# Vision: Convolution and CNNs

EECE454 Intro. to Machine Learning Systems

Fall 2024

# Recap

- **So far.** Basics about neural networks & deep learning
  - Multi-layer perceptrons
  - Training neural networks
    - Backprop
    - Optimizers & Initializations ...

# Recap

- **So far.** Basics about neural networks
  - Multi-layer perceptrons
  - Training neural networks
    - Backprop
    - Optimizers & Initializations ...
- **Today.** Topics in computer vision
  - Convolution
  - Advanced CNNs

# Convolution — overview

# Recap: MLPs

- **MLP.** Takes a simple form:

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

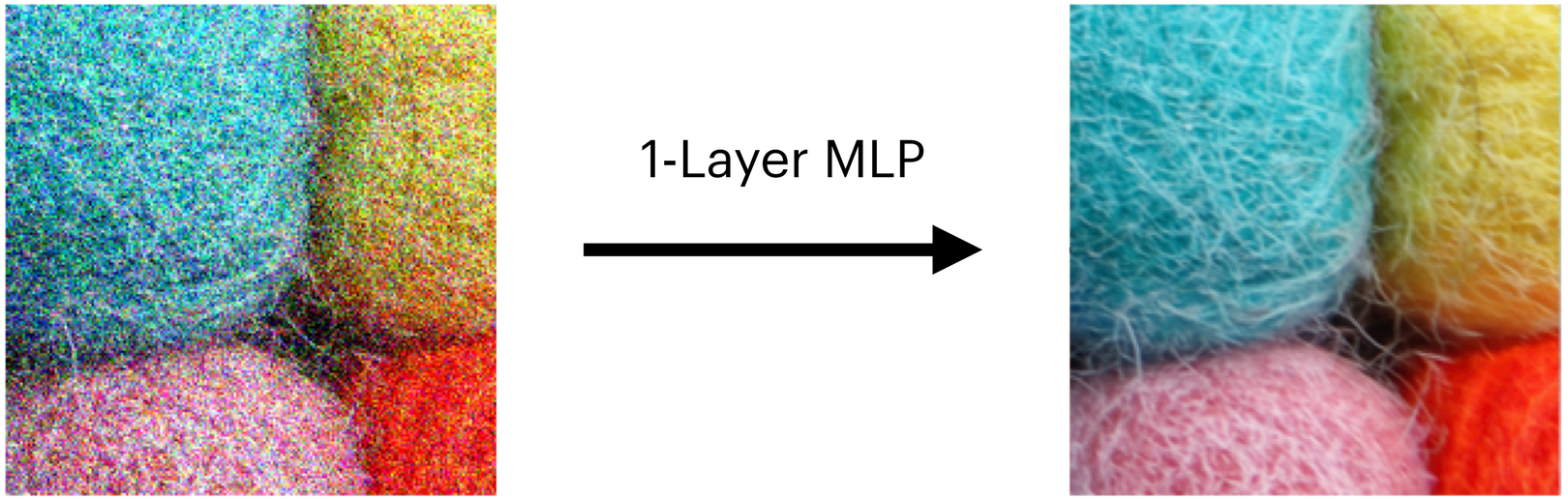
  - Alternatingly applies two operations

    - **Linear operation:** $\mathbf{x} \mapsto \mathbf{W}\mathbf{x} + \mathbf{b}$

    - **Nonlinear activation** $\mathbf{x} \mapsto \sigma(\mathbf{x})$

# Recap: MLPs

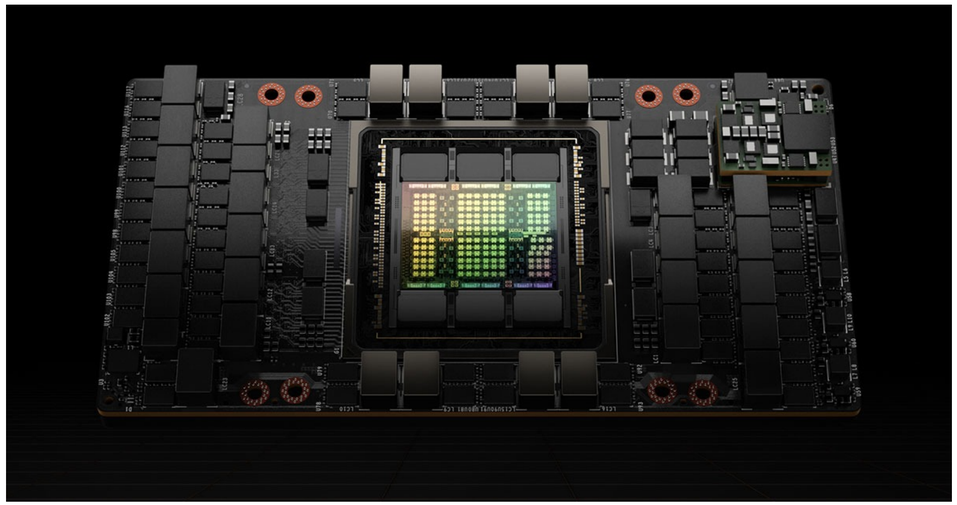
- **Problem.** Matmuls on visual signals require too many parameters & computations
  - Example. We train a model—e.g., an image denoiser—that processes a 1080p image (1920 x 1080 pixels) to another 1080p image.
    - If it is a linear model (i.e., 1-layer MLP), we need
      - 38.7 trillion parameters (≈ 155TB in FP32)
      - 77.4 trillion inference FLOPs



# Recap: MLPs

- **Problem.** Matmuls on visual signals require too many parameters & computations
  - Example. We train a model—e.g., an image denoiser—that processes a 1080p image (1920 x 1080 pixels) to another 1080p image.
    - If it is a linear model (i.e., 1-layer MLP), we need
      - 38.7 trillion parameters (≈ 155TB in FP32)
      - 77.4 trillion inference FLOPs
  - Example. If we have
    - 10 layers
    - 1 million training images
    - train for for 100 epochs

    Then we need $1.5 \times 10^{23}$ FLOPs



**NVIDIA H100**
67 TFLOPS

≈ 70 years
(with infinite bandwidth)

# Convolution

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

# 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 → $(f * g)(t)$

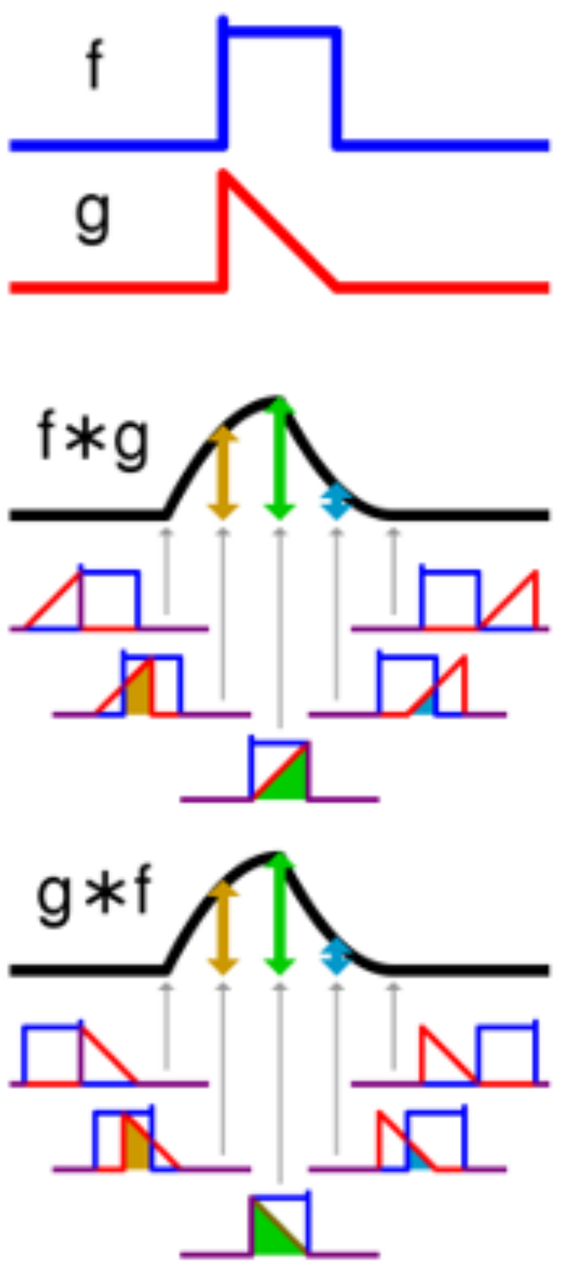
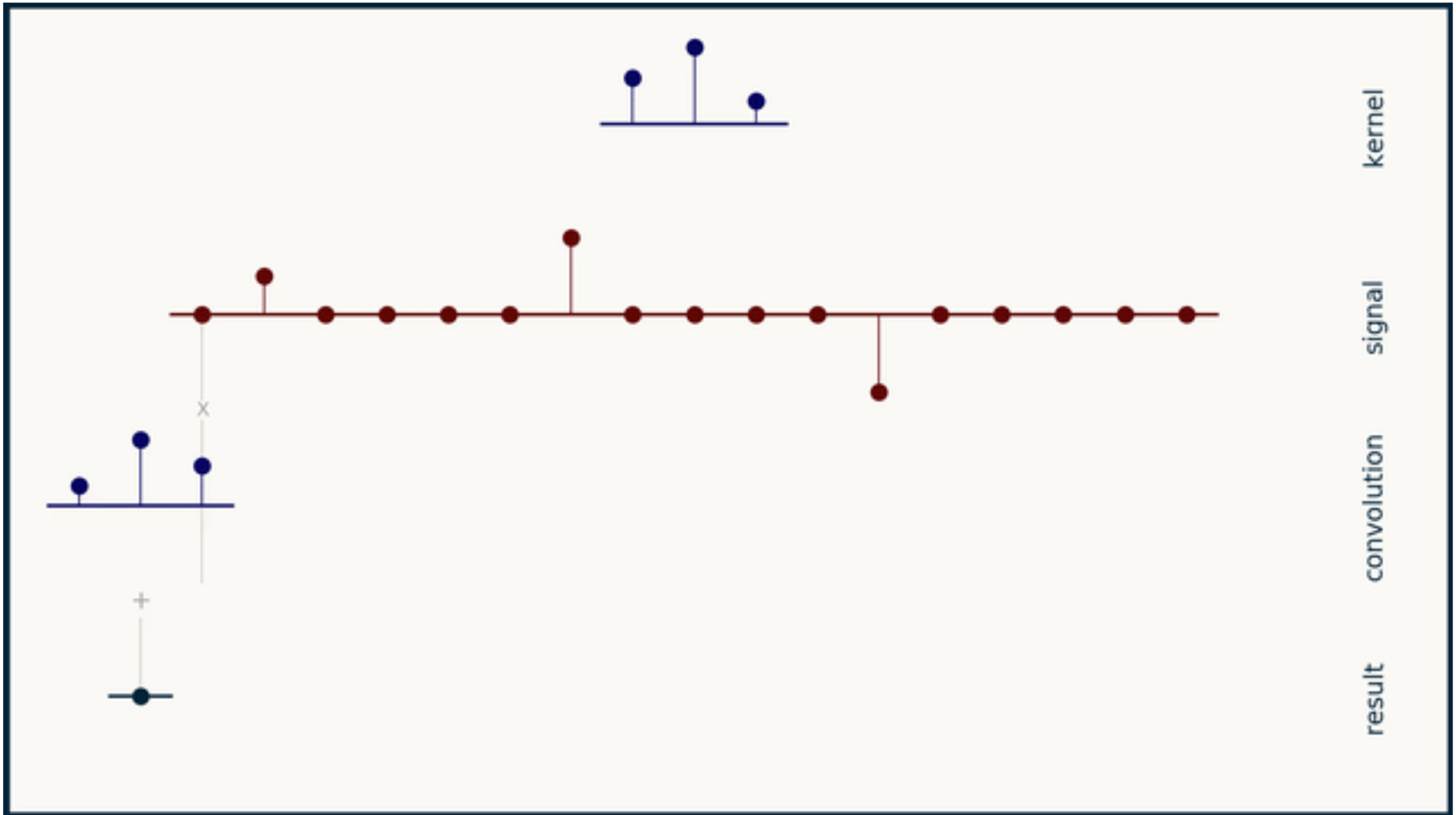
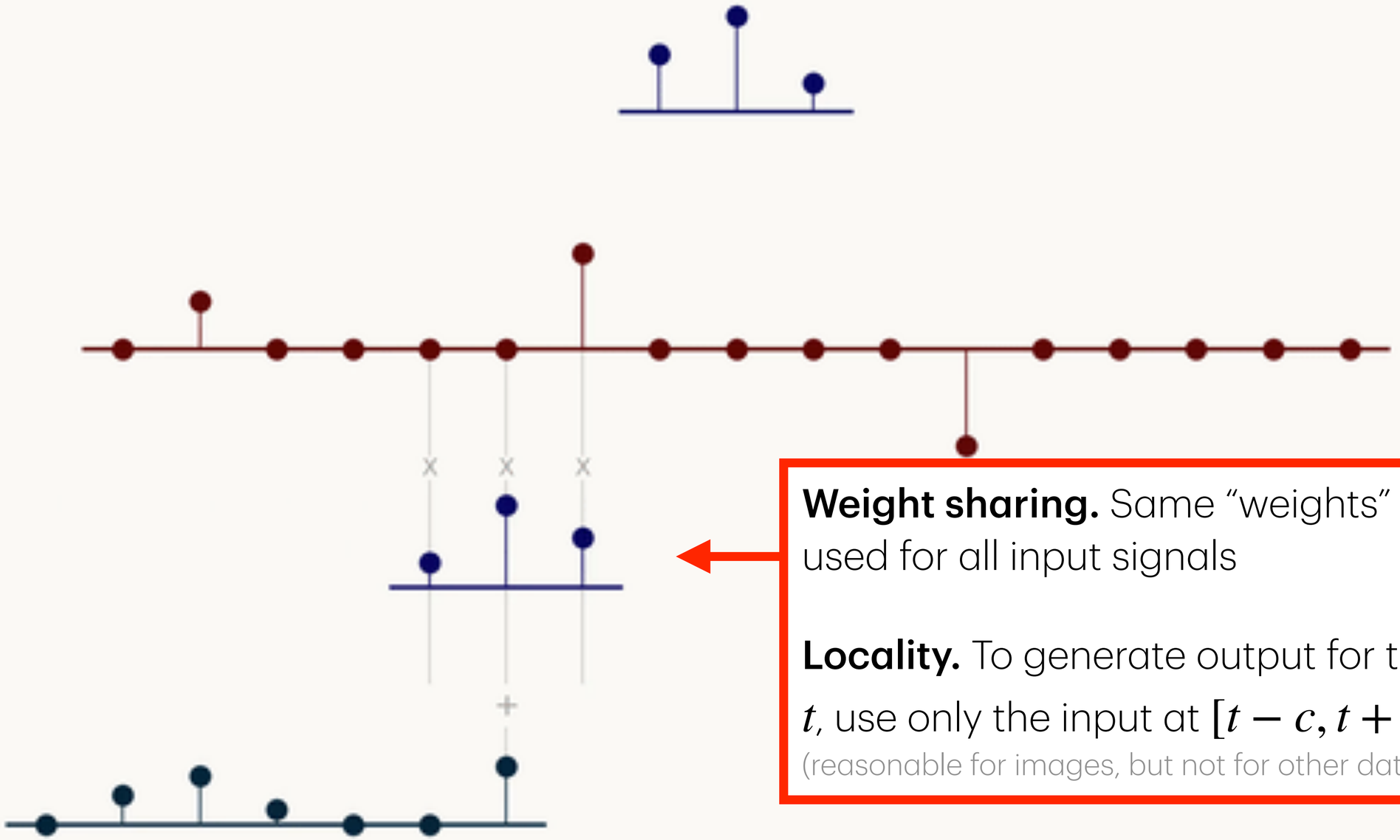


