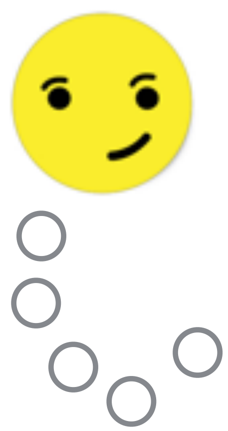
6.867 Fall 2017
Neural Networks

Training NNs, BN, CNN, Resnet

Lecture 11: 17th Oct, 2017

Admin



Exam: **10/19, 7:30pm-9:00pm**

Locations: 54-100, 26-100

(a thru j…@mit.edu, k thru z…@mit.edu)

No class on 10/19

Recitations, exercises as usual

Additional exam day details: sent yest in the stellar announcement

Useful links

User friendly intro

http://neuralnetworksanddeeplearning.com

CNN centric class, many useful tips

http://cs231n.github.io

http://cs231n.stanford.edu

Resnets

http://icml.cc/2016/tutorials/icml2016_tutorial_deep_residual_networks_kaiminghe.pdf

More broadly

(At the least) have a look at tutorials at ICML, NIPS, and CVPR

Outline

* Basic comments on training
* Regularization via Dropout
* Batch Normalization
* Convolutional Neural Nets
* Intro to Resnets

Initialization

Properly initializing a NN is very important.
Reasons: NN loss is highly nonconvex; optimizing it to attain a "good" solution is difficult, requires careful tuning. Training speed can also vary.

Example: When using ReLUs if we initialize the network weights to zero, all the gradients and activations will be zero, and will not change even after seeing training data. (**Explore**: Think & try ways of countering this)

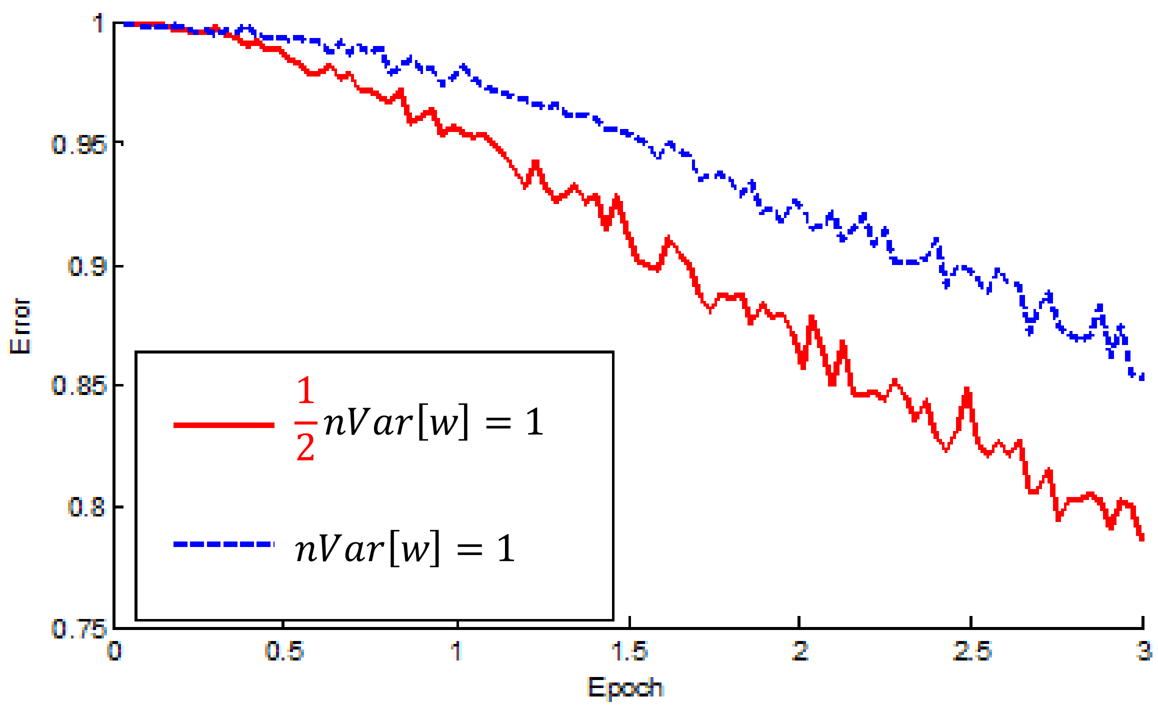
Random: Initialize randomly, e.g., via the Gaussian $N(0, \sigma^2)$, where std σ depends on the number of neurons in a given layer

Why? roughly ensures that random input to a unit does *not depend* on the number of inputs it gets). For ReLUs current recommendation: use $\sigma^2=2/n$

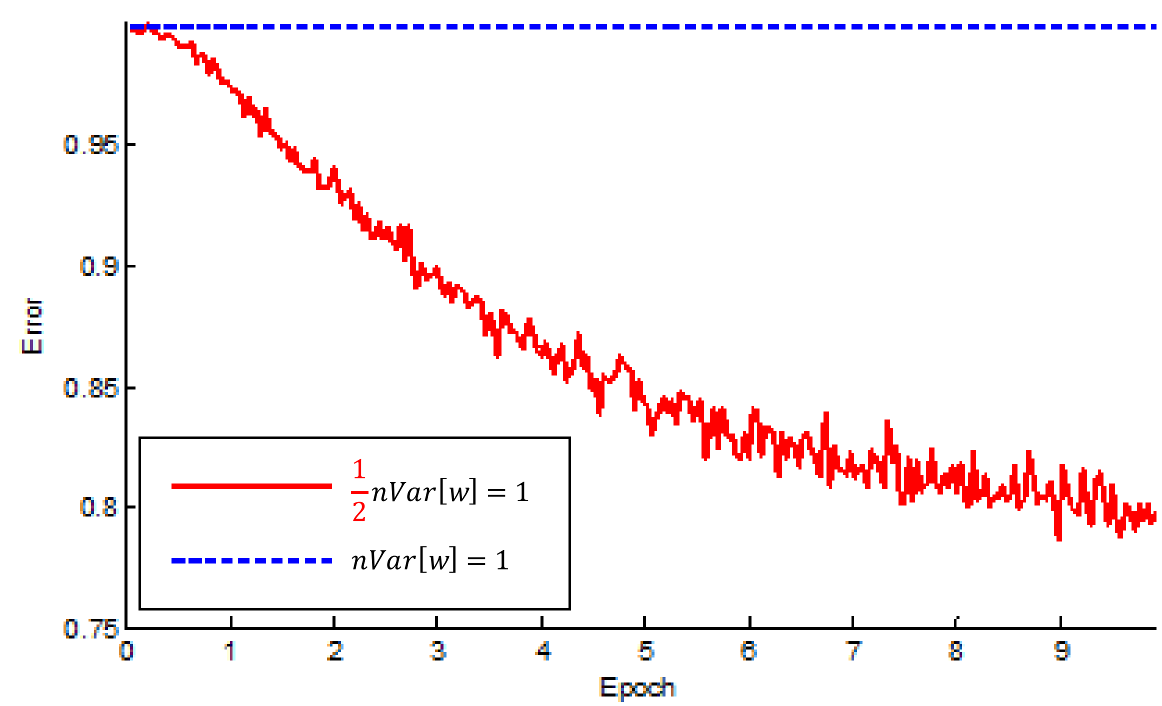
See also: http://cs231n.github.io/neural-networks-2/ for additional practical notes

Impact of initialization

22-layer ReLU net:
good init converges faster



30-layer ReLU net:
good init is able to converge



*Figures show the beginning of training

Kaiming He, Xiangyu Zhang, Shaoqing Ren, & Jian Sun. "Delving Deep into Rectifiers: Surpassing Human-Level Performance on ImageNet Classification". ICCV 2015.

Ultimately, coming up with good initializations is hard; we'll see a technique today that reduces dependence on initialization somewhat, but of course, cannot eliminate.

Initialization

Exercise: When working with image data, can we initialize all parameters to the same initial value (e.g., mean image intensity)

Explore:

Smart-ReLUs: afaik, all NN toolkits use zeros as subgradients at the point of non-differentiability of a ReLU. Explore picking a random subgradient at that point when doing backprop.

Exercise: Argue that Smart-ReLUs fix the "dying units" problem, as well as avoids stalling nets with zero-initialization.

Unstable gradients

$$\delta^l = \frac{\partial \ell}{\partial z^l} = \mathrm{Diag}[f'(z^l)]W^{l+1}\delta^{l+1}.$$

$$\delta^l = \mathrm{Diag}[f'(z^l)]W^{l+1}\mathrm{Diag}[f'(z^{l+1})]W^{l+2}\cdots W^L\delta^L$$

Observations

- Multiplication of a chain of matrices in backprop
- If several of these matrices are "small" (i.e., norms < 1), when we multiply them, the gradient will decrease exponentially fast and tend to *vanish* (hurting learning in lower layers much more)
- Conversely, if several matrices have large norm, the gradient will tend to *explode*. In both cases, the gradients are unstable.
- Coping with unstable gradients poses several challenges, and must be dealt with to achieve good results.

