Deep Learning

2. Convolutional Neural Networks

Term 4, 2025

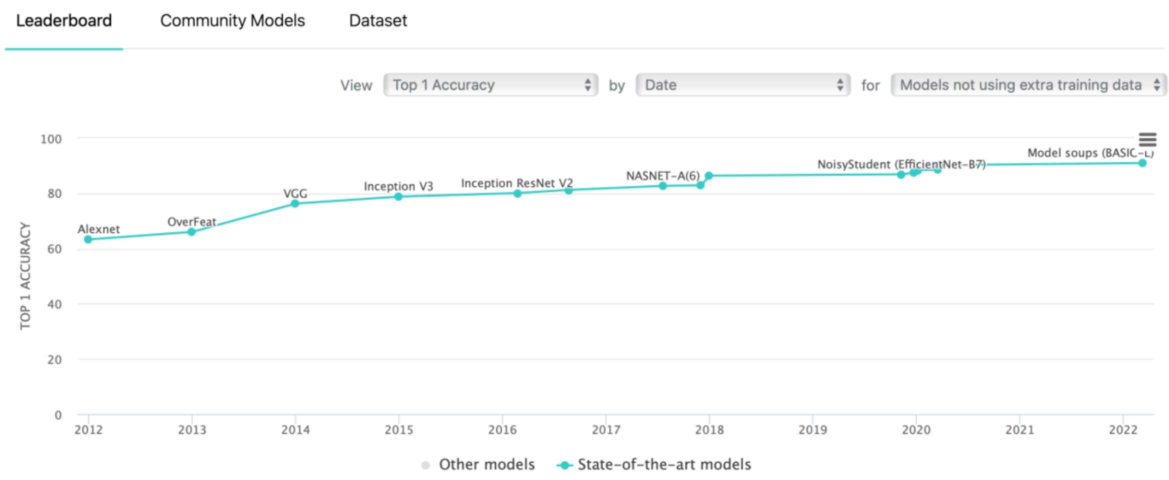
Skoltech

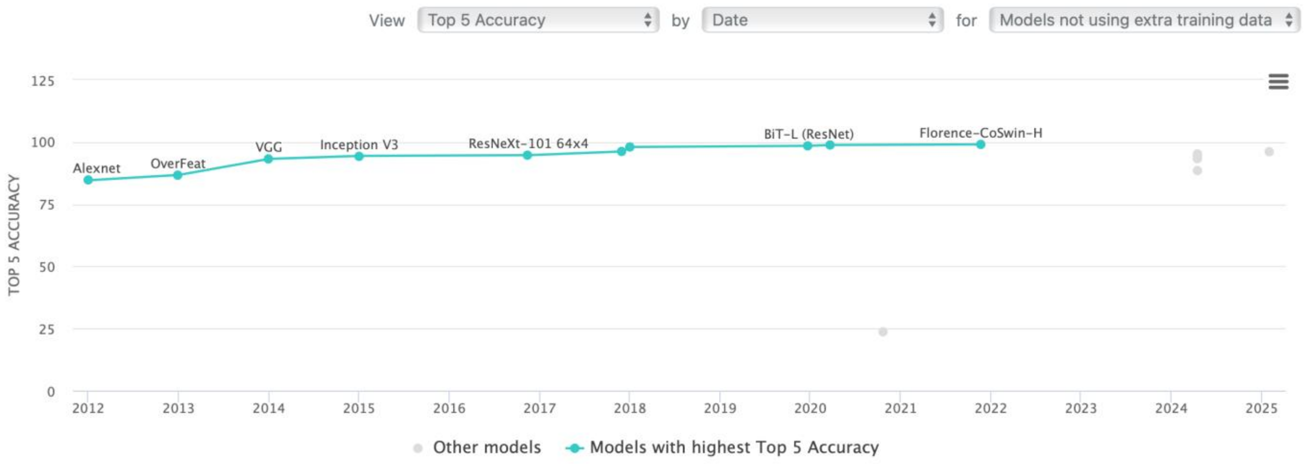
Outline of the lecture

- Introduction
- Biological inspiration
- Foundations in classical computer vision
- Why convolutions work for images
- Mathematical definition
- Early beginnings of CNNs
- The CNN winter and revival
- The deep learning revolution

Introduction

- CNNs revolutionized computer vision:
  - They became the dominant approach for image recognition tasks since 2012
  - They achieved superhuman performance on many visual tasks
    - Very high ImageNet classification accuracy





https://paperswithcode.com/sota/image-classification-on-imagenet

Introduction

- Inspired by the visual cortex of animals
  - The hierarchical structure of CNNs mirrors the organization of the mammalian visual system
    - Early layers detect simple features like edges and textures, while deeper layers recognize complex patterns
  - Hubel and Wiesel's Nobel Prize-winning work on the visual cortex provided the biological foundation
    - https://pmc.ncbi.nlm.nih.gov/articles/PMC1359523/
    - https://www.nobelprize.org/prizes/medicine/1981/press-release/

Introduction

- CNNs are designed to process grid-like data (images)
  - Unlike fully connected networks, CNNs preserve spatial relationships in data
  - CNNs can efficiently process high-dimensional visual data with fewer parameters
  - Mathematical properties of convolutions make them particularly suitable for image processing
    - https://arxiv.org/abs/1803.01164

Introduction

- Bridge between classical signal processing and modern deep learning
  - CNNs incorporate principles from traditional signal processing (filtering, downsampling)
  - CNNs automate the feature engineering process that was previously done manually
  - Zeiler and Fergus visualized learned features (we will see them later)
    - https://arxiv.org/abs/1311.2901

Biological Inspiration

- Hubel and Wiesel's experiments (1959-1962)
  - David Hubel and Torsten Wiesel recorded neural activity in the cat's visual cortex
  - They discovered that neurons respond to specific patterns of light in their receptive fields
  - Their 1962 paper established the foundation for understanding visual processing
    - https://www.ncbi.nlm.nih.gov/pmc/articles/PMC1359523/
  - They received the Nobel Prize in Physiology or Medicine in 1981 for this work
    - https://www.nobelprize.org/prizes/medicine/1981/press-release/

Biological Inspiration

- Discovery of simple and complex cells in visual cortex
  - Simple cells respond to oriented edges at specific positions
  - Complex cells respond to oriented edges regardless of position
  - This hierarchical organization inspired the design of CNNs
  - Fukushima's Neocognitron (1980) was the first model to implement this architecture
    - https://www.cs.princeton.edu/courses/archive/spr08/cos598B/Readings/Fukushima1980.pdf

Biological Inspiration

- Cells respond to specific patterns within receptive fields
  - Each neuron has a limited "view" of the visual field (receptive field)
  - Neurons at different levels have different receptive field sizes
    - This property is implemented in CNNs through varying kernel sizes and network depth
  - Visualization of receptive fields in CNNs shows striking similarities to biological systems
    - https://distill.pub/2017/feature-visualization
    - Will be displayed later

Biological Inspiration

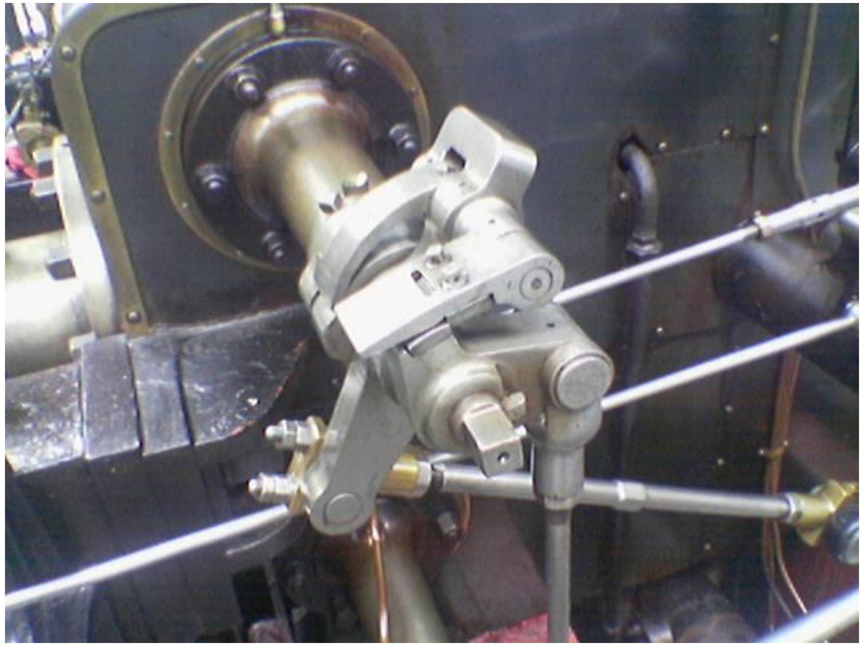
- Hierarchical processing of visual information
  - Information flows from simple features to complex patterns
  - Each layer builds upon the representations from previous layers
  - This hierarchical structure enables CNNs to learn increasingly abstract features
  - Zeiler and Fergus's visualization technique (2013) revealed what each layer learns
    - https://arxiv.org/abs/1311.2901
    - Will be displayed later