Image source: 1D convolution for neural networks, part 1: Sliding dot product, Brandon Rohrer






**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”

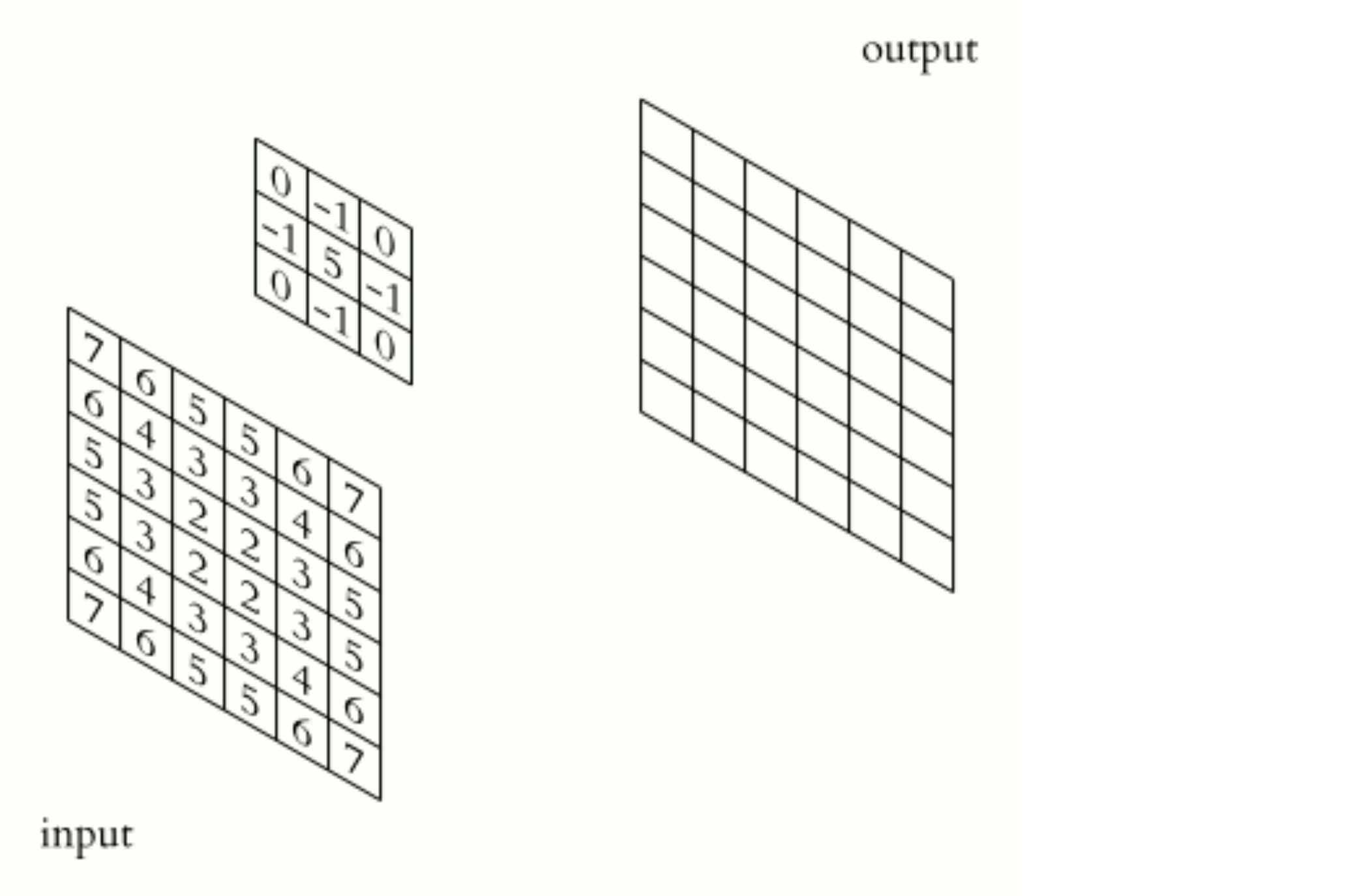
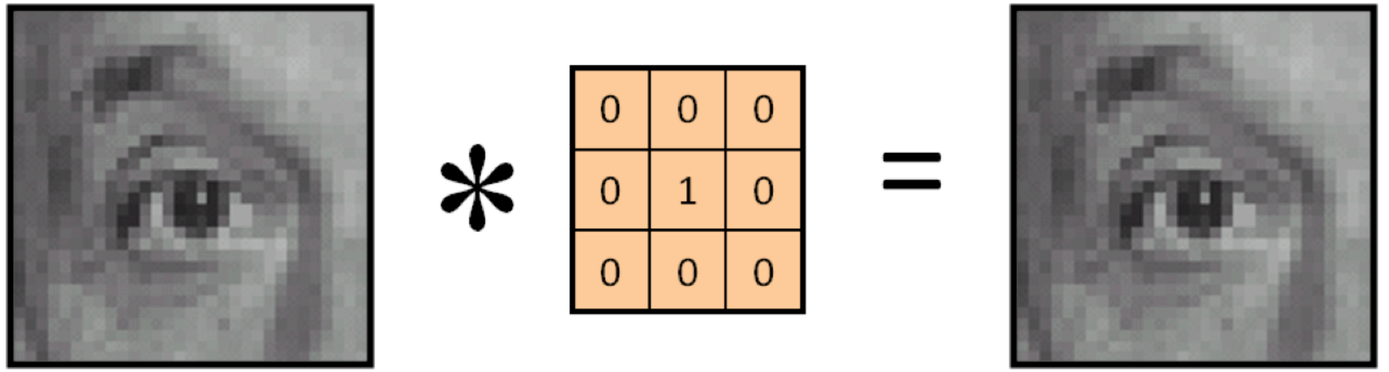


Image source: https://www.cs.cornell.edu/courses/cs1114/2013sp/sections/S06_convolution.pdf

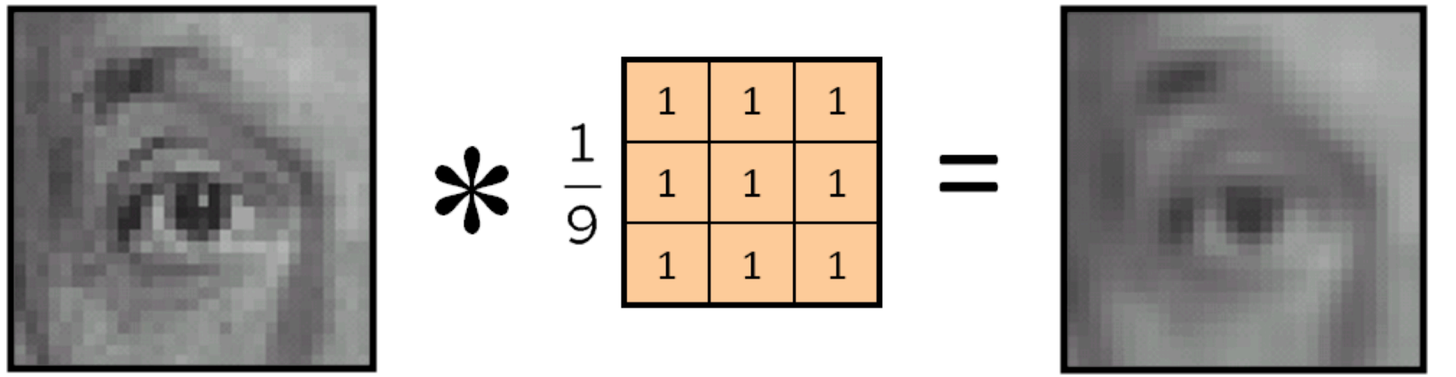
# Convolution

- **Classic.** Different handcrafted “filters” can be used for different purposes.




Original

Identical image



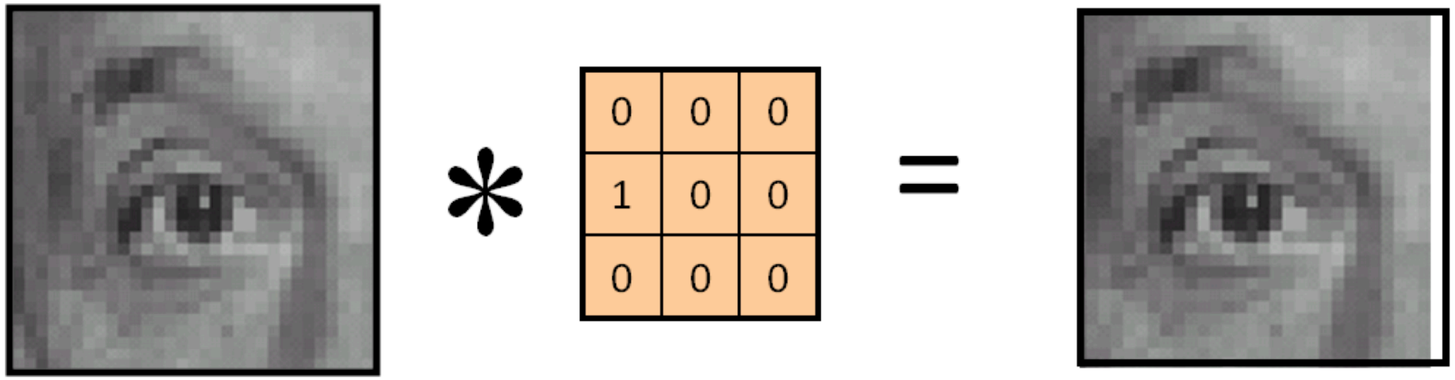

Original

Blur (with a mean filter)

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

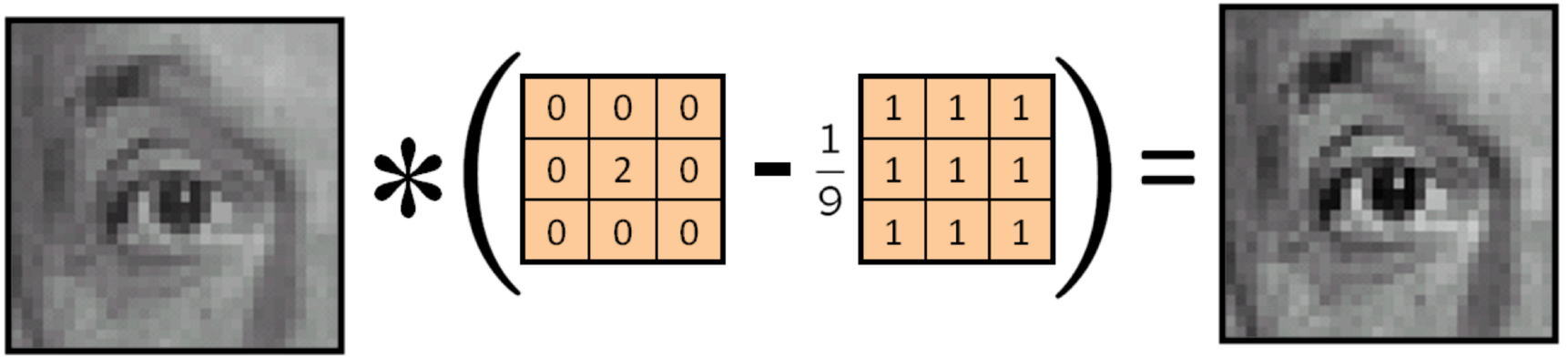
# Convolution

- **Classic.** Different handcrafted “filters” can be used for different purposes.




Original

Shifted left
By 1 pixel




Original

**Sharpening filter**
(accentuates edges)

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

# Convolution

- **ML.** Filters are learned from the data
  - Replaces the linear layers
  - Apply multiple parallel filters

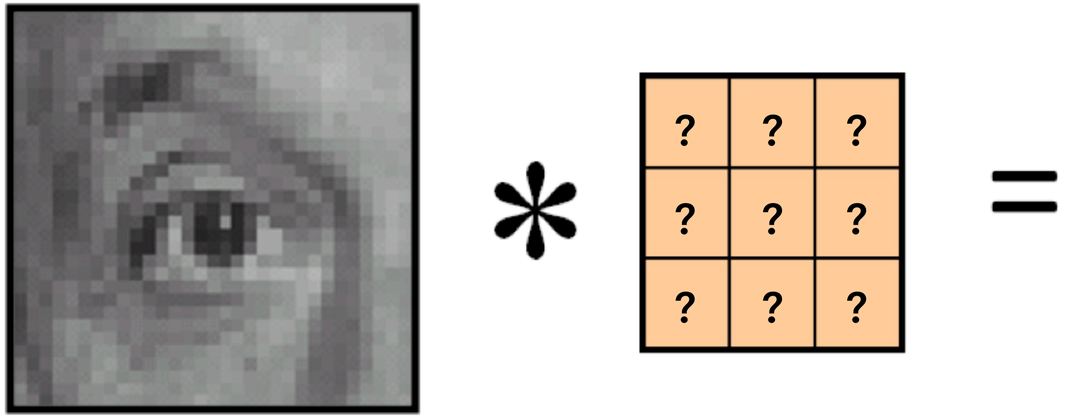


Image source: https://www.cs.cornell.edu/courses/cs1114/2013sp/sections/S06_convolution.pdf

# Convolution

- **ML.** Filters are learned from the data
  - Replaces the linear layers
  - Apply multiple parallel filters



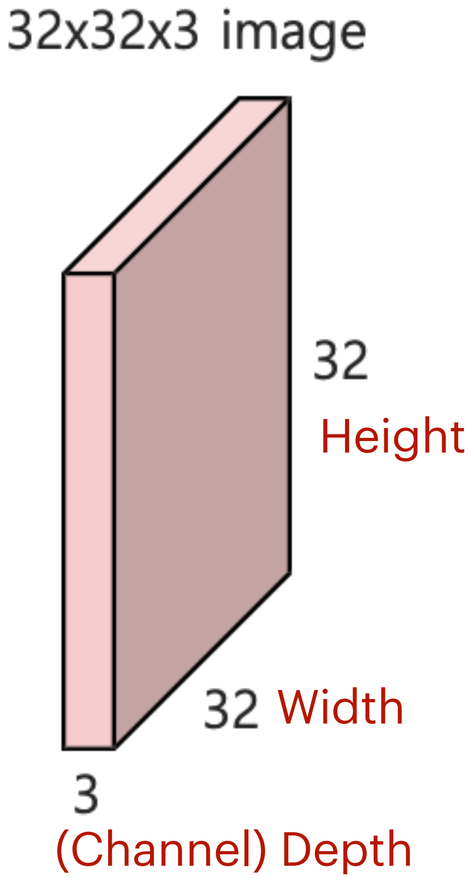

- **Property.** The learned operation is always translation-equivariant,
  i.e., applies the same operation to patches at different locations.

