Convolutional Neural Network

Chixiao Chen

Announcement

- Homework 3 is available

Problem 2: Training a two-layer neural network using Numpy (70 points)

Assuming you have a tiny dataset which has 8 inputs, 4 classes and 500 samples. Please design a two-layer neural network as the classifier. Both forward (inference) and backward (training) propagation are required. The first 400 samples are for training, and the last 100 samples are for test. The dataset is available via: https://cihlab.github.io/course/dataset.txt. The activation function is ReLU in the case.

The following table is an example interpretation of the dataset file. (The first two lines of the file is illustrated.)

x_1	x_2	x_3	x_4	x_5	x_6	x_7	x_8	Class Label
0.4812	0.7790	0.8904	0.7361	0.9552	0.2119	0.7992	0.2409	4
0.4472	0.5985	0.7859	0.5035	0.6912	0.4038	0.0787	0.2301	1

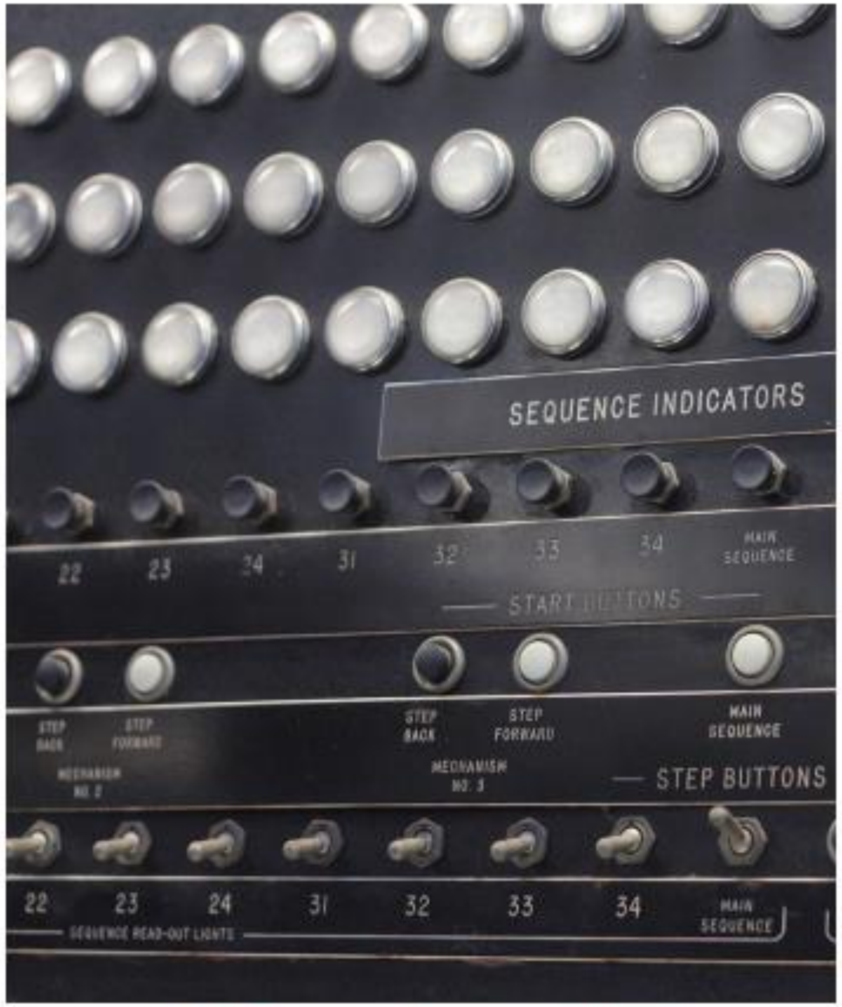
Please submit you code and a brief report with the loss function definition, the final accuracy results, the neuron number in the hidden layers, etc. Also include your strategy for batch size and learning rate. (Hint: It is encouraged to use python and numpy (https://www.numpy.org/). You can refer to the slides 34 in the lecture 7 notes. The problem does not encourage you to use Tensorflow/caffe/pytorch, but if you have no idea about numpy, you can also using these frameworks.)

Overview

- A bit of history / inspiration / Dataset / Status quo
- CNN
- Training on CNN
- Case Study – AlexNet / VGG / ResNet / MobileNet

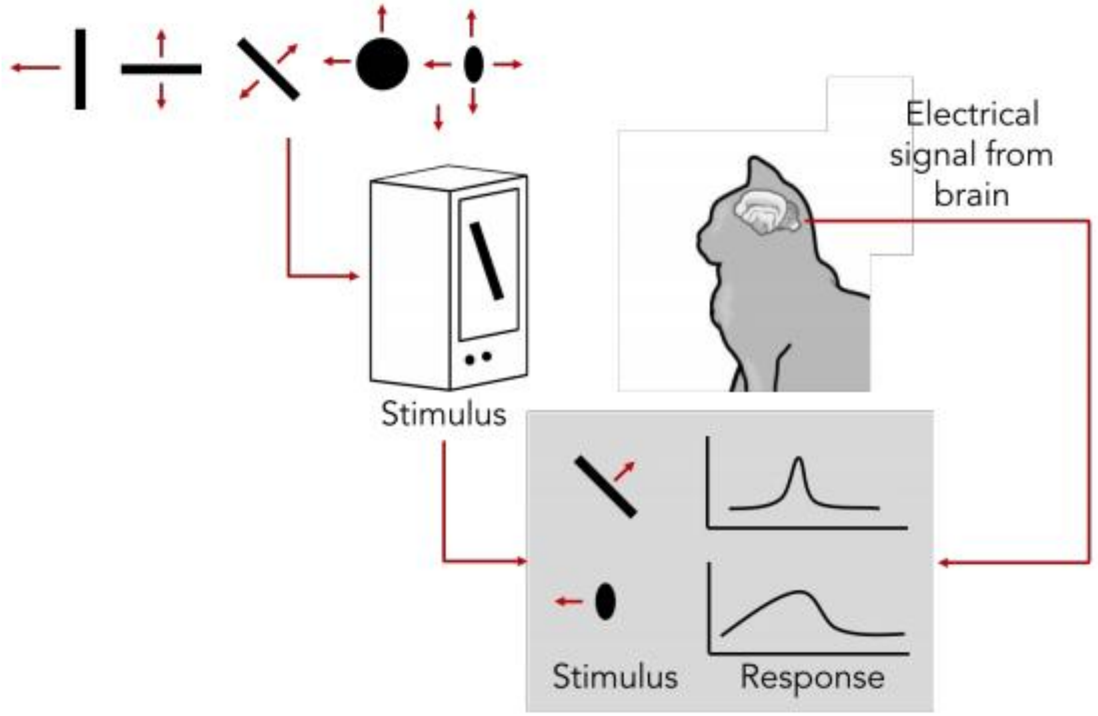
First NN hardware

- Mark I Perception
- The machine was connected to a camera that used 20×20 cadmium sulfide photocells to produce a 400-pixel image.
- recognized letters of the alphabet
- Activation : ReLU



Biomedical Understanding on Cortex

- Hubel & Wiesel
- 1959
  RECEPTIVE FIELDS OF SINGLE NEURONES IN THE CAT'S STRIATE CORTEX
- 1962
  RECEPTIVE FIELDS, BINOCULAR INTERACTION AND FUNCTIONAL ARCHITECTURE IN THE CAT'S VISUAL CORTEX



Human's Cortex

- Topographical mapping in the cortex: nearby cells in cortex represent nearby regions in the visual field

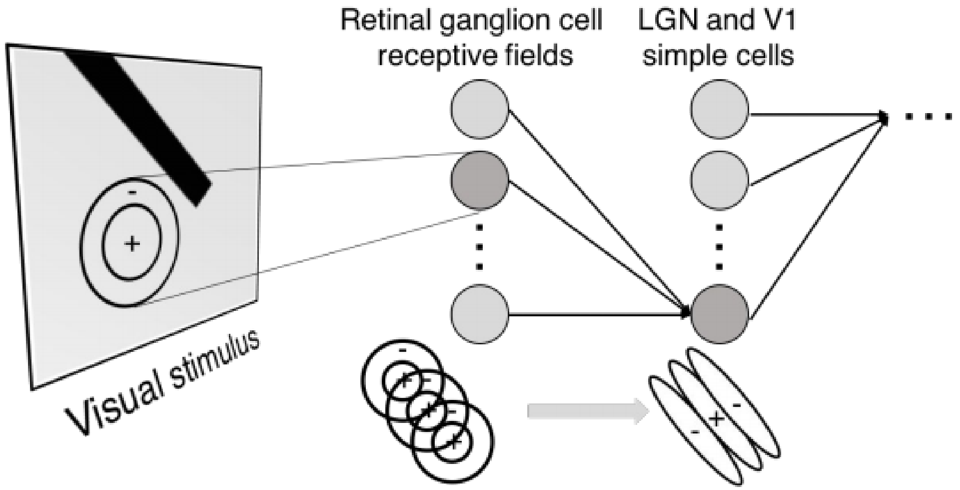
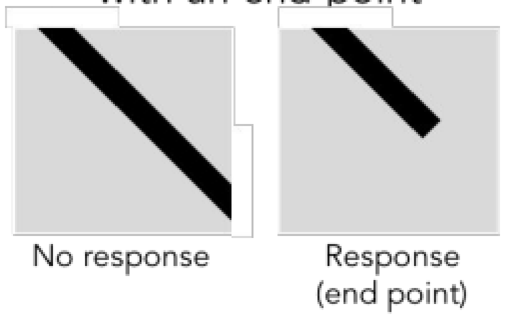


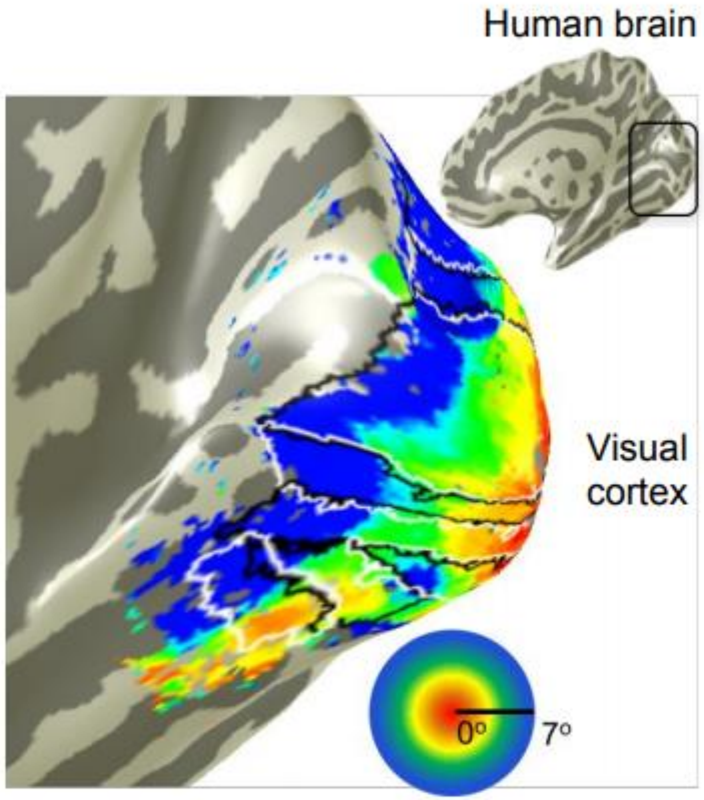
Illustration of hierarchical organization in early visual pathways by Lane McIntosh, copyright CS231n 2017

Simple cells: Response to light orientation

Complex cells: Response to light orientation and movement

Hypercomplex cells: response to movement with an end point

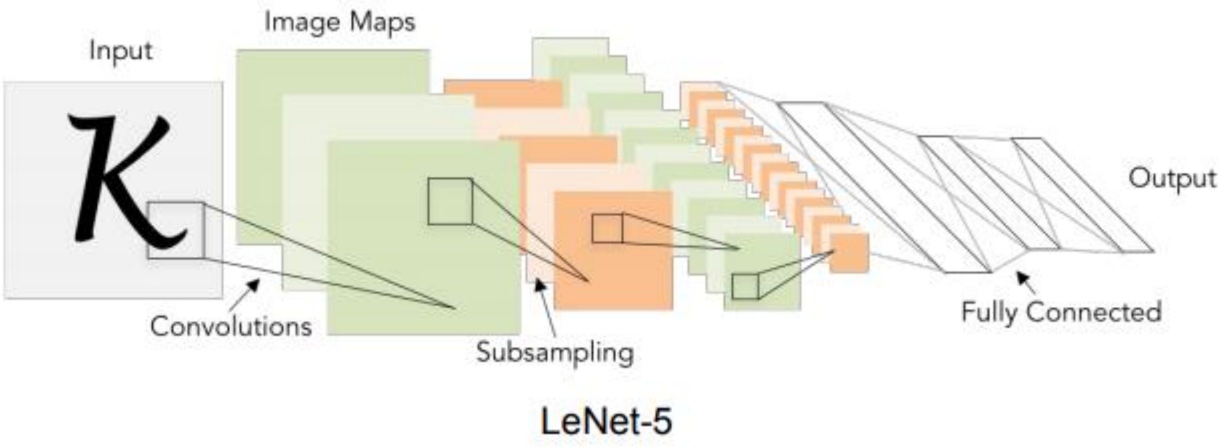




Retinotopy images courtesy of Jesse Gomez in the Stanford Vision & Perception Neuroscience Lab.