Unstable gradients

Partial remedies

- ReLU (ameliorates, does not suffer from saturation)
- Memory (in RNNS)
- Reparameterization, e.g., via orthogonal matrices
- Weight normalization and batch normalization (somewhat)
- Gradient clipping (several choices; $g \leftarrow c\frac{g}{\|g\|}$)
- Numerous other ideas (architecture specific)
- Residual Networks

Regularization

$$+ \lambda \|\theta\|^2$$

definitely use it; but let us look at another way

Overfitting in DNNs

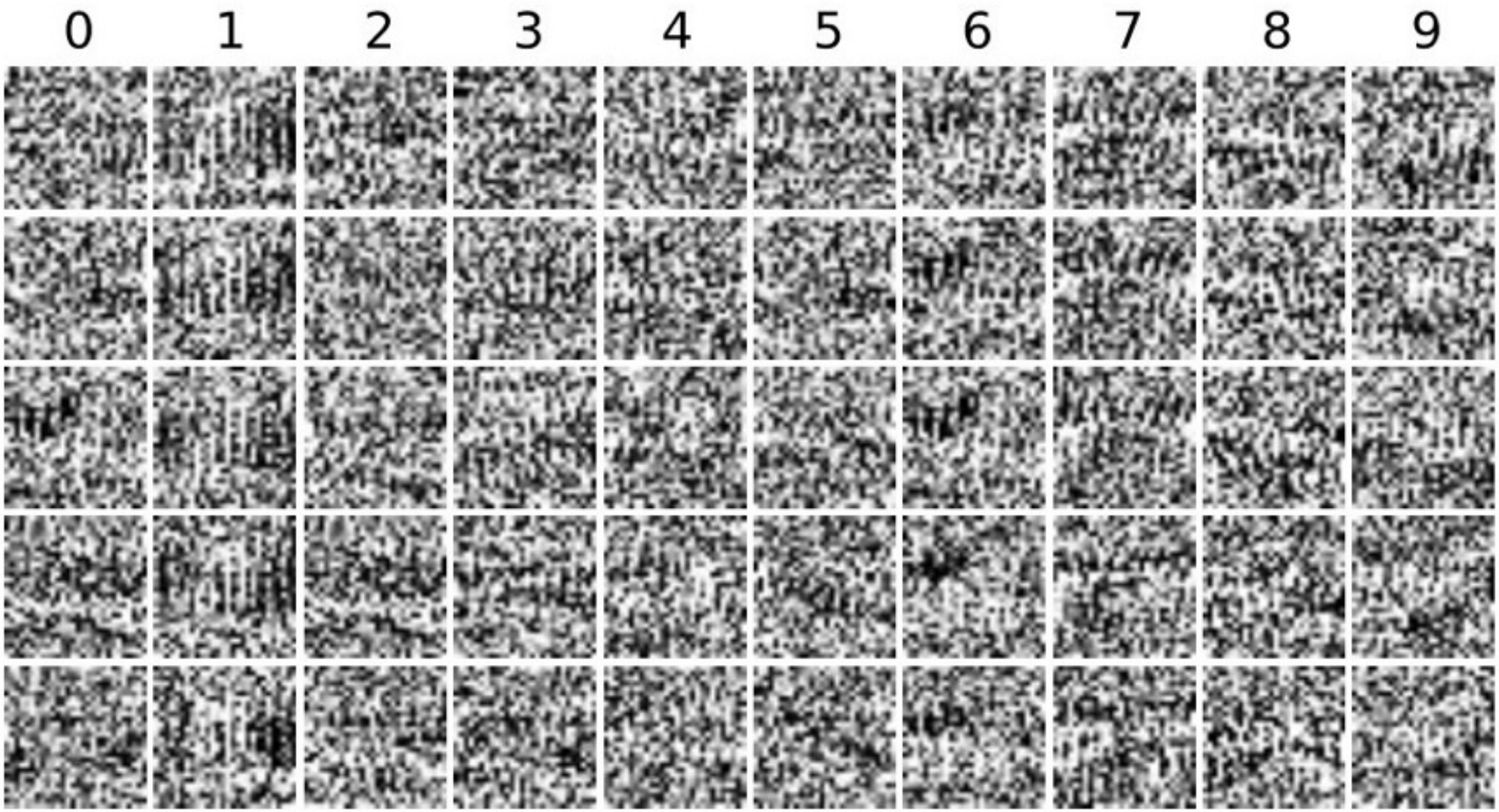


Figure 4. Directly encoded, thus irregular, images that MNIST DNNs believe with 99.99% confidence are digits 0-9. Each col-

[Nguyen, Yosinksi, Clune (CVPR 2015)]

Overfitting in DNNs

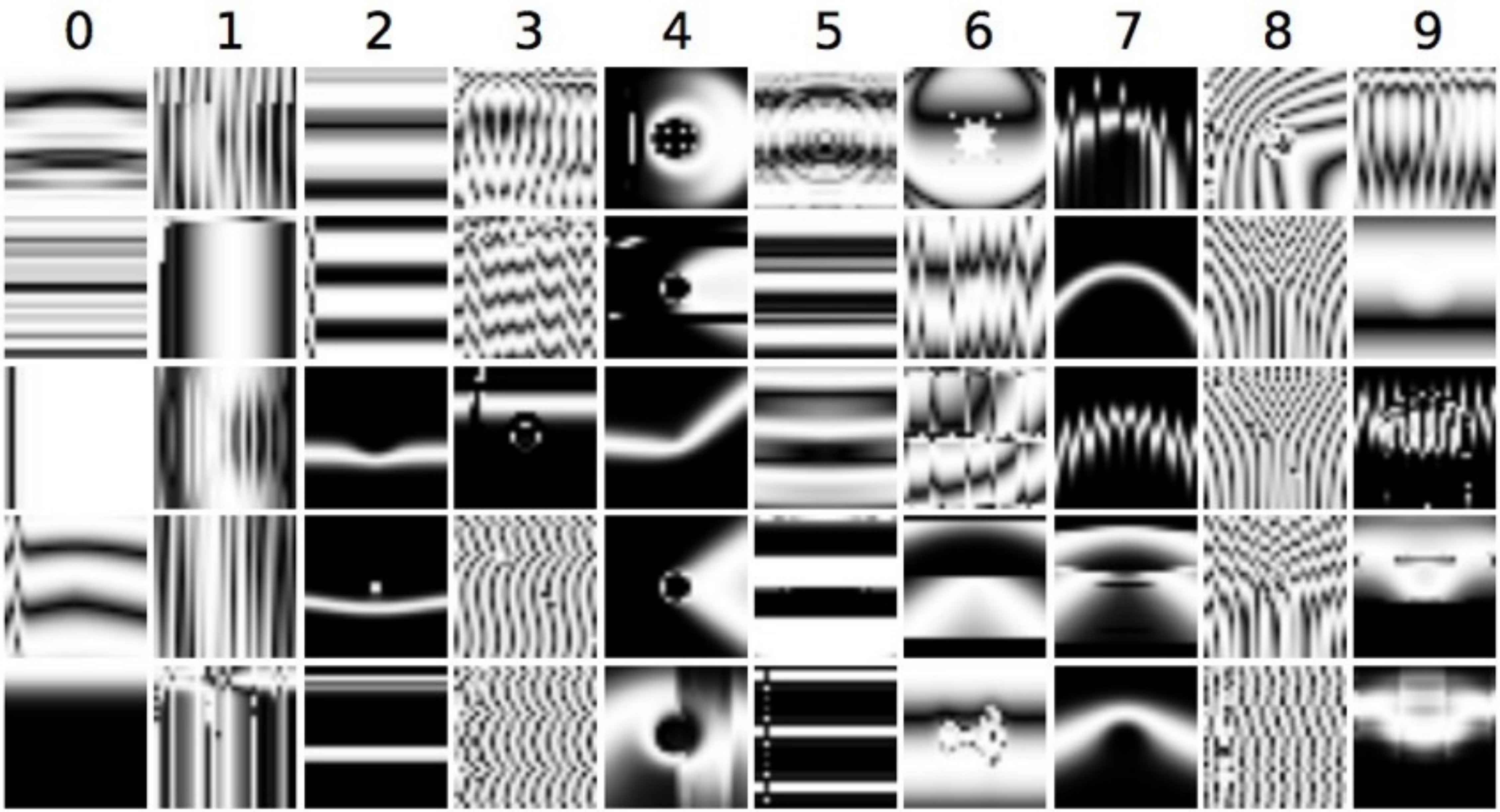


Figure 5. Indirectly encoded, thus regular, images that MNIST DNNs believe with 99.99% confidence are digits 0-9. The column

[Nguyen, Yosinksi, Clune (CVPR 2015)]

Regularizing with Dropout

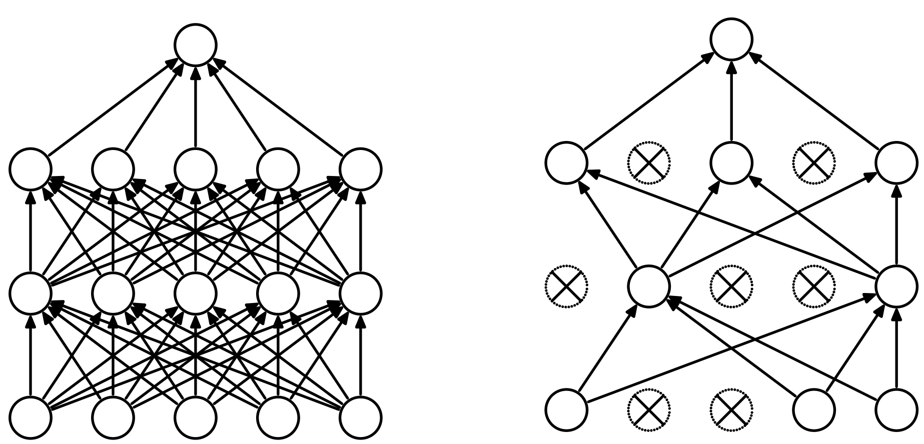
Observation

- When fitting to the nitty-gritty of the input, including noise hidden units must rely on each other to co-adapt and have complementary coverage of the data space.
- To hinder fitting to noise we must avoid overdoing co-adaptation

Dropout

- Randomly turn off units, say with probability 1/2, when training!
  - For each data point, we **randomly** set the output of each hidden unit to zero.
  - The neurons turned off are randomly chosen anew for each training data point
  - Accounted for during backprop (**how?**).
  - For units turned off for that round, input weights and activations **not** updated; unit effectively dropped out for that particular training sample. The units can thus rely on signals from their neighbors only if a large number of them support these. This dropout strategy therefore provides a means to regularizing.

figure from the [dropout] paper



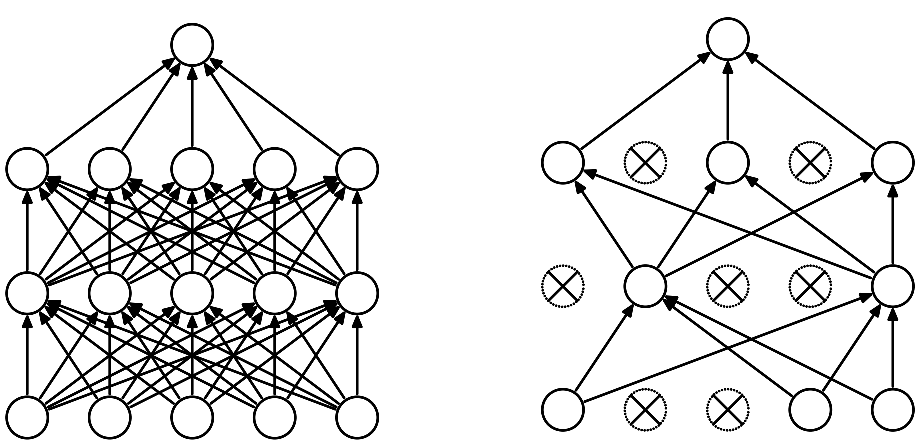
(a) Standard Neural Net (b) After applying dropout.

