SC 4001 CE/CZ 4042 Neural Network and Deep Learning

Last update: 25 Feb 2024

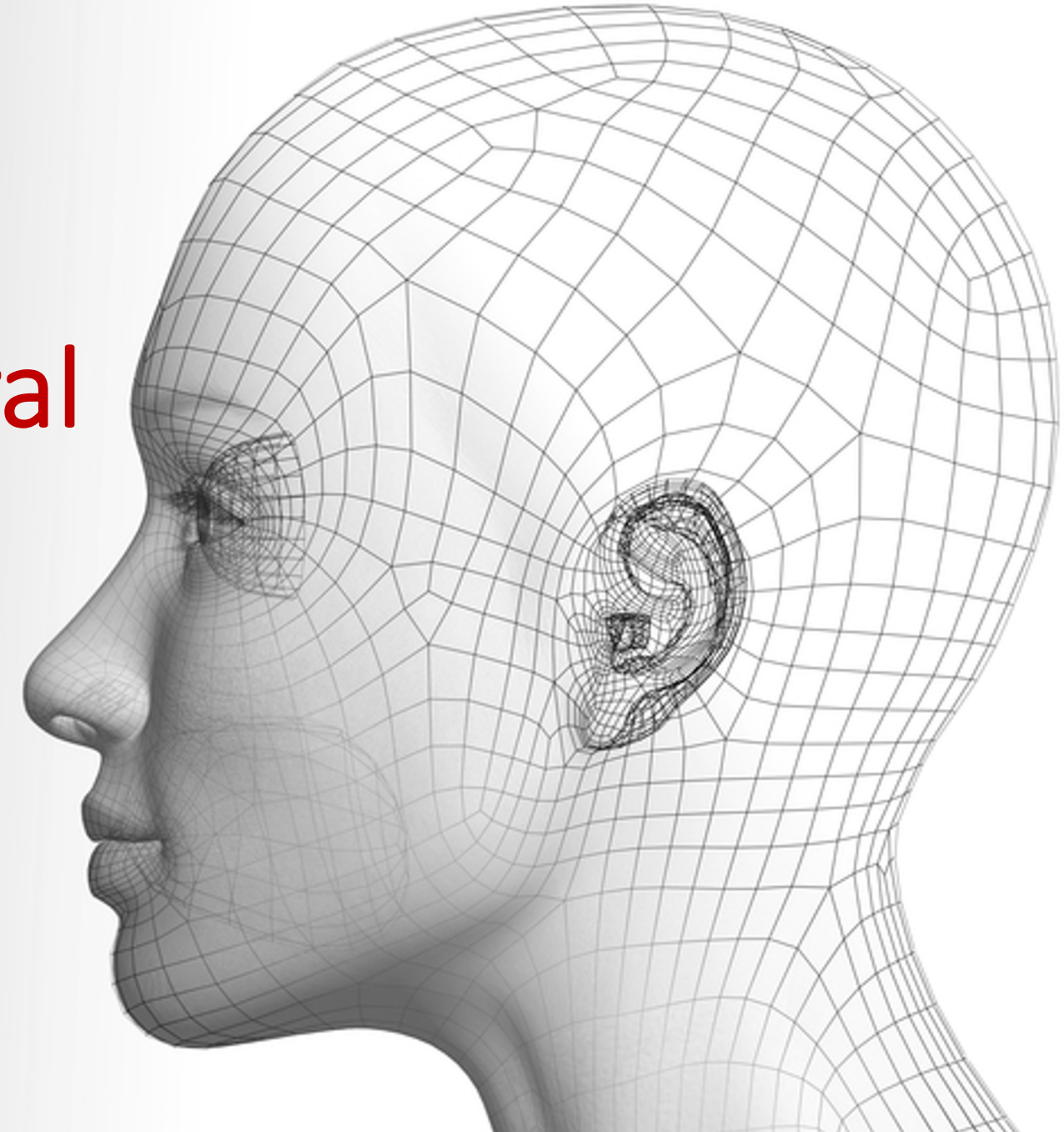
Convolutional Neural Networks II

Xingang Pan

潘新钢

https://xingangpan.github.io/

https://twitter.com/XingangP



Outline

- CNN Architectures

  You learn some classic architectures

- More on convolution
  - How to calculate FLOPs
  - Pointwise convolution
  - Depthwise convolution
  - Depthwise convolution + Pointwise convolution

  You learn how to calculate computation complexity of convolutional layer and how to design a lightweight network

- Batch normalization

  You learn an important technique to improve the training of modern neural networks

- Prevent overfitting
  - Transfer learning
  - Data augmentation

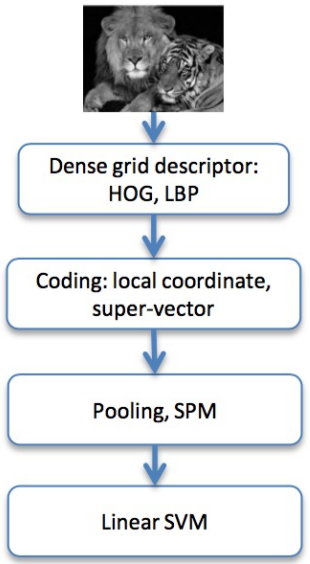
  You learn two important techniques to prevent overfitting in neural networks

CNN Architectures

Deep networks for ImageNet

Year 2010

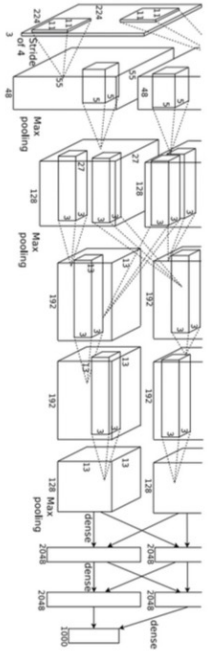
NEC-UIUC



[Lin CVPR 2011]

Year 2012

AlexNet



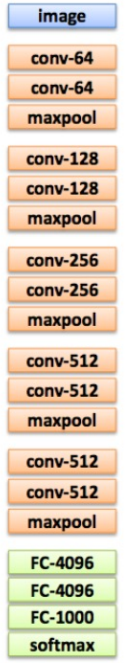
[Krizhevsky NIPS 2012]

Year 2014

GoogLeNet

[Szegedy arxiv 2014]

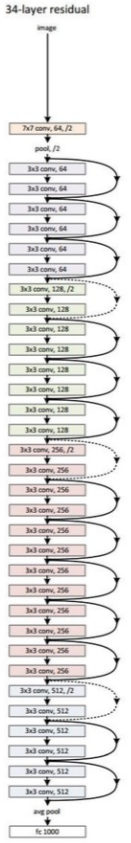
VGG



[Simonyan arxiv 2014]

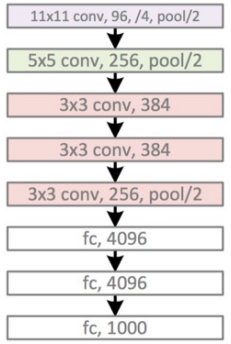
Year 2015

MSRA ResNet

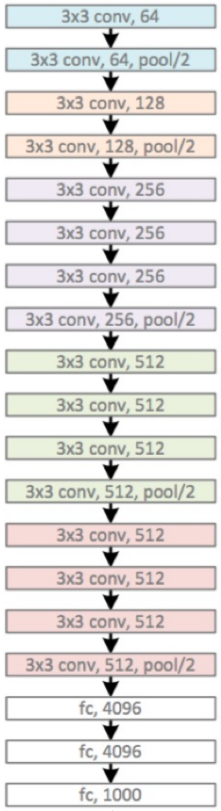


Deep architectures

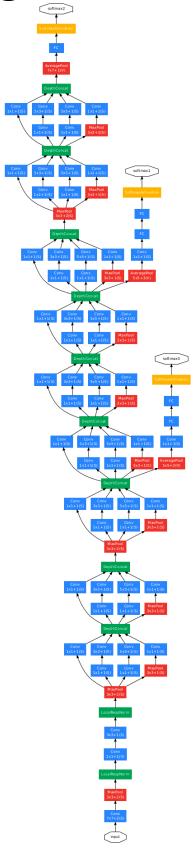
AlexNet 8 layers



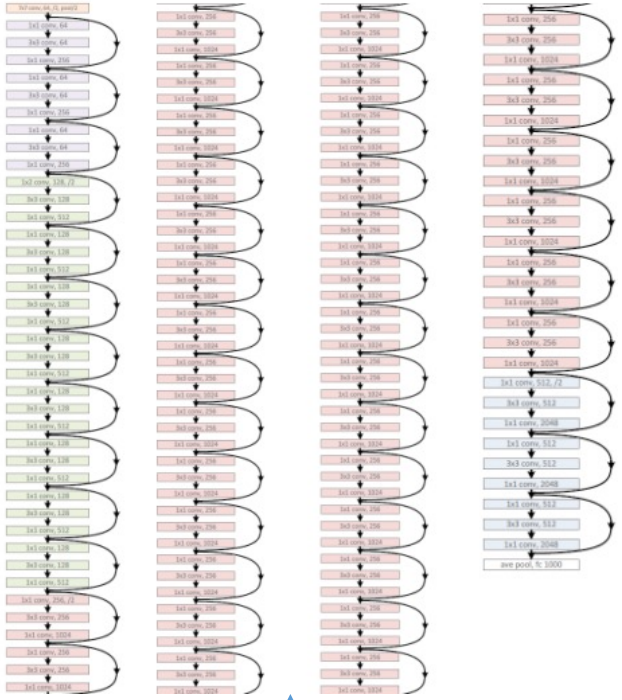
VGG 19 layers



GoogLeNet 22 layers



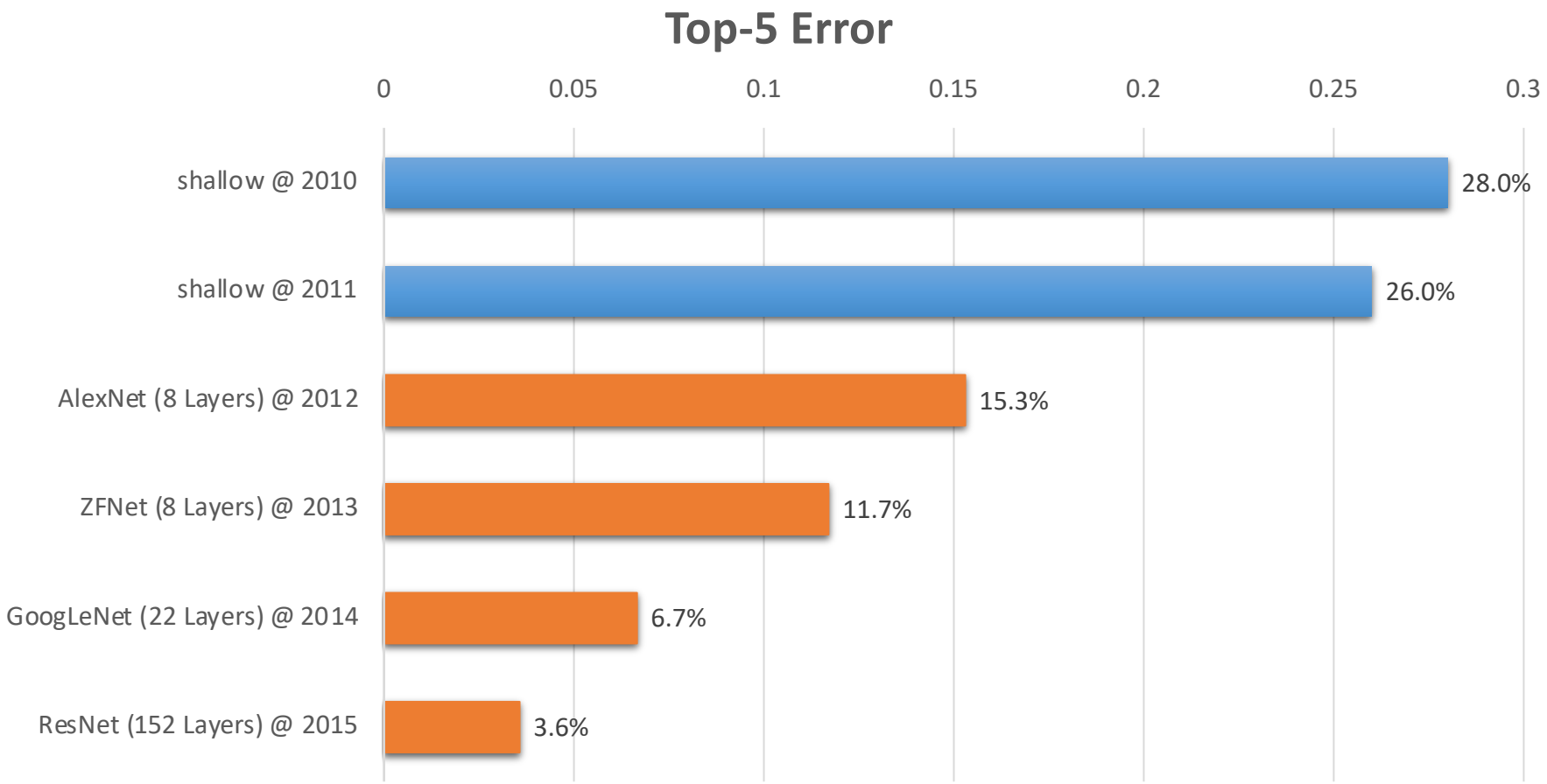
ResNet 152 layers



Deeper
More intricate and different connectivity structures

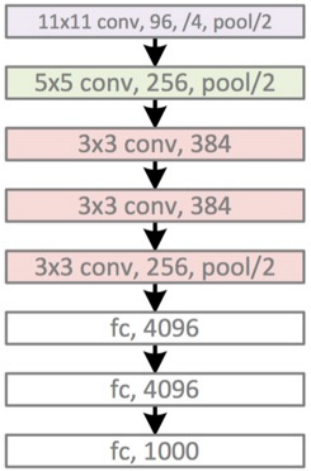
2012 2013 2014 2015 2016 2017 2018 ...

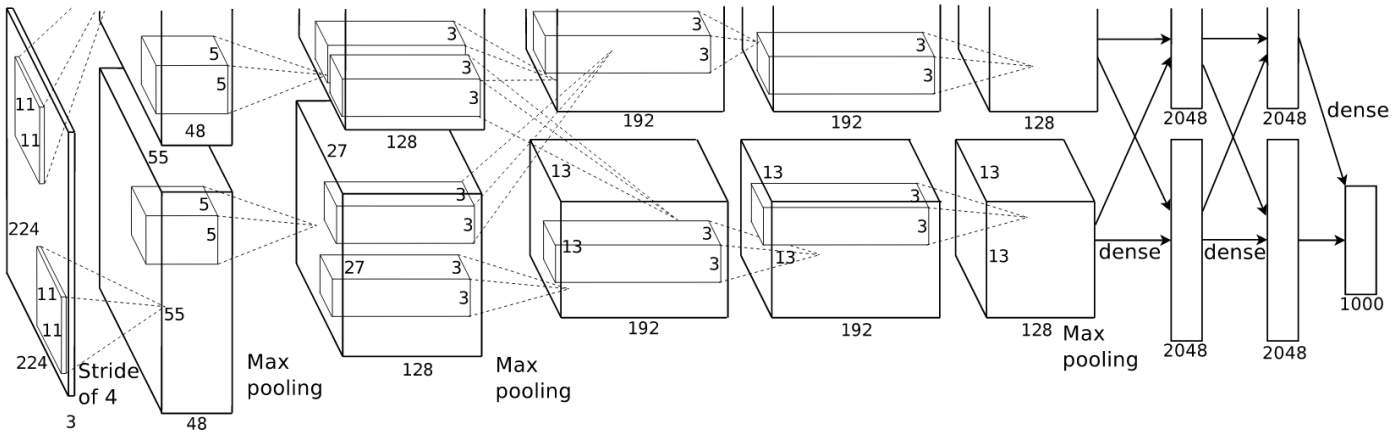
Performance of previous years on ImageNet



Depth is the key to high classification accuracy.