The first CNN

- Gradient-based learning applied to document recognition [Lecun 1998], Dataset is MNIST
  - simple cells: modifiable parameters
  - complex cells: perform pooling



Modern Image Classification Data Set

- Object Classification
- 256x256 pixels (color)
- 1000 Classes
- 1.3M Training
- 100,000 Testing
- 50,000 Validation
- image-net.org/

IMAGENET

Fine grained Classes
(120 breeds)

Image Source: http://karpathy.github.io/

Image Source: Krizhevsky et al., NIPS 2012

Top-5 Error

Winner 2012
(16.42% error)

Winner 2016
(2.99% error)

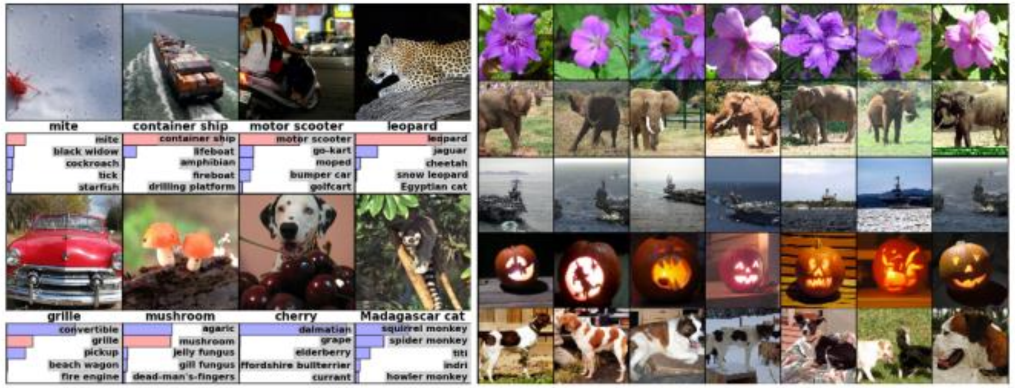
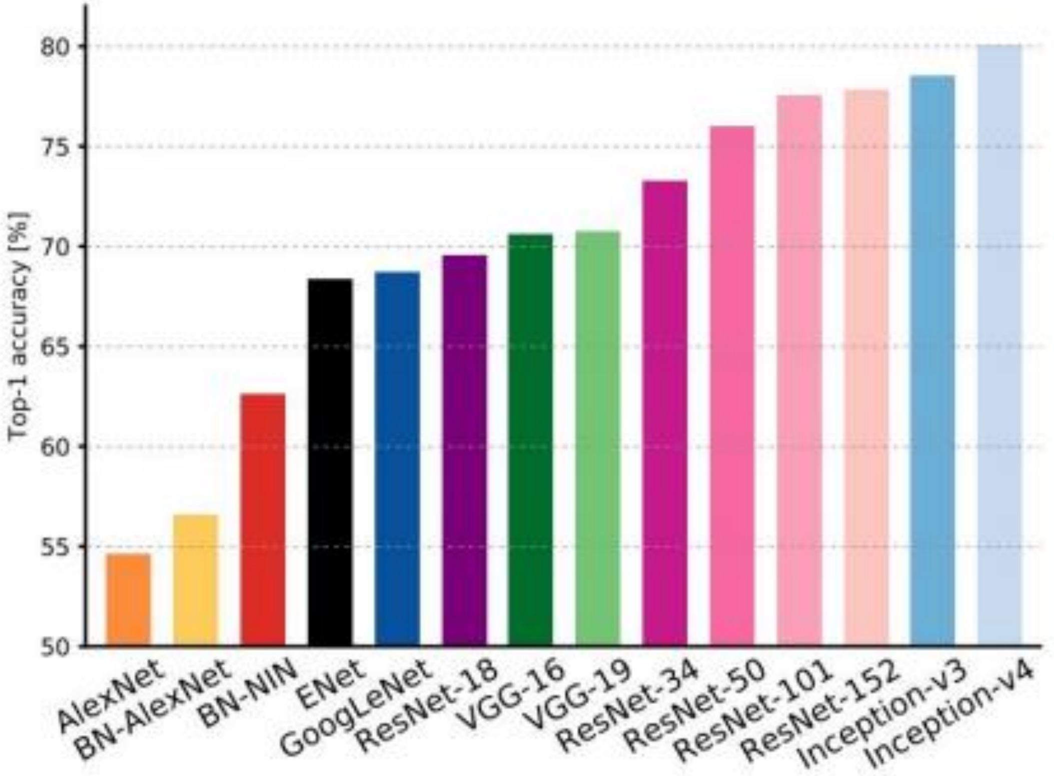
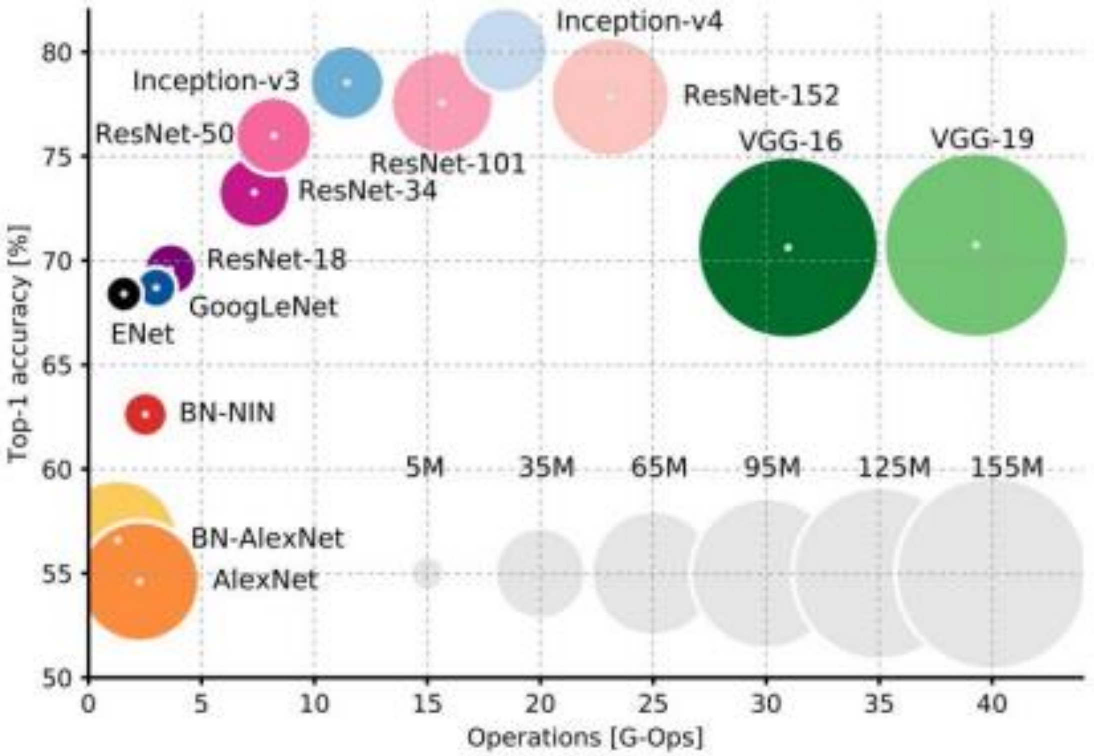


Image Classification Status Quo

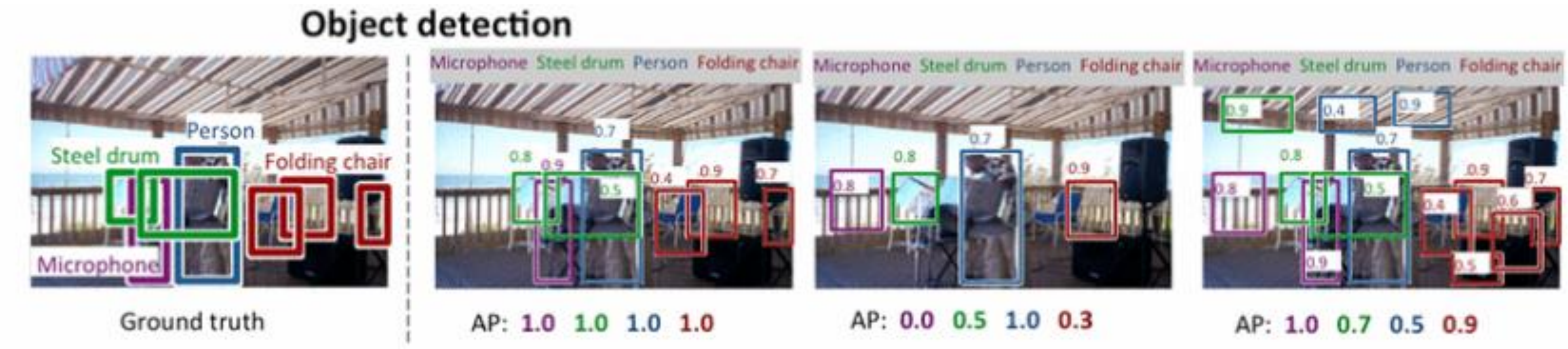
- Up to 2017





Localization and Detection

- Object Detection



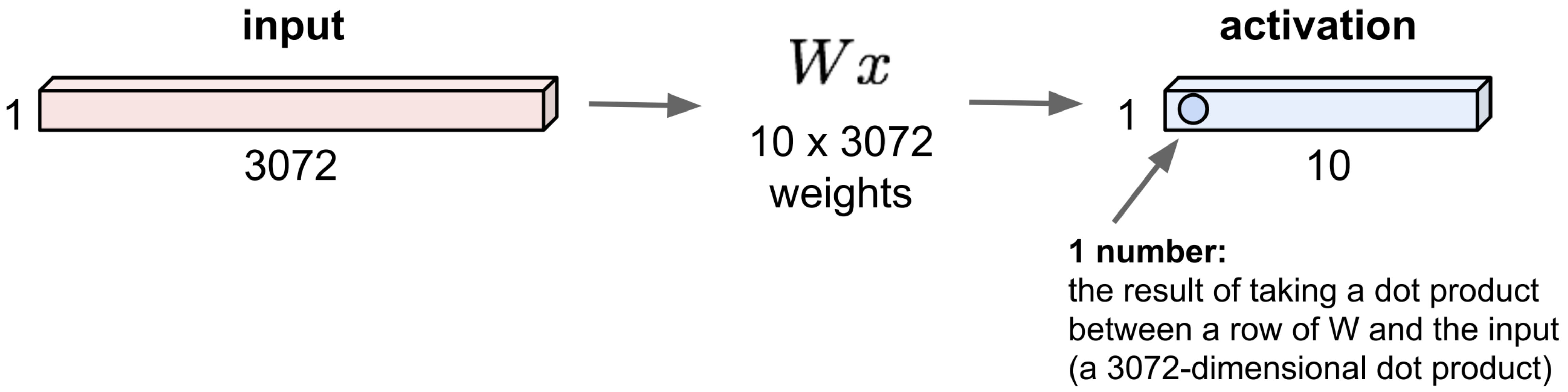
- Pascal VOC:
  - 11k images, 20 classes
- MS COCO:
  - 300k images, even segmentation

