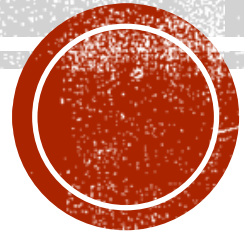
CS7.505: COMPUTER VISION

— TRAINING CNNS, TRANSFER LEARNING AND ARCHITECTURES

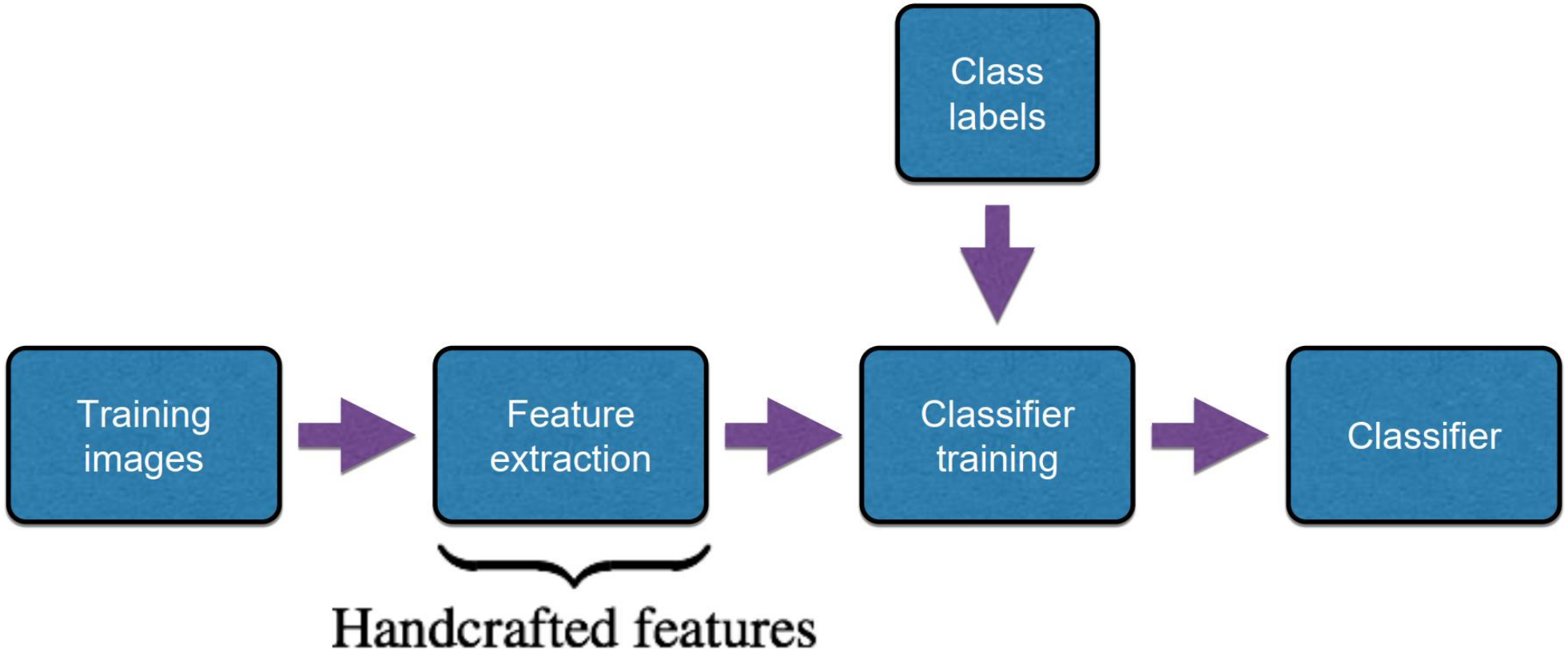


Dr. Sudipta Banerjee, CVIT

22/03/2022

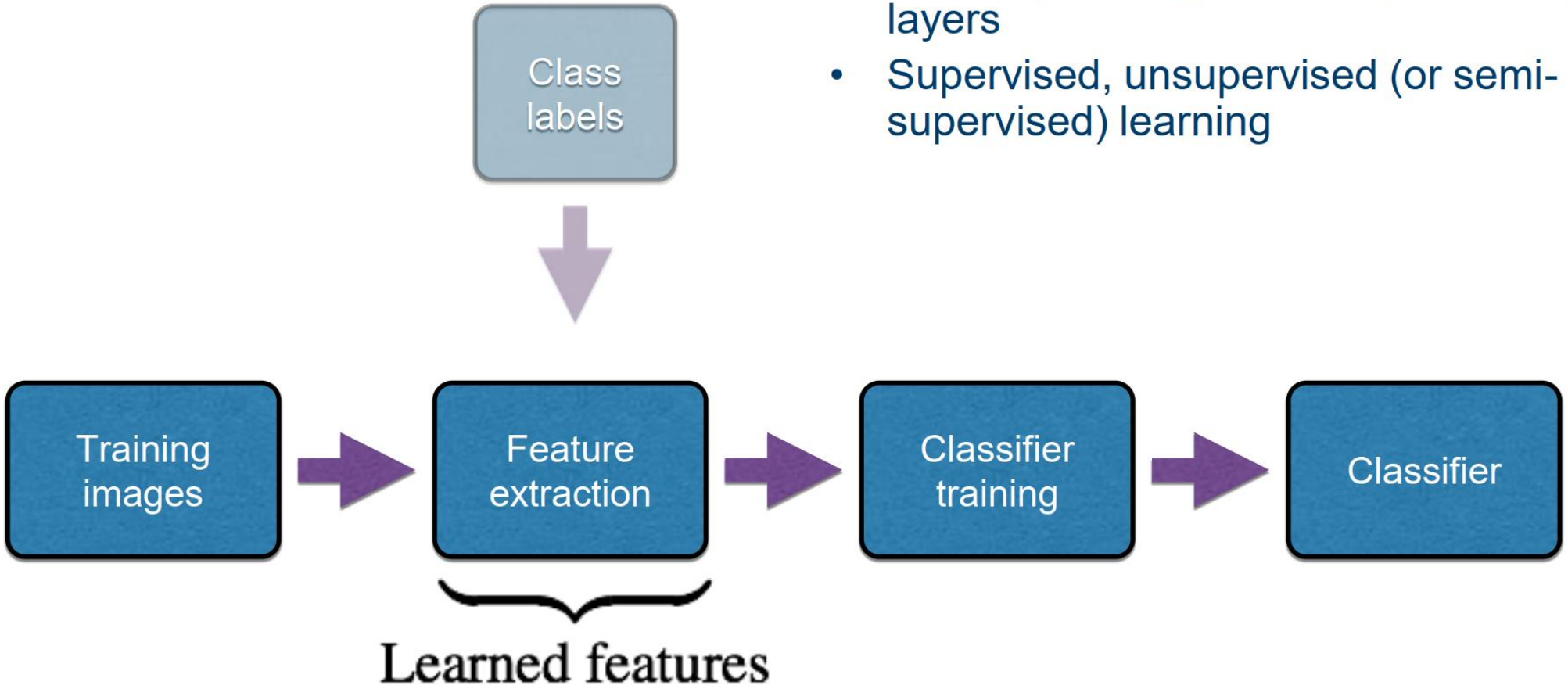
Slides have been adapted from various tutorials and online sources. Proper attribution has been given in the References slide

Traditional supervised learning



Deep learning

- Learning of weights in the processing layers
- Supervised, unsupervised (or semi-supervised) learning



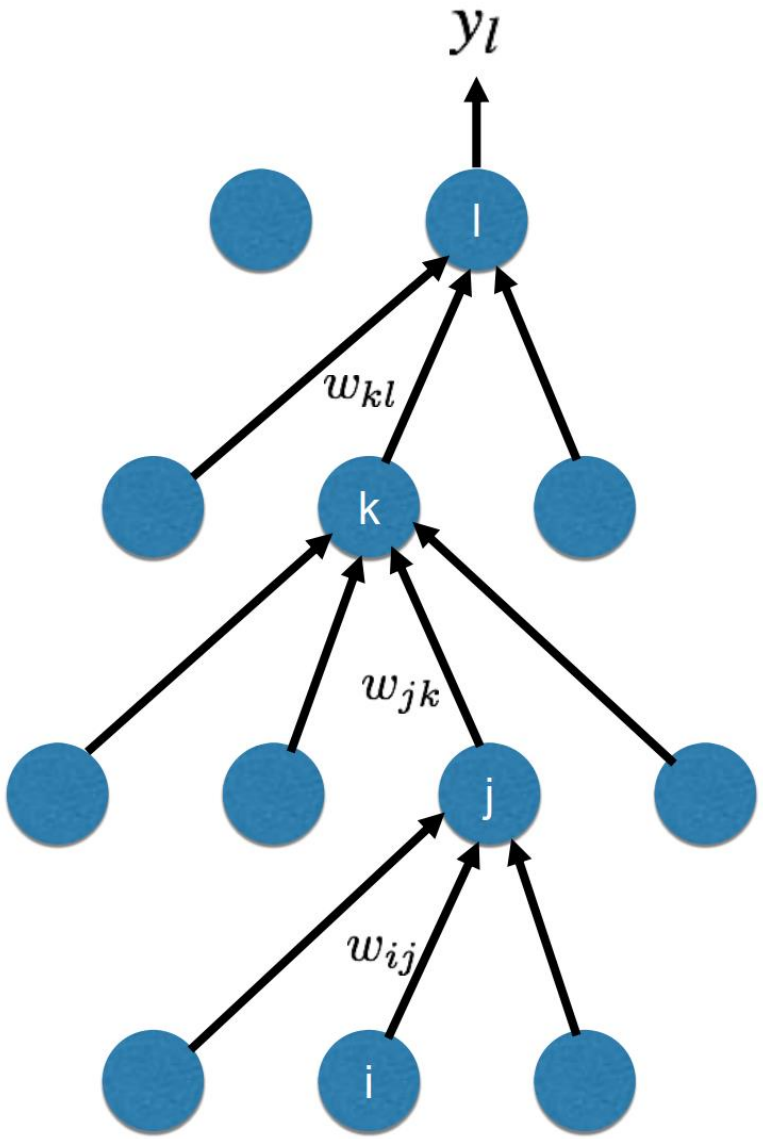
Feed-forward neural network

Output layer

Hidden layer H_2

Hidden layer H_1

Input layer



$$y_l = f(z_l)$$

$$z_l = \sum_{k \in H_2} w_{kl} x_k$$

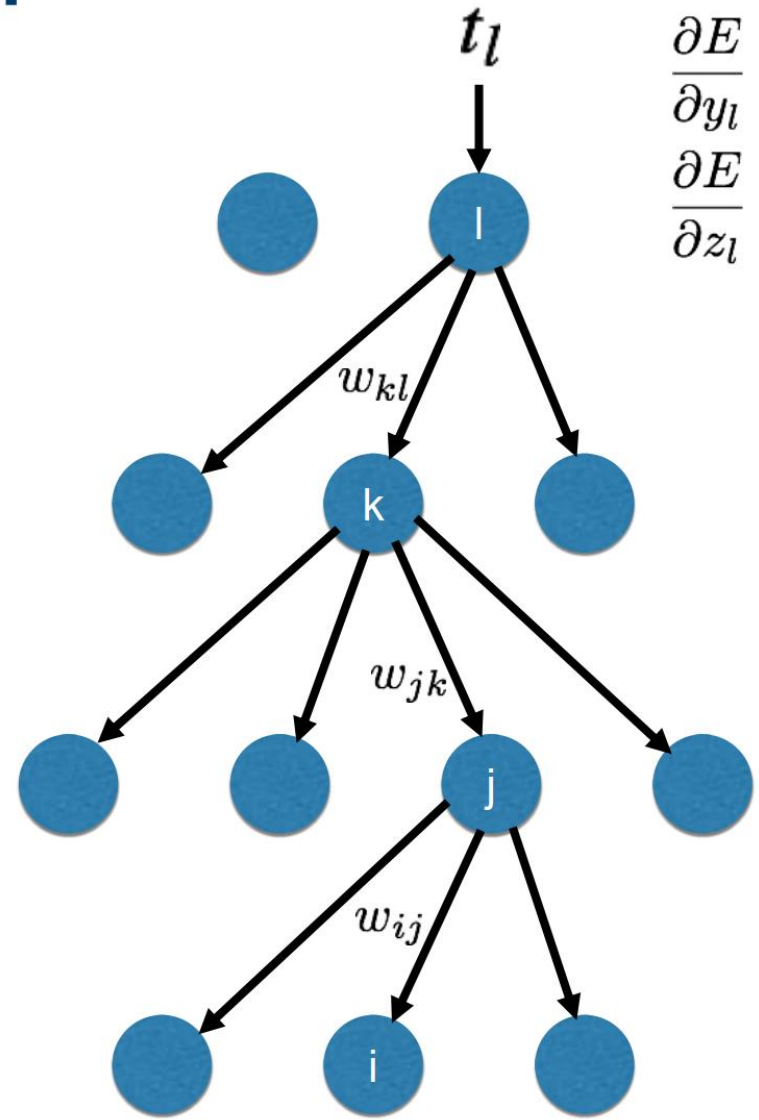
$$y_k = f(z_k)$$

$$z_k = \sum_{j \in H_1} w_{jk} x_j$$

$$y_j = f(z_j)$$

$$z_j = \sum_{i \in Input} w_{ij} x_i$$

Back-propagation



$$\frac{\partial E}{\partial y_l} = y_l - t_l$$

$$\frac{\partial E}{\partial z_l} = \frac{\partial E}{\partial y_l}\frac{\partial y_l}{\partial z_l}$$

$$E(\boldsymbol{w}) = \sum_{k=1}^{n}(t_i - y_i)^2$$

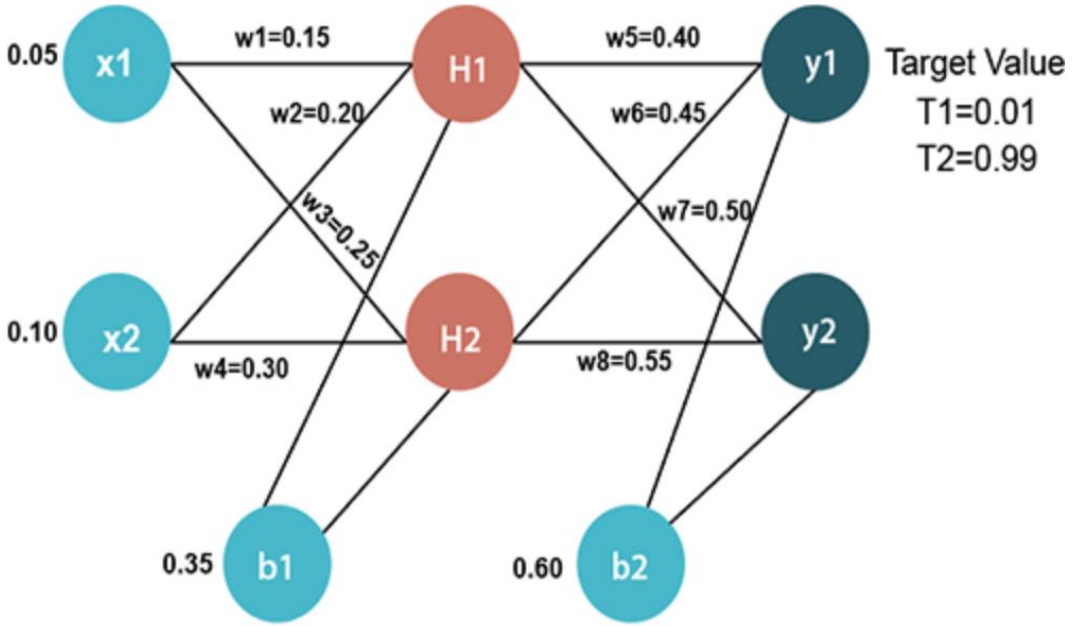
$$\frac{\partial E}{\partial y_k} = \sum_{l \in Output} w_{kl}\frac{\partial E}{\partial z_l}$$

$$\frac{\partial E}{\partial z_k} = \frac{\partial E}{\partial y_k}\frac{\partial y_k}{\partial z_k}$$

$$\frac{\partial E}{\partial y_j} = \sum_{k \in H_2} w_{jk}\frac{\partial E}{\partial z_k}$$

$$\frac{\partial E}{\partial z_j} = \frac{\partial E}{\partial y_j}\frac{\partial y_j}{\partial z_j}$$

EXAMPLE



$$H1 = x1 \times w_1 + x2 \times w_2 + b1$$

$$H1_{final} = \frac{1}{1 + \frac{1}{e^{H1}}}$$

$$H2 = x1 \times w_3 + x2 \times w_4 + b1$$

$$H2_{final} = \frac{1}{1 + \frac{1}{e^{H2}}}$$

$$y1 = H1 \times w_5 + H2 \times w_6 + b2$$

$$y1_{final} = \frac{1}{1 + \frac{1}{e^{y1}}}$$

$$y2 = H1 \times w_7 + H2 \times w_8 + b2$$

$$y2_{final} = \frac{1}{1 + \frac{1}{e^{y2}}}$$

$$E_{total} = \sum \frac{1}{2}(target - output)^2$$

$$Error_w = \frac{\partial E_{total}}{\partial w}$$

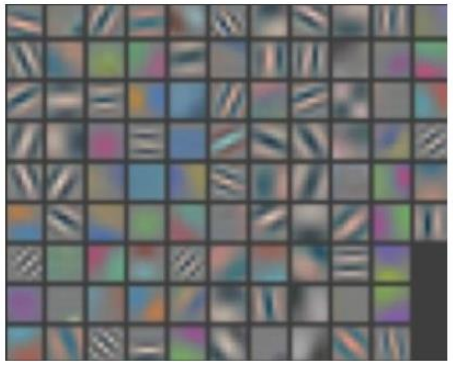
$$Error_{w5} = \frac{\partial E_{total}}{\partial w5} \ldots\ldots\ldots\ldots (1)$$

$$E_{total} = \frac{1}{2}(T1 - y1_{final})^2 + \frac{1}{2}(T2 - y2_{final})^2$$

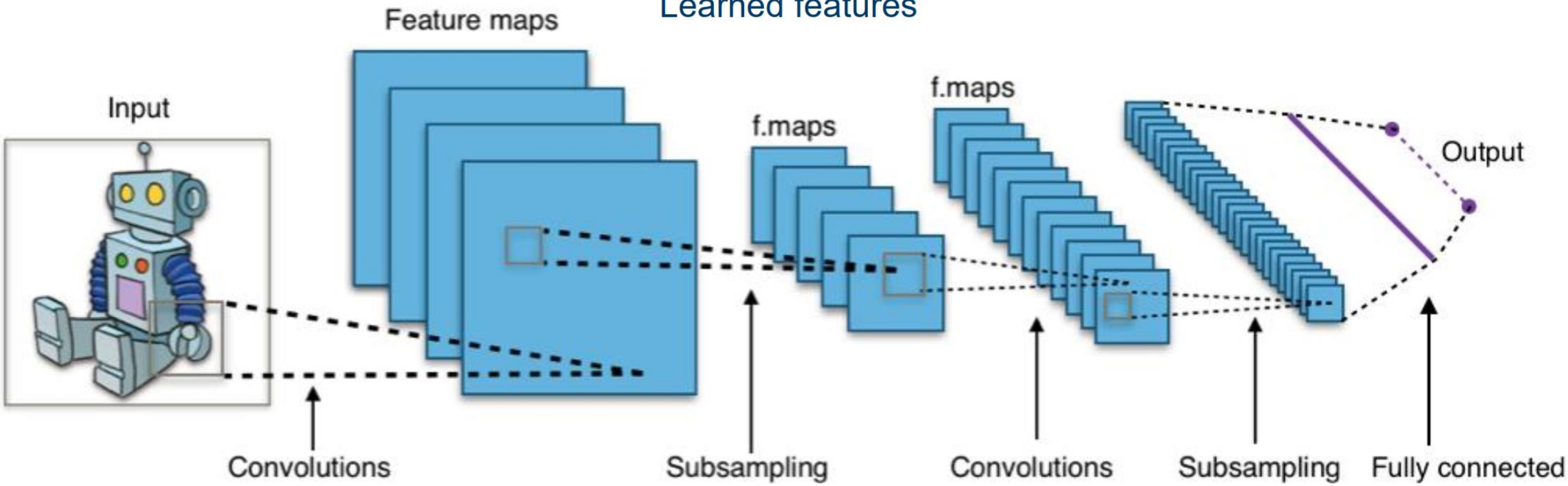
$$\frac{\partial E_{total}}{\partial w5} = \frac{\partial E_{total}}{\partial y1_{final}} \times \frac{\partial y1\_final}{\partial y1} \times \frac{\partial y1}{\partial w5}$$

$$w5_{new} = w5 - \eta \times \frac{\partial E_{total}}{\partial w5} \text{ Here, } \eta = \text{learning rate}$$