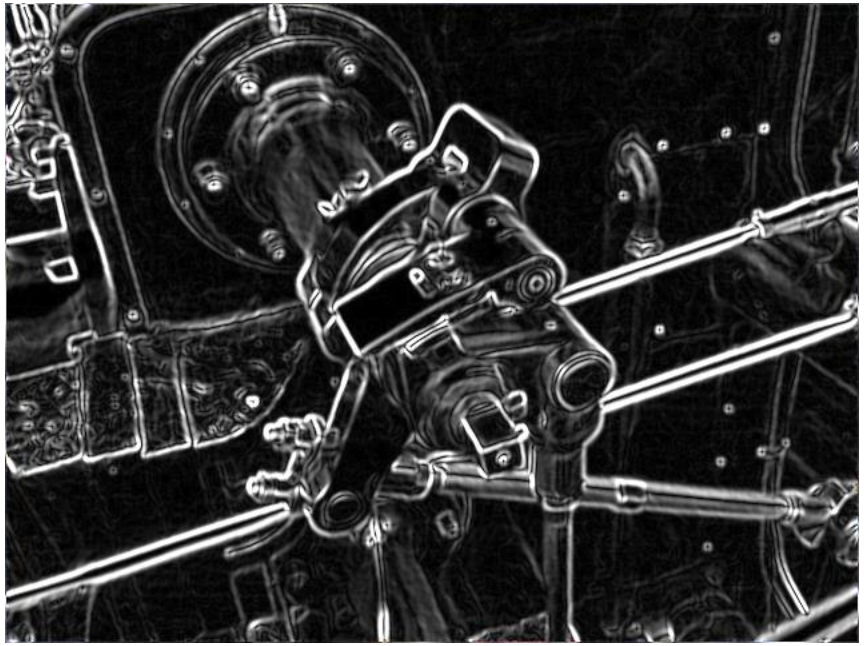
Foundations in Classical Computer Vision

- Traditional CV used hand-crafted filters
  - Edge detection (Sobel)

$$\mathbf{G}_x = \begin{bmatrix} -1 & 0 & +1 \\ -2 & 0 & +2 \\ -1 & 0 & +1 \end{bmatrix} * \mathbf{A}$$

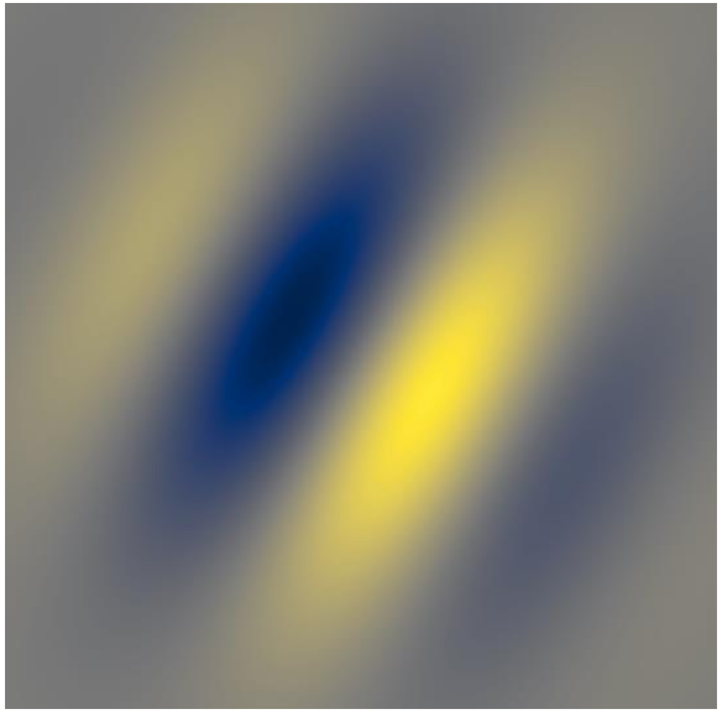
$$\mathbf{G}_y = \begin{bmatrix} -1 & -2 & -1 \\ 0 & 0 & 0 \\ +1 & +2 & +1 \end{bmatrix} * \mathbf{A}$$





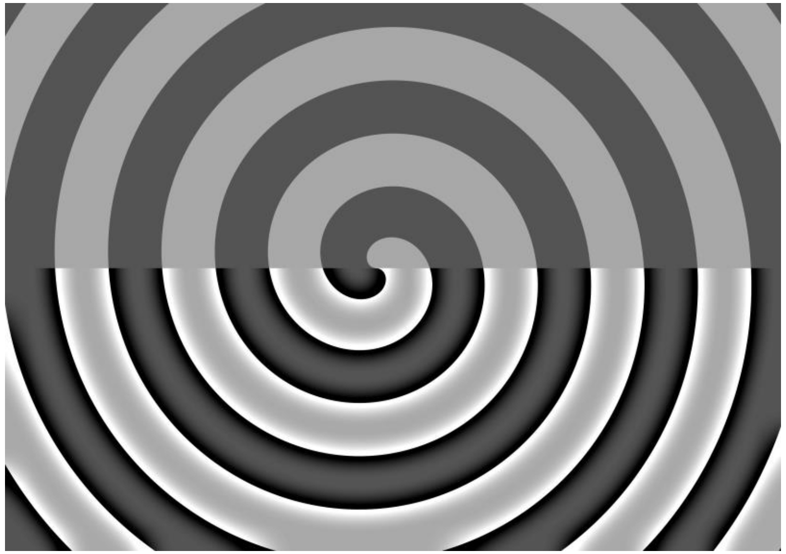
Foundations in Classical Computer Vision

- Traditional CV used hand-crafted filters
  - Gabor filters



Foundations in Classical Computer Vision

- Traditional CV used hand-crafted filters
  - Sharpening
    - Sharpening enhances edges by subtracting a blurred version from the original
    - Example kernel: [[0, -1, 0], [-1, 5, -1], [0, -1, 0]]
    - https://en.wikipedia.org/wiki/Unsharp_masking



By Nevit Dilmen - Own work, Public Domain, https://commons.wikimedia.org/w/index.php?curid=3884055

Foundations in Classical Computer Vision

- Traditional CV used hand-crafted filters
  - Strong edge detection
    - Laplacian of Gaussian (LoG) detects edges at multiple scales
    - Gaussian smoothing + Laplacian (sum of 2nd order derivatives) filters
    - https://homepages.inf.ed.ac.uk/rbf/HIPR2/log.htm

Why Convolutions Work for Images

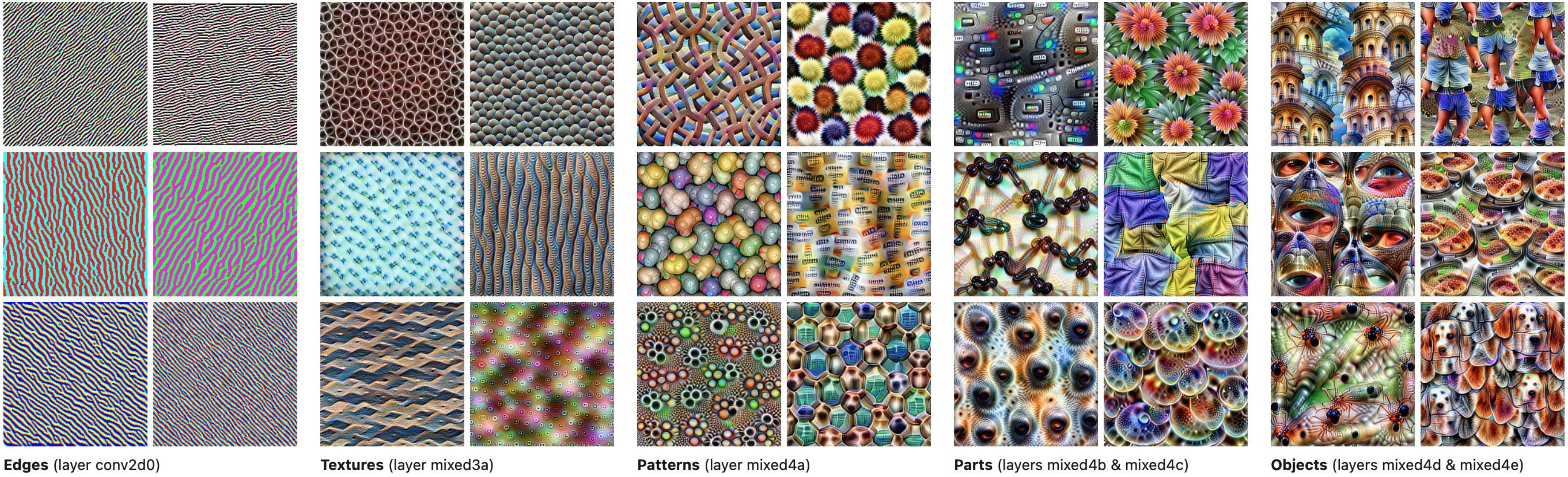
- Local connectivity
  - CNNs process small patches of the image at each layer
    - This reduces the number of parameters compared to fully connected networks
    - For a 1000×1000 image with a single input channel, a fully connected layer would need 10^{12} parameters, while a CNN with 3×3 kernels needs only 9 parameters per filter
  - The same weights are used for all positions in the image
    - This enables translation invariance and further reduces parameters
    - A 3×3 filter with 64 channels has only $3 \times 3 \times 64 = 576$ parameters, regardless of image size