What’s
Convolutional Neural Network

-- Ref F. Li @ CS231 Standford

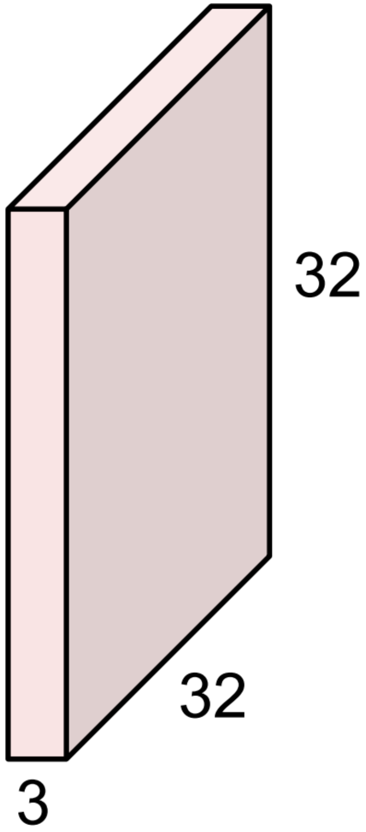
Fully Connected Layer

32x32x3 image -> stretch to 3072 x 1

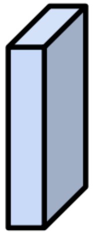


Convolution Layer

32x32x3 image

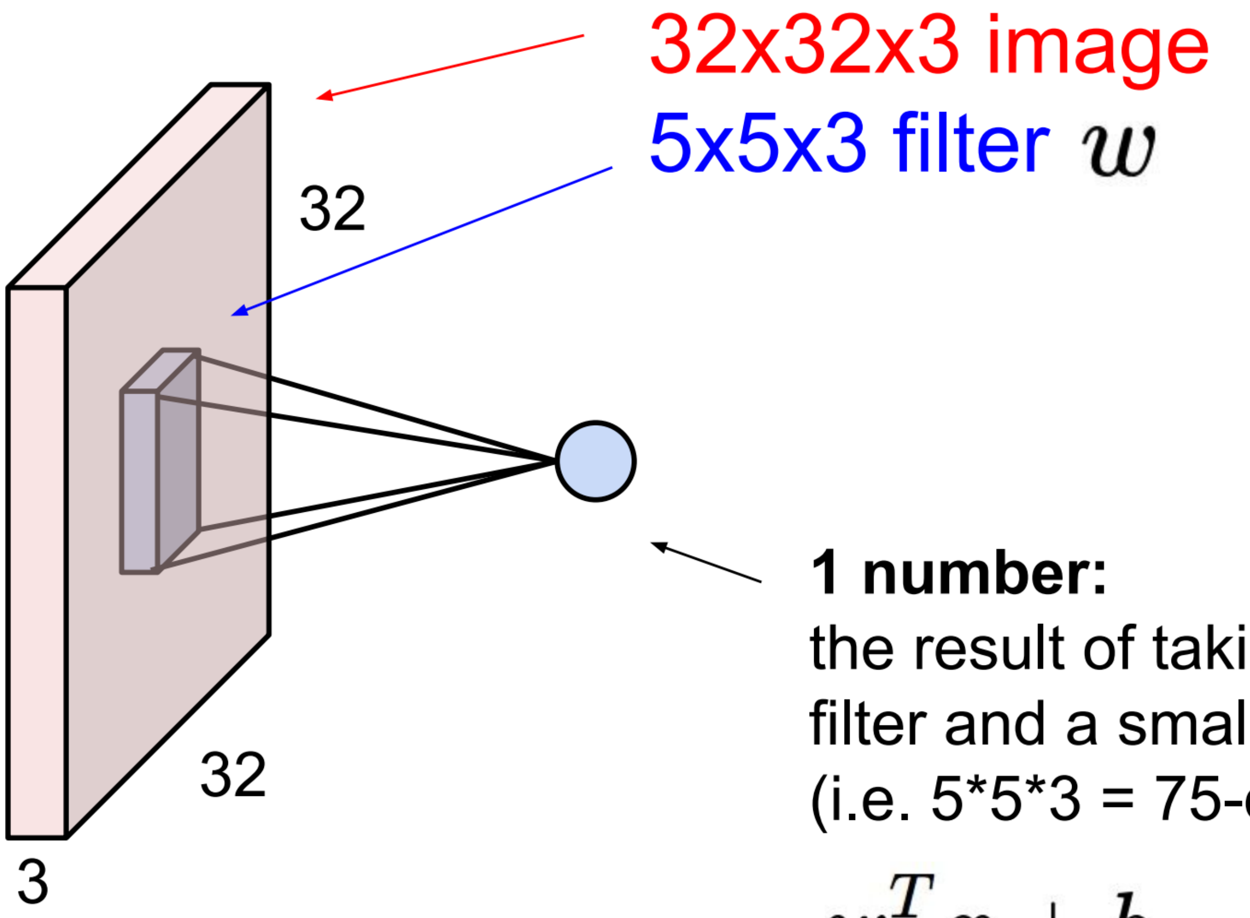


5x5x3 filter



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

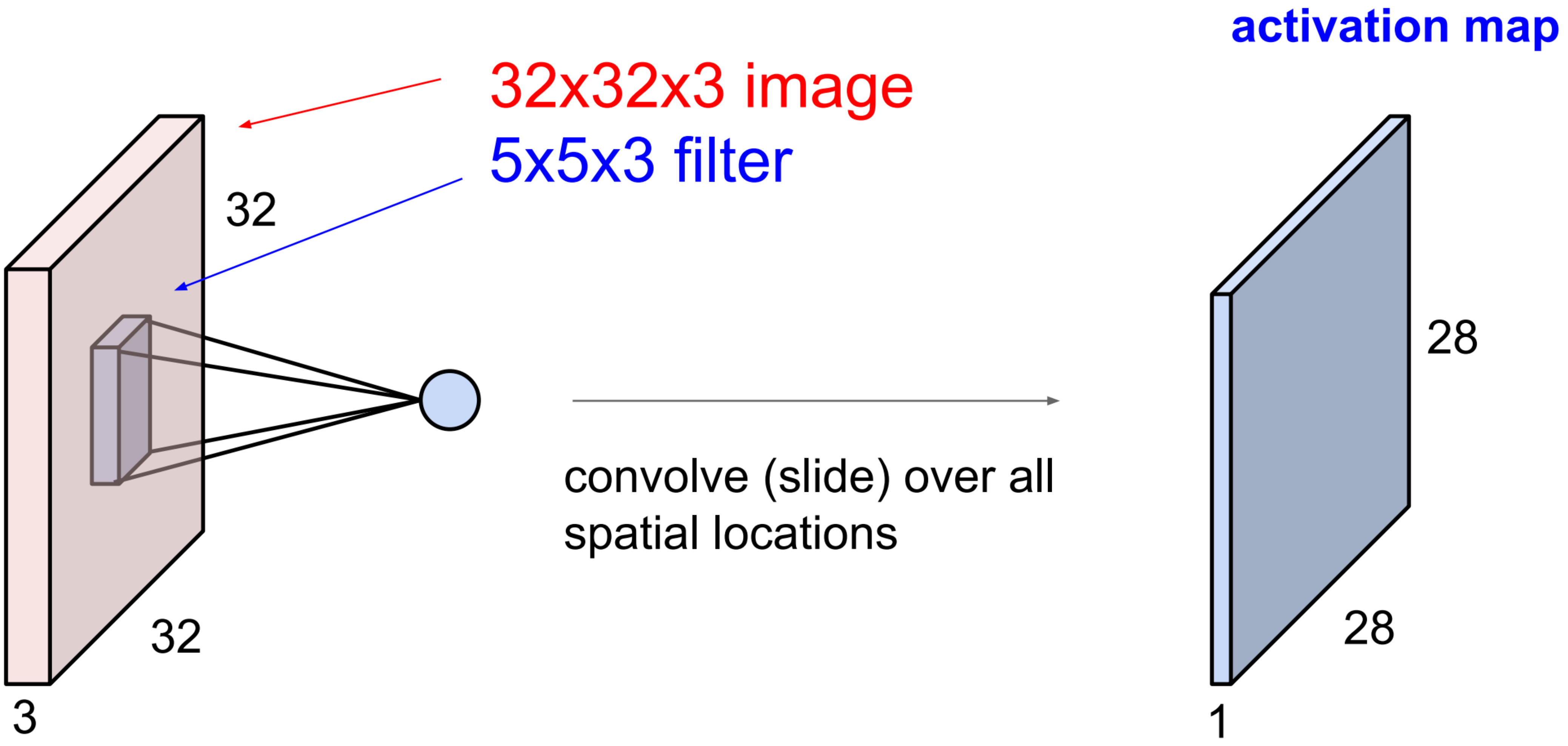
Convolution Layer



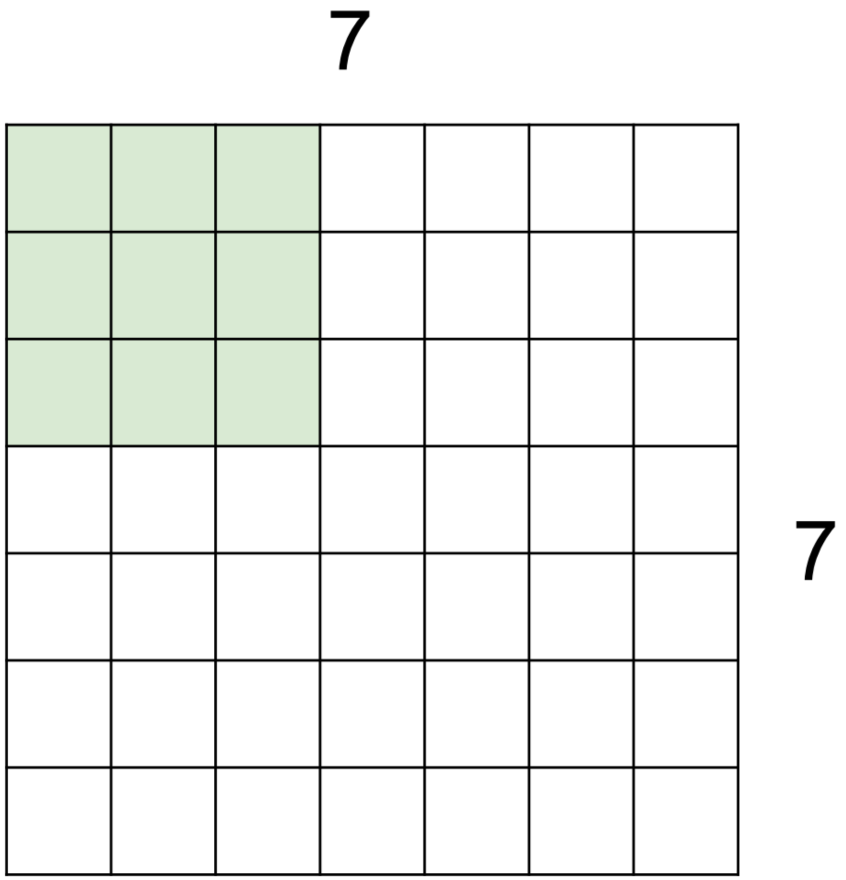
1 number:
the result of taking a dot product between the filter and a small 5x5x3 chunk of the image
(i.e. 5*5*3 = 75-dimensional dot product + bias)

$$w^T x + b$$

Convolution Layer



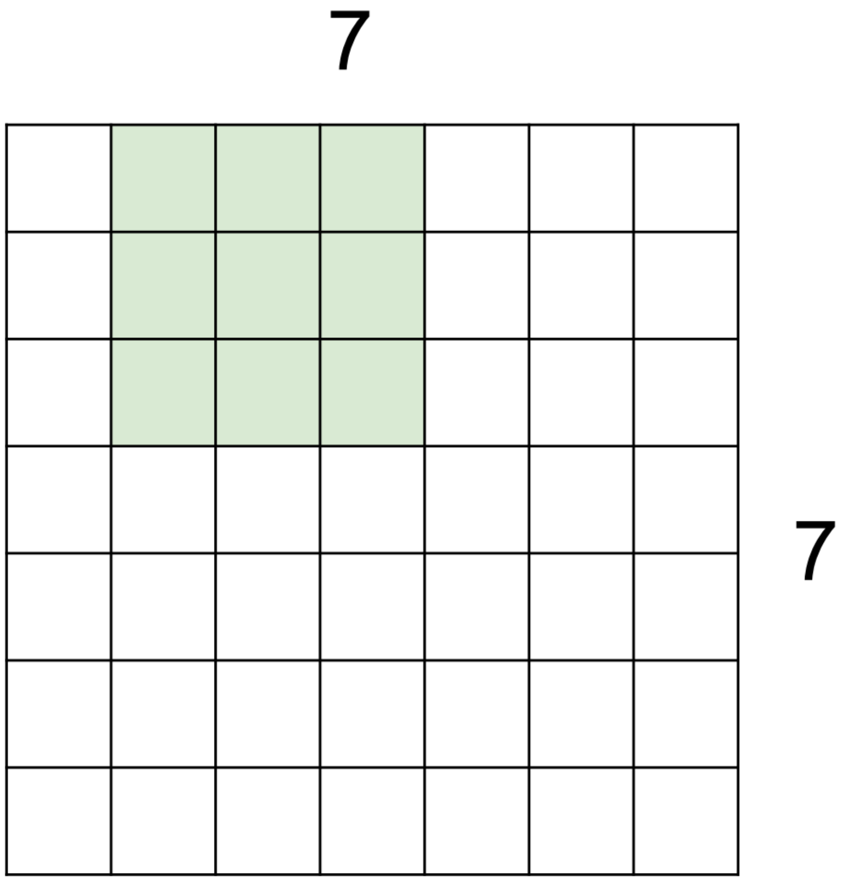
A closer look at spatial dimensions:



7x7 input (spatially)
assume 3x3 filter

Stride = 1 case

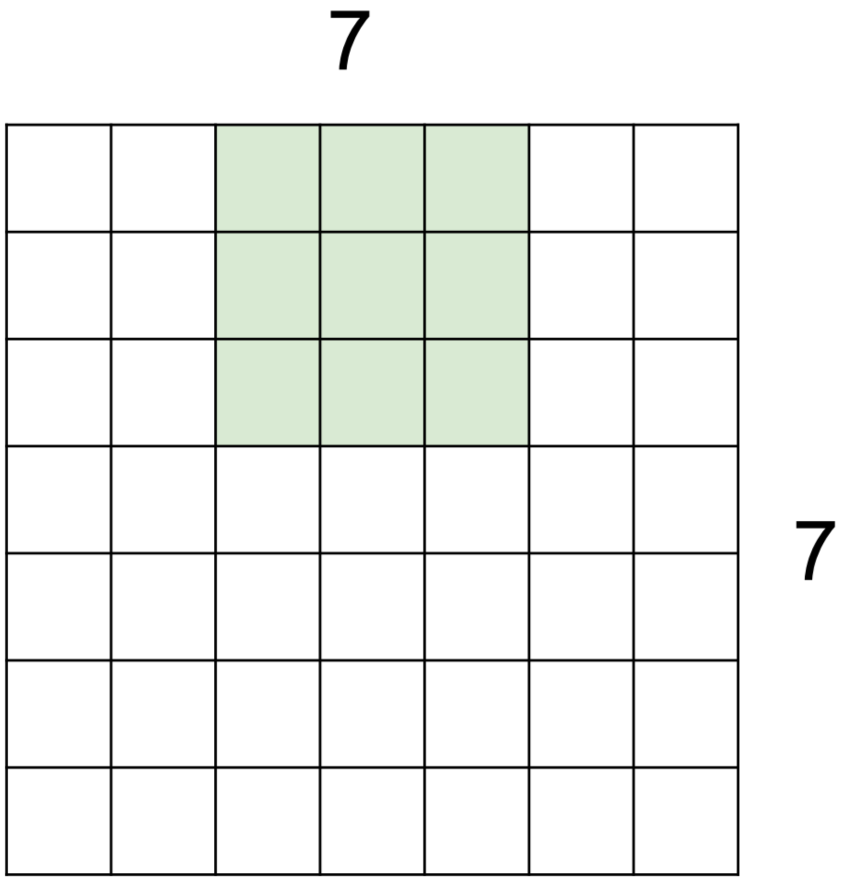
A closer look at spatial dimensions:



7x7 input (spatially)
assume 3x3 filter

Stride = 1 case

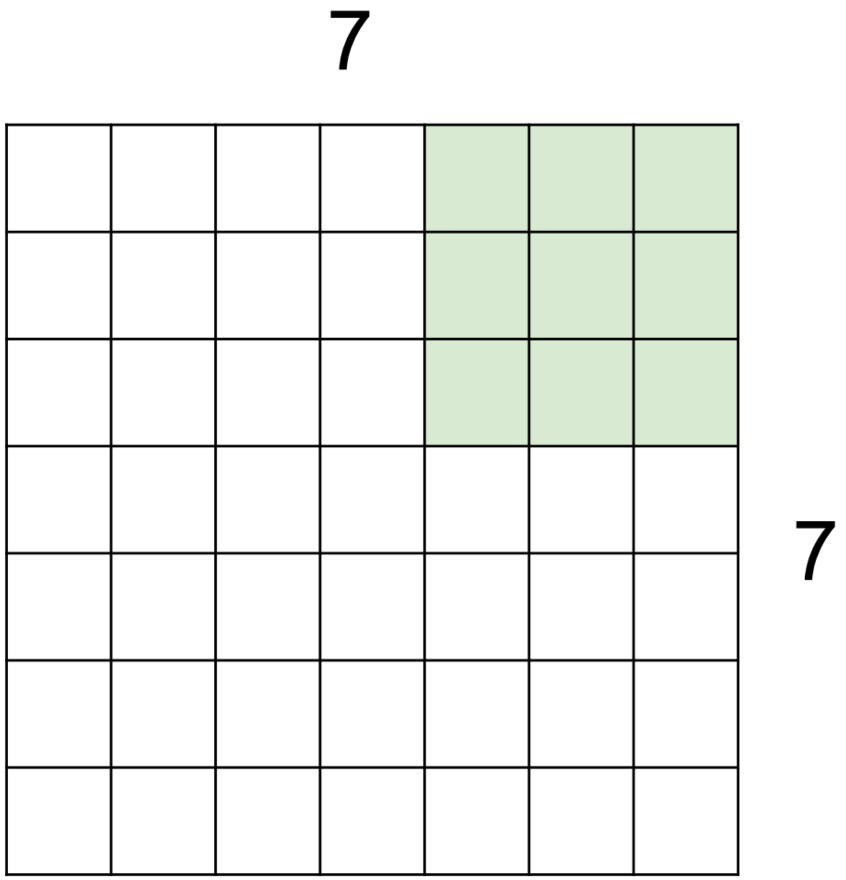
A closer look at spatial dimensions:



7x7 input (spatially)
assume 3x3 filter

Stride = 1 case

A closer look at spatial dimensions:

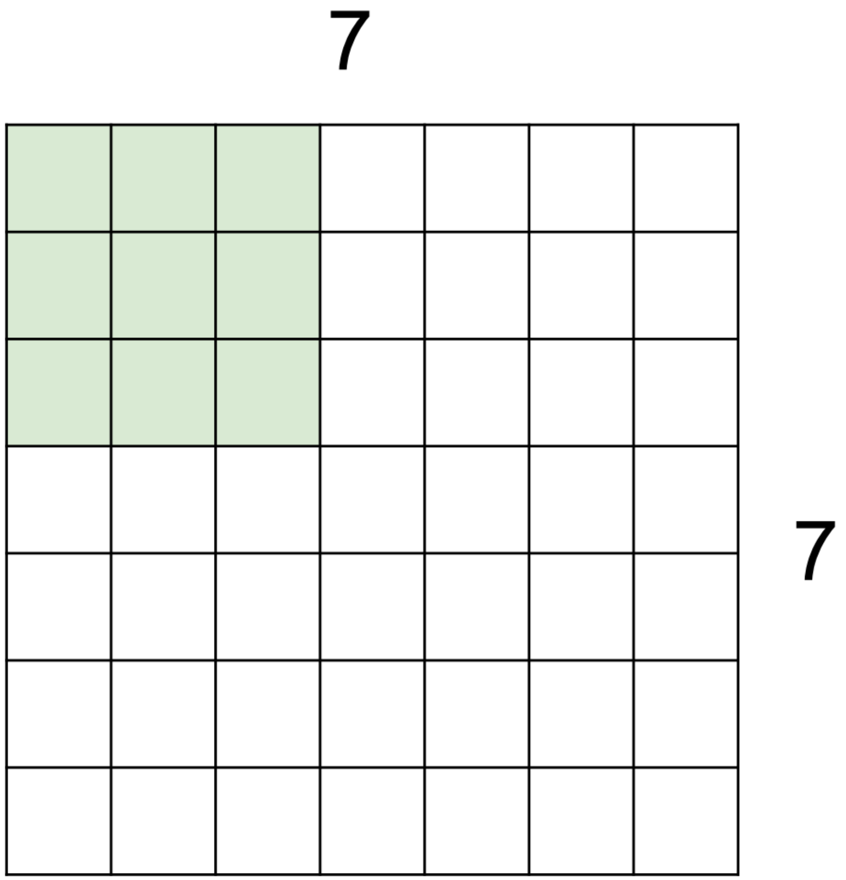


7x7 input (spatially)
assume 3x3 filter

=> 5x5 output

Stride = 1 case

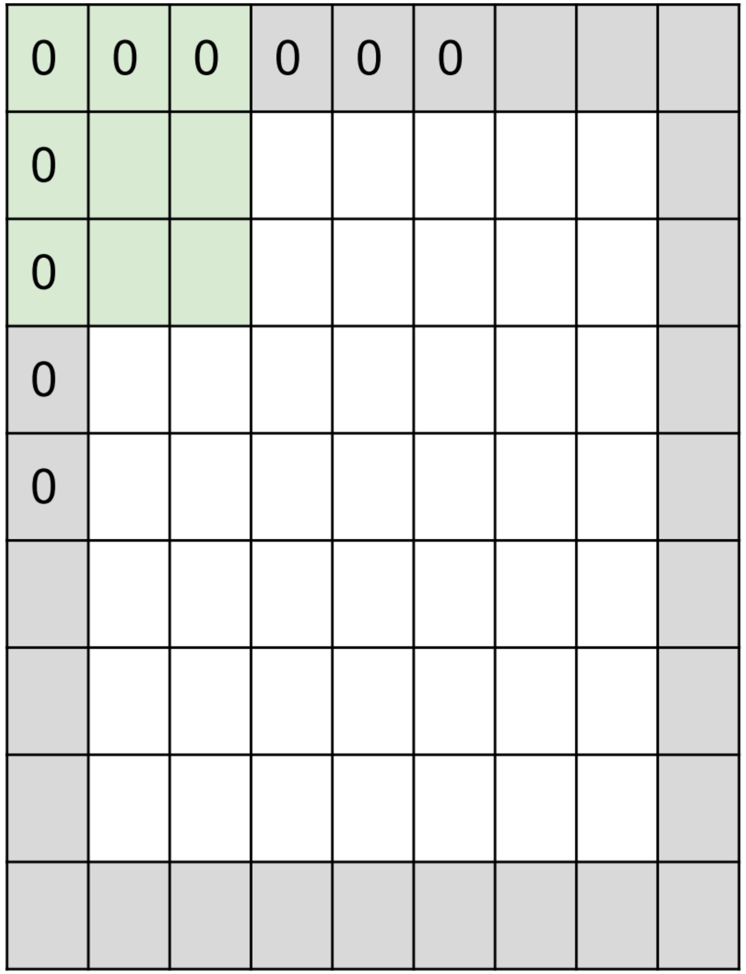
A closer look at spatial dimensions:



7x7 input (spatially)
assume 3x3 filter
applied **with stride 3?**

doesn't fit!
cannot apply 3x3 filter on
7x7 input with stride 3.

In practice: Common to zero pad the border

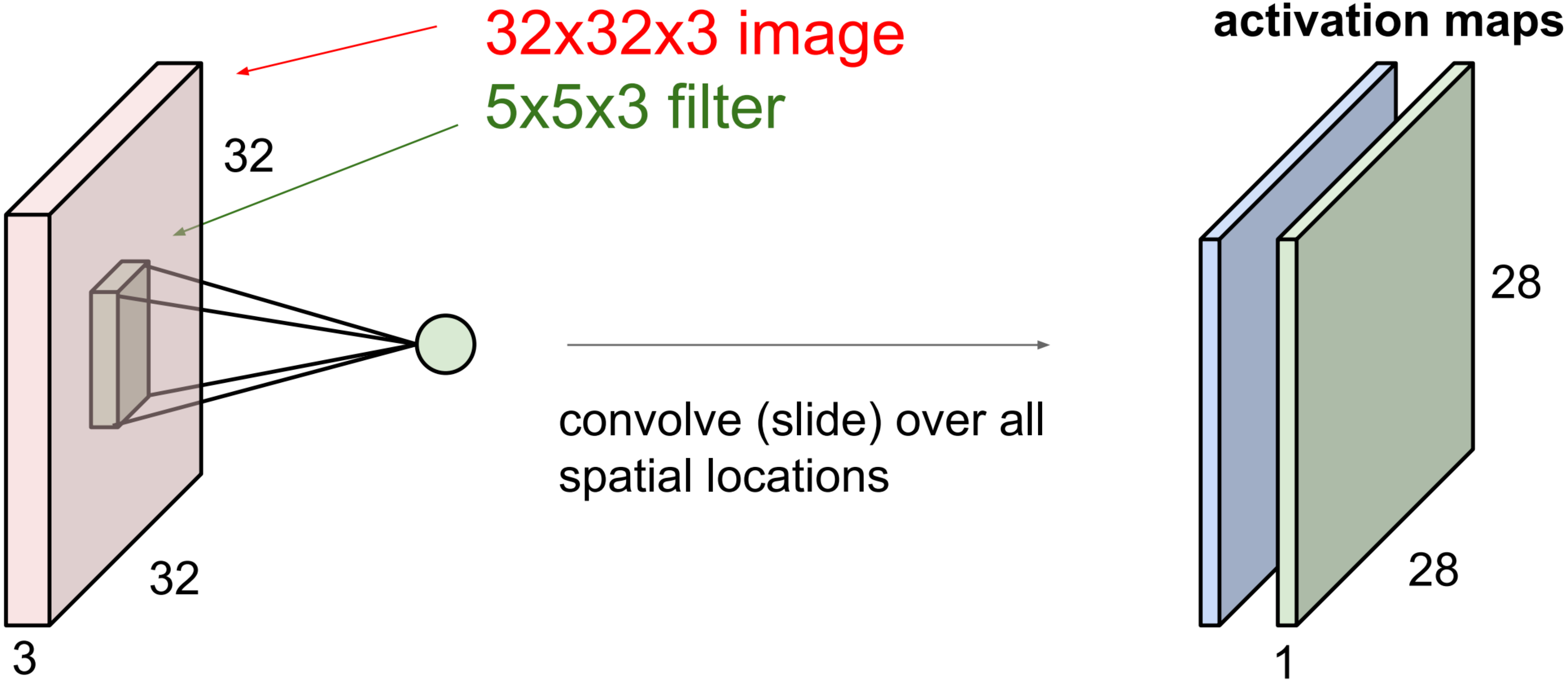


e.g. input 7x7
3x3 filter, applied with **stride 1**
pad with 1 pixel border => what is the output?

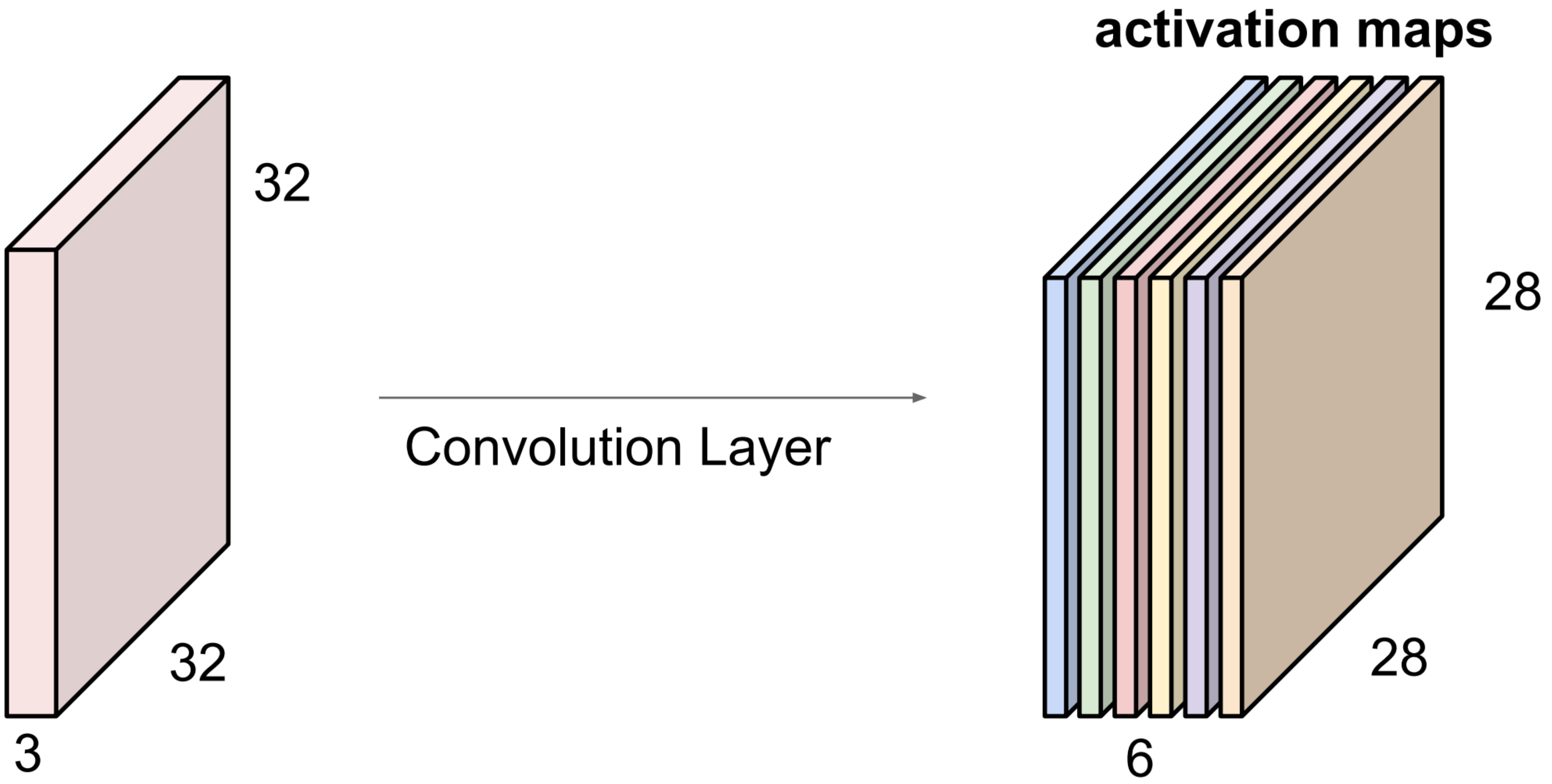
7x7 output!
in general, common to see CONV layers with stride 1, filters of size FxF, and zero-padding with (F-1)/2. (will preserve size spatially)
e.g. F = 3 => zero pad with 1
F = 5 => zero pad with 2
F = 7 => zero pad with 3

Convolution Layer

consider a second, green filter

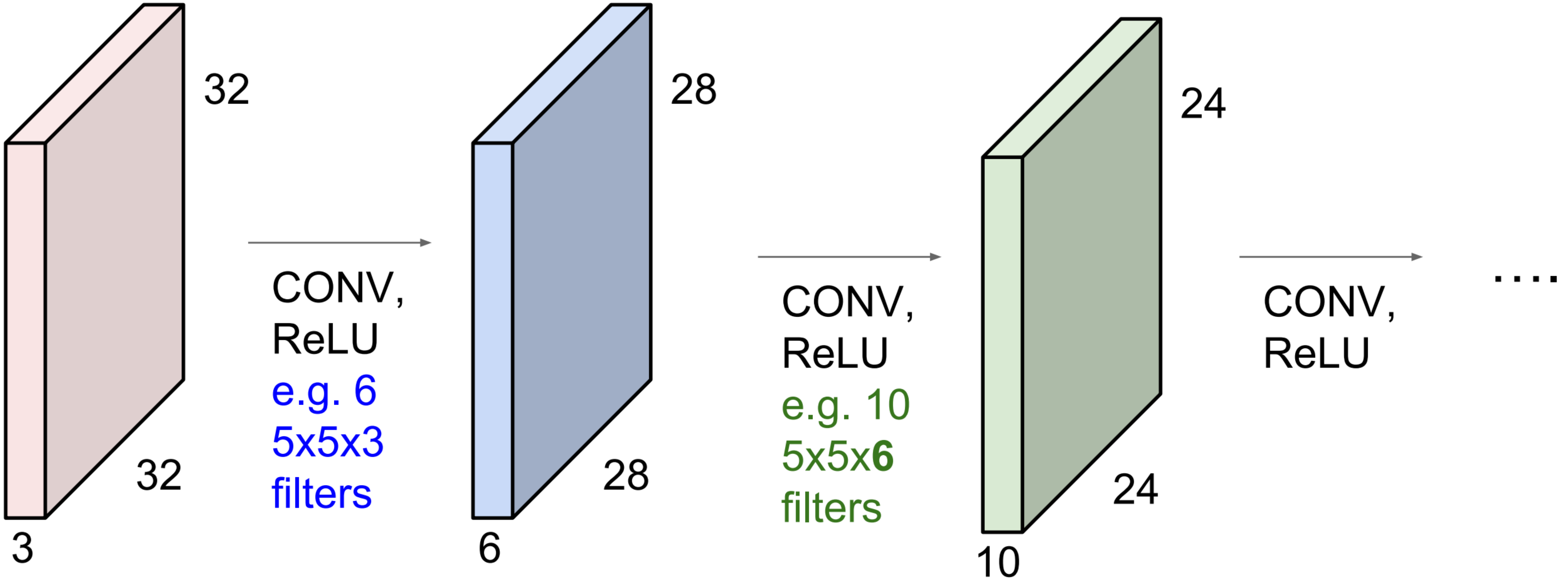


For example, if we had 6 5x5 filters, we'll get 6 separate activation maps:



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

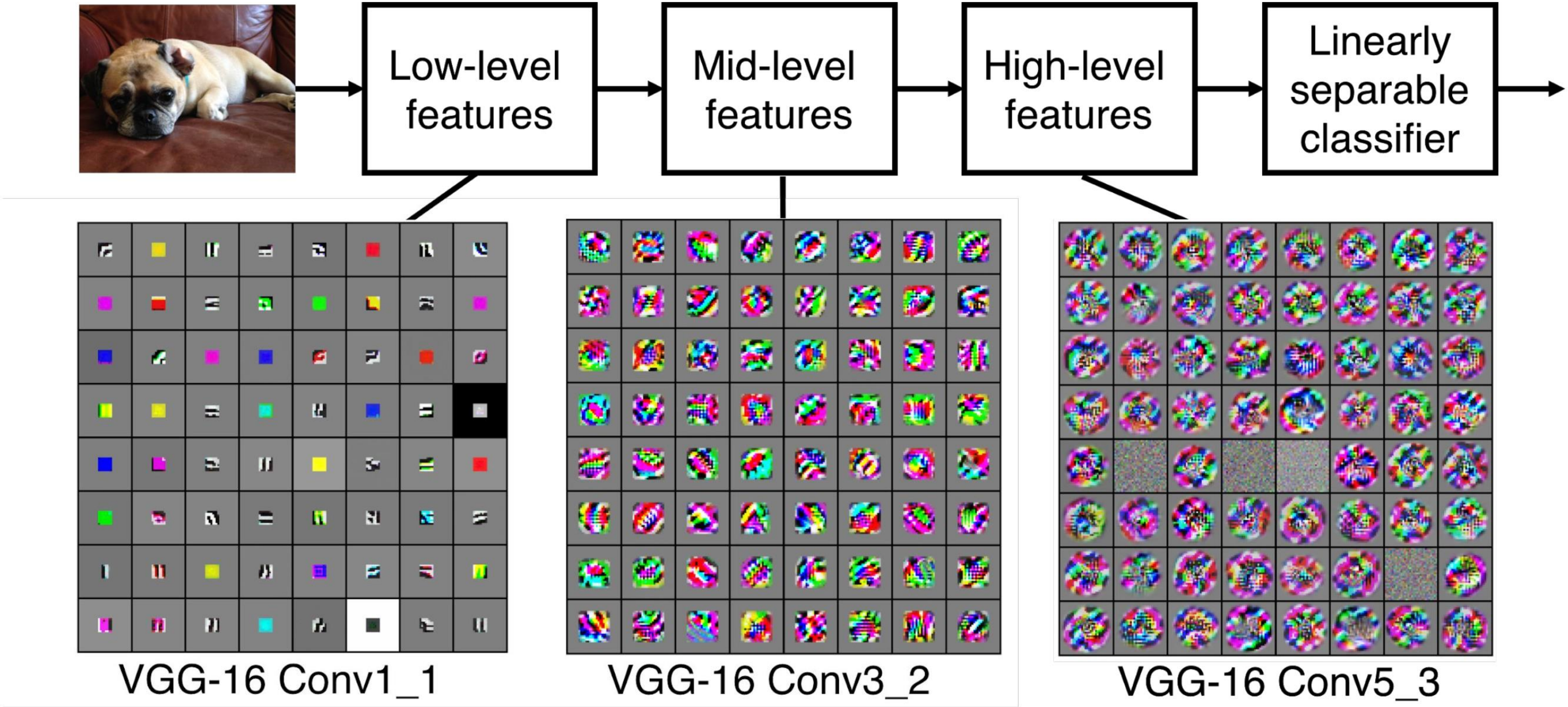
Preview: ConvNet is a sequence of Convolution Layers, interspersed with activation functions



Preview

[Zeiler and Fergus 2013]

Visualization of VGG-16 by Lane McIntosh. VGG-16 architecture from [Simonyan and Zisserman 2014].



Summary. To summarize, the Conv 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.

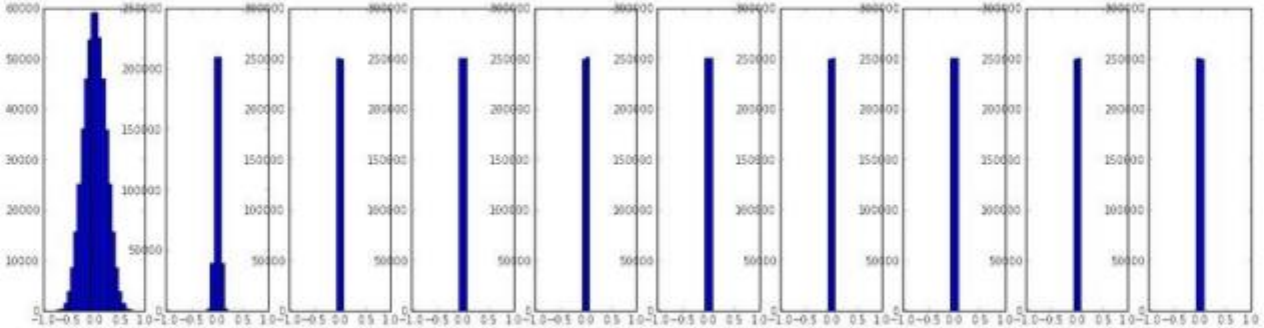
Common settings:

K = (powers of 2, e.g. 32, 64, 128, 512)
- F = 3, S = 1, P = 1
- F = 5, S = 1, P = 2
- F = 5, S = 2, P = ? (whatever fits)
- F = 1, S = 1, P = 0

Tips for Training on CNNs (and other DNNs)

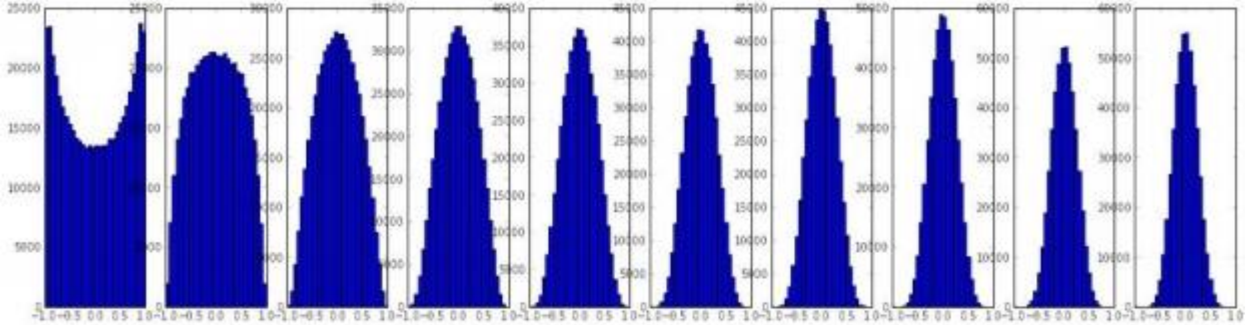
Weight Initialization

- First idea:
  Small random numbers (gaussian with zero mean and 1e-2 d)
  Works ~okay for small networks, but problems with deeper networks.



Random for deep layer

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



Xaiver Initialization For tanh

Xaiver Initialization For relu

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

```
W = np.random.randn(fan_in, fan_out) / np.sqrt(2/fan_in) # layer initialization
```

Learning Rate

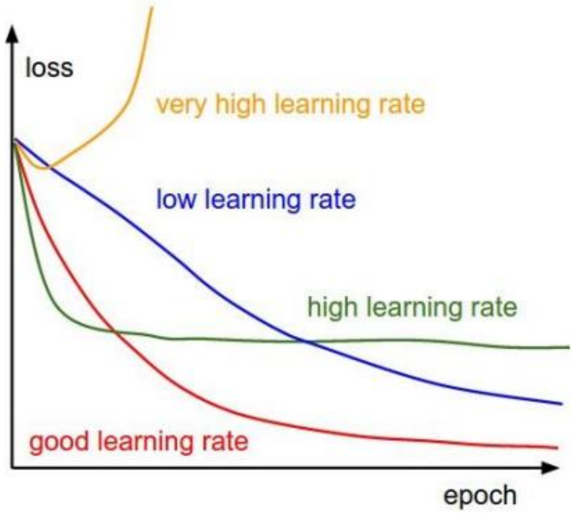
- loss not going down: learning rate too low loss exploding: learning rate too high

```
max_count = 100
for count in xrange(max_count):
      reg = 10**uniform(-5, 5)
      lr = 10**uniform(-3, -6)

      trainer = ClassifierTrainer()
      model = init_two_layer_model(32*32*3, 50, 10) # input size, hidden size, number of classes
      trainer = ClassifierTrainer()
      best_model_local, stats = trainer.train(X_train, y_train, X_val, y_val,
                                   model, two_layer_net,
                                   num_epochs=5, reg=reg,
                                   update='momentum', learning_rate_decay=0.9,
                                   sample_batches = True, batch_size = 100,
                                   learning_rate=lr, verbose=False)
```

note it’s best to optimize in log space!

- Cross Validation for initial Learning Rate Search




```
val_acc: 0.412000, lr: 1.405206e-04, reg: 4.793564e-01, (1 / 100)
val_acc: 0.214000, lr: 7.231888e-06, reg: 2.321281e-04, (2 / 100)
val_acc: 0.208000, lr: 2.119571e-06, reg: 8.011857e+01, (3 / 100)
val_acc: 0.196000, lr: 1.551131e-05, reg: 4.374936e-05, (4 / 100)
val_acc: 0.079000, lr: 1.753300e-05, reg: 1.200424e+03, (5 / 100)
val_acc: 0.223000, lr: 4.215128e-05, reg: 4.196174e+01, (6 / 100)
val_acc: 0.441000, lr: 1.750259e-04, reg: 2.110807e-04, (7 / 100)
val_acc: 0.241000, lr: 6.749231e-05, reg: 4.226413e+01, (8 / 100)
val_acc: 0.482000, lr: 4.296863e-04, reg: 6.642555e-01, (9 / 100)
val_acc: 0.079000, lr: 5.401602e-06, reg: 1.599828e+04, (10 / 100)
val_acc: 0.154000, lr: 1.618508e-06, reg: 4.925252e-01, (11 / 100)
```

Adagrad & RMSprop

- Learning rate should not be the constant.

$$n_t = n_{t-1} + g_t^2$$

$$\Delta\theta_t = -\frac{\eta}{\sqrt{n_t + \epsilon}} * g_t$$

AdaGrad

```
grad_squared = 0
while True:
  dx = compute_gradient(x)
  grad_squared += dx * dx
  x -= learning_rate * dx / (np.sqrt(grad_squared) + 1e-7)
```

$$RMS[g]_t = \sqrt{E[g^2]_t + \epsilon}$$

$$\Delta x_t = -\frac{\eta}{RMS[g]_t} * g_t$$

RMSProp

```
grad_squared = 0
while True:
  dx = compute_gradient(x)
  grad_squared = decay_rate * grad_squared + (1 - decay_rate) * dx * dx
  x -= learning_rate * dx / (np.sqrt(grad_squared) + 1e-7)
```

SGD / mini Batch

- Batch Gradient Descent

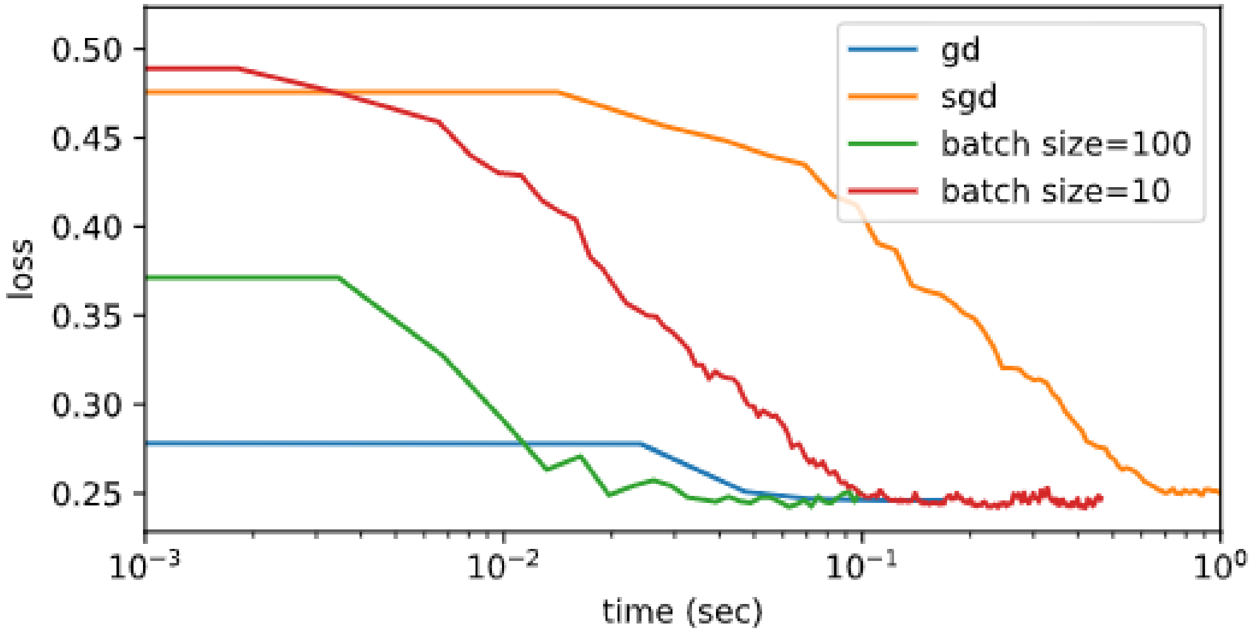
$$x^{(k+1)} = x^{(k)} - t_k \sum_{i=1}^{m} \nabla f_i\left(x^{(k)}\right)$$

- Stochastic Gradient Descent

$$x^{(k+1)} = x^{(k)} - t_k \nabla f_{i_k}\left(x^{(k)}\right)$$

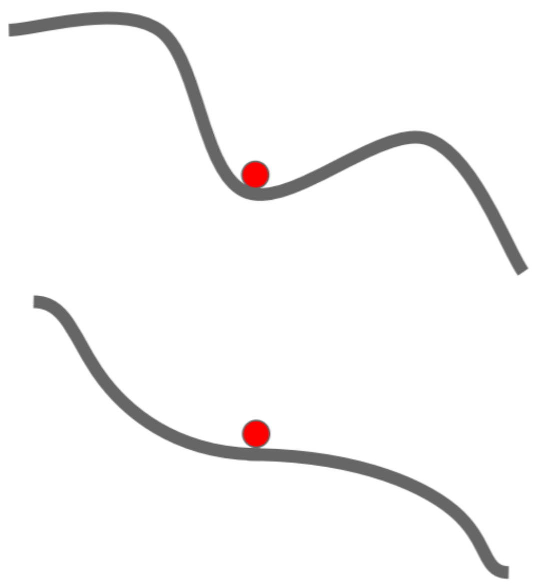
- mini-Batch Gradient Descent

$$x^{(k+1)} = x^{(k)} - t_k \sum_{i=1}^{m'} \nabla f_{i_k}\left(x^{(k)}\right)$$



More than SGD

- What if the loss function has a local minima or saddle point?
- Using SGD + Momentum



SGD

$$x_{t+1} = x_t - \alpha \nabla f(x_t)$$

```
while True:
  dx = compute_gradient(x)
  x -= learning_rate * dx
```

SGD+Momentum

$$v_{t+1} = \rho v_t + \nabla f(x_t)$$
$$x_{t+1} = x_t - \alpha v_{t+1}$$

```
vx = 0
while True:
  dx = compute_gradient(x)
  vx = rho * vx + dx
  x -= learning_rate * vx
```

- Build up "velocity" as a running mean of gradients
- Rho gives "friction"; typically rho=0.9 or 0.99

Adam (Adaptive Moment Estimation)

- AdaGrad/RMSprop + Momentum

```
first_moment = 0
second_moment = 0
for t in range(1, num_iterations):
  dx = compute_gradient(x)
  first_moment = beta1 * first_moment  + (1 - beta1) * dx
  second_moment = beta2 * second_moment + (1 - beta2) * dx * dx
  first_unbias = first_moment / (1 - beta1 ** t)
  second_unbias = second_moment / (1 - beta2 ** t)
  x -= learning_rate * first_unbias / (np.sqrt(second_unbias) + 1e-7))
```

Momentum

Bias correction

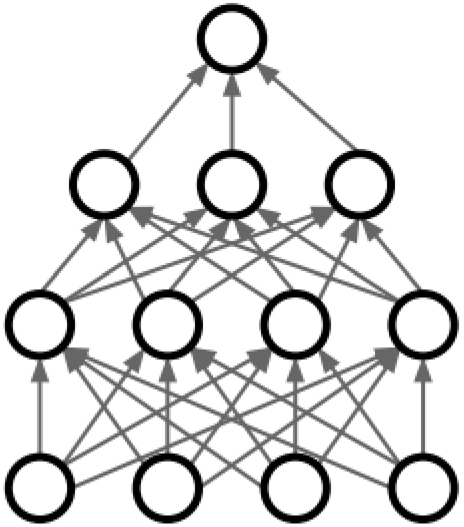
AdaGrad / RMSProp

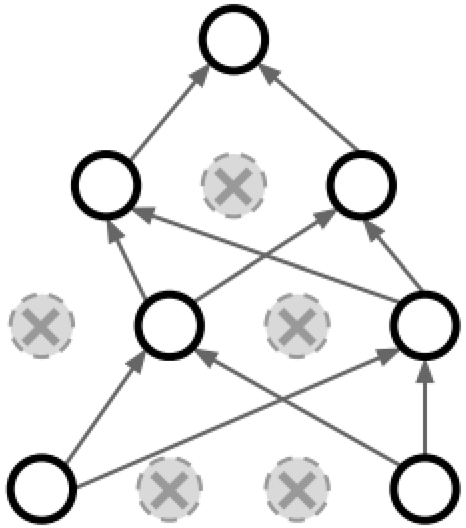
Bias correction for the fact that first and second moment estimates start at zero

Adam with beta1 = 0.9, beta2 = 0.999, and learning_rate = 1e-3 or 5e-4 is a great starting point for many models!