Regularizing with Dropout

Dropout at test time

- At test time simulate the impact of dropout as follows: since each neuron was on approximately 1/2 the time during training, its contribution to the units in subsequent layers is just 1/2 of what it would be
- Thus, we simply multiply outgoing weights of units by 1/2 during test time.
- There exist more detailed explanations of dropout, as well as theoretical interpretations (e.g., ensemble averaging over a large number of network configurations, hence less prone to overfitting). We omit those from our discussion; check wider literature if interested.

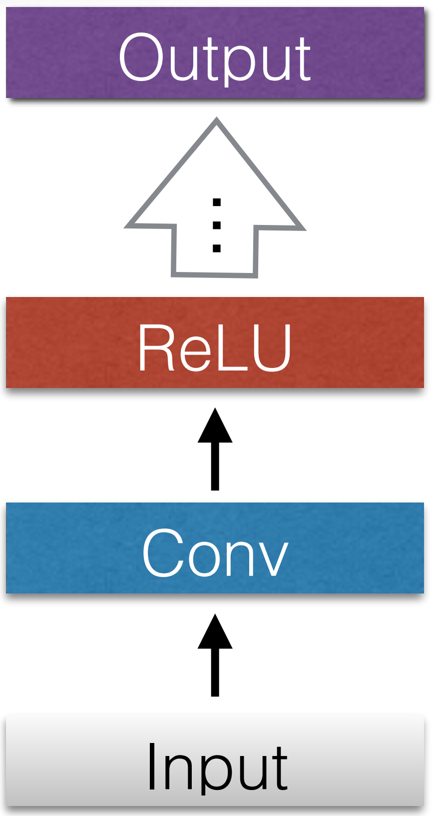
figure from the [dropout] paper



(a) Standard Neural Net

(b) After applying dropout.

Batch Normalization



Loss occurs at last layer

Last layers learn more quickly

Input flows in at first layer

Any change to lower layer impacts rest of network

Subsequent layers need to "relearn" many times

Convergence speed takes a hit

Internal covariate shift

Internal Covariate Shift: change in distribution of network activations due to change in network parameters during training

Aim: improve training speedy by reducing internal covariate shift

Idea: Known that training converges faster if inputs "whitened", i.e., linearly transformed to have mean zero, unit variance, and decorrelated.

Batch Normalization

Observation: Known that training converges faster if inputs "whitened", i.e., linearly transformed to have mean zero, unit variance, and decorrelated.

Idea 0: Activations of one layer, inputs to another. If we do similar whitening of the inputs of each layer might help towards improving training.
(BN: view it as "differentiable" preprocessing of a layer's inputs)

Full whitening involves inverting large matrices, a no-go

Idea 1: Normalize features individually, not jointly

$$x = (x^1, \ldots, x^p)$$

(features at a layer)

$$\hat{x}^k = \frac{x^k - \mathbb{E}[x^k]}{\sqrt{\mathrm{Var}[x^k]}}$$

Expectation and Variance computed over training data set (LeCun98— this speeds up training)

Batch Normalization

Idea 1: Normalize features individually, not jointly

$$x = (x^1, \ldots, x^p)$$

(features at a layer)

$$\hat{x}^k = \frac{x^k - \mathbb{E}[x^k]}{\sqrt{\mathrm{Var}[x^k]}}$$

Expectation and Variance computed over training data set (LeCun98— this speeds up training)

Idea 1: mini-batch normalization

BN transform applied to activation *x* over a mini-batch

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

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

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

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

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

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

figure: [Ioffe, Szegedy, 2015]

Batch Normalization

Idea 2: Restore representation power" / Undo damage by learning γ and β

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

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

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

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

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

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

Intuition: Allow the transformation to represent the identity (this idea recurs)

Exercise: Derive backprop rules to figure out how to update scale γ and shift β

figure: [Ioffe, Szegedy, 2015]

Batch Normalization

- ✓ BN layer can be added to many networks (e.g., CNNs, Resnets, etc.)
  - ➡ *Current Challenge*: BN for RNNs
- ✓ BN enables higher learning rates: backprop through a BN layer is unaffected by the scale of its parameters, BN(Wx)=BN((aW)x)
- ✓ BN has a regularizing effect (Dropout can even be dropped out)

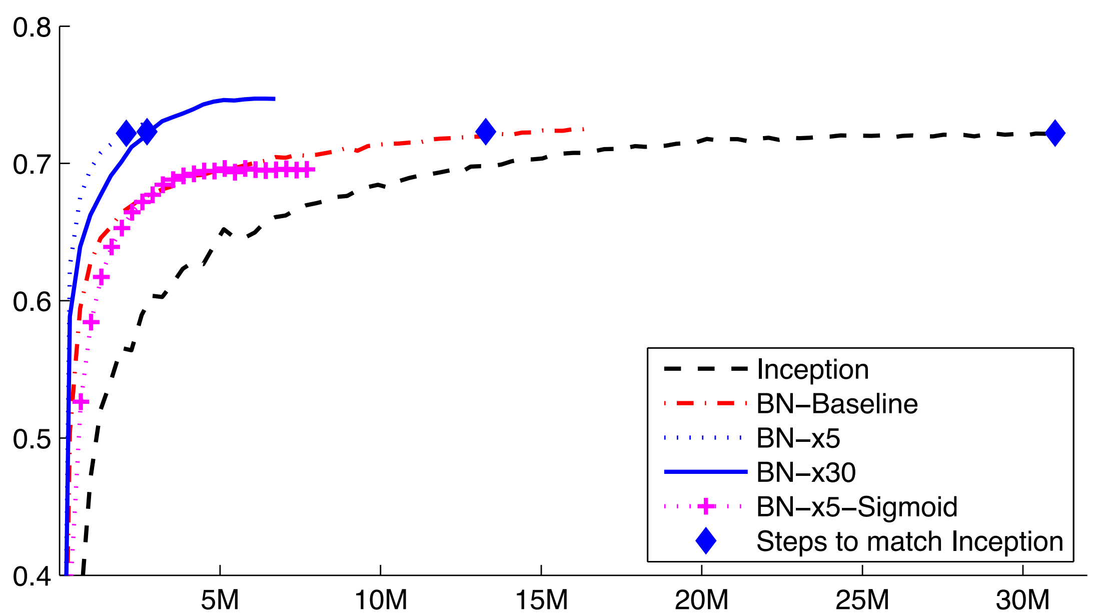


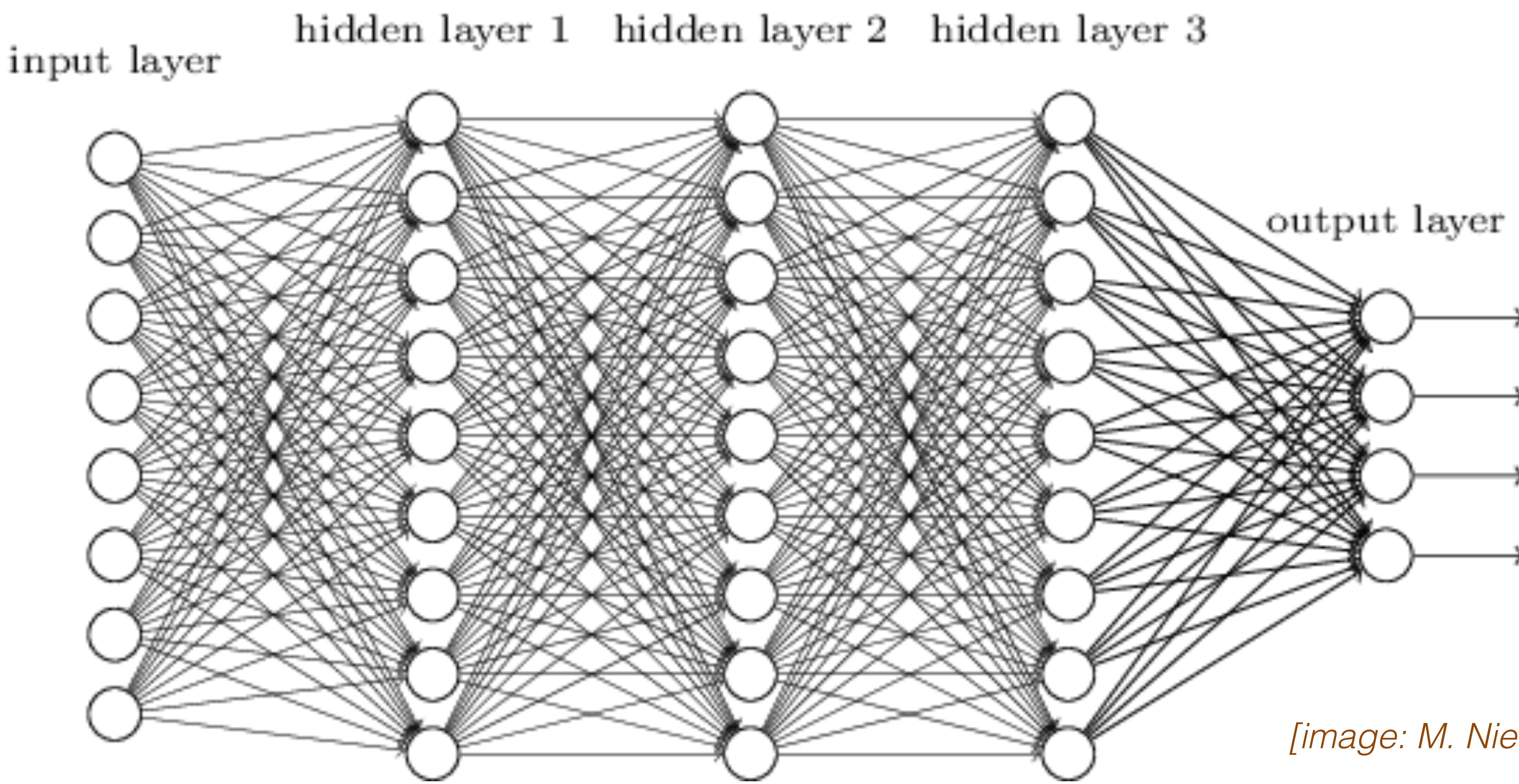
Figure 2: *Single crop validation accuracy of Inception and its batch-normalized variants, vs. the number of training steps.*