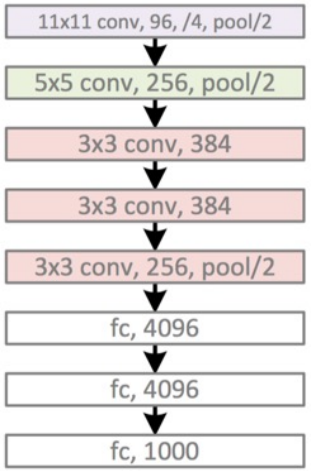
Deep architectures - AlexNet

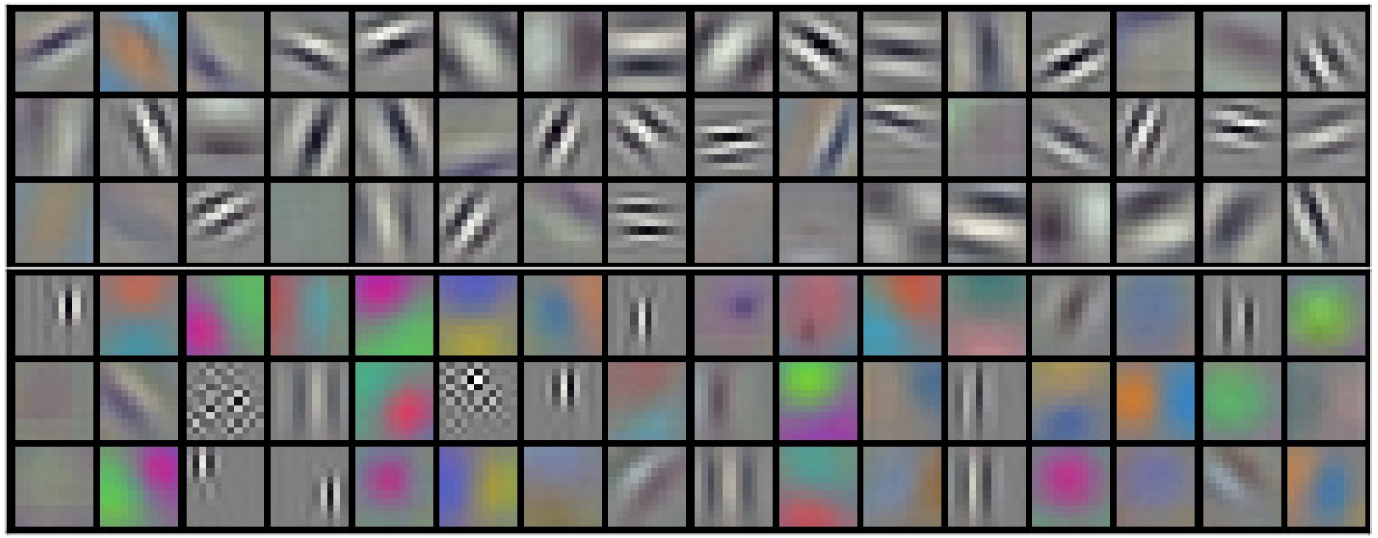




- The split (i.e. two pathways) in the image above are the split between two GPUs.
- Trained for about a week on two NVIDIA GTX 580 3GB GPU
- 60 million parameters
- Input layer: size 227x227x3
- 8 layers deep: 5 convolution and pooling layers and 3 fully connected layers

Deep architectures - AlexNet

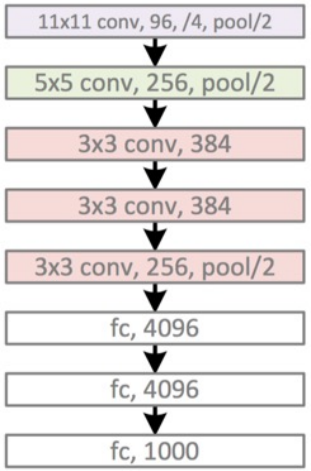


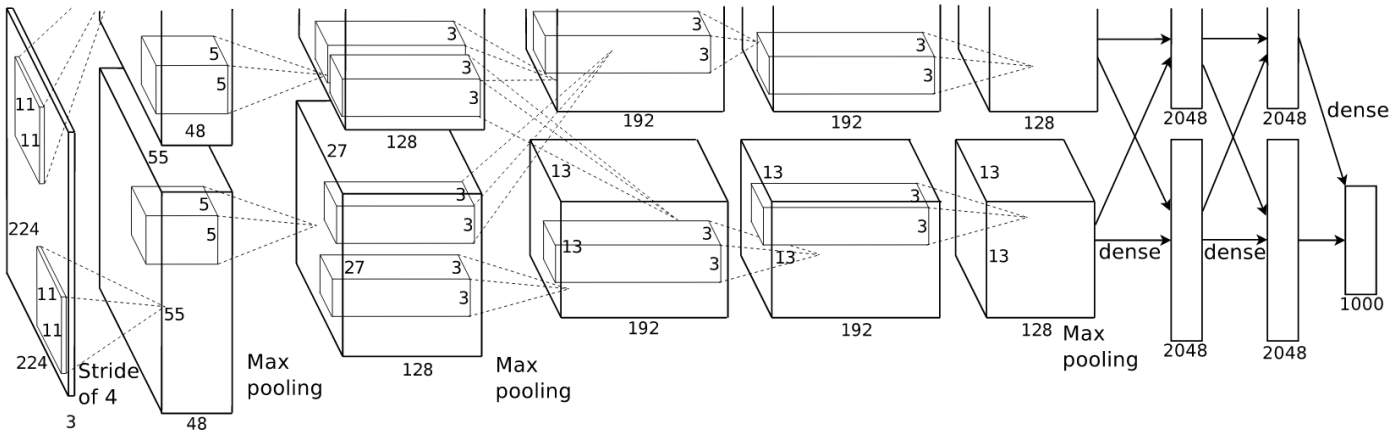


96 kernels learned by first convolution layer; 48 kernels were learned by each GPU

2012 2013 2014 2015 2016 2017 2018 …

Deep architectures - AlexNet

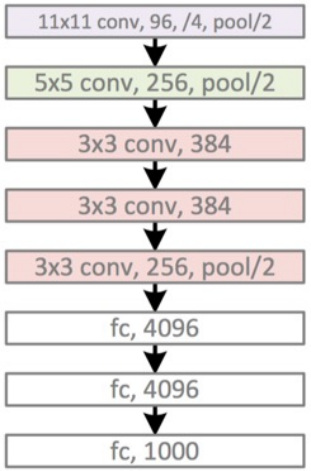




- Escape from a few layers
  - ReLU nonlinearity for solving gradient vanishing
  - Data augmentation
  - Dropout
  - Outperformed all previous models on ILSVRC by 10%

2012 2013 2014 2015 2016 2017 2018 …

Deep architectures - AlexNet

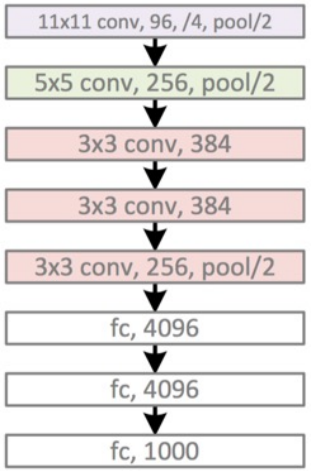


Layer	Weights
L1 (Conv)	
L2 (Conv)	
L3 (Conv)	885,120
L4 (Conv)	1,327,488
L5 (Conv)	884,992
L6 (FC)	
L7 (FC)	16,781,312
L8 (FC)	4,097,000
Conv Subtotal	3,747,200
FC Subtotal	58,631,144
Total	**62,378,344**

First convolution layer: 96 kernels of size 11x11x3, with a stride of 4 pixels

2012 2013 2014 2015 2016 2017 2018 …

Deep architectures - AlexNet



Layer	Weights
L1 (Conv)	34,944
L2 (Conv)	
L3 (Conv)	885,120
L4 (Conv)	1,327,488
L5 (Conv)	884,992
L6 (FC)	
L7 (FC)	16,781,312
L8 (FC)	4,097,000
Conv Subtotal	3,747,200
FC Subtotal	58,631,144
Total	**62,378,344**

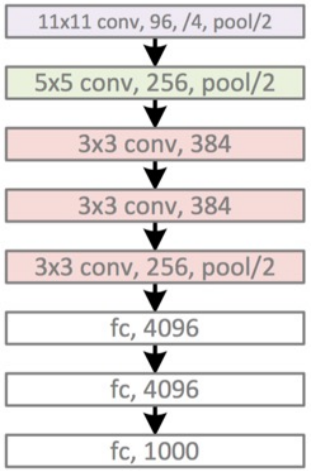
First convolution layer: 96 kernels of size 11x11x3, with a stride of 4 pixels

Number of parameters = (11x11x3 + 1)*96 = 34,944

Note: There are no parameters associated with a pooling layer. The pool size, stride, and padding are hyperparameters.

2012 **2013** **2014** **2015** **2016** **2017** **2018 …**

Deep architectures - AlexNet

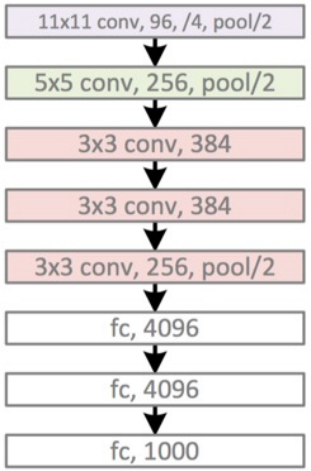


Layer	Weights
L1 (Conv)	34,944
L2 (Conv)	
L3 (Conv)	885,120
L4 (Conv)	1,327,488
L5 (Conv)	884,992
L6 (FC)	
L7 (FC)	16,781,312
L8 (FC)	4,097,000
Conv Subtotal	3,747,200
FC Subtotal	58,631,144
Total	**62,378,344**

← Second convolution layer: 256 kernels of size 5x5x96

2012 **2013** **2014** **2015** **2016** **2017** **2018 …**

Deep architectures - AlexNet



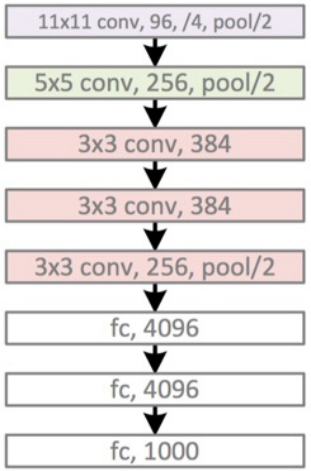
Layer	Weights
L1 (Conv)	34,944
L2 (Conv)	614,656
L3 (Conv)	885,120
L4 (Conv)	1,327,488
L5 (Conv)	884,992
L6 (FC)	
L7 (FC)	16,781,312
L8 (FC)	4,097,000
Conv Subtotal	3,747,200
FC Subtotal	58,631,144
Total	**62,378,344**

Second convolution layer: 256 kernels of size 5x5x96

Number of parameters = (5x5x96 + 1)*256 = 614,656

2012 2013 2014 2015 2016 2017 2018 …

Deep architectures - AlexNet

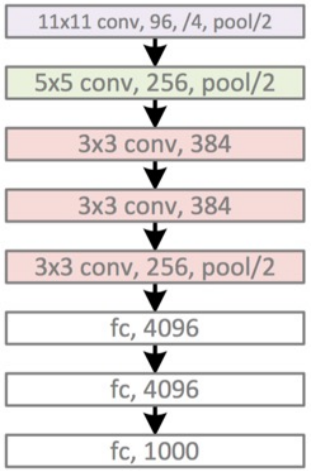


Layer	Weights
L1 (Conv)	34,944
L2 (Conv)	614,656
L3 (Conv)	885,120
L4 (Conv)	1,327,488
L5 (Conv)	884,992
L6 (FC)	
L7 (FC)	16,781,312
L8 (FC)	4,097,000
Conv Subtotal	3,747,200
FC Subtotal	58,631,144
Total	**62,378,344**

First FC layer:
Number of neurons = 4096
Number of kernels in the previous Conv Layer = 256
Size (width) of the output image of the previous Conv Layer = 6

2012 2013 2014 2015 2016 2017 2018 …

Deep architectures - AlexNet

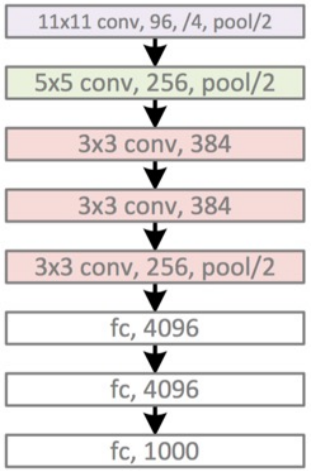


Layer	Weights
L1 (Conv)	34,944
L2 (Conv)	614,656
L3 (Conv)	885,120
L4 (Conv)	1,327,488
L5 (Conv)	884,992
L6 (FC)	37,752,832
L7 (FC)	16,781,312
L8 (FC)	4,097,000
Conv Subtotal	3,747,200
FC Subtotal	58,631,144
Total	**62,378,344**

First FC layer:
Number of neurons = 4096
Number of kernels in the previous Conv Layer = 256
Size (width) of the output image of the previous Conv Layer = 6

Number of parameters = (6x6x256*4096)+4096 = 37,752,832

Deep architectures - AlexNet



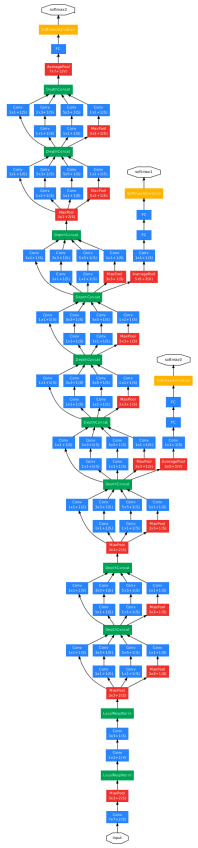
Layer	Weights
L1 (Conv)	34,944
L2 (Conv)	614,656
L3 (Conv)	885,120
L4 (Conv)	1,327,488
L5 (Conv)	884,992
L6 (FC)	37,752,832
L7 (FC)	16,781,312
L8 (FC)	4,097,000
Conv Subtotal	3,747,200
FC Subtotal	58,631,144
Total	**62,378,344**

The last FC layer:
Number of neurons = 1000
Number of neurons in the previous FC Layer = 4096

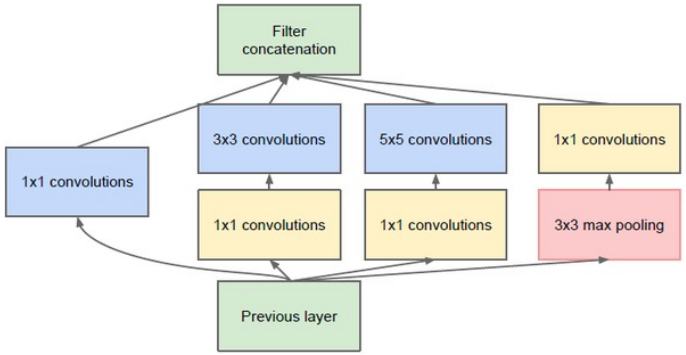
Number of parameters = (1000*4096)+1000 = 4,097,000

2012 2013 2014 2015 2016 2017 2018 …

Deep architectures - GoogLeNet



- An important lesson - go deeper
- Inception structures (v2, v3, v4)
  - Reduce parameters (4M vs 60M in AlexNet)

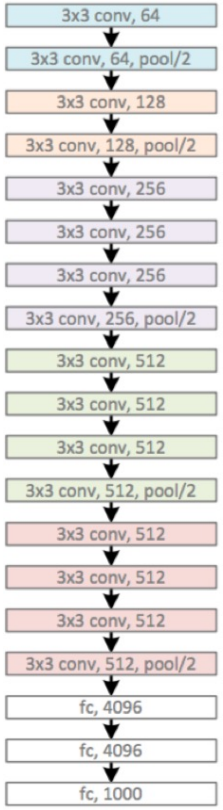


The 1x1 convolutions are performed to reduce the dimensions of input/output

- Batch normalization
  - Normalization the activation for each training mini-batch
  - Allows us to use much higher learning rates and be less careful about initialization

2012 2013 2014 2015 2016 2017 2018 …

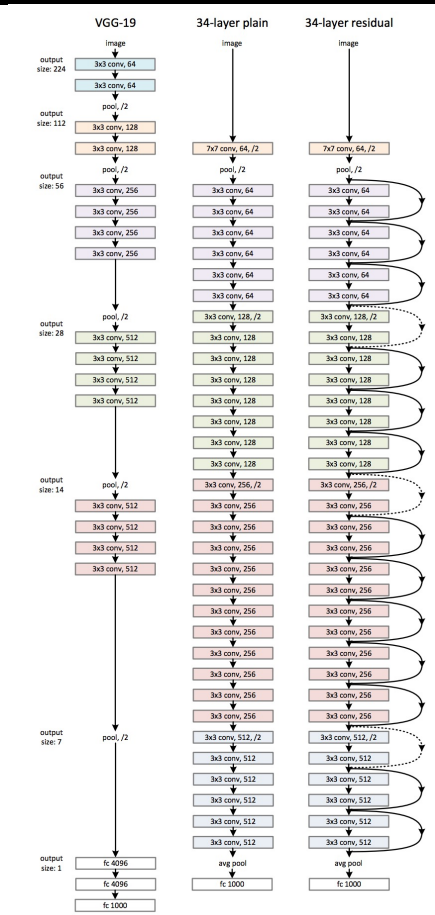
Deep architectures - VGG



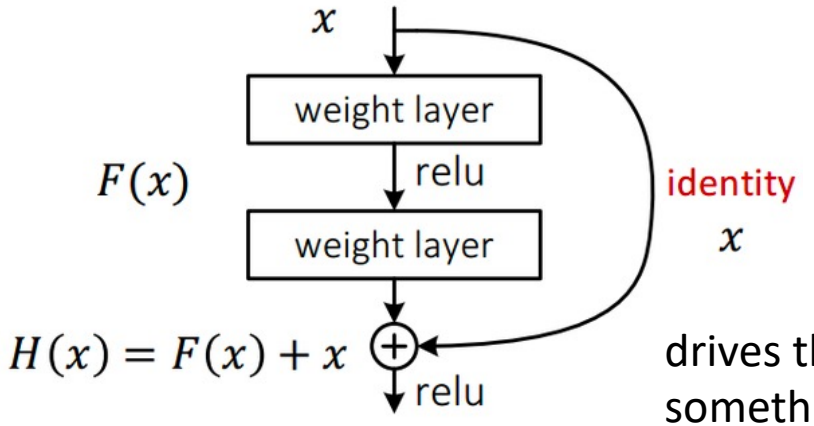
- An important lesson - go deeper
- 140M parameters
- Now commonly used for computing perceptual loss

