Bits of Vision: Convolution and CNNs

Agenda

- **So far.** Neural network basics
  - Multi-layer perceptrons
  - Training neural networks
    - Backprop
    - Optimizers & Initializations

- **Today.** Topics in computer vision
  - Convolution
  - Advanced CNNs

Convolution

Recap: MLPs

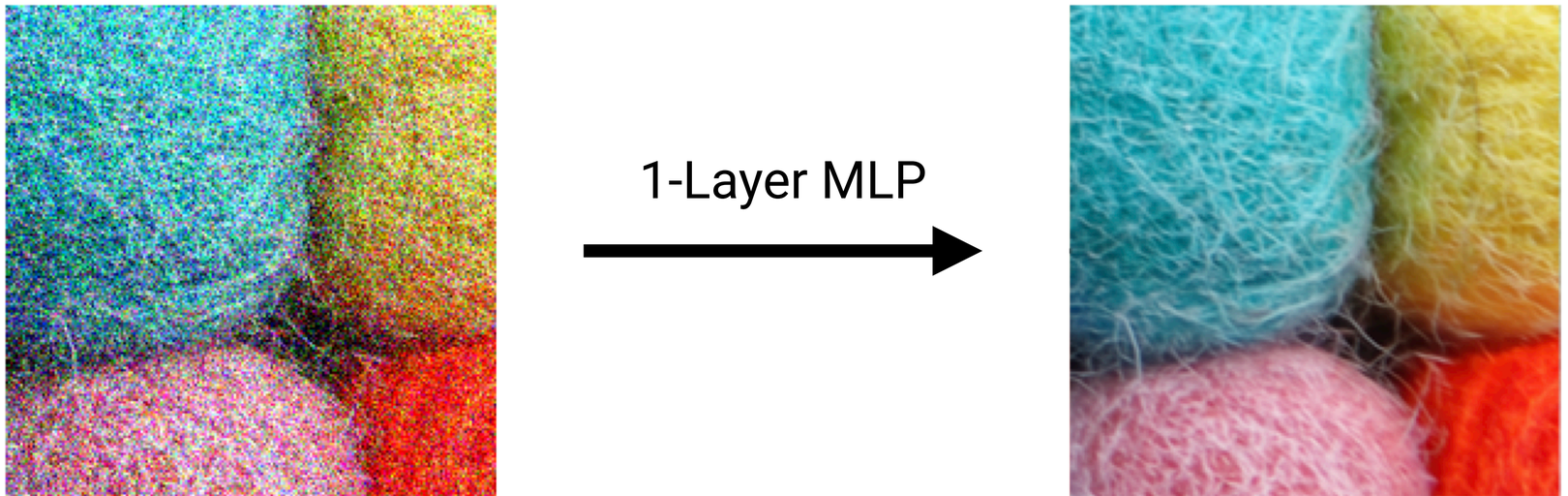
- MLPs take a simple form

$$f(\mathbf{x}) = \mathbf{W}_L \sigma(\mathbf{W}_{L-1} \sigma(\cdots \sigma(\mathbf{W}_1 \mathbf{x} + \mathbf{b}_1) \cdots + \mathbf{b}_{L-1}) + \mathbf{b}_L$$

  - Alternatingly applies two operations
    - Linear operation: $\mathbf{x} \mapsto \mathbf{W}\mathbf{x} + \mathbf{b}$
    - Nonlinear activation: $\mathbf{x} \mapsto \sigma(\mathbf{x})$

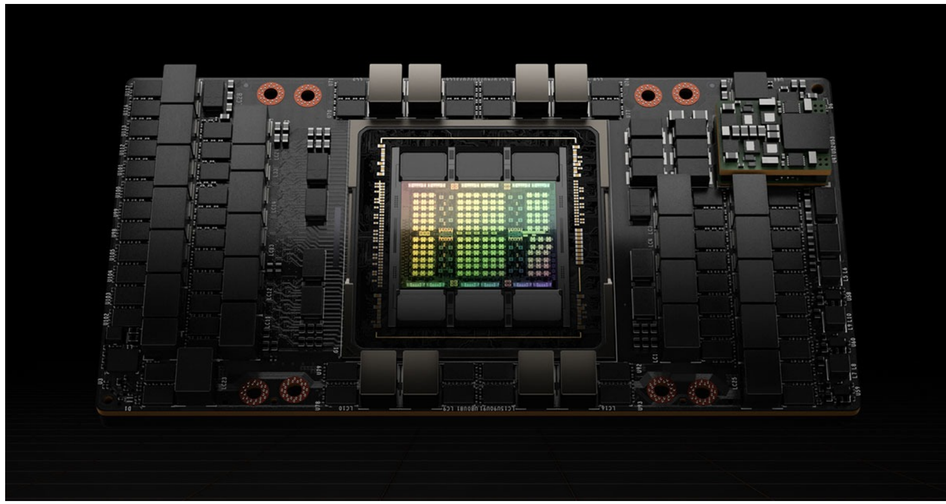
Recap: MLPs

- Matmuls on visual signals require too many params & compute
- **Example.** Suppose that we train an image denoising model
    - Process a 1080p image to another 1080p image
      (1920 x 1080 pixels)
    - If it is a linear model (i.e., 1-layer MLP), then we need:
        - 38.7 trillion parameters = 155TB in FP32
        - 77.4 trillion inference FLOPs



Recap: MLPs

- **Example.** Suppose that we have a similar denoising model
  - Furthermore, assume that:
    - we have 10 layers
    - train for 100 epochs
    - 1 million training images
  - Then we need 1.5×10^{23} FLOPs
    - = 70 years of training on H100



NVIDIA H100
67 TFLOPS

Convolution

- **Idea.** Mitigate this issue with two ideas: locality and weight sharing

- **Definition.** A convolution of two functions is:

$$(f * g)(t) = \int_{-\infty}^{+\infty} f(\tau) \cdot g(t - \tau)\, \mathrm{d}\tau$$

  - The response of a system, which has impulse response $f(t)$, when given an input signal $g(t)$

Unit Pulse → System → $f(t)$

$g(t)$ → System → $(f * g)(t)$

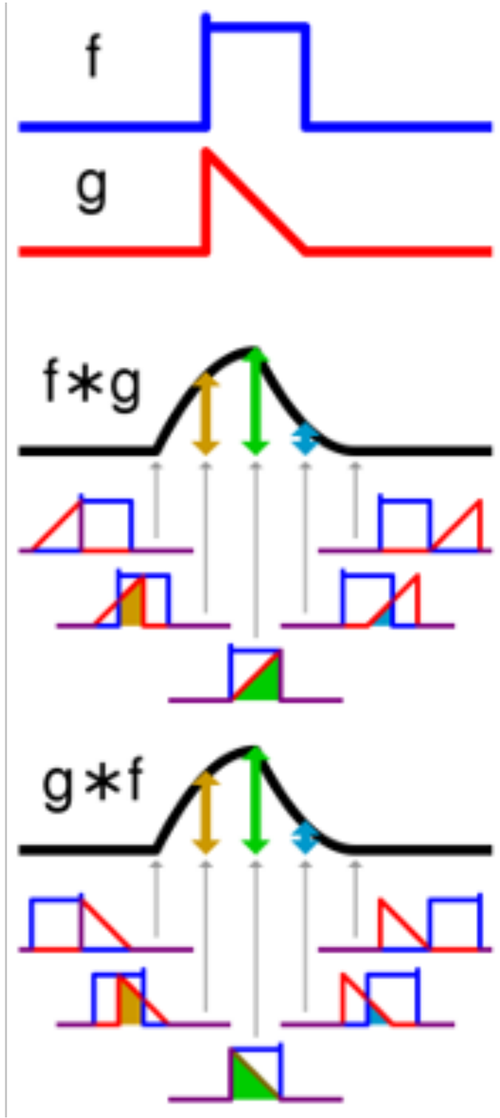
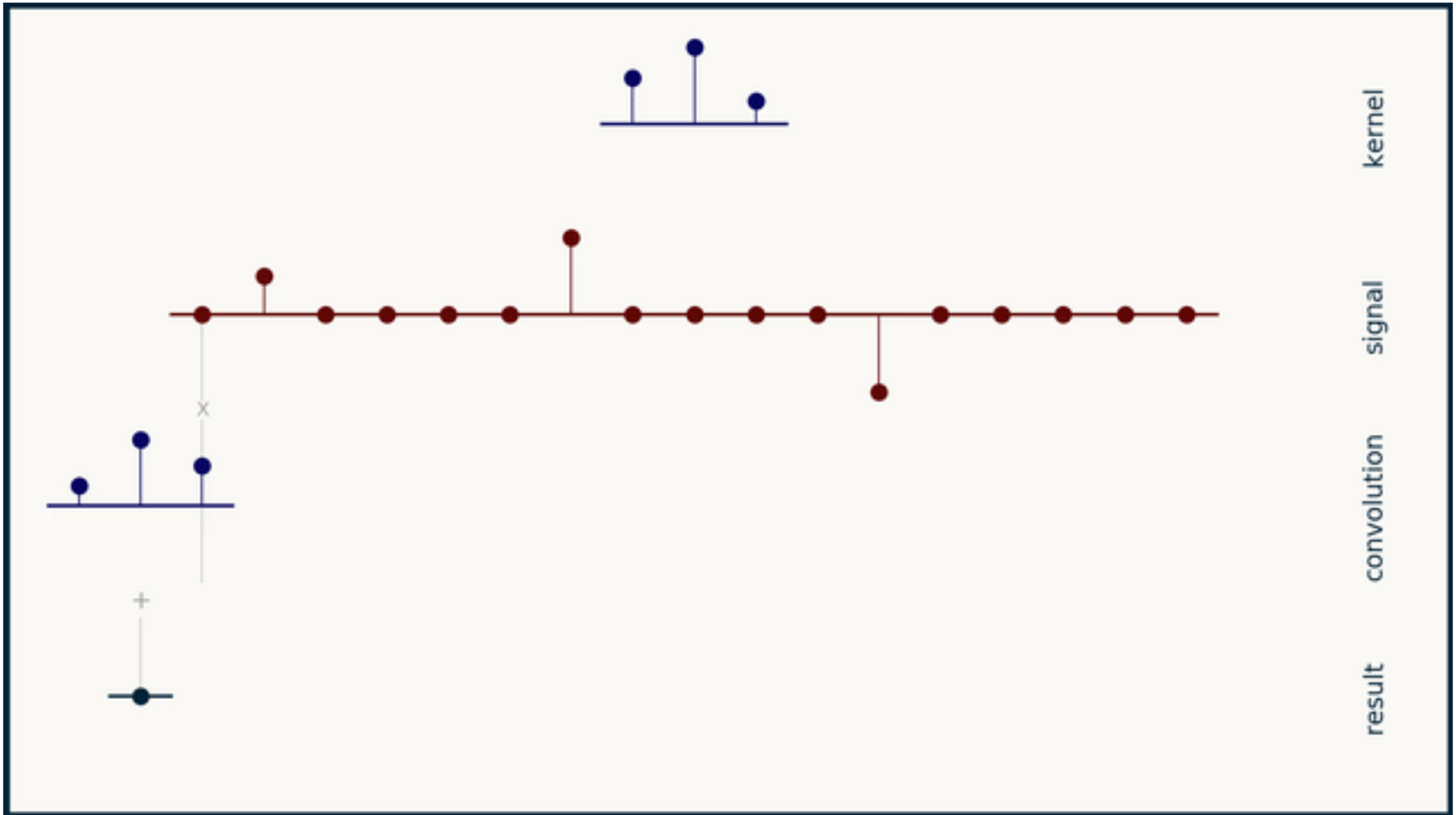
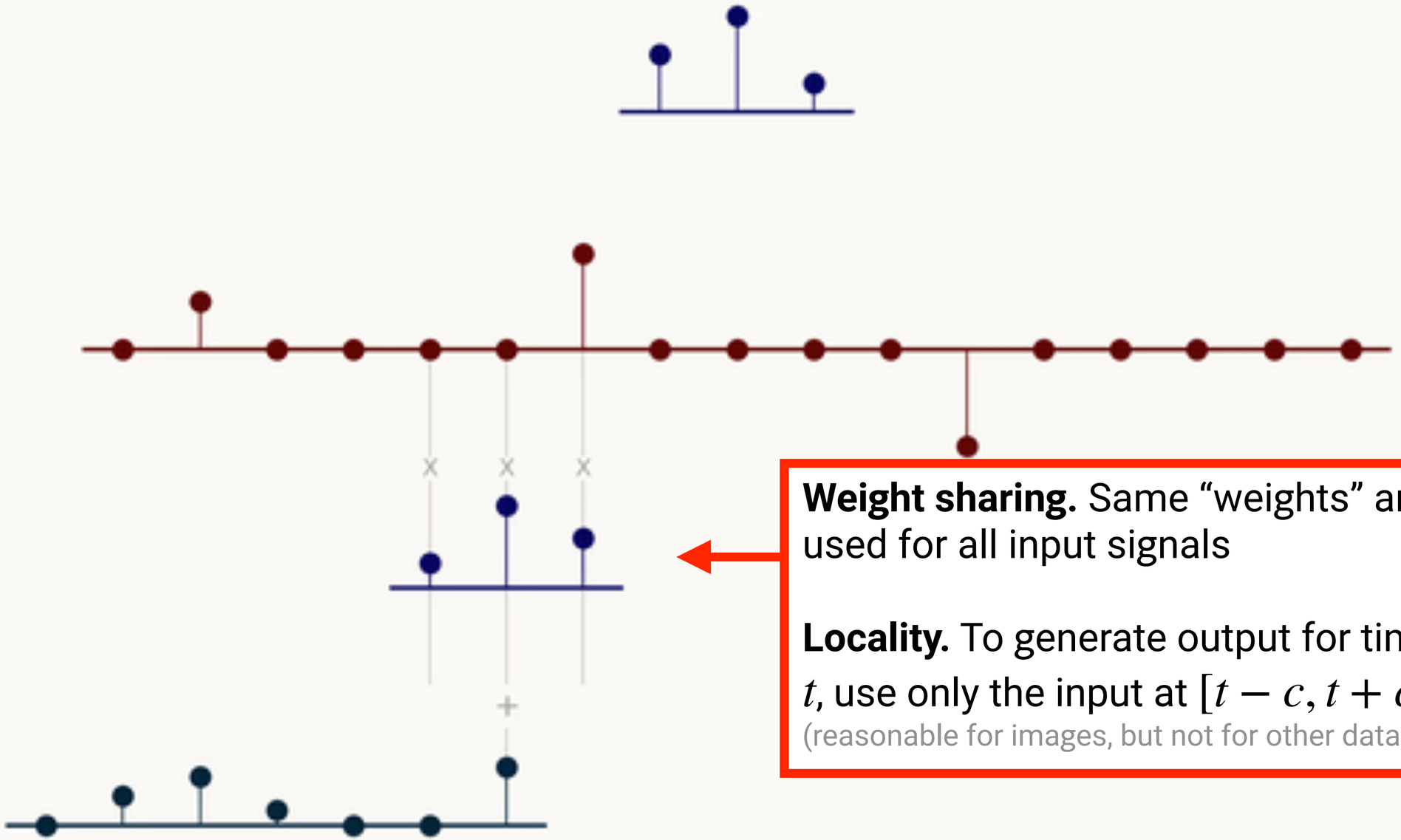


Image source: 1D convolution for neural networks, part 1: Sliding dot product, Brandon Rohrer





Weight sharing. Same "weights" are used for all input signals

Locality. To generate output for time t, use only the input at $[t-c, t+c]$
(reasonable for images, but not for other data?)