Why Convolutions Work for Images

- Spatial hierarchy
  - Deeper layers capture more complex patterns
  - Early layers detect simple features (edges, textures)
  - Middle layers detect parts (eyes, wheels)
  - Deep layers detect objects (faces, cars)
  - Visualization of hierarchical features
    - https://distill.pub/2017/feature-visualization/

Why Convolutions Work for Images

- Spatial hierarchy: Deeper layers capture more complex patterns
  - Feature visualization allows us to see how GoogLeNet



https://distill.pub/2017/feature-visualization/

Why Convolutions Work for Images

- Translation invariance
  - Detect features regardless of position
  - CNNs can recognize objects even if they appear in different locations
  - This is achieved through the combination of convolutions and pooling

- Images exhibit locality properties
  - Nearby pixels are more related than distant ones
  - This property makes convolutions a natural choice for image processing
  - It's why CNNs are more efficient than fully connected networks for images

Why Convolutions Work for Images

- Cross-correlation vs. Convolution
  - In practice, CNNs use cross-correlation, not true convolution
  - In true convolution, the kernel is flipped before applying
  - CNNs typically use cross-correlation (no flipping)
  - This is a minor difference that doesn't affect the learning process
  - Convolution is actually used in the backward propagation phase
  - https://en.wikipedia.org/wiki/Cross-correlation

Mathematical Definition

- Mathematical definition of a 2D convolution

$$(I * K)(x, y) = \sum_{i=-a}^{a} \sum_{j=-b}^{b} I(x-i, y-j) K(i, j)$$

- The learnable kernel K defines the weights of convolution
- I is the input image or feature map
- (x, y) are the coordinates in the output feature map

Mathematical Definition

- Mathematical definition of a 2D cross correlation

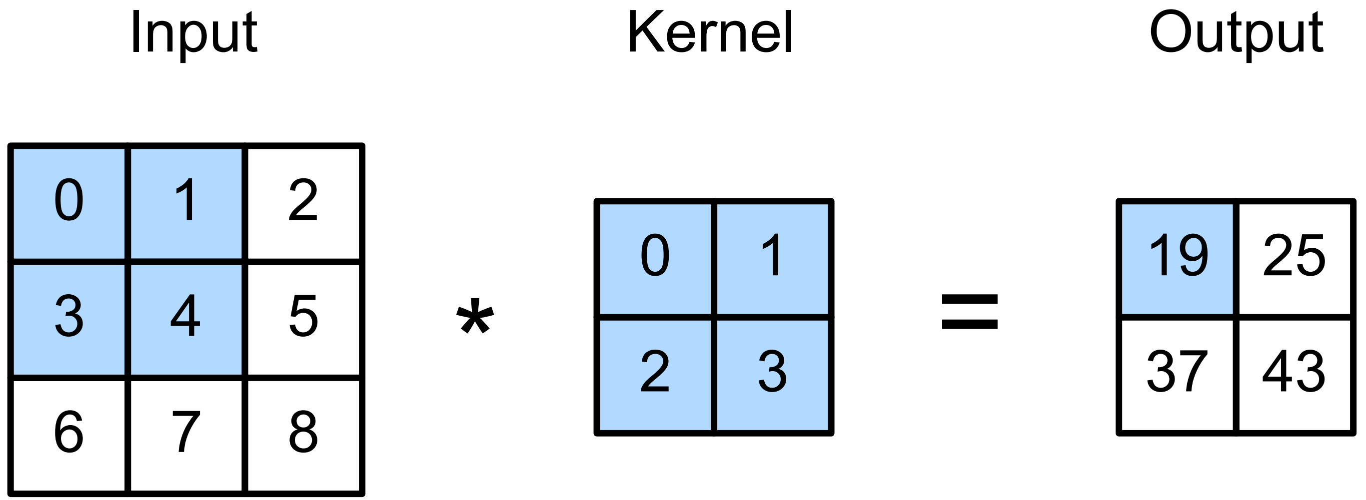
$$(I * K)(x, y) = \sum_{i=-a}^{a} \sum_{j=-b}^{b} I(x+i, y+j)K(i, j)$$

- The learnable kernel K defines the weights of cross correlation
- I is the input image or feature map
- (x, y) are the coordinates in the output feature map

Mathematical Definition

- Mathematical definition of a 2D cross correlation

$$(I * K)(x, y) \ = \sum_{i=-a}^{a} \sum_{j=-b}^{b} I(x+i, y+j)K(i, j)$$



Source: https://raw.githubusercontent.com/oseledets/dl2024/8d739d06892c9cc71f198c1c478d51b4a6801866/lectures/lecture-2/correlation.svg

Mathematical definition

- Cross correlation

$$O(x,y) = \sum_{i=-a}^{a} \sum_{j=-b}^{b} I\left(s_x x + d_x i, s_y y + d_y j\right) K(i,j)$$

  - Padding expands bounds of an input
    - Extension is filled with zeros or by some other rule
  - Strides $\left(s_x, s_y\right)$
    - Defines size of a step in each direction of output
  - Dilation $\left(d_x, d_y\right)$
    - Defines size of a step in each direction of kernel
  - Visualised at https://github.com/vdumoulin/conv_arithmetic/blob/master/README.md

Data Formats

- Shape of a batch of images is described by
  - N: Batch size (number of images)
  - C: Number of channels (e.g., 3 for RGB)
  - H: Height of the image
  - W: Width of the image
- Batches are stored in one of the standard row-major formats
  - NHWC: better for CPUs and Tensor-core GPUs, default for Tensorflow
  - NCHW: better for non-Tensor-core GPUs, default for Pytorch
    - https://forums.developer.nvidia.com/t/nhwc-vs-nchw-convolution/111065

NCHW Data Format

- NCHW Format (2 images, 3 channels, 2x4 pixels):

```
┌─────────────────────────────────────────────────────────────────┐
| Image 1      ┌───────────────┐ ┌───────────────┐ ┌───────────────┐ |
|              | R Channel     | | G Channel     | | B Channel     | |
|              |   1  2  3  4  | |   9 10 11 12  | |  17 18 19 20  | |
|              |   5  6  7  8  | |  13 14 15 16  | |  21 22 23 24  | |
|              └───────────────┘ └───────────────┘ └───────────────┘ |
| Image 2      ┌───────────────┐ ┌───────────────┐ ┌───────────────┐ |
|              | R Channel     | | G Channel     | | B Channel     | |
|              |  25 26 27 28  | |  33 34 35 36  | |  41 42 43 44  | |
|              |  29 30 31 32  | |  37 38 39 40  | |  45 46 47 48  | |
|              └───────────────┘ └───────────────┘ └───────────────┘ |
└─────────────────────────────────────────────────────────────────┘
```

Forward pass of a Convolutional Layer

- Input
  - Feature map of shape $H \times W \times C_i$
  - H and W are the height and width of the input
  - C_i is the number of input channels (e.g., 3 for RGB images)
- Kernel
  - Shape is $H_K \times W_K \times C_i \times C_o$
  - H_K and W_K are the height and width of the kernel
  - C_i is the number of input channels
  - C_o is the number of output channels (number of filters)

Forward pass of a Convolutional Layer

- Output

$$O(n, m, c_o) = \sum_{i,j,c_o} [I(n+i, m+j, c_i) \times K(i, j, c_i, c_o)] + b(c_o)$$

  - Feature map of shape $H' \times W' \times C_o$
  - H' and W' are the height and width of the output
    - Depend on input size, kernel sizem, padding, stride and dilation
    - What is the exact dependency?
  - C_o is the number of output channels (number of filters)
  - $b(c_o)$ is the bias term for each output channel

Backward pass of a Convolutional Layer

- Gradient w.r.t. input:

$$\frac{\partial L}{\partial I(n,m,c_i)} = \sum_{i,j,c_o} \left[\frac{\partial L}{\partial O(n-i, m-j, c_o)} \times K(i,j,c_i,c_o) \right]$$

  - This is a convolution
  - This formula computes how the loss changes with respect to each input element
  - It is used for backpropagation through the network

Backward pass of a Convolutional Layer

- Gradient w.r.t. kernel:

$$\frac{\partial L}{\partial K(i, j, c_i, c_o)} = \sum_{n,m} \left[\frac{\partial L}{\partial O(n, m, c_o)} \times I(n + i, m + j, c_i) \right]$$

  - This formula computes how the loss changes with respect to each kernel element
  - It is used to update the kernel weights during training

Backward pass of a Convolutional Layer

- Gradient w.r.t. bias:

$$\frac{\partial L}{\partial b(c_o)} = \sum_{n,m} \frac{\partial L}{\partial O(n,m,c_o)}$$

  - This formula computes how the loss changes with respect to each bias element
  - It is used to update the convolutional bias during training

Early beginnings of CNNs

- Kunihiko Fukushima's Neocognitron (1980)
  - Hierarchical, multi-layered network
    - First model with a hierarchical structure similar to the visual cortex
    - https://www.cs.princeton.edu/courses/archive/spr08/cos598B/Readings/Fukushima1980.pdf
  - First implementation of convolutional structure
    - Used local receptive fields and shared weights
    - Pre-dated modern CNNs by almost two decades
  - Self-organizing model inspired by visual cortex
    - Could learn to recognize patterns without explicit supervision
    - Based on biological principles of visual processing