Typical CNN



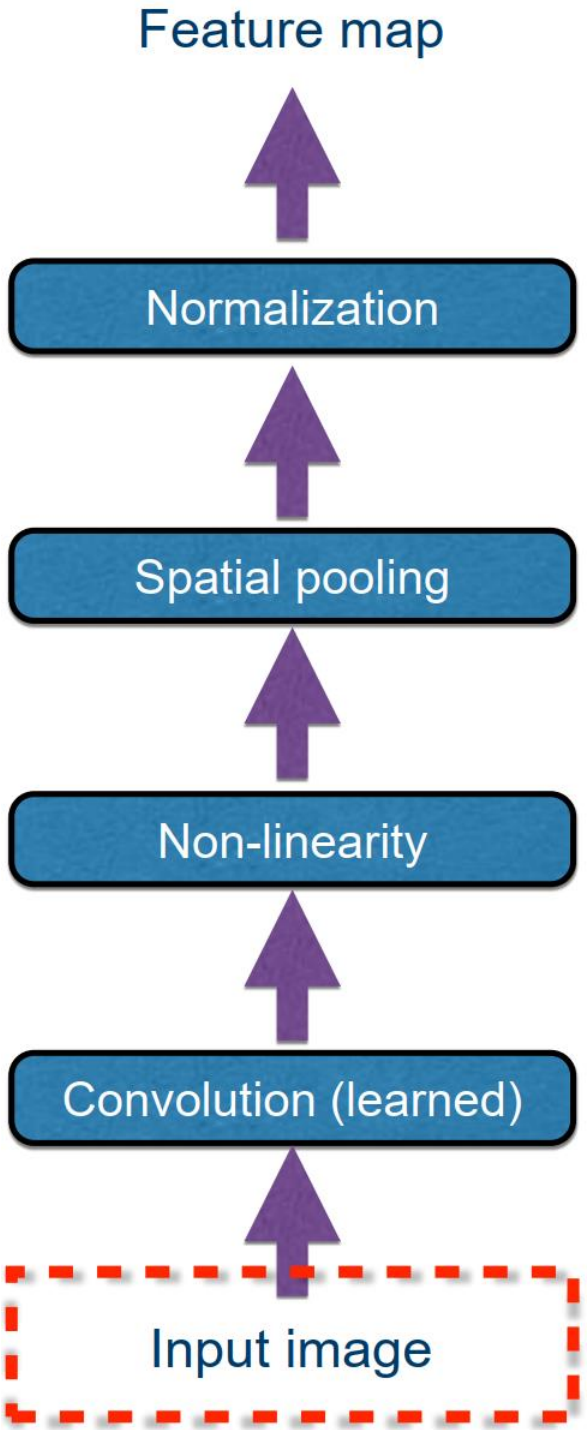
Learned features



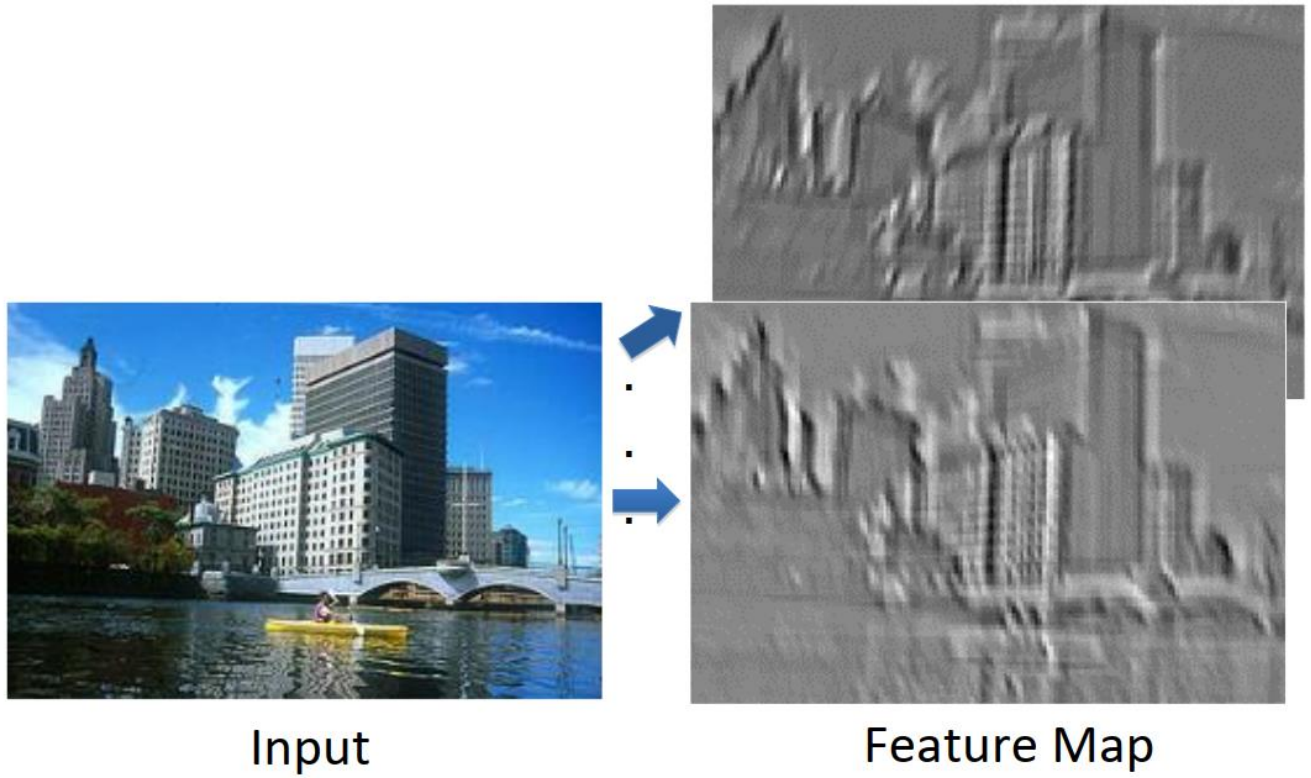
Convolutional neural net

Input image

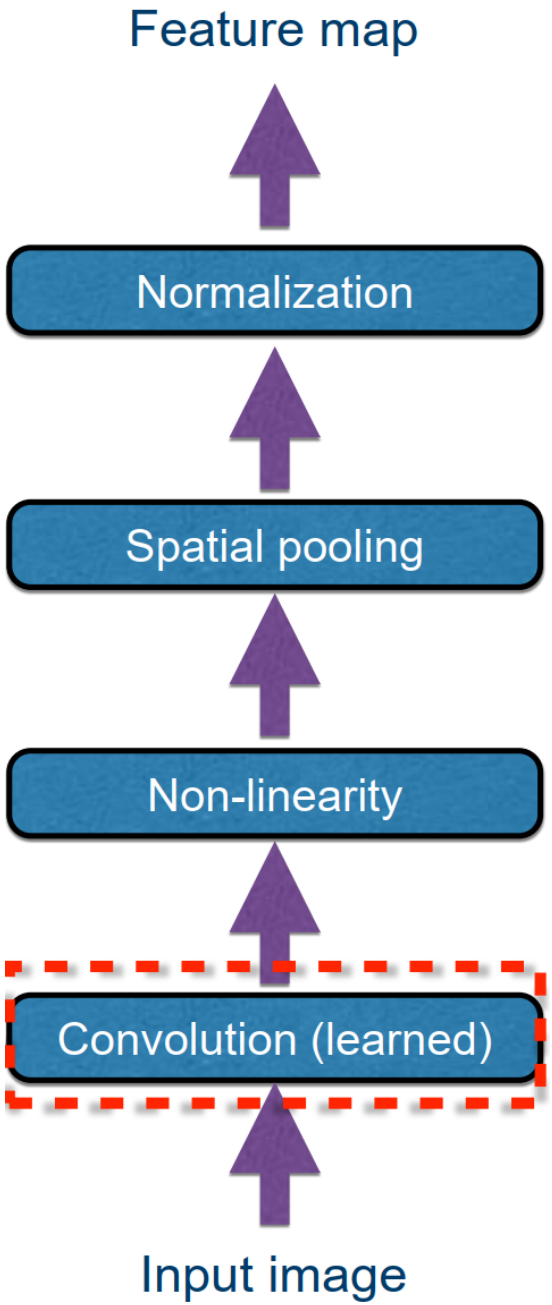
(credit: S. Lazebnik)



Convolutional neural net

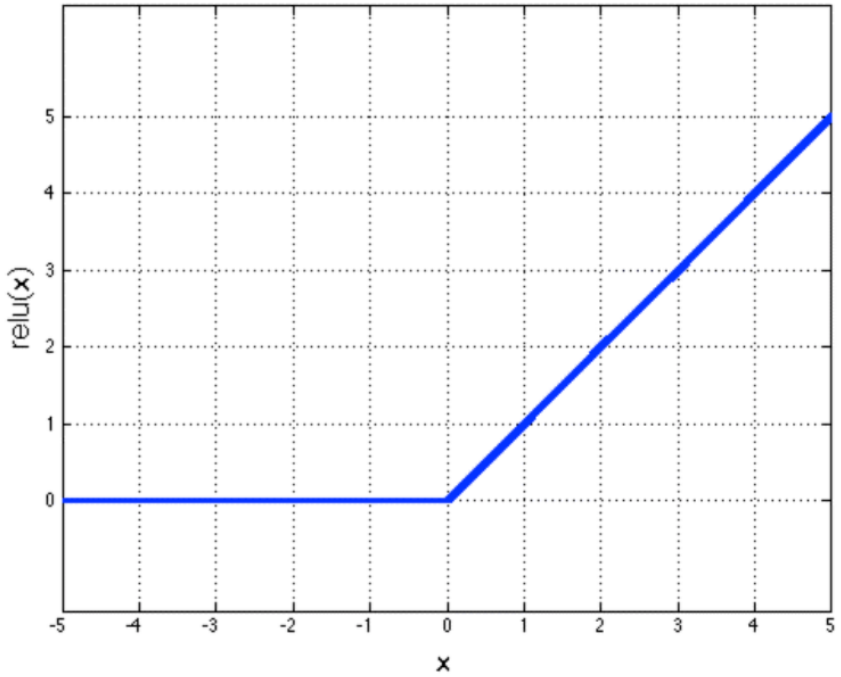


(credit: S. Lazebnik)

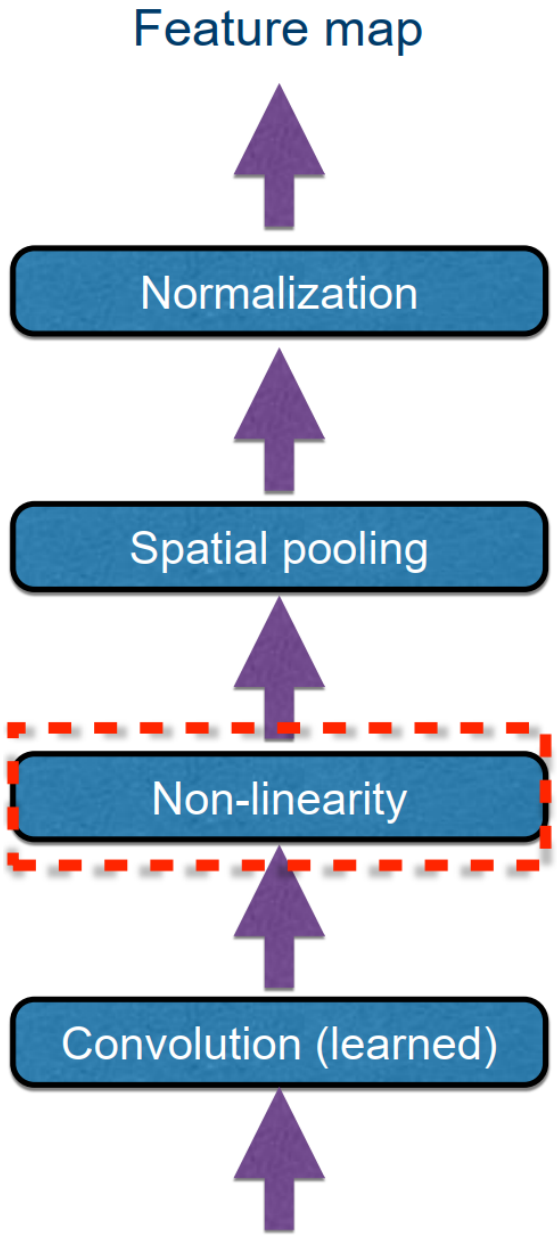


Convolutional neural net

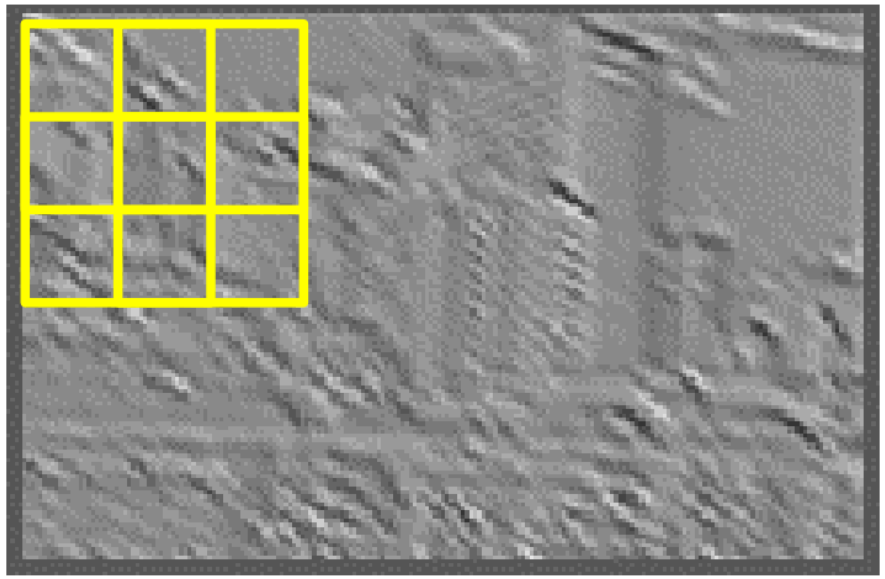
Rectified Linear Unit (ReLU)



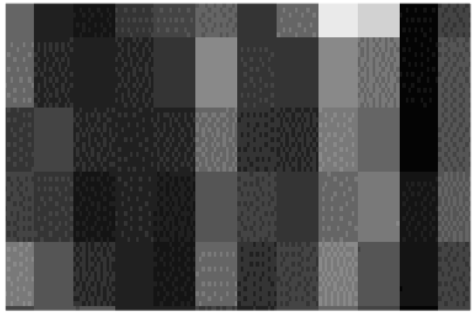
(credit: S. Lazebnik)



Convolutional neural net



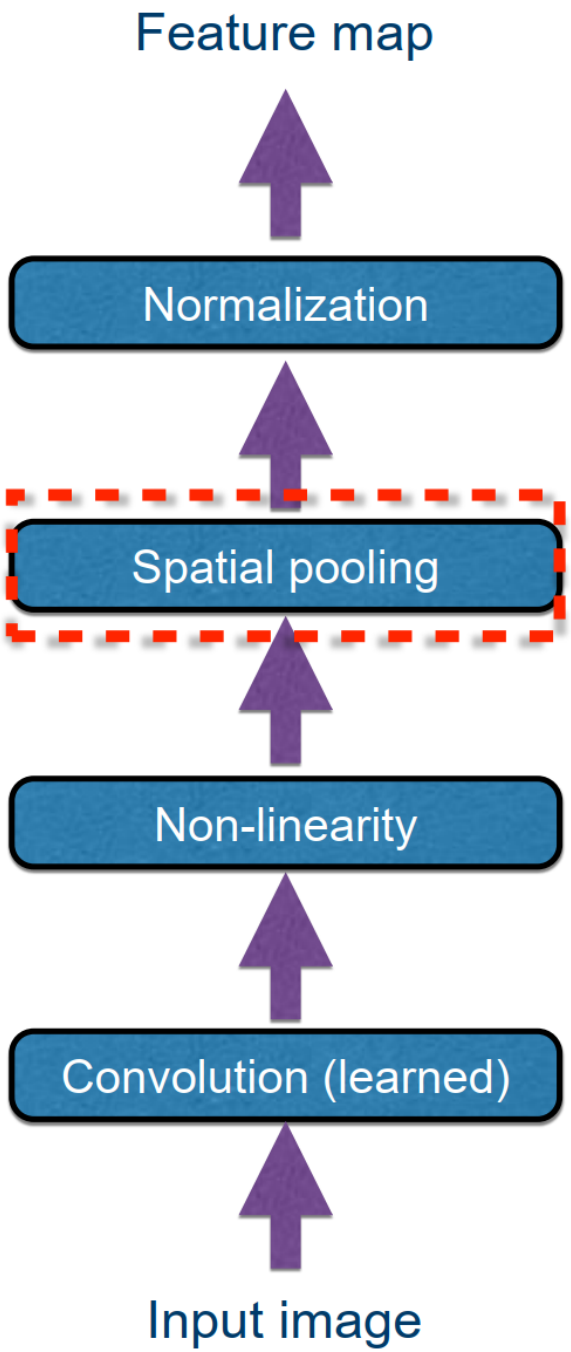
Max pooling



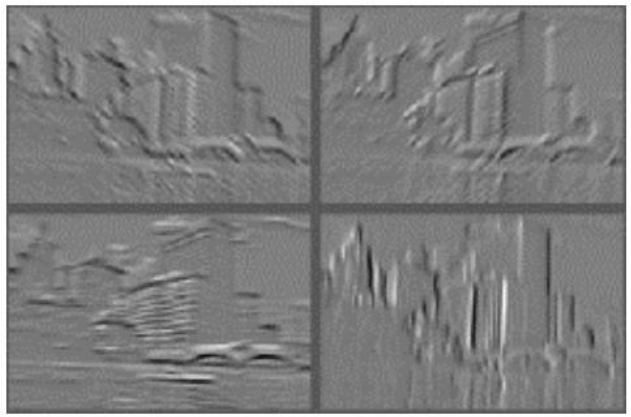
Max-pooling: a non-linear down-sampling

Provide *translation invariance*

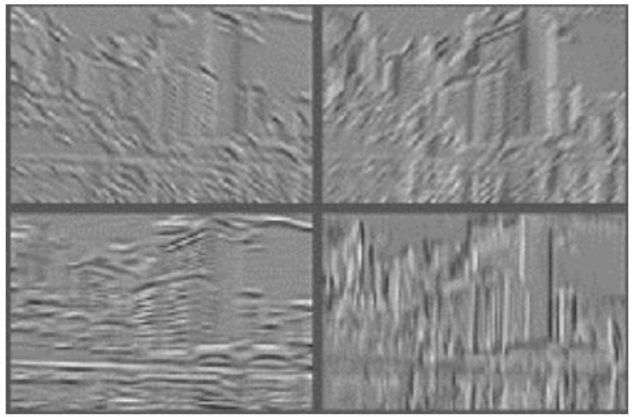
(credit: S. Lazebnik)



Convolutional neural net

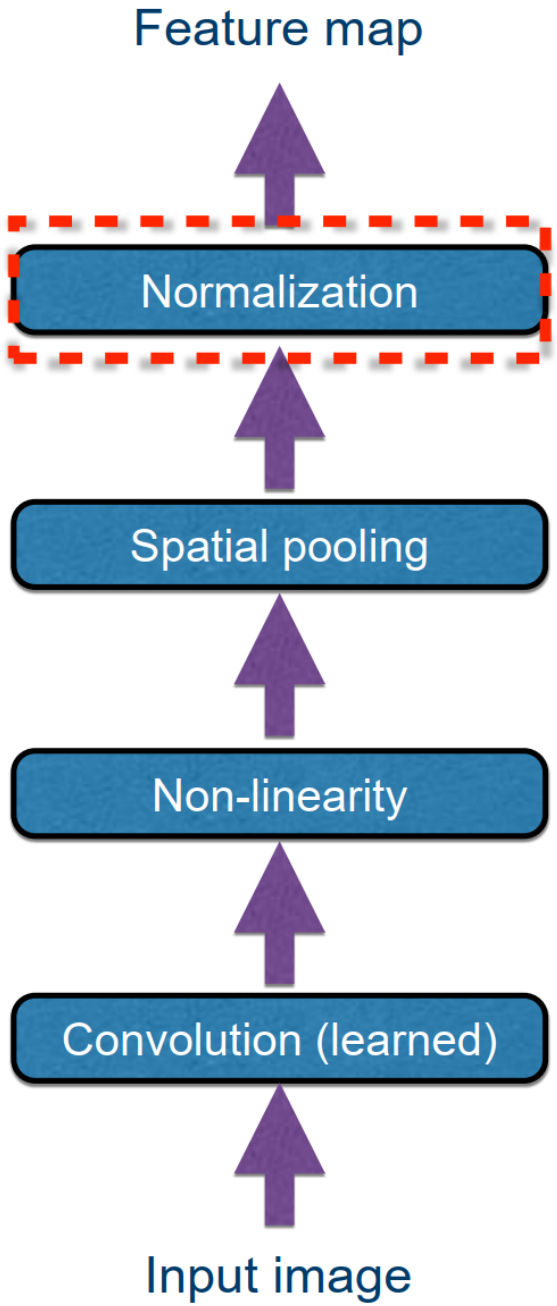


Feature Maps

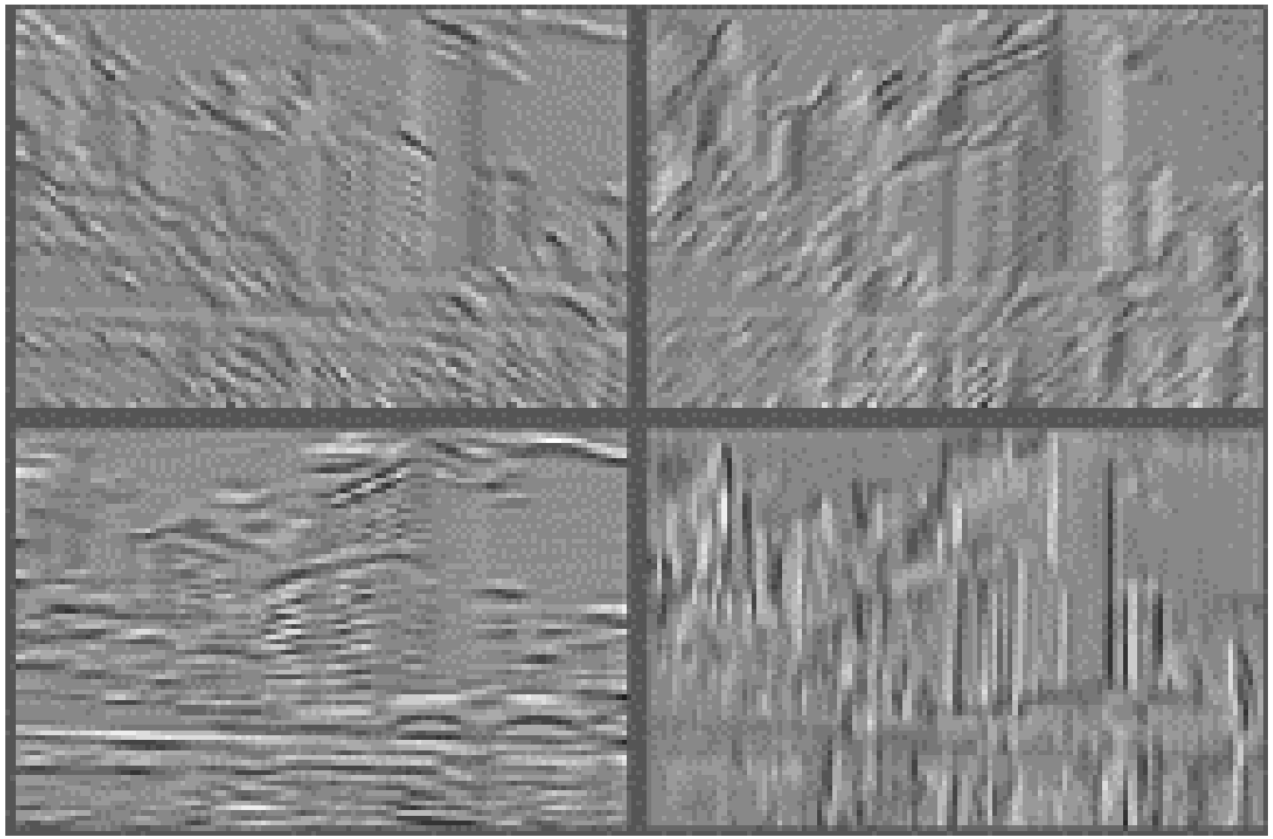


Feature Maps
After Contrast
Normalization

(credit: S. Lazebnik)