Image source: wikipedia “convolution”

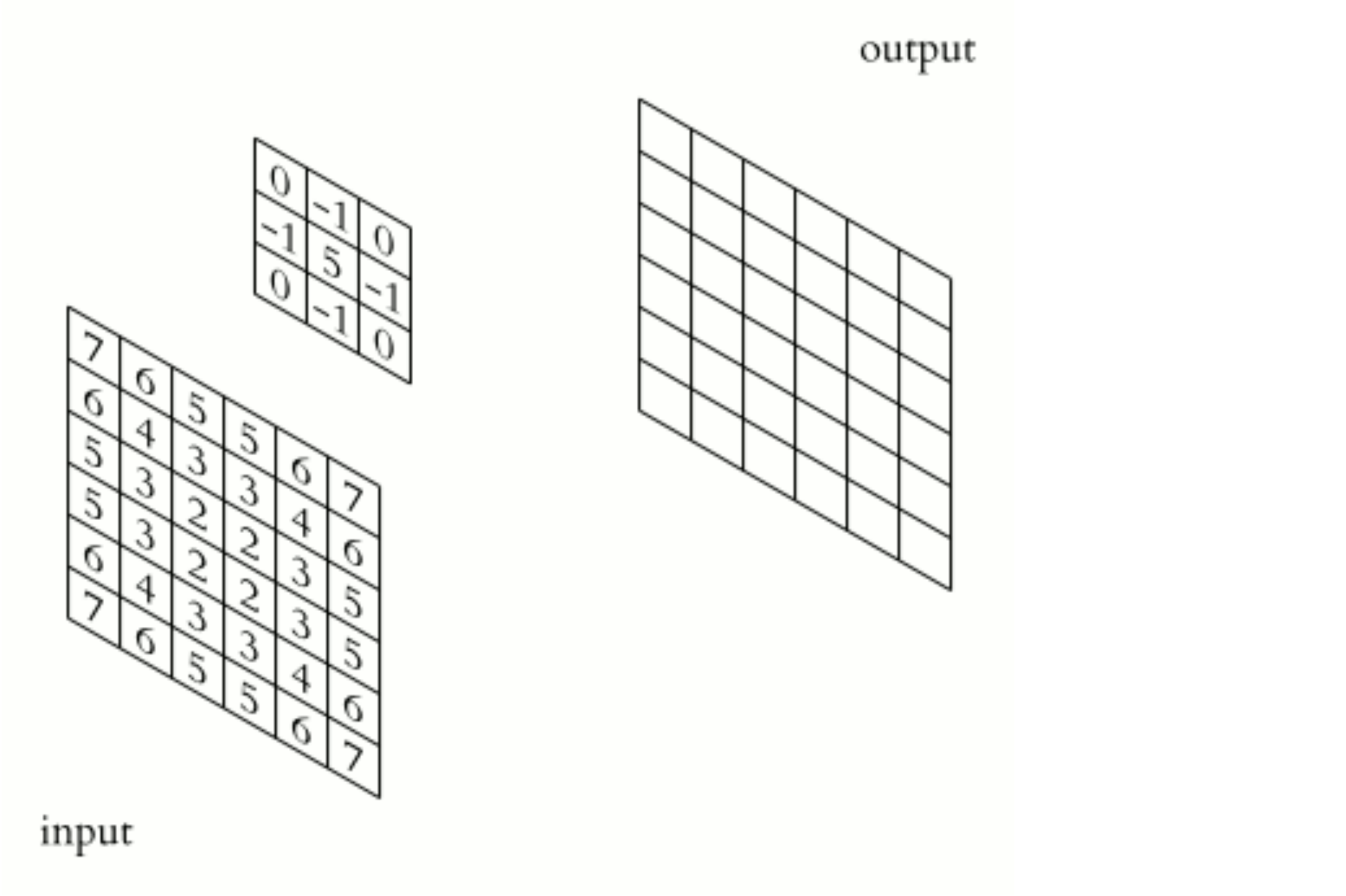
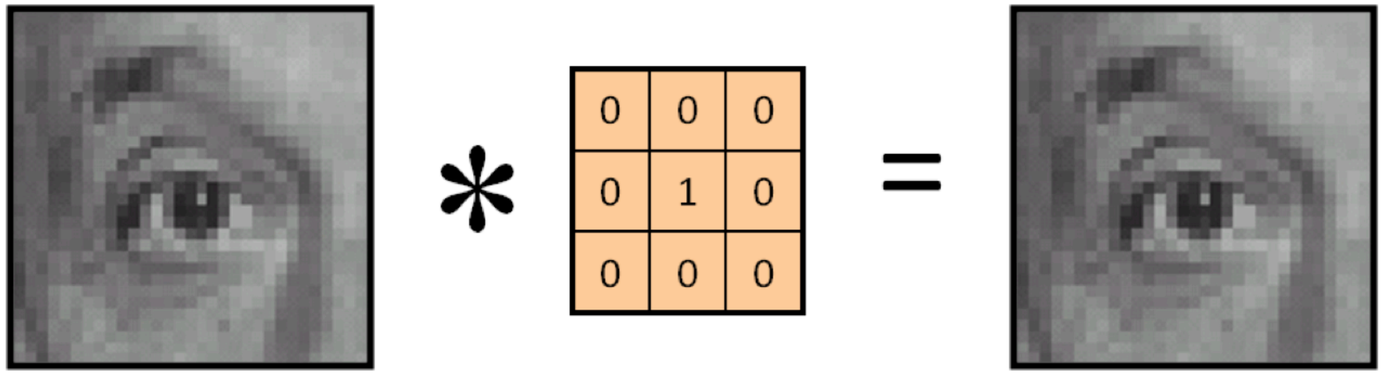


Image source: https://www.cs.cornell.edu/courses/cs1114/2013sp/sections/S06_convolution.pdf

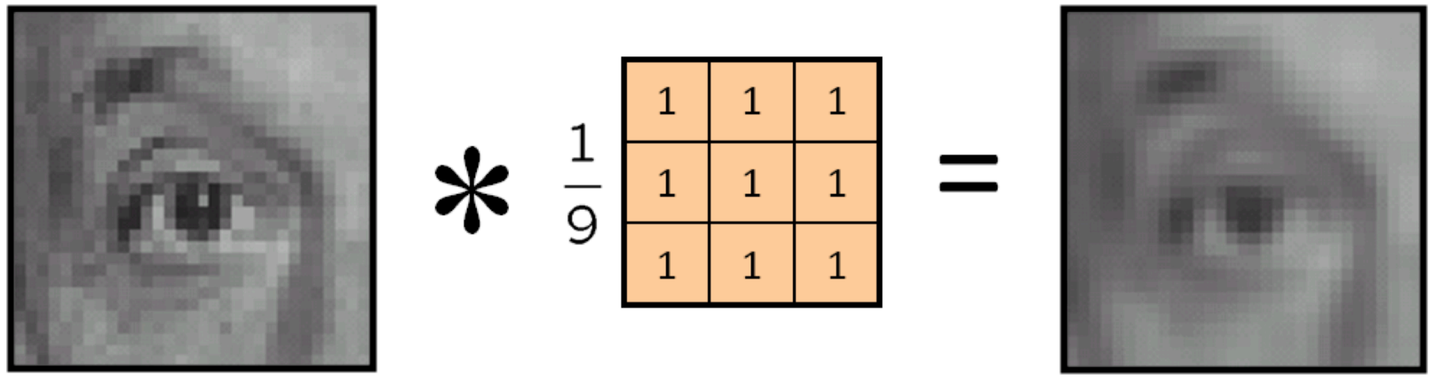
Convolution

- In classic computer vision, different handcrafted “filters” has been used for different purposes



Original

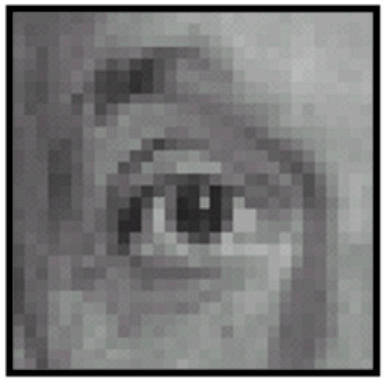
Identical image



Original

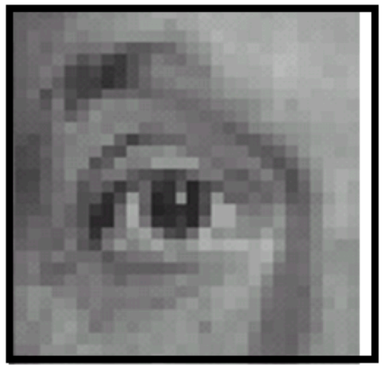
Blur (with a mean filter)

Convolution

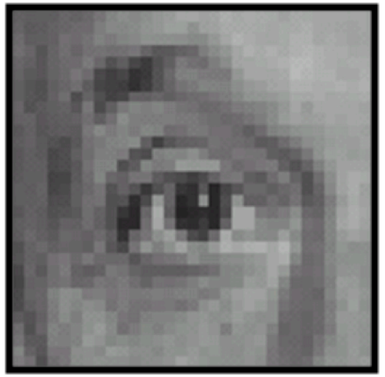


Original

$$*\begin{bmatrix} 0 & 0 & 0 \\ 1 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix} =$$



Shifted left
By 1 pixel



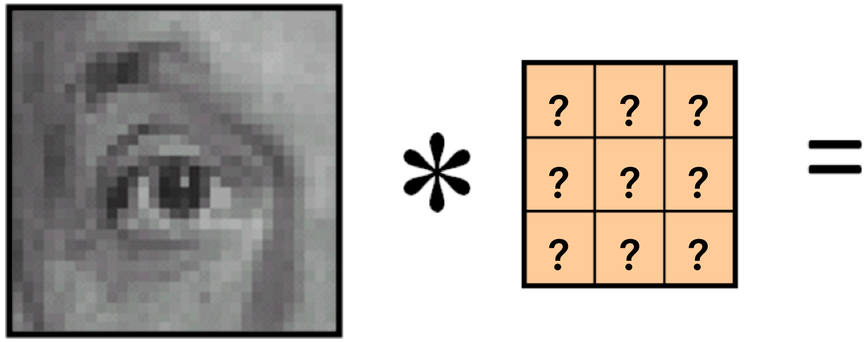
Original

$$*\left(\begin{bmatrix} 0 & 0 & 0 \\ 0 & 2 & 0 \\ 0 & 0 & 0 \end{bmatrix} - \frac{1}{9}\begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix}\right) =$$

Sharpening filter
(accentuates edges)

Convolution

- In modern AI, filters are learned from data
  - Replaces linear layers
  - Apply multiple parallel filters



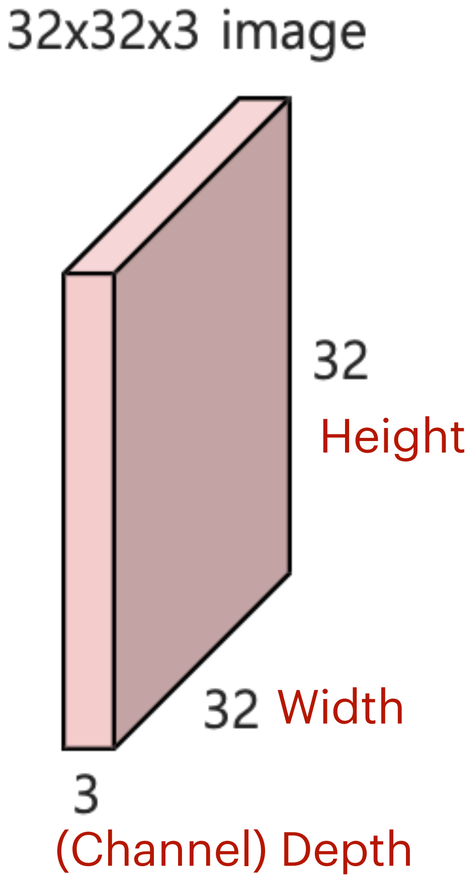
- **Properties.** Learned operations are translation-equivariant
  - applies same operation to patches at different locations

$$f(\text{shift}(\mathbf{x})) = \text{shift}(f(\mathbf{x}))$$

Building a convolutional network

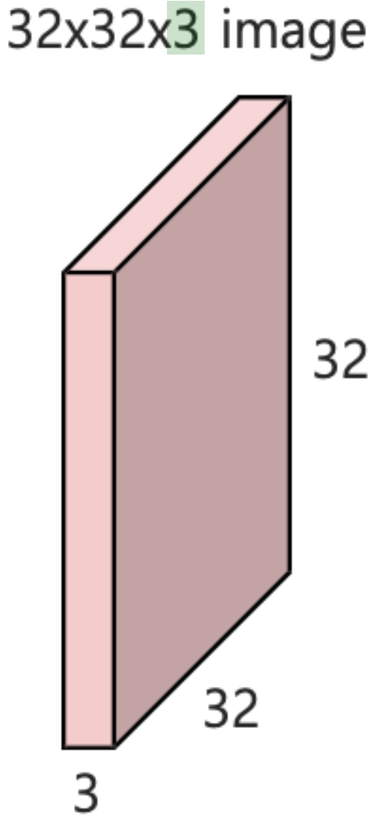
Convolutional layer

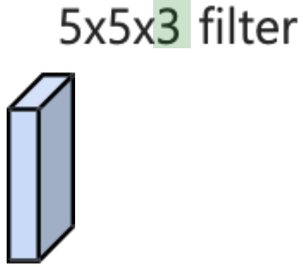
- Begin with a 32x32 image with 3 channels (RGB)



Convolutional layer

- Convolve with a convolution filter
  - Dot product with a sliding “receptive field”

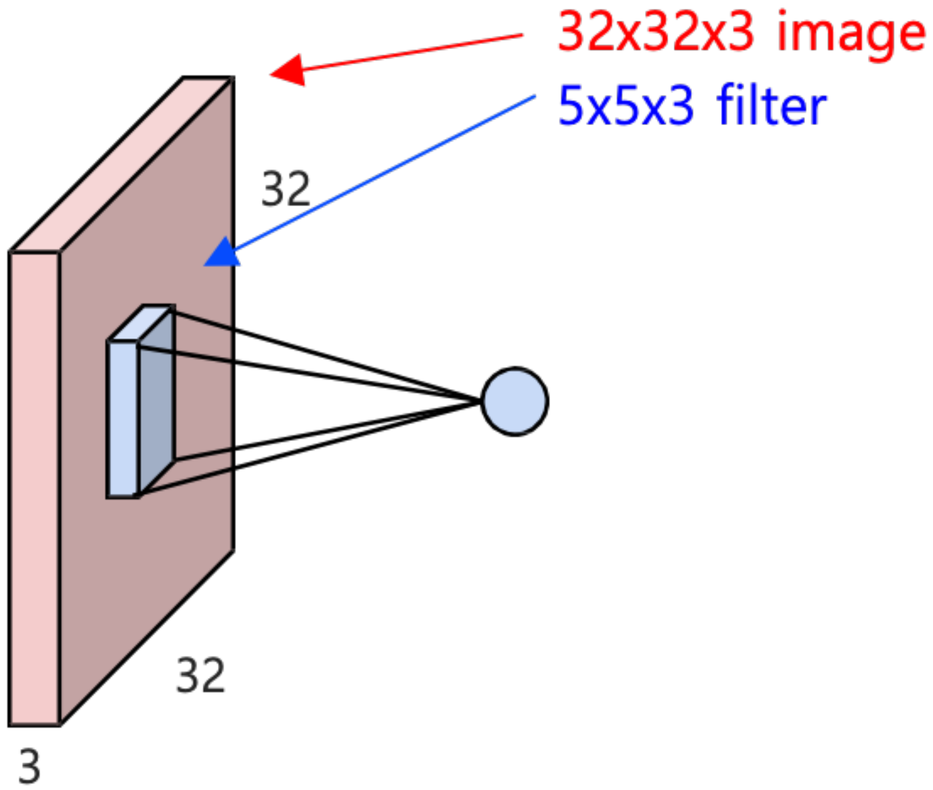




- **Classic.** Filters have full channel depth
  - i.e., uses all input channels
- **Modern.** Apply depth-1 convolution for each input channel, for efficiency
  - called “depthwise convolution”