Early beginnings of CNNs

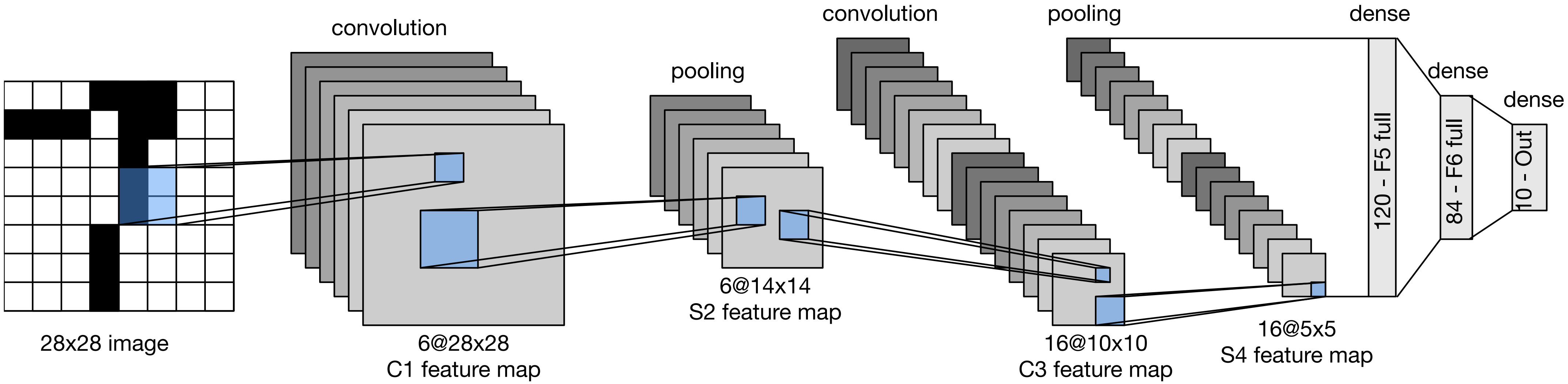
- Yann LeCun's LeNet-5 (1998)
  - First modern CNN architecture
    - http://vision.stanford.edu/cs598_spring07/papers/Lecun98.pdf
  - Handwritten digit recognition (MNIST)
    - Achieved 0.8% error rate on the MNIST dataset
    - Used by banks to recognize handwritten digits on checks
  - Used backpropagation for training
    - Demonstrated that CNNs could be trained end-to-end
    - Showed the power of gradient-based learning
  - Used average pooling for downsampling

Early beginnings of CNNs

- What is an average pooling?
  - Pooling is a common operation in CNNs used to downsample feature maps
  - Average pooling takes the average value of each sub-region of the input
  - It is a special case of a convolutional layer with predefined kernel
  - It reduces the size of the feature map while introducing a form of spatial invariance
  - Average pooling helps to reduce overfitting and improve generalization
  - However, it may lose information about the precise location of features
  - Alternatives to average pooling include **max pooling**, **adaptive pooling**
  - Size of the **adaptive pooling** is defined by the input and output shapes

Early beginnings of CNNs

- Yann LeCun's LeNet-5 (1998)



Early beginnings of CNNs

- Yann LeCun's LeNet-5 (1998)
  - Trained and tested on MNIST dataset (60,000 samples)
    - Short for "Modified National Institute of Standards and Technology database "
    - MNIST became the "Hello World" of deep learning
    - "Everything works on MNIST"
    - https://yann.lecun.org/exdb/mnist/index.html

The CNN Winter and Revival

- CNNs faced limited adoption (1990s-2000s)
  - Computational limitations
    - Training deep networks required significant computational resources
    - GPUs were not yet widely used for deep learning
  - Limited training data
    - Small datasets like MNIST were insufficient for deep networks
    - Data augmentation techniques were not yet developed
  - SVM and other methods dominated CV
    - Support Vector Machines and other shallow models were more popular
    - They performed well on small datasets and were easier to train

The CNN Winter and Revival

- ImageNet
  - 2006: Fei-Fei Li proposed the creation of ImageNet
    - "While most people pay attention to models, let's pay attention to data“
  - July 2008: ImageNet had zero images
  - December 2008: 3 million images across 6000 synsets
  - April 2010: 11 million images across 15000 synsets
  - Today (March 2025): 14.2 million images across 21841 synsets
  - Made possible through crowdsourcing on Amazon's Mechanical Turk
  - Provided enough data to train deep networks
  - https://image-net.org/

The CNN Winter and Revival

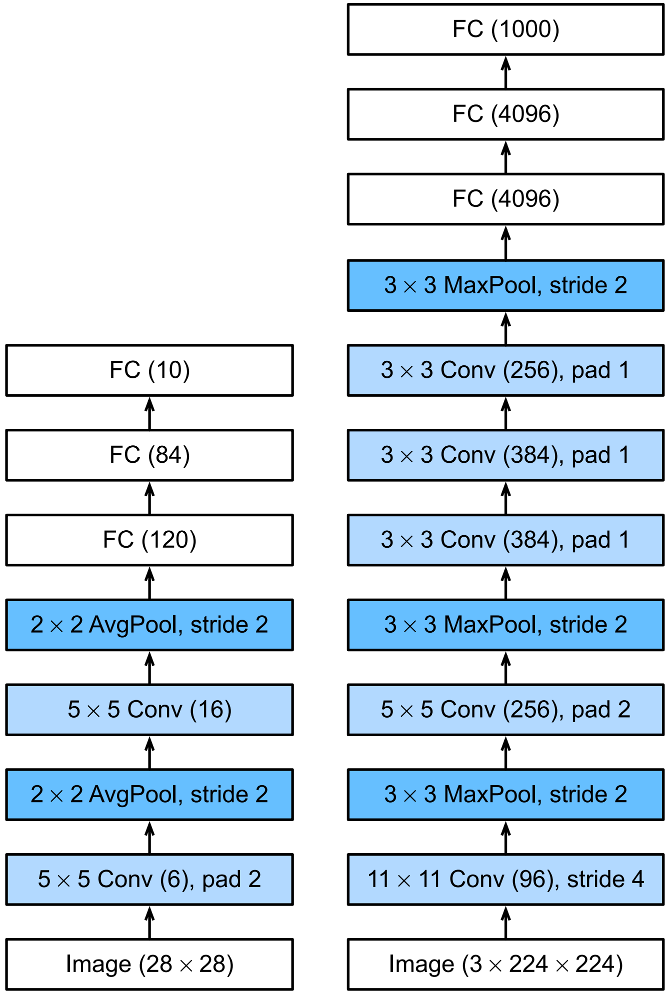
- ILSVRC (ImageNet Large Scale Visual Recognition Competition)
  - Organized in 2010 for the first time
  - Provided an ImageNet large scale subset for training
    - 1.2 million training images across 1,000 categories
  - Annual competition for image classification (2010-2017)
  - ILSVRC became the benchmark for computer vision
  - https://image-net.org/challenges/LSVRC/

The AlexNet Breakthrough

- The first deep CNN to win ILSVRC
  - Achieved 15.3% top-5 error rate (previous best: 26%)
  - This breakthrough sparked the deep learning revolution
  - https://image-net.org/challenges/LSVRC/2012/results.html
  - https://papers.nips.cc/paper_files/paper/2012/file/c399862d3b9d6b76c8436e924a68c45b-Paper.pdf
- Key innovations
  - Simple ReLU activation: $f(x) = \max(x, 0)$
  - Dropout: zeros random values during training
  - GPU training: 5-6 days to train on two GTX 580 GPUs with 3GB VRAM