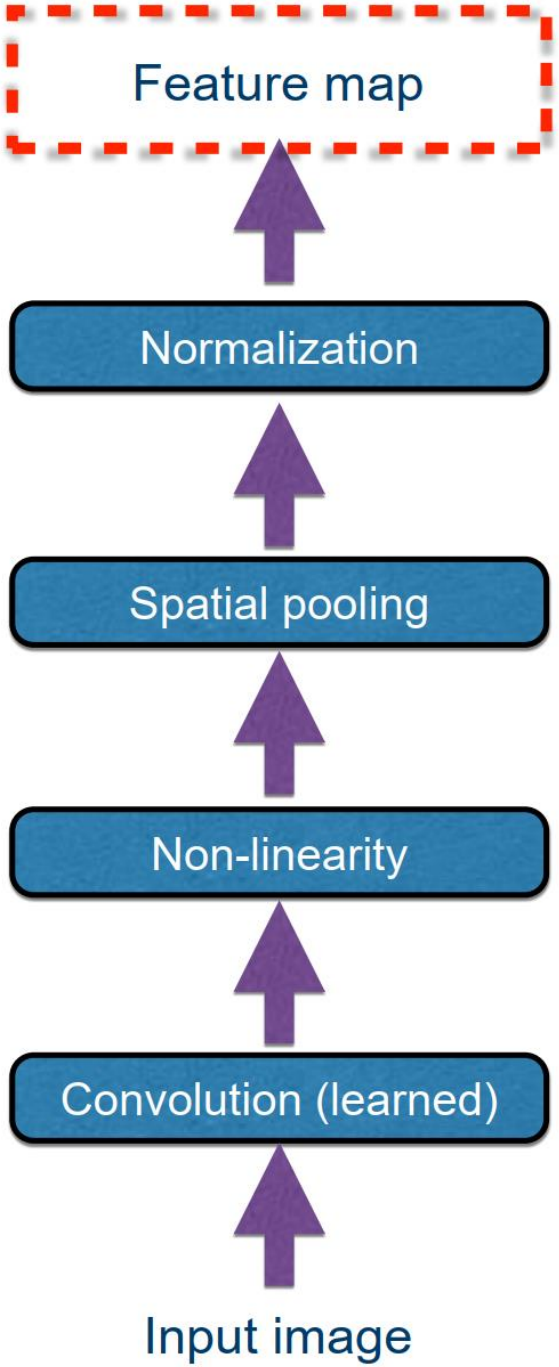


Convolutional neural net



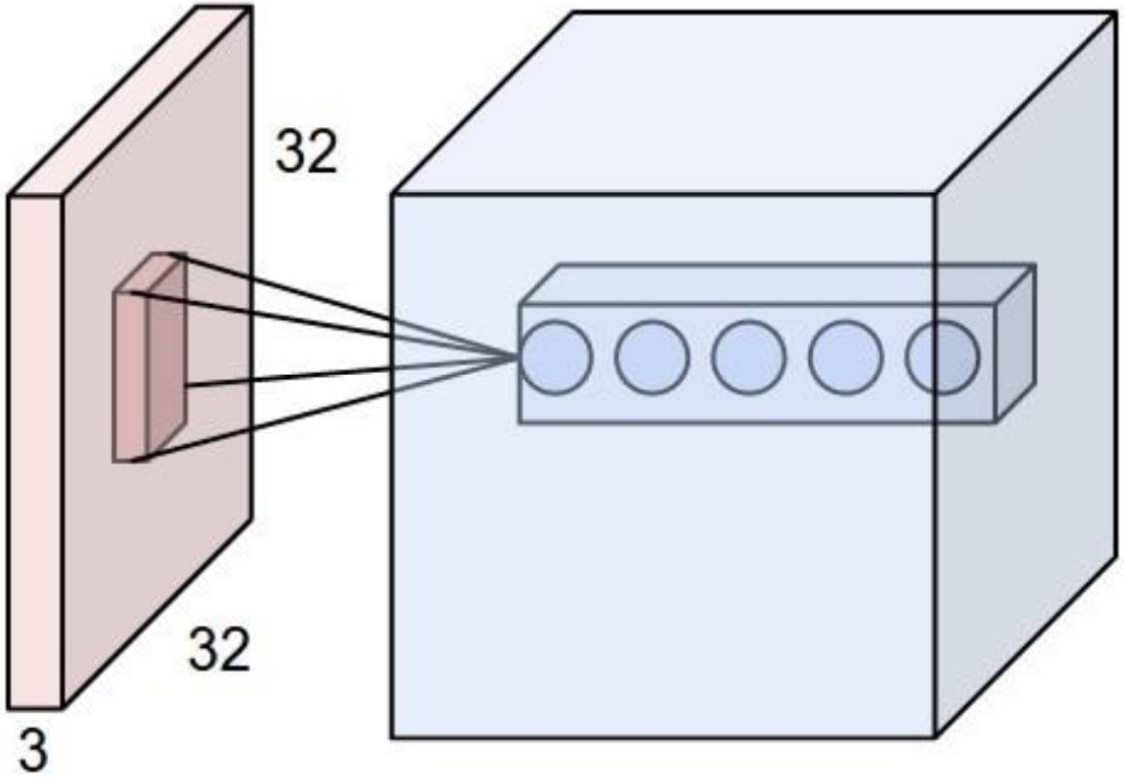
Feature maps after contrast normalization

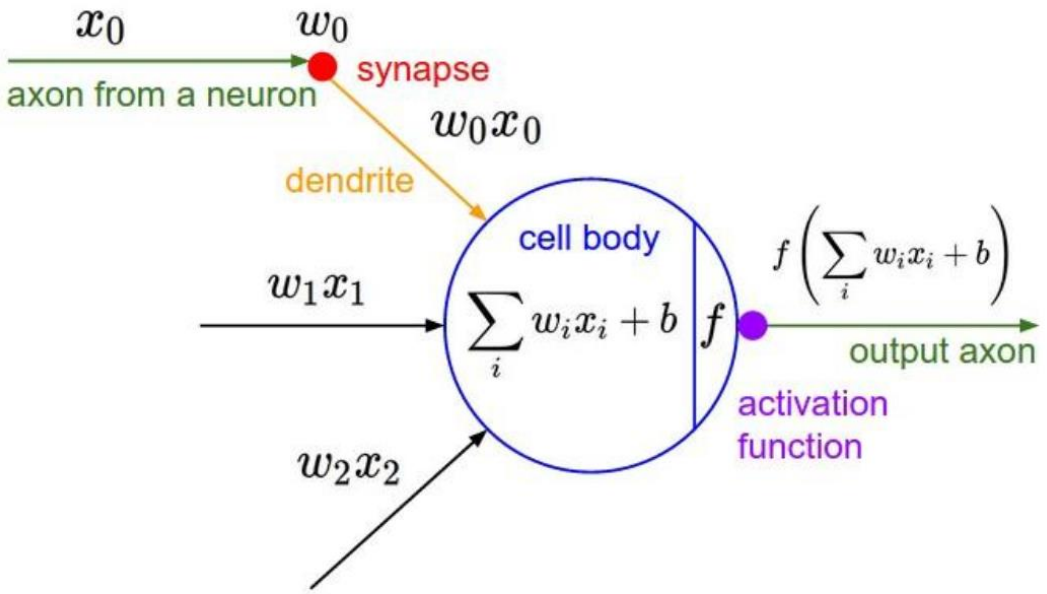
(credit: S. Lazebnik)



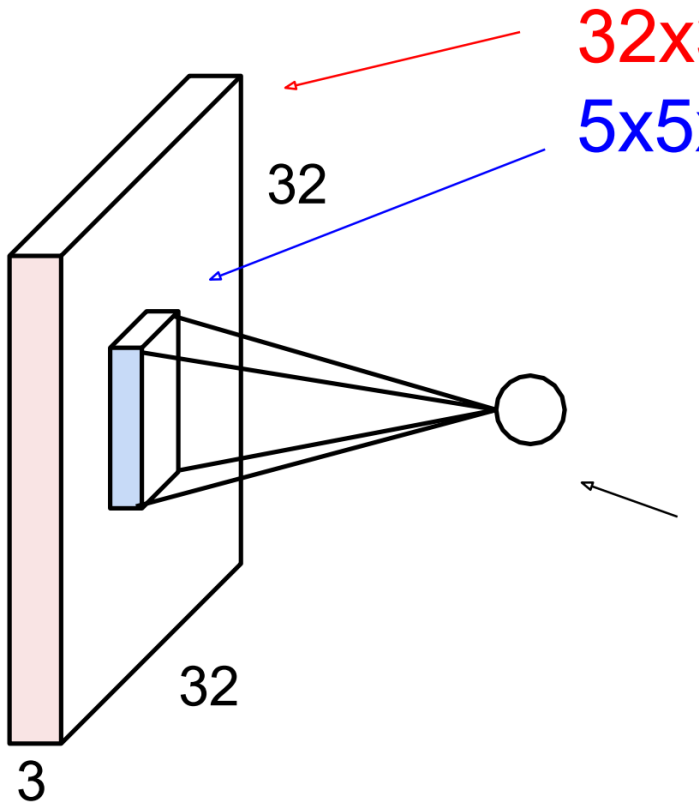
THE HEART OF CNN

- Convolution layers are the driving force behind CNN





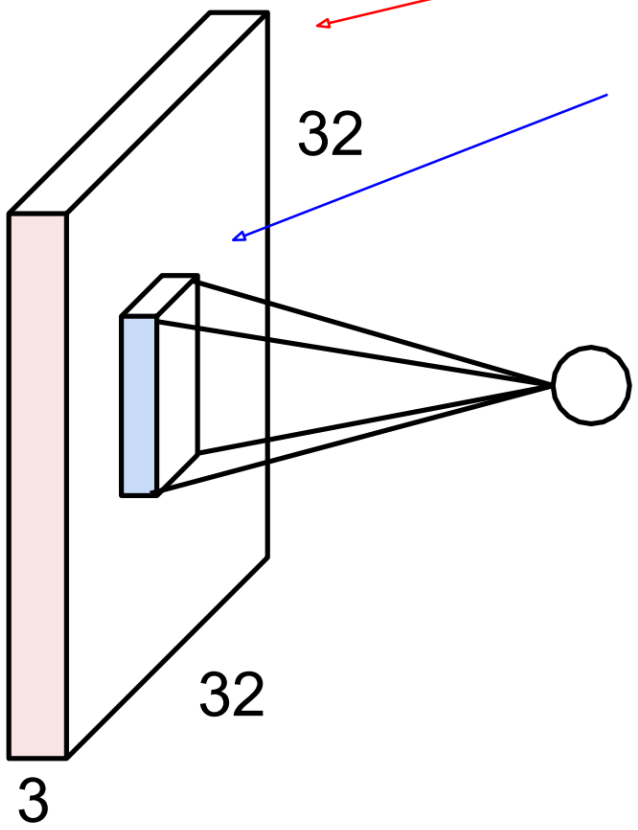
CONVOLUTION LAYER



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

$$w^T x + b$$

CONVOLUTION LAYER



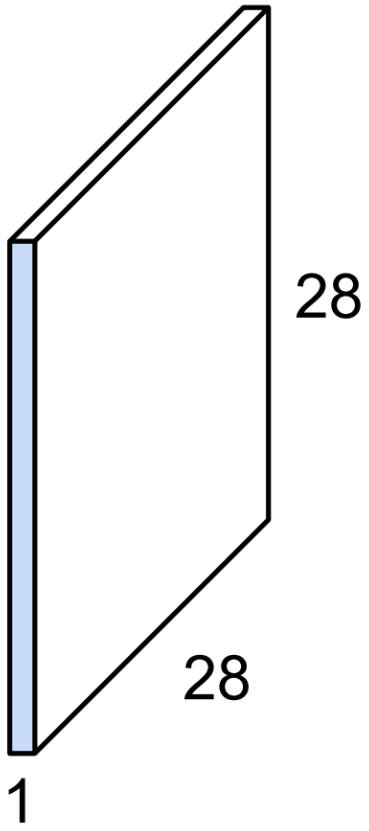
32x32x3 image

5x5x3 filter

$P = 0, S = 1$

convolve (slide) over all spatial locations

activation map



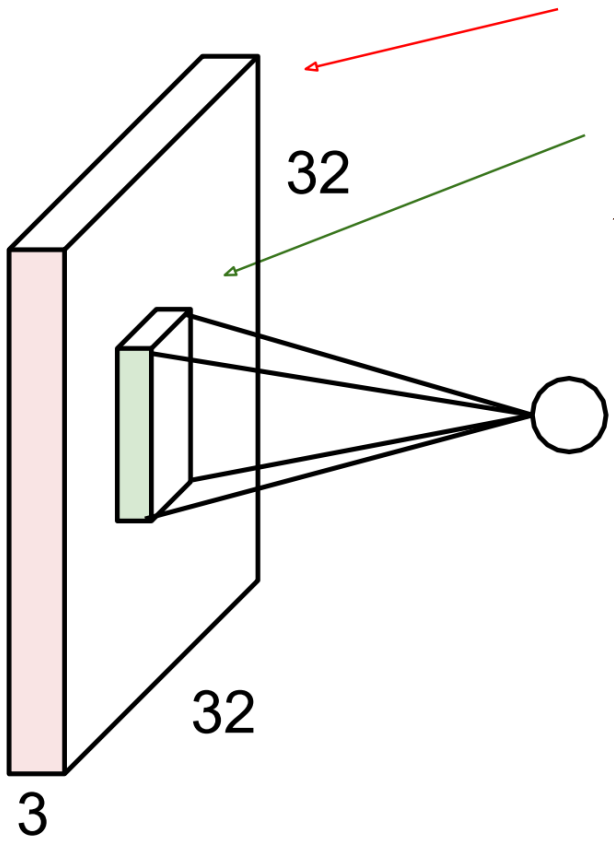
CONVOLUTION LAYER

consider a second, green filter

32x32x3 image

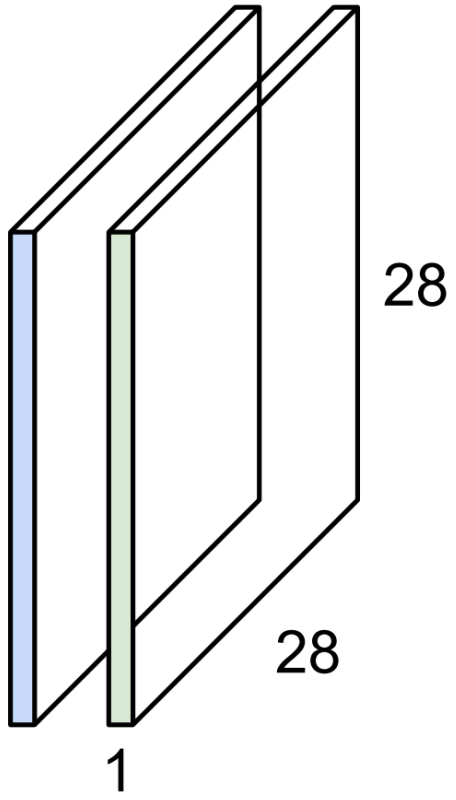
5x5x3 filter

$P = 0, S = 1$



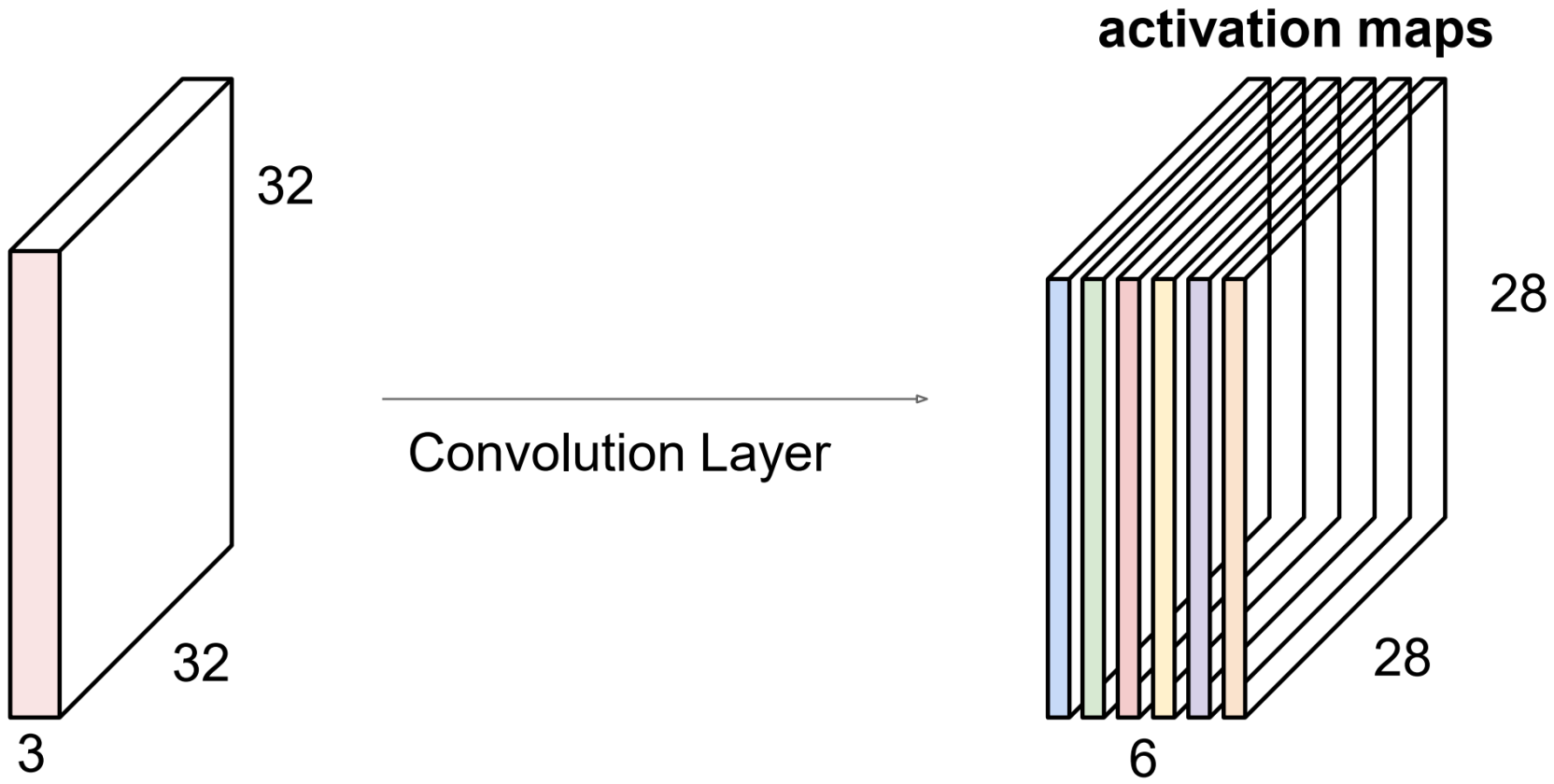
convolve (slide) over all spatial locations

activation maps

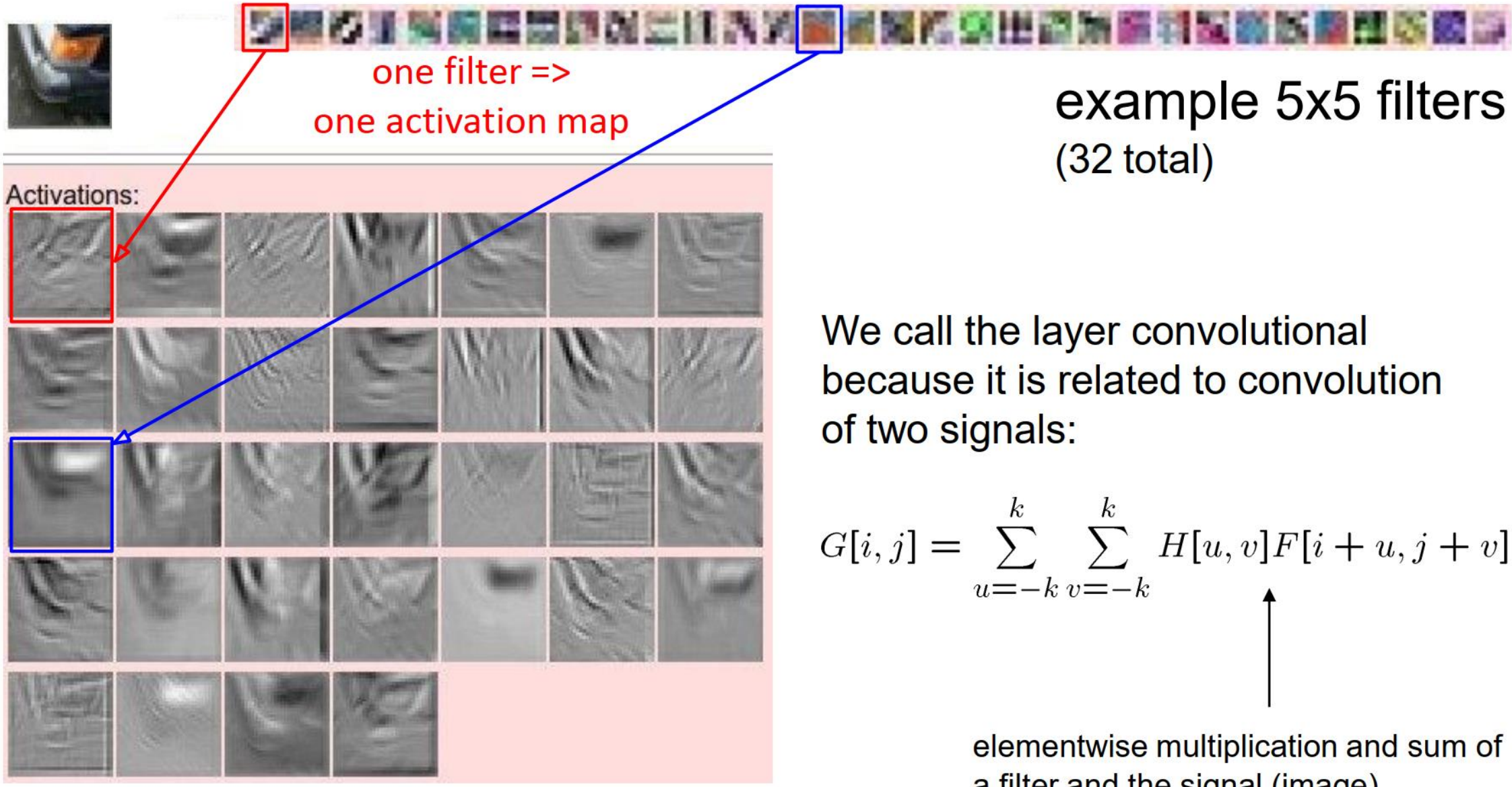


CONVOLUTION LAYER

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



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



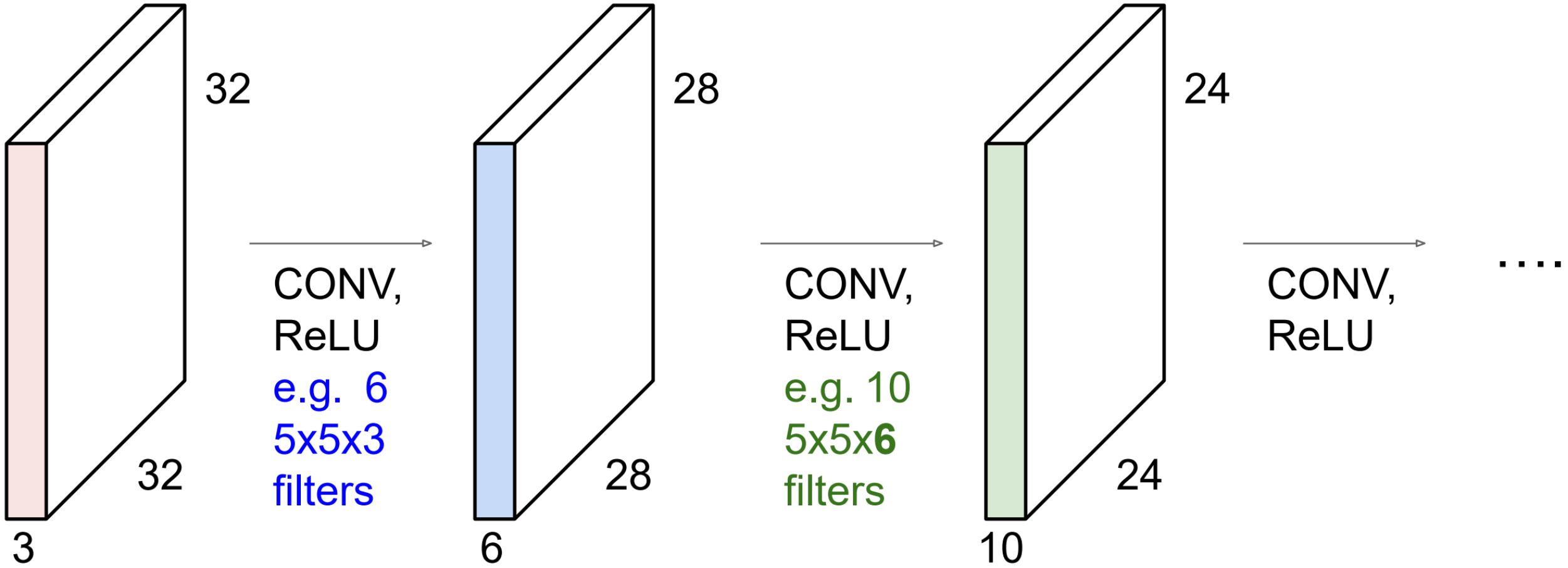
example 5x5 filters
(32 total)