(several other speedups enabled, and used for this plot)

figure: [Ioffe, Szegedy, 2015]

Convolutional Neural Nets

Feed-forward neural net

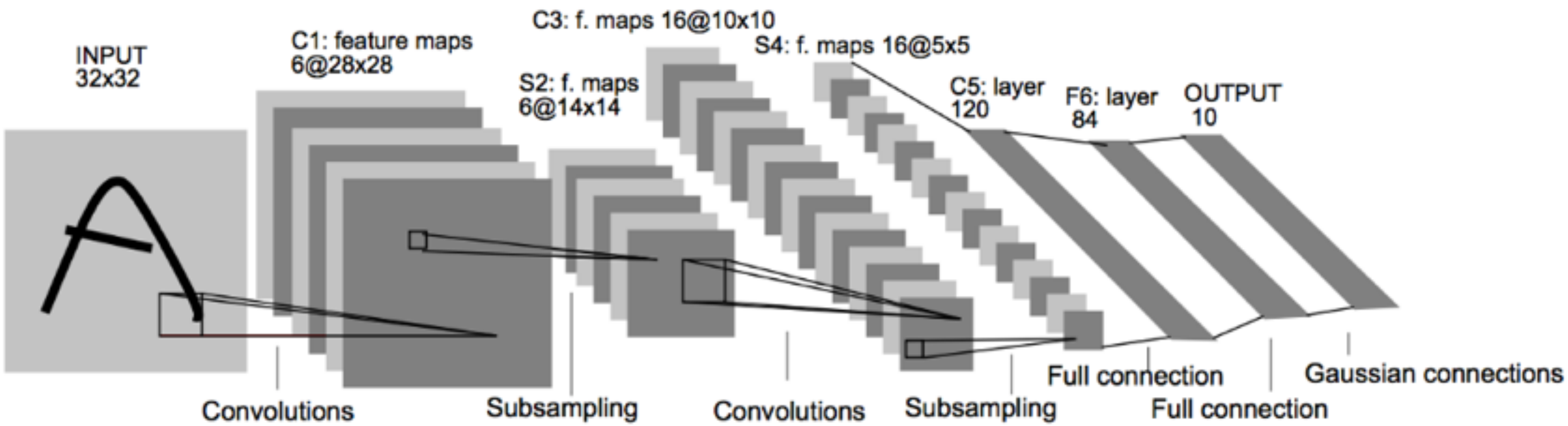


[image: M. Nielsen]

A memory hog!

Exercise: If input is 1000 dimensional, and we have a 2 FC layers with 4096 units each, how many hidden params do we have?

Original CNN



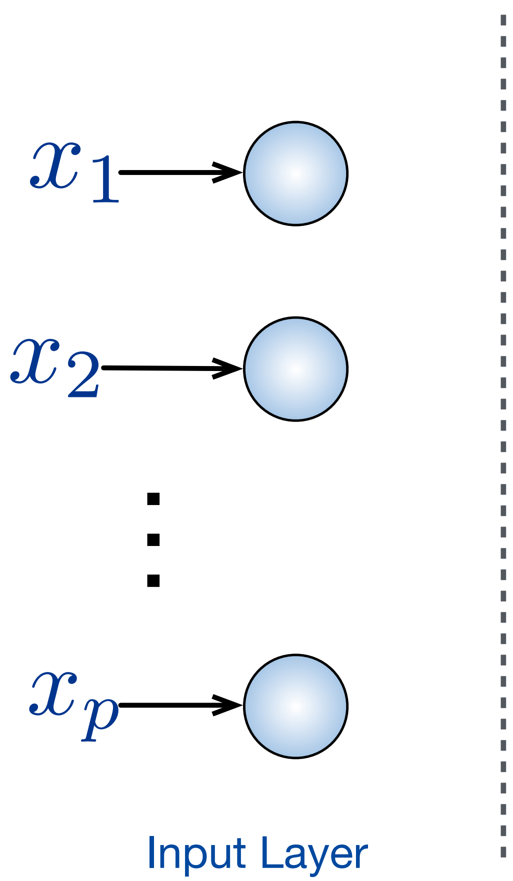
CNN called LeNet by Yann LeCun (1998)

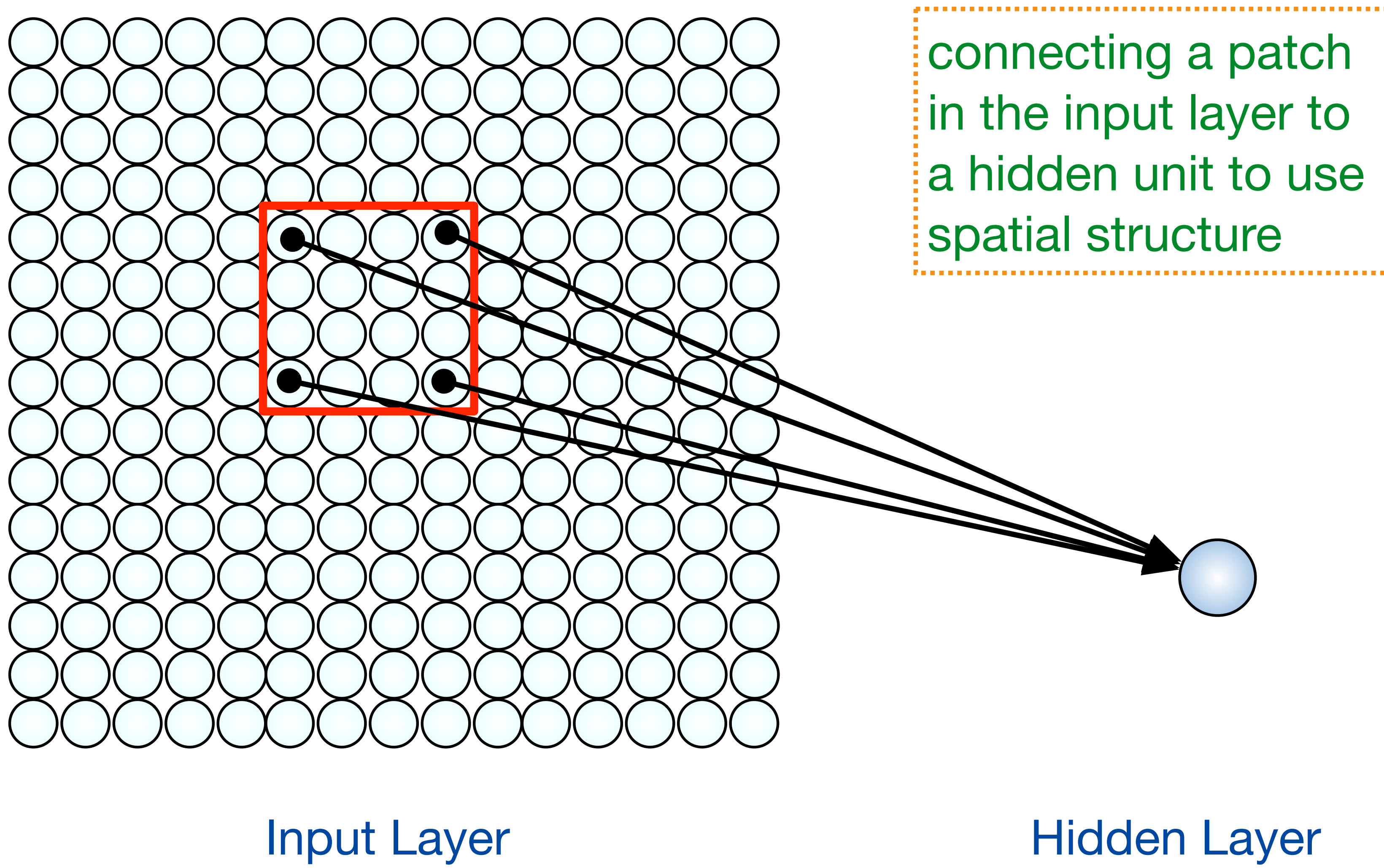
Using structure in the input

FC network: input a vector and each coordinate connected to every unit in FC layer

Nearby features may have additional local structure, but otherwise roughly independent

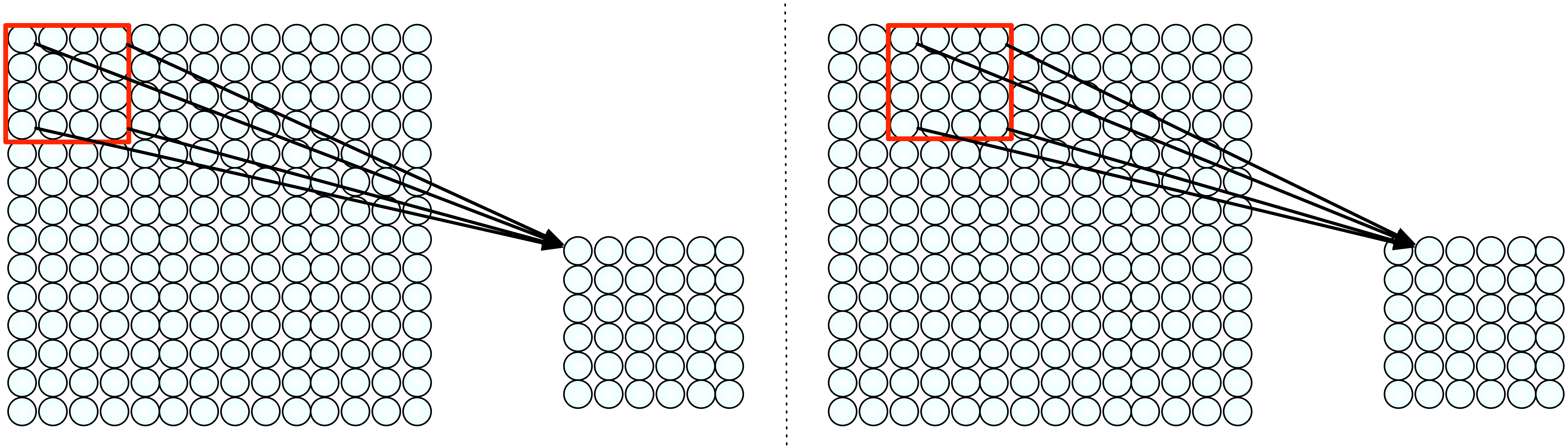
Example: Imagine a 2D input image. Treated as a vector we ignore its spatial structure