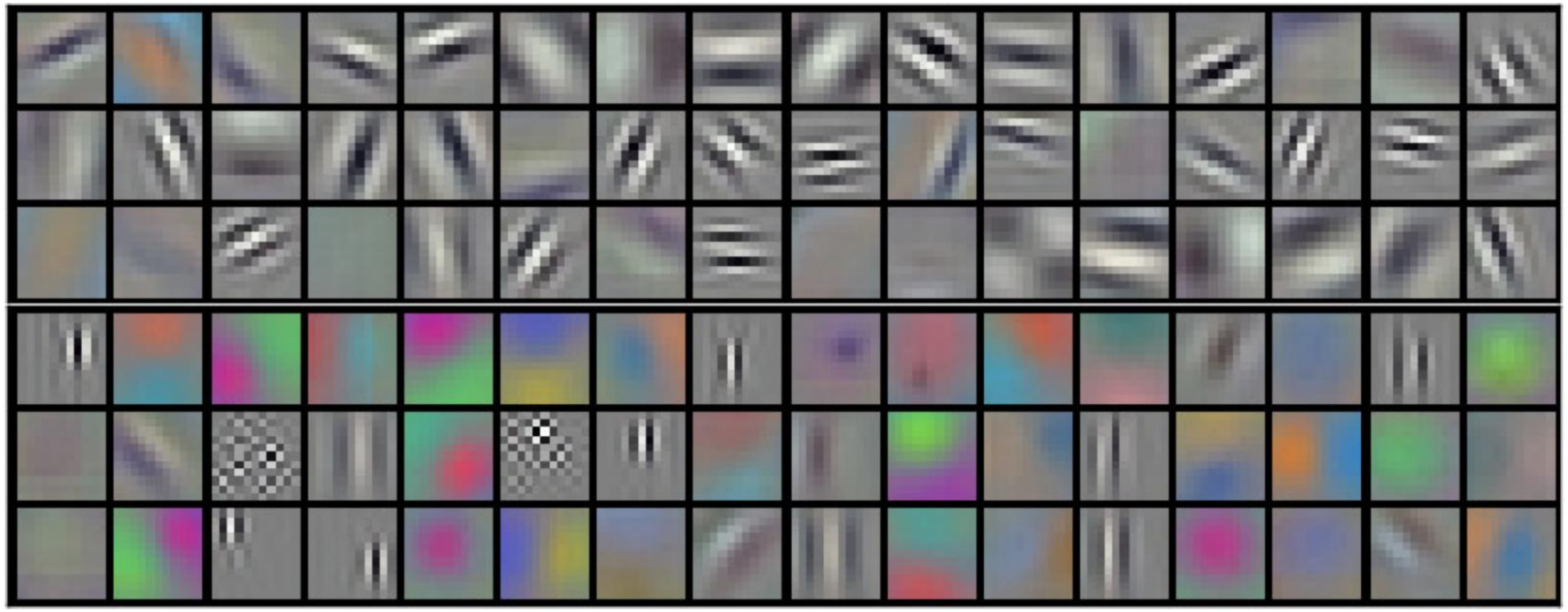
The AlexNet Breakthrough

- Architecture limitations
  - Large convolutional filters in early layers
    - 11×11 and 5×5 filters in early layers
    - Inefficient compared to smaller filters
  - Two large MLP layers at the end
    - 6400×4096 and 4096×4096 linear layers
    - These fully connected layers contain most of the parameters
    - Inefficient for modern CNN architectures
  - Total of around 60M parameters
    - Large number of parameters makes the model computationally expensive



The AlexNet Breakthrough

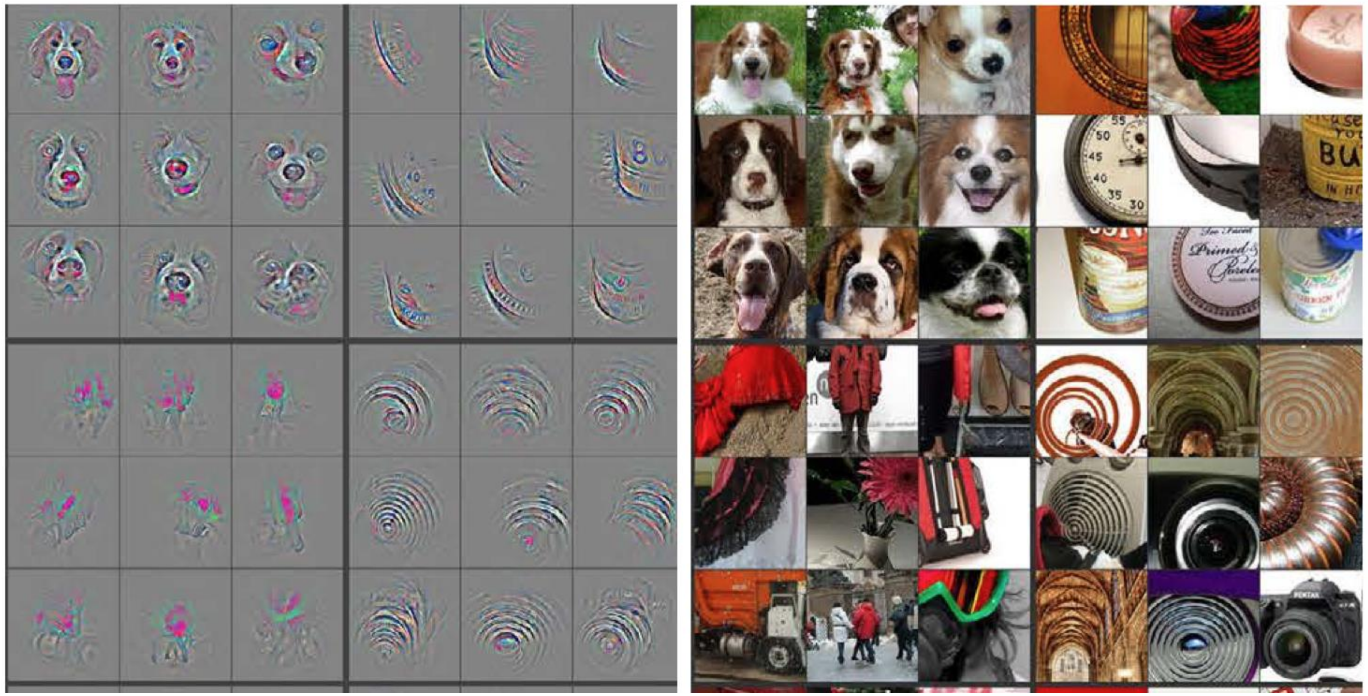
- Visually presented 96 convolutional kernels of size 11x11x3 learned by the first convolutional layer on the 224x224x3 input images



https://papers.nips.cc/paper_files/paper/2012/file/c399862d3b9d6b76c8436e924a68c45b-Paper.pdf

The Deep Learning Revolution

- ZFNet (2013): Visualization of CNN layers
  - Improved AlexNet by adjusting hyperparameters
  - Developed visualization techniques to understand what CNNs learn
  - https://arxiv.org/abs/1311.2901

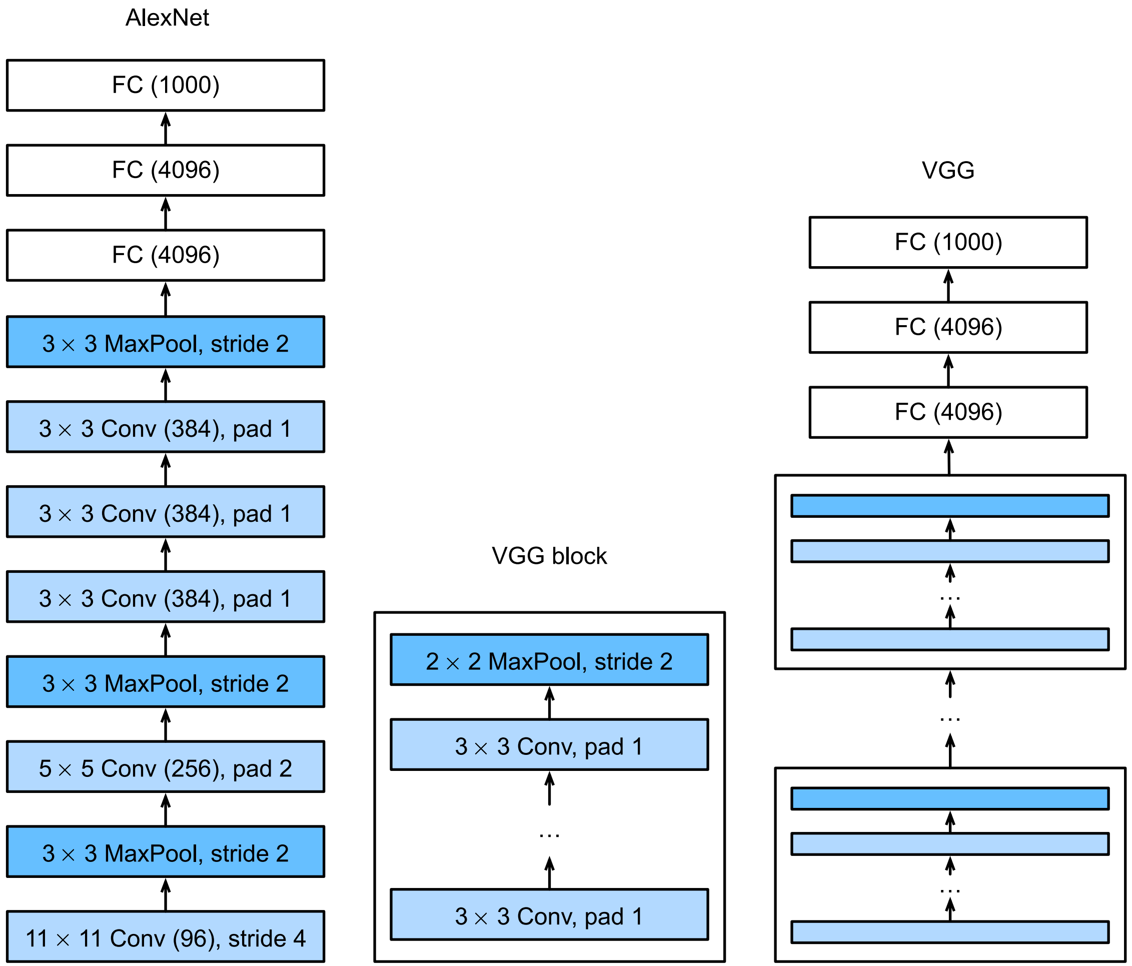


The Deep Learning Revolution

- VGGNet (2014): Simplicity and depth (3x3 convolutions)
  - Won ILSVRC 2014
  - Key idea: Use multiple 3×3 convolutions between MaxPooling downsampling
  - Example: two successive 3x3 convolutional layers against a single 5x5 layer
    - Receptive field is the same and output feature maps are of the same shape
    - Two 3x3 layers require 18 parameters
    - Single 5x5 layer requires 25 parameters
  - Showed that deep and narrow convolutions outperform wider counterparts
  - Replaces larger filters with stacks of 3×3 filters
  - 3 x 3 convolutions became de-facto standard.
  - https://arxiv.org/abs/1409.1556