Convolutional layer

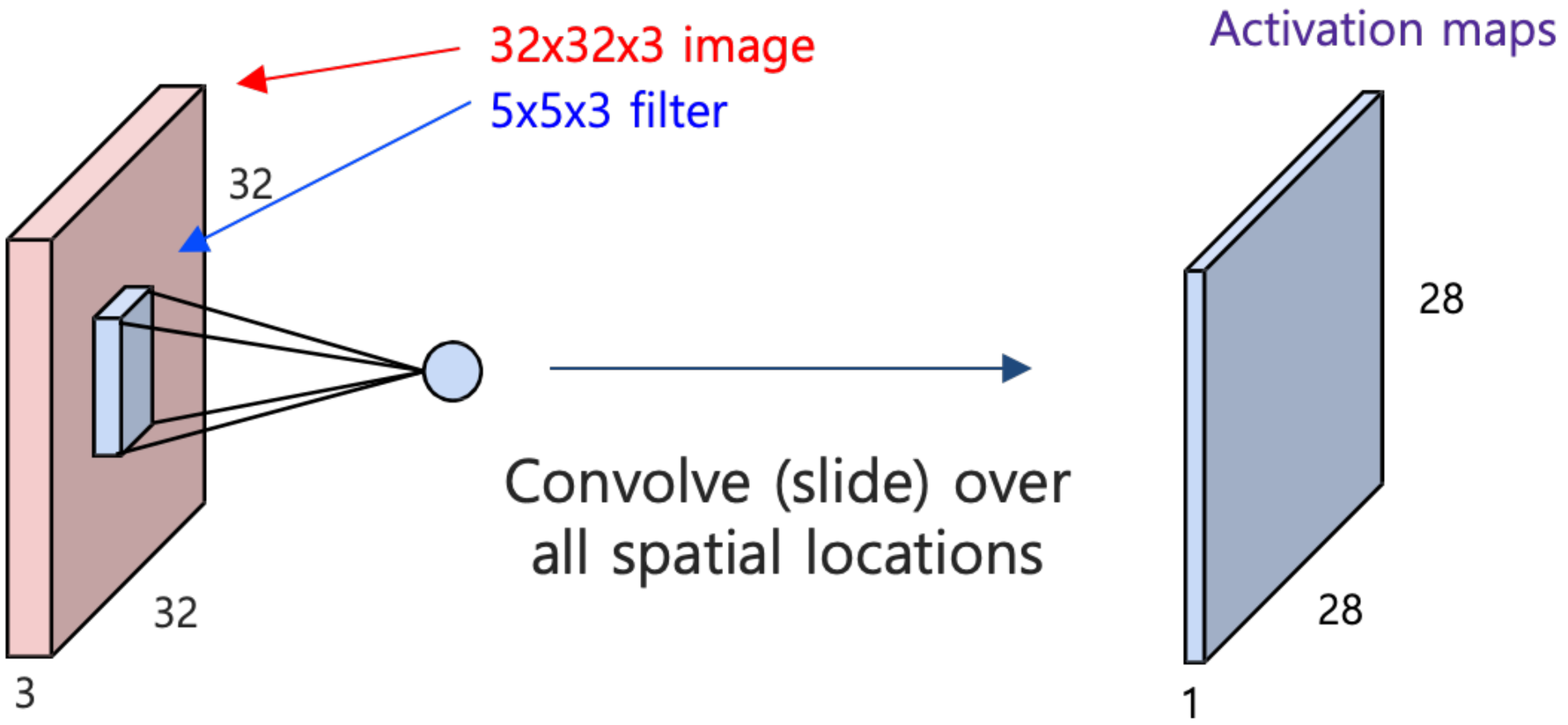
- Convolving generates a single entry of the layer output



- **Compute.** Dot product of two tensors with 5x5x3 =75 dimensions, plus a bias addition

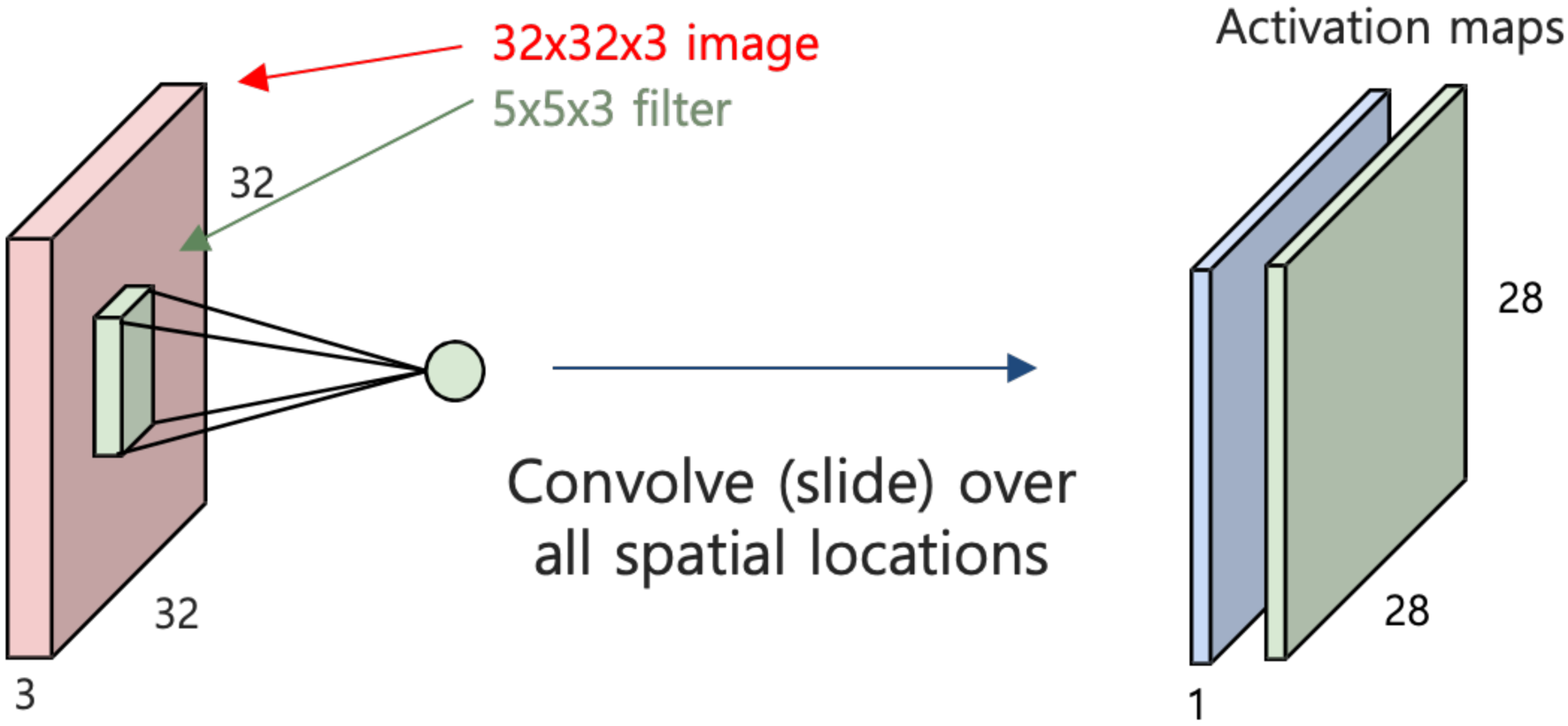
Convolutional layer

- Convolving generates a single entry of the layer output



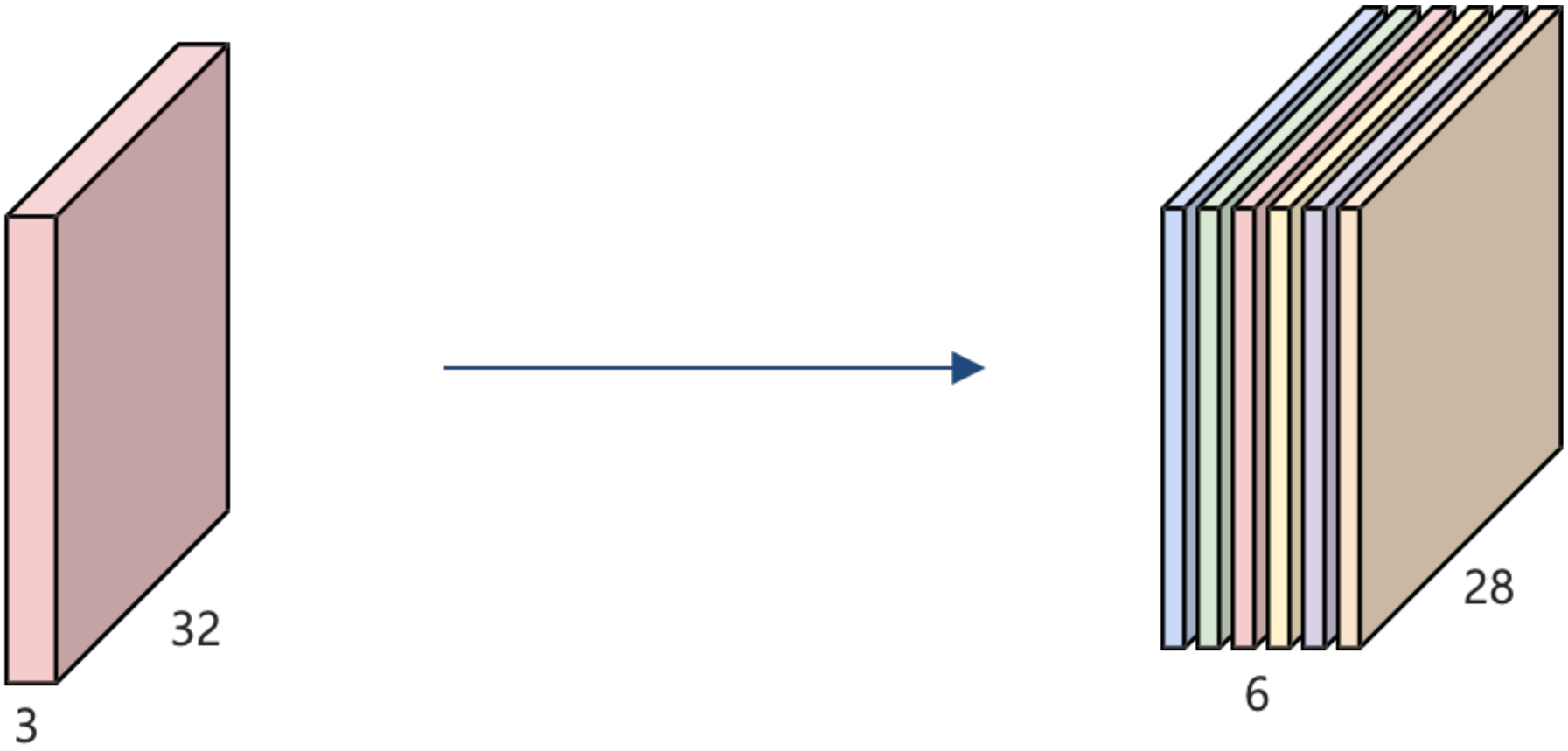
Convolutional layer

- Do the same for the second filter



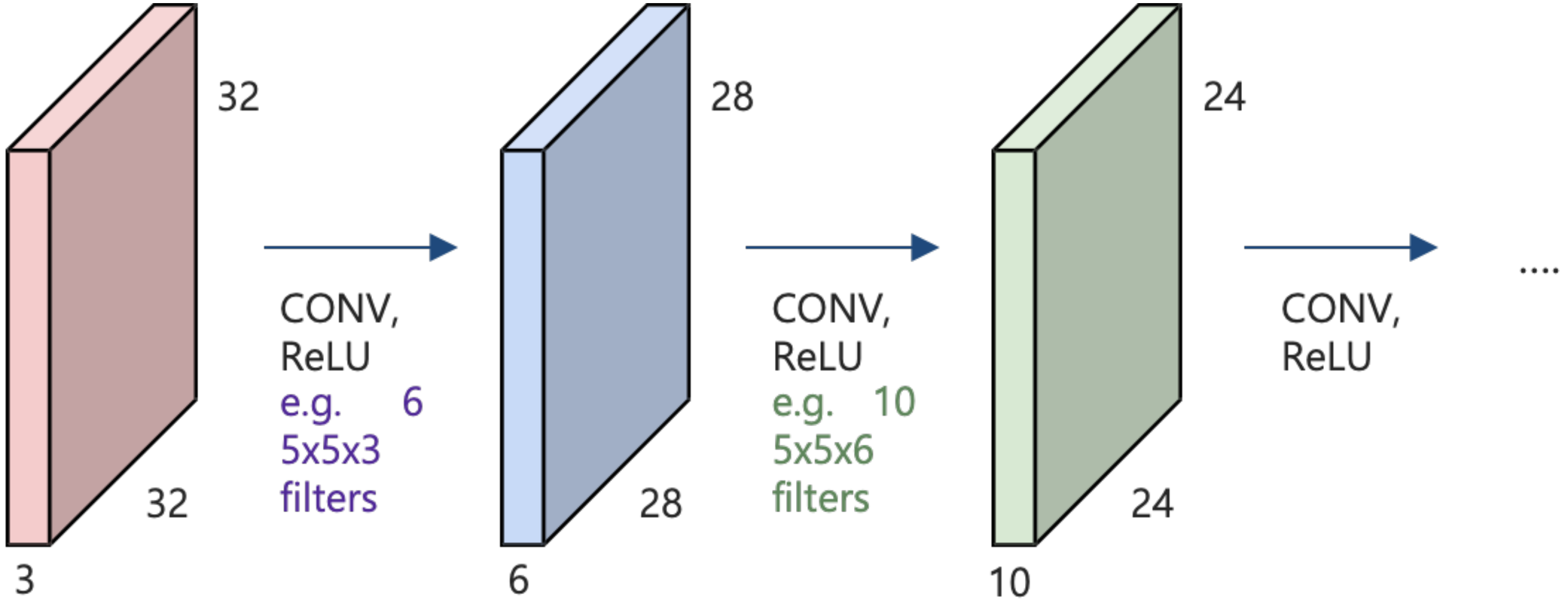
Convolutional layer

- Generate the pre-activation with multiple depths
  - also called “channels”