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

$$G[i,j] = \sum_{u=-k}^{k} \sum_{v=-k}^{k} H[u,v]F[i+u,j+v]$$

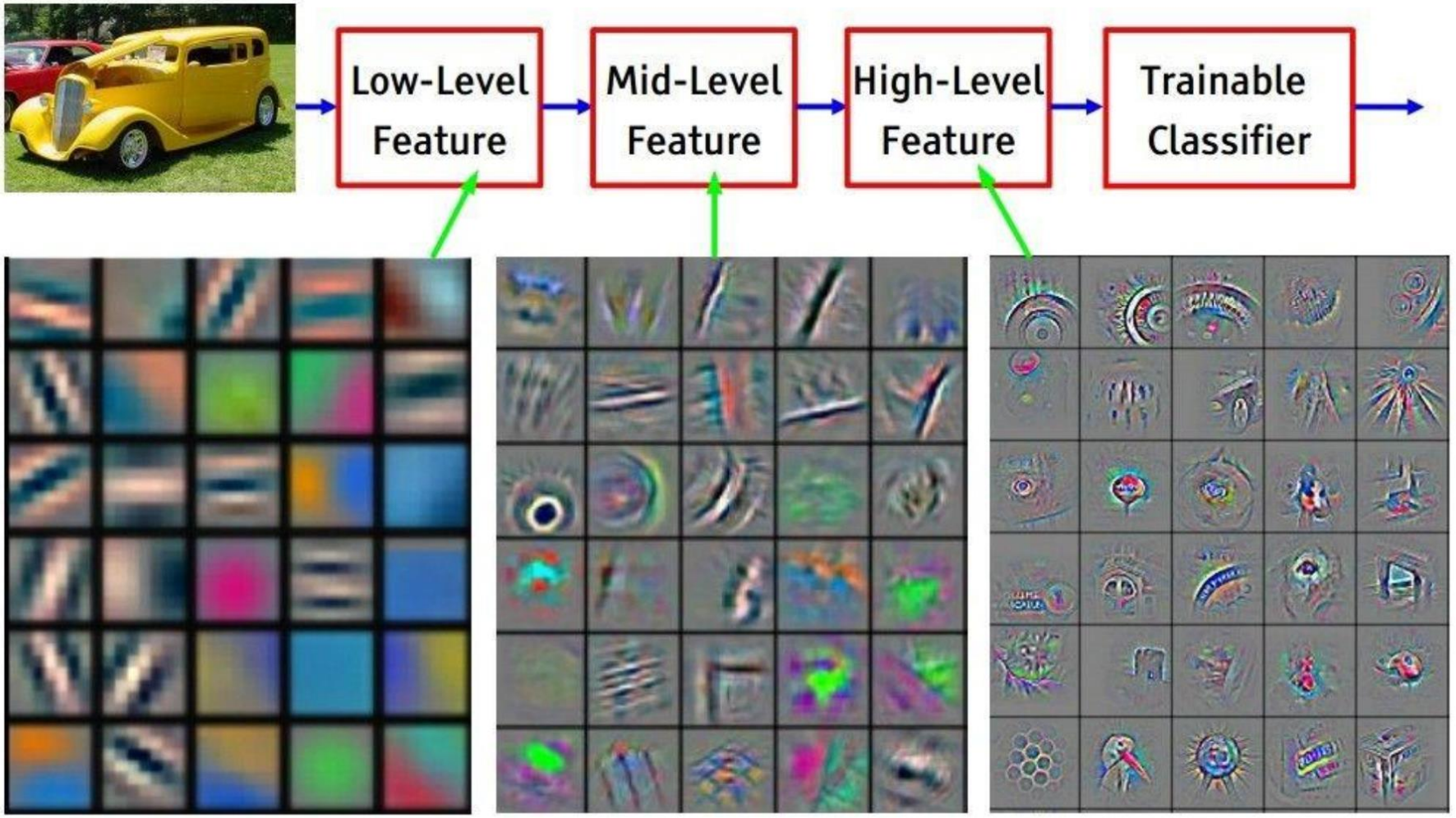
elementwise multiplication and sum of a filter and the signal (image)

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



FEATURE VISUALIZATION Preview

[From recent Yann LeCun slides]



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

SUMMARY OF CONVOLUTION LAYER

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

TRAINING OVERVIEW

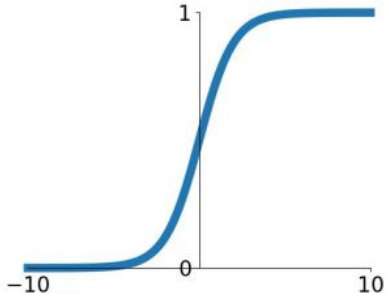
1. **One time setup**
   activation functions, preprocessing, weight initialization, regularization, gradient checking
2. **Training dynamics**
   babysitting the learning process, parameter updates, hyperparameter optimization
3. **Evaluation**
   model ensembles, test-time augmentation, transfer learning

Slides credit: Fei-Fei Li, Ranjay Krishna and Danfei Xu

ACTIVATION FUNCTIONS

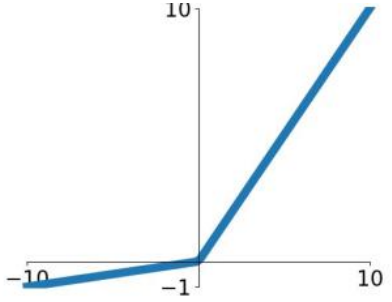
Sigmoid

$$\sigma(x) = \frac{1}{1+e^{-x}}$$



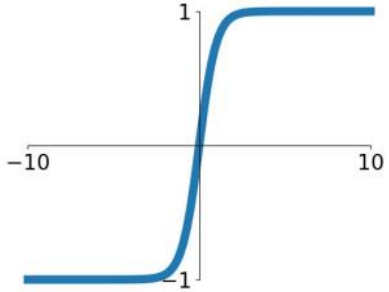
Leaky ReLU

$$\max(0.1x, x)$$



tanh

$$\tanh(x)$$

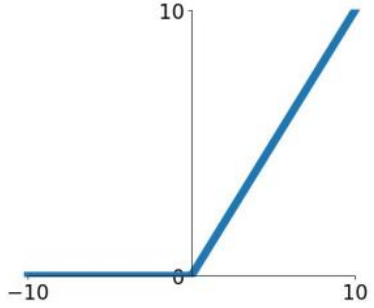


Maxout

$$\max(w_1^T x + b_1, w_2^T x + b_2)$$

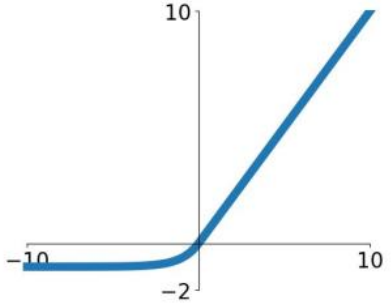
ReLU

$$\max(0, x)$$



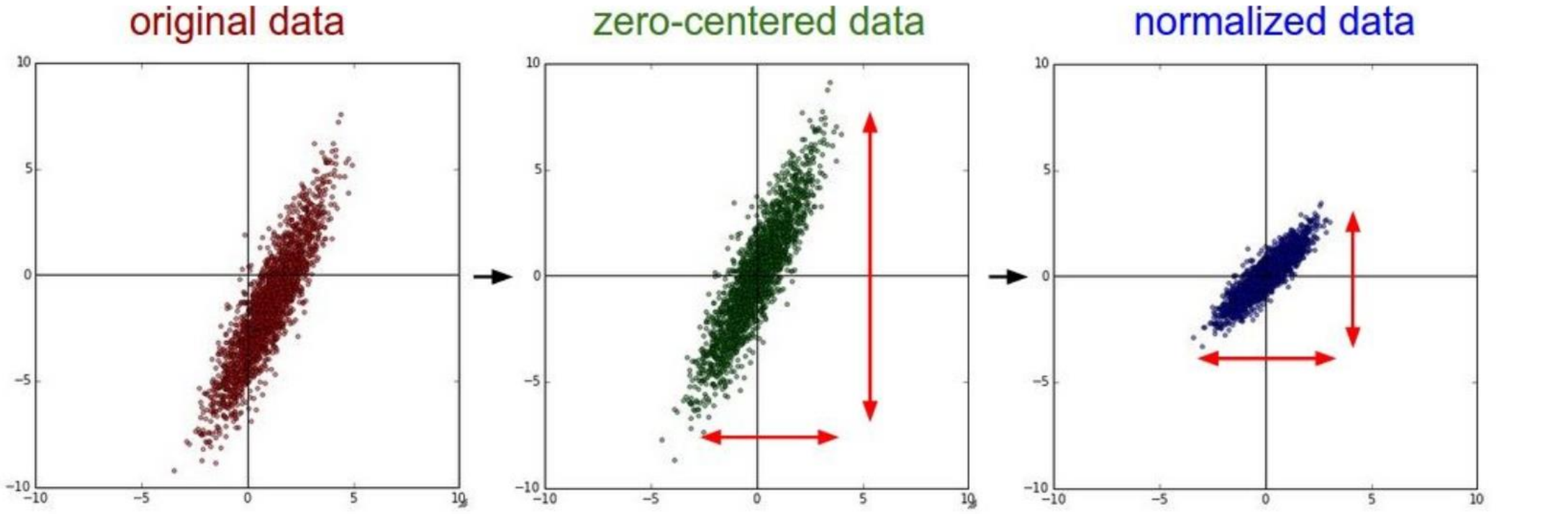
ELU

$$\begin{cases} x & x \geq 0 \\ \alpha(e^x - 1) & x < 0 \end{cases}$$



Slides credit: Fei-Fei Li, Ranjay Krishna and Danfei Xu

DATA PREPROCESSING




```
X -= np.mean(X, axis = 0)
```

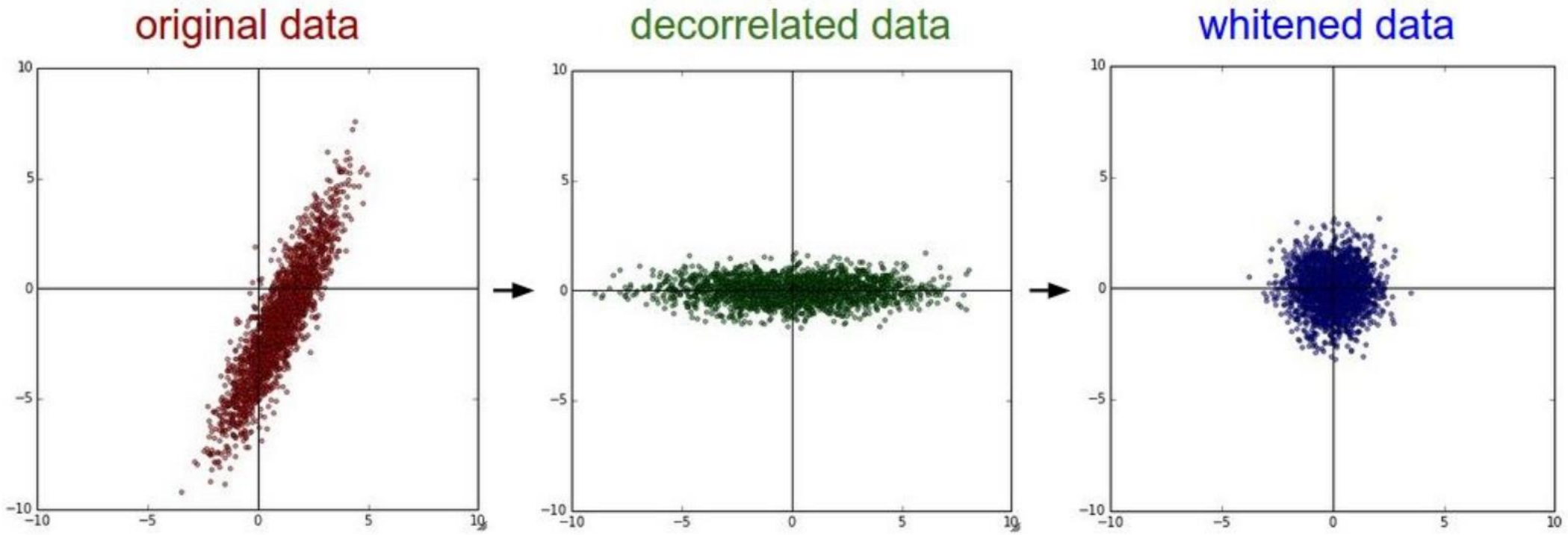
```
X /= np.std(X, axis = 0)
```

(Assume X [NxD] is data matrix, each example in a row)

Slides credit: Fei-Fei Li, Ranjay Krishna and Danfei Xu

DATA PREPROCESSING

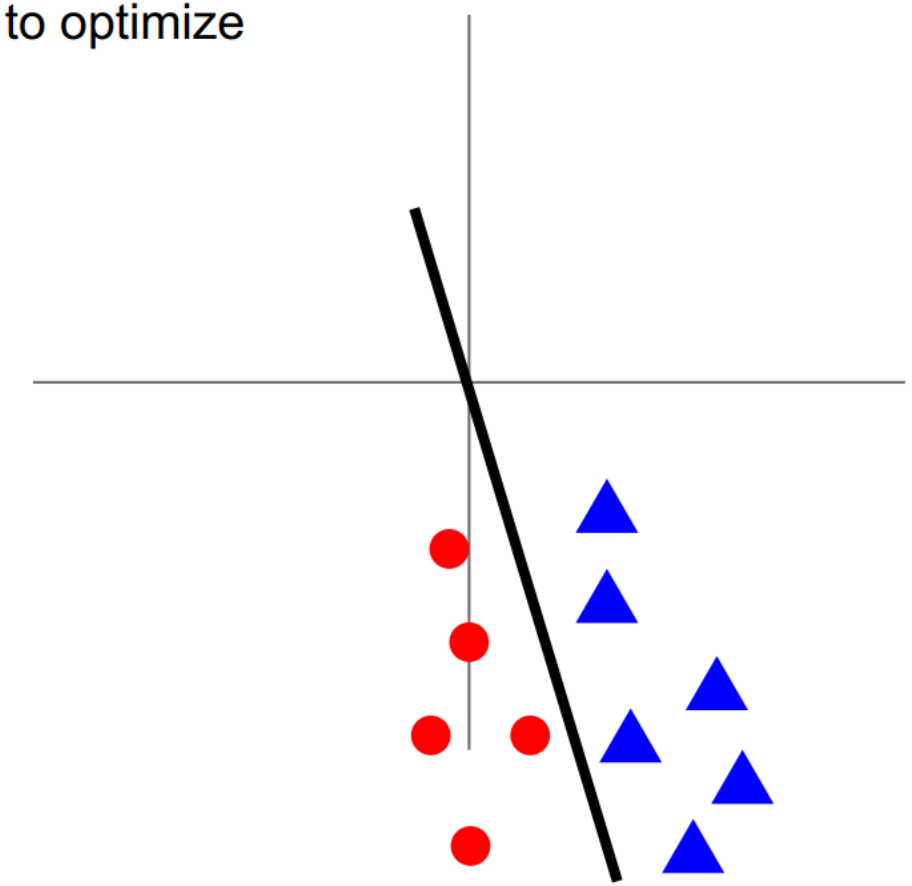
In practice, you may also see **PCA** and **Whitening** of the data



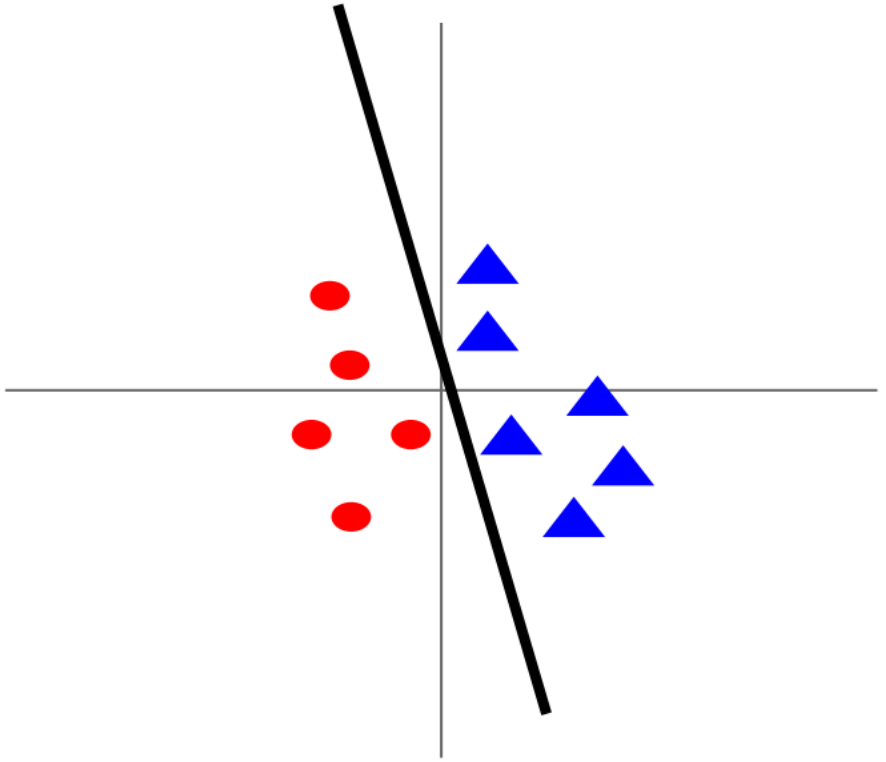
Slides credit: Fei-Fei Li, Ranjay Krishna and Danfei Xu

DATA PREPROCESSING

Before normalization: classification loss very sensitive to changes in weight matrix; hard to optimize



After normalization: less sensitive to small changes in weights; easier to optimize



Slides credit: Fei-Fei Li, Ranjay Krishna and Danfei Xu

WEIGHT INITIALIZATION

- First idea: **Small random numbers**

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

```
W = 0.01 * np.random.randn(Din, Dout)
```

Works ~okay for small networks, but problems with deeper networks.

Slides credit: Fei-Fei Li, Ranjay Krishna and Danfei Xu

Weight Initialization: Activation statistics

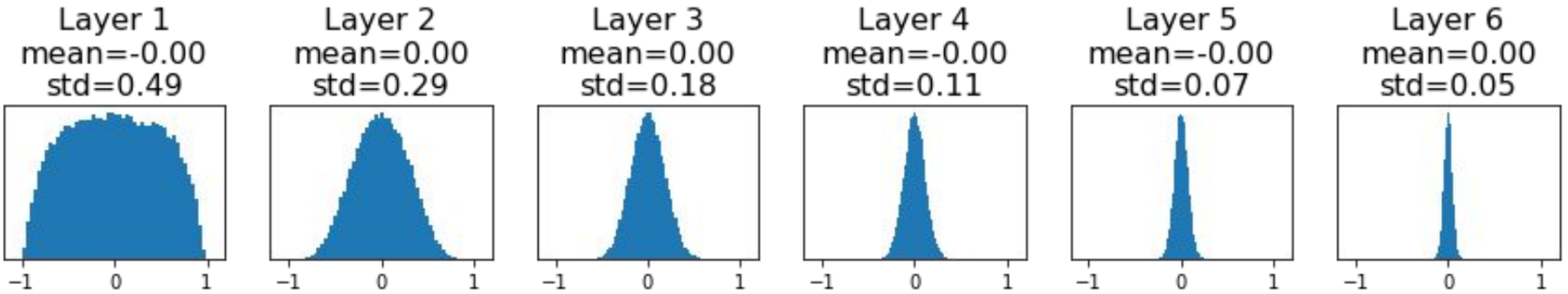
```
dims = [4096] * 7
hs = []
x = np.random.randn(16, dims[0])
for Din, Dout in zip(dims[:-1], dims[1:]):
    W = 0.01 * np.random.randn(Din, Dout)
    x = np.tanh(x.dot(W))
    hs.append(x)
```

Forward pass for a 6-layer net with hidden size 4096

All activations tend to zero for deeper network layers

Q: What do the gradients dL/dW look like?

A: All zero, no learning =(



Slides credit: Fei-Fei Li, Ranjay Krishna and Danfei Xu

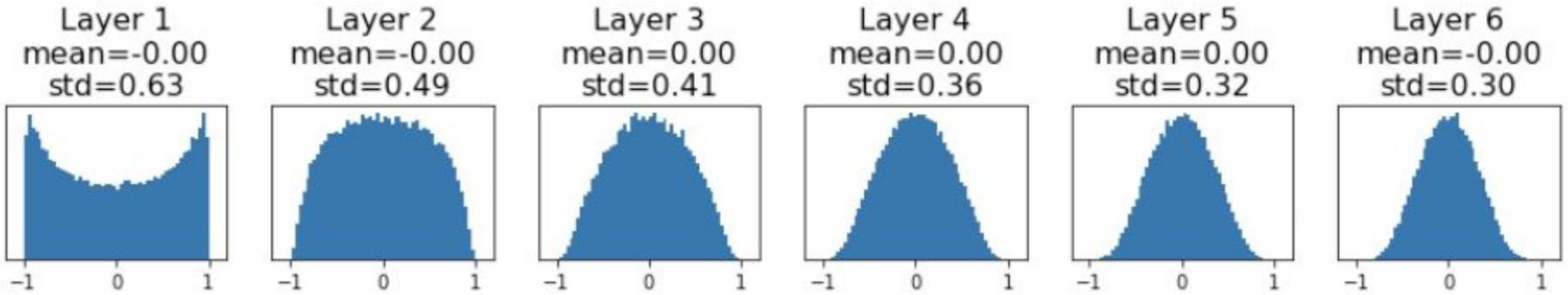
Weight Initialization: “Xavier” Initialization

```
dims = [4096] * 7
hs = []
x = np.random.randn(16, dims[0])
for Din, Dout in zip(dims[:-1], dims[1:]):
    W = np.random.randn(Din, Dout) / np.sqrt(Din)
    x = np.tanh(x.dot(W))
    hs.append(x)
```

“Xavier” initialization:
std = 1/sqrt(Din)

“Just right”: Activations are nicely scaled for all layers!