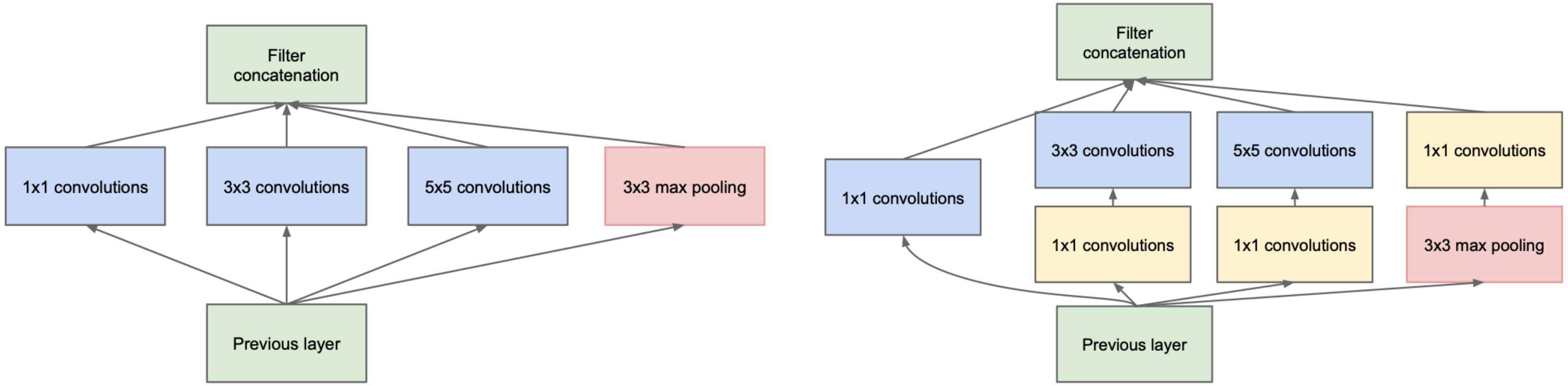
The Deep Learning Revolution

- VGGNet (2014)
  - VGG networks trained on ImageNet are excellent **feature extractors**.
  - VGG-11: 8 convolutional and 3 fully-connected layers
  - VGG-16: 13 convolutional and 3 fully-connected layers
  - VGG-19: 16 convolutional and 3 fully-connected layers



The Deep Learning Revolution

- GoogLeNet/Inception (2014): Inception modules
  - Won ILSVRC 2014
  - Multi-branch networks
  - Parallel paths with different filter sizes
  - https://arxiv.org/abs/1409.4842



(a) Inception module, naïve version (b) Inception module with dimension reductions

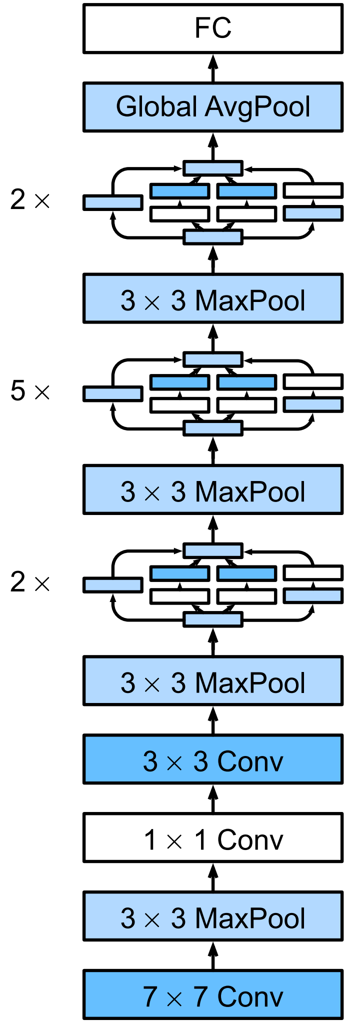
https://arxiv.org/abs/1409.4842

The Deep Learning Revolution

- GoogLeNet/Inception (2014)
  - First few layers: feature extracting stem
  - Last layer: classification head
  - Everything in between: body

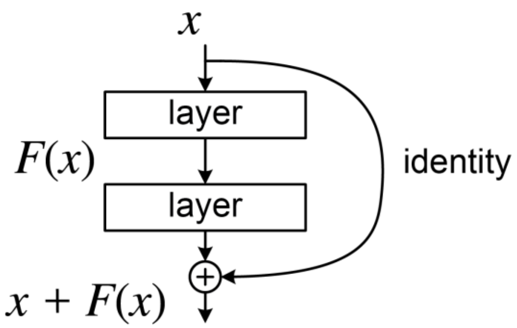
Team	Year	Place	Error (top-5)	Uses external data
SuperVision	2012	1st	16.4%	no
SuperVision	2012	1st	15.3%	Imagenet 22k
Clarifai	2013	1st	11.7%	no
Clarifai	2013	1st	11.2%	Imagenet 22k
MSRA	2014	3rd	7.35%	no
VGG	2014	2nd	7.32%	no
GoogLeNet	2014	1st	6.67%	no

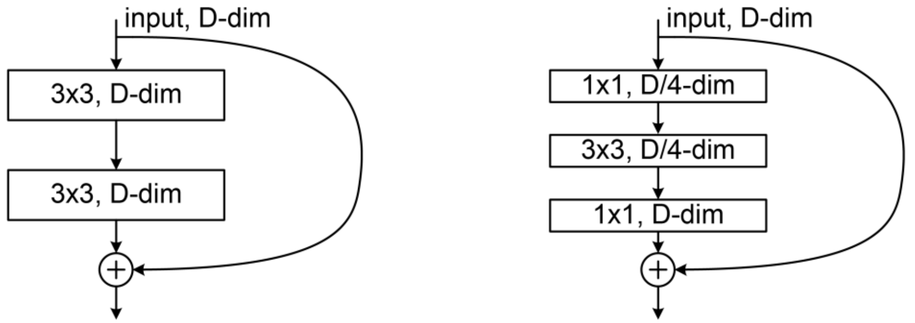
https://arxiv.org/pdf/1409.4842



The Deep Learning Revolution

- ResNet (2015): Skip connections, ultra-deep networks
  - Residual block: Instead of mapping y := f(x), learn y := x + f(x)
    - Backprop without residual: $\frac{\partial L}{\partial x} = \frac{\partial L}{\partial y} \cdot \frac{\partial y}{\partial x}$
    - Backprop with residual: $\frac{\partial L}{\partial x} = \frac{\partial L}{\partial y} \cdot \frac{\partial y}{\partial x} + \frac{\partial L}{\partial y}$