Dropout

- Sometimes the training sample is not enough.
- In each forward pass, randomly set some neurons to zero Probability of dropping is a hyper parameter; 0.5 is common





Another interpretation:

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

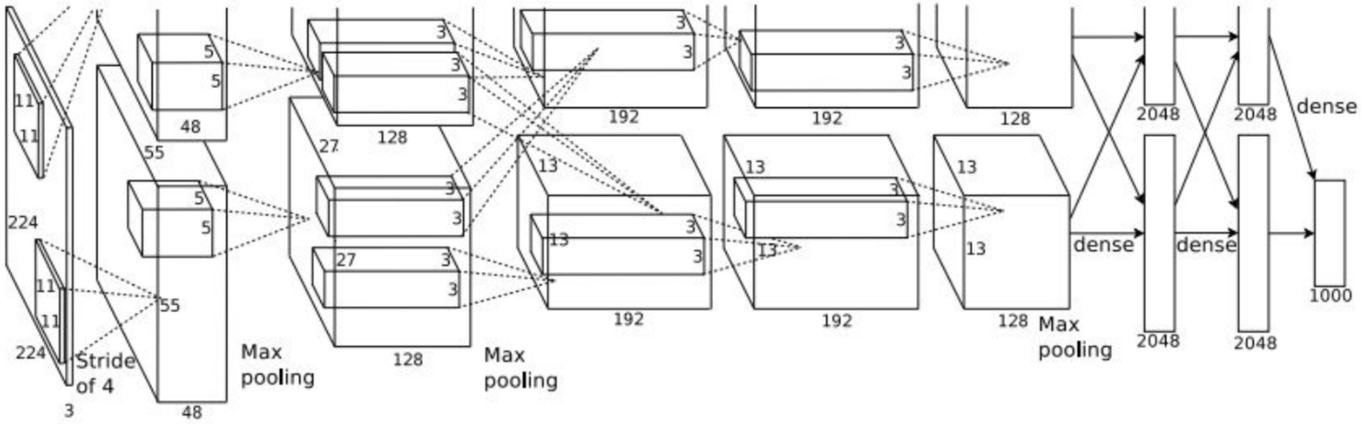
Each binary mask is one model

An FC layer with 4096 units has $2^{4096} \sim 10^{1233}$ possible masks!
Only $\sim 10^{82}$ atoms in the universe...

CNN Case Study

Case Study: AlexNet

[Krizhevsky et al. 2012]



Input: 227x227x3 images

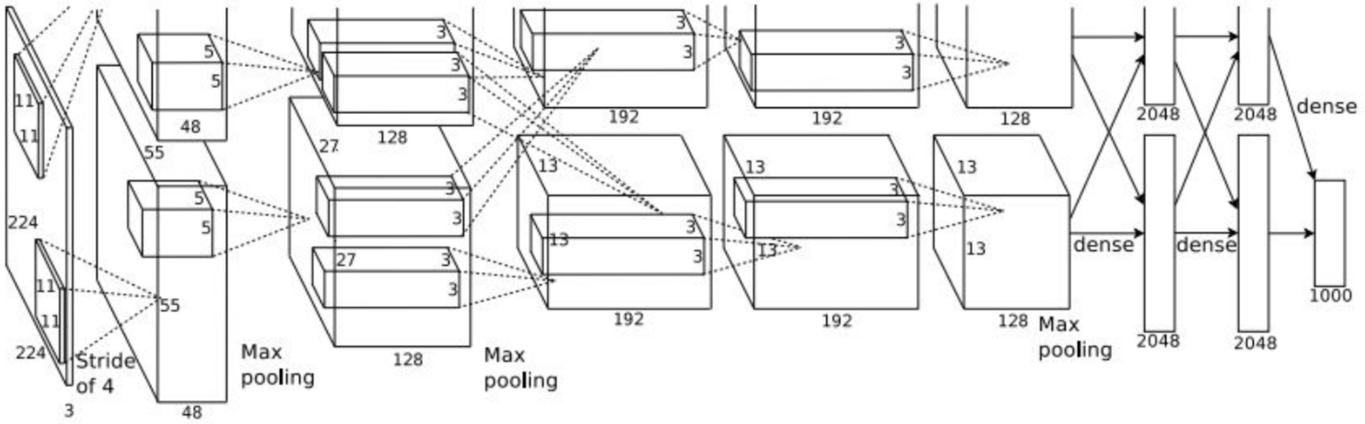
First layer (CONV1): 96 11x11 filters applied at stride 4

=>

Q: what is the output volume size? Hint: (227-11)/4+1 = 55

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%

Case Study: VGGNet

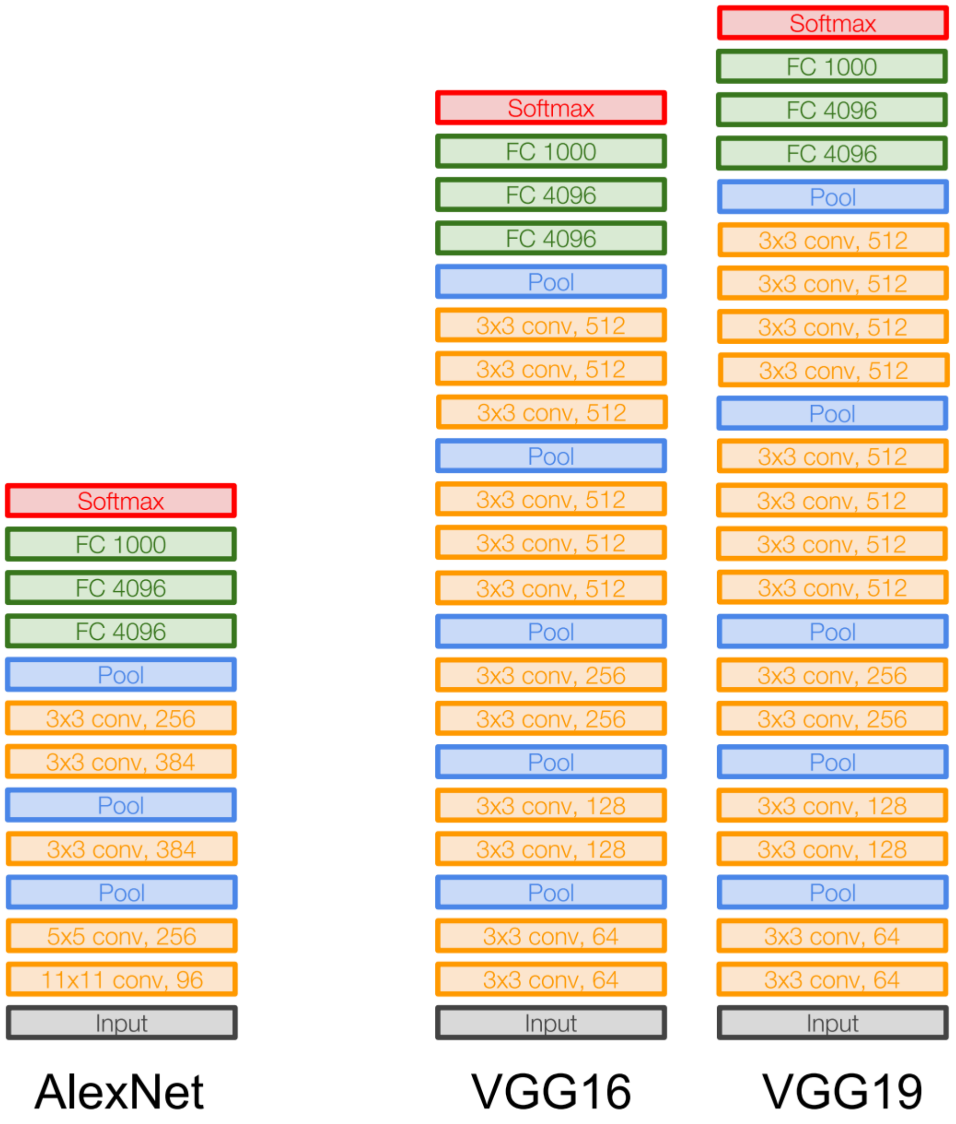
[Simonyan and Zisserman, 2014]

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

But deeper, more non-linearities

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



(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
POOL2: [56x56x128] memory: 56*56*128=400K params: 0
CONV3-256: [56x56x256] memory: 56*56*256=800K params: (3*3*128)*256 = 294,912
CONV3-256: [56x56x256] memory: 56*56*256=800K params: (3*3*256)*256 = 589,824
CONV3-256: [56x56x256] memory: 56*56*256=800K params: (3*3*256)*256 = 589,824
POOL2: [28x28x256] memory: 28*28*256=200K params: 0
CONV3-512: [28x28x512] memory: 28*28*512=400K params: (3*3*256)*512 = 1,179,648
CONV3-512: [28x28x512] memory: 28*28*512=400K params: (3*3*512)*512 = 2,359,296
CONV3-512: [28x28x512] memory: 28*28*512=400K params: (3*3*512)*512 = 2,359,296
POOL2: [14x14x512] memory: 14*14*512=100K params: 0
CONV3-512: [14x14x512] memory: 14*14*512=100K params: (3*3*512)*512 = 2,359,296
CONV3-512: [14x14x512] memory: 14*14*512=100K params: (3*3*512)*512 = 2,359,296
CONV3-512: [14x14x512] memory: 14*14*512=100K params: (3*3*512)*512 = 2,359,296
POOL2: [7x7x512] memory: 7*7*512=25K params: 0
FC: [1x1x4096] 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

Note:

Most memory is in early CONV

Most params are in late FC

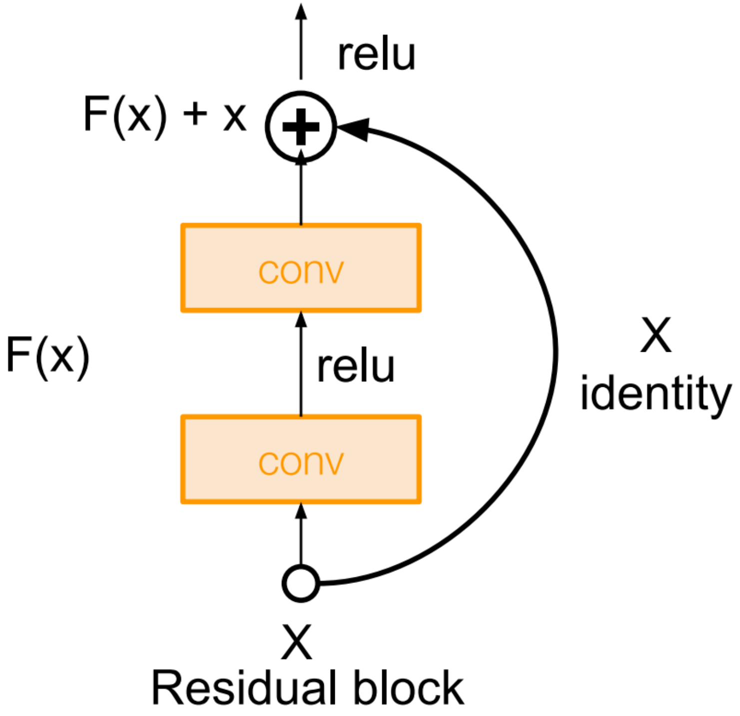
TOTAL memory: 24M * 4 bytes ~= 96MB / image (only forward! ~*2 for bwd)
TOTAL params: 138M parameters

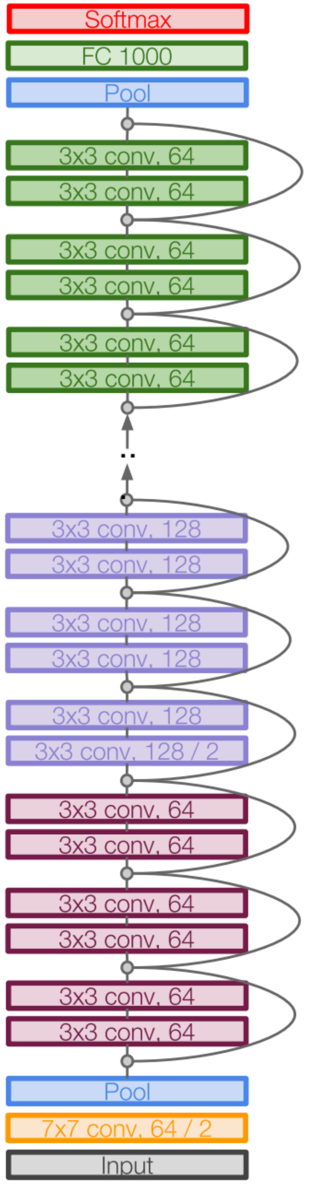
Case Study: ResNet

[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!