Moving window of filters

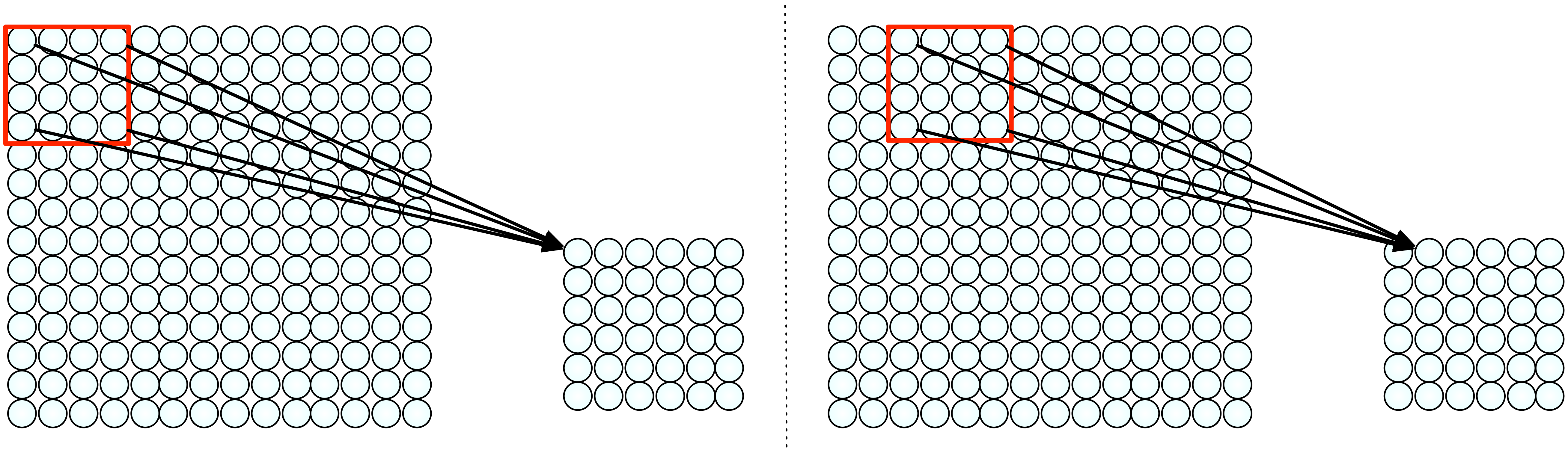
We take different patches in input image, connect them to units in hidden layer. Common approach is to use a sliding window



We are using 4x4 patches here with a *stride* of 2
(i.e., we shift by 2 pixels for next patch)

Explore: Connect patches in a different style. Discuss pros and cons

Moving window of filters



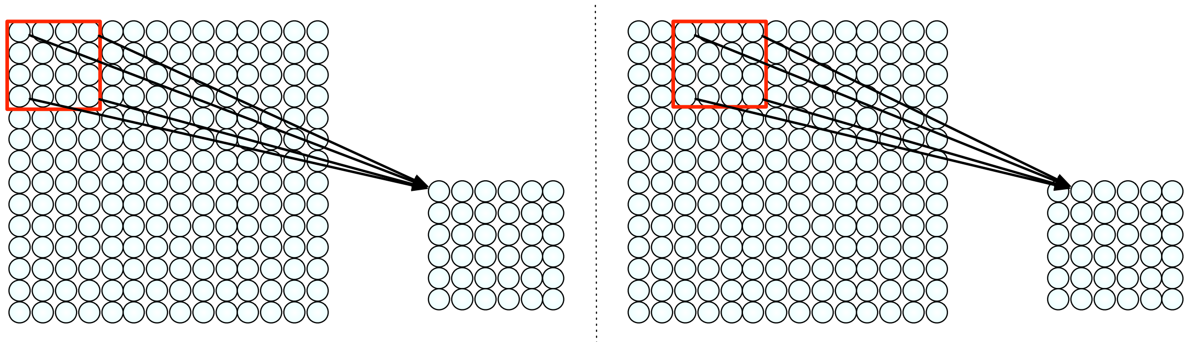
- Each 4x4 patch connected to a hidden unit in 1st hidden layer
- Thus, 16 weights for encoding the patch (each hidden unit also has a bias term)
- Total 36 overlapping patches in above picture, total 16 x 36 = 576 weights
- Thus: 14 x 14 input image mapped into 6 x 6 image via 576 weights.
- If we use a stride of 1, number of hidden units is 11, so 121x16=1936 weights!

Exercise: How many weights if input image is 512 x 512?

CNNs: key idea

Idea: Don’t use so many weights. It’s overparametrization, hard to learn, unhealthy

Weight-sharing: use same weights for each patch!



Thus, for these 36 hidden neurons we end up with 4x4=16 weights

Patchy operation aka convolution

$$f\left(\sum_{i=1}^{4}\sum_{j=1}^{4} w_{ij}x_{i+p,j+q} + b\right)$$

Hidden unit (p,q) takes pixels in its patch and computes weighted sum (note, weights are indp. of the patch), add a bias, then activate using ‘f’

Convolutions

orig | sharpen | edge detect | strong edges

* Fig illustrates idea of extracting local features
* Different weights for patch extract different local features
* Important point: convolution is a linear operation. We move same window of weights over all patches and compute linear combinations.
* Viewing an input d×d image as d^2 vector, convolution can be represented by a matrix of size $d^2 \times d^2$, albeit a very sparse one.

Exercise: Say input is 5×5 and we use a 2×2 patch with stride of 1. Write down the matrix representation of the convolution operation. Redo with stride 2; discover zero-padding.

Convolutions

Practical details about training CNNs

http://cs231n.stanford.edu/slides/2016/winter1516_lecture11.pdf

(Implementation points, other useful details about training CNNs)

Important remarks

For dxd image, convolution written as matrix is $d^2 \times d^2$. Direct matvec costs d^4. Can be greatly accelerated by FFT / variants ~ $d^2 \log d$ (or even $d^2 \log(P)$, using OLA tricks)

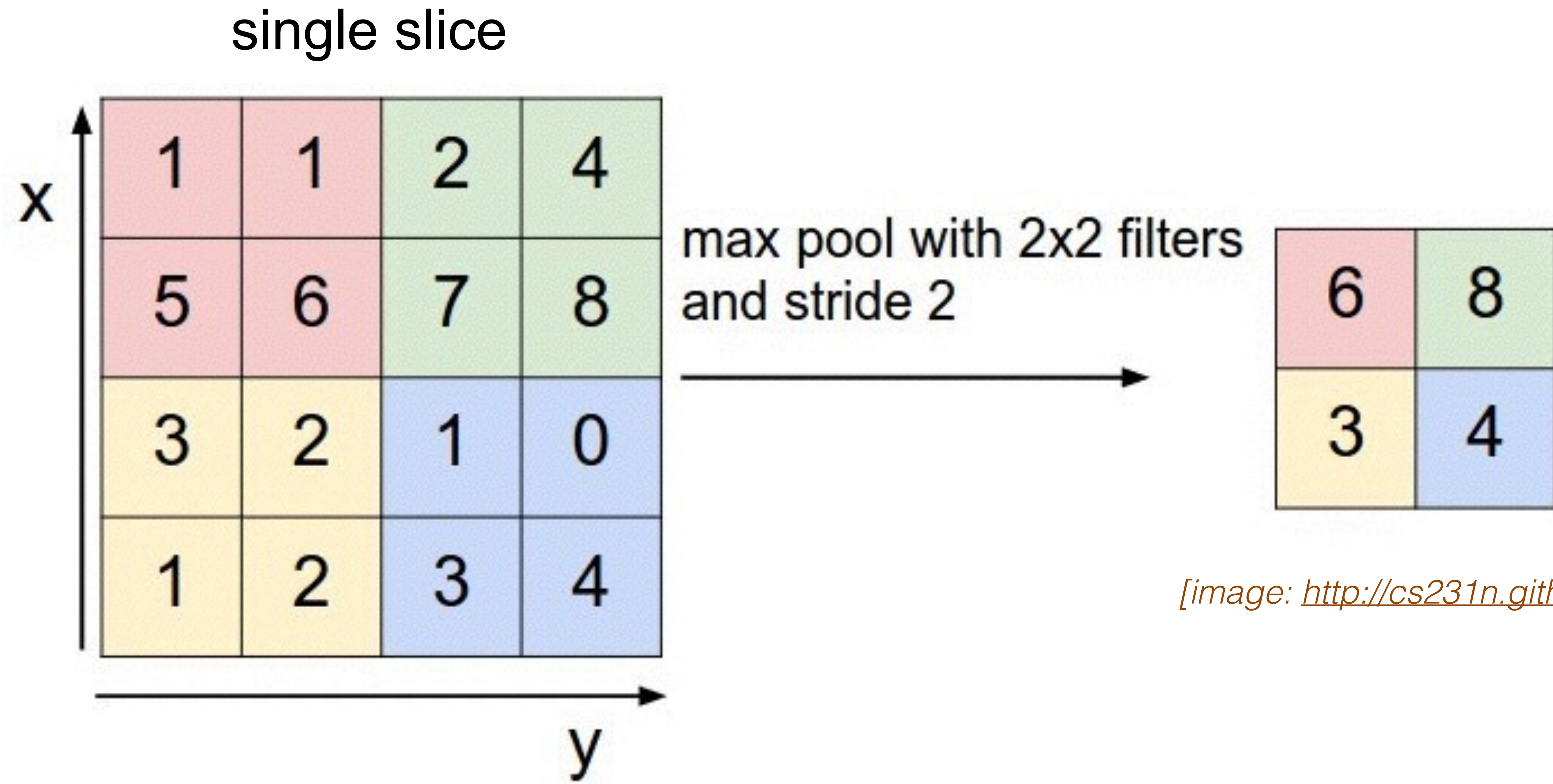
Currently, smaller patch sizes such as 3x3 very popular. Turns out brute-force implementation (without FFT) can give higher throughput! (impt. lesson!)

Even 1x1 “convolutions” (justified more easily for color images with 3 channels)

Efficient support from architecture, GPUs, etc. has been critical

wrapping up CNNs….