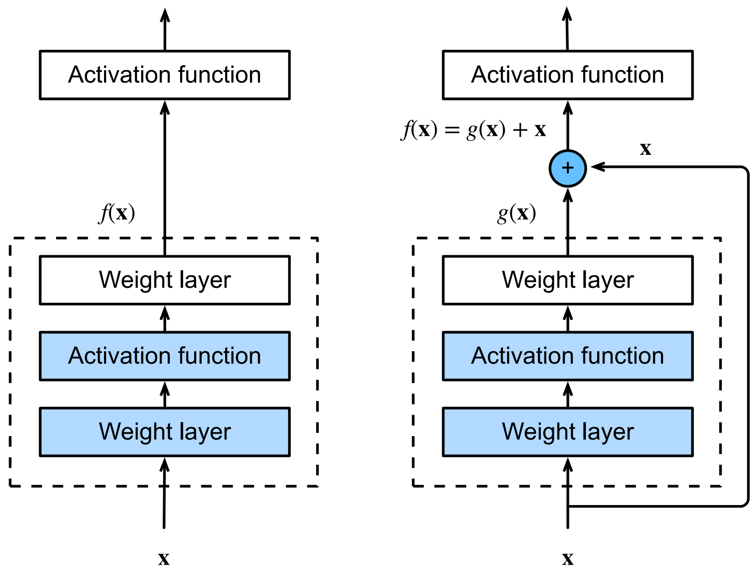


The Deep Learning Revolution

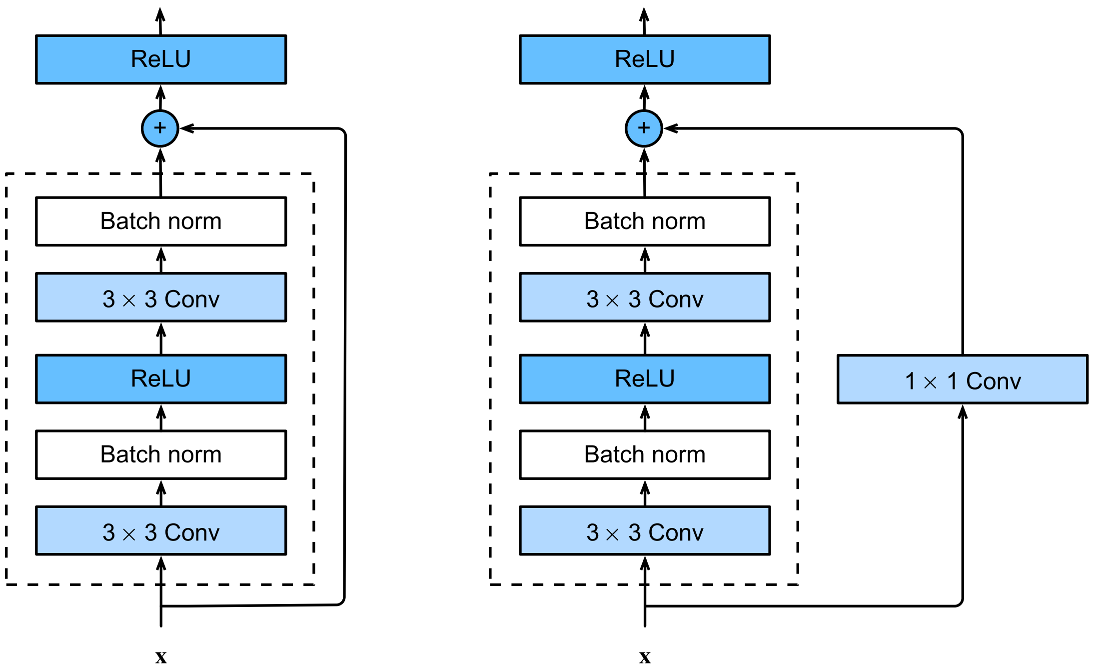
- ResNet (2015): Skip connections, ultra-deep networks
  - Residual block: Instead of mapping y := f(x), learn y := x + f(x)
    - Allows the network to learn residual functions
  - Allows training of very deep networks
    - ResNet-152 has 152 layers (vs. 19 in VGG-19)
  - Learn correction to previous features
    - Each block learns to correct the input rather than transform it completely
  - Family of models: ResNet-18, ResNet-34, ResNet-50, ResNet-101
    - Different depths for different computational budgets
  - https://arxiv.org/abs/1512.03385
  - https://arxiv.org/abs/1603.05027

The Deep Learning Revolution

- ResNet (2015): Skip connections, ultra-deep networks



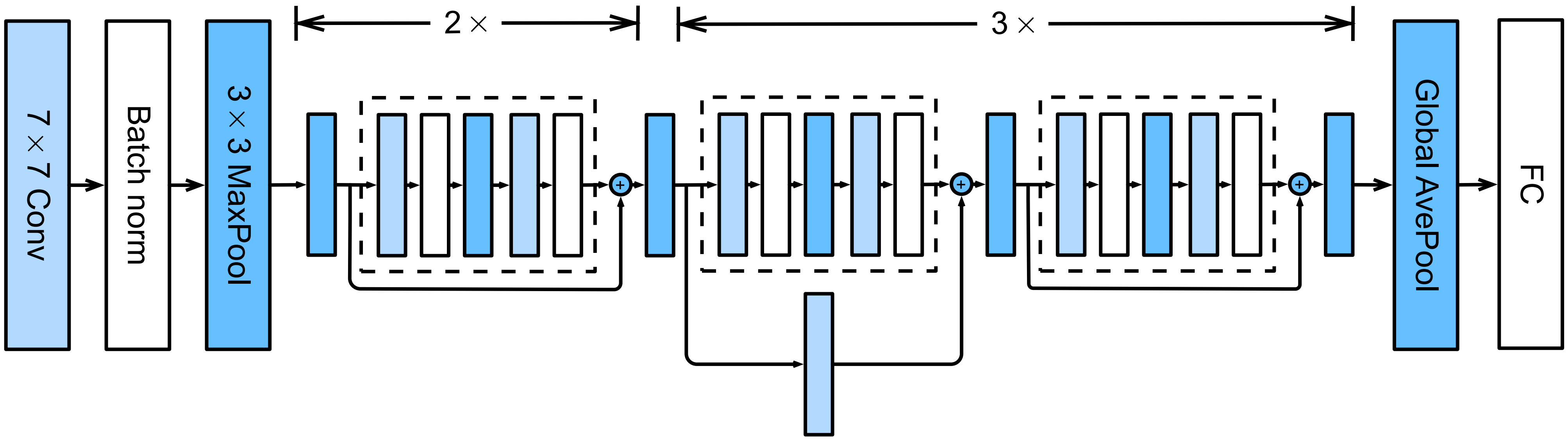
https://d2l.ai/chapter_convolutional-modern/resnet.html



https://d2l.ai/chapter_convolutional-modern/resnet.html

The Deep Learning Revolution

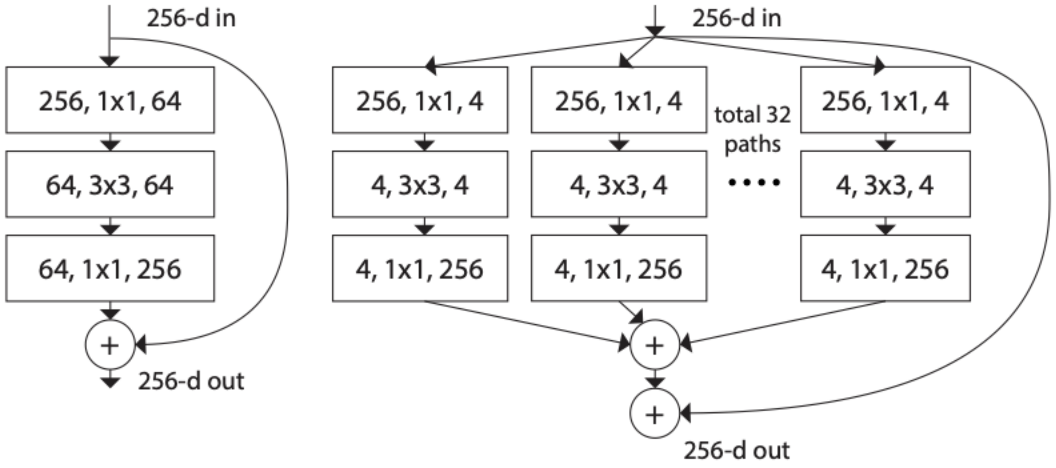
- ResNet (2015): ResNet-18



https://d2l.ai/chapter_convolutional-modern/resnet.html

The Deep Learning Revolution

- Variants of ResNet
  - ResNext: Information flows through several groups then aggregated
    - Combines ResNet with grouped convolutions (like inception block in GoogLeNet)
    - https://arxiv.org/abs/1611.05431



https://arxiv.org/abs/1611.05431

The Deep Learning Revolution

- Variants of ResNet
  - WideResNet: Wider blocks shown to be superior
    - Increases width instead of depth
    - https://arxiv.org/abs/1605.07146

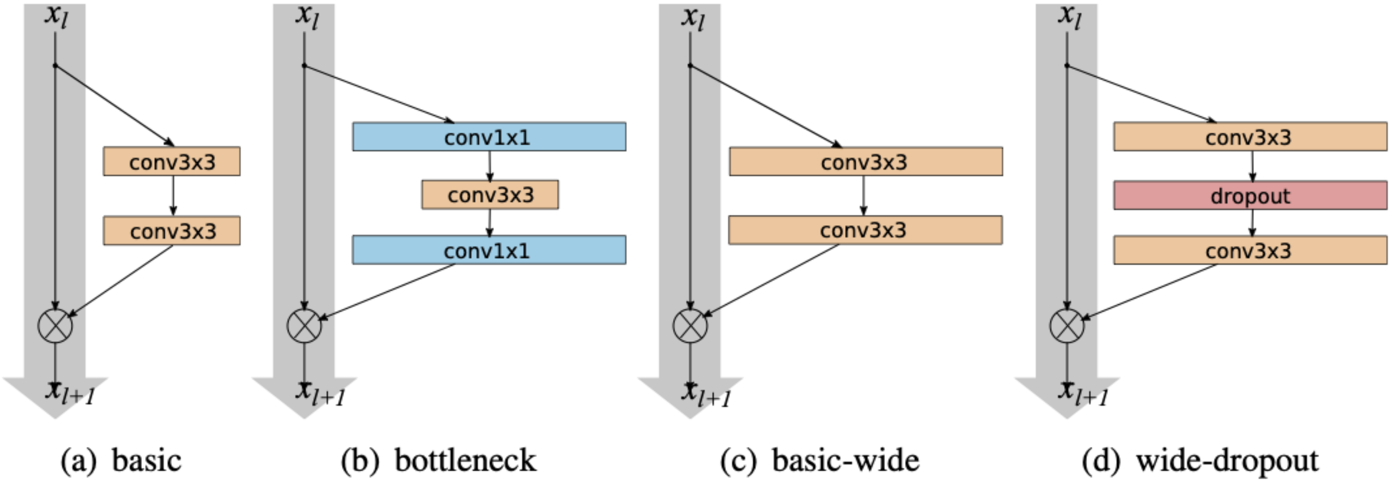


Figure 1: Various residual blocks used in the paper. Batch normalization and ReLU precede each convolution (omitted for clarity)

https://arxiv.org/abs/1605.07146

The Deep Learning Revolution

- DenseNet (2017): Dense connections between layers
  - Each layer is connected to all preceding layers
  - Improves gradient flow and feature reuse
  - https://arxiv.org/abs/1608.06993