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

# Building a convolutional net

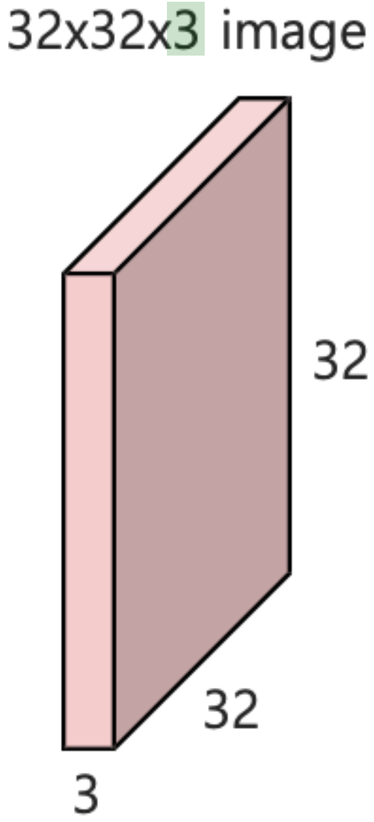
# Convolutional layer

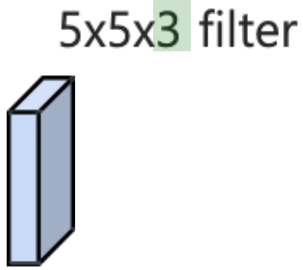
- Begin with a 32x32 image with 3 channels (RGB)



# Convolutional layer

- Convolve with a conv filter — dot product with a sliding “receptive field”




**Classic.** Filters have full channel depth

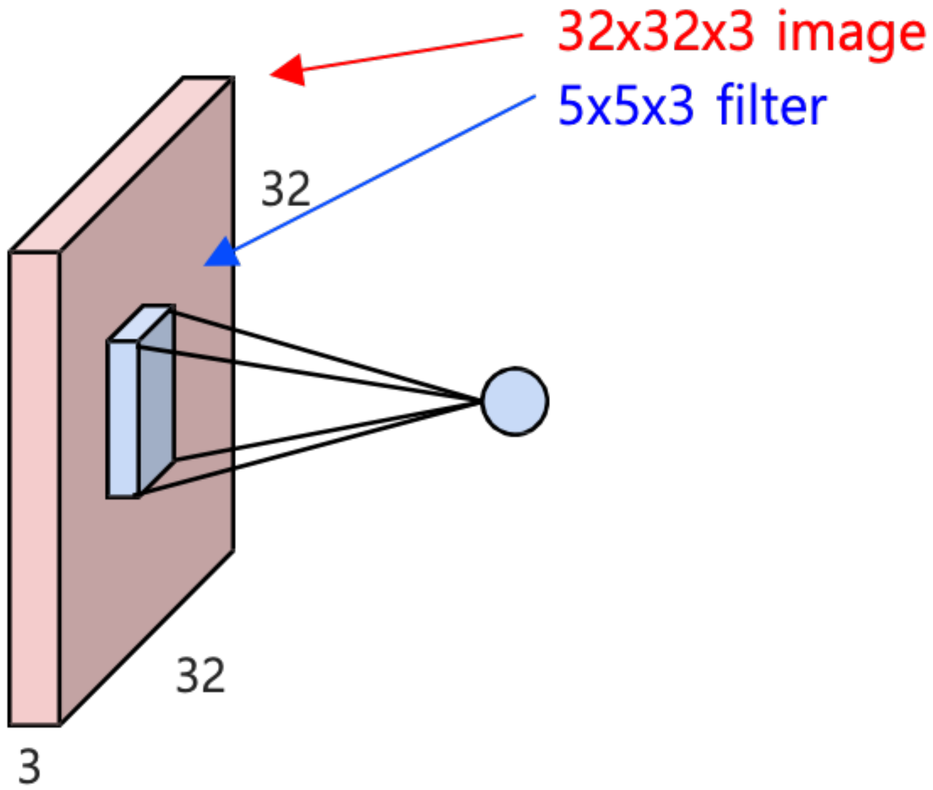
- i.e., uses all input channels

**Modern.** Apply depth-1 convolution for each input channel, for efficiency

- called “depthwise convolution”

# Convolutional layer

- Convolve with a conv filter — dot product with a sliding "receptive field"
  - Convolving generates a single entry of the layer output



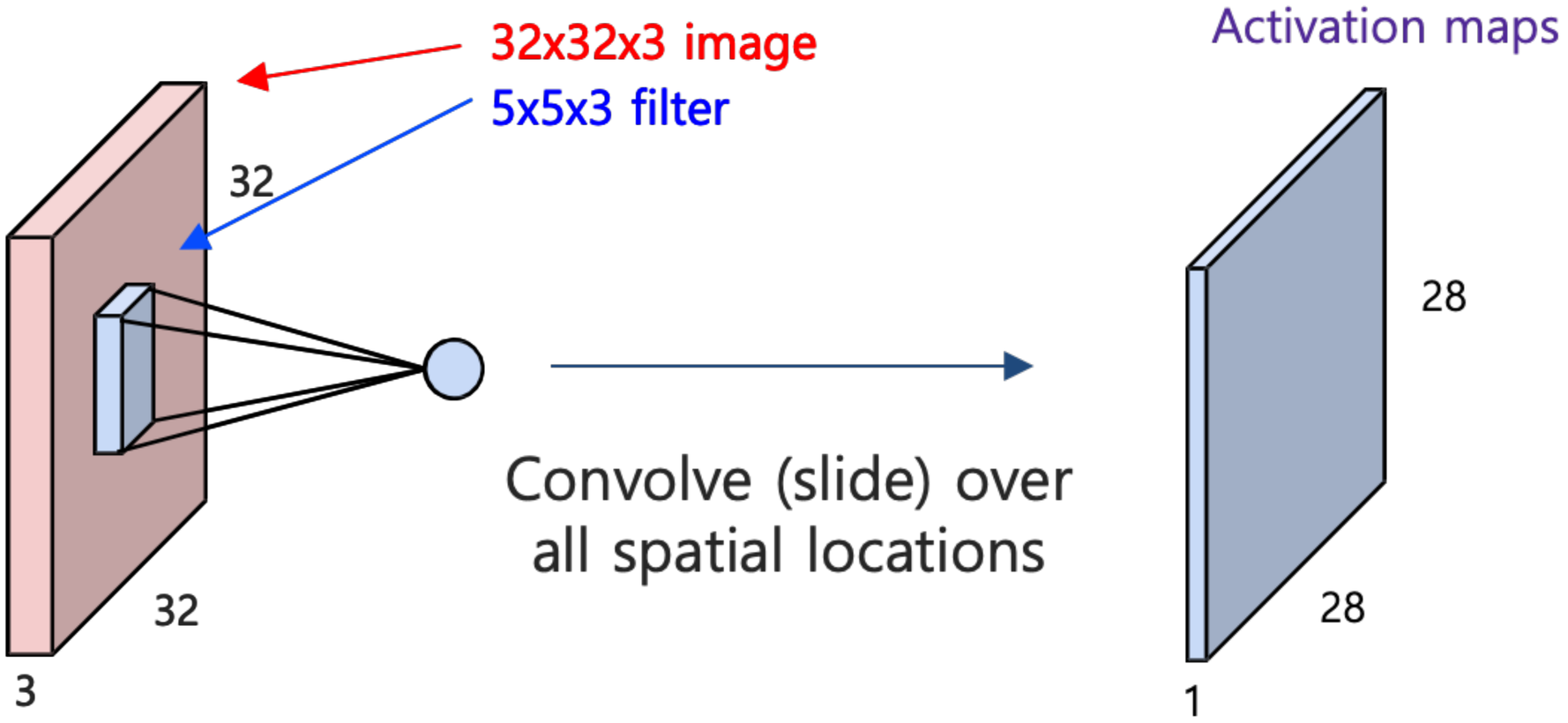

**Compute.** Dot product of two tensors with

$5 \times 5 \times 3 = 75$ dimensions

(+ bias addition)

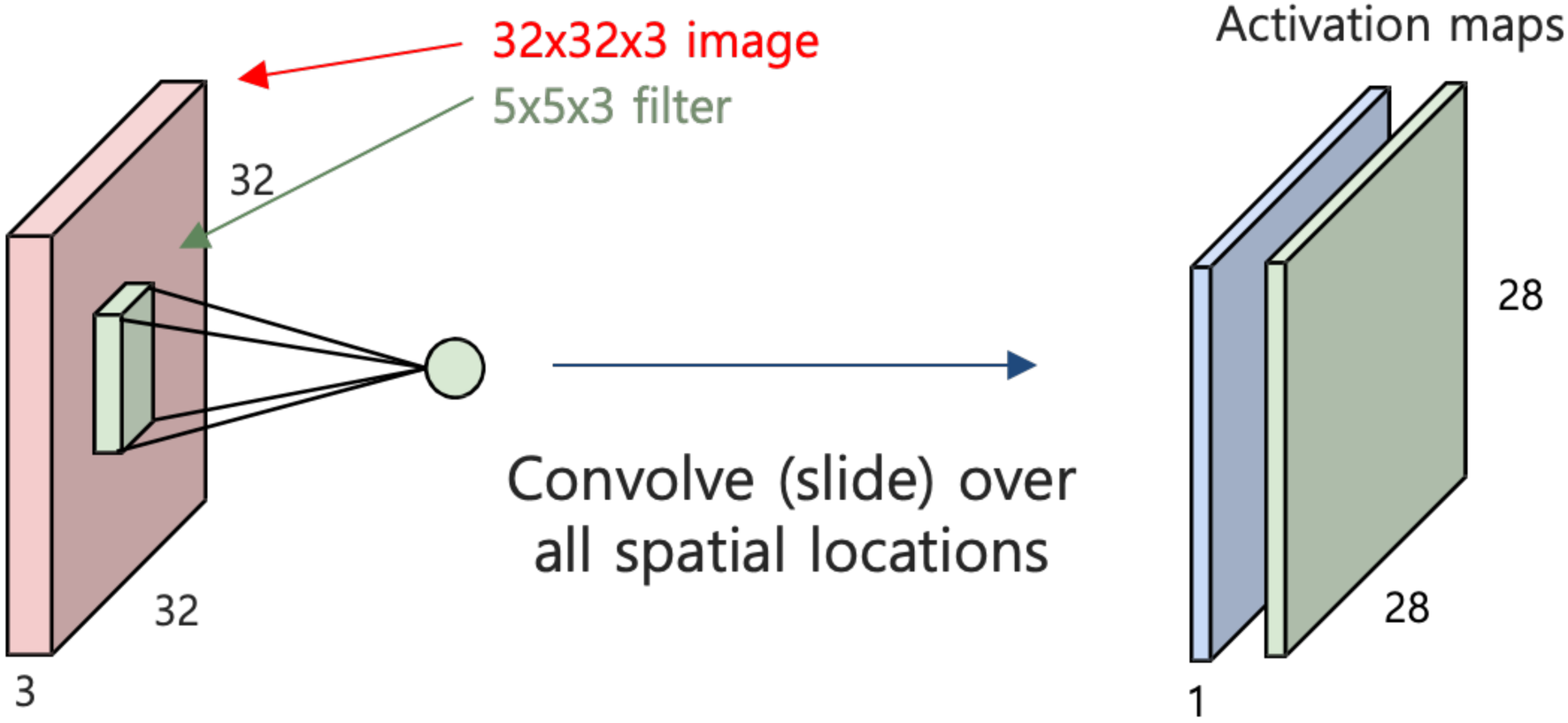
# Convolutional layer

- Convolve with a conv filter — dot product with a sliding "receptive field"
  - Convolving generates a single entry of the layer output



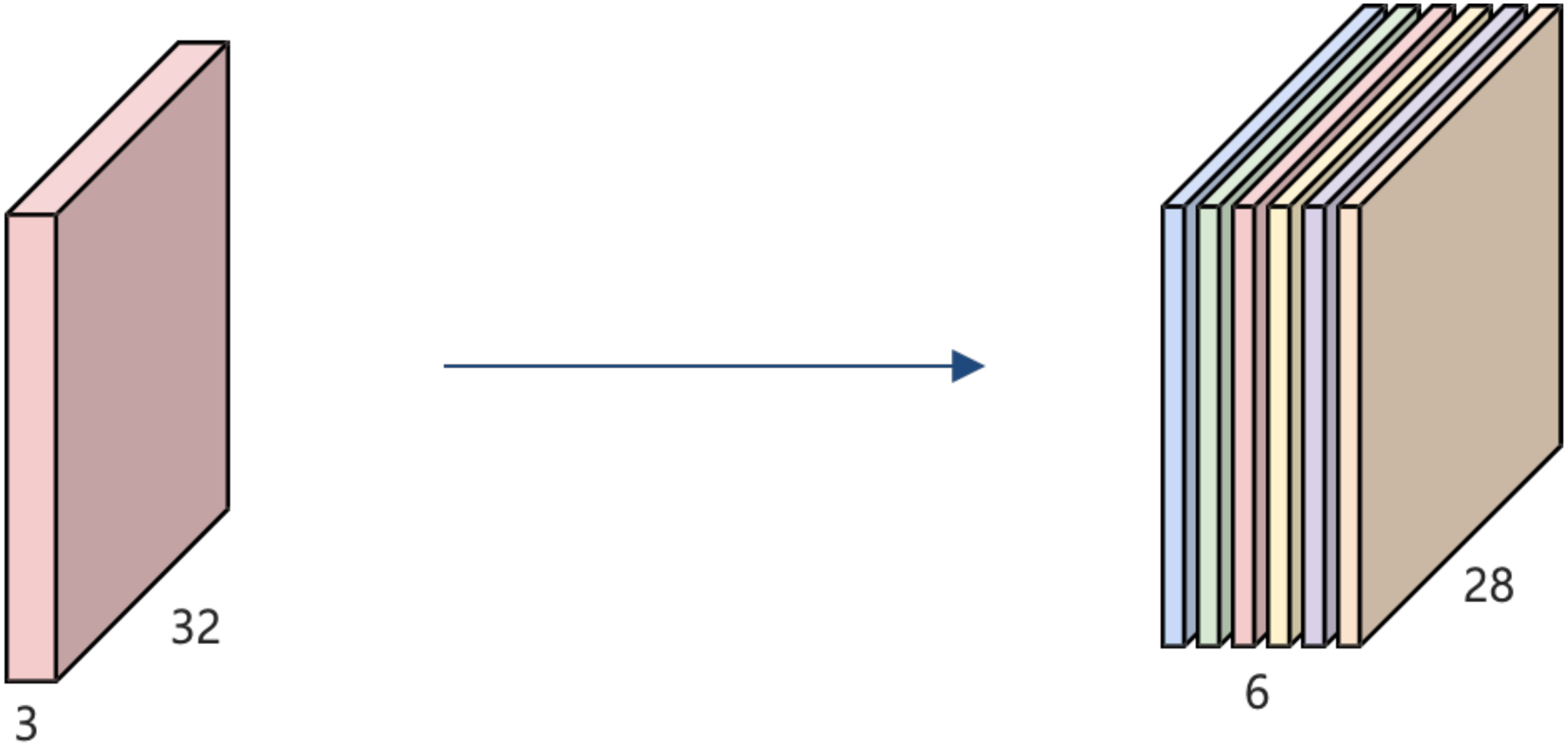
# Convolutional layer

- Convolve with a conv filter — dot product with a sliding "receptive field"
  - Convolving generates a single entry of the layer output
  - Do the same for the second filter