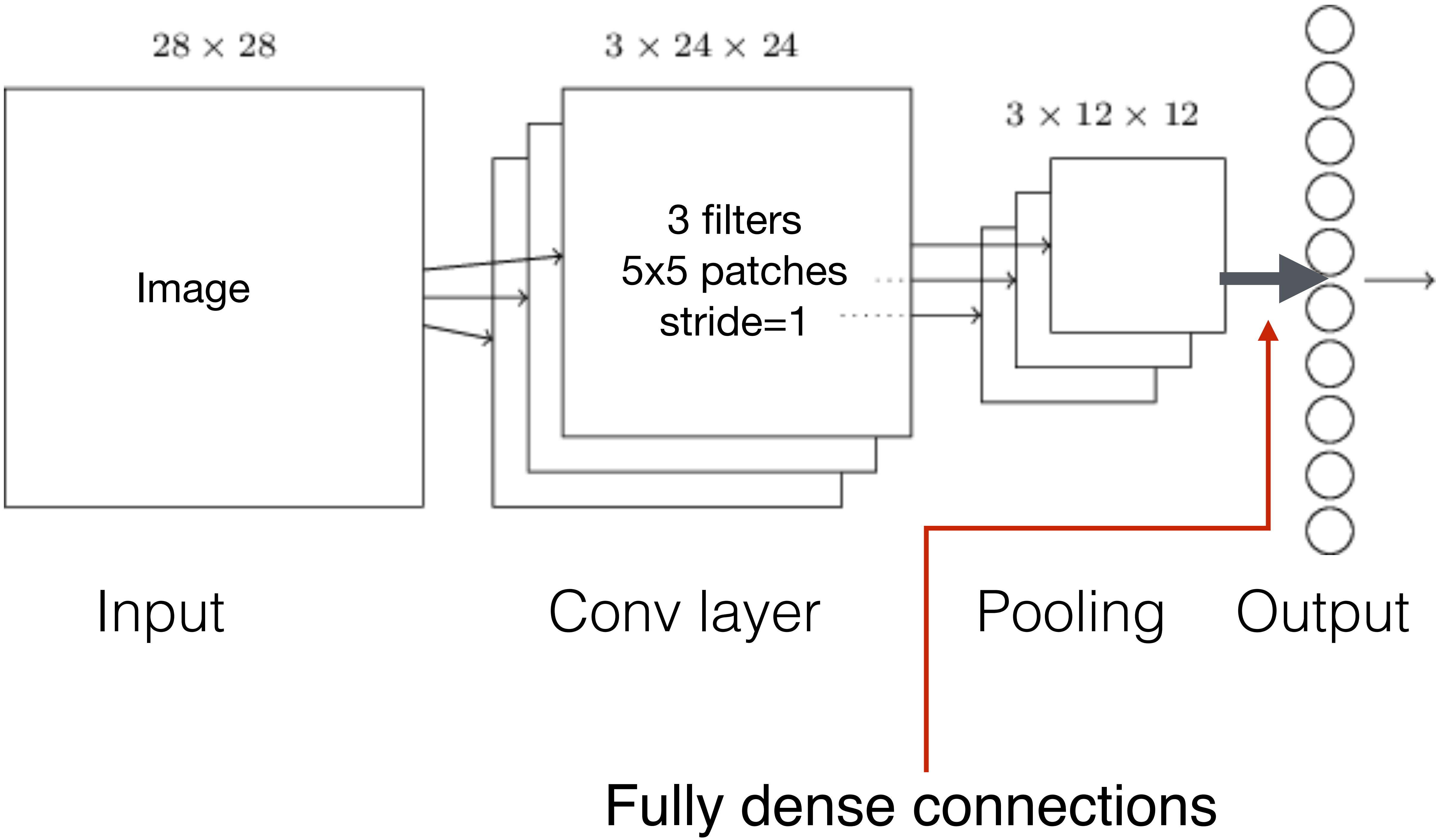
max-pooling / subsampling



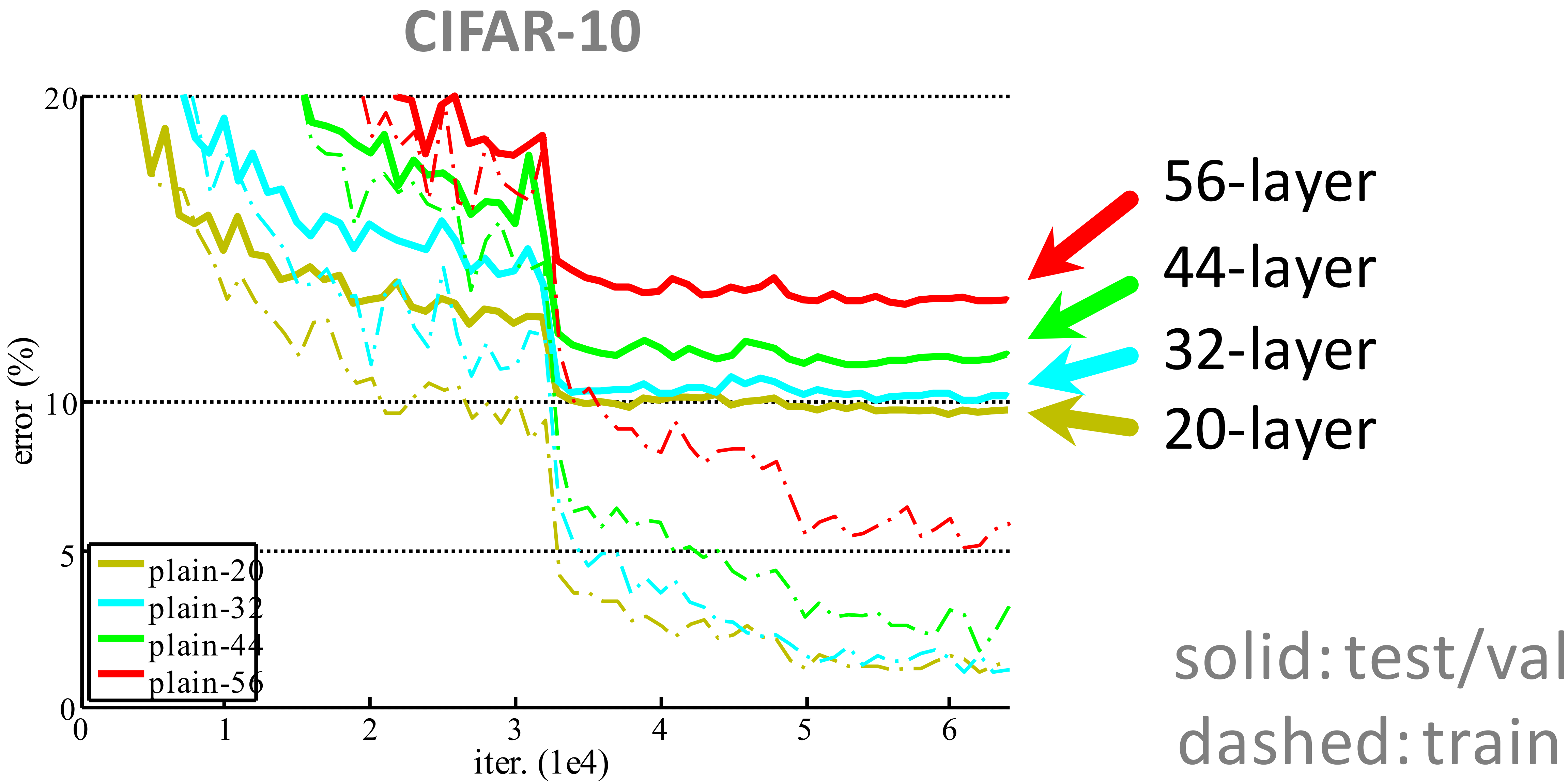
[image: http://cs231n.github.io]

Explore: Try out other types of "pooling" operations to reduce size, enhance accuracy

Example convnet



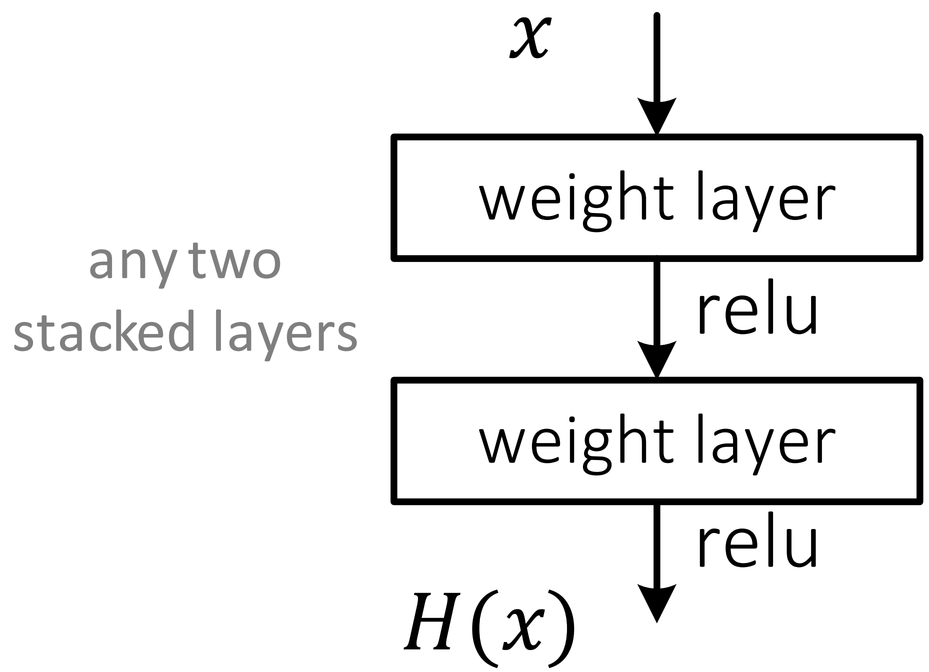
Residual Networks
(Resnets)