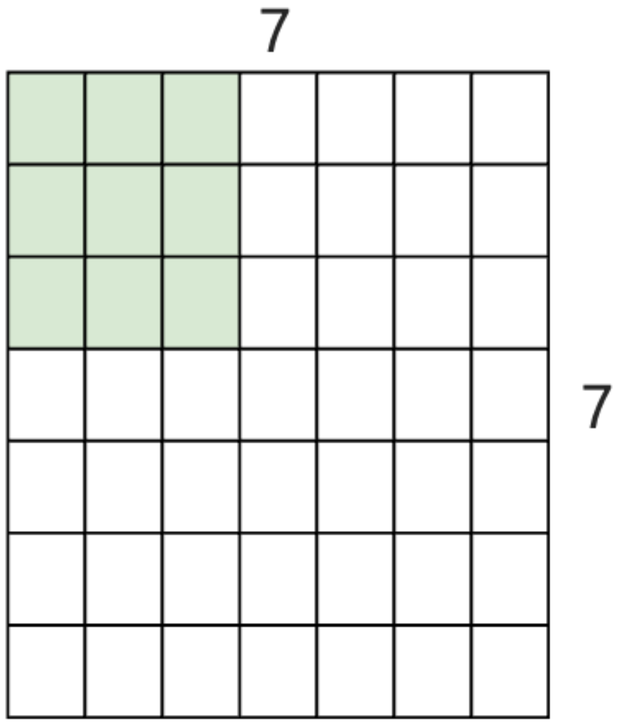
Convolutional layer

- Stack the layers, with activation functions in between



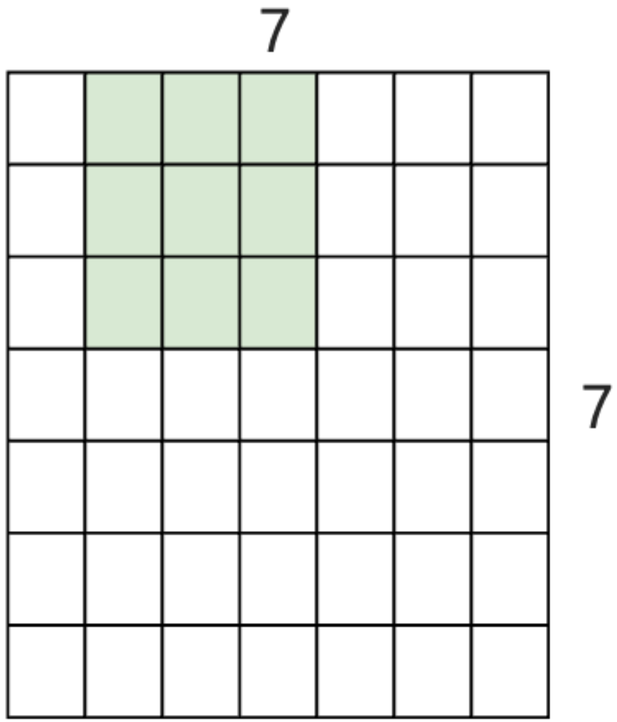
Spatial dimension: Stride

- Consider a 7x7 image with 3x3 filters



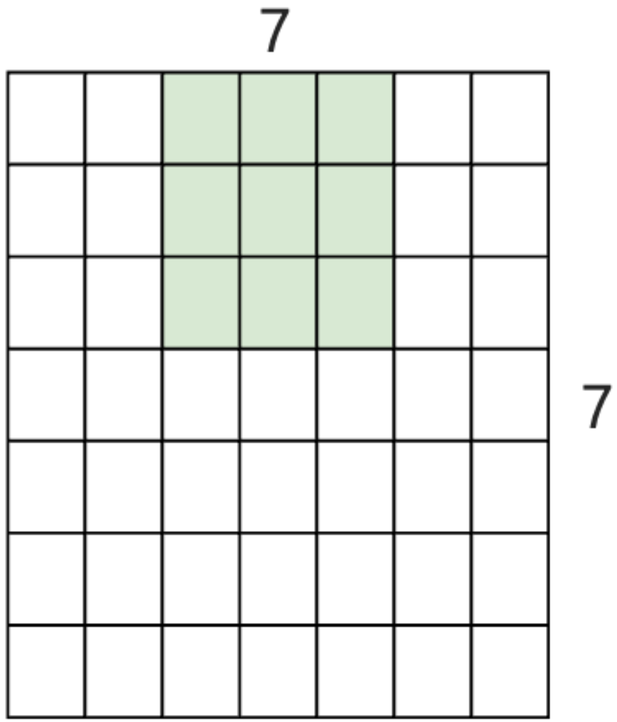
Spatial dimension: Stride

- Consider a 7x7 image with 3x3 filters



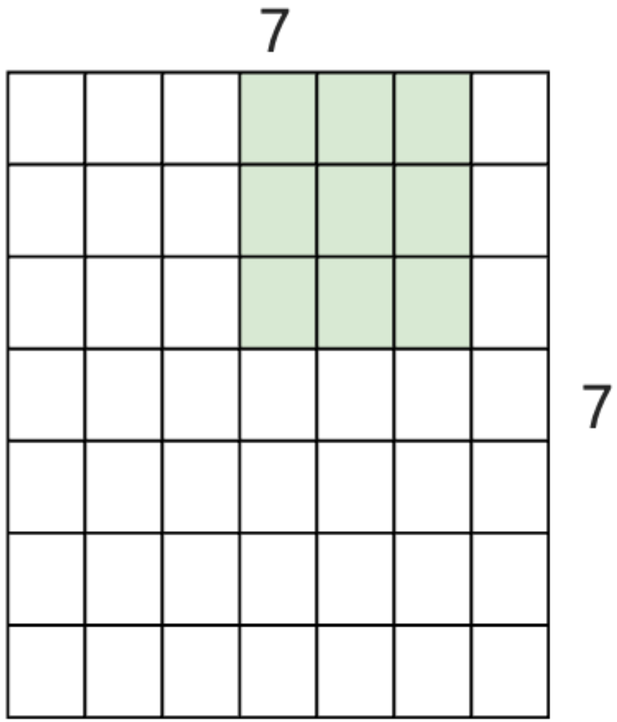
Spatial dimension: Stride

- Consider a 7x7 image with 3x3 filters



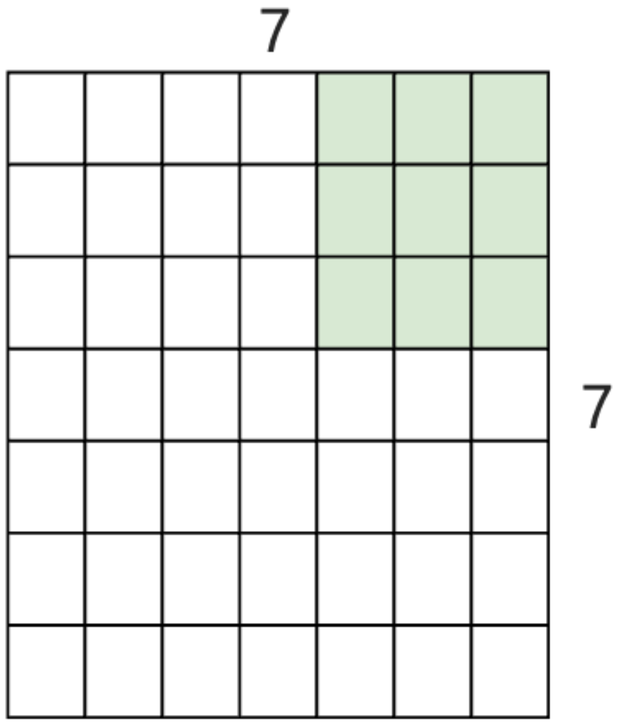
Spatial dimension: Stride

- Consider a 7x7 image with 3x3 filters



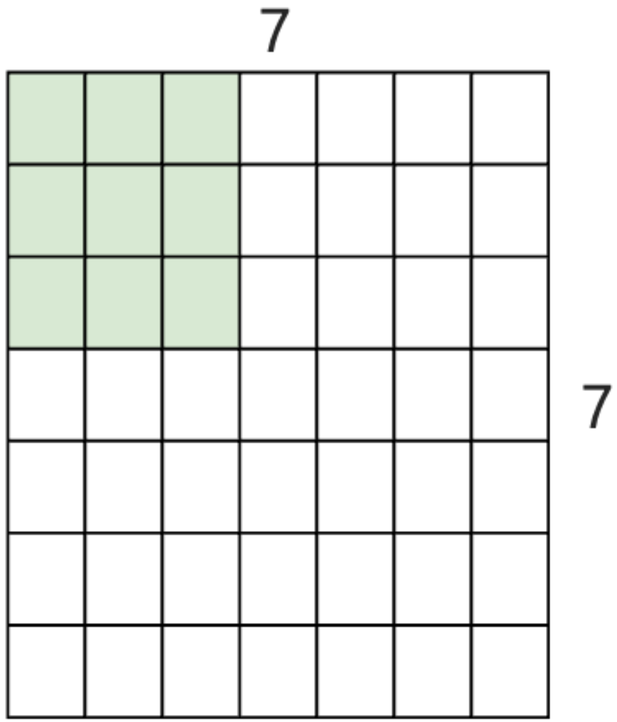
Spatial dimension: Stride

- Consider a 7x7 image with 3x3 filters = 5x5 output



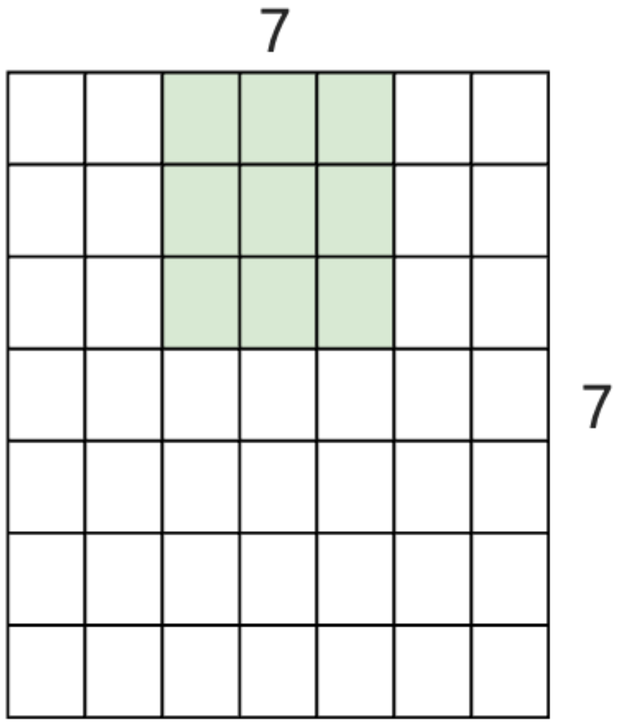
Spatial dimension: Stride

- It is common to apply strides:
  - With stride 2



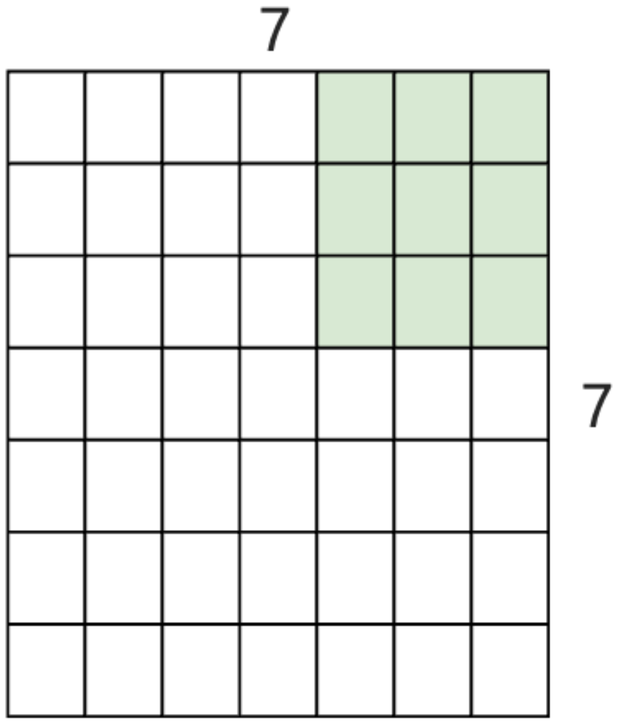
Spatial dimension: Stride

- It is common to apply strides:
  - With stride 2



Spatial dimension: Stride

- It is common to apply strides:
  - With stride 2 = 3x3 output



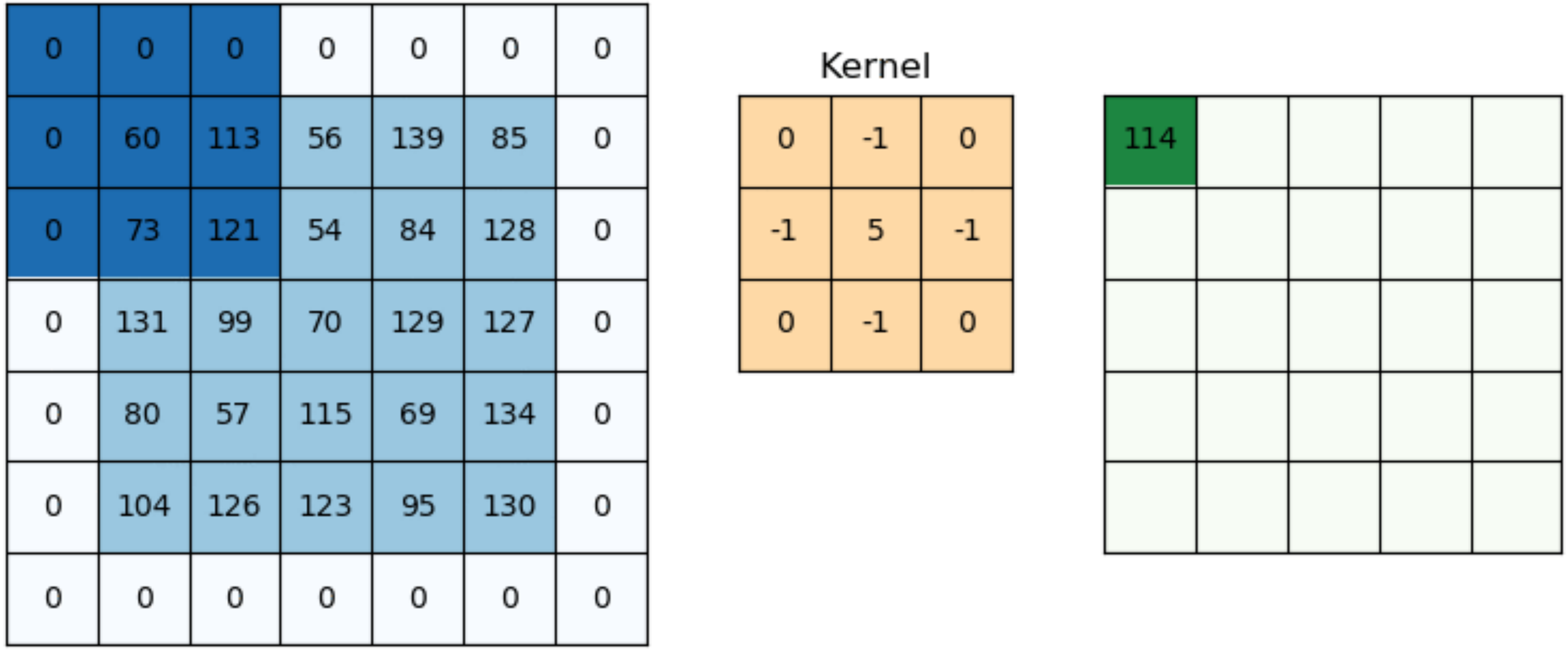
- **Output size.** will be:

$$\frac{(\text{img length} - \text{filter length})}{\text{stride}} + 1$$

  - Stride 1: (7-3)/1 + 1 = 5
  - Stride 2: (7-3)/2 + 1 = 3
  - Stride 4: (7-3)/4 + 1 = 2
- Note. The stride 3 does not fit for this case