For conv layers, Din is filter_size^2 * input_channels



Glorot and Bengio, “Understanding the difficulty of training deep feedforward neural networks”, AISTAT 2010

Slides credit: Fei-Fei Li, Ranjay Krishna and Danfei Xu

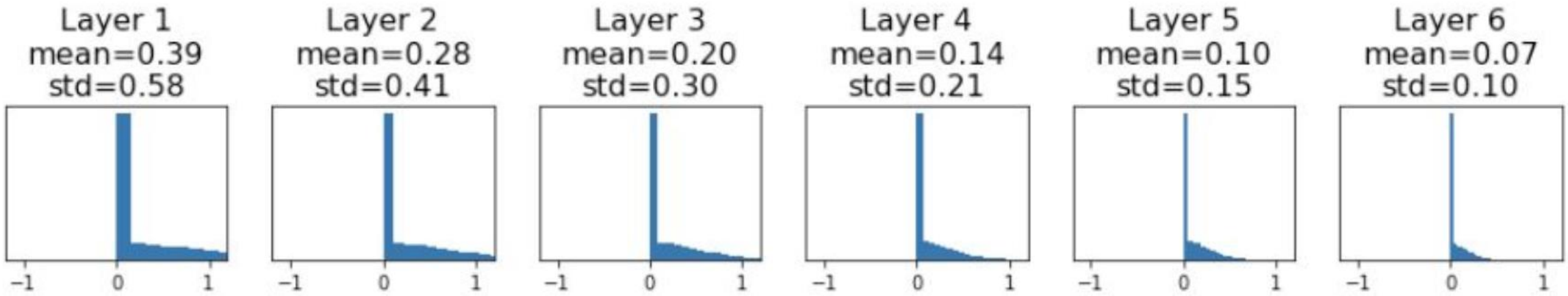
Weight Initialization: What about ReLU?

```
dims = [4096] * 7
hs = []
x = np.random.randn(16, dims[0])
for Din, Dout in zip(dims[:-1], dims[1:]):
    W = np.random.randn(Din, Dout) / np.sqrt(Din)
    x = np.maximum(0, x.dot(W))
    hs.append(x)
```

Change from tanh to ReLU

Xavier assumes zero centered activation function

Activations collapse to zero again, no learning =(



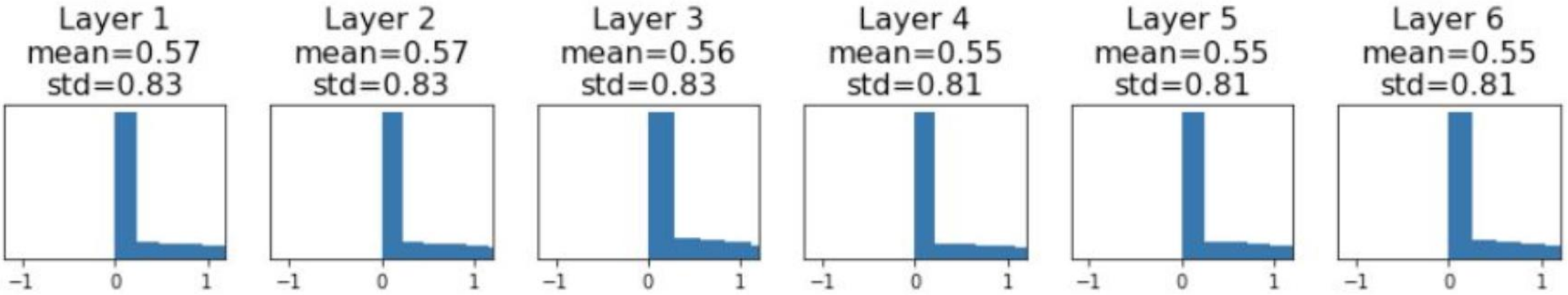
Slides credit: Fei-Fei Li, Ranjay Krishna and Danfei Xu

Weight Initialization: Kaiming / MSRA Initialization

```
dims = [4096] * 7
hs = []
x = np.random.randn(16, dims[0])
for Din, Dout in zip(dims[:-1], dims[1:]):
    W = np.random.randn(Din, Dout) * np.sqrt(2/Din)
    x = np.maximum(0, x.dot(W))
    hs.append(x)
```

ReLU correction: std = sqrt(2 / Din)

"Just right": Activations are nicely scaled for all layers!



He et al, "Delving Deep into Rectifiers: Surpassing Human-Level Performance on ImageNet Classification", ICCV 2015

Slides credit: Fei-Fei Li, Ranjay Krishna and Danfei Xu

BATCH NORMALIZATION

consider a batch of activations at some layer. To make each dimension zero-mean unit-variance, apply:

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

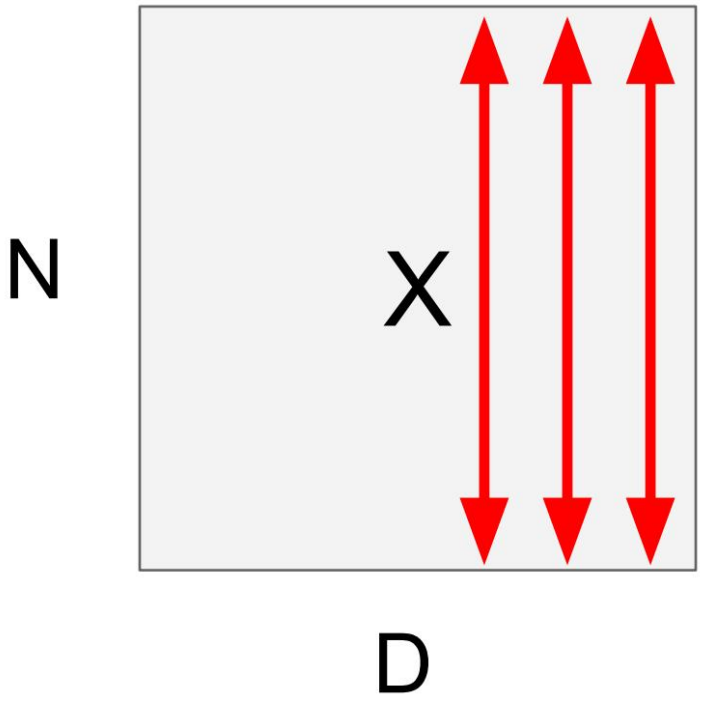
this is a vanilla differentiable function...

Slides credit: Fei-Fei Li, Ranjay Krishna and Danfei Xu

Batch Normalization

[Ioffe and Szegedy, 2015]

Input: $x : N \times D$



$$\mu_j = \frac{1}{N}\sum_{i=1}^{N} x_{i,j}$$

Per-channel mean, shape is D

$$\sigma_j^2 = \frac{1}{N}\sum_{i=1}^{N}(x_{i,j} - \mu_j)^2$$

Per-channel var, shape is D

$$\hat{x}_{i,j} = \frac{x_{i,j} - \mu_j}{\sqrt{\sigma_j^2 + \varepsilon}}$$

Normalized x, Shape is N x D

Slides credit: Fei-Fei Li, Ranjay Krishna and Danfei Xu

Batch Normalization: Test-Time

Input: $x : N \times D$

Learnable scale and shift parameters:

$$\gamma, \beta : D$$

During testing batchnorm becomes a linear operator!
Can be fused with the previous fully-connected or conv layer

$$\mu_j = \text{(Running) average of values seen during training}$$

Per-channel mean, shape is D

$$\sigma_j^2 = \text{(Running) average of values seen during training}$$

Per-channel var, shape is D

$$\hat{x}_{i,j} = \frac{x_{i,j} - \mu_j}{\sqrt{\sigma_j^2 + \varepsilon}}$$

Normalized x, Shape is N x D

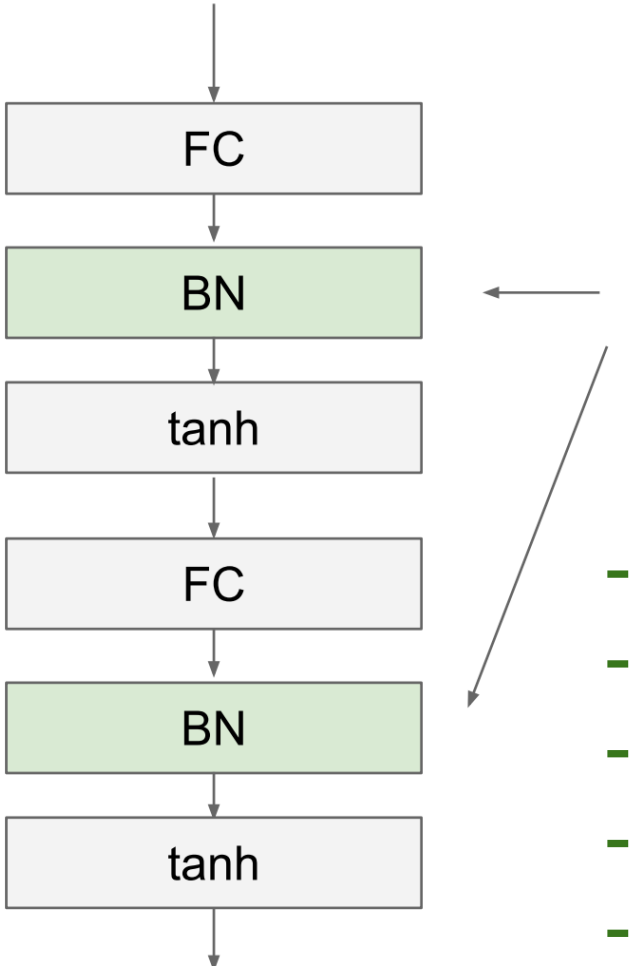
$$y_{i,j} = \gamma_j \hat{x}_{i,j} + \beta_j$$

Output, Shape is N x D

Slides credit: Fei-Fei Li, Ranjay Krishna and Danfei Xu

Batch Normalization

[Ioffe and Szegedy, 2015]



Usually inserted after Fully Connected or Convolutional layers, and before nonlinearity.

- Makes deep networks **much** easier to train!
- Improves gradient flow
- Allows higher learning rates, faster convergence
- Networks become more robust to initialization
- Acts as regularization during training

Slides credit: Fei-Fei Li, Ranjay Krishna and Danfei Xu

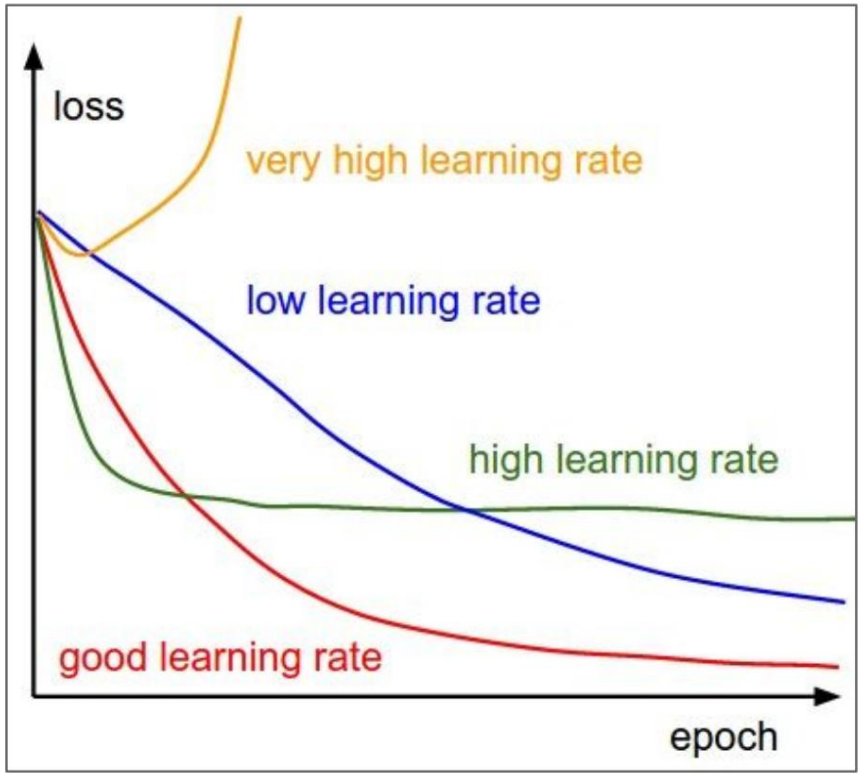
TRAINING AND TESTING DYNAMICS

- Improve your training error:
  - (Fancier) Optimizers
  - Learning rate schedules
- Improve your test error:
  - Regularization
  - Choosing Hyperparameters

Slides credit: Fei-Fei Li, Ranjay Krishna and Danfei Xu

TRAINING ERRORS

SGD, SGD+Momentum, Adagrad, RMSProp, Adam all have **learning rate** as a hyperparameter.

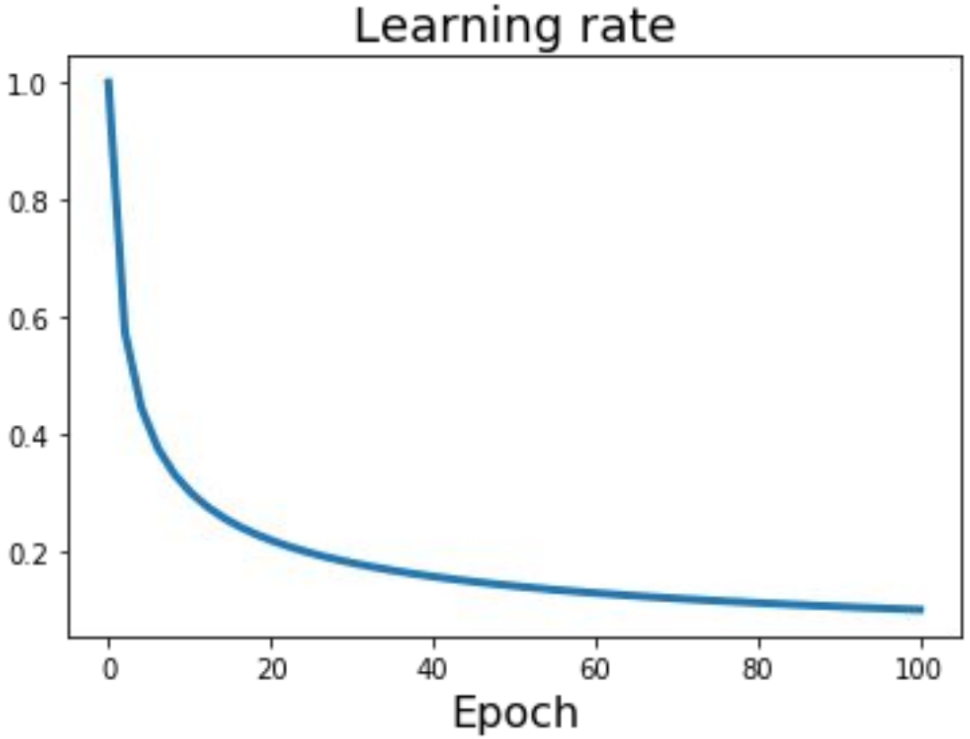


Q: Which one of these learning rates is best to use?

A: In reality, all of these are good learning rates.

Slides credit: Fei-Fei Li, Ranjay Krishna and Danfei Xu

TRAINING ERRORS – LEARNING RATE DECAY



Step: Reduce learning rate at a few fixed points. E.g. for ResNets, multiply LR by 0.1 after epochs 30, 60, and 90.

Cosine: $\alpha_t = \frac{1}{2}\alpha_0 \left(1 + \cos(t\pi/T)\right)$

Linear: $\alpha_t = \alpha_0(1 - t/T)$

Inverse sqrt: $\alpha_t = \alpha_0/\sqrt{t}$

α_0 : Initial learning rate
α_t : Learning rate at epoch t
T : Total number of epochs

Vaswani et al, "Attention is all you need", NIPS 2017

Slides credit: Fei-Fei Li, Ranjay Krishna and Danfei Xu

TESTING ERRORS

Regularization: Add term to loss

$$L = \frac{1}{N}\sum_{i=1}^{N}\sum_{j\neq y_i}\max(0, f(x_i; W)_j - f(x_i; W)_{y_i} + 1) + \boxed{\lambda R(W)}$$

In common use:

L2 regularization $R(W) = \sum_k \sum_l W_{k,l}^2$ (Weight decay)

L1 regularization $R(W) = \sum_k \sum_l |W_{k,l}|$

Elastic net (L1 + L2) $R(W) = \sum_k \sum_l \beta W_{k,l}^2 + |W_{k,l}|$

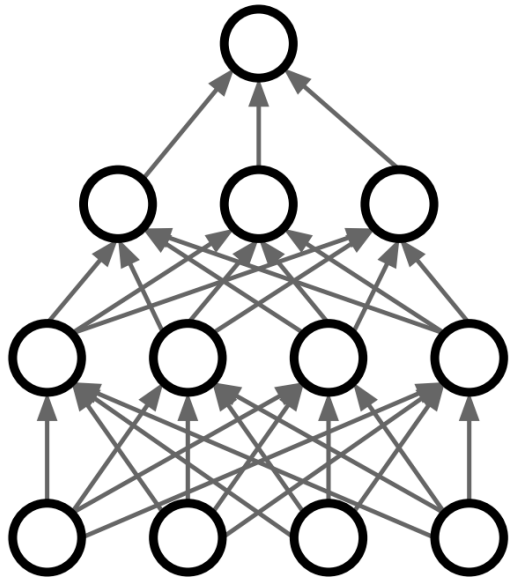
Slides credit: Fei-Fei Li, Ranjay Krishna and Danfei Xu

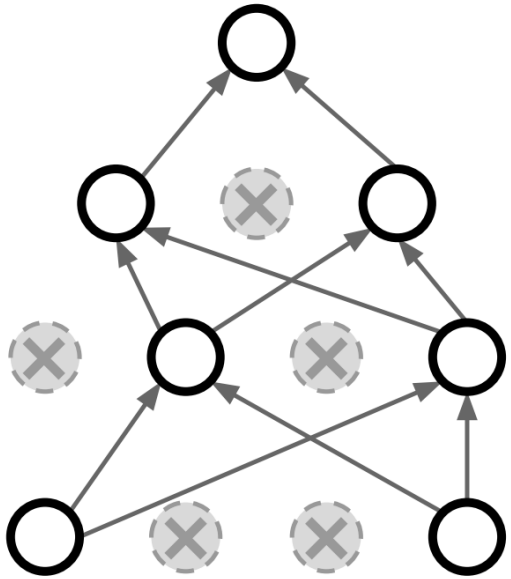
TESTING ERRORS

Regularization: Dropout

In each forward pass, randomly set some neurons to zero

Probability of dropping is a hyperparameter; 0.5 is common





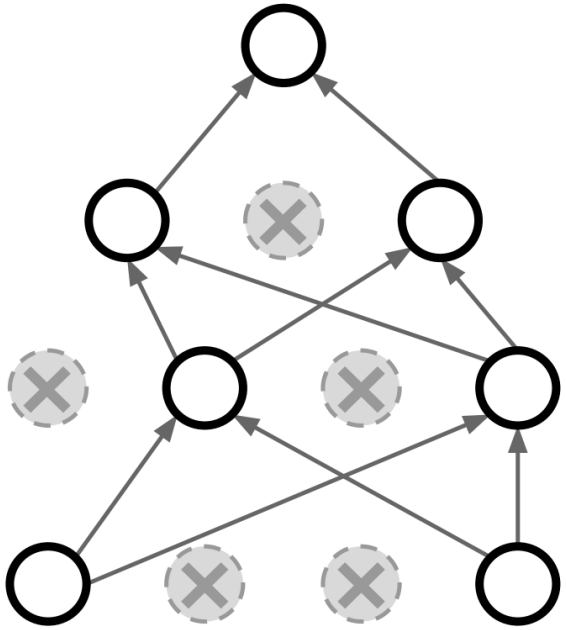
Srivastava et al, “Dropout: A simple way to prevent neural networks from overfitting”, JMLR 2014

Slides credit: Fei-Fei Li, Ranjay Krishna and Danfei Xu

TESTING ERRORS

Regularization: Dropout

How can this possibly be a good idea?



Forces the network to have a redundant representation;
Prevents co-adaptation of features

Slides credit: Fei-Fei Li, Ranjay Krishna and Danfei Xu

Data Augmentation

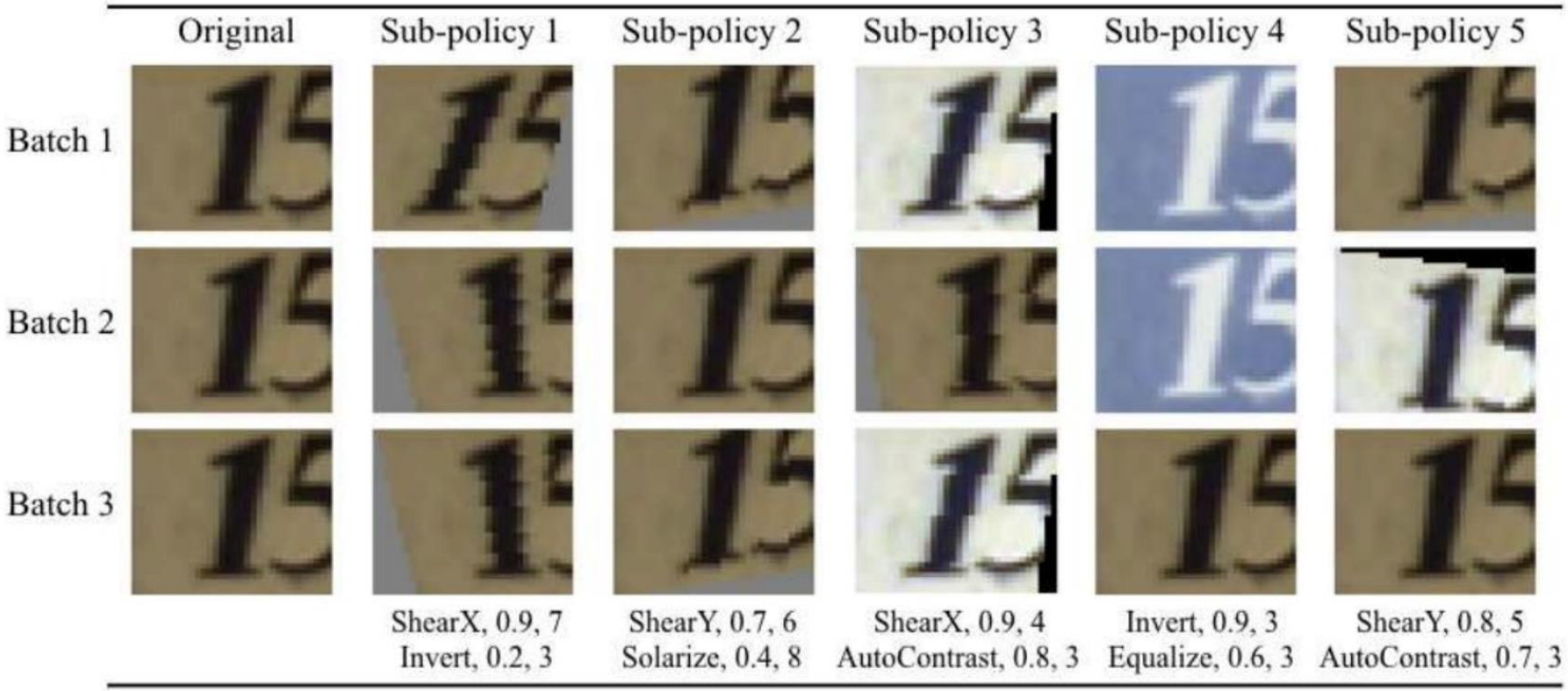
Get creative for your problem!