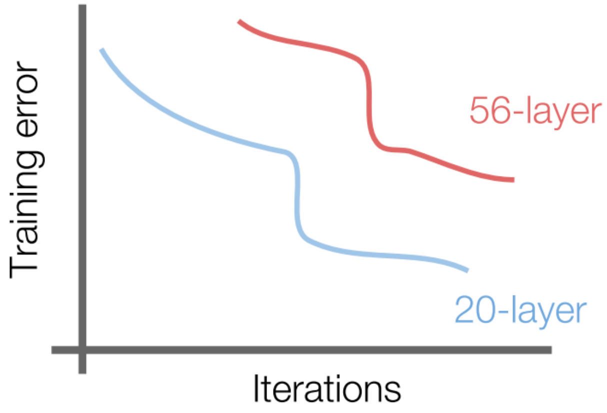




Case Study: ResNet

[He et al., 2015]

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



The deeper model should be able to perform at least as well as the shallower model.

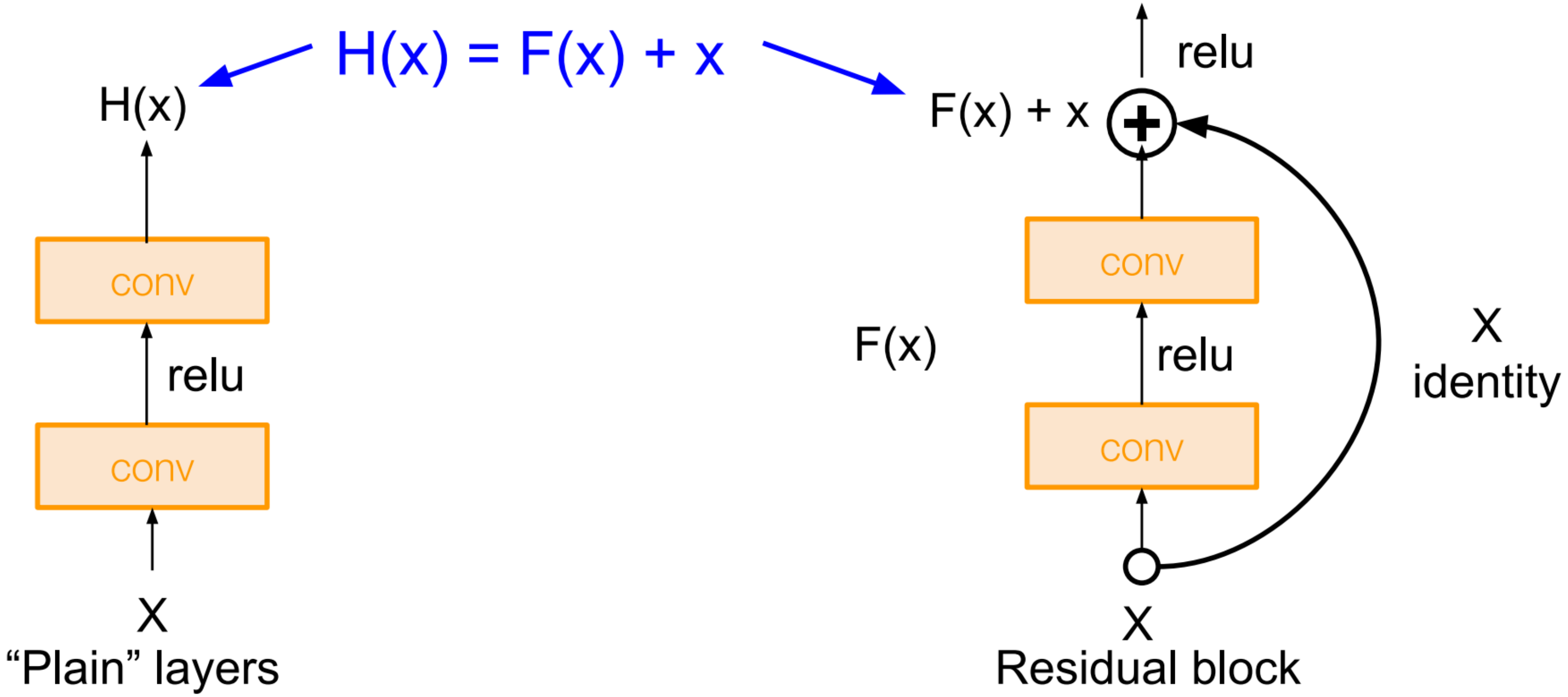
A solution by construction is copying the learned layers from the shallower model and setting additional layers to identity mapping.

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

Case Study: ResNet

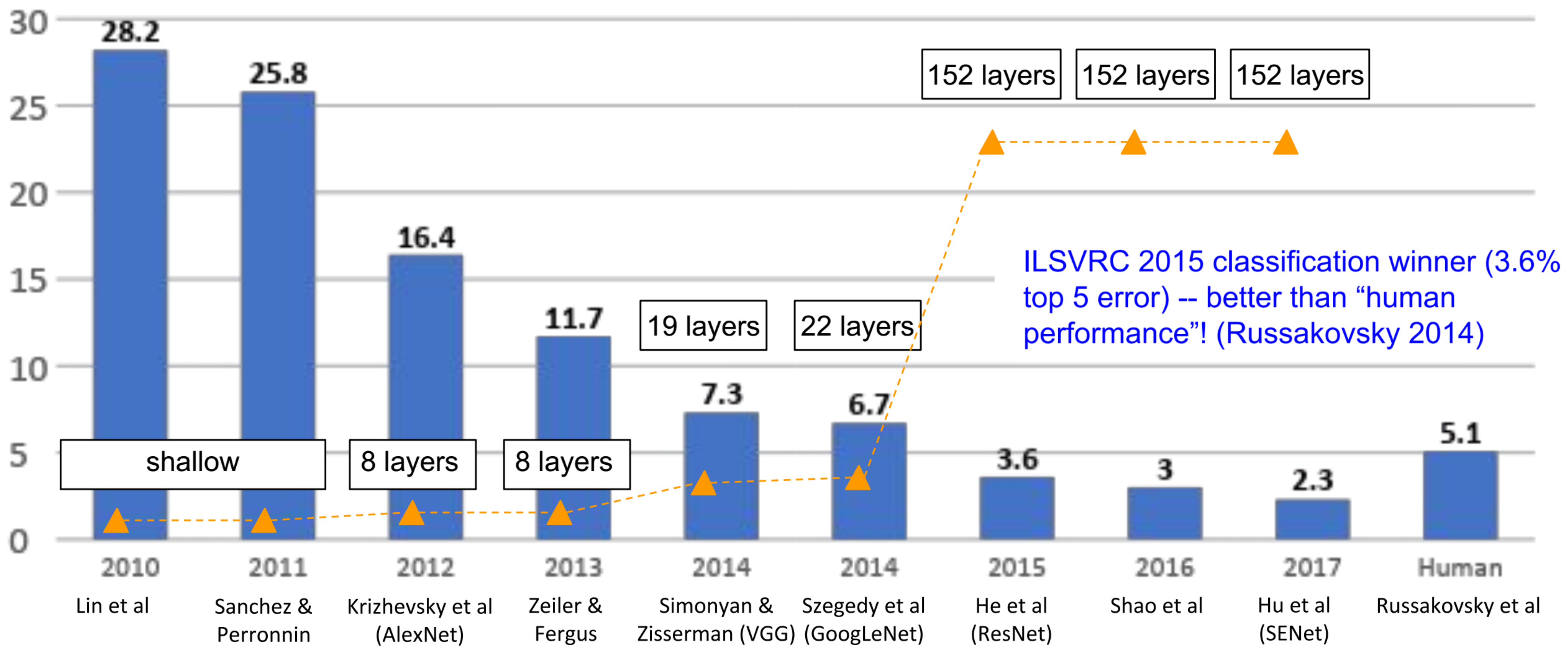
[He et al., 2015]

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



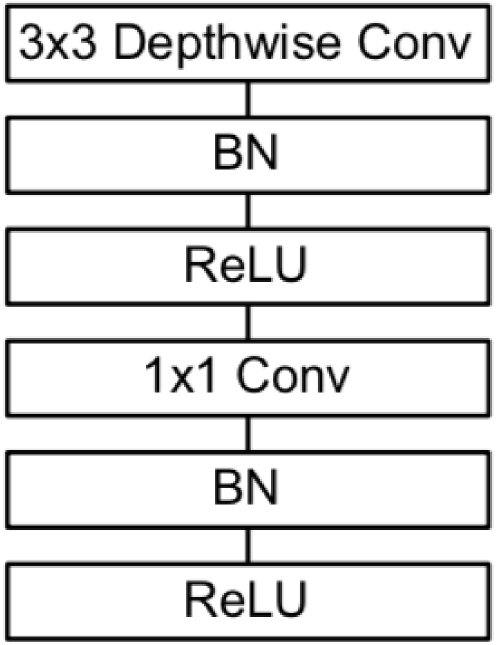
Use layers to fit residual F(x) = H(x) - x instead of H(x) directly

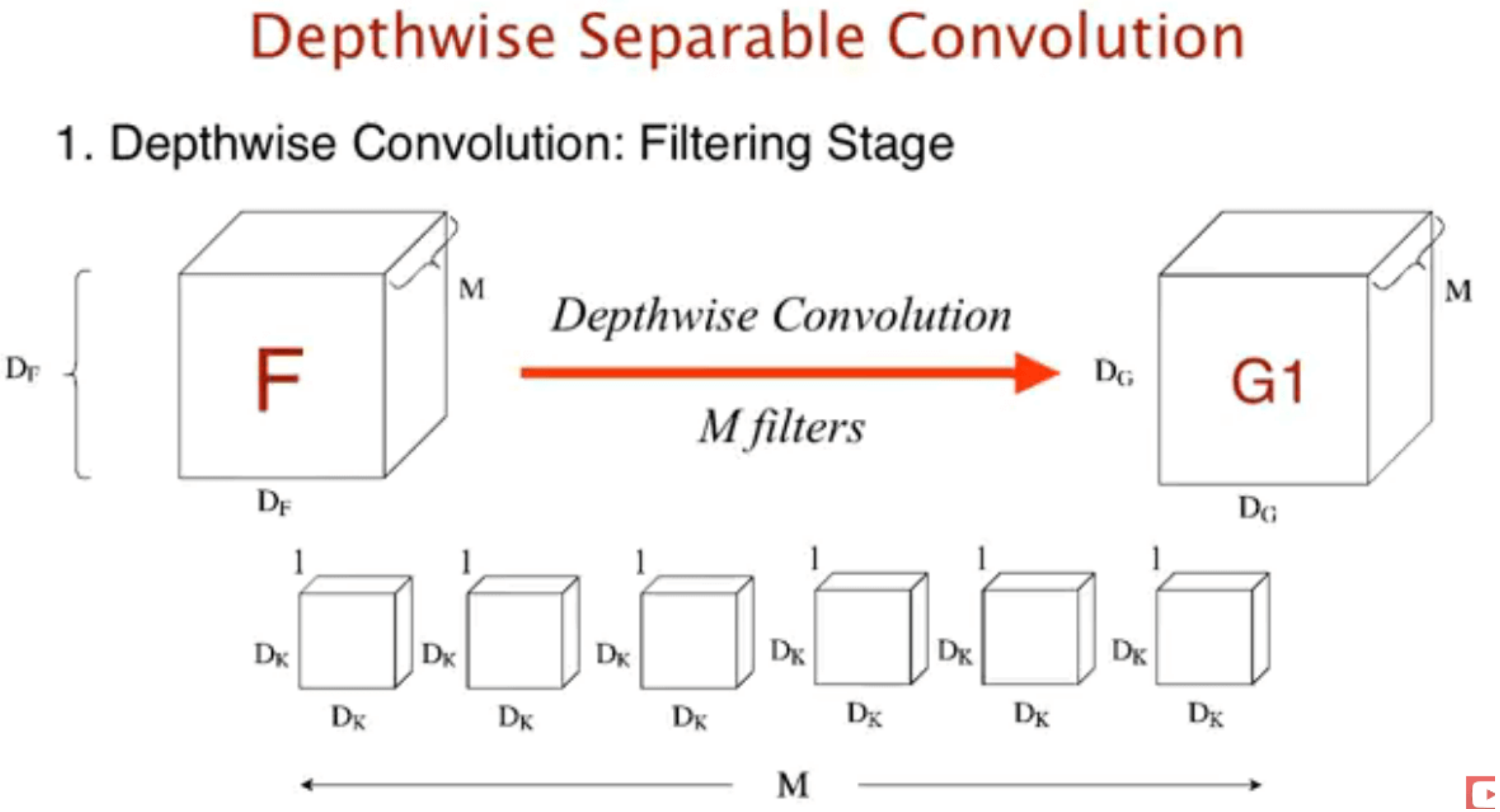
ImageNet Large Scale Visual Recognition Challenge (ILSVRC) winners