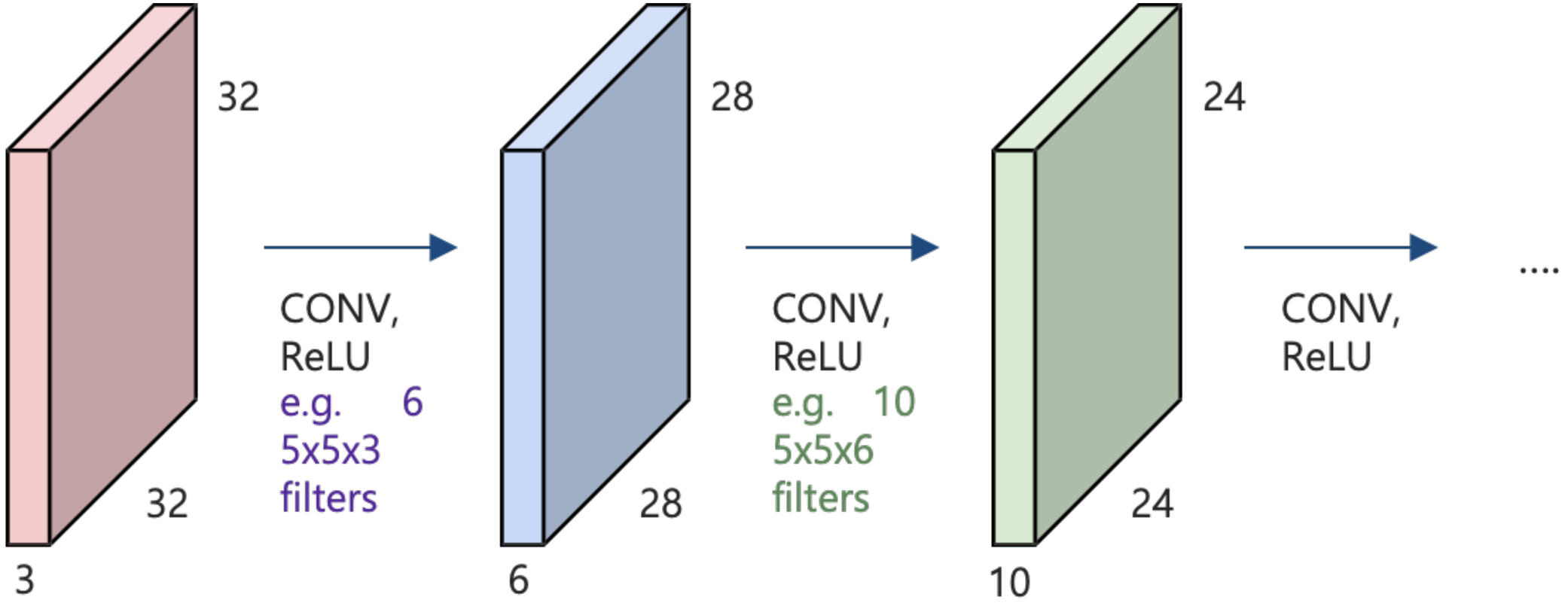
# Convolutional layer

- Generate the pre-activation with **multiple depth** (called channels)



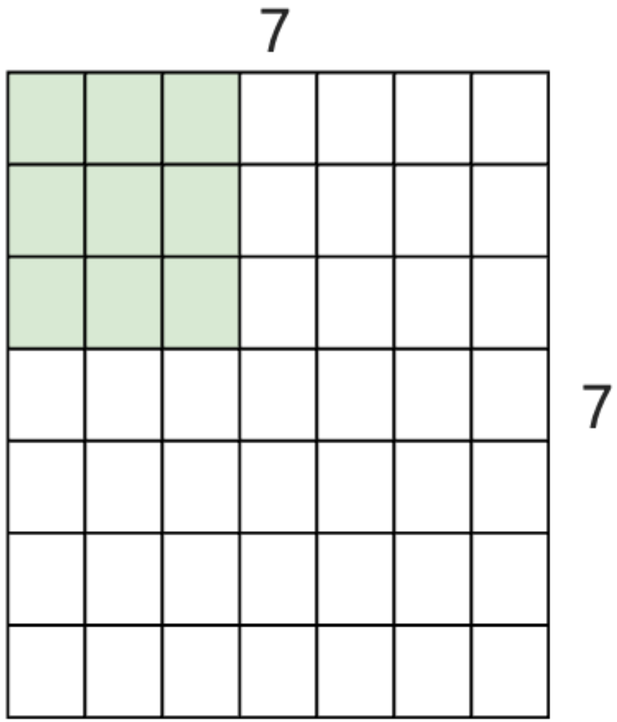
# Convolutional layer

- Stack the layers, with activation functions in between.



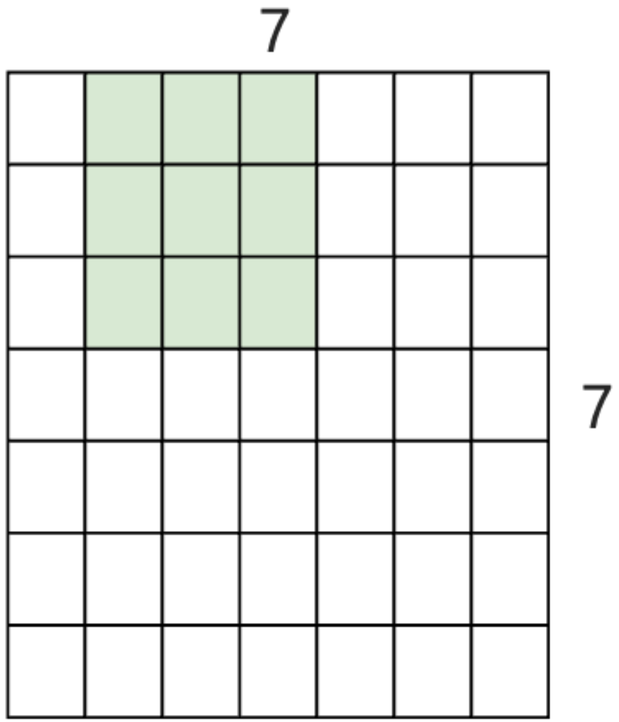
# Spatial Dimension: Stride

- Consider a 7x7 image, with 3x3 filters



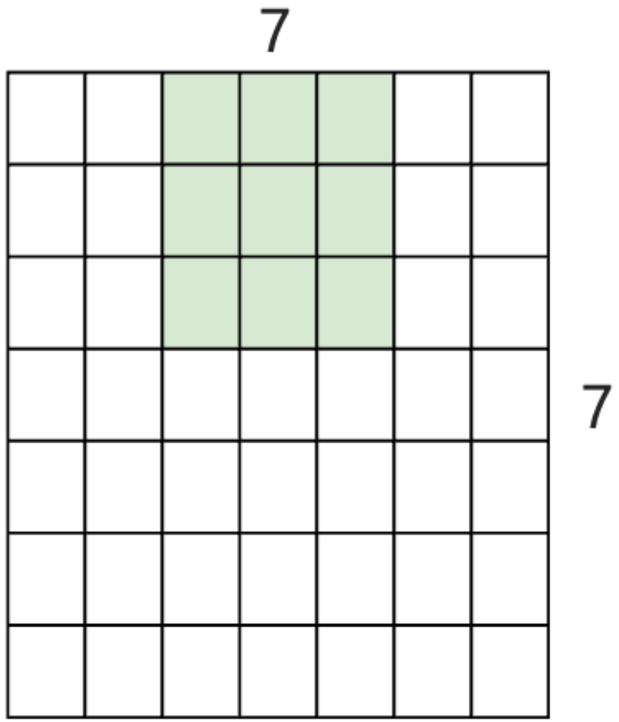
# Spatial Dimension: Stride

- Consider a 7x7 image, with 3x3 filters



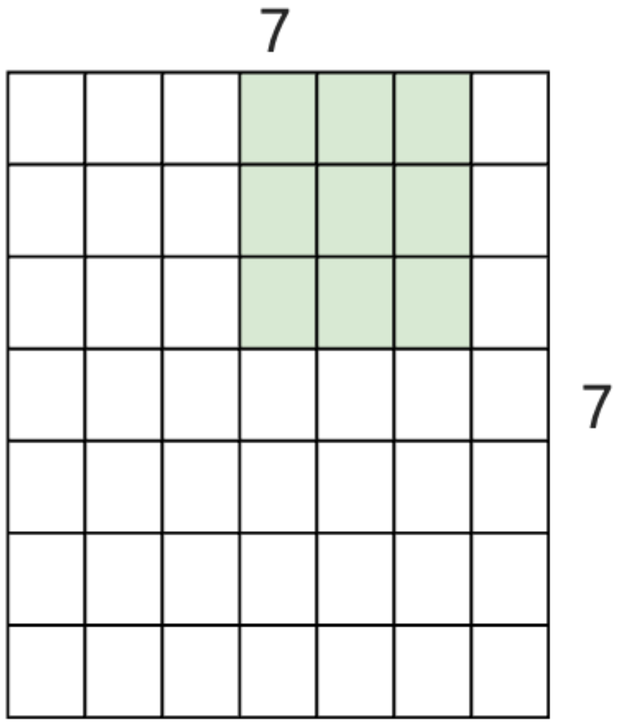
# Spatial Dimension: Stride

- Consider a 7x7 image, with 3x3 filters



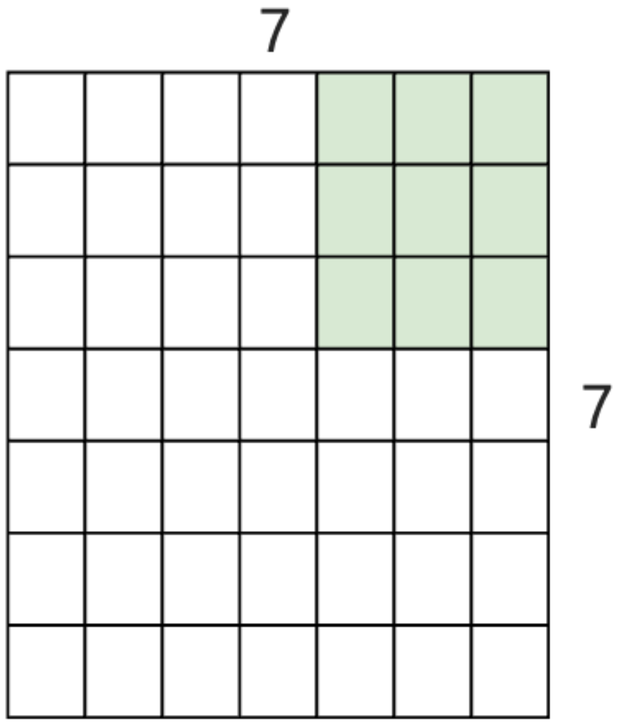
# Spatial Dimension: Stride

- Consider a 7x7 image, with 3x3 filters



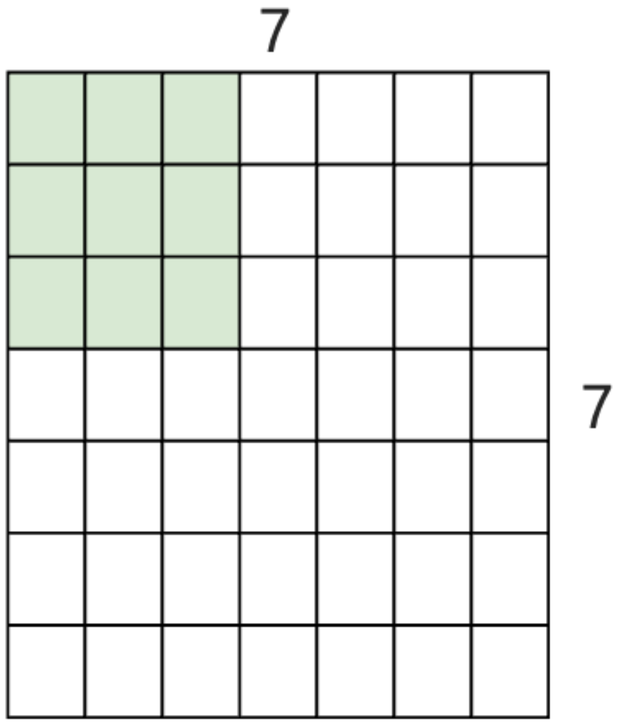
# Spatial Dimension: Stride

- Consider a 7x7 image, with 3x3 filters $\Rightarrow 5 \times 5$ output



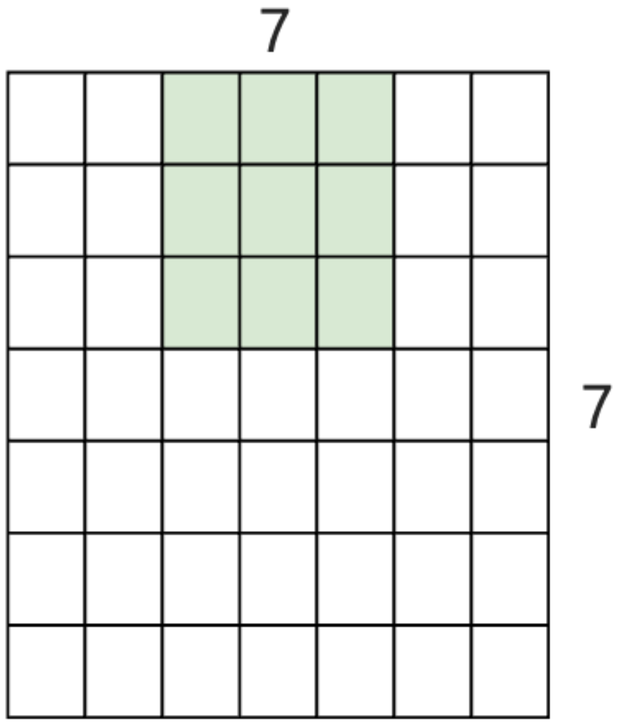
# Spatial Dimension: Stride

- Consider a 7x7 image, with 3x3 filters ⇒ 5x5 output
- It is common to apply strides — with stride 2



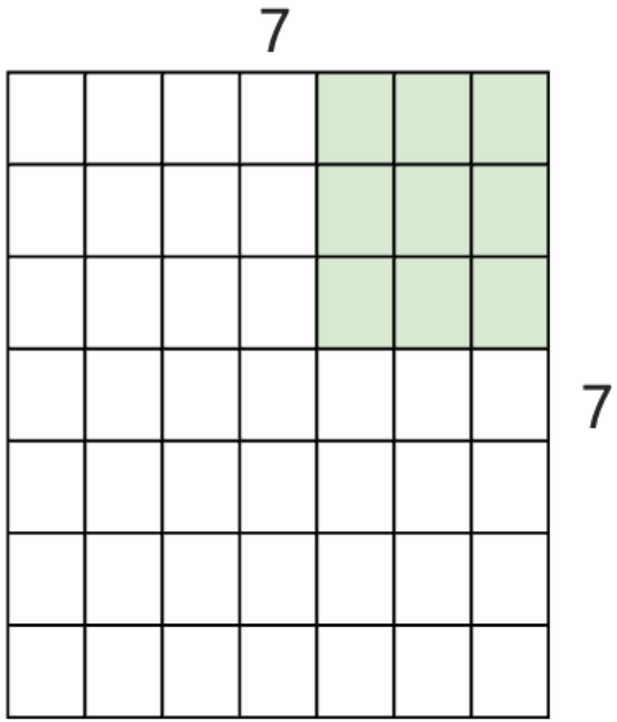
# Spatial Dimension: Stride

- Consider a 7x7 image, with 3x3 filters ⇒ 5x5 output
- It is common to apply strides — with stride 2



# Spatial Dimension: Stride

- Consider a 7x7 image, with 3x3 filters ⇒ 5x5 output
- It is common to apply strides — with stride 2 ⇒ 3x3 output




**Output Size.** (Image length - Filter length) / 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, and thus cannot be used.