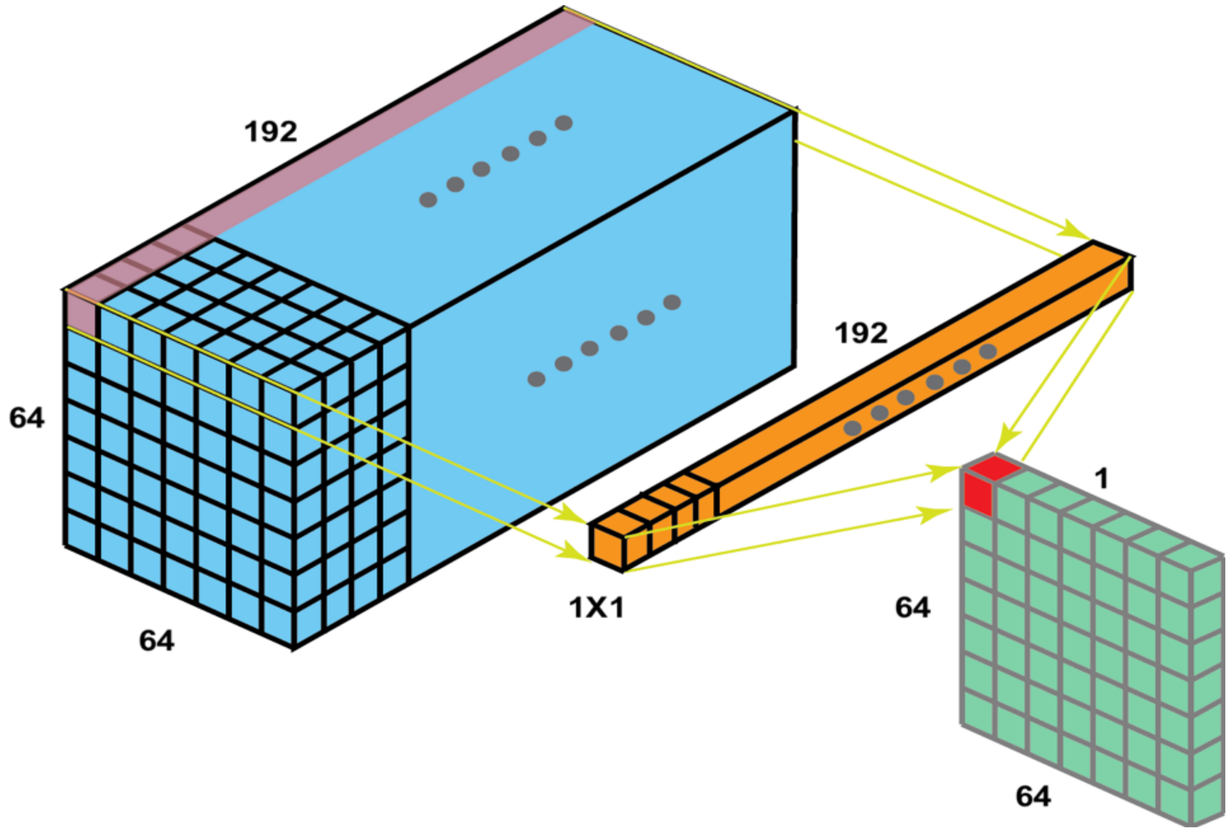
Spatial dimension: Padding

- **Zero-padding.** Adding zeros to the side, before applying convolution
  - Image size does not reduce, and thus can use more layers



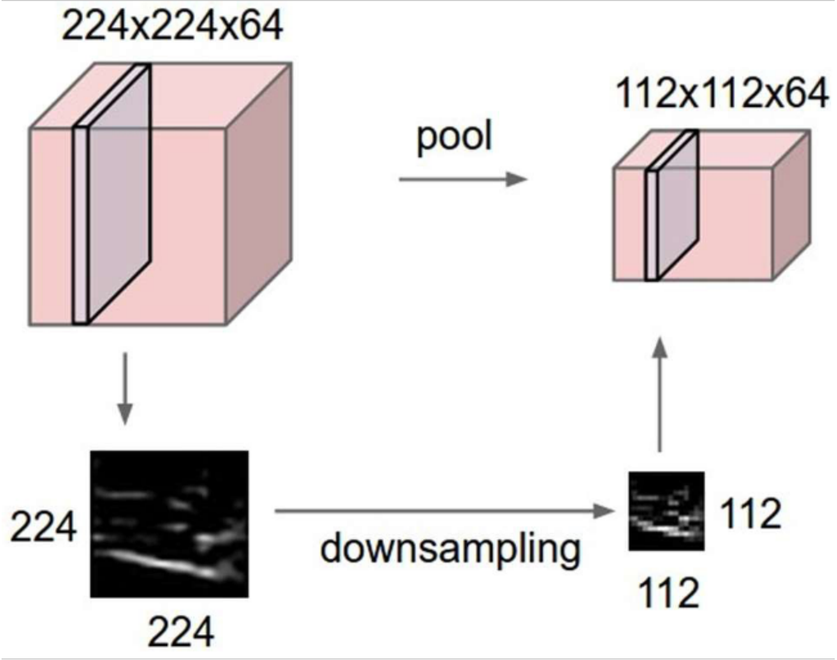
Spatial dimension: 1x1

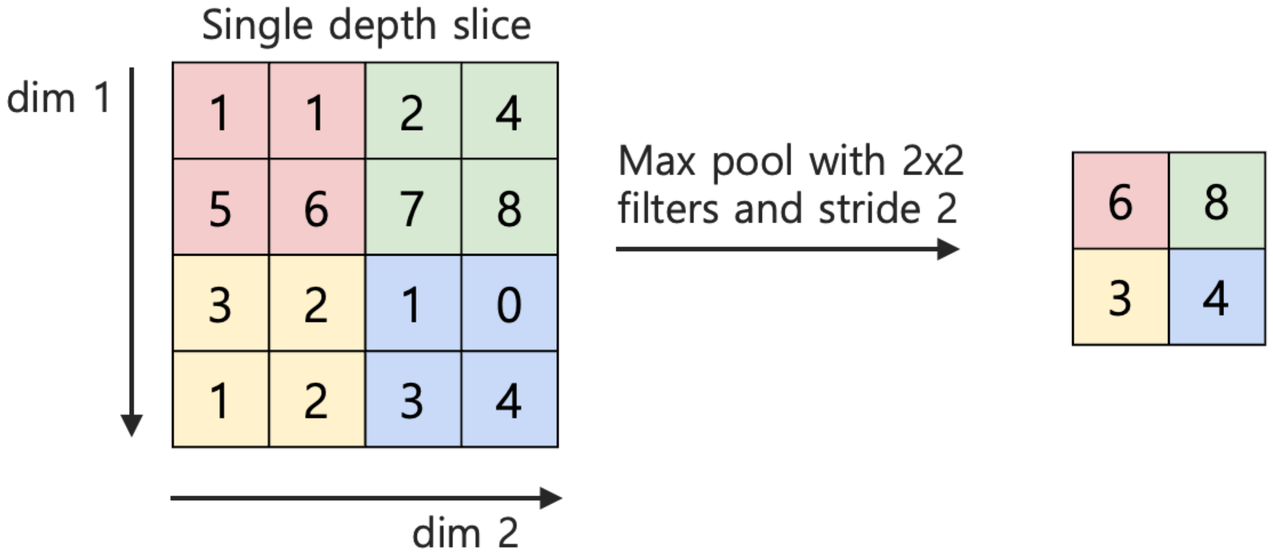
- Sometimes, we use 1x1 convolutions
  - Increase or decrease the number of channels via linear combination
  - Often used together with depthwise convolution



Pooling layer

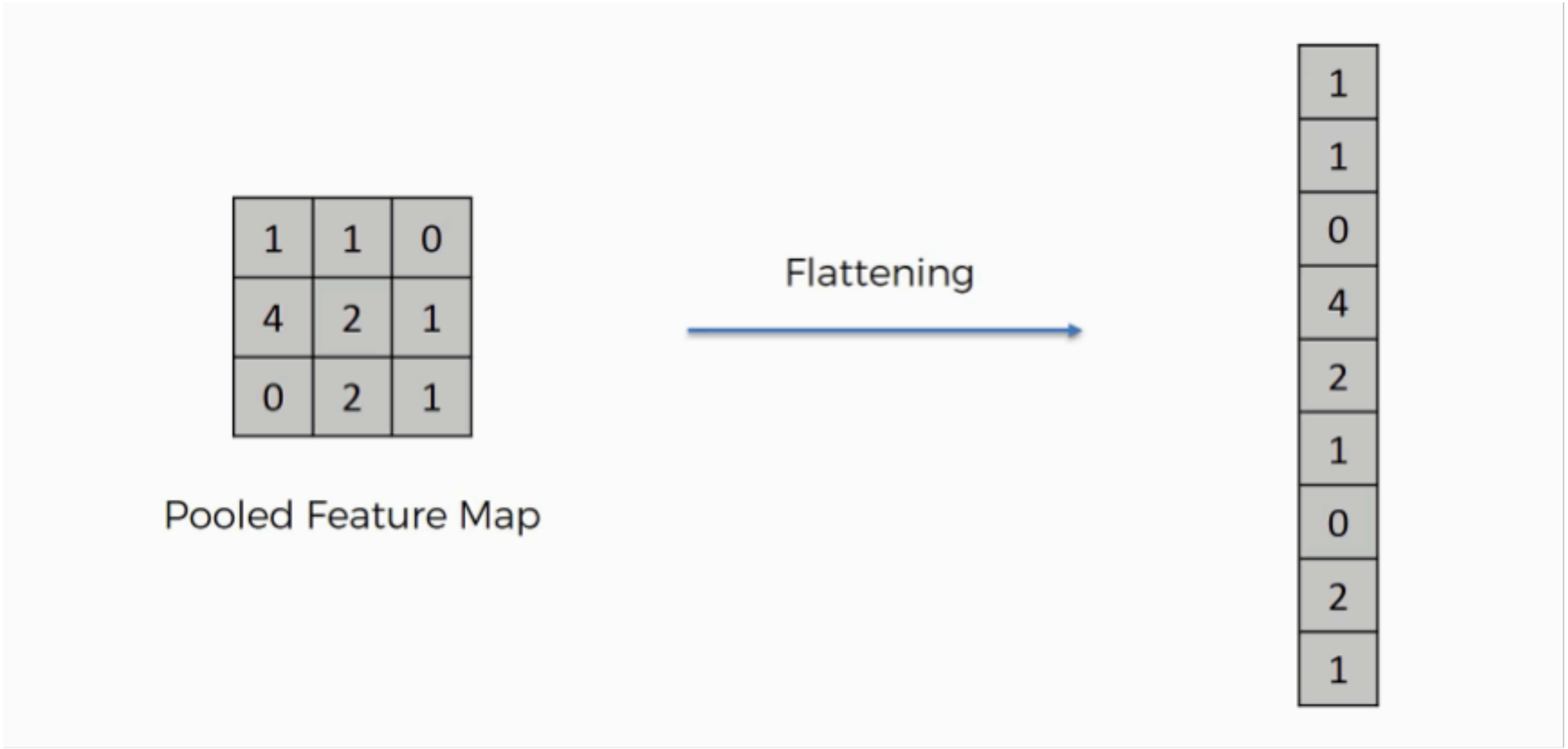
- **Pooling.** Reduces the spatial dimension by taking max/mean/else of multiple adjacent pixels
  - Gets smaller resolution without losing information (e.g., the activation represents a specific feature)





Final layer — Fully-connected

- In the final layer, we typically use fully-connected (i.e., linear) layer to perform classification/regression
  - To do this, we “flatten” the 2D/3D features into a vector form



Additional Remarks

- Each convolutional layer can handle images of any size
  - Unlike a fully-connected layer, which can only handle fixed dimension
- For denoising/segmentation tasks, we can use all-convolutional networks to process images of any size

Popular network architectures

Architectures

- Now let’s take a look at some popular CNN architectures
  - **Basic.** LeNet, AlexNet
  - **Deep.** VGG, GoogLeNet, ResNet
  - **Tiny.** MobileNets
  - **Scalable.** EfficientNets / NFNets

- **Others.** U-Net

Basic models

LeCun et al., “Gradient-based learning applied to document recognition,” ACM DAES 2022

LeNet-5 (1998)

- First practically useful ConvNet
  - Convolutional layers followed by fully-connected layers
  - Pooling after convolution

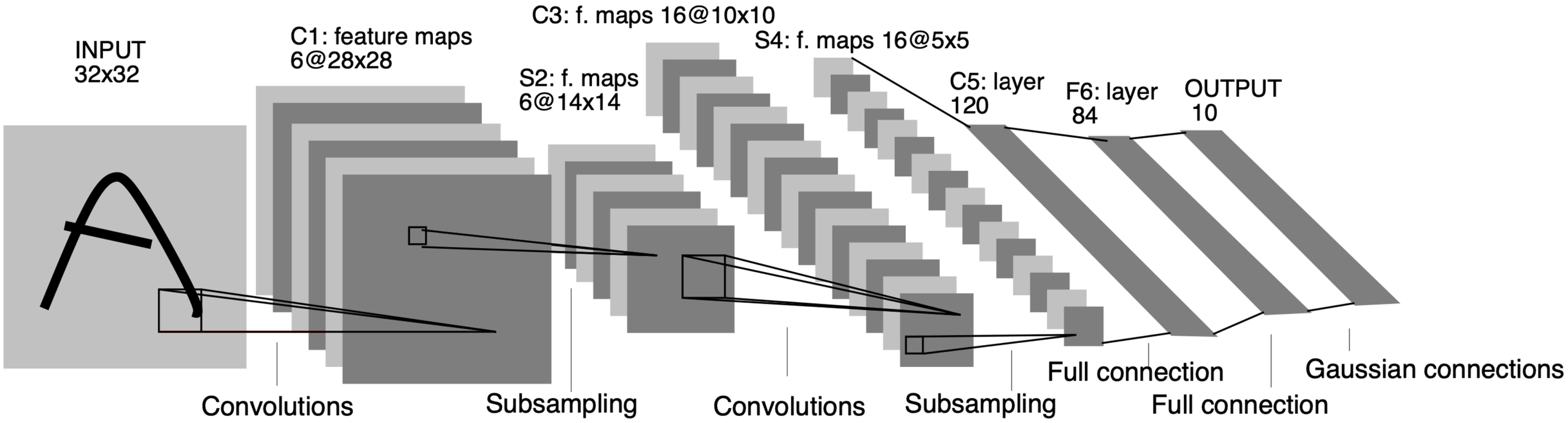


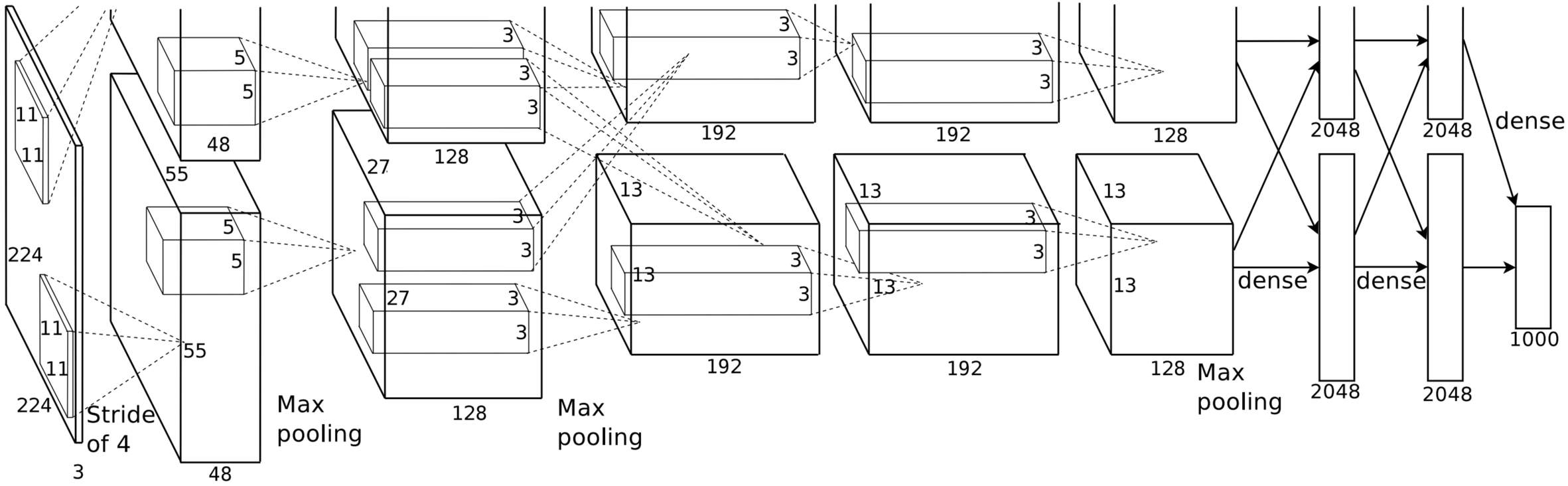
Fig. 2. Architecture of LeNet-5, a Convolutional Neural Network, here for digits recognition. Each plane is a feature map, i.e. a set of units whose weights are constrained to be identical.