Examples of data augmentations:

- translation
- rotation
- stretching
- shearing,
- lens distortions, … (go crazy)

Slides credit: Fei-Fei Li, Ranjay Krishna and Danfei Xu

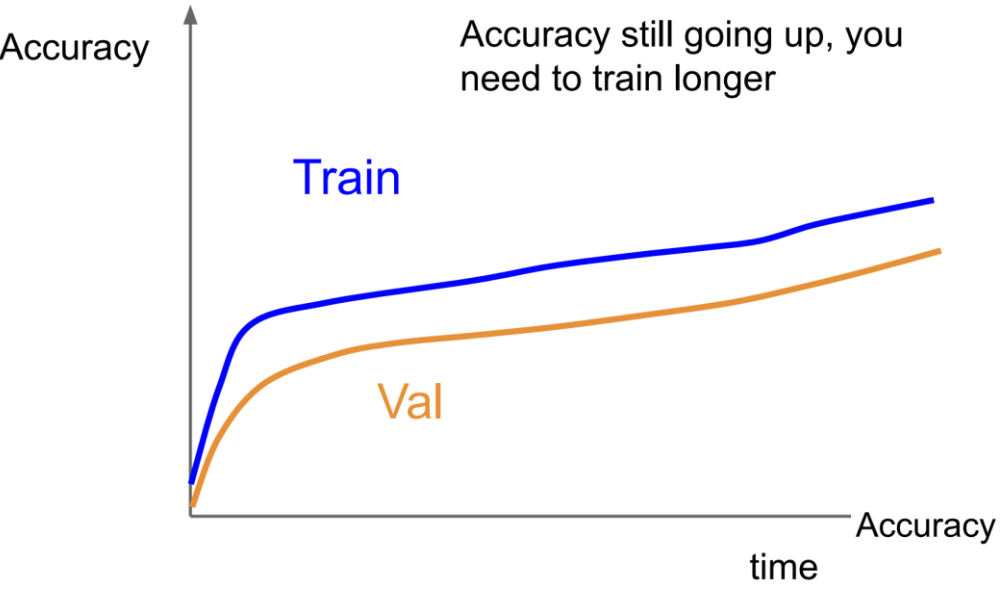
Automatic Data Augmentation

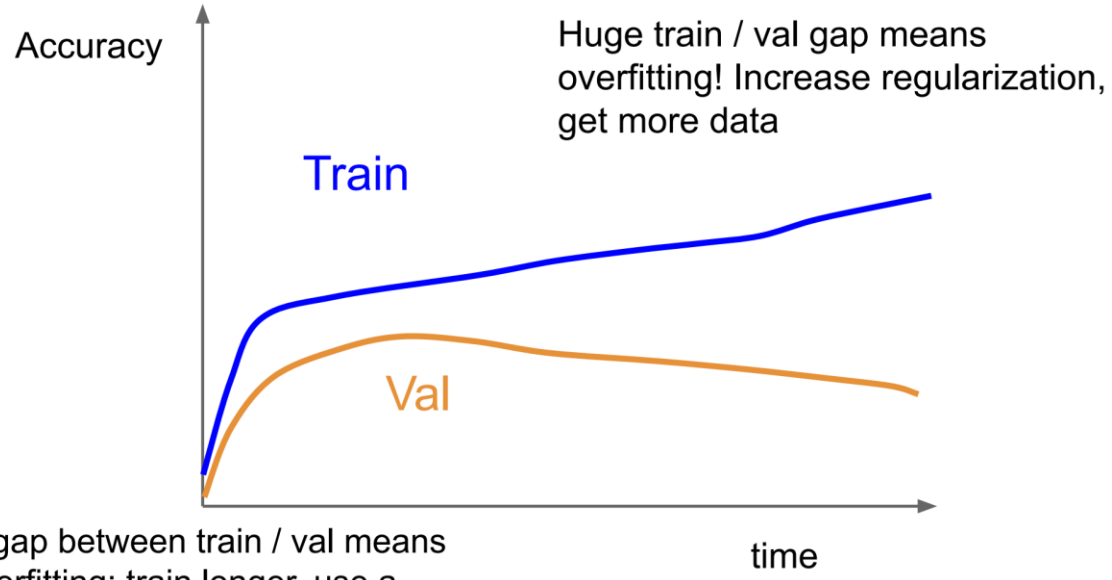


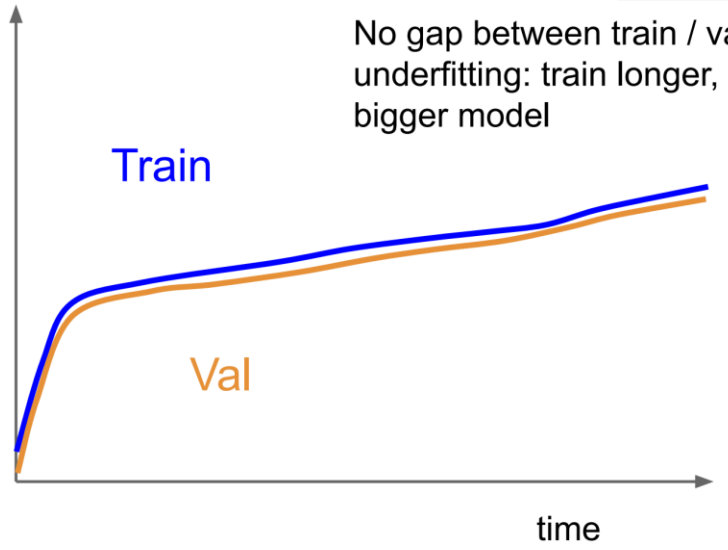
Cubuk et al., "AutoAugment: Learning Augmentation Strategies from Data", CVPR 2019

Slides credit: Fei-Fei Li, Ranjay Krishna and Danfei Xu

CHOOSING HYPERPARAMETERS

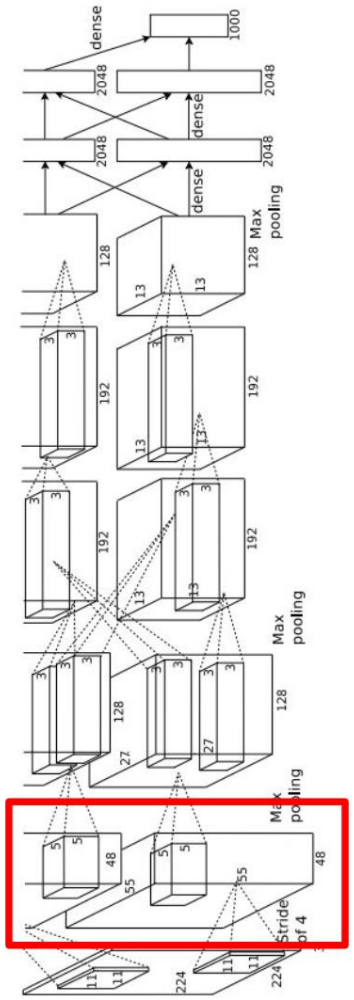


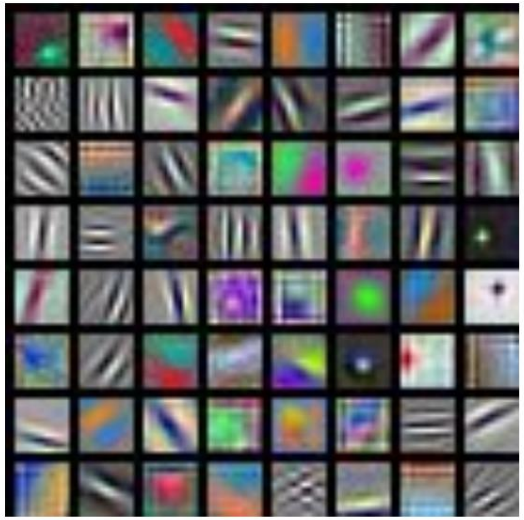




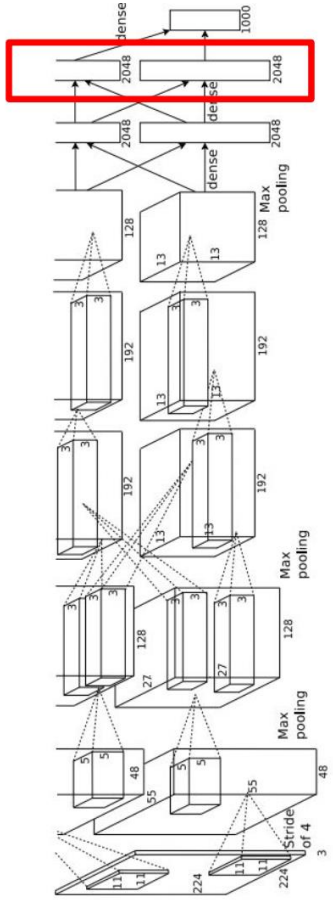
Slides credit: Fei-Fei Li, Ranjay Krishna and Danfei Xu

Transfer Learning with CNNs

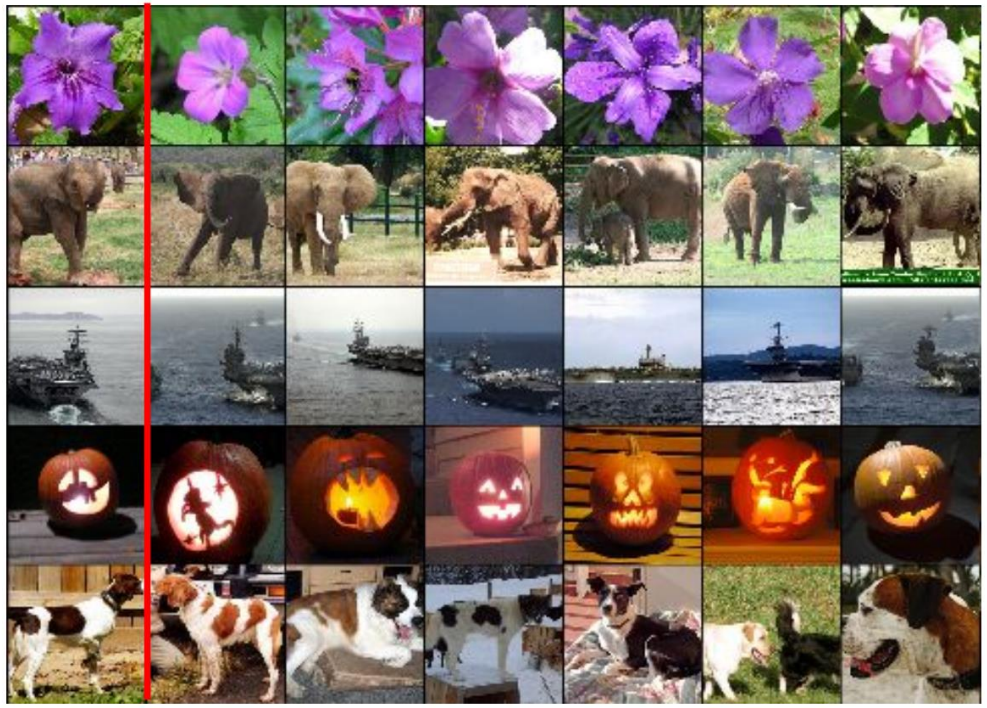




AlexNet:
64 x 3 x 11 x 11



Test image L2 Nearest neighbors in feature space

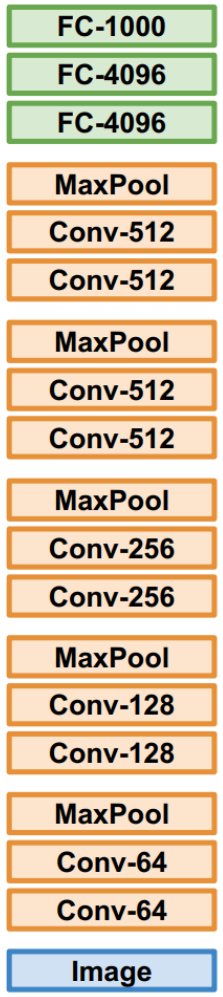


Slides credit: Fei-Fei Li, Ranjay Krishna and Danfei Xu

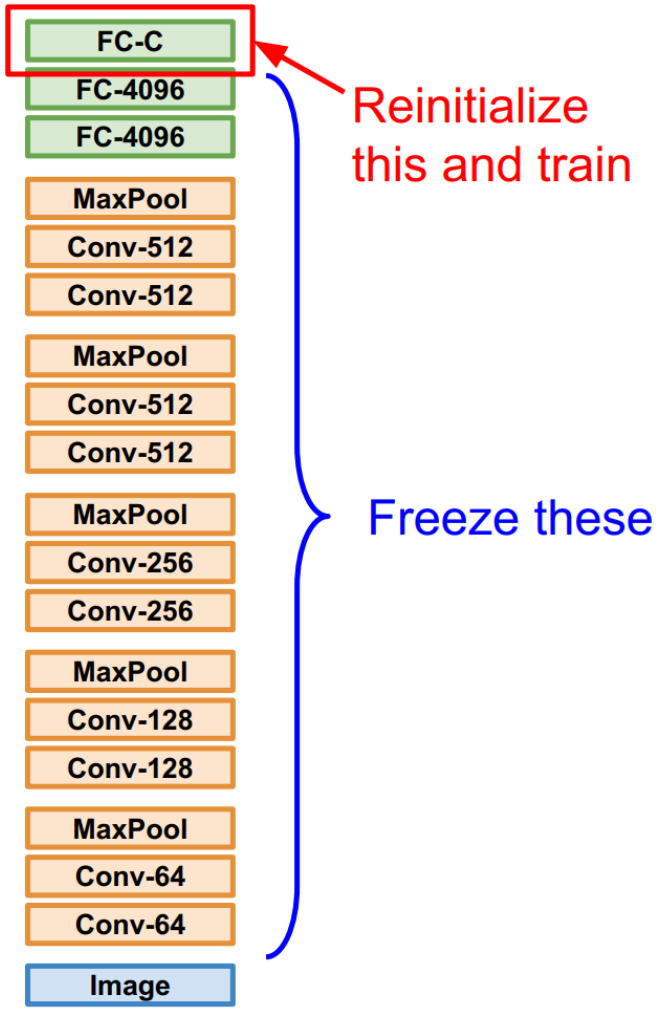
Transfer Learning with CNNs

Donahue et al, "DeCAF: A Deep Convolutional Activation Feature for Generic Visual Recognition", ICML 2014
Razavian et al, "CNN Features Off-the-Shelf: An Astounding Baseline for Recognition", CVPR Workshops 2014

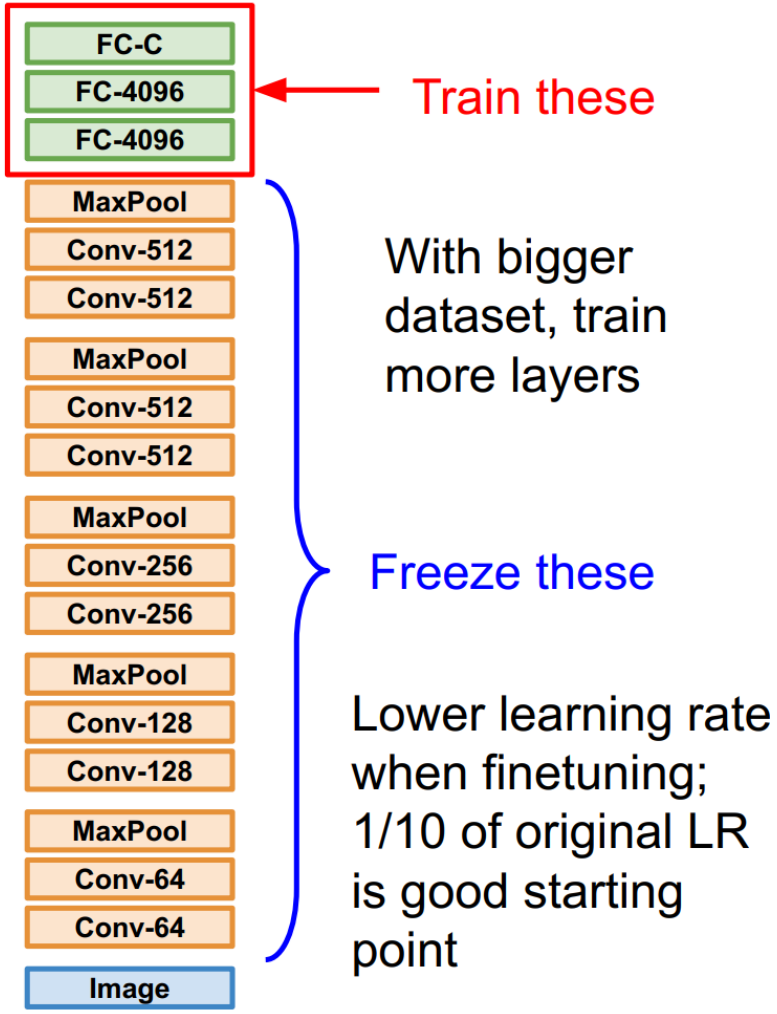
1. Train on Imagenet



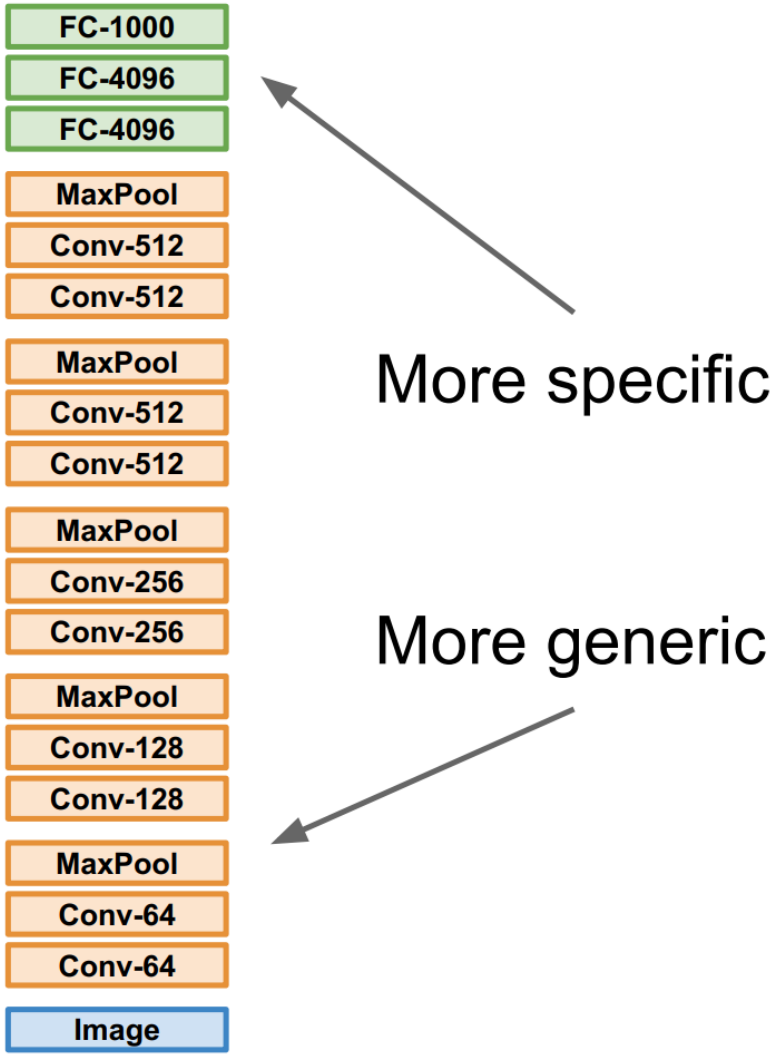
2. Small Dataset (C classes)



3. Bigger dataset



Slides credit: Fei-Fei Li, Ranjay Krishna and Danfei Xu

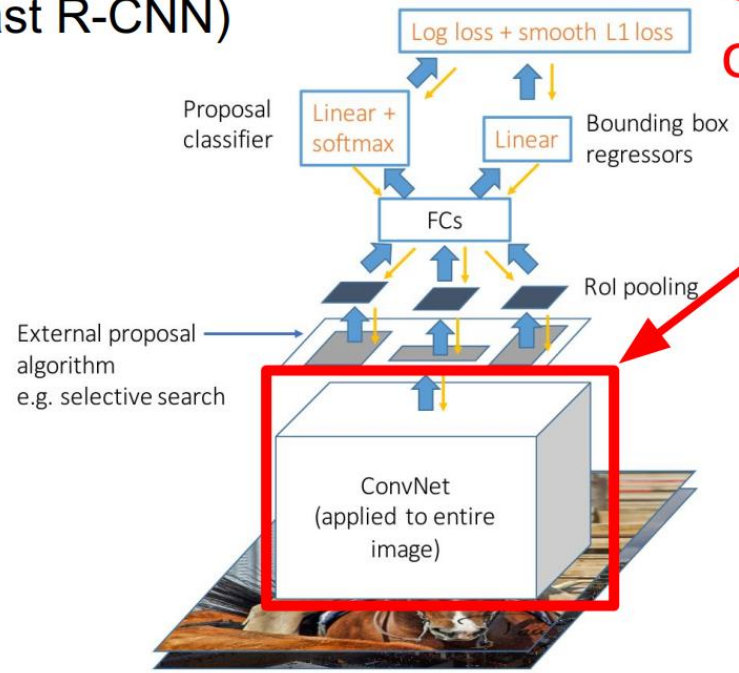


TRAIN \ TEST	**very similar dataset**	**very different dataset**
very little data	Use Linear Classifier on top layer	You're in trouble… Try linear classifier from different stages
quite a lot of data	Finetune a few layers	Finetune a larger number of layers

Slides credit: Fei-Fei Li, Ranjay Krishna and Danfei Xu

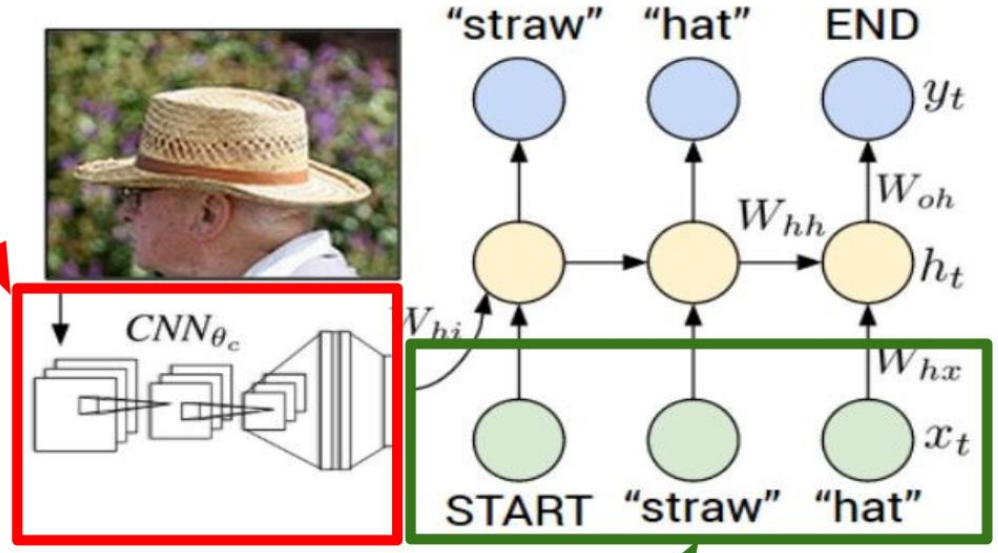
Transfer learning with CNNs is pervasive…
(it's the norm, not an exception)