MobileNet

- Separable Convolution



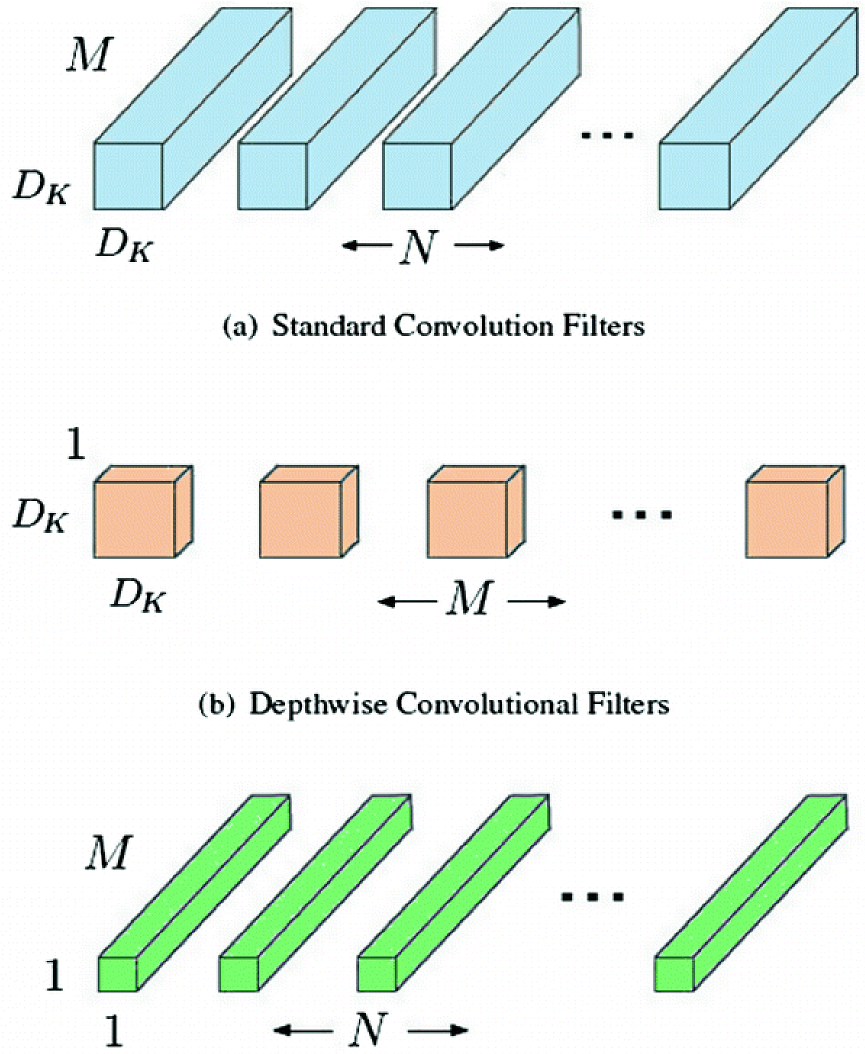


MobileNet

- Very compact for mobile devices

Table 1. MobileNet Body Architecture

Type / Stride	Filter Shape	Input Size
Conv / s2	$3 \times 3 \times 3 \times 32$	$224 \times 224 \times 3$
Conv dw / s1	$3 \times 3 \times 32$ dw	$112 \times 112 \times 32$
Conv / s1	$1 \times 1 \times 32 \times 64$	$112 \times 112 \times 32$
Conv dw / s2	$3 \times 3 \times 64$ dw	$112 \times 112 \times 64$
Conv / s1	$1 \times 1 \times 64 \times 128$	$56 \times 56 \times 64$
Conv dw / s1	$3 \times 3 \times 128$ dw	$56 \times 56 \times 128$
Conv / s1	$1 \times 1 \times 128 \times 128$	$56 \times 56 \times 128$
Conv dw / s2	$3 \times 3 \times 128$ dw	$56 \times 56 \times 128$
Conv / s1	$1 \times 1 \times 128 \times 256$	$28 \times 28 \times 128$
Conv dw / s1	$3 \times 3 \times 256$ dw	$28 \times 28 \times 256$
Conv / s1	$1 \times 1 \times 256 \times 256$	$28 \times 28 \times 256$
Conv dw / s2	$3 \times 3 \times 256$ dw	$28 \times 28 \times 256$
Conv / s1	$1 \times 1 \times 256 \times 512$	$14 \times 14 \times 256$
5× Conv dw / s1	$3 \times 3 \times 512$ dw	$14 \times 14 \times 512$
5× Conv / s1	$1 \times 1 \times 512 \times 512$	$14 \times 14 \times 512$
Conv dw / s2	$3 \times 3 \times 512$ dw	$14 \times 14 \times 512$
Conv / s1	$1 \times 1 \times 512 \times 1024$	$7 \times 7 \times 512$
Conv dw / s2	$3 \times 3 \times 1024$ dw	$7 \times 7 \times 1024$
Conv / s1	$1 \times 1 \times 1024 \times 1024$	$7 \times 7 \times 1024$
Avg Pool / s1	Pool 7×7	$7 \times 7 \times 1024$
FC / s1	1024×1000	$1 \times 1 \times 1024$
Softmax / s1	Classifier	$1 \times 1 \times 1000$



(a) Standard Convolution Filters

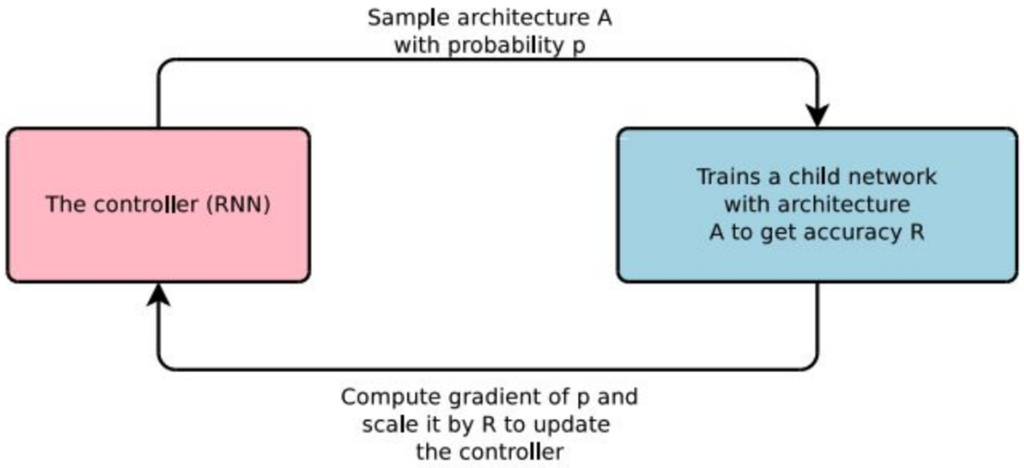
(b) Depthwise Convolutional Filters

(c) 1×1 Convolutional Filters called Pointwise Convolution in the context of Depthwise Separable Convolution

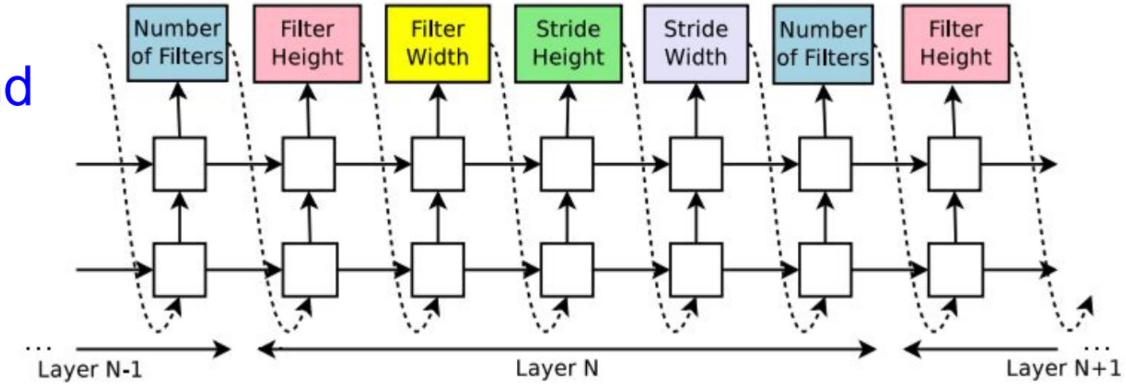
Meta-learning: Learning to learn network architectures...

Neural Architecture Search with Reinforcement Learning (NAS)

[Zoph et al. 2016]



- "Controller" network that learns to design a good network architecture (output a string corresponding to network design)
- Iterate:
  1) Sample an architecture from search space
  2) Train the architecture to get a "reward" R corresponding to accuracy
  3) Compute gradient of sample probability, and scale by R to perform controller parameter update (i.e. increase likelihood of good architecture being sampled, decrease likelihood of bad architecture)

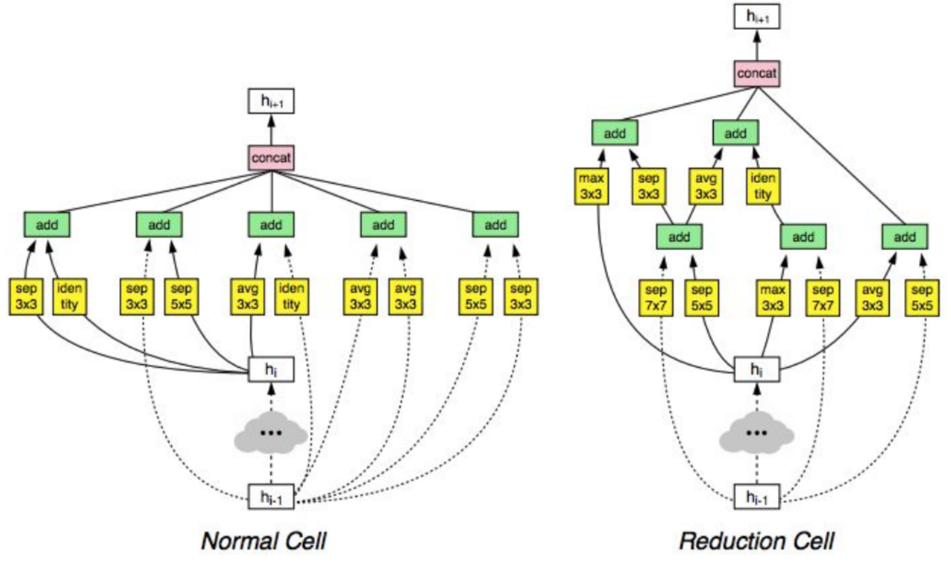


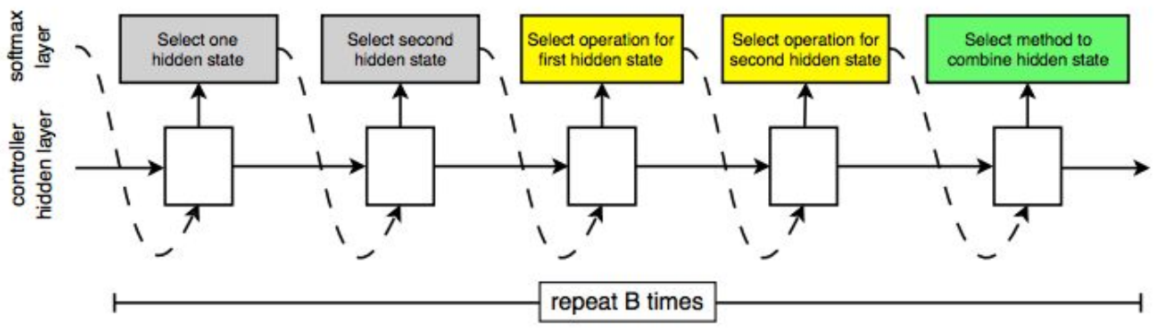
Meta-learning: Learning to learn network architectures...

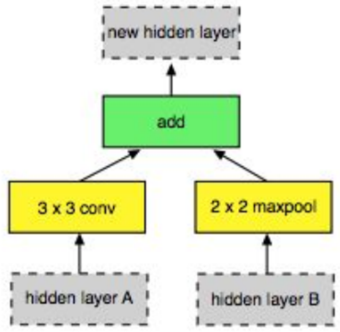
Learning Transferable Architectures for Scalable Image Recognition

[Zoph et al. 2017]

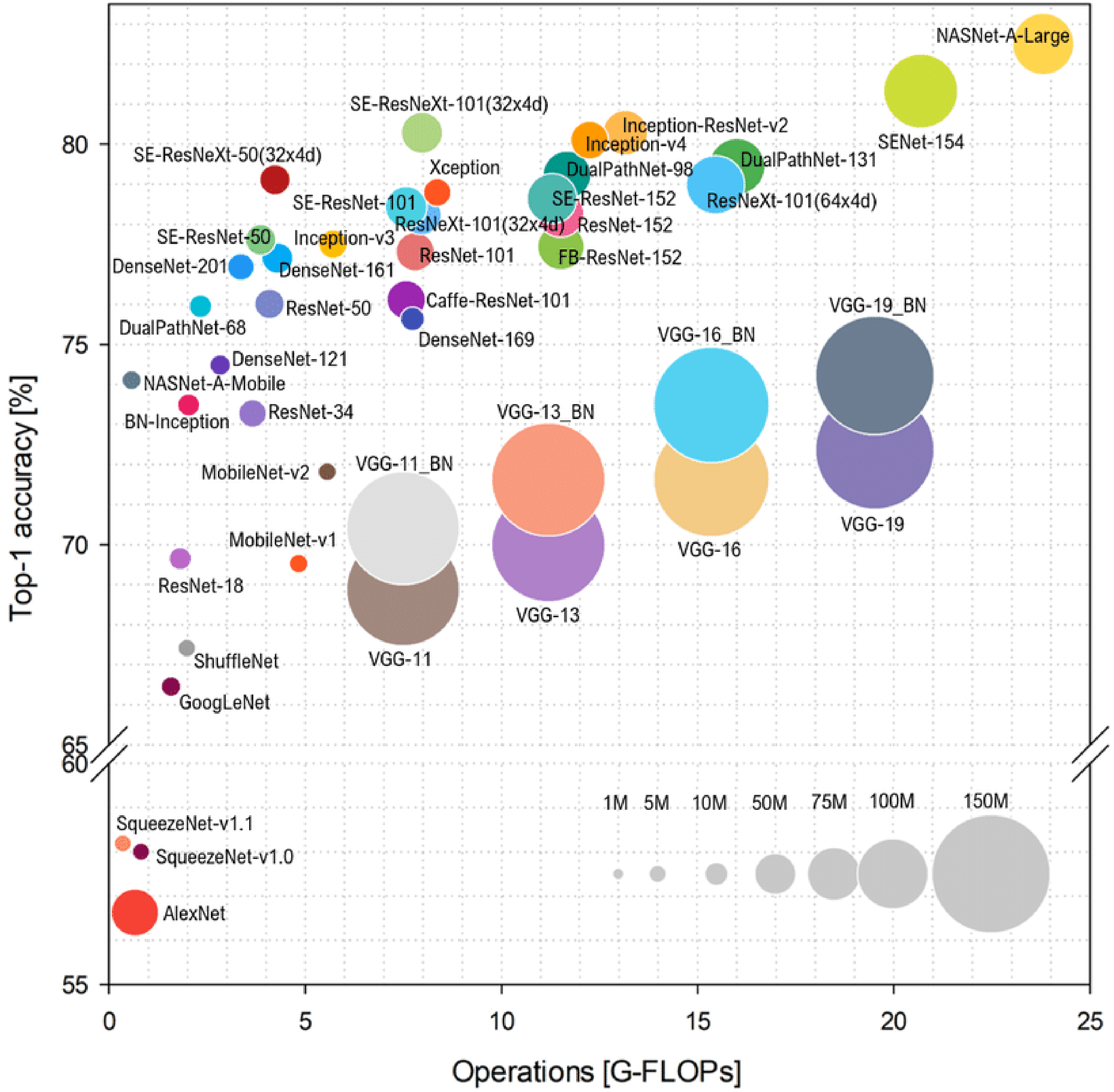
- Applying neural architecture search (NAS) to a large dataset like ImageNet is expensive
- Design a search space of building blocks ("cells") that can be flexibly stacked
- NASNet: Use NAS to find best cell structure on smaller CIFAR-10 dataset, then transfer architecture to ImageNet







Conclusion



Top-1 accuracy [%]
Operations [G-FLOPs]
80
75
70
65
60
55
0
5
10
15
20
25
NASNet-A-Large
SENet-154
SE-ResNeXt-101(32x4d)
Inception-ResNet-v2
Inception-v4
DualPathNet-131
DualPathNet-98
SE-ResNeXt-50(32x4d)
Xception
SE-ResNet-101
SE-ResNet-152
ResNeXt-101(64x4d)
ResNeXt-101(32x4d)
ResNet-152
SE-ResNet-50
Inception-v3
DenseNet-201
DenseNet-161
ResNet-101
FB-ResNet-152
Caffe-ResNet-101
ResNet-50
DualPathNet-68
DenseNet-169
VGG-19_BN
VGG-16_BN
DenseNet-121
NASNet-A-Mobile
ResNet-34
BN-Inception
VGG-13_BN
MobileNet-v2
VGG-11_BN
VGG-19
VGG-16
MobileNet-v1
ResNet-18
VGG-13
ShuffleNet
VGG-11
GoogLeNet
1M
5M
10M
50M
75M
100M
150M
SqueezeNet-v1.1
SqueezeNet-v1.0
AlexNet