Making network deeper does not necessarily work better

Limits on what initialization and batch normalization give us

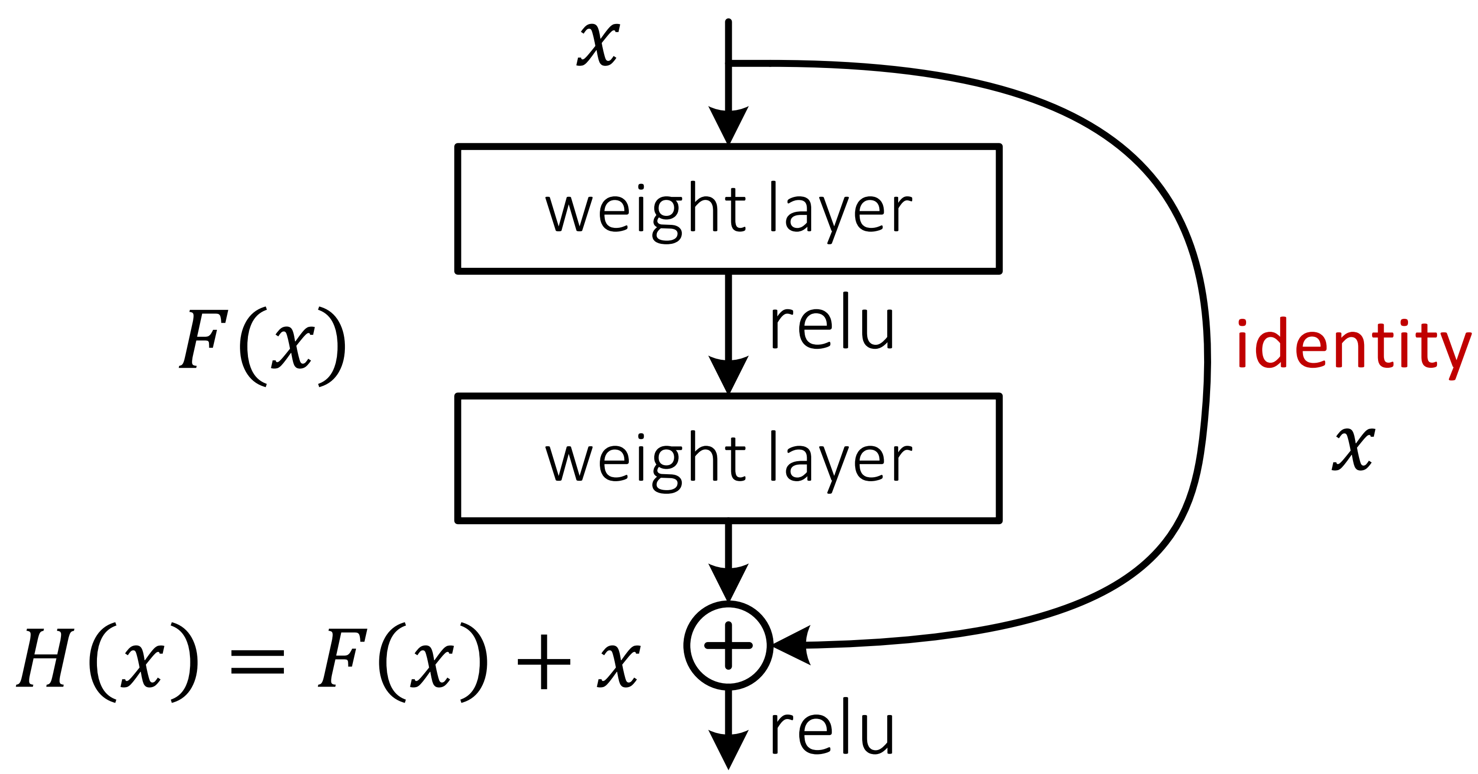
Key idea: Identity maps



Aim: Learn map H(x).
Approach: Hope the deep net fits H(x)

Kaiming He, Xiangyu Zhang, Shaoqing Ren, & Jian Sun. "Deep Residual Learning for Image Recognition". CVPR 2016.

Key idea: Identity maps



If identity were optimal easy to fit by setting weights=0

If optimal not too far from identity, easier to find small fluctuations

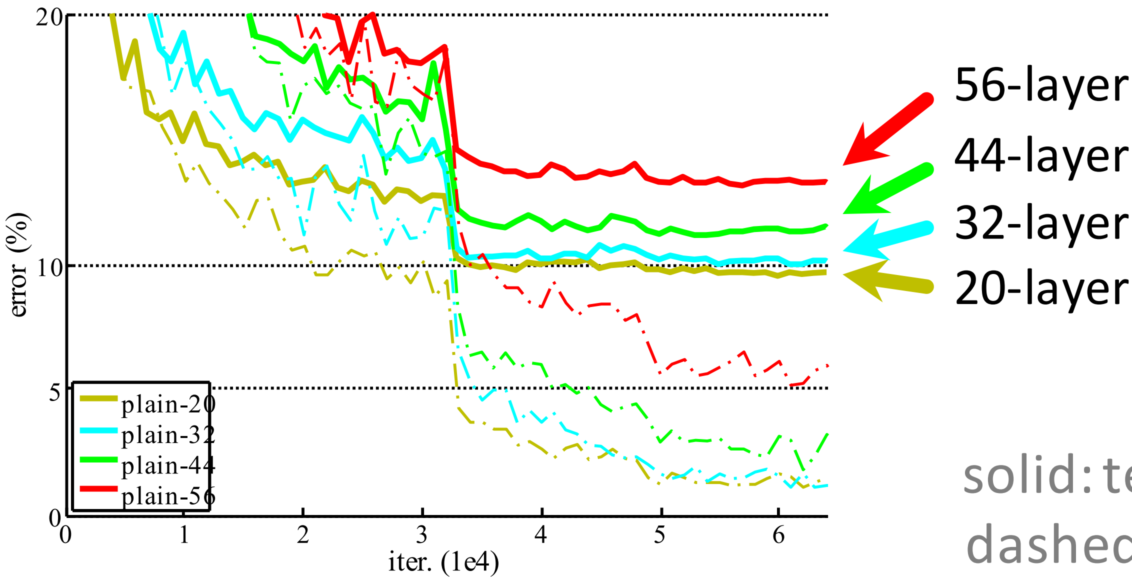
Aim: Learn map H(x) = F(x)+x
Approach: Hope the deep net fits F(x)

F(x) is a **residual** mapping wrt identity

Explore: Try residual wrt other distinguished mappings

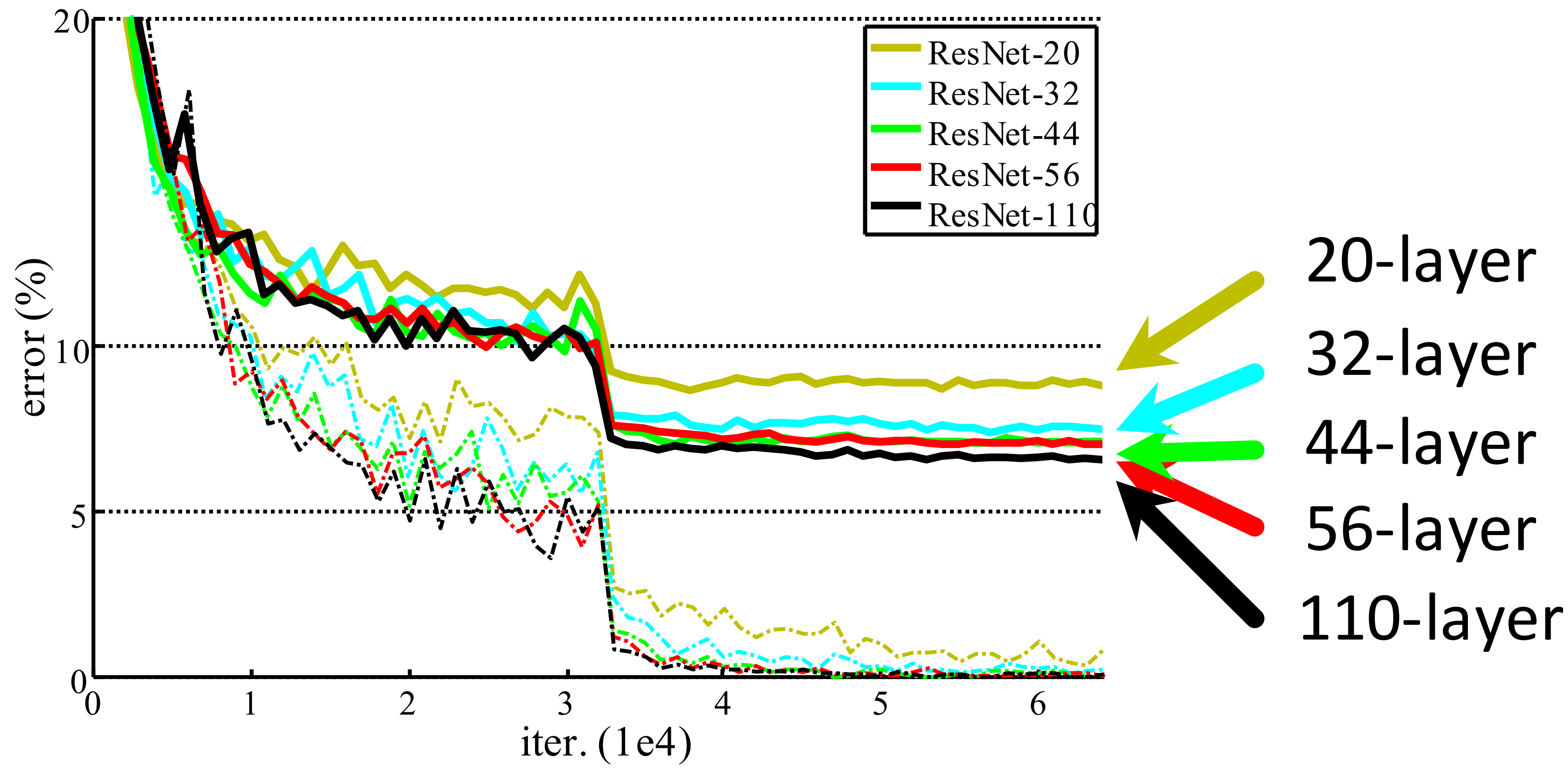
Kaiming He, Xiangyu Zhang, Shaoqing Ren, & Jian Sun. "Deep Residual Learning for Image Recognition". CVPR 2016.