2012 2013 2014 2015 2016 2017 2018 …

Deep architectures - ResNet



- An important lesson - go deeper
- Escape from 100 layers
  - Residual learning

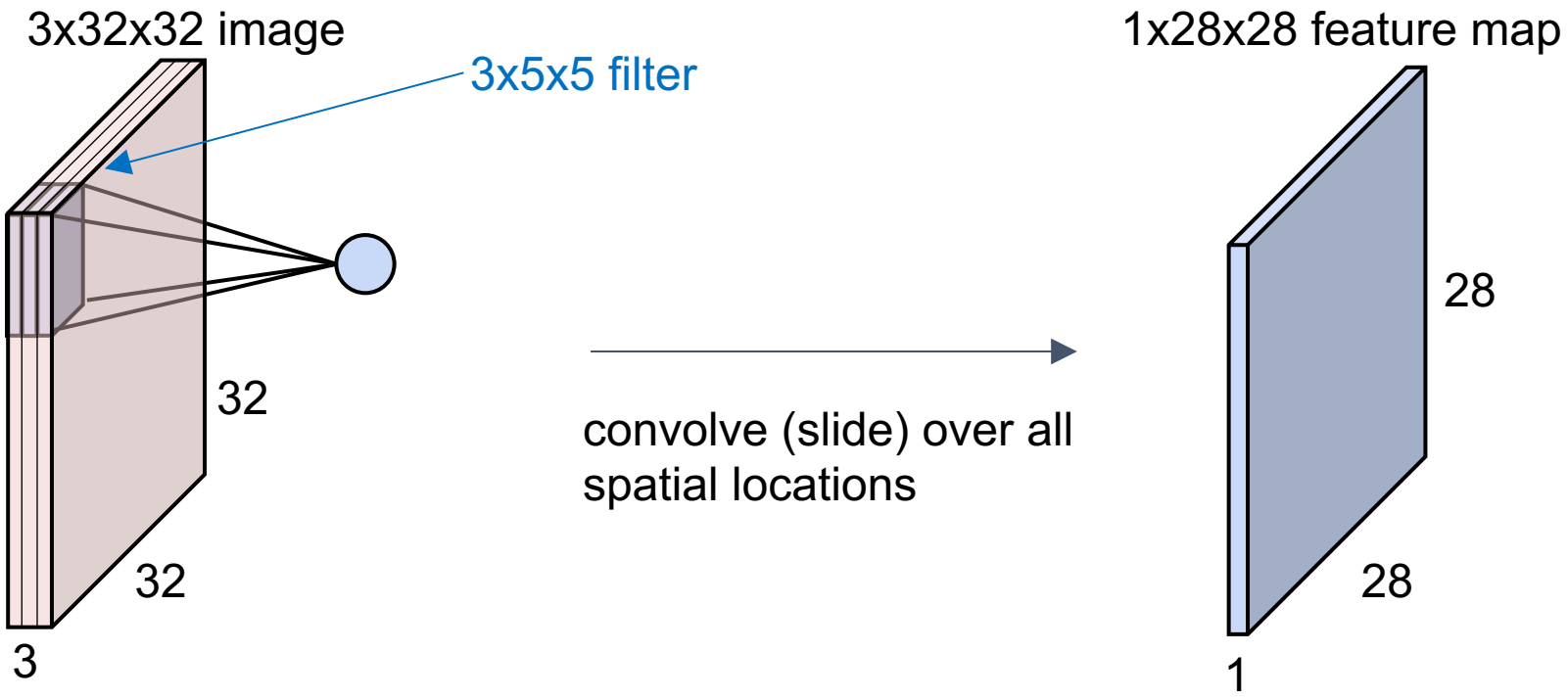


drives the new layer to learn something different

2012 2013 2014 2015 2016 2017 2018 …

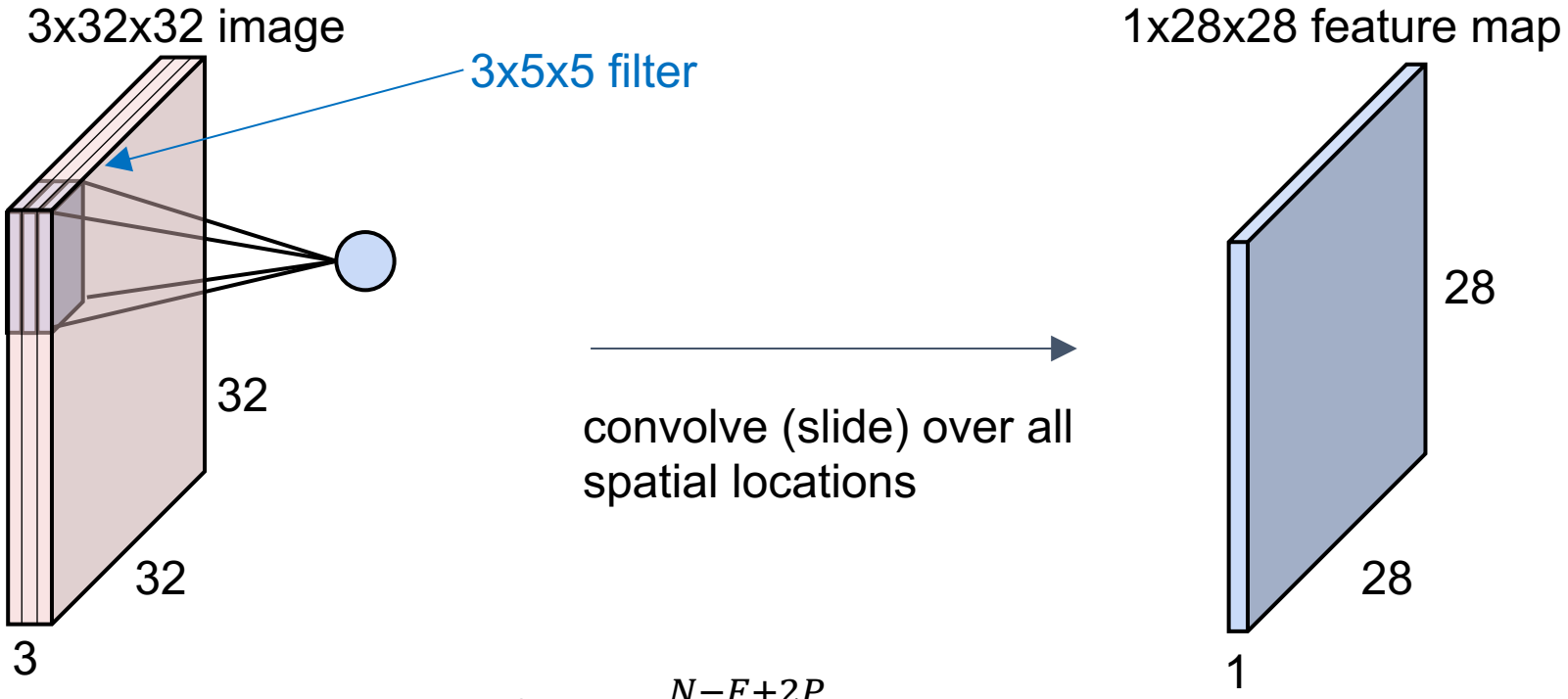
More on Convolution

Standard convolution



- 😀 How to calculate the spatial size of output? Lecture 6
- 😀 How to calculate the number of parameters? Lecture 6
- 😅 How to calculate the computations involved?

Recap: How to calculate the spatial size of output?

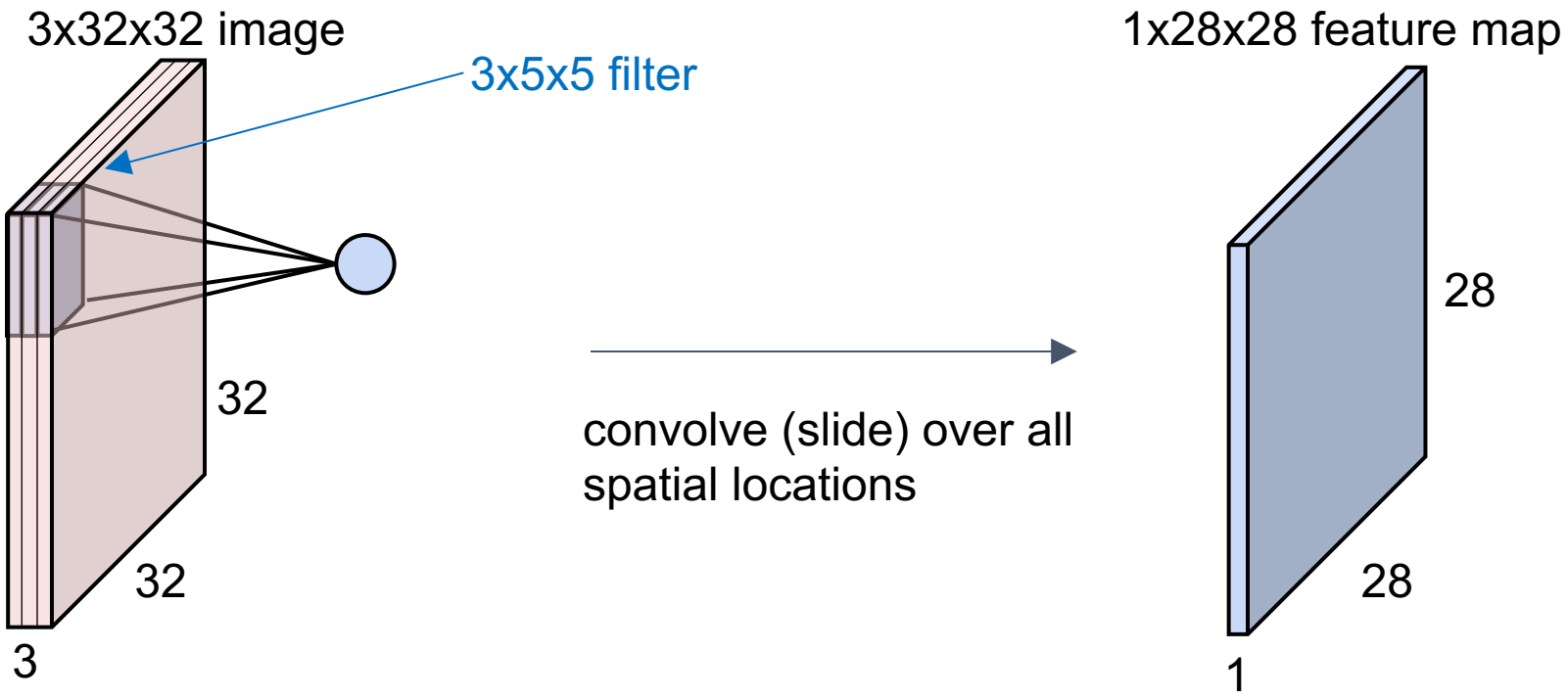


Output spatial size = $\frac{N-F+2P}{S} + 1$

In this example, $N = 32, F = 5, P = 0, S = 1$

Thus, output spatial size = $\frac{32-5+2(0)}{1} + 1 = 28$

Recap: How to calculate the number of parameters?

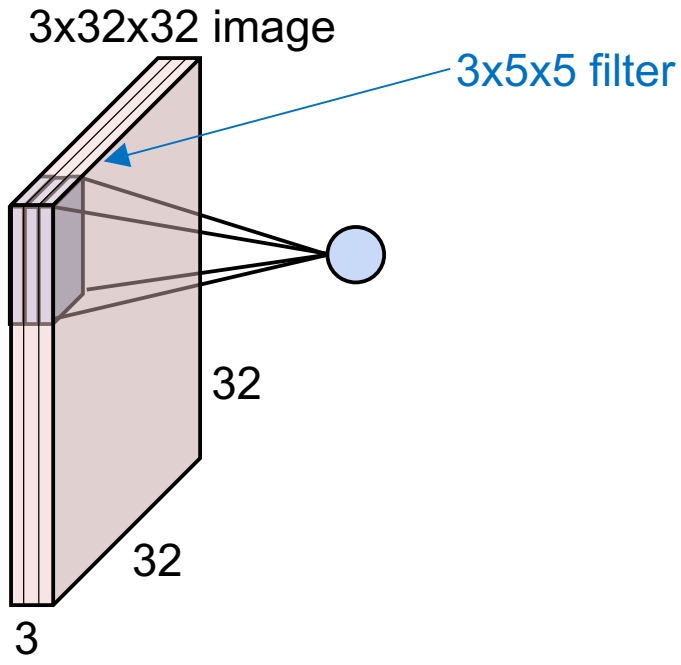


Let say we have ten 3×5×5 filters with stride 1, pad 0

Recap: How to calculate the number of parameters?

Input volume: $3 \times 32 \times 32$
Ten $3 \times 5 \times 5$ filters with stride 1, pad 0

Number of parameters in this layer: ?



Each filter has $5 \times 5 \times 3 + 1 = 76$ params (+1 for bias)

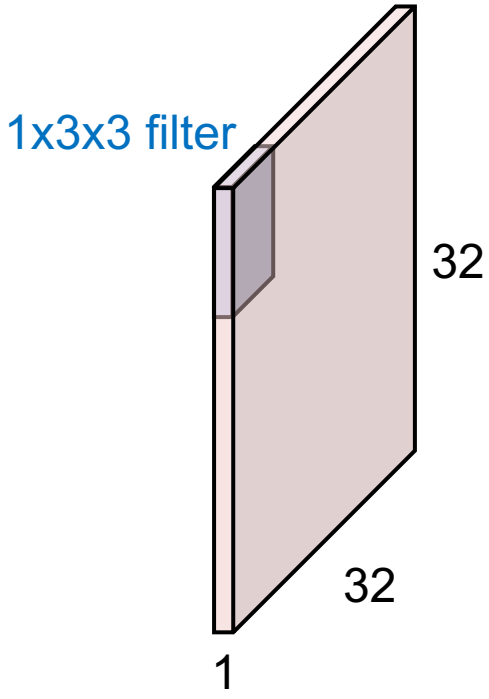
Ten filters so $76 \times 10 =$ **760 parameters**

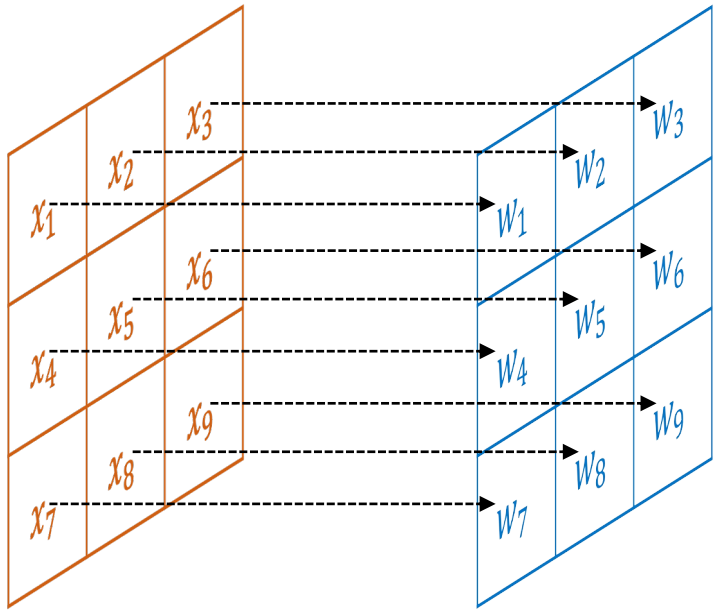
How to calculate the computations involved?

Let's focus on one input channel first

$$u(i,j) = \sum_{l=-a}^{a} \sum_{m=-b}^{b} x(i+l, j+m)w(l,m) + b$$

Element-wise multiplication of input and weights





$x_1 w_1$

$x_2 w_2$

$x_3 w_3$

$x_4 w_4$

$x_5 w_5$

$x_6 w_6$

$x_7 w_7$

$x_8 w_8$

$x_9 w_9$

How many multiplication operations?

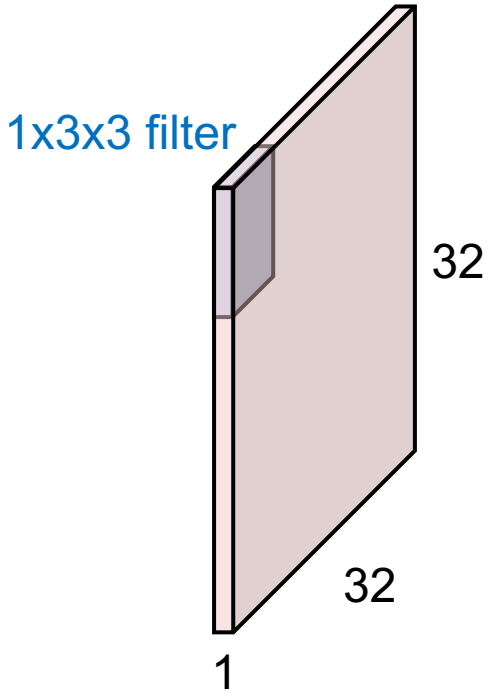
In general, there are F^2 multiplication operations, where F is the filter spatial size

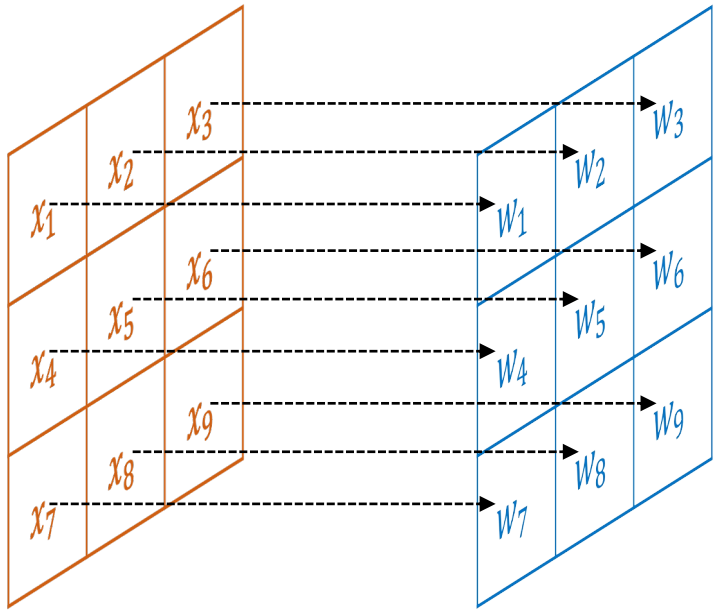
How to calculate the computations involved?

Let's focus on one input channel first

$$u(i,j) = \sum_{l=-a}^{a} \sum_{m=-b}^{b} x(i+l, j+m)w(l,m) + b$$

Element-wise multiplication of input and weights





x_1w_1
+
x_2w_2
+
x_3w_3
+
x_4w_4
+
x_5w_5
+
x_6w_6
+
x_7w_7
+
x_8w_8
+
x_9w_9

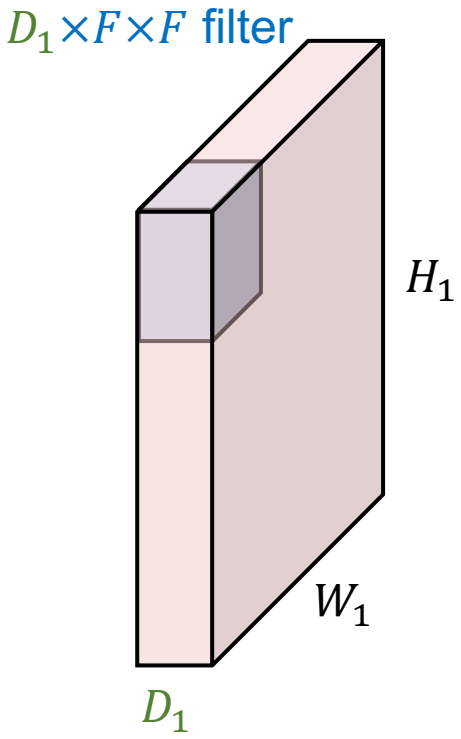
How many adding operations?