4627
4 6 2 7

Krizhevsky et al., “ImageNet Classification with Deep Convolutional Neural Networks,” NeurIPS 2012

AlexNet (2012)

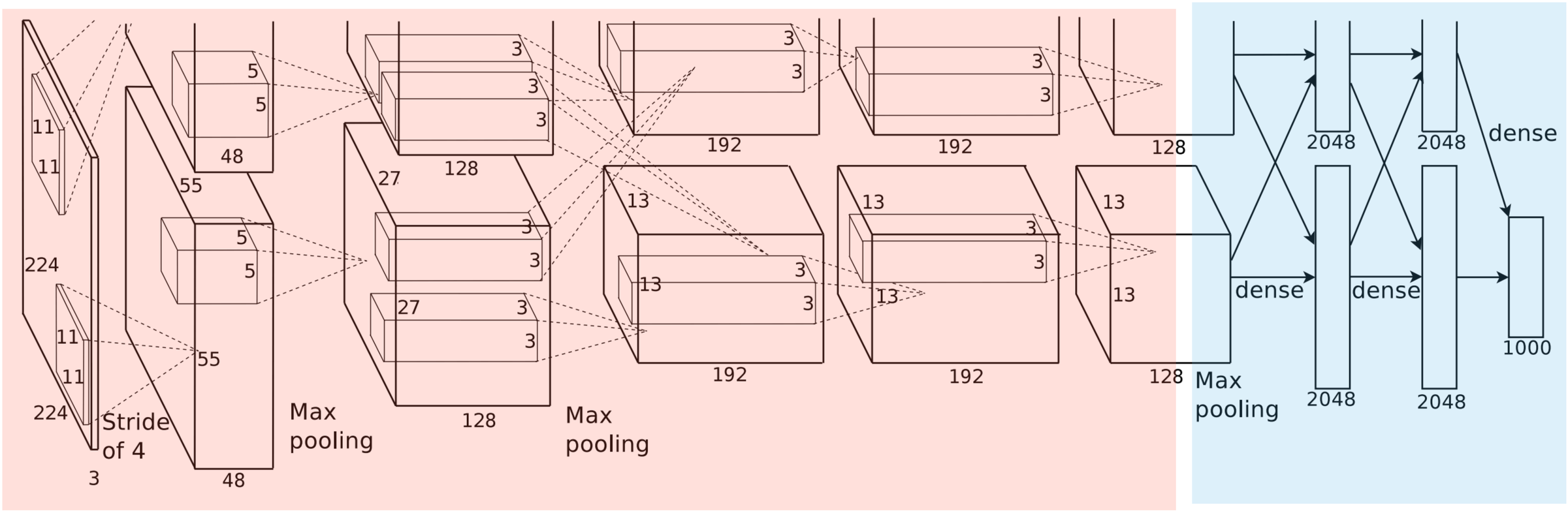
- Bigger and deeper LeNet — 2012 ImageNet challenge winner
  - 7 hidden layers, 60 million parameters, and dropout



AlexNet (2012)

- Bigger and deeper LeNet — 2012 ImageNet challenge winner
  - 7 hidden layers, 60 million parameters, and dropout

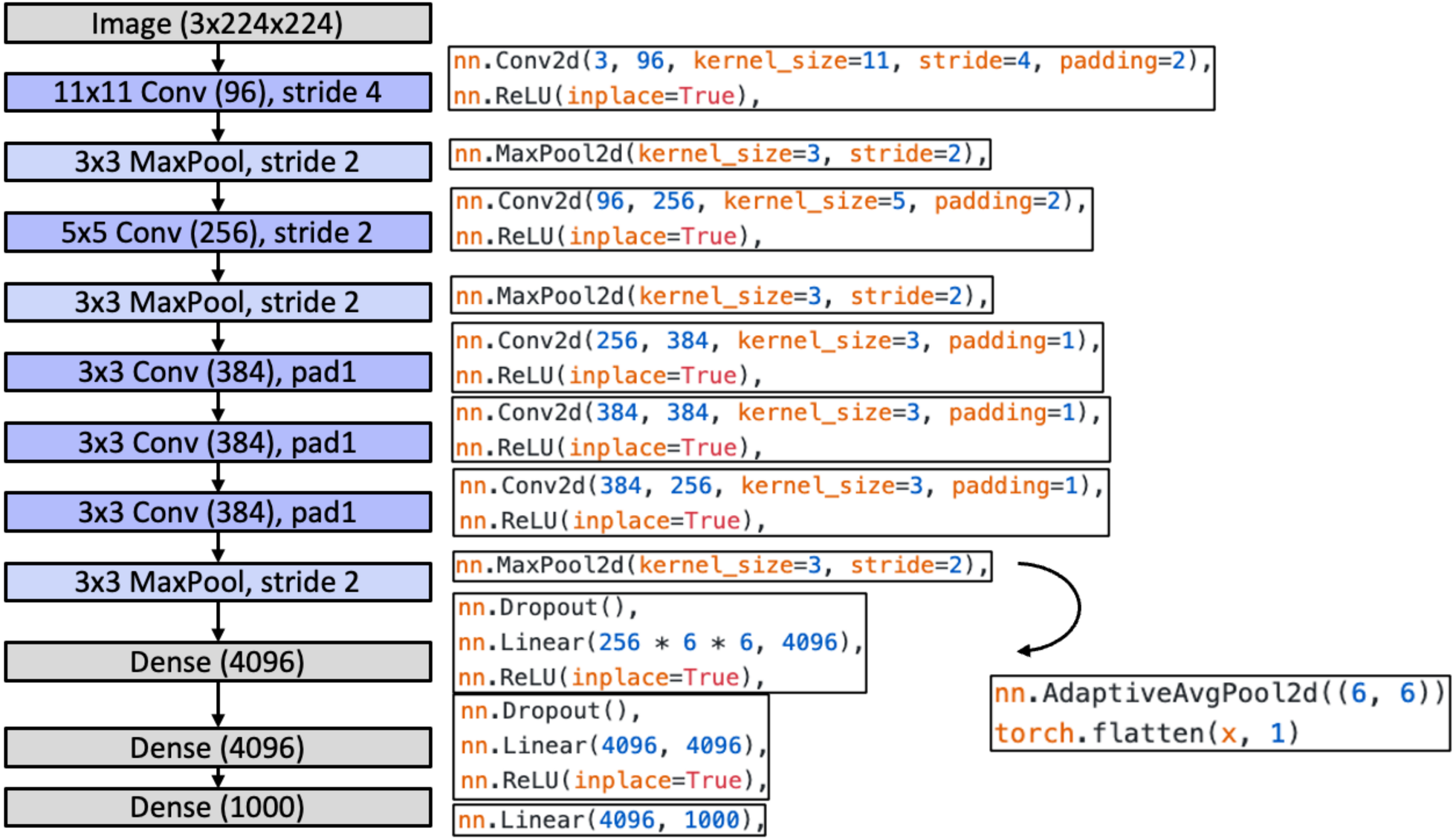
Conv-Pool-Norm.-Conv-Pool-Norm.-Conv-Conv-Conv-Pool-FC-FC-FC



AlexNet (2012)

- How did they succeed in training bigger and deeper networks?
  - **Dataset.** Large-scale, high-quality dataset
    - ImageNet
  - **Optimization.** Better activation
    - ReLU
  - **Generalization.** Better regularization
    - DropOut
  - **Computation.** Faster hardware
    - Distributed GPU training (two GTX 580)
- And thus the scale race began

AlexNet (2012)



AlexNet (2012)

SuperVision
(Krizhevsky, Sutskever, Hinton)

Our model is a large, deep convolutional neural network trained on raw RGB pixel values. The neural network, which has 60 million parameters and 650,000 neurons, consists of five convolutional layers, some of which are followed by max-pooling layers, and three globally-connected layers with a final 1000-way softmax. It was trained on two NVIDIA GPUs for about a week. To make training faster, we used non-saturating neurons and a very efficient GPU implementation of convolutional nets. To reduce overfitting in the globally-connected layers we employed hidden-unit "dropout", a recently-developed regularization method that proved to be very effective.

VGG
(Simonyan, Aytar, Vedaldi, Zisserman)

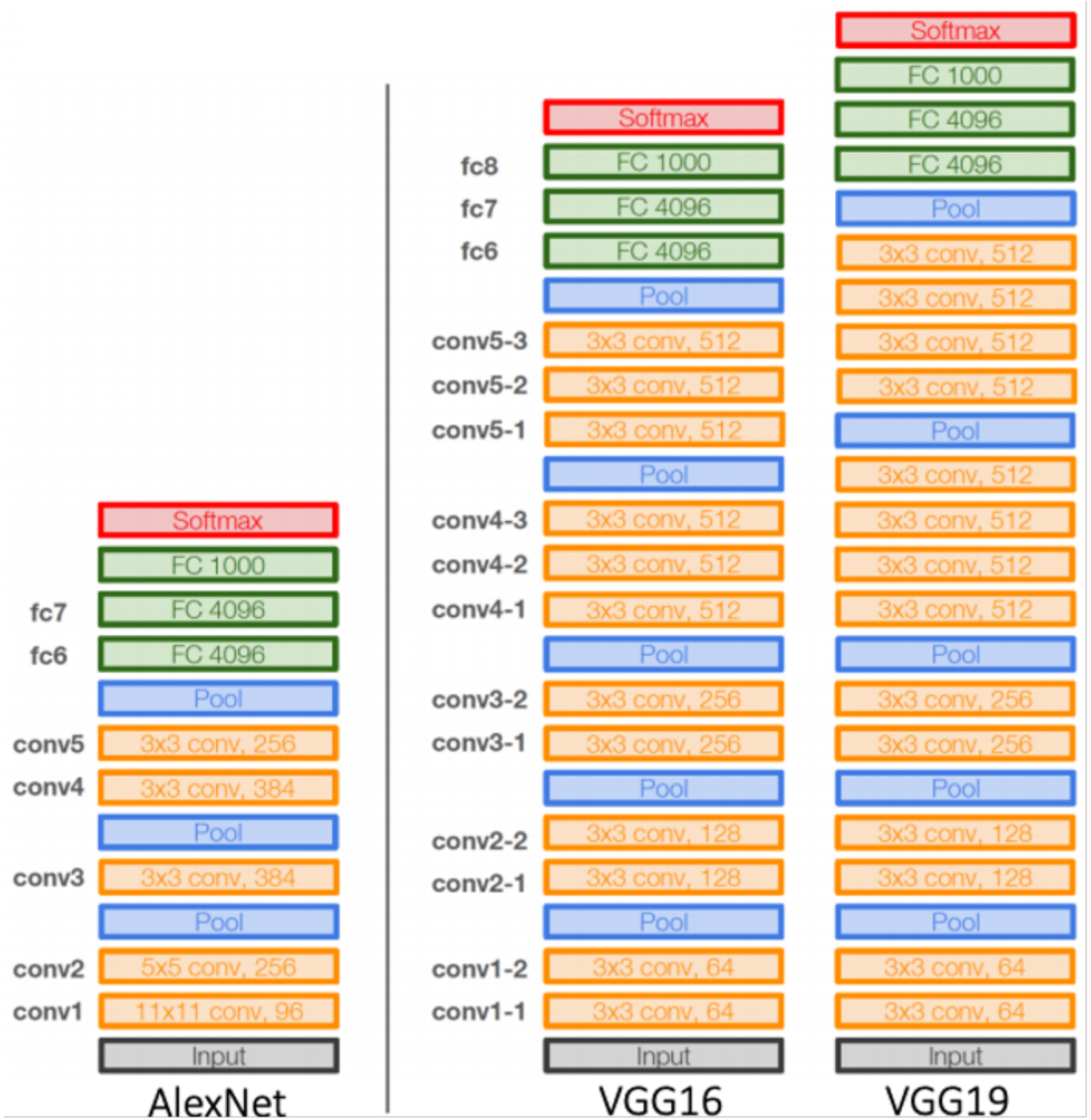
In this submission, image classification was performed using a conventional pipeline based on Fisher vector image representation and one-vs-rest linear SVM classifiers. In more detail, two types of local patch features were densely extracted over multiple scales: SIFT and colour statistics. The features were then augmented with patch spatial coordinates and aggregated into two Fisher vectors corresponding to the two feature types. Fisher vectors were computed using GMMs with 1024 Gaussians, resulting in 135K-dimensional representations. To obtain a single feature vector per image, the two Fisher vectors were then stacked. We did not use spatial pyramid representation. To be able to deal with large amounts of training data, product quantisation was employed to compress the image features. Finally, an ensemble of one-vs-rest linear SVMs was trained over stacked features using stochastic sub-gradient method (Pegasos).

Deep models

Simonyan and Zisserman, “Very deep convolutional networks for large-scale image recognition,” ICLR 2015

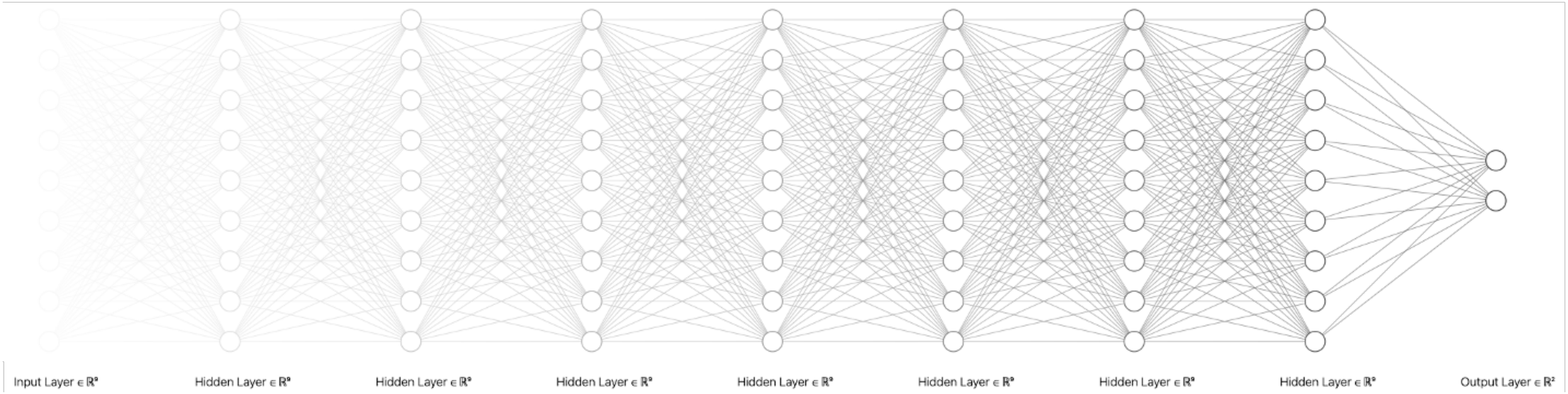
VGG (2014)

- **Deeper network**
  - up to 19 layers
- **Simpler architecture**
  - Only 3x3 convolution
    - vs 5x5. Less params but deeper
  - Only 2x2 pooling
  - No “local response normalization”