Object Detection
(Fast R-CNN)



CNN pretrained on ImageNet

Image Captioning: CNN + RNN



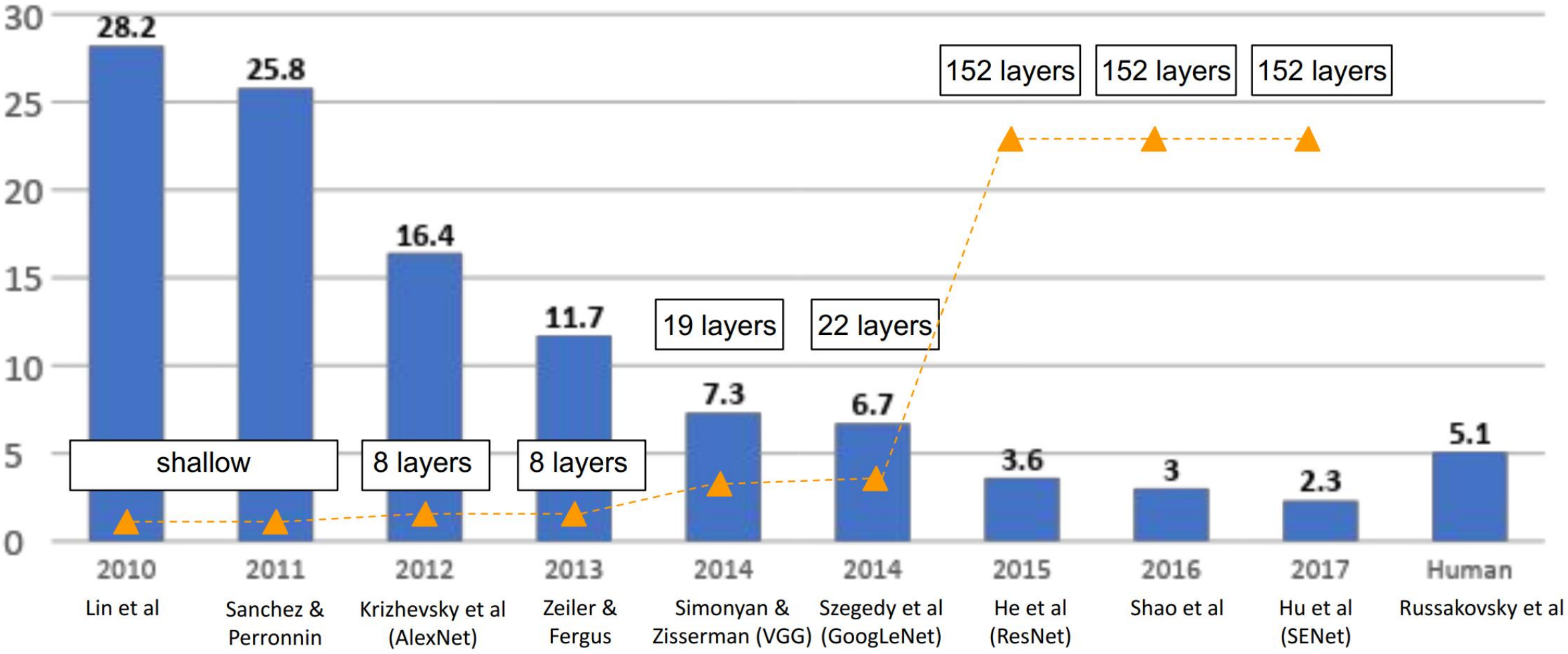
Word vectors pretrained with word2vec

Girshick, "Fast R-CNN", ICCV 2015
Figure copyright Ross Girshick, 2015. Reproduced with permission.

Karpathy and Fei-Fei, "Deep Visual-Semantic Alignments for Generating Image Descriptions", CVPR 2015
Figure copyright IEEE, 2015. Reproduced for educational purposes.

Slides credit: Fei-Fei Li, Ranjay Krishna and Danfei Xu

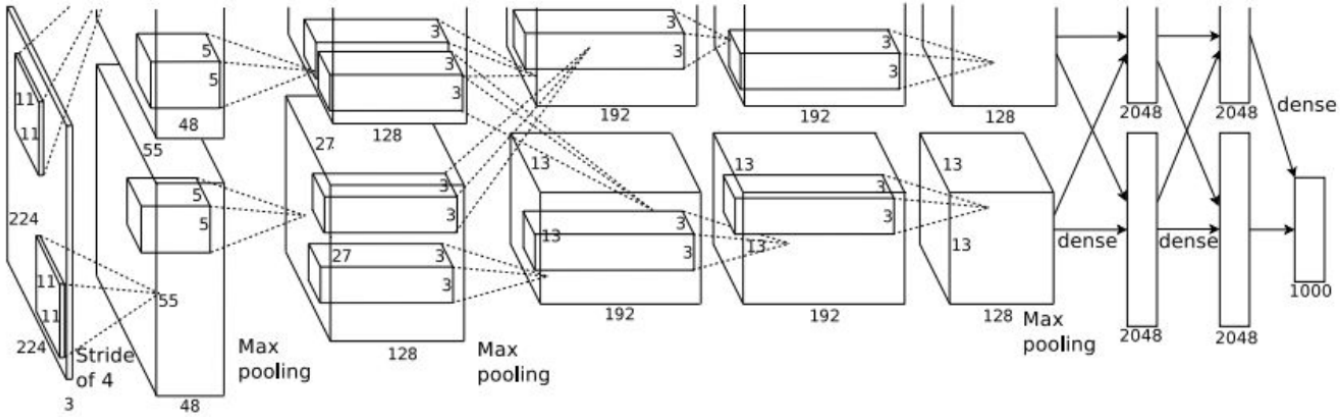
ImageNet Large Scale Visual Recognition Challenge (ILSVRC) winners



Slides credit: Fei-Fei Li, Ranjay Krishna and Danfei Xu

Case Study: AlexNet

[Krizhevsky et al. 2012]



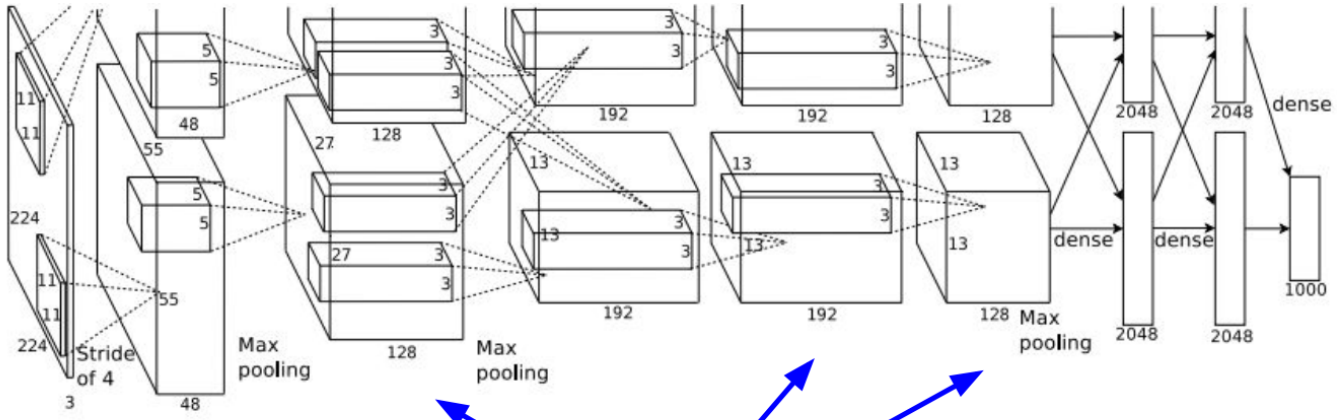
Full (simplified) AlexNet architecture:
[227x227x3] INPUT
[55x55x96] CONV1: 96 11x11 filters at stride 4, pad 0
[27x27x96] MAX POOL1: 3x3 filters at stride 2
[27x27x96] NORM1: Normalization layer
[27x27x256] CONV2: 256 5x5 filters at stride 1, pad 2
[13x13x256] MAX POOL2: 3x3 filters at stride 2
[13x13x256] 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%

Slides credit: Fei-Fei Li, Ranjay Krishna and Danfei Xu

Case Study: AlexNet

[Krizhevsky et al. 2012]



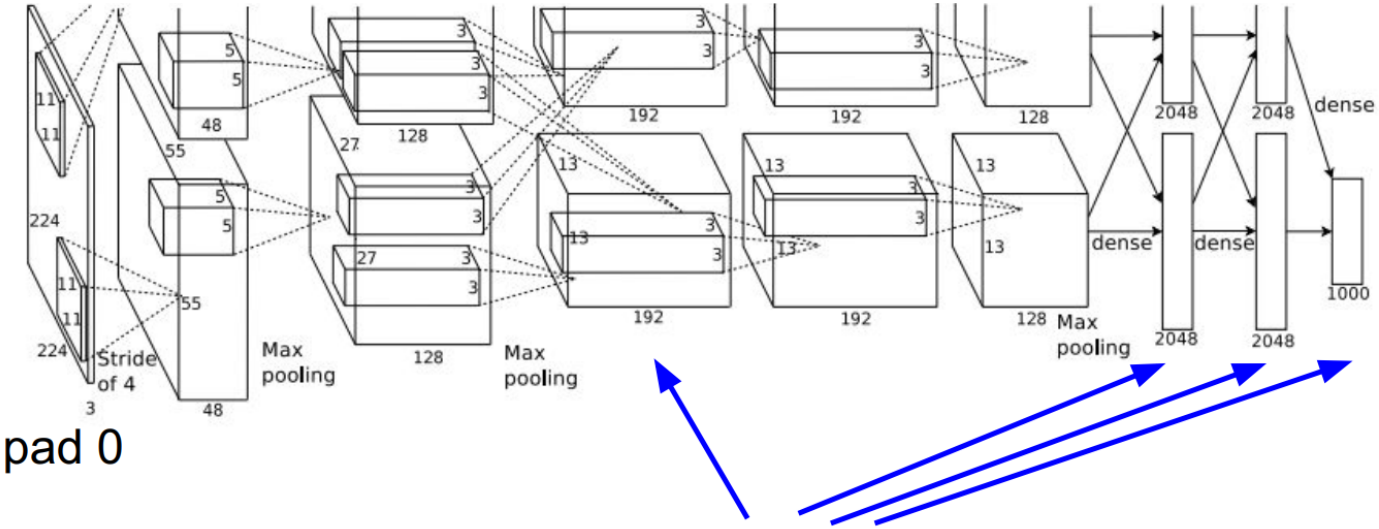
Full (simplified) AlexNet architecture:
[227x227x3] INPUT
[55x55x96] CONV1: 96 11x11 filters at stride 4, pad 0
[27x27x96] MAX POOL1: 3x3 filters at stride 2
[27x27x96] NORM1: Normalization layer
[27x27x256] CONV2: 256 5x5 filters at stride 1, pad 2
[13x13x256] MAX POOL2: 3x3 filters at stride 2
[13x13x256] 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)

CONV1, CONV2, CONV4, CONV5:
Connections only with feature maps on same GPU

Slides credit: Fei-Fei Li, Ranjay Krishna and Danfei Xu

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)

CONV3, FC6, FC7, FC8:
Connections with all feature maps in preceding layer, communication across GPUs

Slides credit: Fei-Fei Li, Ranjay Krishna and Danfei Xu

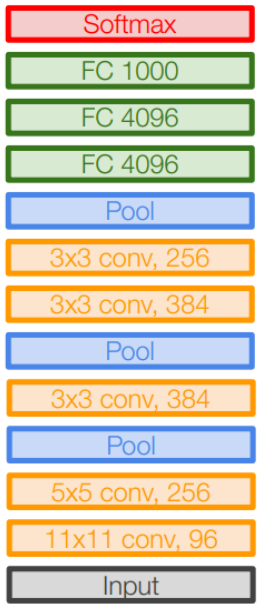
Case Study: VGGNet

[Simonyan and Zisserman, 2014]

Q: Why use smaller filters? (3x3 conv)

Stack of three 3x3 conv (stride 1) layers has same **effective receptive field** as one 7x7 conv layer

Q: What is the effective receptive field of three 3x3 conv (stride 1) layers?



AlexNet

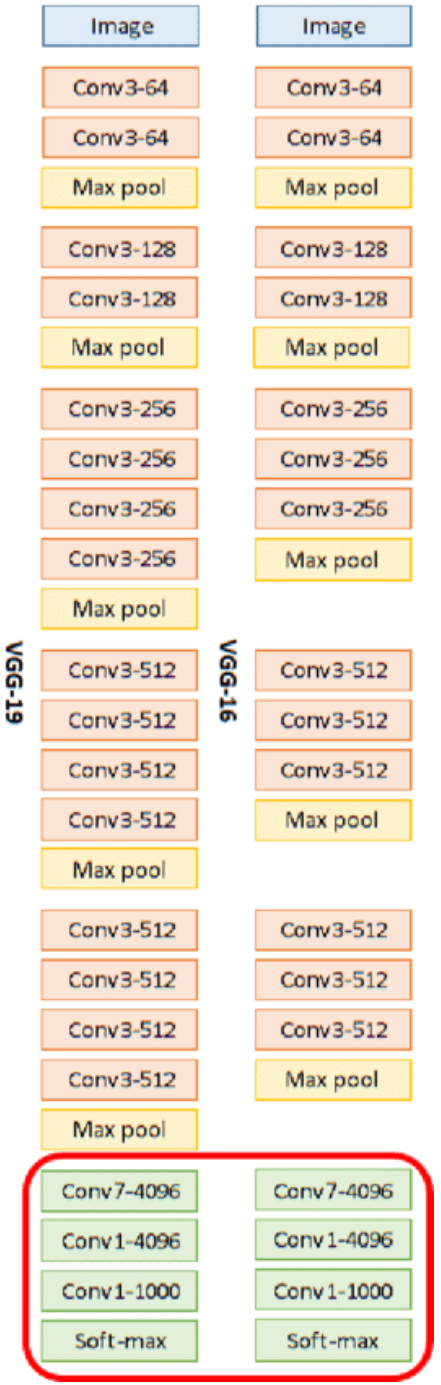
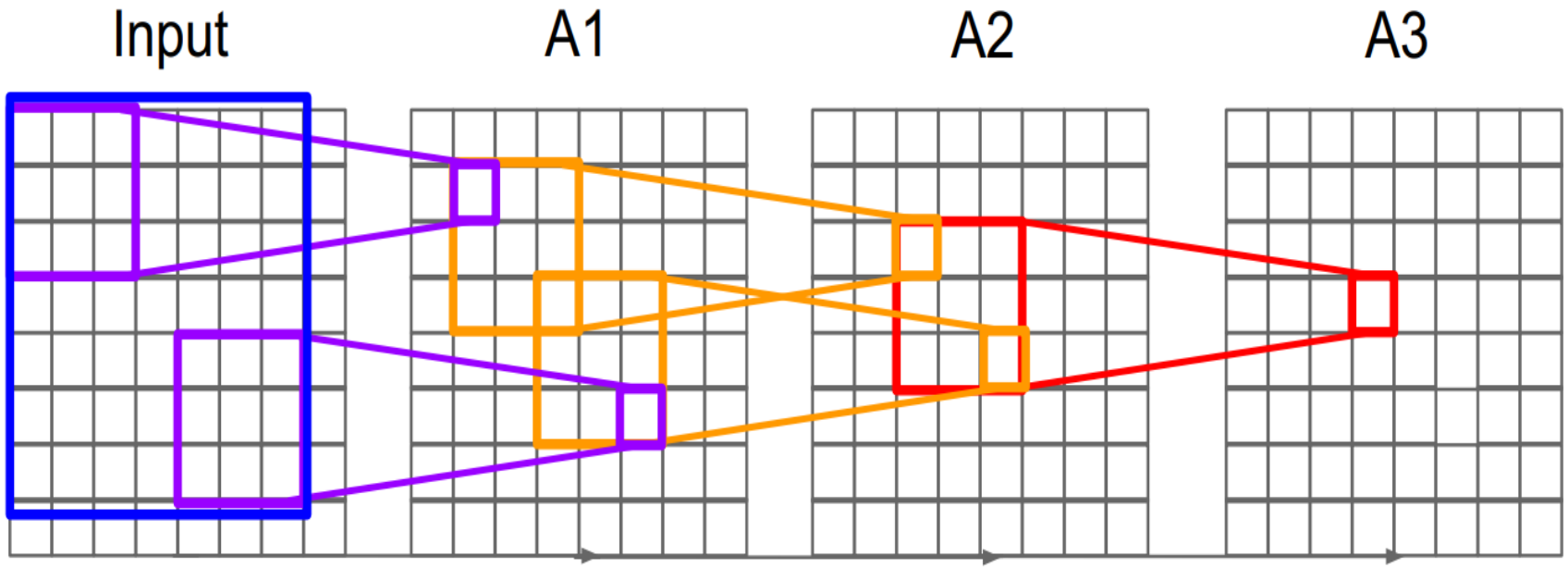


Image courtesy: Automatic Medical Images Segmentation Based on Deep Learning Networks

Slides credit: Fei-Fei Li, Ranjay Krishna and Danfei Xu



But deeper, more non-linearities

And fewer parameters: $3 * (3^2C^2)$ vs. 7^2C^2 for C channels per layer

Slides credit: Fei-Fei Li, Ranjay Krishna and Danfei Xu

VGGNET SUMMARY

- ILSVRC'14 2nd in classification, 1st in localization
- Similar training procedure as Krizhevsky 2012
- No Local Response Normalisation (LRN)
- Use VGG16 or VGG19 (VGG19 only slightly better, more memory)
- Use ensembles for best results
- FC7 features generalize well to other tasks

(not counting biases)

INPUT: [224x224x3] memory: 224*224*3=150K params: 0
CONV3-64: [224x224x64] memory: **224*224*64=3.2M** params: (3*3*3)*64 = 1,728
CONV3-64: [224x224x64] memory: **224*224*64=3.2M** params: (3*3*64)*64 = 36,864
POOL2: [112x112x64] memory: 112*112*64=800K params: 0
CONV3-128: [112x112x128] memory: 112*112*128=1.6M params: (3*3*64)*128 = 73,728
CONV3-128: [112x112x128] memory: 112*112*128=1.6M params: (3*3*128)*128 = 147,456
x128] memory: 56*56*128=400K params: 0
3x56x256] memory: 56*56*256=800K params: (3*3*128)*256 = 294,912
3x56x256] memory: 56*56*256=800K params: (3*3*256)*256 = 589,824
3x56x256] memory: 56*56*256=800K params: (3*3*256)*256 = 589,824
x256] memory: 28*28*256=200K params: 0
3x28x512] memory: 28*28*512=400K params: (3*3*256)*512 = 1,179,648
3x28x512] memory: 28*28*512=400K params: (3*3*512)*512 = 2,359,296
3x28x512] memory: 28*28*512=400K params: (3*3*512)*512 = 2,359,296
x512] memory: 14*14*512=100K params: 0
4x14x512] memory: 14*14*512=100K params: (3*3*512)*512 = 2,359,296
4x14x512] memory: 14*14*512=100K params: (3*3*512)*512 = 2,359,296
4x14x512] memory: 14*14*512=100K params: (3*3*512)*512 = 2,359,296
12] memory: 7*7*512=25K params: 0
memory: 4096 params: 7*7*512*4096 = **102,760,448**
FC: [1x1x4096] memory: 4096 params: 4096*4096 = 16,777,216
FC: [1x1x1000] memory: 1000 params: 4096*1000 = 4,096,000

TOTAL memory: 24M * 4 bytes ~= 96MB / image (only forward! ~*2 for bwd)
TOTAL params: 138M parameters

Note:

Most memory is in early CONV

Most params are in late FC

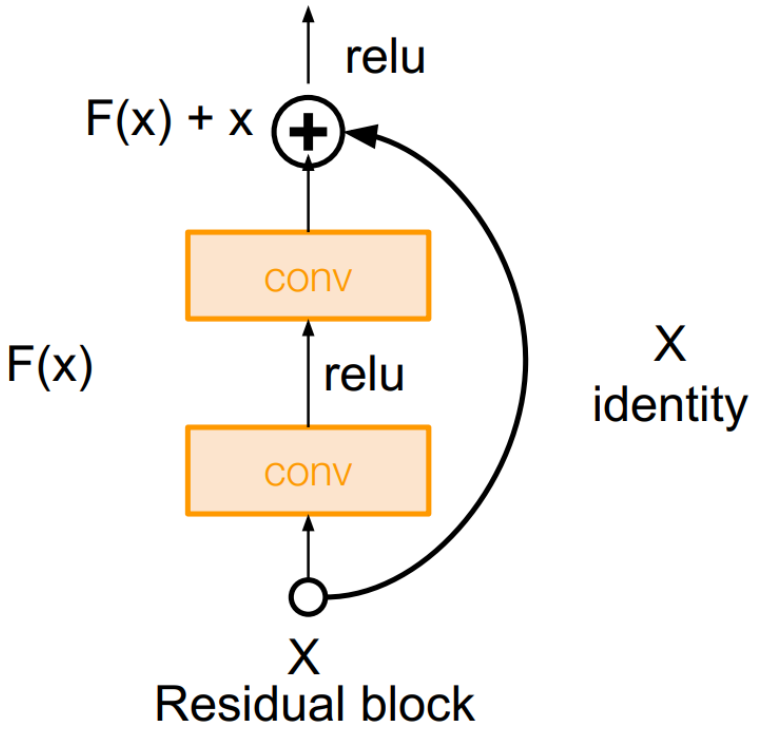
Slides credit: Fei-Fei Li, Ranjay Krishna and Danfei Xu

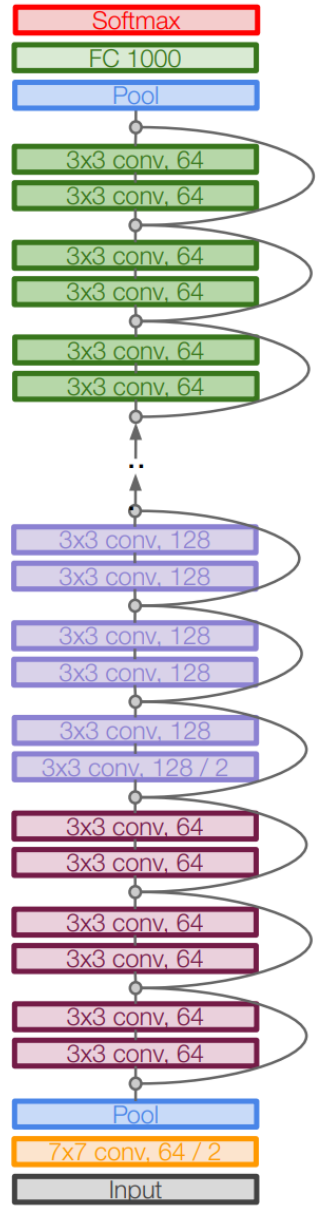
Case Study: ResNet

[He et al., 2015]

Very deep networks using residual connections

- 152-layer model for ImageNet
- ILSVRC'15 classification winner (3.57% top 5 error)
- Swept all classification and detection competitions in ILSVRC'15 and COCO'15!