Adding the elements after multiplication, we need $n-1$ adding operations for n elements

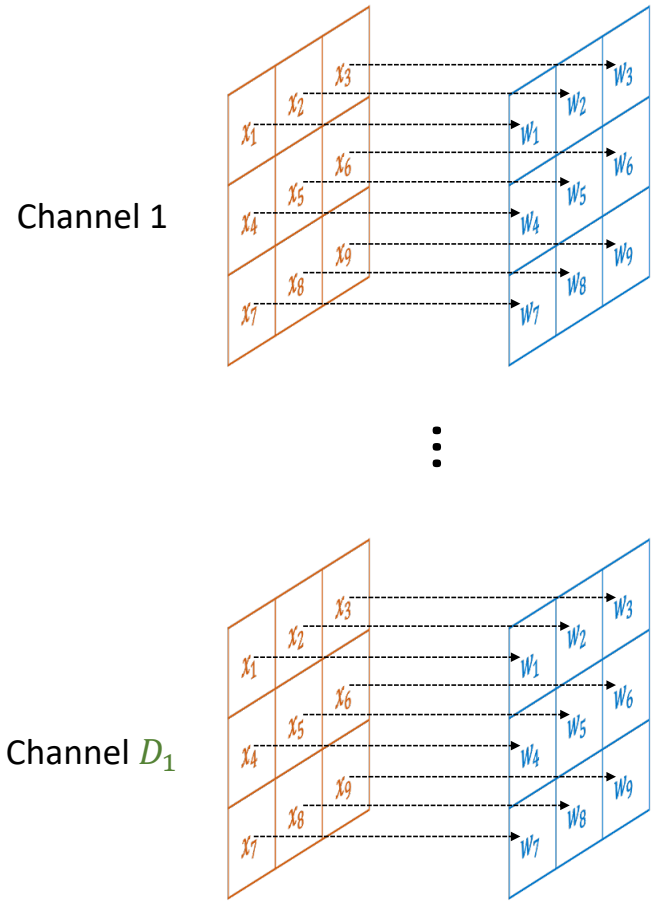
In general, there are F^2-1 adding operations, where F is the filter spatial size

How to calculate the computations involved?

Let's say we have D_1 input channels now



Element-wise multiplication of input and weights, for each channel



F^2 multiplication operations
$F^2 - 1$ adding operations

F^2 multiplication operations
$F^2 - 1$ adding operations

If we have only one filter, and apply it to generate output of one spatial location, the total operations

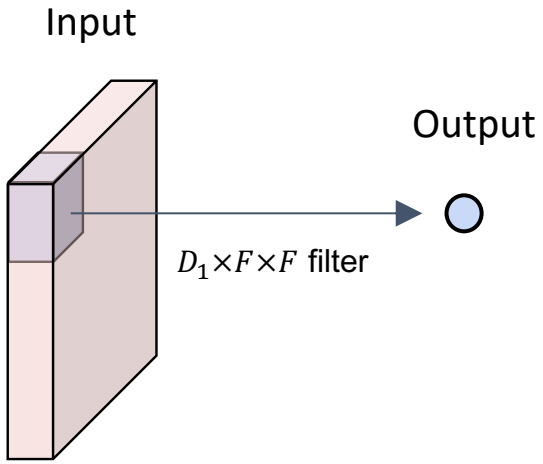
The total cost is

$$(D_1 \times F^2) + (D_1 \times F^2 - 1)$$

Let's not forget to add the bias

$$(D_1 \times F^2) + (D_1 \times F^2 - 1) + 1$$

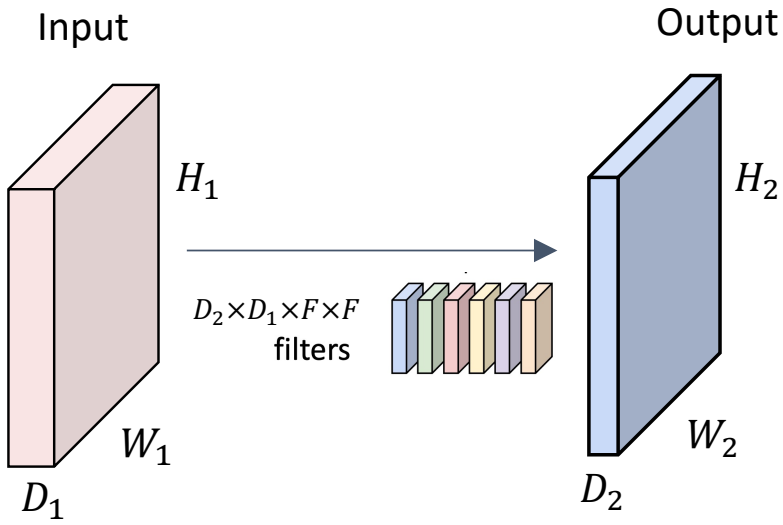
How to calculate the computations involved?



If we have only one filter, and apply it to generate output of one spatial location, the total operations

The total cost is

$$(D_1 \times F^2) + (D_1 \times F^2 - 1) + 1$$



If we have D_2 filter, and apply it to generate output of $H_2 \times W_2$ spatial location, the total operations

The total cost is

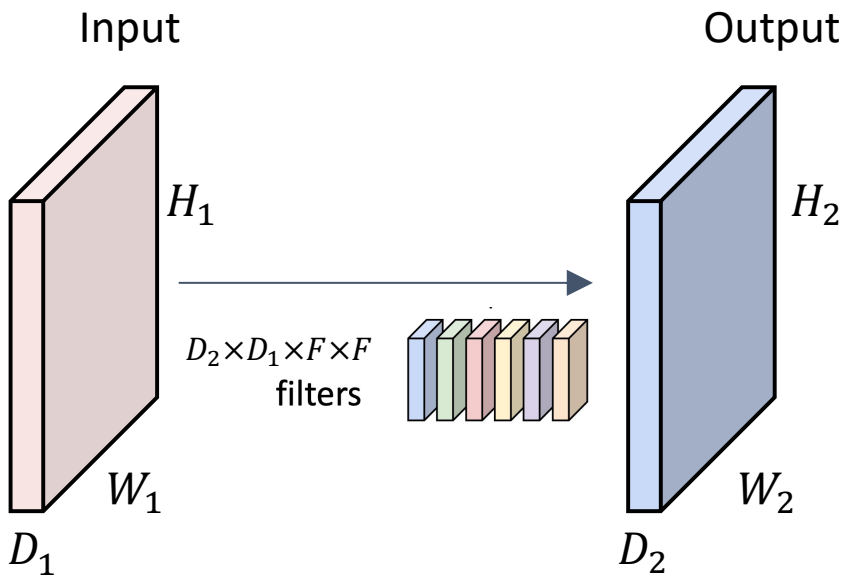
$$[(D_1 \times F^2) + (D_1 \times F^2 - 1) + 1] \times D_2 \times H_2 \times W_2$$

$$= (2 \times D_1 \times F^2) \times D_2 \times H_2 \times W_2$$

Summary

FLOPs (floating point operations)

Not FLOPS (floating point operations per second)



Assume:

- Filter size F
- Accepts a volume of size $D_1 \times H_1 \times W_1$
- Produces a output volume of size $D_2 \times H_2 \times W_2$

The FLOPs of the convolution layer is given by

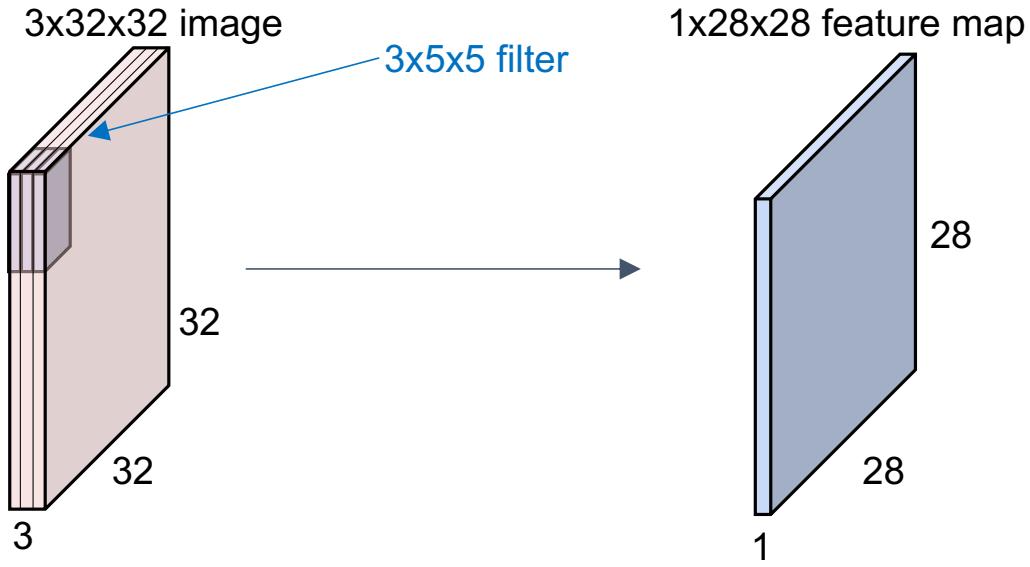
$$\text{FLOPs} = \underbrace{[(D_1 \times F^2)}_{\text{Elementwise multiplication of each filter on a spatial location}} + \underbrace{(D_1 \times F^2 - 1)}_{\text{Adding the elements after multiplication, we need } n-1 \text{ adding operations for } n \text{ elements}} + \underbrace{1}_{\text{Add the bias}}] \times D_2 \times H_2 \times W_2 = (2 \times D_1 \times F^2) \times D_2 \times H_2 \times W_2$$

Try this

Assume:

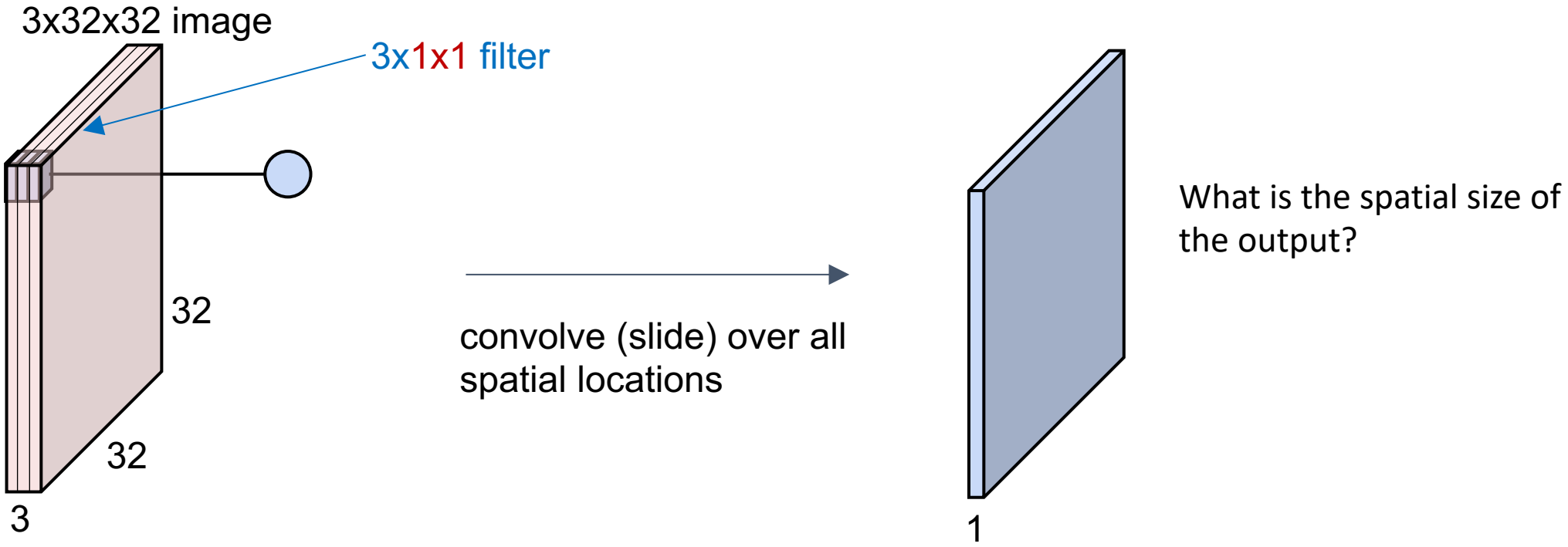
- One filter with spatial size of 5×5
- Accepts a volume of size $3 \times 32 \times 32$
- Produces a output volume of size $1 \times 28 \times 28$

The FLOPs of the convolution layer is given by?



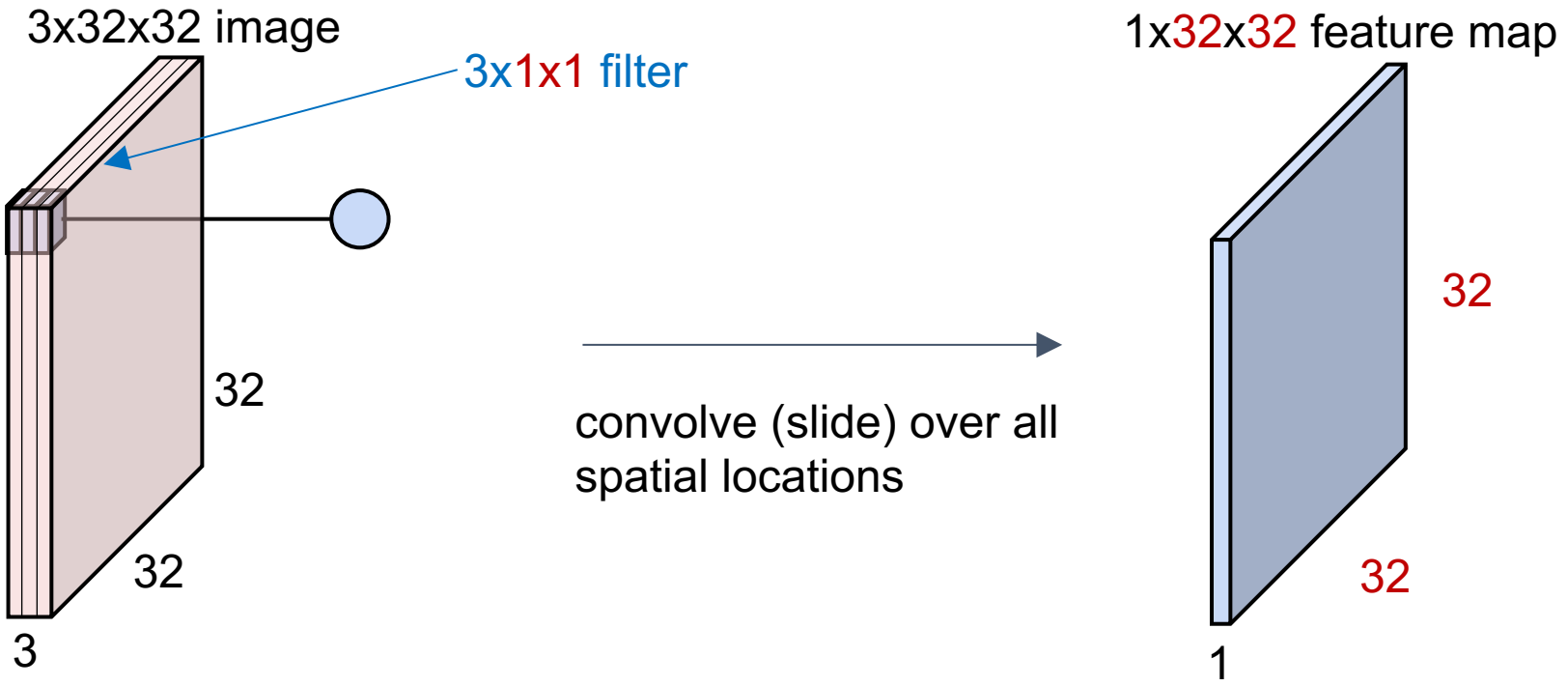
Pointwise convolution

- We have seen convolution with spatial size of 3×3, 5×5, 7×7
  - Can we have other sizes?
  - Can we have filter of spatial size 1×1?



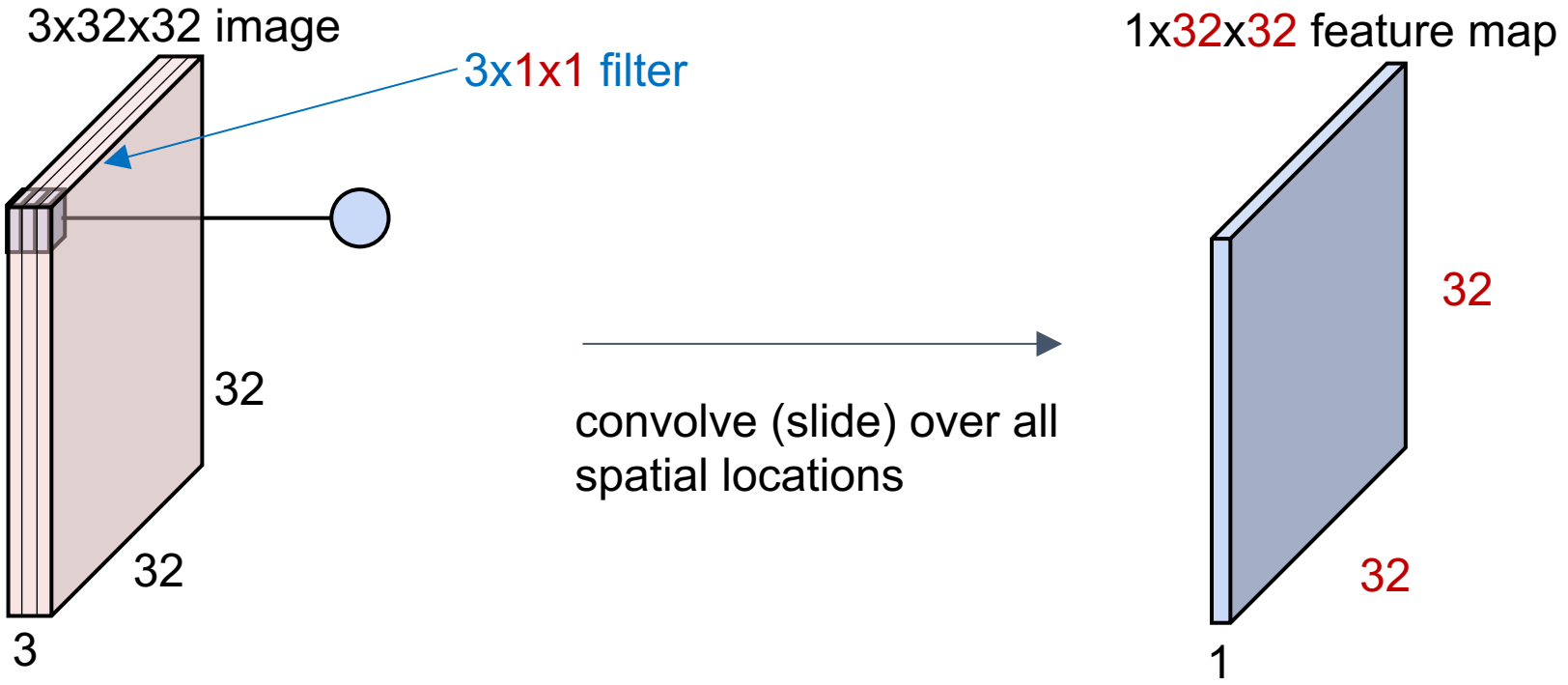
Pointwise convolution

- We have seen convolution with spatial size of 3×3, 5×5, 7×7
  - Can we have other sizes?
  - Can we have filter of spatial size 1×1?