# Spatial Dimension: Padding

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

| | | | | | | |
|---|---|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 0 | 60 | 113 | 56 | 139 | 85 | 0 |
| 0 | 73 | 121 | 54 | 84 | 128 | 0 |
| 0 | 131 | 99 | 70 | 129 | 127 | 0 |
| 0 | 80 | 57 | 115 | 69 | 134 | 0 |
| 0 | 104 | 126 | 123 | 95 | 130 | 0 |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 |

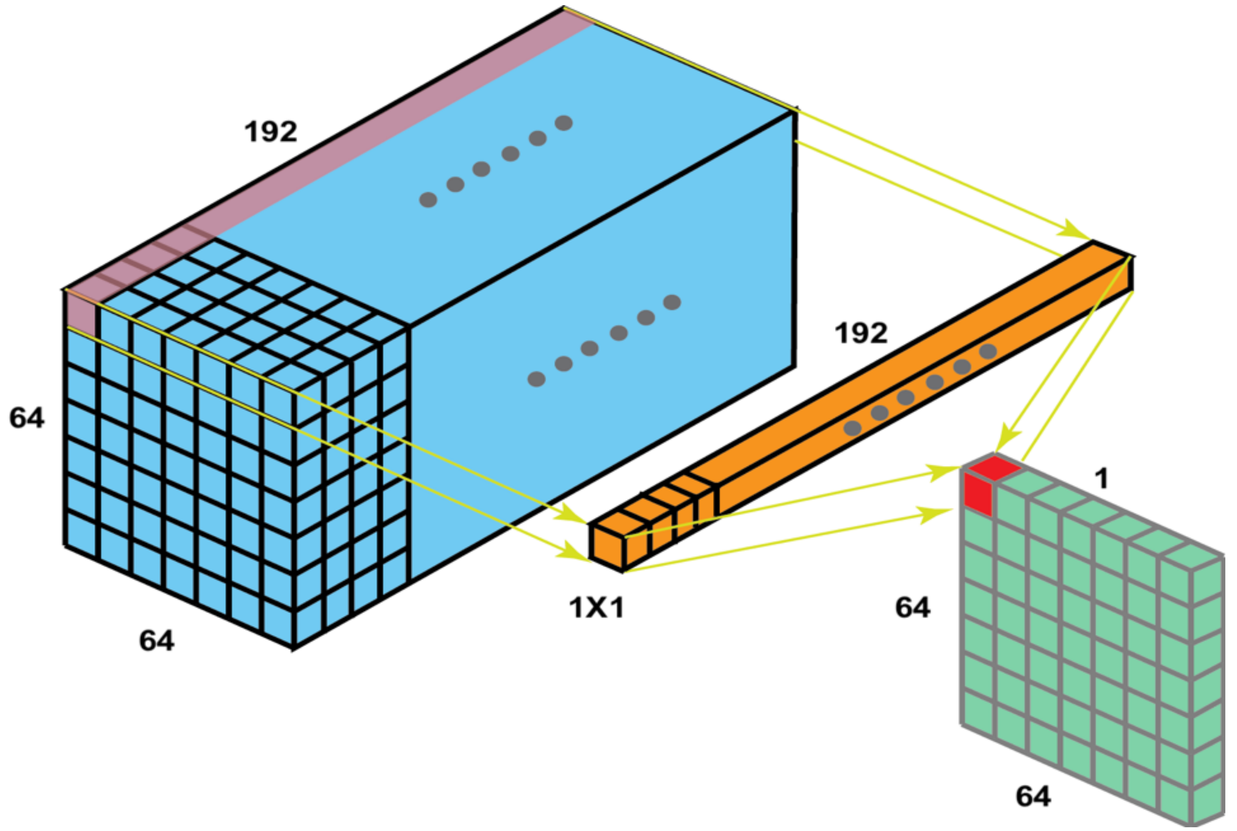
Kernel

| | | |
|---|---|---|
| 0 | -1 | 0 |
| -1 | 5 | -1 |
| 0 | -1 | 0 |

| | | | | |
|---|---|---|---|---|
| 114 | | | | |
| | | | | |
| | | | | |
| | | | | |
| | | | | |

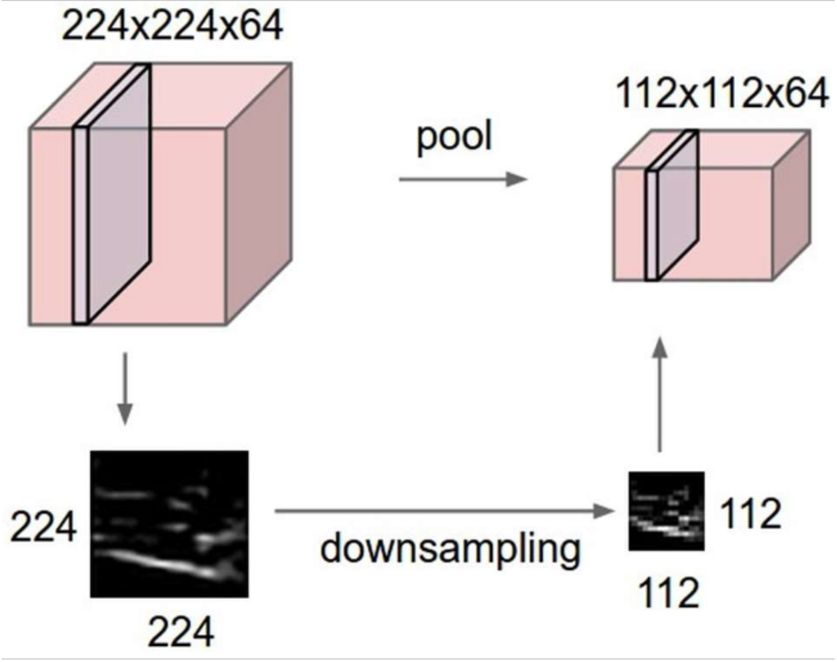
# Spatial Dimension: 1x1

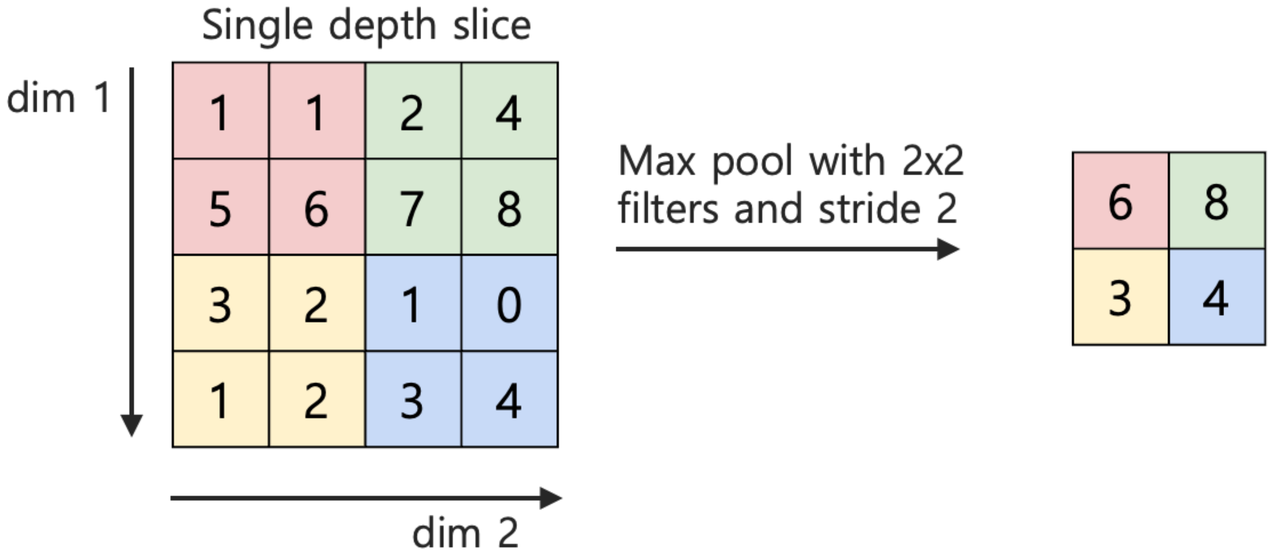
- **1x1 Convolution.** Some layers, using only 1x1 convolutions
  - Increase or decrease the number of channels via linear combination
  - Often use together with depthwise convolution.



# Pooling layer

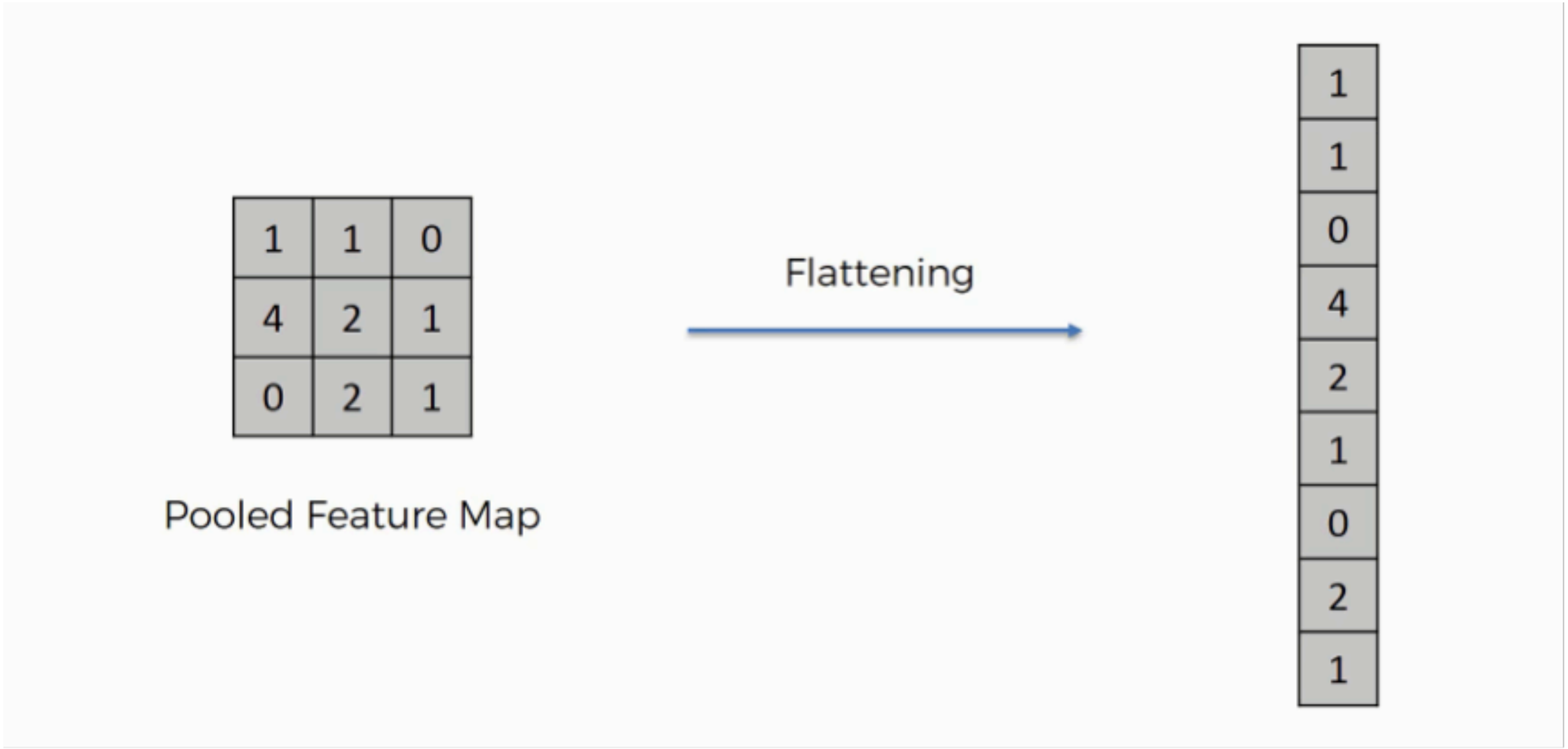
- **Pooling.** Reduces the spatial dimension by taking max/mean/else of pixels
  - Gets smaller resolution without losing information
    (e.g., the activation represents a specific feature)



# Final layer — Fully-connected

- In the final layer, we 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

- A convolutional layer can be applied on images of any size
  - However, a FC layer cannot.
  - For cases like denoising or segmentation (no FC layer), a model trained on 178x178 image can be used on 256x256 images.

# Popular network architectures

# Architectures

- Now let’s take a look at some popular architectures
  - Basic models: LeNet / AlexNet
  - Deeper models: VGG / ResNet
  - Tiny models: MobileNets / MCUNets
  - Efficiently scalable models: EfficientNet / NFNets

# Basic models

# Basic models

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

# LeNet-5 (1998)

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

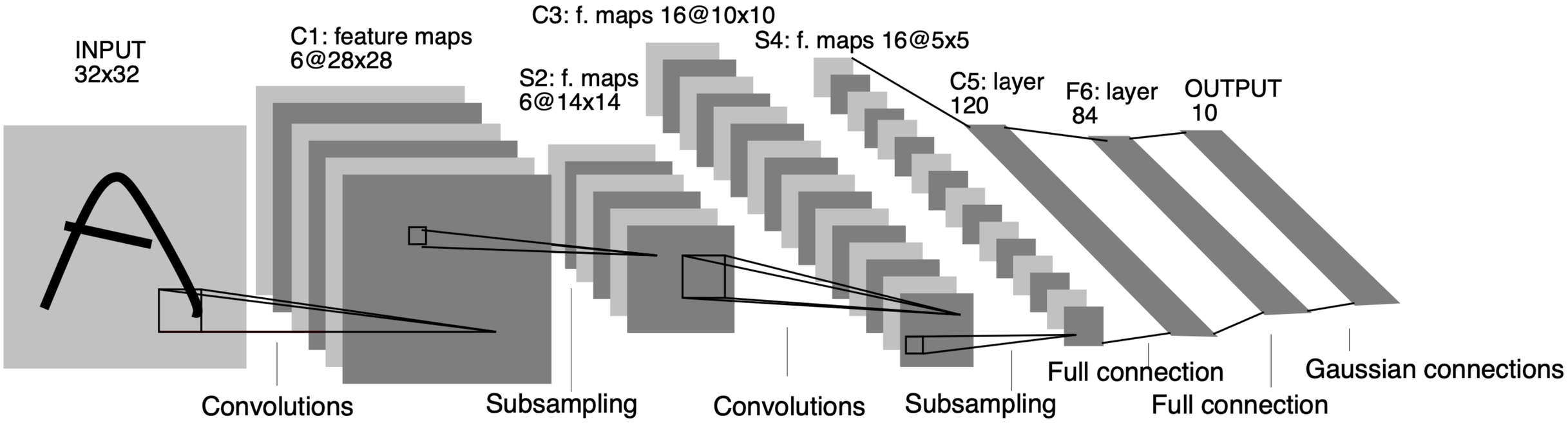


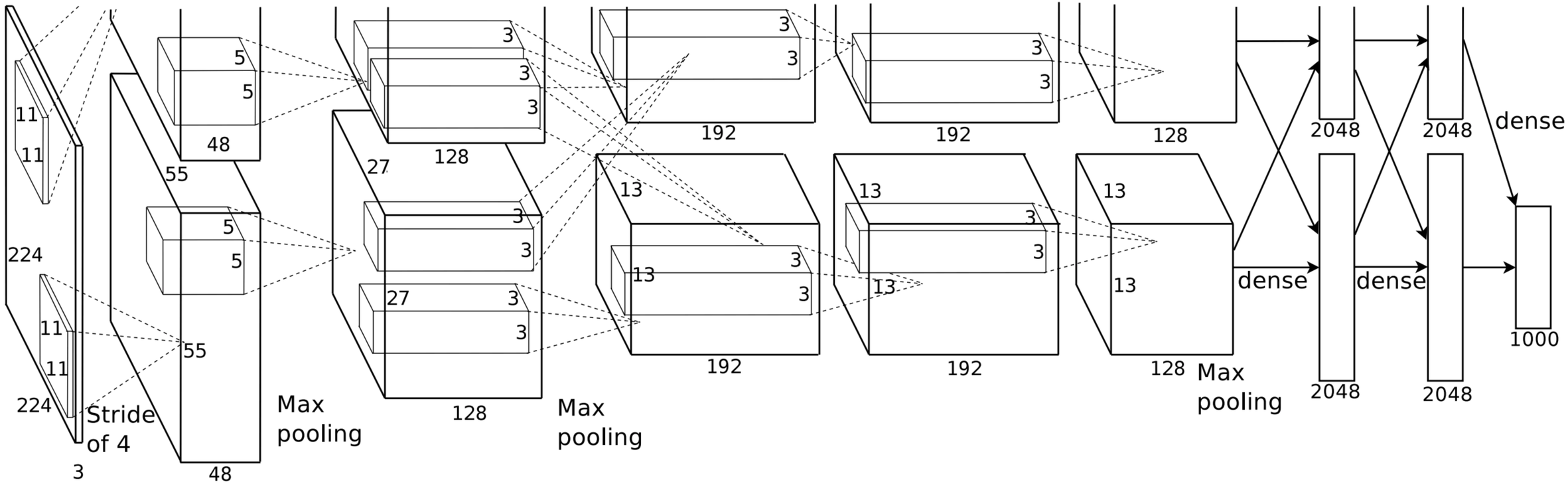

Fig. 2. Architecture of LeNet-5, a Convolutional Neural Network, here for digits recognition. Each plane is a feature map, i.e. a set of units whose weights are constrained to be identical.