Stacking Deeper

- **Key obstacles**
  - Gradient vanishing and exploding (no batchnorm back then)
  - Too many parameters



Stacking Deeper

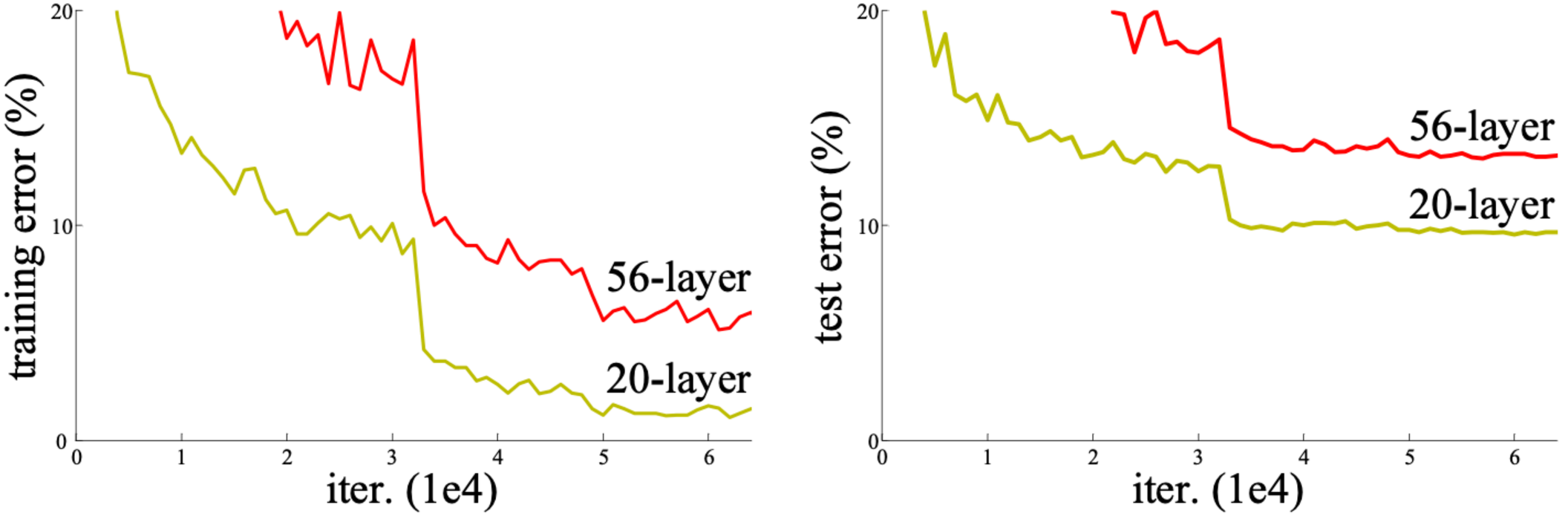
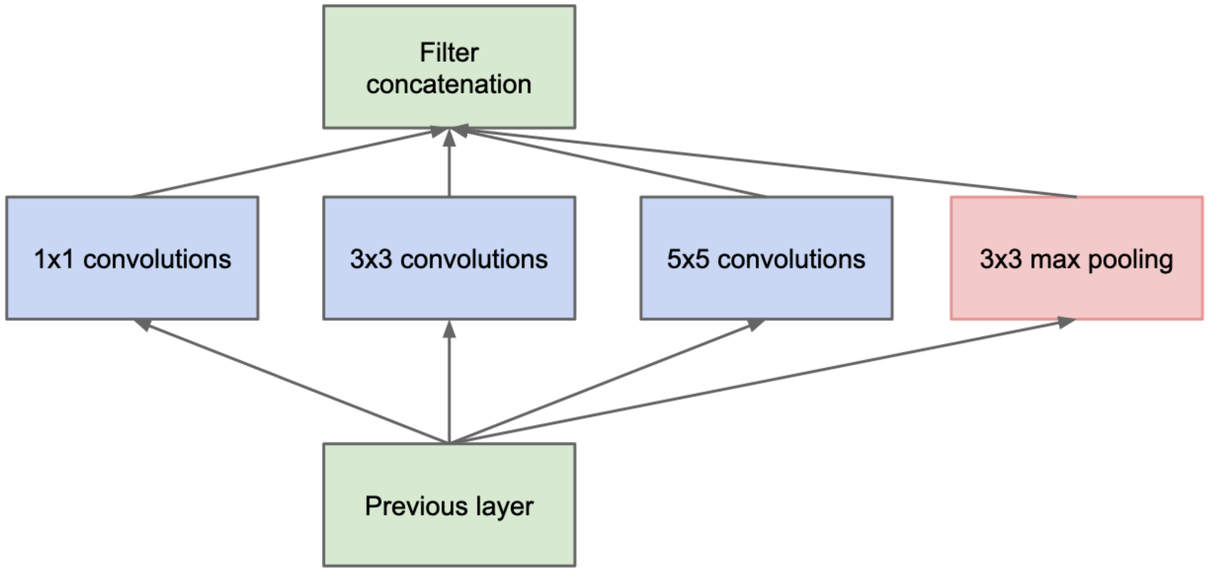


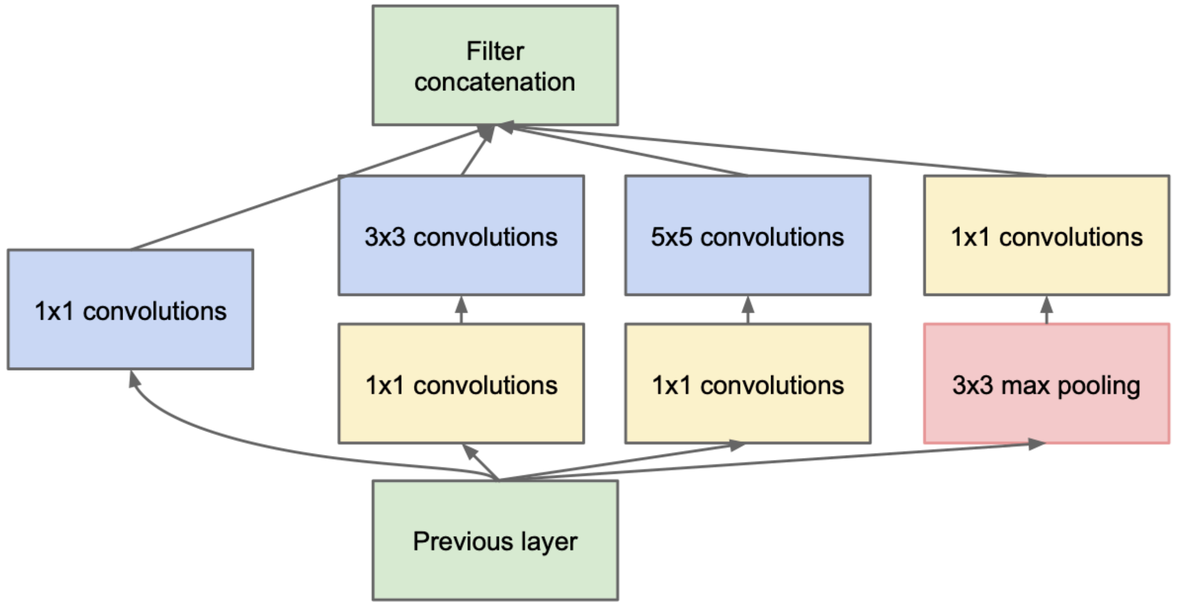
Figure 1. Training error (left) and test error (right) on CIFAR-10 with 20-layer and 56-layer “plain” networks. The deeper network has higher training error, and thus test error. Similar phenomena on ImageNet is presented in Fig. 4.

GoogLeNet (2014)

- Two notable differences
- **Inception module.** Works as a bottleneck to reduce #channels



(a) Inception module, naïve version

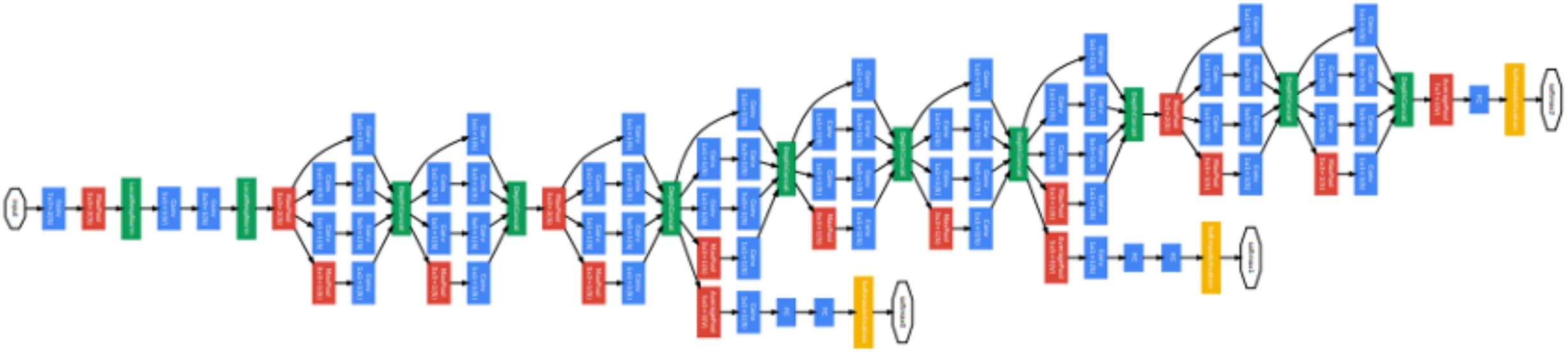


(b) Inception module with dimension reductions

Figure 2: Inception module

GoogLeNet (2014)

- **Auxiliary Classifier.** Resolves vanishing gradient by adding more sources of the backpropagation signal
  - Note. Only one FC layer for classification (reduces #params)



He et al., “Deep residual learning for image recognition,” CVPR 2016

ResNet (2016)

- A more elegant solution to the vanishing gradient

- **Residual learning.** Outputs $\mathbf{x} + f_\theta(\mathbf{x})$, not $f_\theta(\mathbf{x})$
  - f_θ represents an update that each layer contribute to the features
- Example. Suppose that $\mathbf{y} = x + \sigma(wx)$
  - Then, we have

$$\frac{\partial y}{\partial x} = 1 + \sigma'(wx)w$$

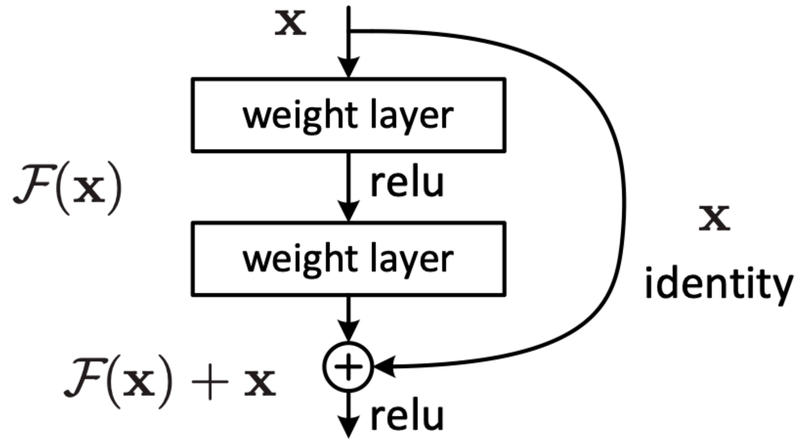
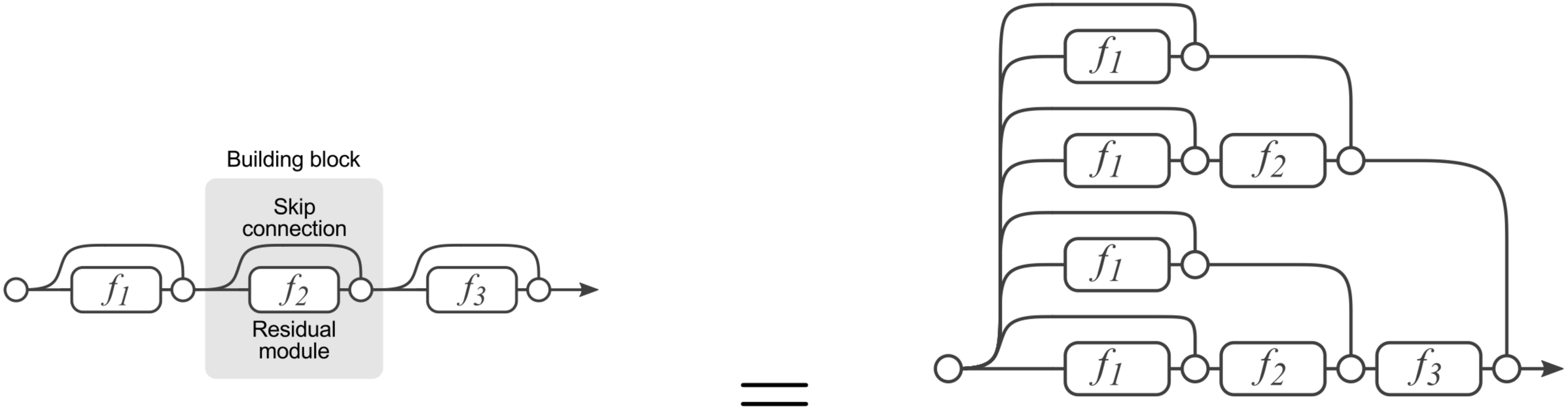


Figure 2. Residual learning: a building block.

ResNet (2016)

- In ResNets, gradients come from shorter paths
  - Similar to ensembling with many shortcus predictors



(a) Conventional 3-block residual network

(b) Unraveled view of (a)

ResNet (2016)

- Also introduces bottleneck blocks
  - Accelerates training
  - Utilizes higher channel dimension, but 3x3 convolution is done in bottlenecks

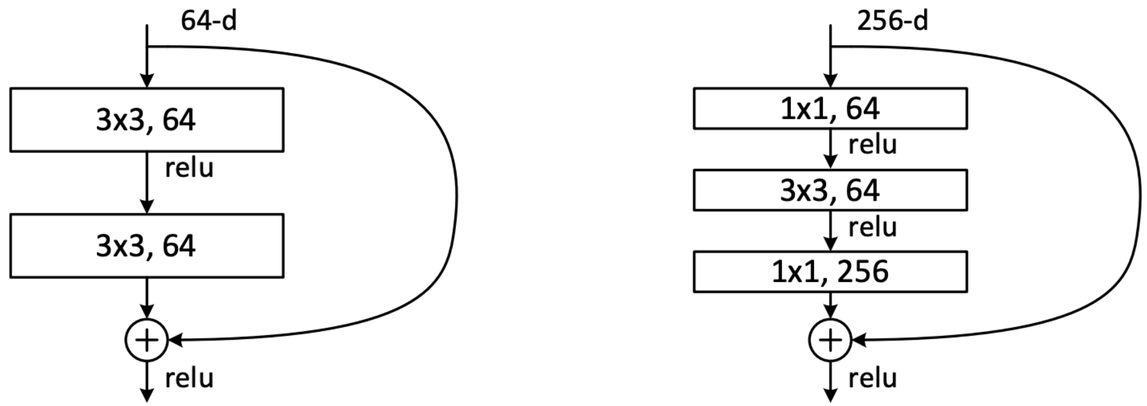
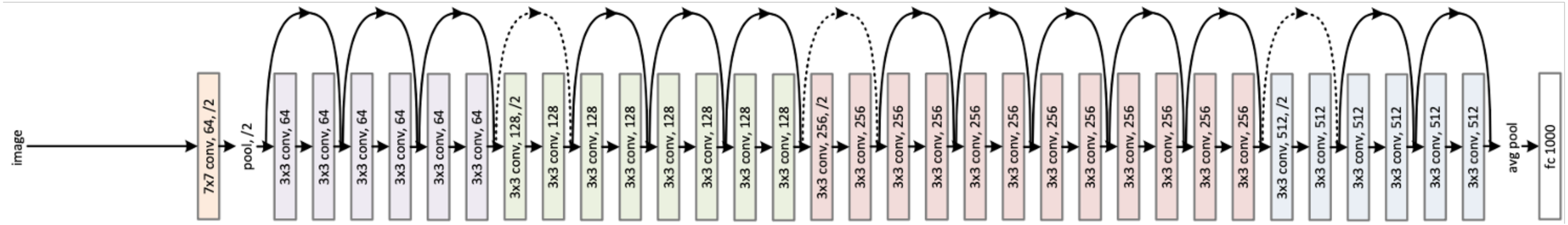


Figure 5. A deeper residual function $\mathcal{F}$ for ImageNet. Left: a building block (on 56×56 feature maps) as in Fig. 3 for ResNet-34. Right: a "bottleneck" building block for ResNet-50/101/152.