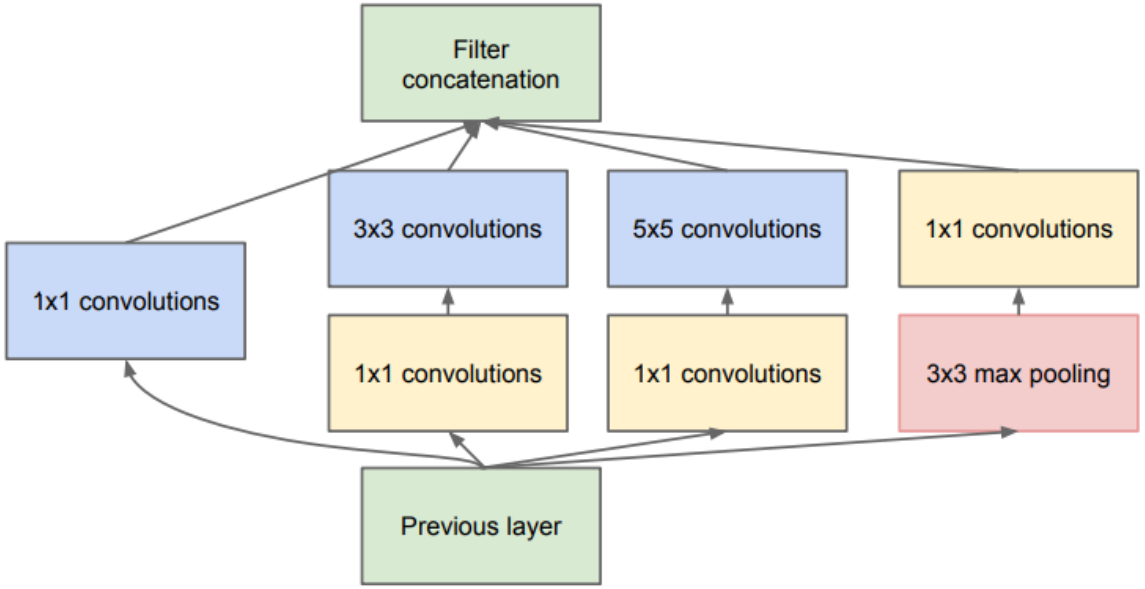
Pointwise convolution

- Why having filter of spatial size 1×1?
  - Change the size of channels
  - "Blend" information among channels by linear combination



Pointwise convolution

- A real-world example
  - 1×1 convolutions are used for compute reductions before the expensive 3×3 and 5×5 convolutions

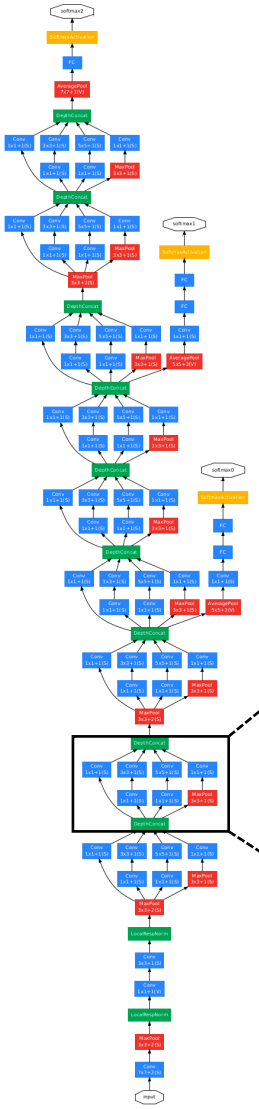


E.g., Given the output from the previous layer, reduce the number of channels of from $256\times32\times32$ to $128\times32\times32$ before the 5×5 convolutions

Referring to the FLOPs equation, reducing input channels help reduce the FLOPs

$$\text{FLOPs} = (2\times D_1\times F^2)\times D_2\times H_2\times W_2$$

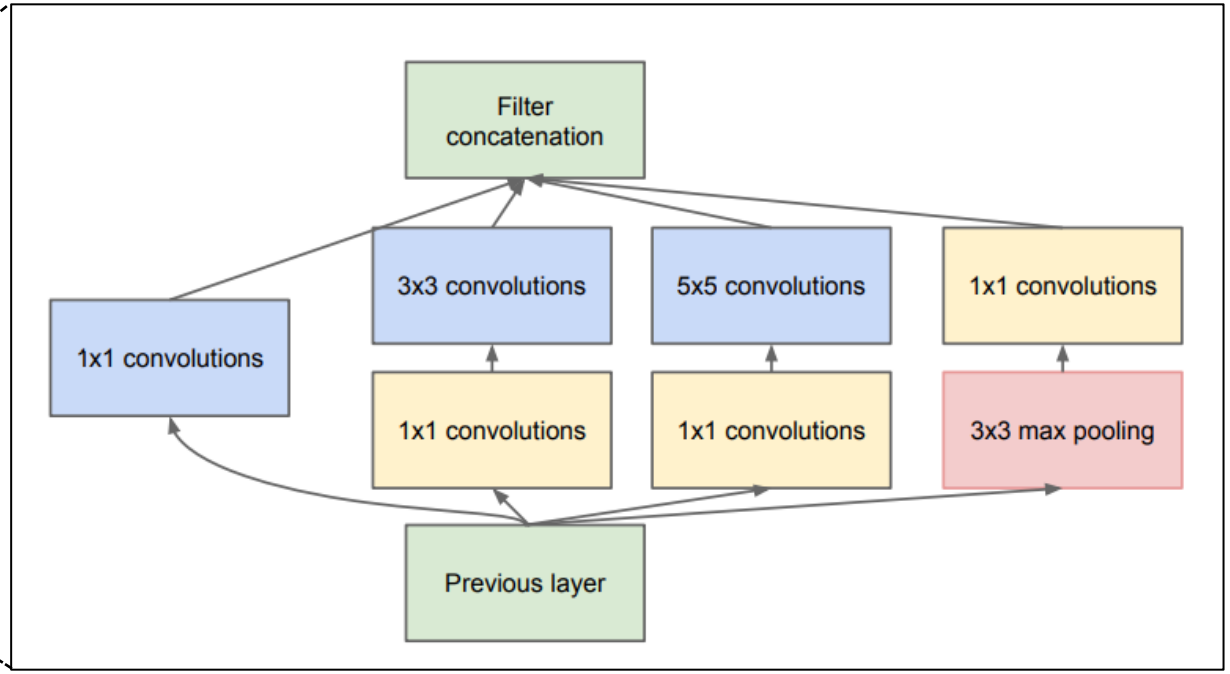
Pointwise convolution



This is known as inception structure used in GoogLeNet

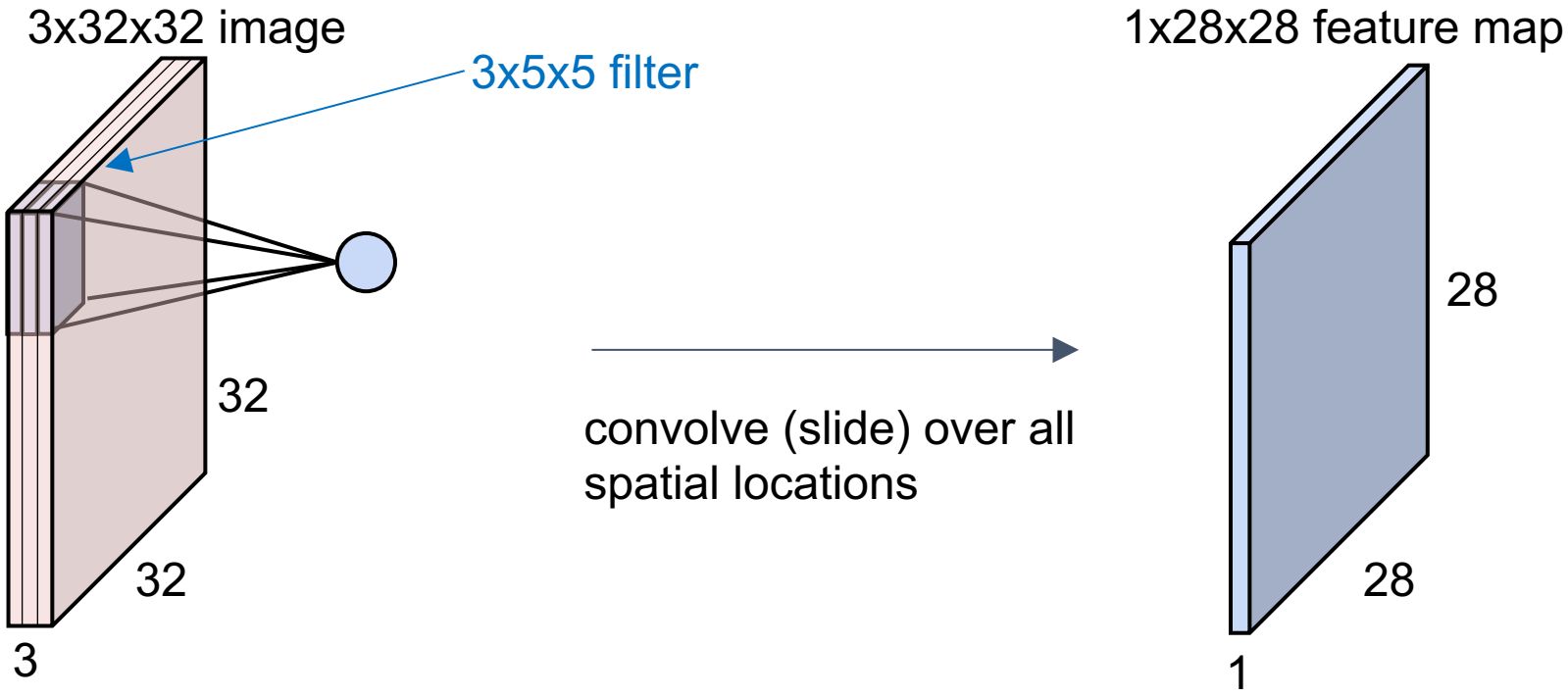
Proposed by Google in 2014, winning the ImageNet competition that year

4M parameters vs 60M parameters of AlexNet



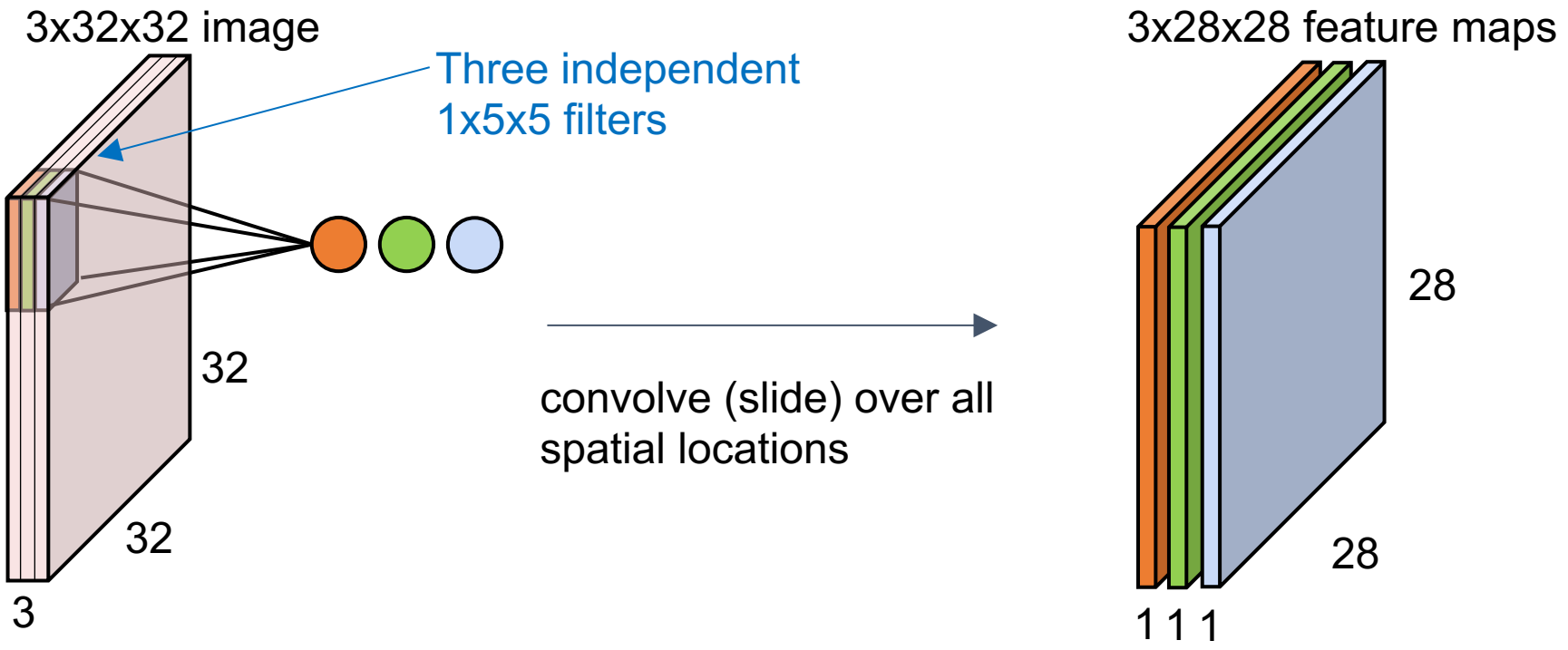
Standard convolution

- Standard convolution
  - The input and output are locally connected in spatial domain
  - In channel domain, they are fully connected

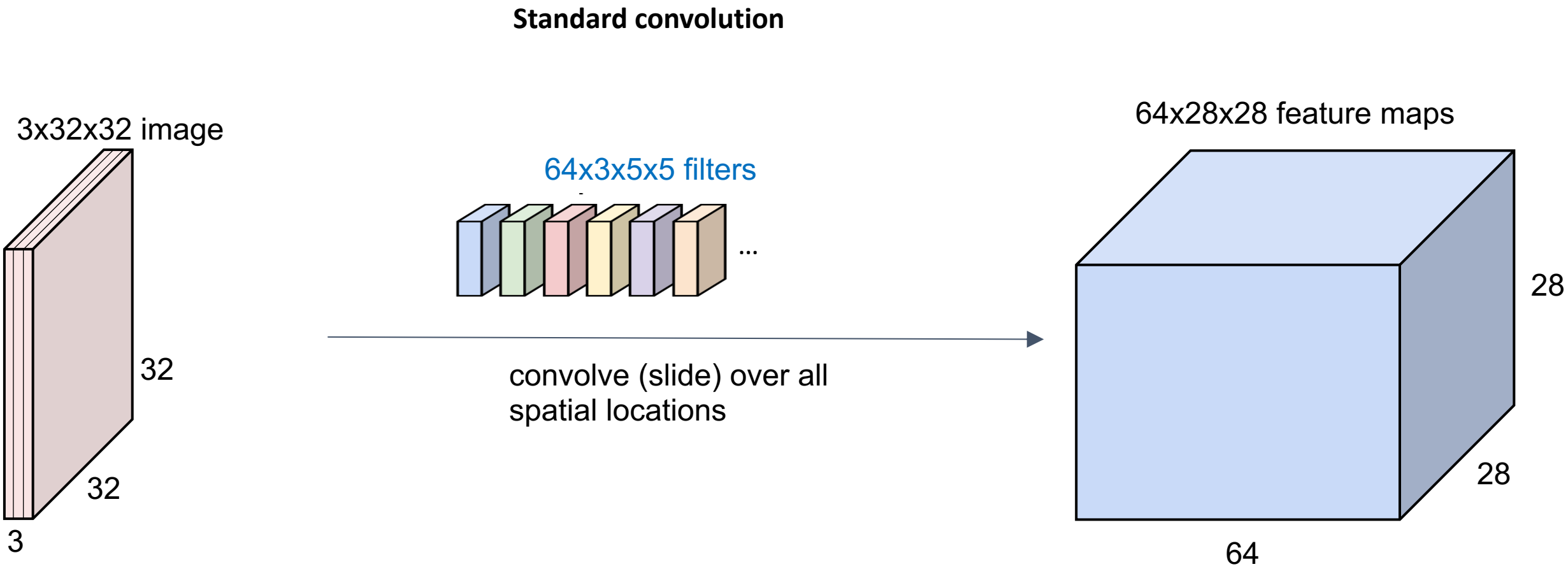


Depthwise convolution

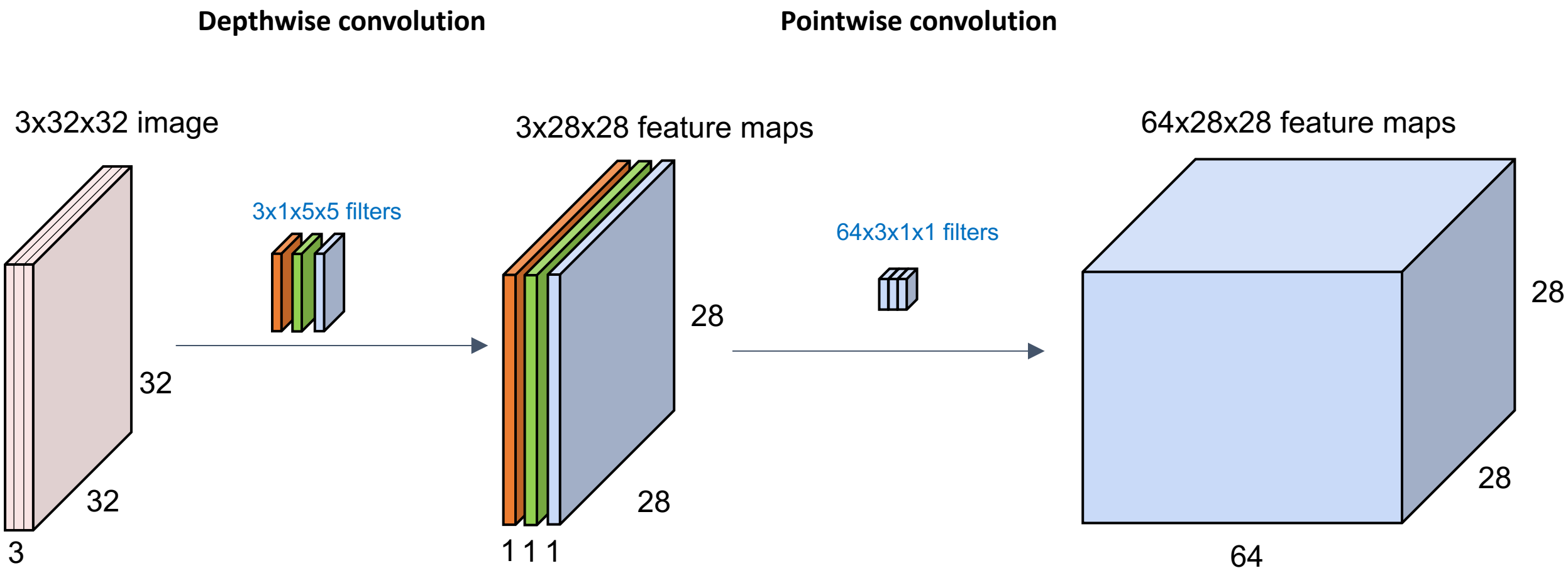
- Depthwise convolution
  - Convolution is performed independently for each of input channels