4627
4 6 2 7

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

# AlexNet (2012)

- Bigger and deeper version of LeNet — won 2012 ImageNet challenge
  - 7 hidden layers, 650k neurons, 60 million parameters, plus dropout

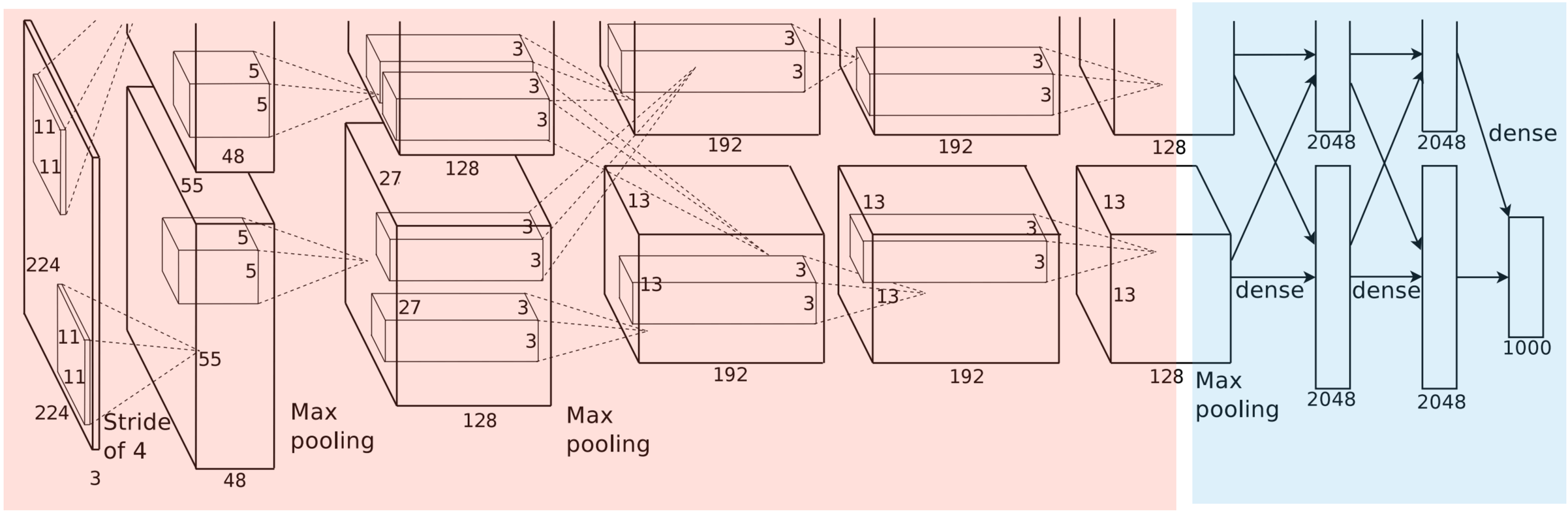


Krizhevsky et al., “ImageNet Classification with Deep Convolutional Neural Networks,” NeurIPS 2012

# AlexNet (2012)

- Bigger and deeper version of LeNet — won 2012 ImageNet challenge
  - 7 hidden layers, 650k neurons, 60 million parameters, plus dropout

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



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

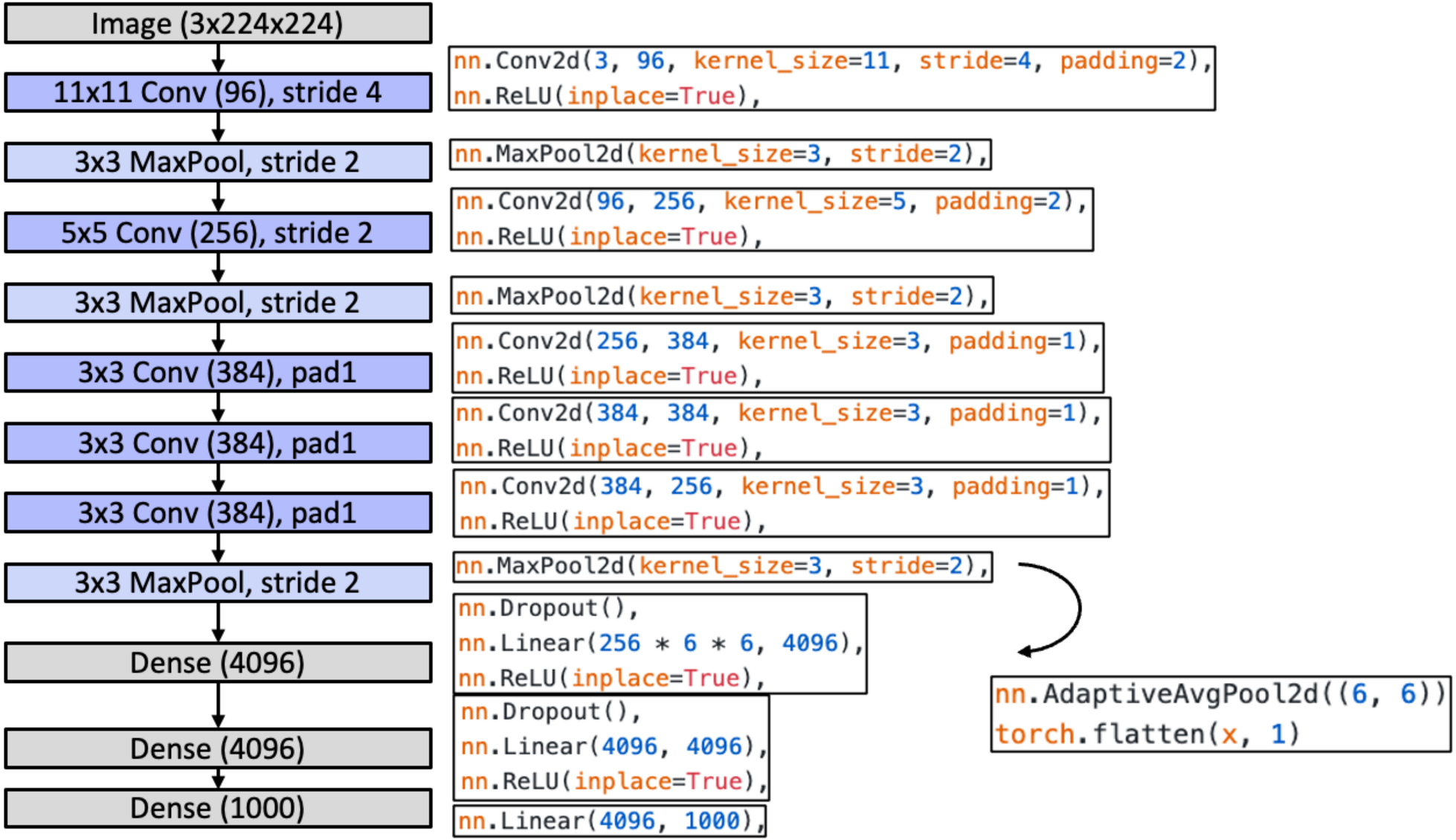
# AlexNet (2012)

- Bigger and deeper version of LeNet — won 2012 ImageNet challenge
  - 7 hidden layers, 605k neurons, 60 million parameters, plus dropout
- **Question.** How could they succeed in training bigger & deeper network?
  - Dataset. ImageNet large-scale visual recognition challenge
  - Optimization. Better activation (ReLU)
  - Generalization. Better regularization (Dropout)
  - Computation. Distributed GPU training (two GTX 580)

- And thus the beginning of the **scale race**!

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

# 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)

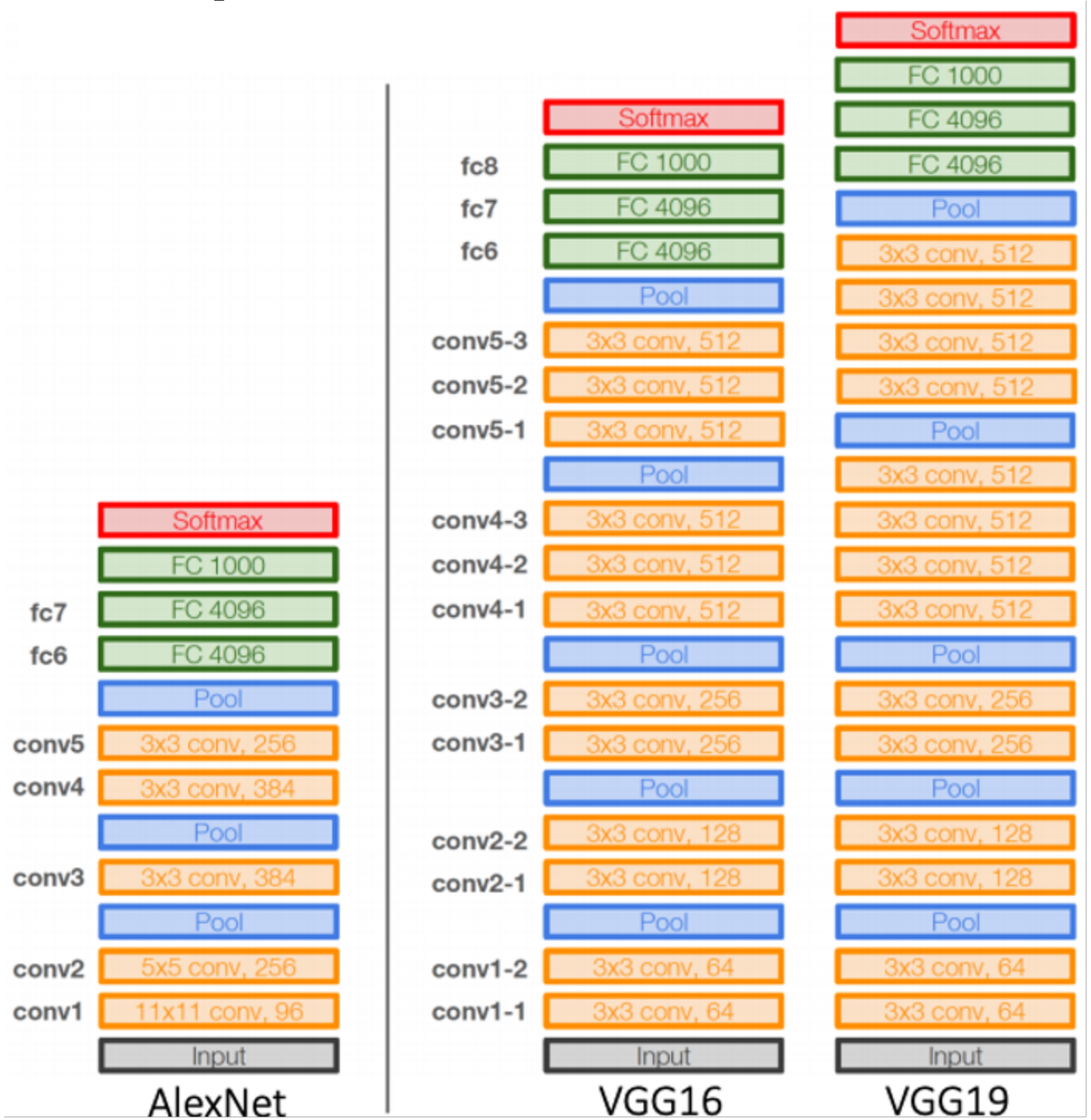
In this submission, image classification was performed using a conventional pipeline based on Fisher vector image representation and one-vs-rest linear SVM classifiers. In more detail, two types of local patch features were densely extracted over multiple scales: SIFT and colour statistics. The features 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).

# Deeper models

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

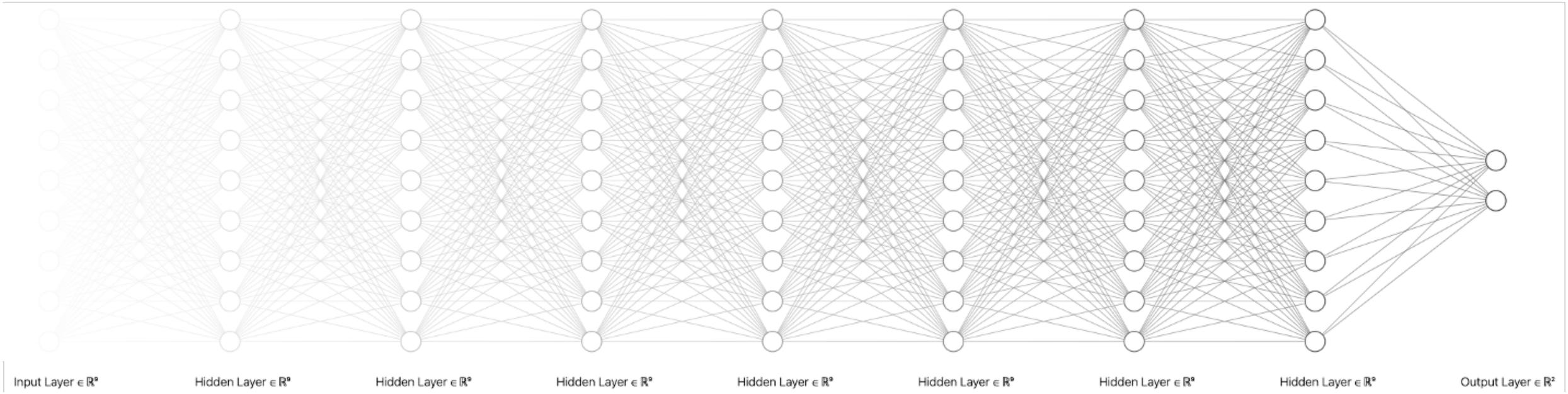
# VGG (2014)