CIFAR-10



solid: test/val
dashed: train

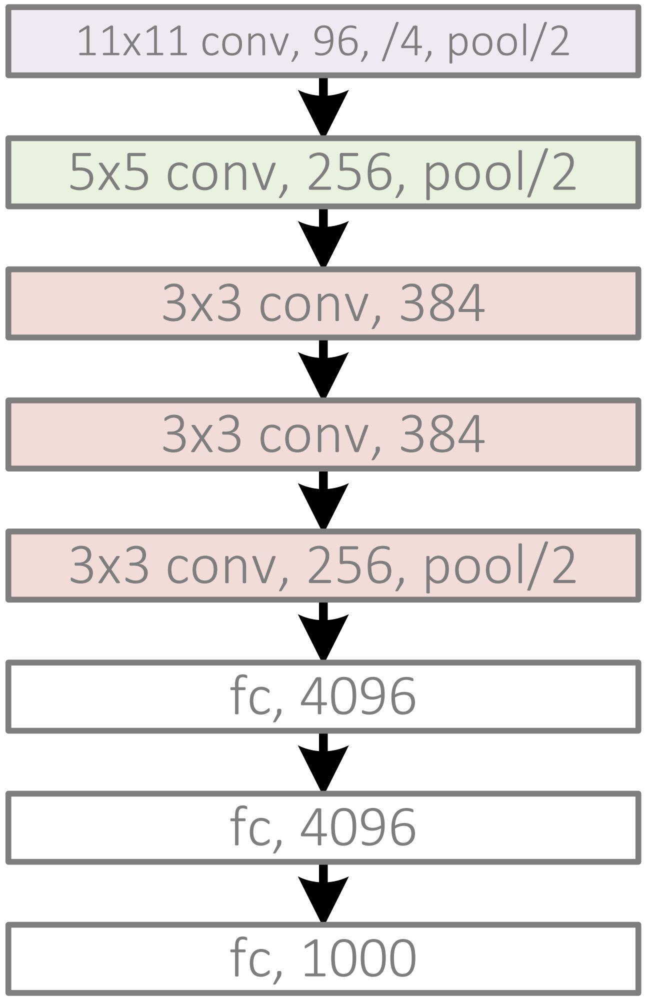
CIFAR-10 ResNets



Kaiming He, Xiangyu Zhang, Shaoqing Ren, & Jian Sun. “Deep Residual Learning for Image Recognition”. CVPR 2016.

Resnet architecture

AlexNet, 8 layers
(ILSVRC 2012)

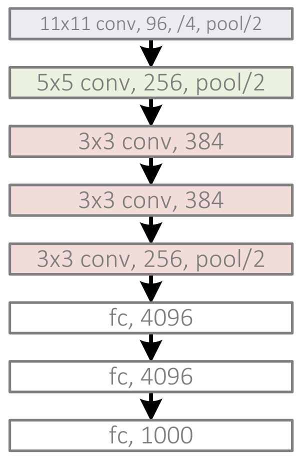


Kaiming He, Xiangyu Zhang, Shaoqing Ren, & Jian Sun. “Deep Residual Learning for Image Recognition”. CVPR 2016.

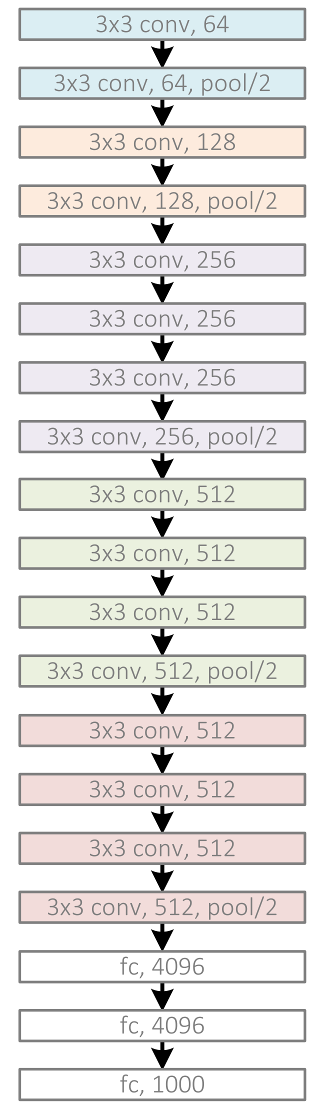
[Kaimeng He, ICML 2016 Tutorial]

Resnet architecture

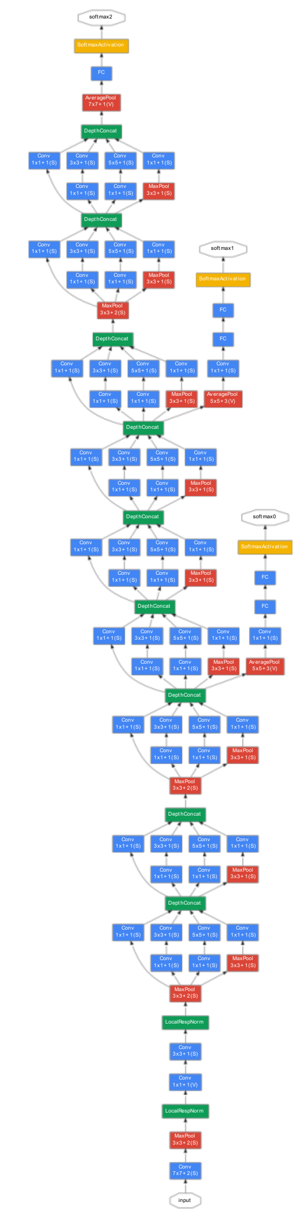
AlexNet, 8 layers
(ILSVRC 2012)



VGG, 19 layers
(ILSVRC 2014)



GoogleNet, 22 layers
(ILSVRC 2014)

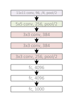


Kaiming He, Xiangyu Zhang, Shaoqing Ren, & Jian Sun. "Deep Residual Learning for Image Recognition". CVPR 2016.

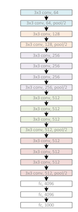
[Kaimeng He, ICML 2016 Tutorial]

Resnet architecture

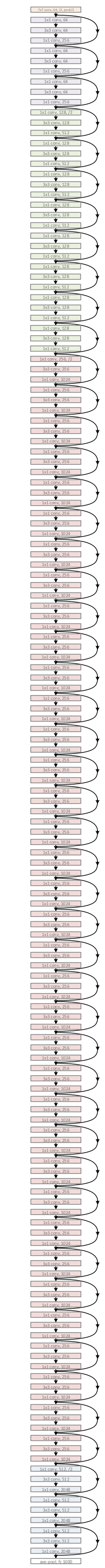
AlexNet, 8 layers
(ILSVRC 2012)



VGG, 19 layers
(ILSVRC 2014)



ResNet, 152 layers
(ILSVRC 2015)



Exercise: Work out the forward-pass for a Resnet

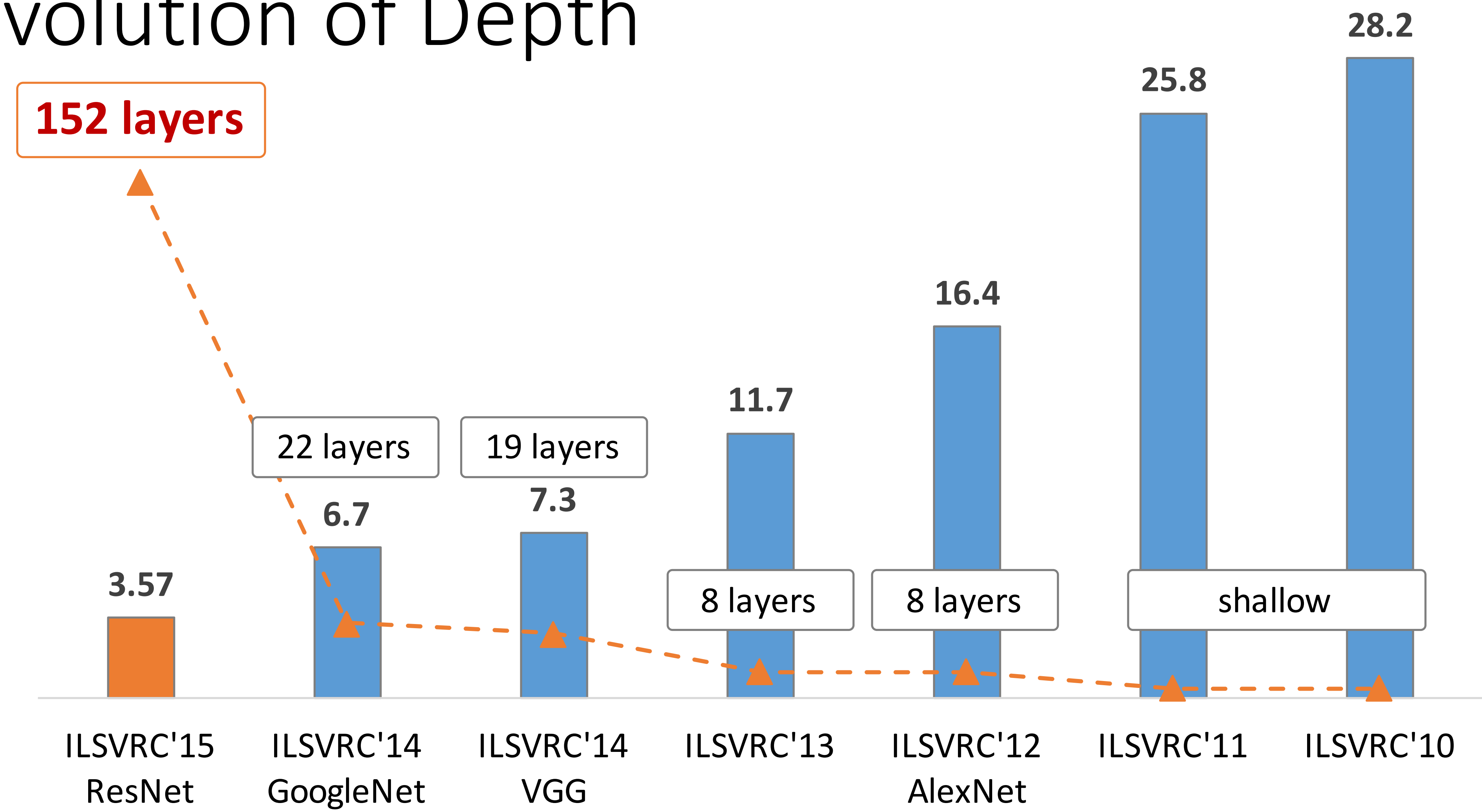
Exercise: Obtain backprop for computing gradients in Resnets

Think: Infinitely deep networks?

Kaiming He, Xiangyu Zhang, Shaoqing Ren, & Jian Sun. “Deep Residual Learning for Image Recogniti

[Kaimeng He, ICML 2016 Tutorial]

Revolution of Depth



ImageNet Classification top-5 error (%)

Kaiming He, Xiangyu Zhang, Shaoqing Ren, & Jian Sun. "Deep Residual Learning for Image Recognition". CVPR 2016.

[Kaimeng He, ICML 2016 Tutorial]