Standard convolution



Depthwise convolution + Pointwise convolution



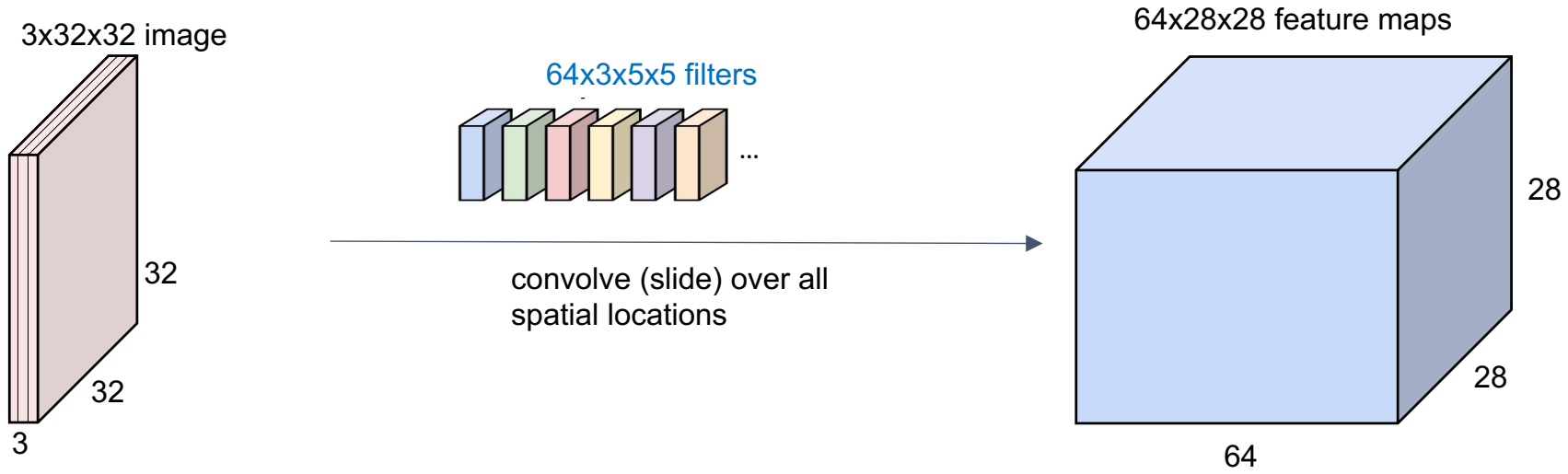
Depthwise convolution + Pointwise convolution

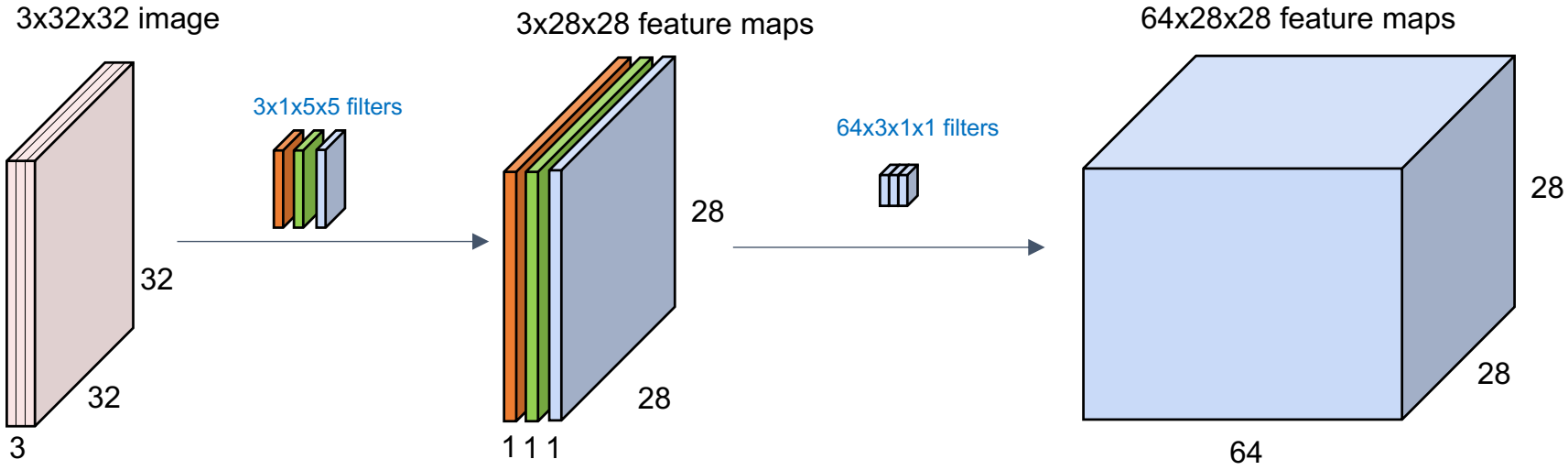
Replace standard convolution

with

Depthwise convolution + pointwise convolution

And we still get the same size of output volume!





Depthwise convolution + Pointwise convolution

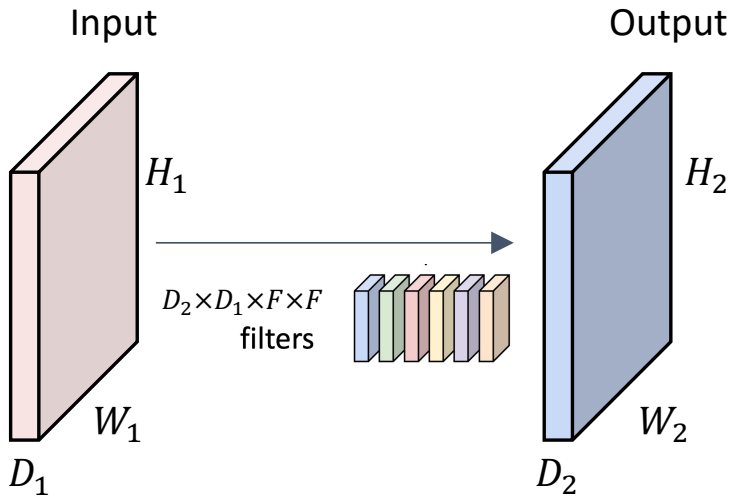
- Why replacing standard convolution with depthwise convolution + pointwise convolution?
  - The computational cost reduction rate is roughly 1/8–1/9 at only a small reduction in accuracy
  - Good if you want to deploy small networks on devices with CPU
  - Used in network such as MobileNet

G. Howard et al., MobileNets: Efficient Convolutional Neural Networks for Mobile Vision Applications, CVPR 2017

Depthwise convolution + Pointwise convolution

- Standard convolution cost

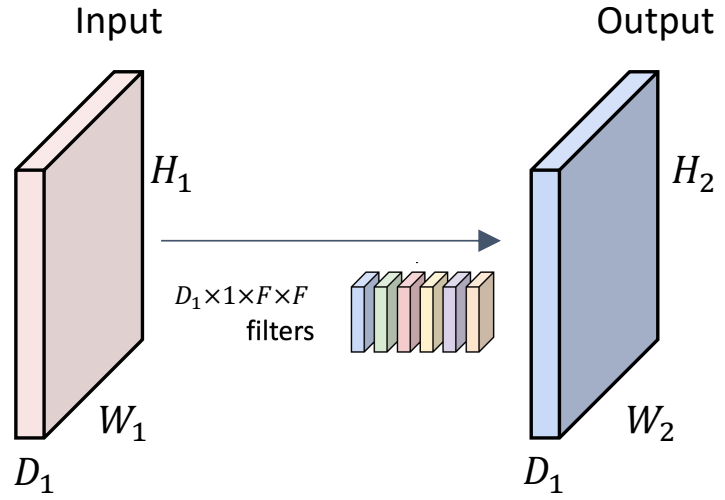
$$\text{FLOPs}_{\text{standard}} = [(D_1 \times F^2) + (D_1 \times F^2 - 1) + 1] \times D_2 \times H_2 \times W_2$$



- Depthwise convolution cost

$$\text{FLOPs}_{\text{depthwise}} = [(1 \times F^2) + (1 \times F^2 - 1) + 1] \times D_1 \times H_2 \times W_2$$

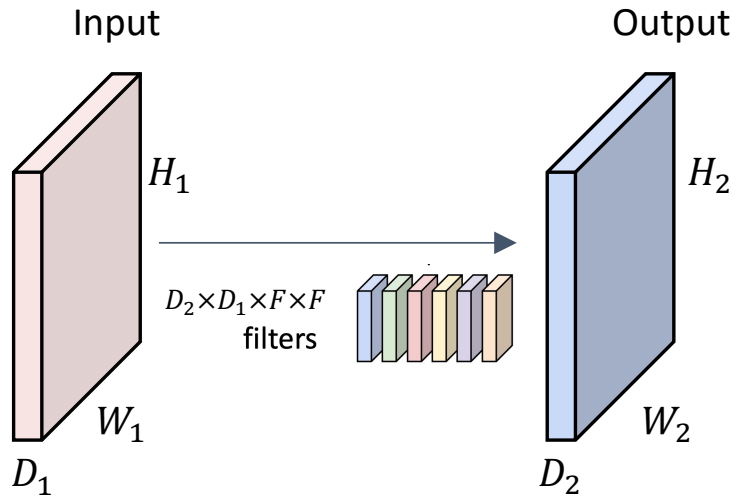
- Each filter is applied only to one channel
- The number of output channels equals to the number of input channels



Depthwise convolution + Pointwise convolution

- Standard convolution cost

$$\text{FLOPs}_{\text{standard}} = [(D_1 \times F^2) + (D_1 \times F^2 - 1) + 1] \times D_2 \times H_2 \times W_2$$



- Pointwise convolution cost

$$\text{FLOPs}_{\text{depthwise}} = [(D_1 \times 1) + (D_1 \times 1 - 1) + 1] \times D_2 \times H_2 \times W_2$$

  - The filter spatial size is 1×1

Depthwise convolution + Pointwise convolution

- Standard convolution cost

$$\text{FLOPs}_{\text{standard}} = (2 \times D_1 \times F^2) \times D_2 \times H_2 \times W_2$$

- Depthwise convolution cost

$$\text{FLOPs}_{\text{depthwise}} = (2 \times F^2) \times D_1 \times H_2 \times W_2$$

- Pointwise convolution cost

$$\text{FLOPs}_{\text{pointwise}} = (2 \times D_1) \times D_2 \times H_2 \times W_2$$

Depthwise convolution + Pointwise convolution

- How much computation do you save by replacing standard convolution with depthwise+pointwise?

$$\text{Reduction} = \frac{\text{Cost of depthwise convolution} + \text{pointwise convolution}}{\text{Cost of standard convolution}}$$

$$\text{Reduction} = \frac{(2\times F^2)\times D_1\times H_2\times W_2}{(2\times D_1\times F^2)\times D_2\times H_2\times W_2} + \frac{(2\times D_1)\times D_2\times H_2\times W_2}{(2\times D_1\times F^2)\times D_2\times H_2\times W_2}$$

$$\text{Reduction} = \frac{1}{D_2} + \frac{1}{F^2}$$

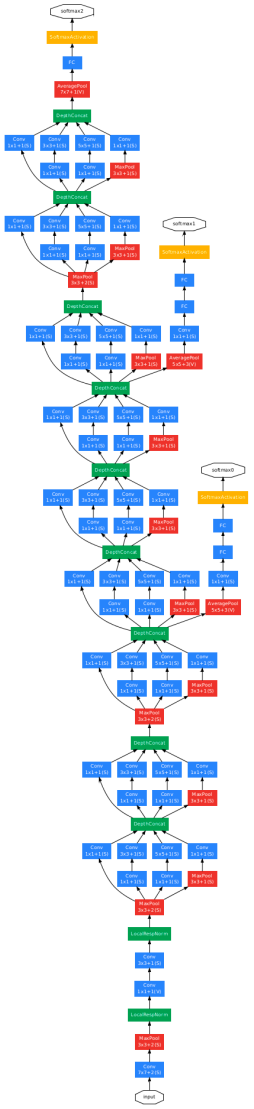
D_2 is usually large. Reduction rate is roughly 1/8–1/9 if 3×3 depthwise separable convolutions are used

Summary

- More on convolutions
  - You learn how to calculate computation complexity of convolutional layer
  - Pointwise convolution can be used to change the size of channels. This can be used to achieve channel reduction and thus saving computational cost
  - Depthwise convolution + Pointwise convolution yields lower computations than standard convolution

Batch Normalization

GoogLeNet



Proposed by Google in 2014, winning the ImageNet competition that year

Apart from the inception structure, there is another important technique called **batch normalization**

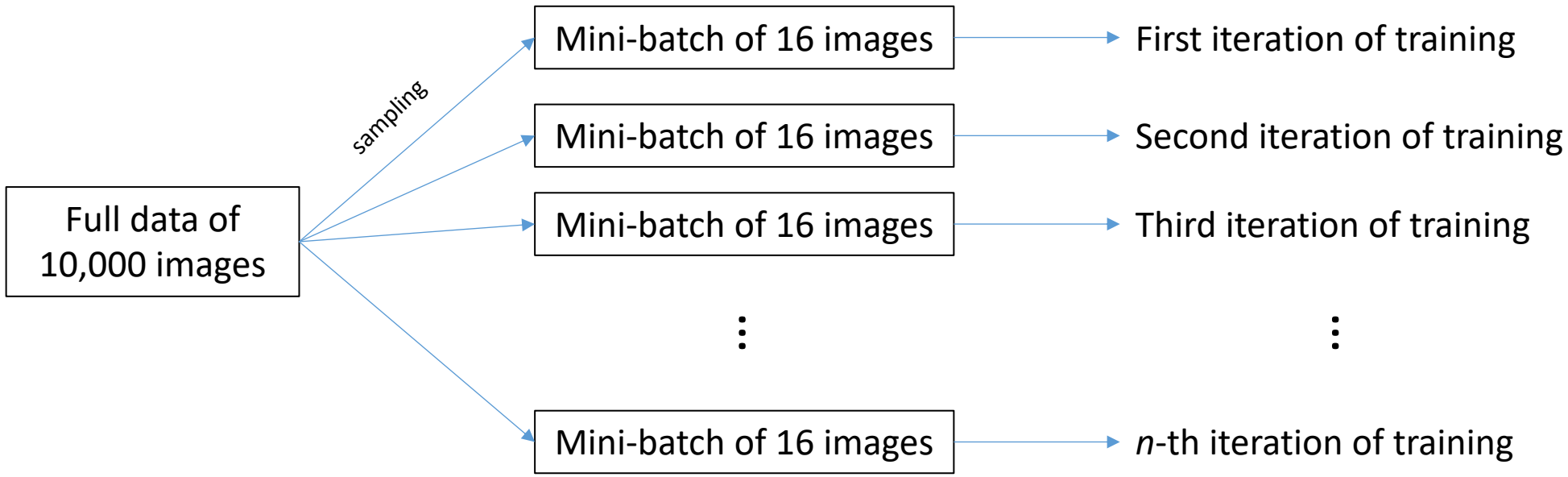
Problem: deep networks are very hard to train!