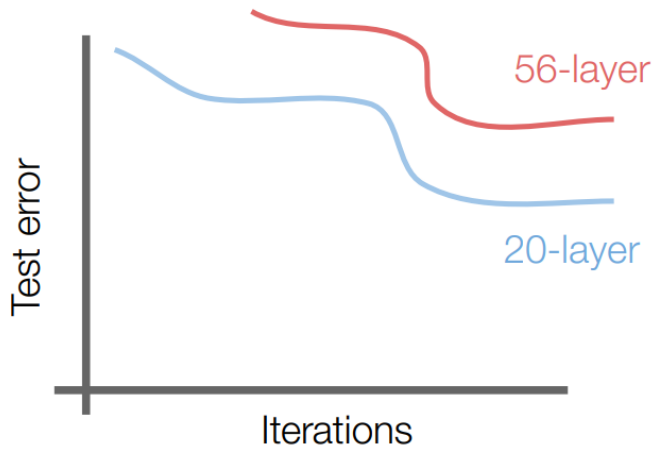


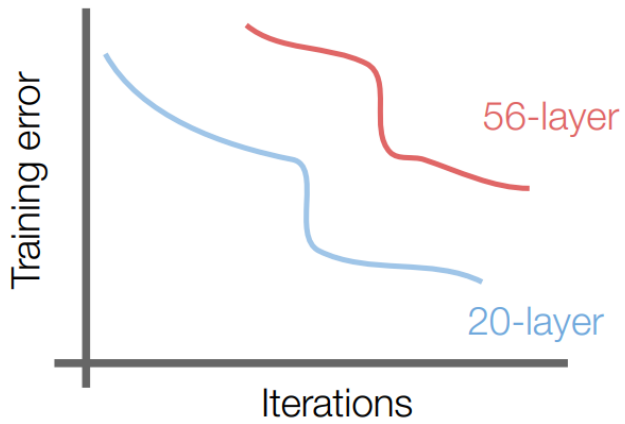
Slides credit: Fei-Fei Li, Ranjay Krishna and Danfei Xu

Case Study: ResNet

[He et al., 2015]

What happens when we continue stacking deeper layers on a “plain” convolutional neural network?





56-layer model performs worse on both test and training error
-> The deeper model performs worse, but it’s not caused by overfitting!

Slides credit: Fei-Fei Li, Ranjay Krishna and Danfei Xu

Case Study: ResNet

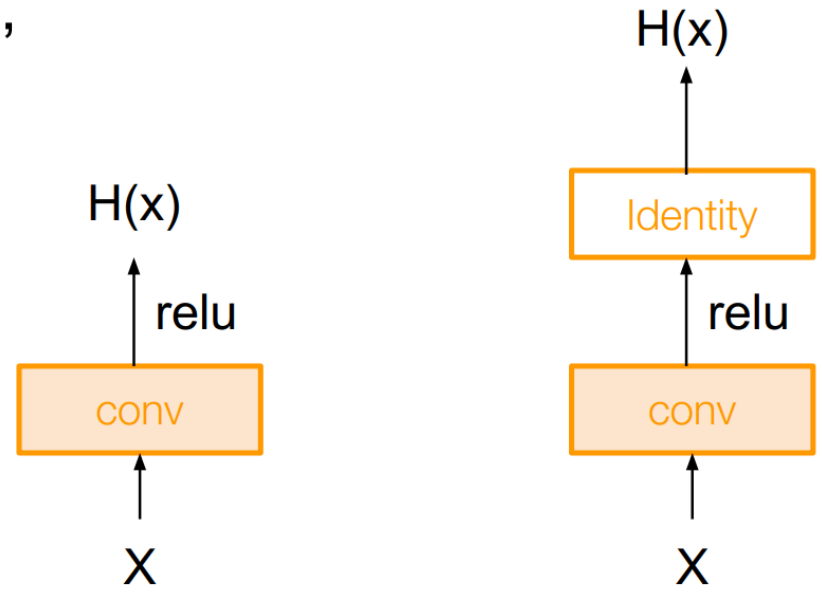
[He et al., 2015]

Fact: Deep models have more representation power (more parameters) than shallower models.

Hypothesis: the problem is an *optimization* problem, deeper models are harder to optimize

What should the deeper model learn to be at least as good as the shallower model?

A solution by construction is copying the learned layers from the shallower model and setting additional layers to identity mapping.

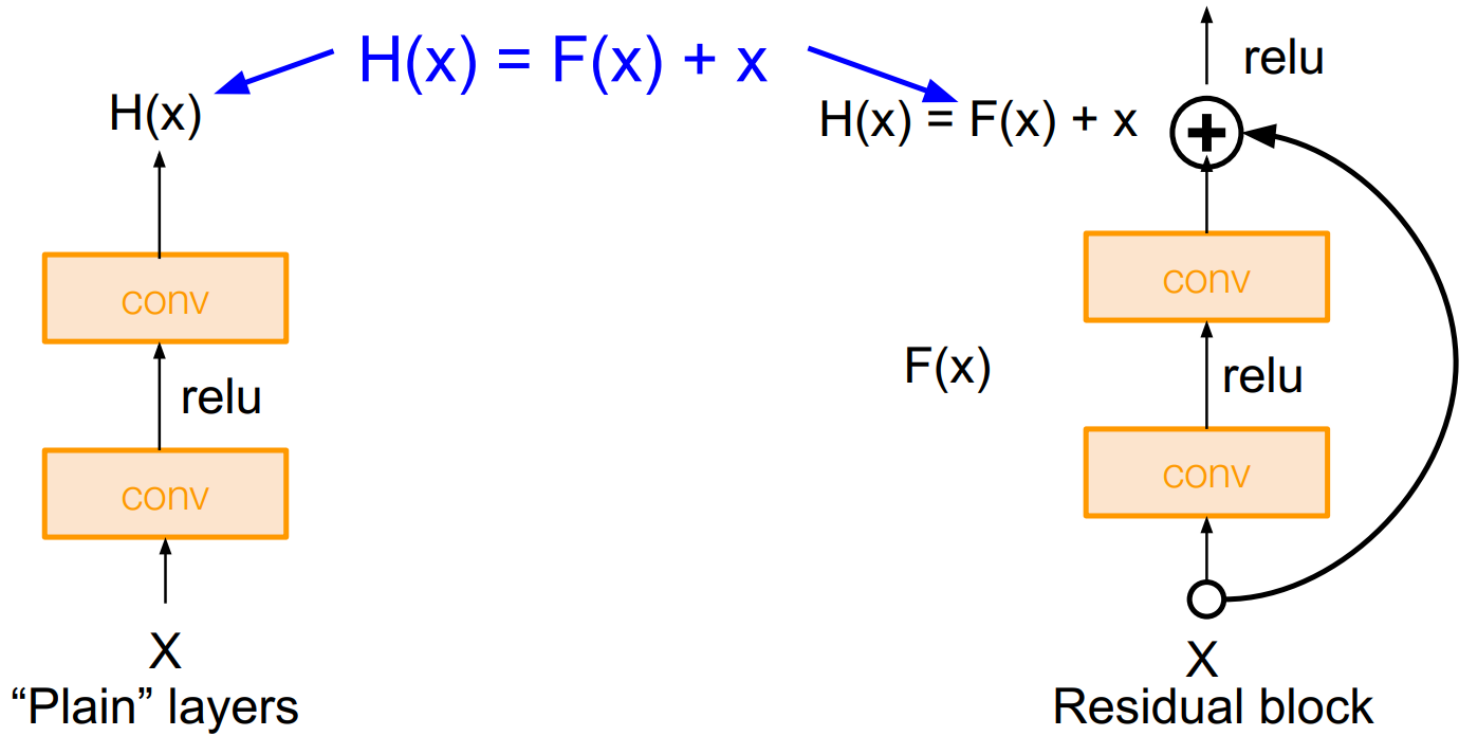


Slides credit: Fei-Fei Li, Ranjay Krishna and Danfei Xu

Case Study: ResNet

[He et al., 2015]

Solution: Use network layers to fit a residual mapping instead of directly trying to fit a desired underlying mapping



Identity mapping:
H(x) = x if F(x) = 0

Use layers to fit **residual** F(x) = H(x) - x instead of H(x) directly

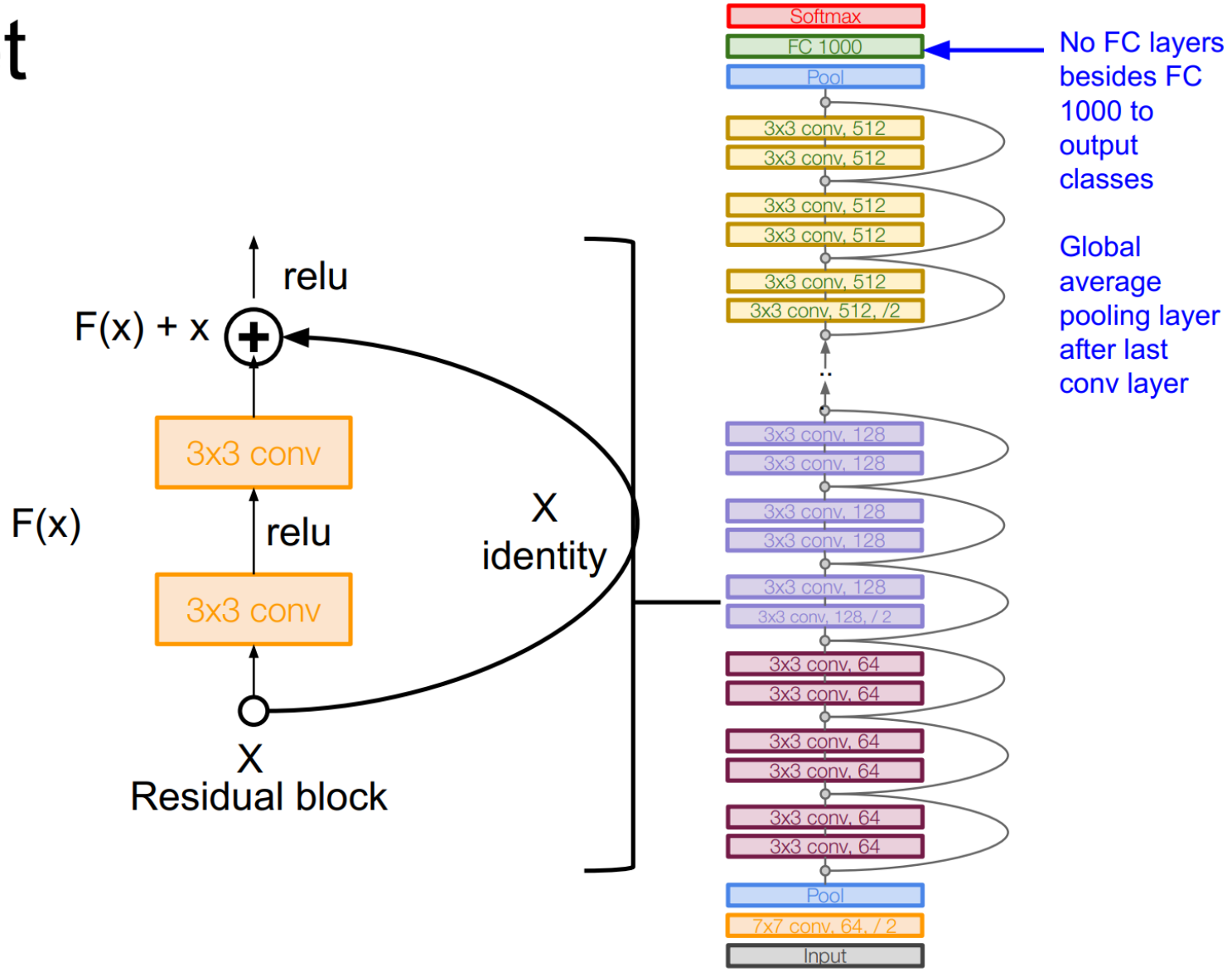
Slides credit: Fei-Fei Li, Ranjay Krishna and Danfei Xu

Case Study: ResNet

[He et al., 2015]

Full ResNet architecture:

- Stack residual blocks
- Every residual block has two 3x3 conv layers
- Periodically, double # of filters and downsample spatially using stride 2 (/2 in each dimension)
- Additional conv layer at the beginning (stem)
- No FC layers at the end (only FC 1000 to output classes)
- (In theory, you can train a ResNet with input image of variable sizes)



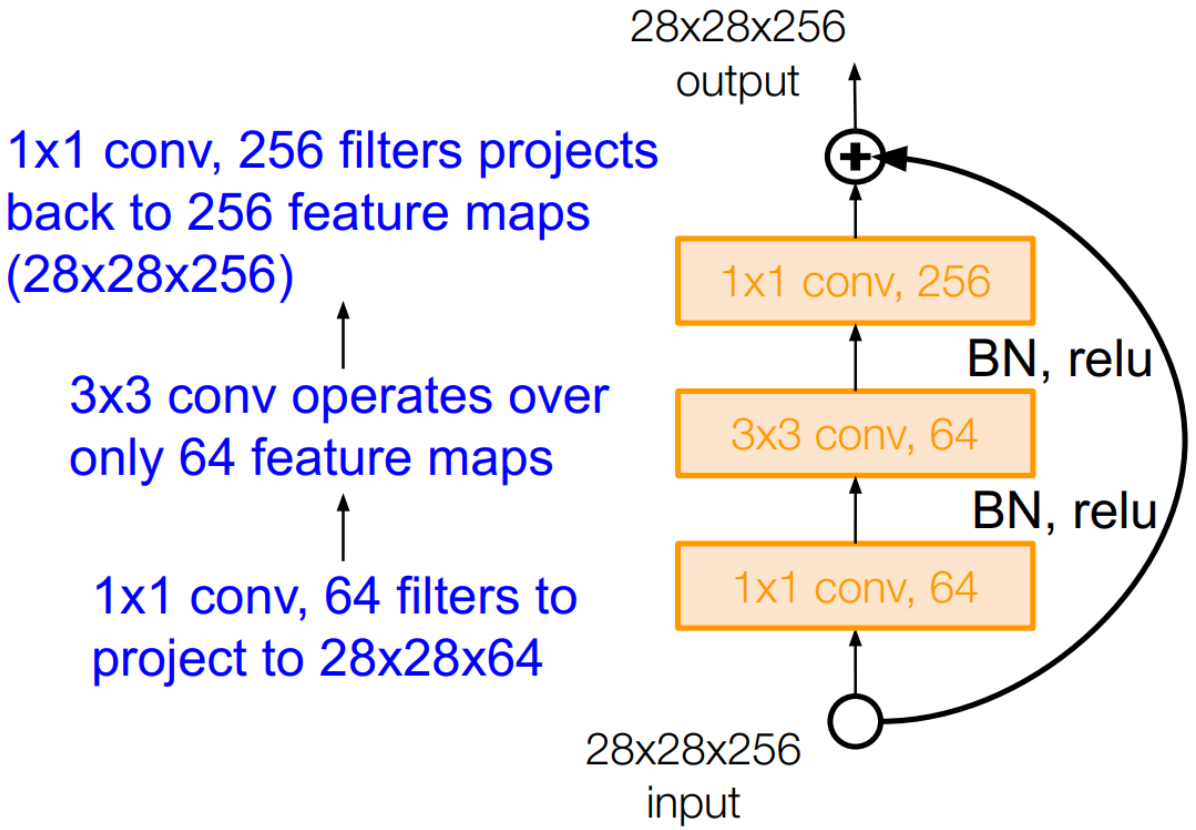
Slides credit: Fei-Fei Li, Ranjay Krishna and Danfei Xu

Case Study: ResNet

[He et al., 2015]

For deeper networks (ResNet-50+), use "bottleneck" layer to improve efficiency (similar to GoogLeNet)

Total depths of 18, 34, 50, 101, or 152 layers for ImageNet

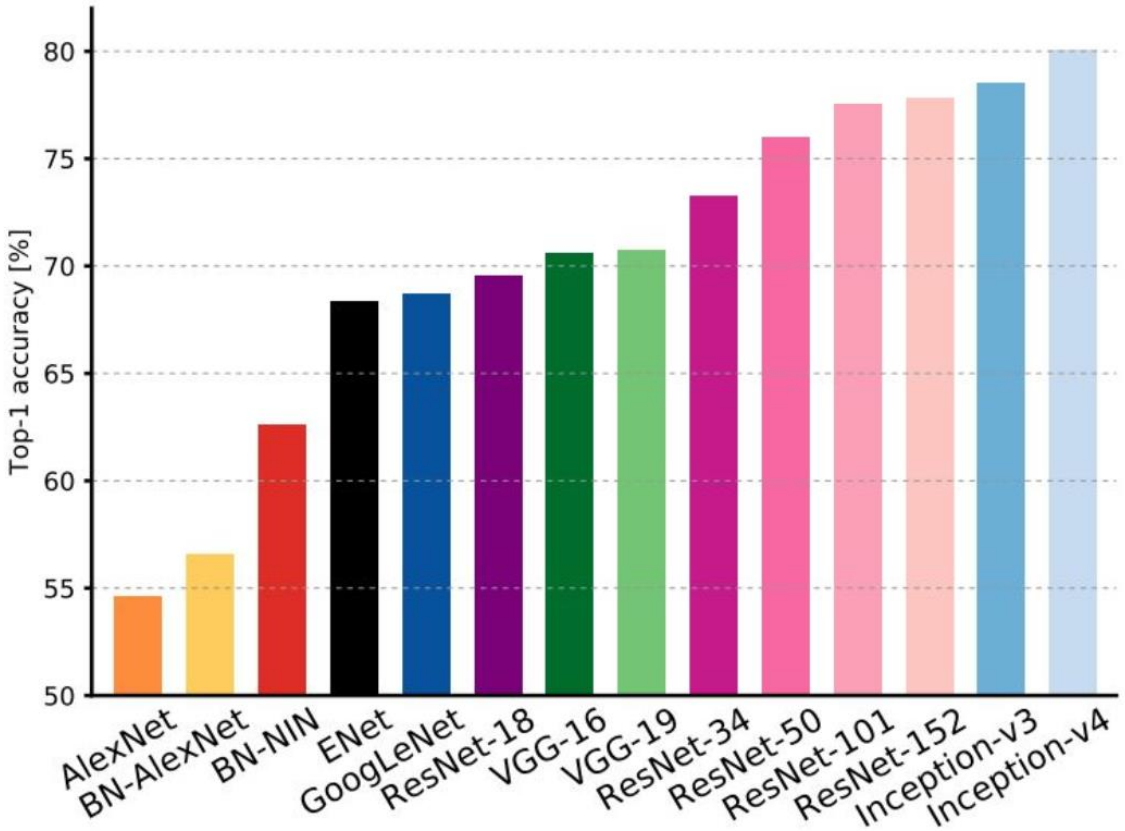


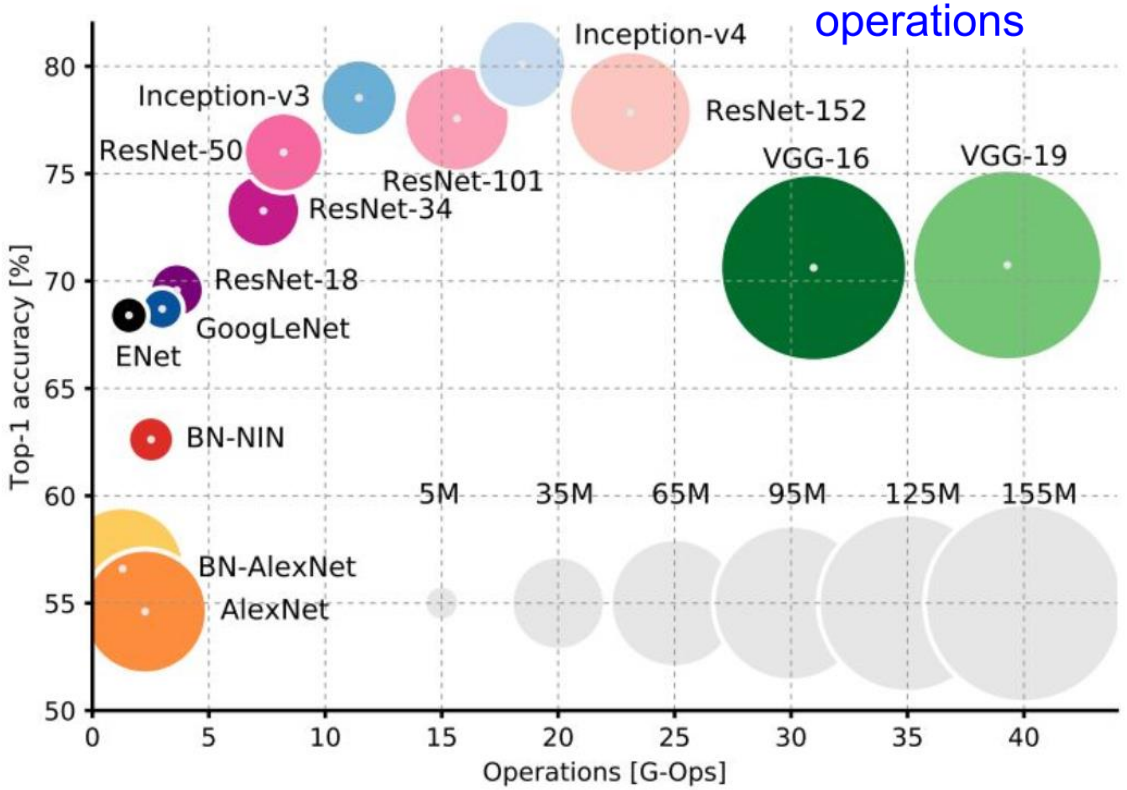
Slides credit: Fei-Fei Li, Ranjay Krishna and Danfei Xu

Comparing complexity...

ResNet:
Moderate efficiency depending on model, highest accuracy

VGG: most parameters, most operations





An Analysis of Deep Neural Network Models for Practical Applications, 2017.

AlexNet:
Smaller compute, still memory heavy, lower accuracy

Slides credit: Fei-Fei Li, Ranjay Krishna and Danfei Xu

Main takeaways

AlexNet showed that you can use CNNs to train Computer Vision models.
ZFNet, **VGG** shows that bigger networks work better
GoogLeNet is one of the first to focus on efficiency using 1x1 bottleneck convolutions and global avg pool instead of FC layers
ResNet showed us how to train extremely deep networks

- Limited only by GPU & memory!
- Showed diminishing returns as networks got bigger

After ResNet: CNNs were better than the human metric and focus shifted to Efficient networks:

- Lots of tiny networks aimed at mobile devices: **MobileNet**, **ShuffleNet**

Neural Architecture Search can now automate architecture design

Slides credit: Fei-Fei Li, Ranjay Krishna and Danfei Xu

REFERENCES

- Textbook
  - Deep Learning by Ian Goodfellow, Yoshua Bengio and Aaron Courville, MIT Press
- Online Resources:
  - https://sebastianraschka.com/blog/2021/dl-course.html (Lectures and code examples)
  - https://www.cs.ubc.ca/~lsigal/532S_2018W2/Lecture5.pdf
  - http://cse.iitkgp.ac.in/~sudeshna/courses/DL17/CNN-22mar2017.pdf
  - http://cs231n.stanford.edu/slides/2021/lecture_7.pdf
  - http://cs231n.stanford.edu/slides/2021/lecture_8.pdf
  - http://cs231n.stanford.edu/slides/2021/lecture_9.pdf