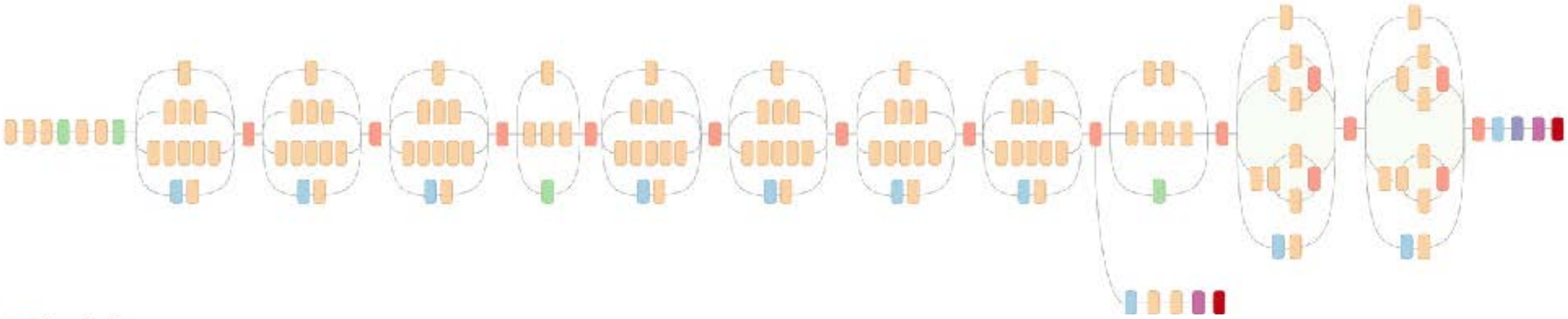


https://papers.nips.cc/paper/4824-imagenet-classification-with-deep-convolutional-neural-networks.pdf

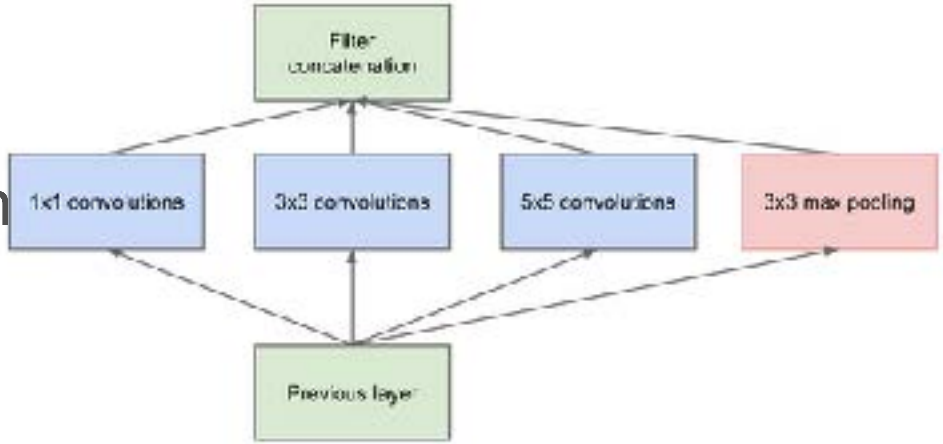
- Introduced by Krizhevsky, Sutskever and Hinton
- 1000-class object recognition
- 60 million parameters, 650k neurons, 1.1 billion computation units in a forward pass