Main idea: “Normalize” the outputs of a layer so they have zero mean and unit variance

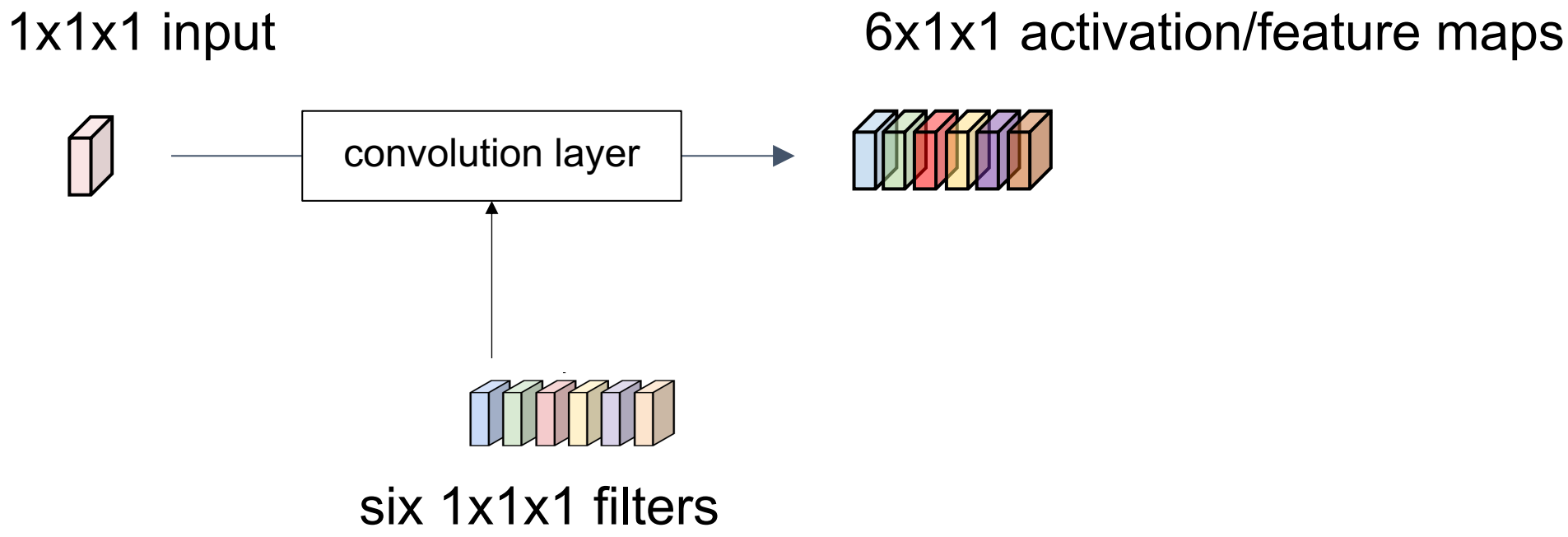
Effect: allowing higher learning rates and reducing the strong dependence on initialization

Mini-batch

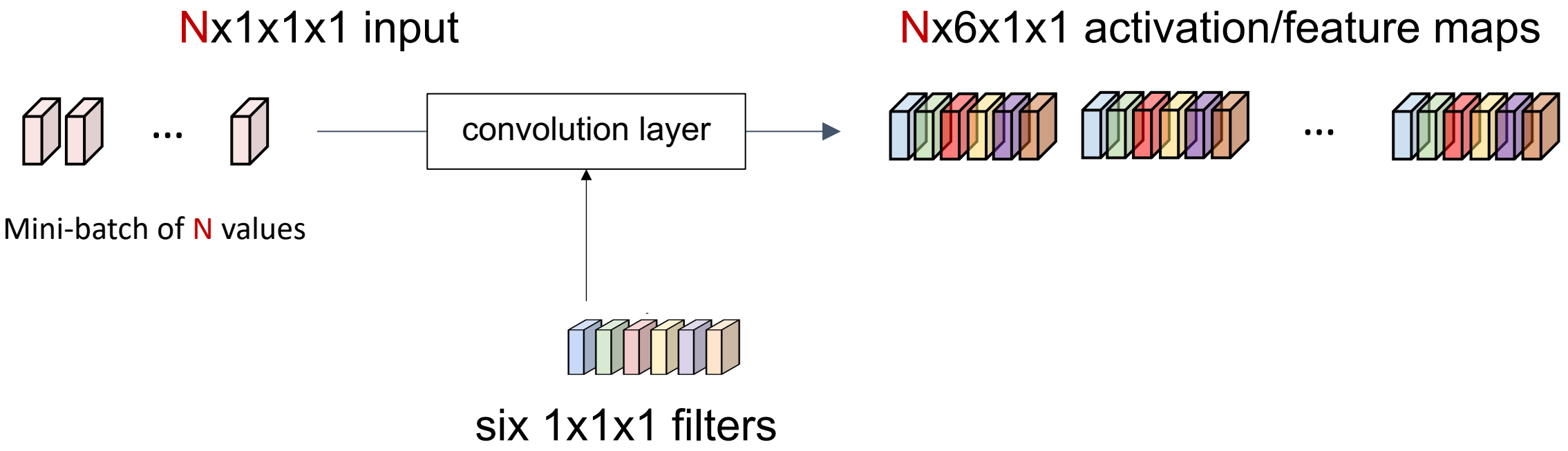
- What is mini-batch?
  - A subset of all data during one iteration to compute the gradient



Mini-batch


1x1x1 input
convolution layer
6x1x1 activation/feature maps
six 1x1x1 filters

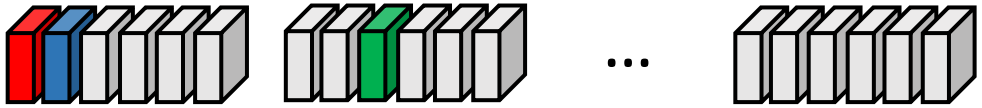
Mini-batch


Nx1x1x1 input
Nx6x1x1 activation/feature maps
convolution layer
Mini-batch of N values
six 1x1x1 filters

Batch Normalization

Let's arrange the activations of the mini batch in a matrix of $N \times D$

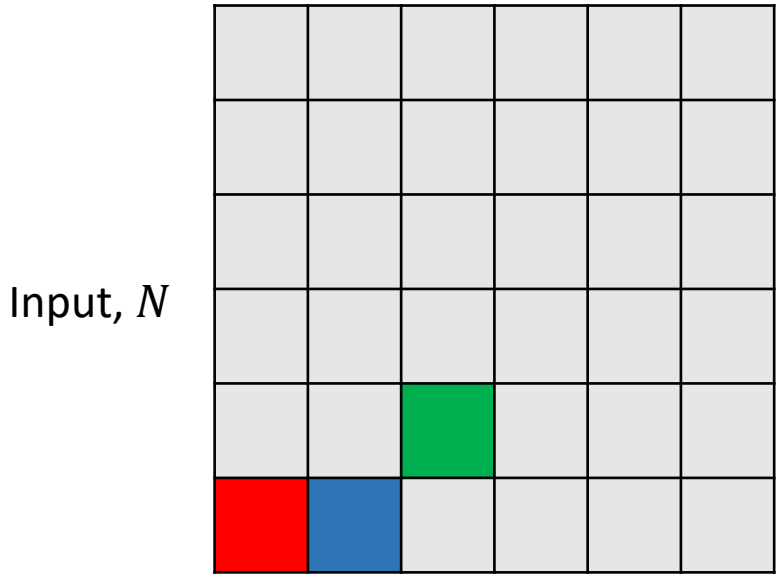
Nx6x1x1 activation/feature maps



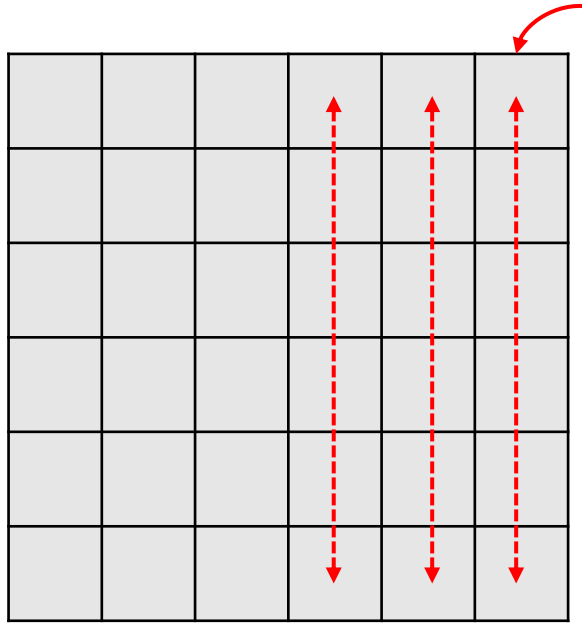
Activation of the first input in the mini-batch

Activation of the second input in the mini-batch

Activation of the N-th input in the mini-batch

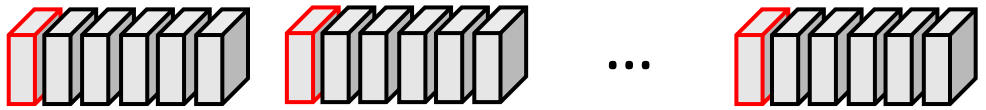


Batch Normalization

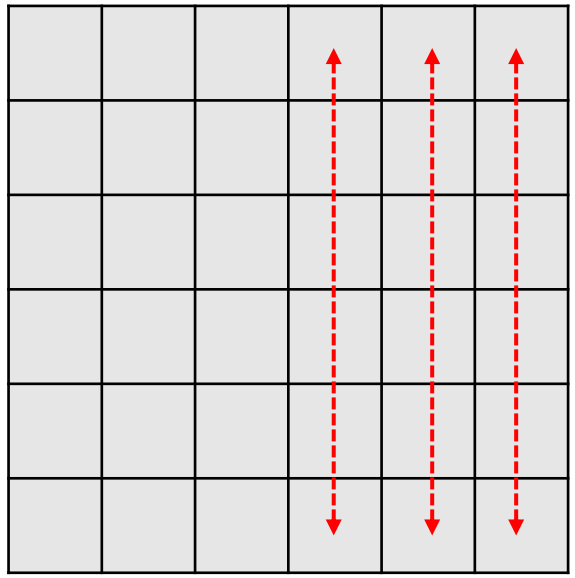


The goal of batch normalization is to normalize the values across each column so that the values of the column have zero mean and unit variance

For instance, normalizing the first column of this matrix means normalizing the activations (highlighted in red) below



Batch Normalization - Training



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

Per-channel mean, shape is $1 \times D$

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

Per-channel variance, shape is $1 \times D$

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

Normalize the values, shape is $N \times D$

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

Scale and shift the **normalized** values to add flexibility, output shape is $N \times D$

Learnable parameters: γ and β, shape is $1 \times D$

Batch Normalization – Test Time

Input

Channel, D

Problem: Estimates depend on minibatch; can't do this at test-time

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

Per-channel mean, shape is $1 \times D$

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

Per-channel variance, shape is $1 \times D$

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

Normalize the values, shape is $N \times D$

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

Scale and shift the normalized values to add flexibility, output shape is $N \times D$

Learnable parameters: γ and β, shape is $1 \times D$

Batch Normalization – Test Time

Input

Channel, D

Average of values seen during training

Per-channel mean, shape is $1 \times D$

Average of values seen during training

Per-channel variance, shape is $1 \times D$

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

Normalize the values, shape is $N \times D$

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

Scale and shift the normalized values to add flexibility, output shape is $N \times D$

Learnable parameters: γ and β, shape is $1 \times D$

Batch Normalization – Test Time

Input

Channel, D

Average of values seen during training

Per-channel mean, shape is $1 \times D$

Average of values seen during training

Per-channel variance, shape is $1 \times D$

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

Normalize the values, shape is $N \times D$

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

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

Scale and shift the normalized values to add flexibility, output shape is $N \times D$

Learnable parameters: γ and β, shape is $1 \times D$

Batch Normalization

Batch Normalization for
fully-connected networks

$x: N \times D$

Normalize ↓

$\mu, \sigma: 1 \times D$

$\gamma, \beta: 1 \times D$

$y = \gamma(x-\mu)/\sigma+\beta$

Batch Normalization for
convolutional networks
(Spatial Batchnorm, BatchNorm2D)

$x: N \times C \times H \times W$

Normalize ↓ ↓ ↓

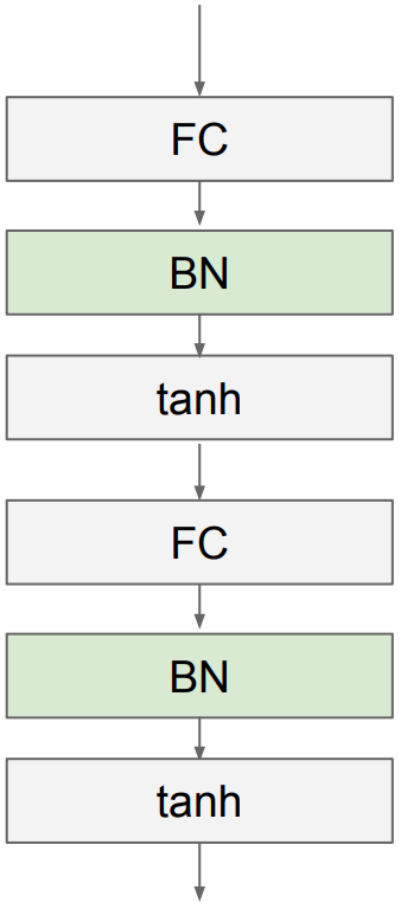
Normalize also on the spatial dimensions

$\mu, \sigma: 1 \times C \times 1 \times 1$

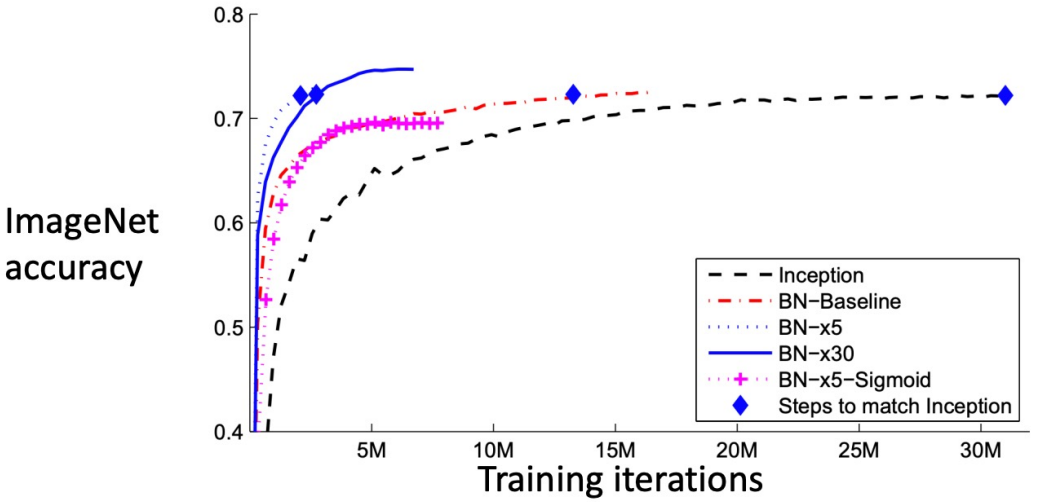
$\gamma, \beta: 1 \times C \times 1 \times 1$

$y = \gamma(x-\mu)/\sigma+\beta$

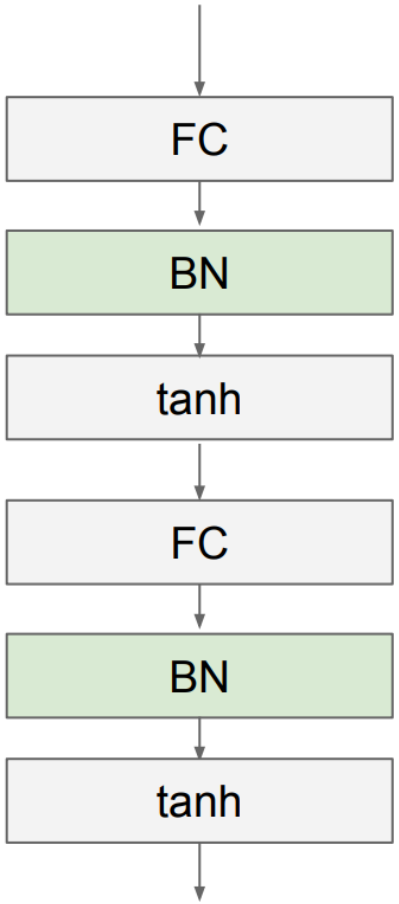
Batch Normalization



- It is usually done after a fully connected/convolutional layer and before a non-linearity layer
- Zero overhead at test-time: can be fused with conv
- Allows higher learning rates, faster convergence
- Networks becomes more robust to initialization



Batch Normalization



- It is usually done after a fully connected/convolutional layer and before a non-linearity layer
- Zero overhead at test-time: can be fused with conv
- Allows higher learning rates, faster convergence
- Networks becomes more robust to initialization
- Not well-understood theoretically (yet)
- Behaves differently during training and testing: this is a very common source of bugs!

Prevent Overfitting

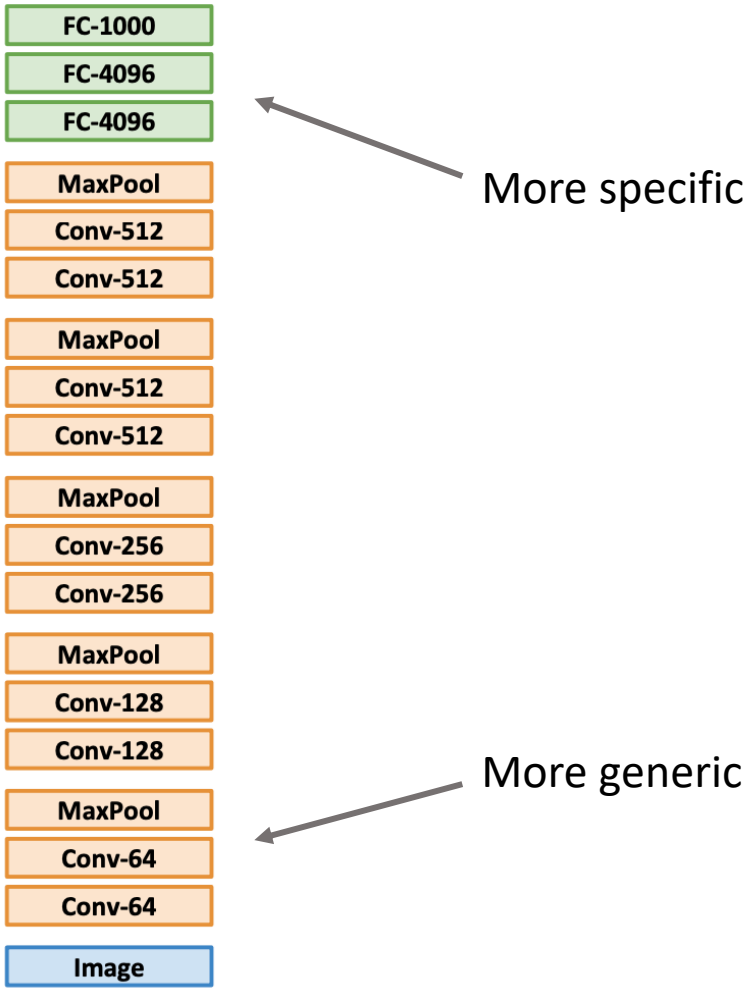

I WAS WINNING IMAGENET
UNTIL A DEEPER MODEL CAME ALONG
imgflip.com

Why overfitting?

- This happens when our model is too complex and too specialized on a small number of training data.

- Increase the size of the data, remove outliers in data, reduce the complexity of the model, reduce the feature dimension

Transfer learning



In a network with an N-dimensional softmax output layer that has been successfully trained toward a supervised classification objective, each output unit will be specific to a particular class

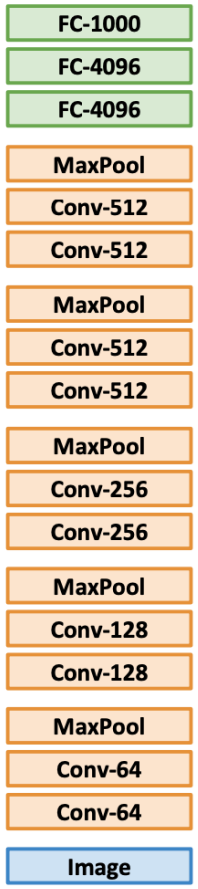
When trained on images, deep networks tend to learn first-layer features that resemble either Gabor filters or color blobs. These first-layer features are general.