ResNet (2016)

- Up to over 150 layers in total
  - Better initialization (He init.)
  - Batch normalization after each convolution
- Dotted line. Doubles #channels and downsample by 2



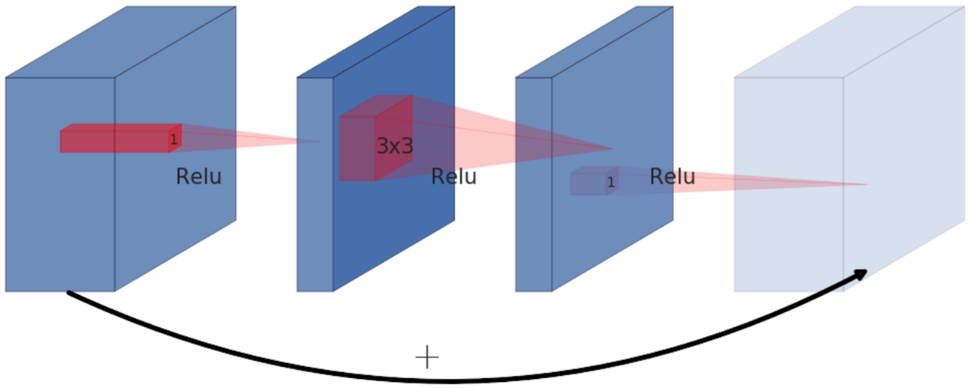
Tiny / Scalable models

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

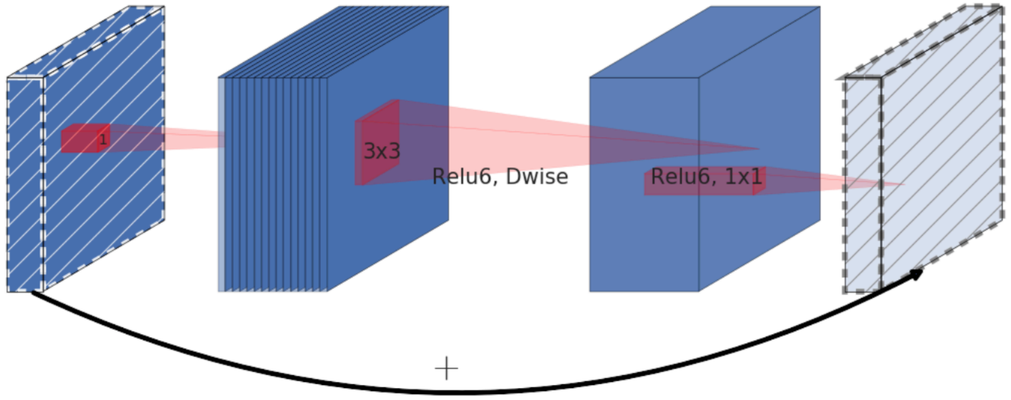
MobileNets (2017~2019)

- **Motivation.** Less number of parameters for inference on mobile devices
- **Innovations.**
  - v1. Depthwise convolution
  - v2. Inverted residuals
  - v3. Architecture search

(a) Residual block



(b) Inverted residual block



Tan and Le, “Rethinking Model Scaling for Convolutional Neural Networks,” ICML 2019

EfficientNets (2019~2021)

- **Question.** If we have more budget, how should we increase #params?
  - Increase depth / width / resolution (i.e., less downsampling)
- **Answer.** All of them jointly, with some scaling factors

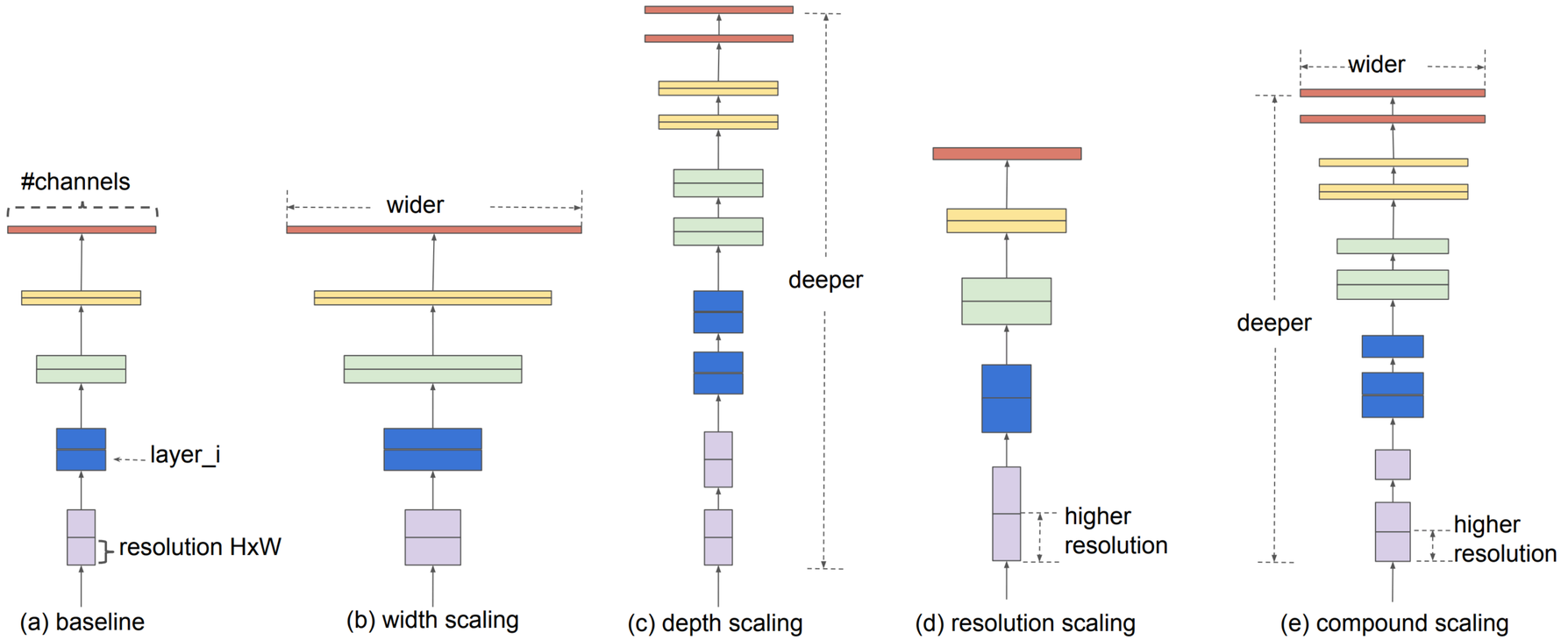
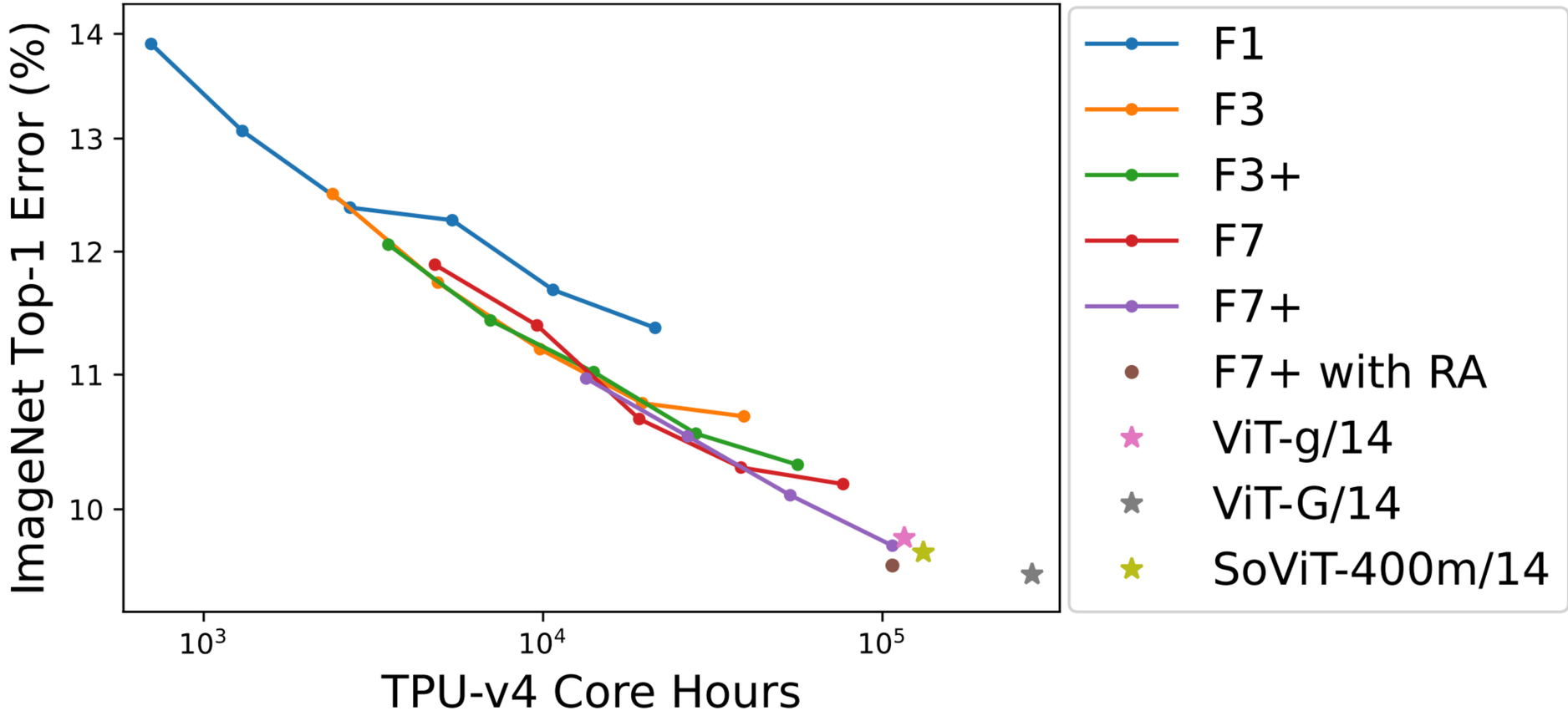


Figure 2. **Model Scaling.** (a) is a baseline network example; (b)-(d) are conventional scaling that only increases one dimension of network width, depth, or resolution. (e) is our proposed compound scaling method that uniformly scales all three dimensions with a fixed ratio.

Brock et al., “High-performance large-scale image recognition without normalization,” ICML 2021

NFNets (2021~2023)

- People observed that removing batch norms is a key to build very big ConvNets
  - Before this, people thought large ConvNets cannot work as good as transformer-based models

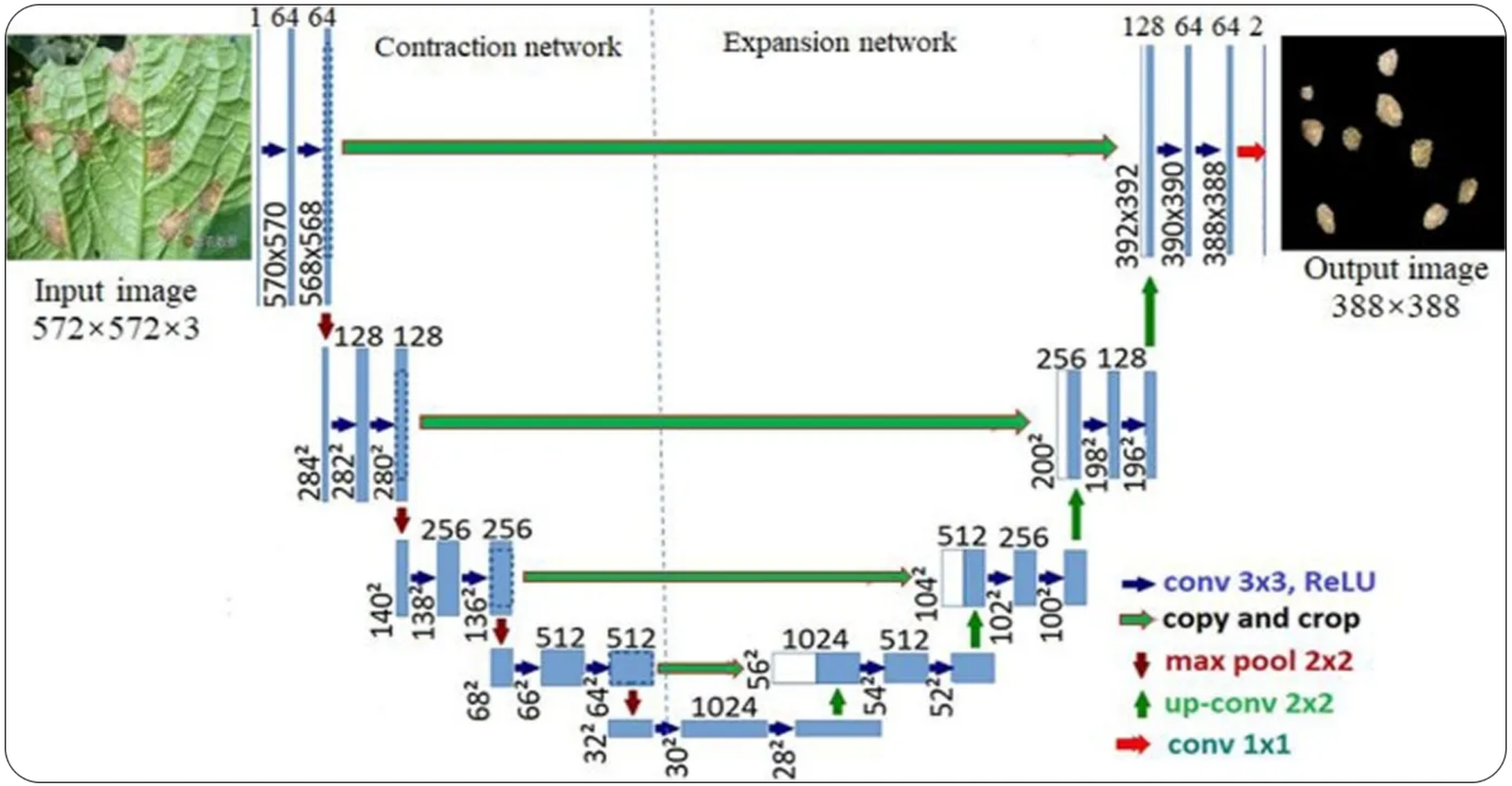


Other models

Ronneberger et al., “U-Net: Convolutional Networks for Biomedical Image Segmentation,” MICCAI 2015

U-Nets (2015)

- One of the most popular models for biomedical applications
  - Image-to-image tasks: Segmentation, denoising, …
  - All-convolutional network, with residuals



</lecture 15>