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



# Stacking deeper

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



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

# Stacking deeper

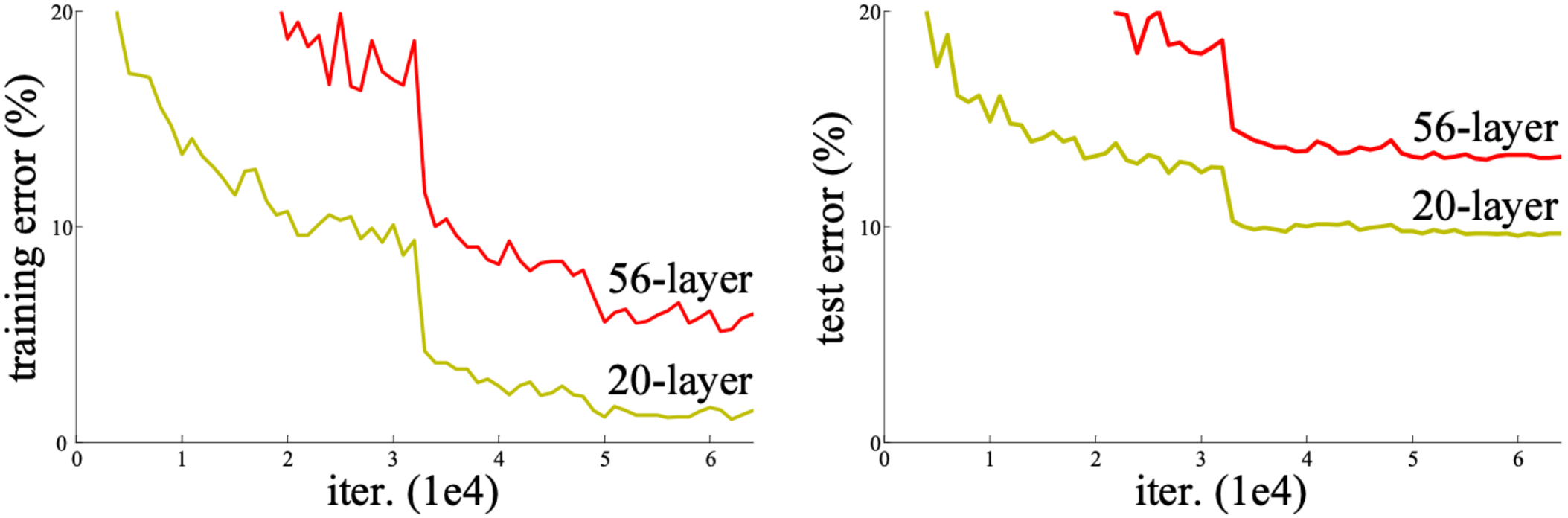
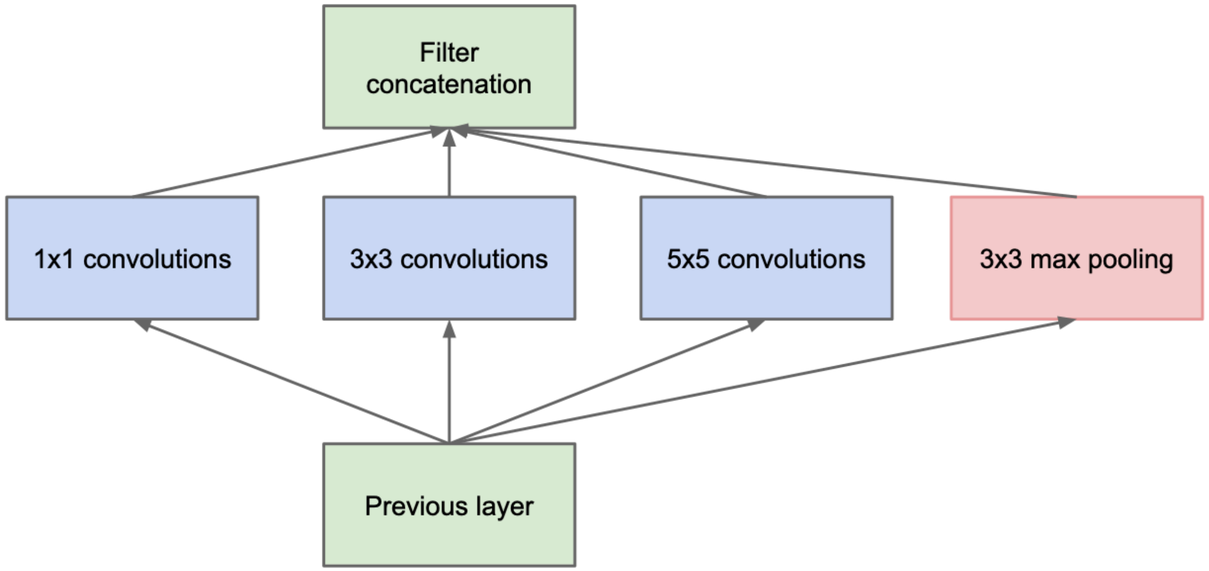



Figure 1. Training error (left) and test error (right) on CIFAR-10 with 20-layer and 56-layer “plain” networks. The deeper network has higher training error, and thus test error. Similar phenomena on ImageNet is presented in Fig. 4.

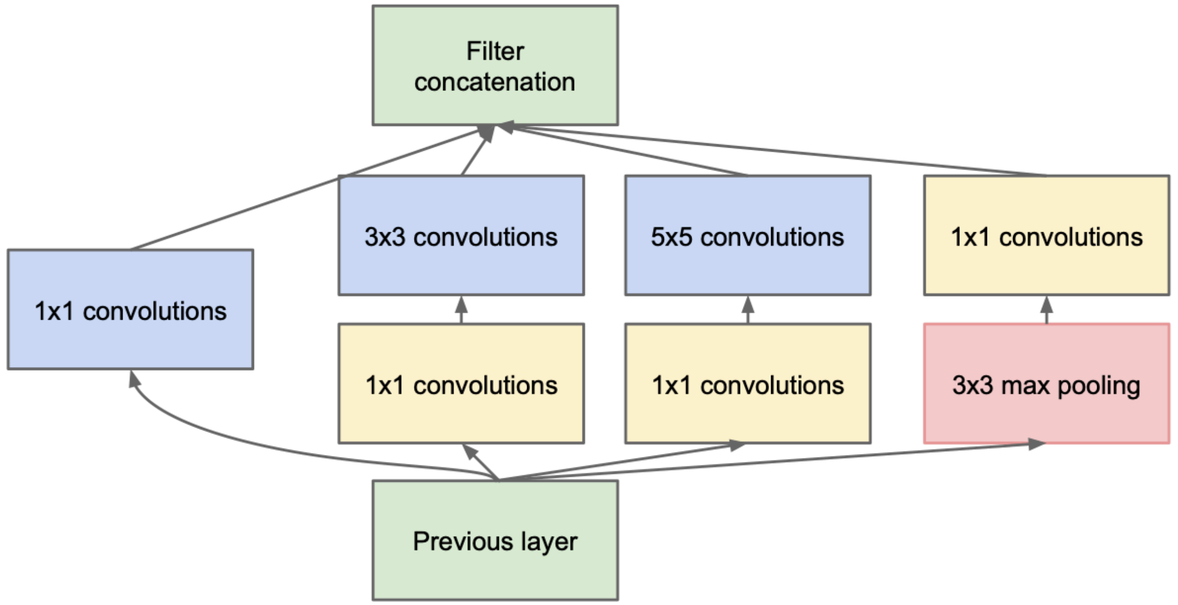
Szegedy et al., “Going deeper with convolutions,” arXiv 2014

# GoogLeNet (2014)

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




(a) Inception module, naïve version



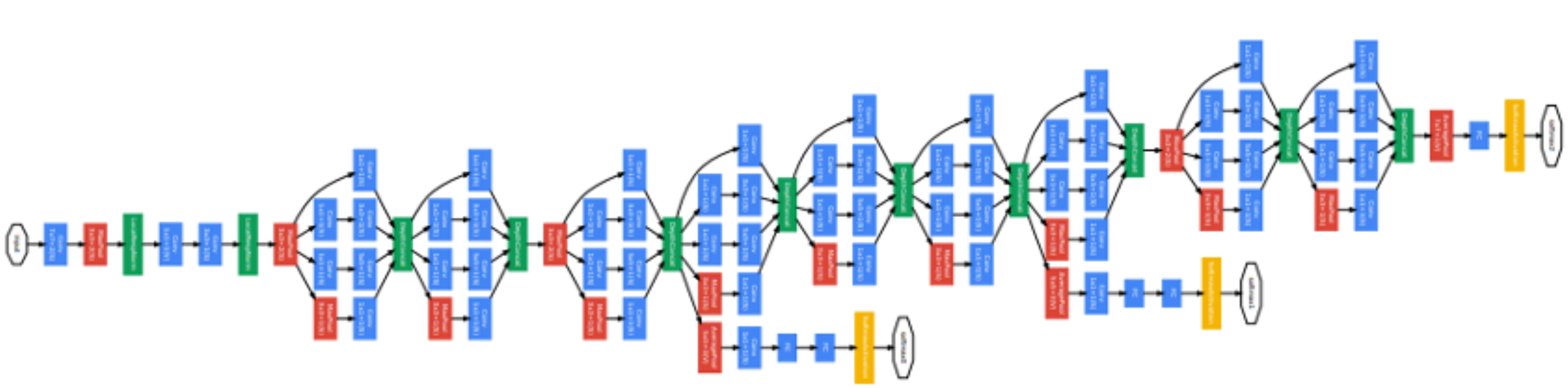

(b) Inception module with dimension reductions

Figure 2: Inception module

Szegedy et al., “Going deeper with convolutions,” arXiv 2014

# GoogLeNet (2014)

- Two notable differences
  - **Inception Module.** Works as a bottleneck to reduce #channels
  - **Auxiliary classifier.** Resolves vanishing gradient by adding more sources of backprop
    - Note. Only one FC layer for classification (reduces #params)



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

# ResNet (2016)

- Provides a more elegant solution to the vanishing gradient problem

- **Residual learning.** Outputs $\mathbf{x} + f_\theta(\mathbf{x})$, not $f_\theta(\mathbf{x})$

  - That is, $f_\theta$ represents an update that each layer contributes to the features

  - Example. Suppose that

$$\mathbf{y} = x + \sigma(wx)$$

Then, we have

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

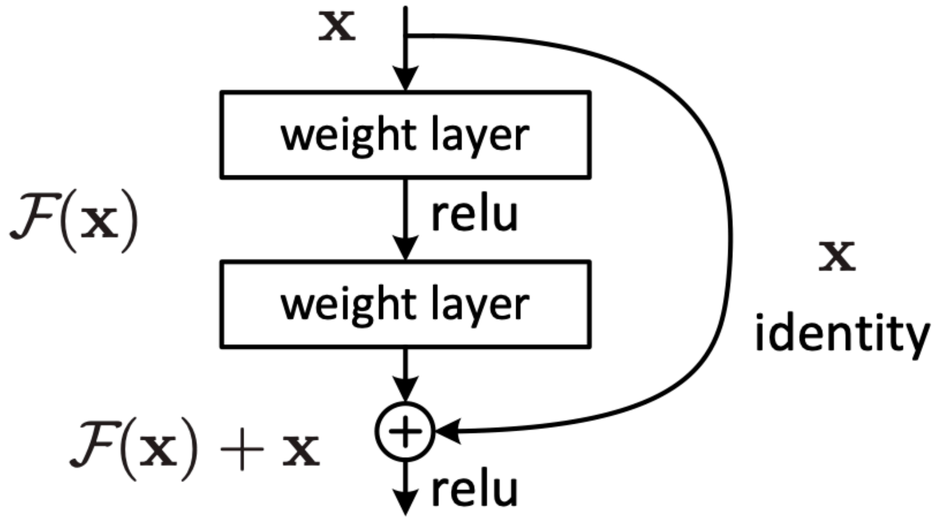
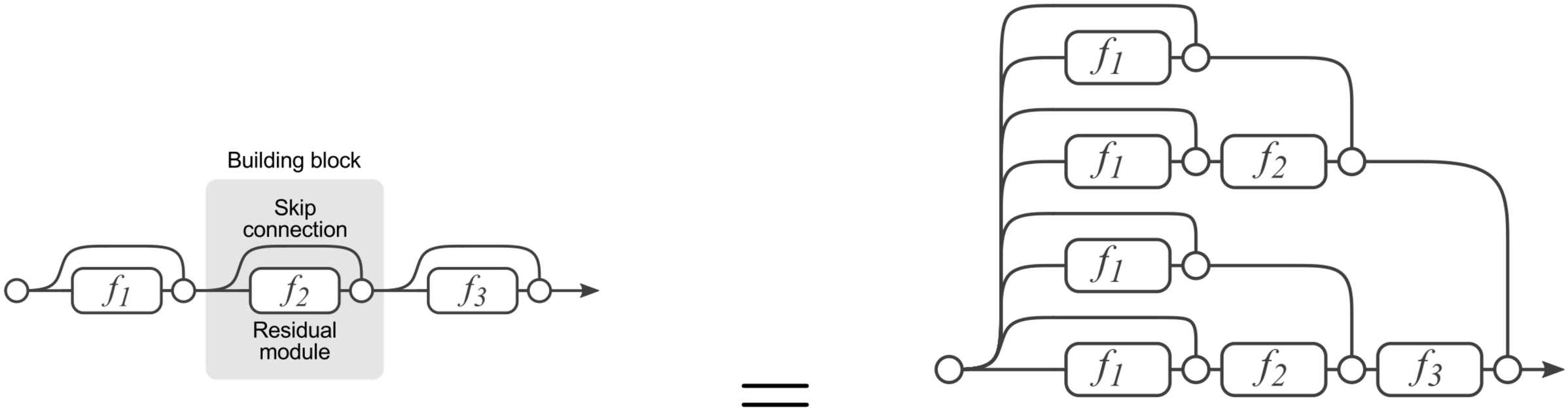



Figure 2. Residual learning: a building block.

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

# ResNet (2016)

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




(a) Conventional 3-block residual network

(b) Unraveled view of (a)

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

# ResNet (2016)

- In ResNets, the gradients come from the shorter paths
  - Similar to ensembling with many shortcut predictors
- Also introduces bottleneck blocks — accelerates training
  - Utilizes higher channel dimension, but 3x3 convolution is done in bottlenecks

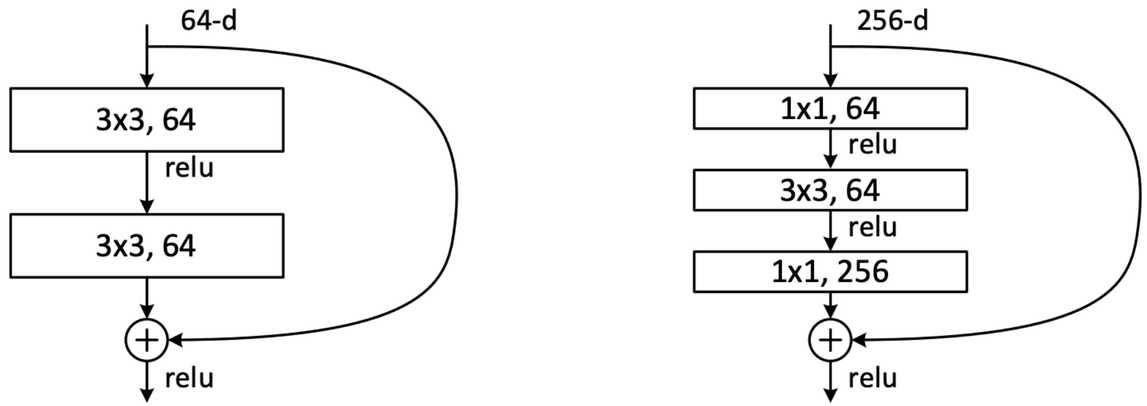
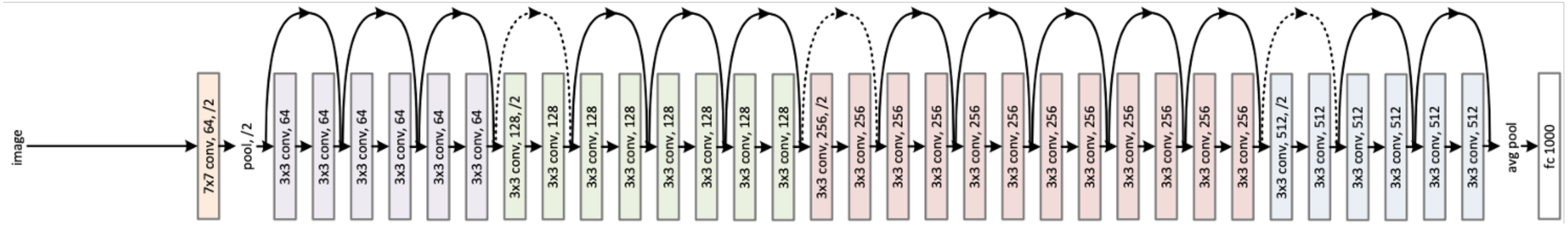



Figure 5. A deeper residual function $\mathcal{F}$ for ImageNet. Left: a building block (on 56×56 feature maps) as in Fig. 3 for ResNet-34. Right: a “bottleneck” building block for ResNet-50/101/152.