Inception

https://arxiv.org/pdf/1409.4842.pdf

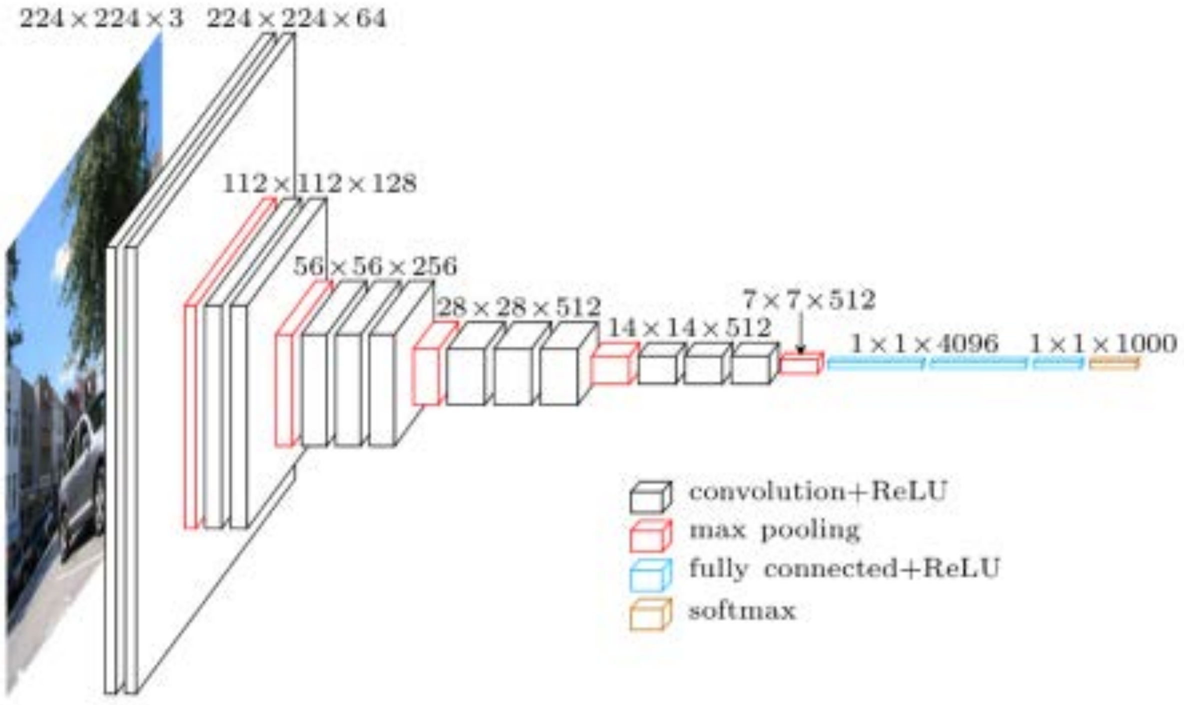


- Inception takes its name from the movie's title
- Many repetitive structures called inception modules
- 1000-class ILSVRC winner



VGG

- Introduced by Simnyan and Zisserman in 2014
- VGG after Visual Geometry Group at Oxford
- Conv3 layers stacked on top of each other
- The latest: VGG19 – 19 layers in the network



ResNet

- Introduced by He, Zhang, Ren, Sun from MS Research
- Use residual learning as a building block where the identity is propagated through the network along with detected features
- Won ILSVRC2015

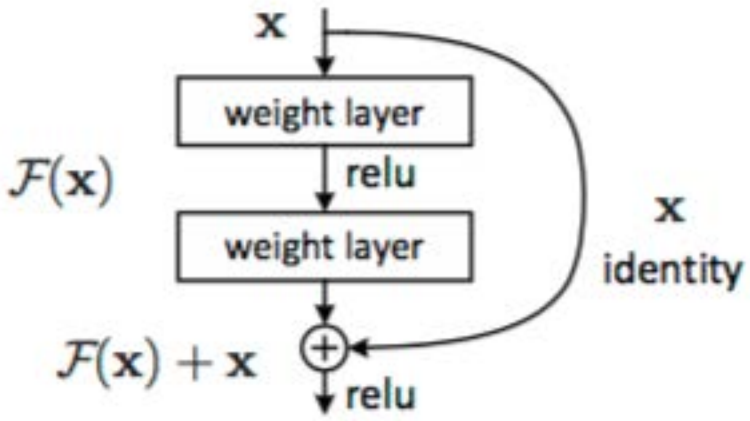
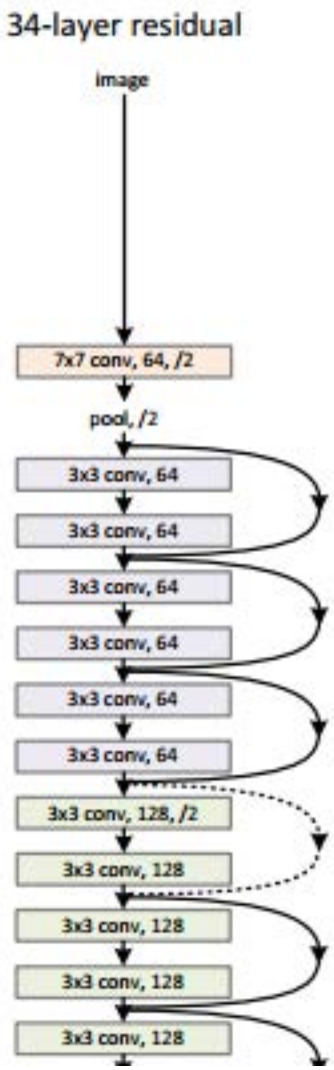


Figure 2. Residual learning: a building block.



DEMO

Neurons …

Activate!

I’d like to thank…

Great References

- Andrej Karparthy's ConvNetJS MNIST Demo
- What is back propagation in neural networks?
- CS231n: Convolutional Neural Networks for Visual Recognition
  - Syllabus and Slides | Course Notes | YouTube
  - With particular focus on CS231n: Lecture 7: Convolution Neural Networks
- Neural Networks and Deep Learning
- TensorFlow

Great References

- Deep Visualization Toolbox
- Back Propagation with TensorFlow
- TensorFrames: Google TensorFlow with Apache Spark
- Integrating deep learning libraries with Apache Spark
- Build, Scale, and Deploy Deep Learning Pipelines with Ease

Attribution

Tomek Drabas

Brooke Wenig

Timothee Hunter

Cyrielle Simeone

Q&A