Yosinski et al., How transferable are features in deep neural networks? NIPS'14

Transfer learning

Pre-training + Fine-tuning

Step 1: Pre-training on large-scale dataset like ImageNet



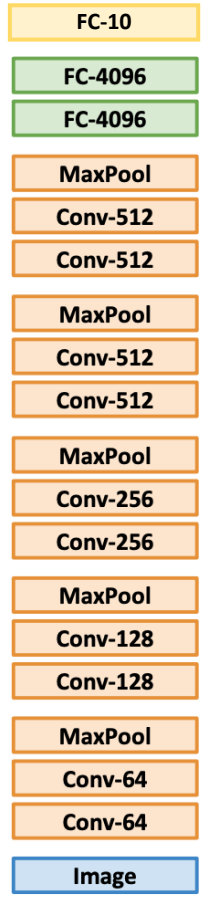
Transfer weights

Step 2: Use pre-trained network as initialization

Remove last layer

Freeze these

Step 3: Fine-tuning

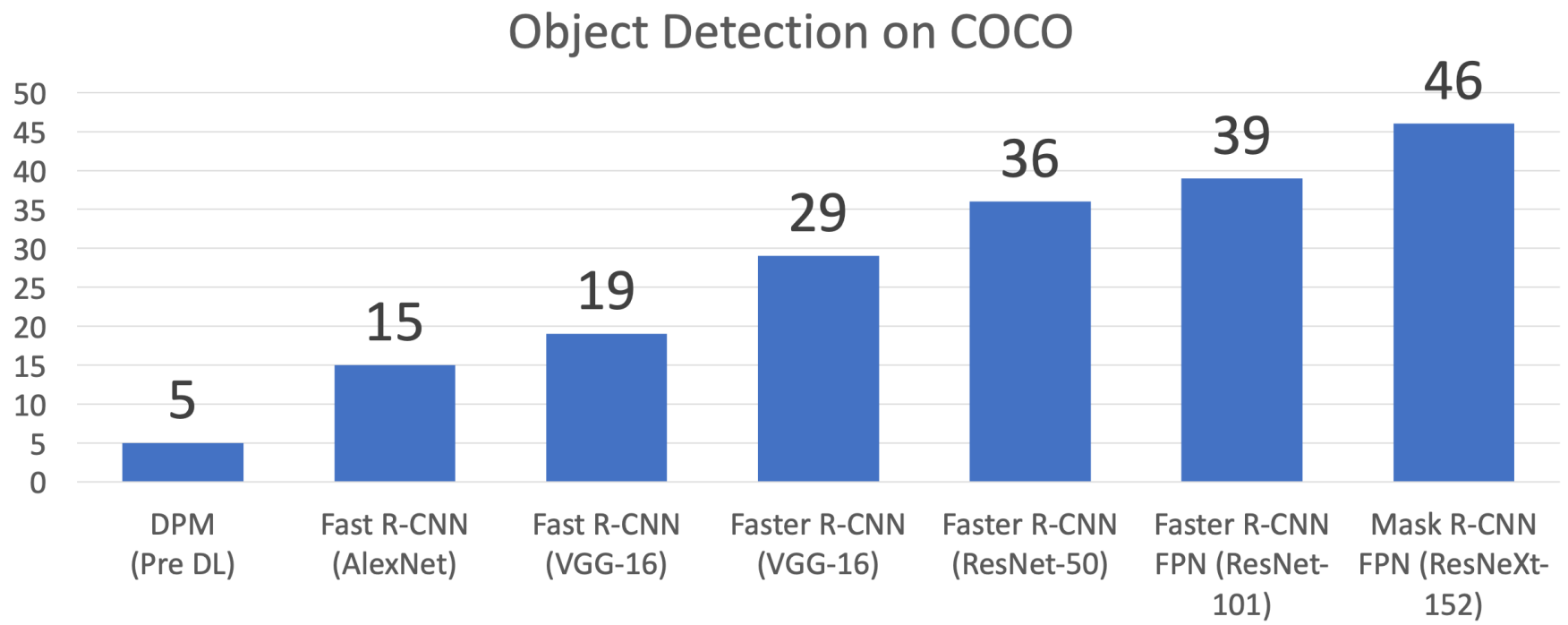


Add new layer correspond to the target class number

Train on new data

Transfer learning

Architecture matters

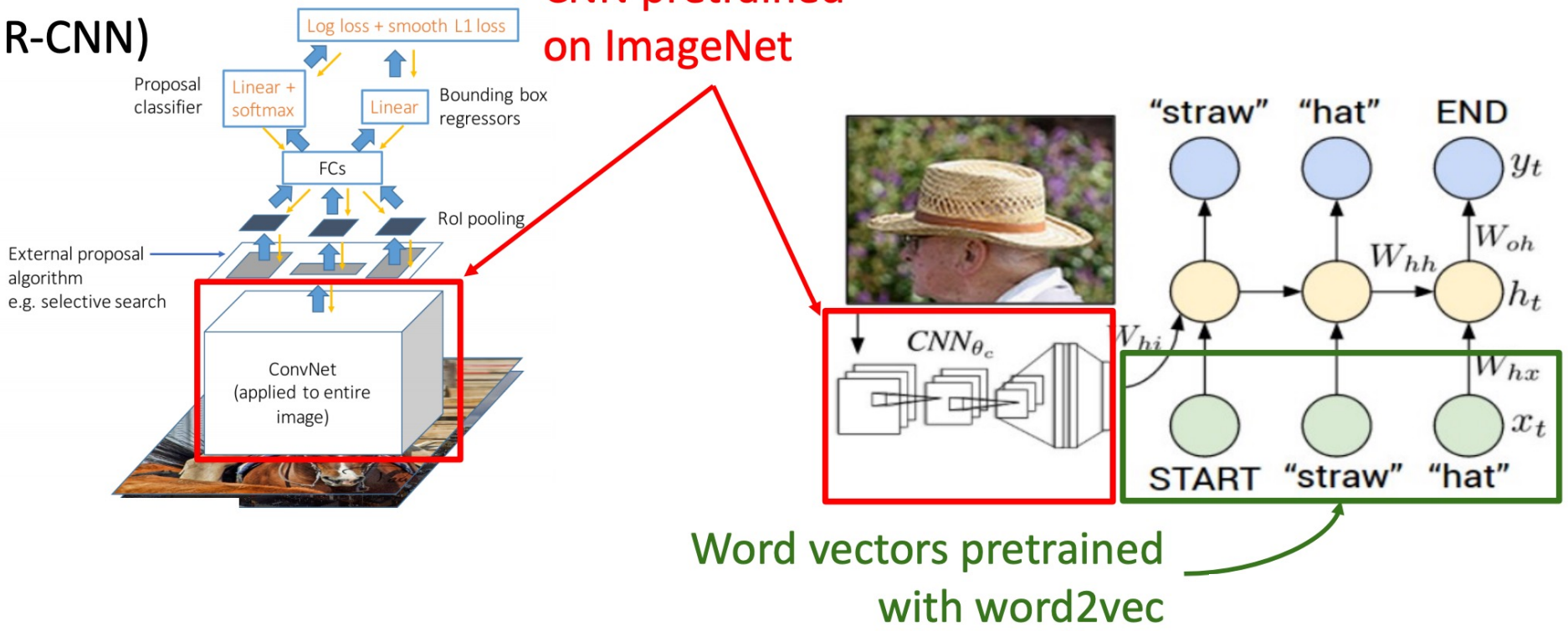


Ross Girshick, “The Generalized R-CNN Framework for Object Detection”, ICCV 2017 Tutorial on Instance-Level Visual Recognition

Transfer learning

Transfer learning is pervasive

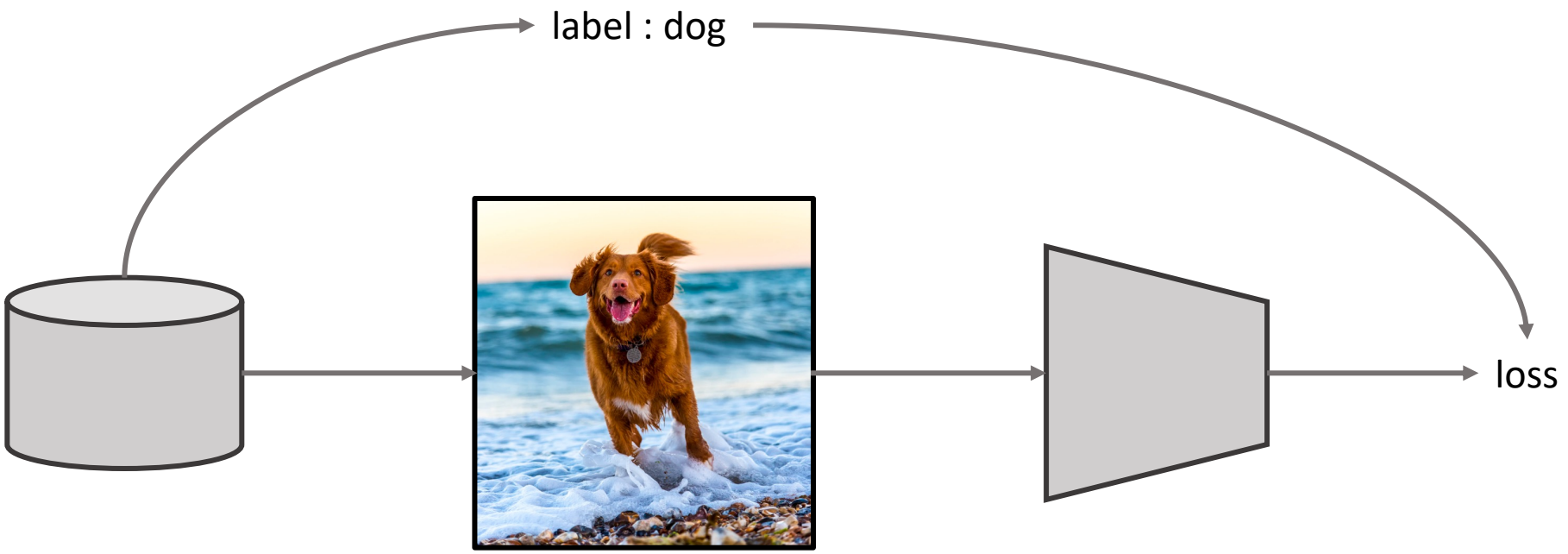
Object Detection (Fast R-CNN)



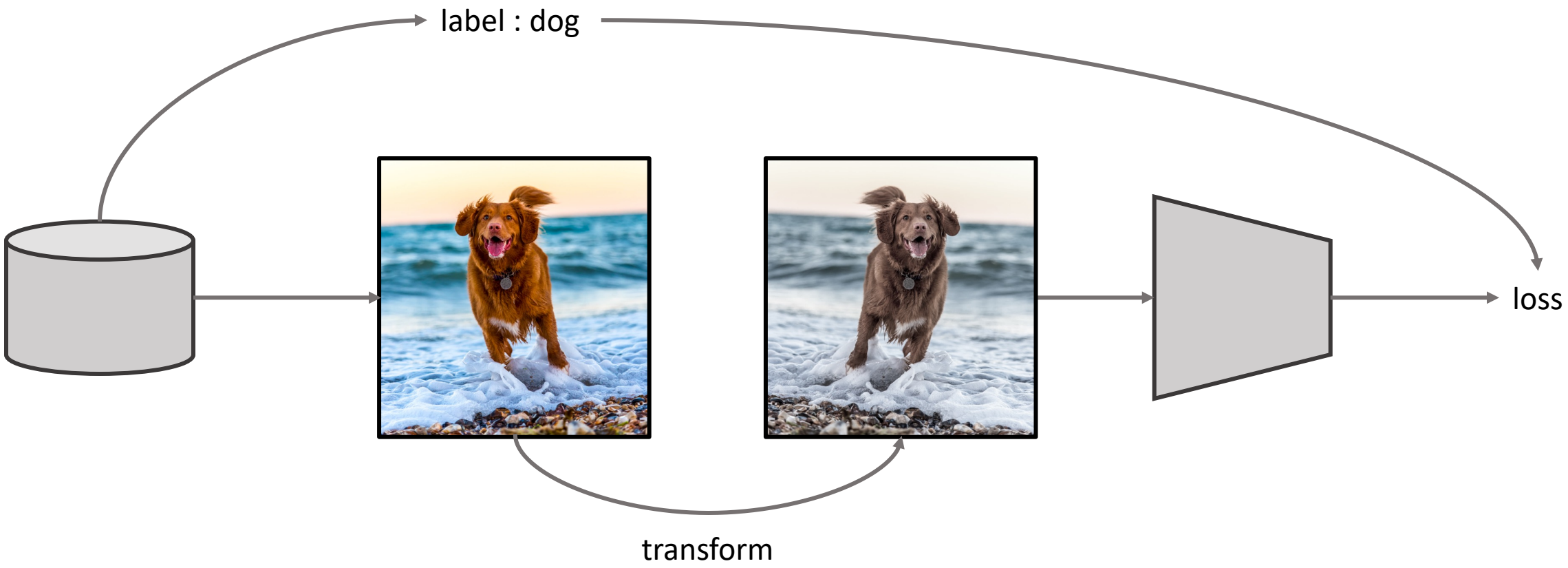
Girshick, "Fast R-CNN", ICCV 2015

Karpathy and Fei-Fei, "Deep Visual-Semantic Alignments for Generating Image Descriptions", CVPR 2015

Data augmentation



Data augmentation



Data augmentation

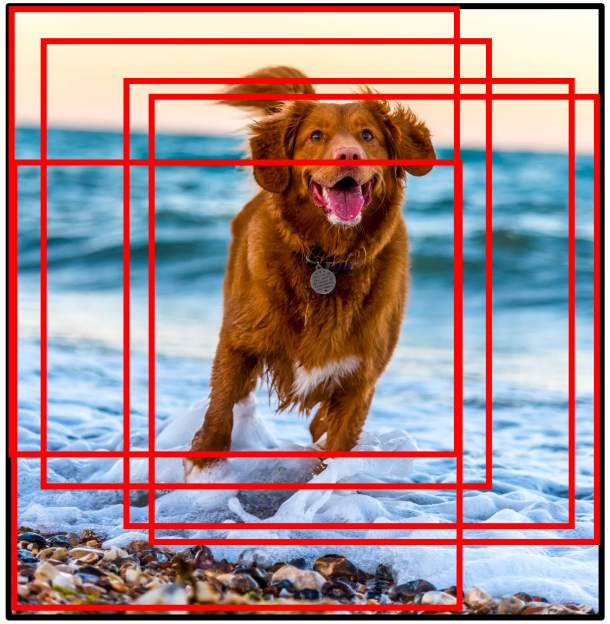
Horizontal flip

Data augmentation

Random crops and scales

Training:

1. Pick random L in range [256, 480]
2. Resize training image, short side = L
3. Sample random 224 x 224 patch



Data augmentation

Color jitter

Simple :

1. Randomize contrast and brightness

Complex:

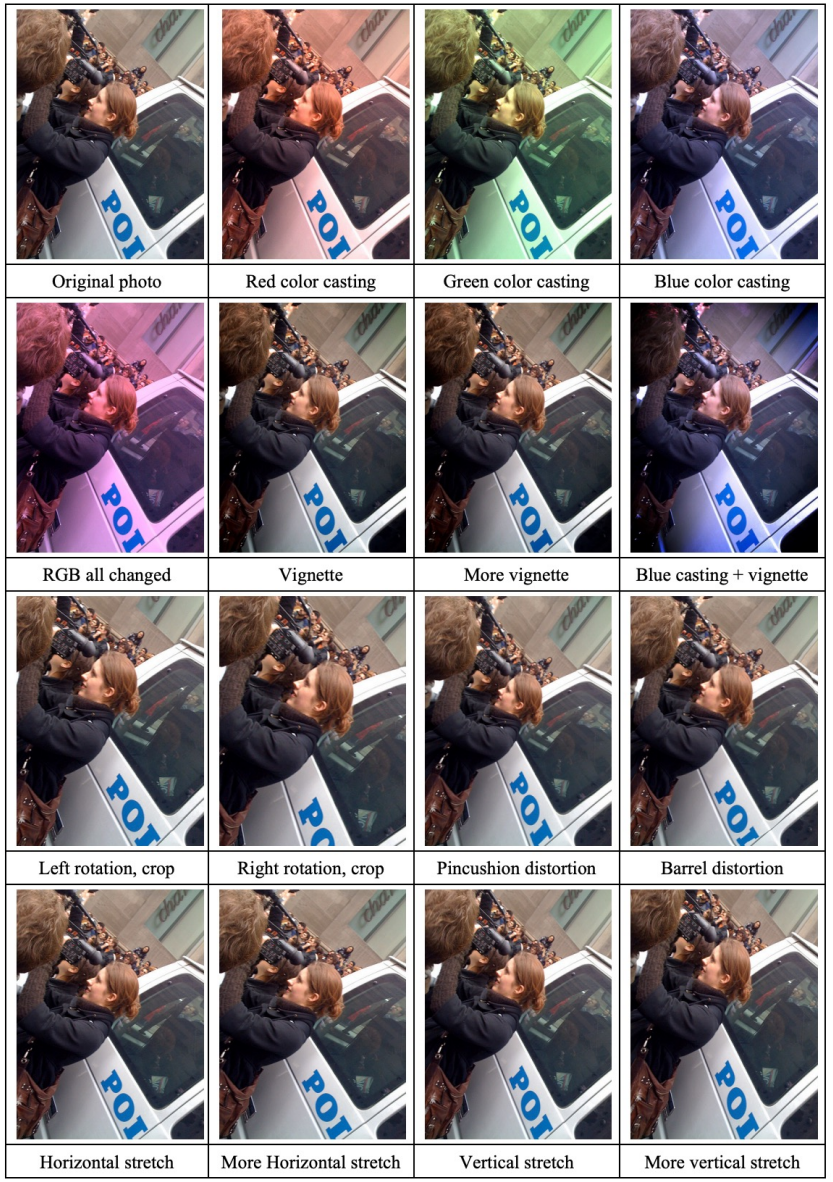
1. Apply PCA to all [R, G, B] pixels in training set
2. Sample a “color offset” along principal component directions
3. Add offset to all pixels of a training image

(Used in AlexNet, ResNet, etc.)

Data augmentation

There are many more data augmentation schemes

Random mix/combinations of:
translation - rotation - stretching - shearing, - lens distortions ….



Ren Wu et al., Deep Image: Scaling up Image Recognition, https://arxiv.org/pdf/1501.02876v2.pdf?