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

# ResNet (2016)

- Up to over 150 layers in total
  - A better initialization (He init.)
  - Batch normalization after each convolution
- Dotted line: doubles #channels & downsample by 2



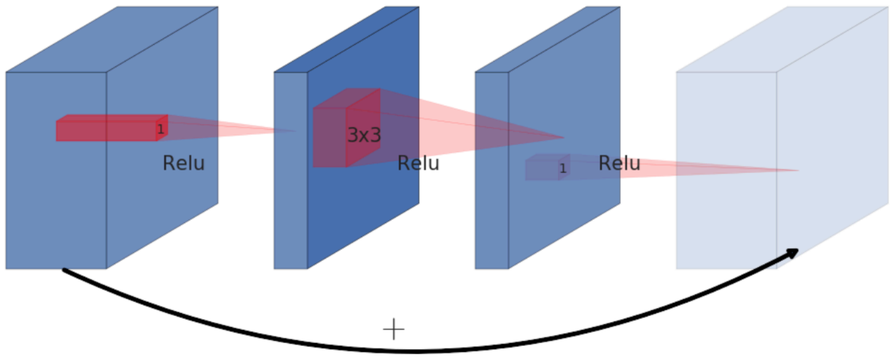
# Tiny models

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

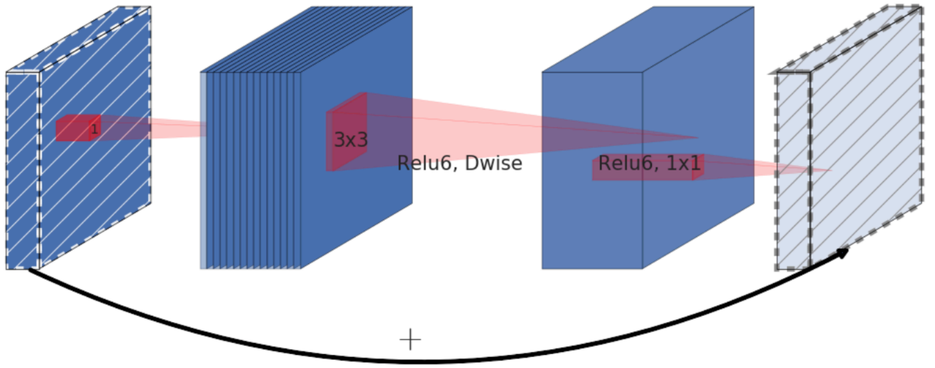
# MobileNets (2017, 2018, 2019)

- **Motivation.** Less number of parameters, for inference on mobile devices
- **Innovations.**
  - V1. Depthwise convolution
  - V2. Inverted Residual
    - Increase the number of channels, and do depthwise convolution
  - V3. Architecture search

(a) Residual block




(b) Inverted residual block



Lin et al., “MCUNet: Tiny Deep Learning on IoT Devices,” NeurIPS 2020

# MCUNets (2020, 2021, 2022)

- **Motivation.** Less memory requirement, for training on microcontrollers
- **Innovations.**
  - Algorithm / System Co-design
  - Neural Architecture Search
  - Better quantization-awareness



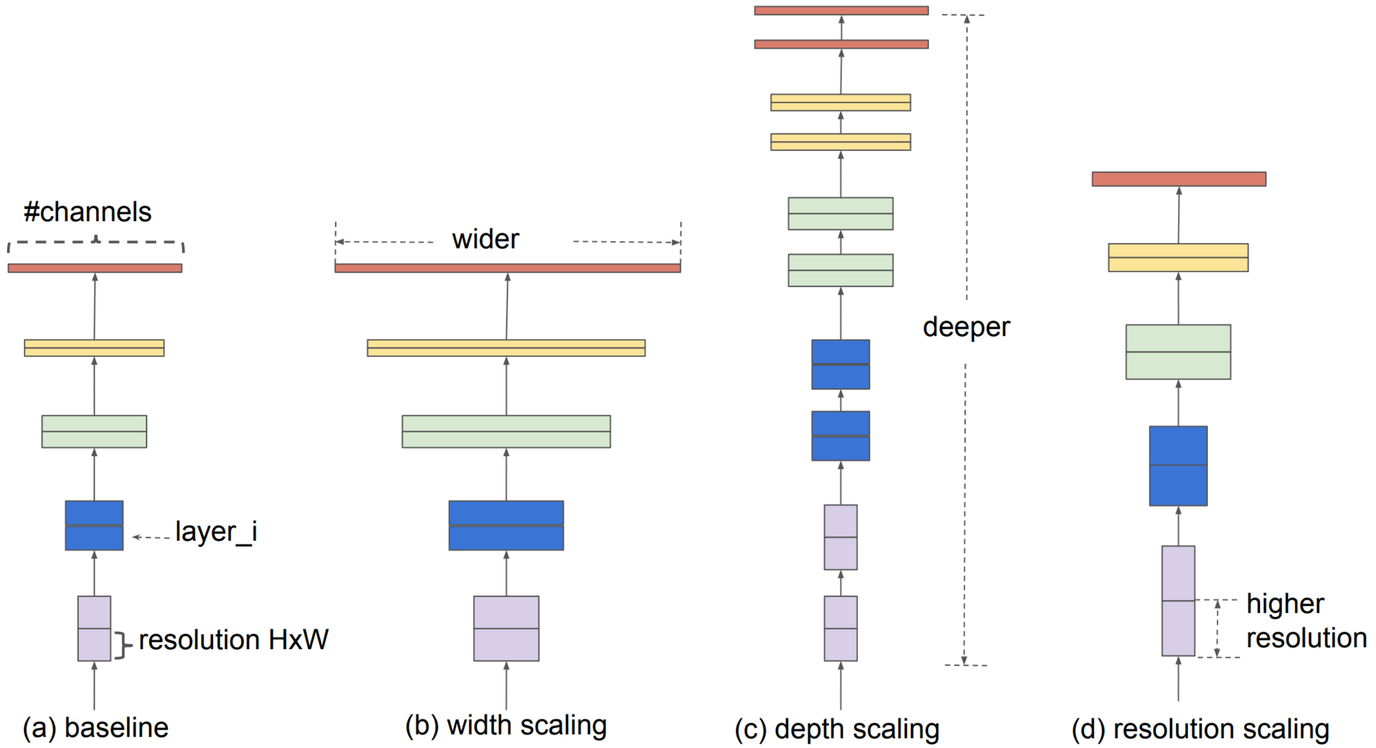
| | ResNet<br>**Cloud AI** | MobileNet<br>**Mobile AI** | | MCUNet<br>**Tiny AI** |
|---|---|---|---|---|
| **Memory** (Activation) | 16GB | 4GB | **13,000x smaller** → | 320kB |
| **Storage** (Weights) | ~TB/PB | 256GB | | 1MB |

# Efficiently scalable models

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 #param?
  - Increase depth / width / resolution (i.e., less downsampling)?



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 #param?
  - Increase depth / width / resolution (i.e., less downsampling)?
- **Answer.** All of them, by some ratio (empirically discovered)
  - v2 added NAS.

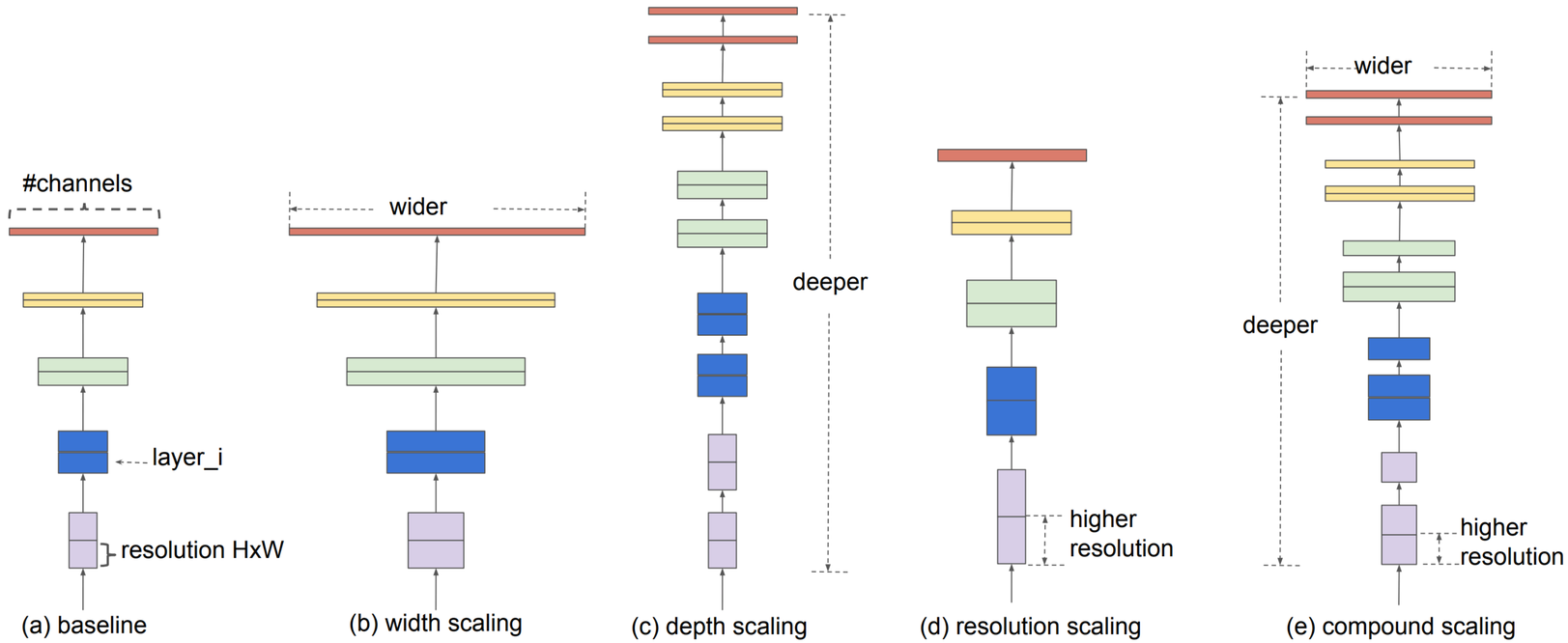
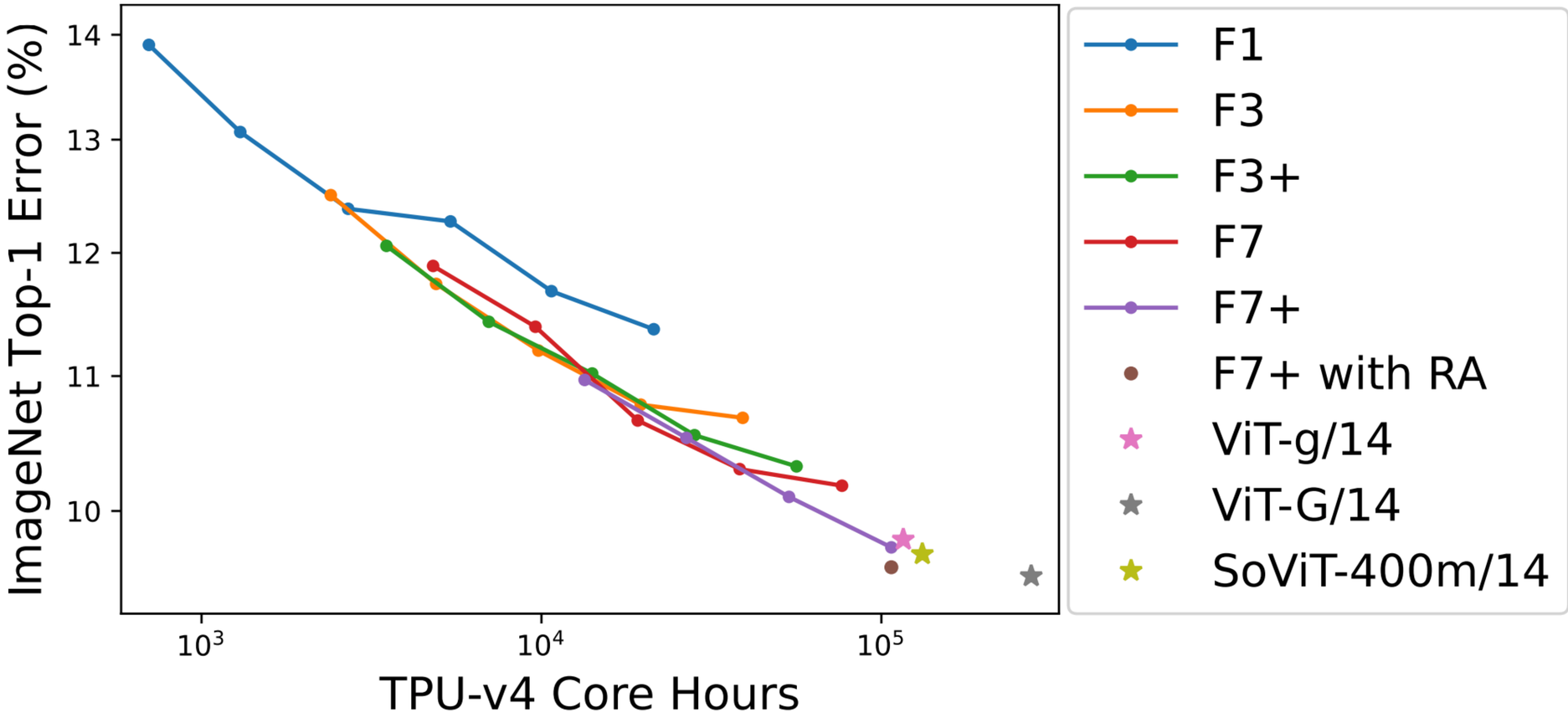



*Figure 2.* **Model Scaling.** (a) is a baseline network example; (b)-(d) are conventional scaling that only increases one dimension of network width, depth, or resolution. (e) is our proposed compound scaling method that uniformly scales all three dimensions with a fixed ratio.

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

# NFNets (2021, 2023)

- As of 2024, it seems like removing batch norms is a key to build very big ConvNets
  - Before this, people thought large convnets cannot work as good as transformers.



# Wrapping up

- **Next class.** Will talk about generative models for vision

# Cheers