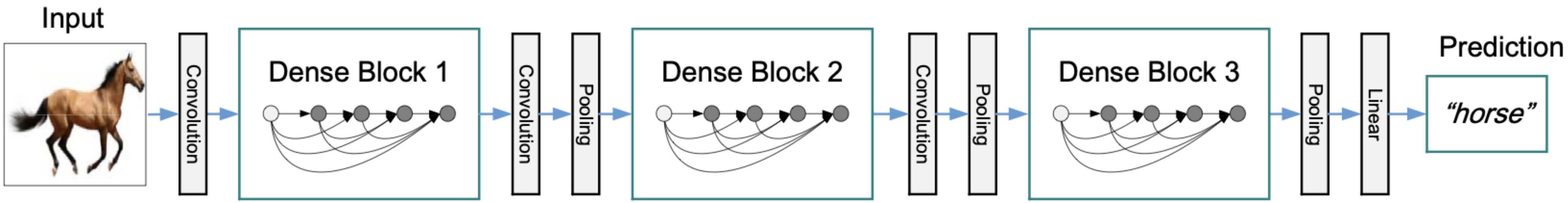


Figure 2: A deep DenseNet with three dense blocks. The layers between two adjacent blocks are referred to as transition layers and change feature-map sizes via convolution and pooling.

https://arxiv.org/abs/1608.06993

The Deep Learning Revolution

- DenseNet (2017): Dense connections between layers

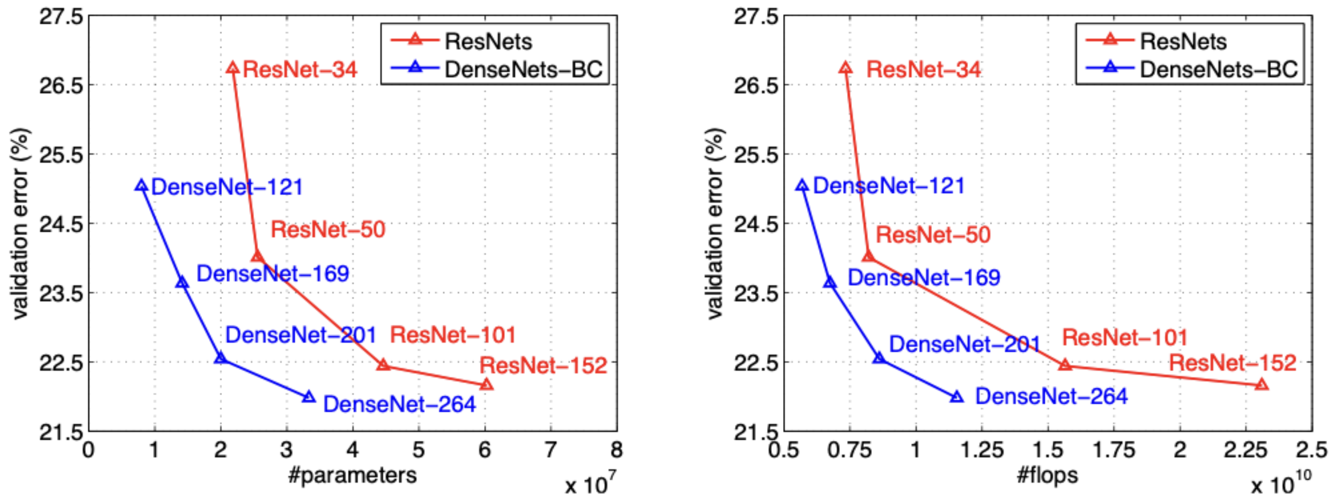


Figure 3: Comparison of the DenseNets and ResNets top-1 error rates (single-crop testing) on the ImageNet validation dataset as a function of learned parameters (*left*) and FLOPs during test-time (*right*).

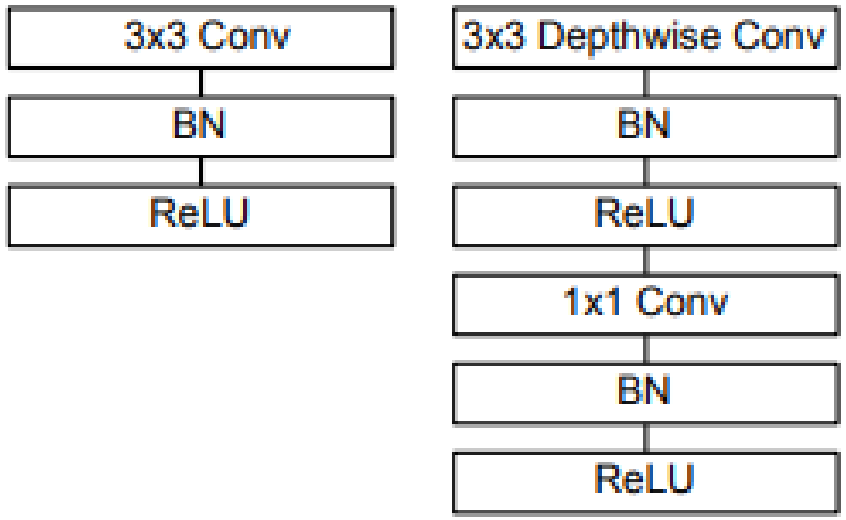
https://arxiv.org/abs/1608.06993

The Deep Learning Revolution

- MobileNet (2017): Efficient CNNs for mobile devices
  - Uses 1×1 convolutions plus depthwise separable convolutions
  - Based on Canonical decomposition of a convolution kernel:
    - $V(x,y,t) = \sum_{i=x-\delta}^{x+\delta} \sum_{j=y-\delta}^{y+\delta} \sum_{s=1}^{S} K(i-x+\delta, j-y+\delta, s, t) U(i,j,s)$
    - $K(i,j,s,t) = \sum_{r=1}^{R} K_x(i,r) K_y(j,r) K_s(s,r) K_t(t,r)$
    - Converts many-dimensional kernel into several one-dimensional kernels
    - Depthwise convolutions treat each input channel independently
    - 1x1 convolution treat all channels of a single pixel
  - No pooling blocks
    - convolution with stride=2 used for downsampling
  - https://arxiv.org/abs/1704.04861

The Deep Learning Revolution

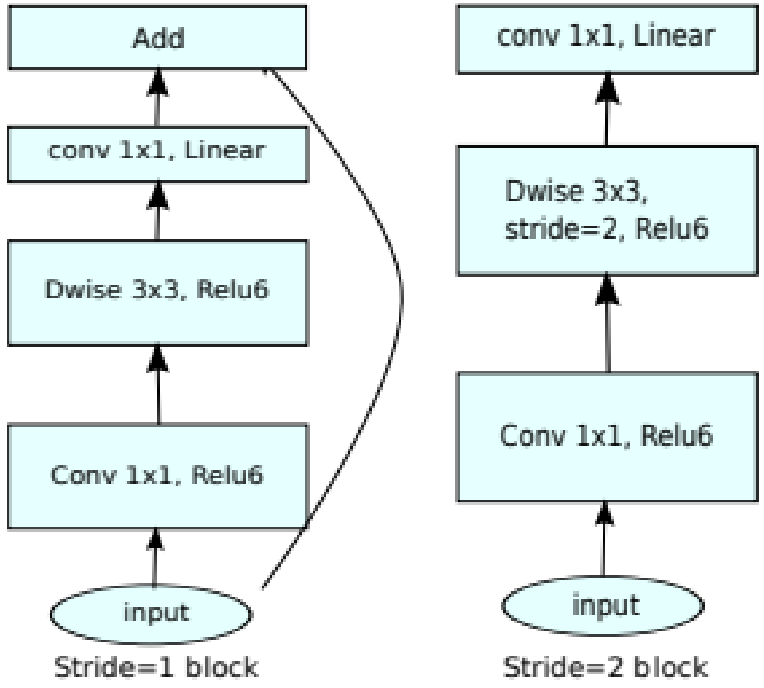
- MobileNet (2017): Efficient CNNs for mobile devices
  - Uses 1×1 convolutions plus depthwise separable convolutions
  - Based on Canonical decomposition of a convolution kernel:
  - No pooling blocks
  - https://arxiv.org/abs/1704.04861



https://raw.githubusercontent.com/oseledets/dl2024/8d739d06892c9cc71f198c1c478d51b4a6801866/lectures/lecture-2/mobilenet-block.png

The Deep Learning Revolution

- MobileNetV2 (2018): Inverted bottleneck design
  - First 1×1 convolution increases channels, last decreases
    - Expands channels in the middle of the block
  - Some blocks have residual connections, others do not
    - Combines ideas from ResNet and MobileNet
  - https://arxiv.org/abs/1801.04381



https://raw.githubusercontent.com/oseledets/dl2024/8d739d06892c9cc71f198c1c478d51b4a6801866/lectures/lecture-2/mobilenetV2.png

The Deep Learning Revolution

Network Architecture	Number of Parameters	Top-1 Accuracy	Top-5 Accuracy
Xception	22.91M	0.790	0.945
VGG16	138.35M	0.715	0.901
MobileNetV1 (alpha=1, rho=1)	4.20M	0.709	0.899
MobileNetV1 (alpha=0.75, rho=0.85)	2.59M	0.672	0.873
MobileNetV1 (alpha=0.25, rho=0.57)	0.47M	0.415	0.663
MobileNetV2 (alpha=1.4, rho=1)	6.06M	0.750	0.925
MobileNetV2 (alpha=1, rho=1)	3.47M	0.718	0.910
MobileNetV2 (alpha=0.35, rho=0.43)	1.66M	0.455	0.704

https://github.com/oseledets/dl2024/blob/main/lectures/lecture-2/lecture-2.ipynb

The Deep Learning Revolution

- EfficientNet (2019): Balanced scaling of network dimensions
  - All CNNs are similar and contain several stages
  - Scaling only a single parameter quickly saturates after 80% accuracy

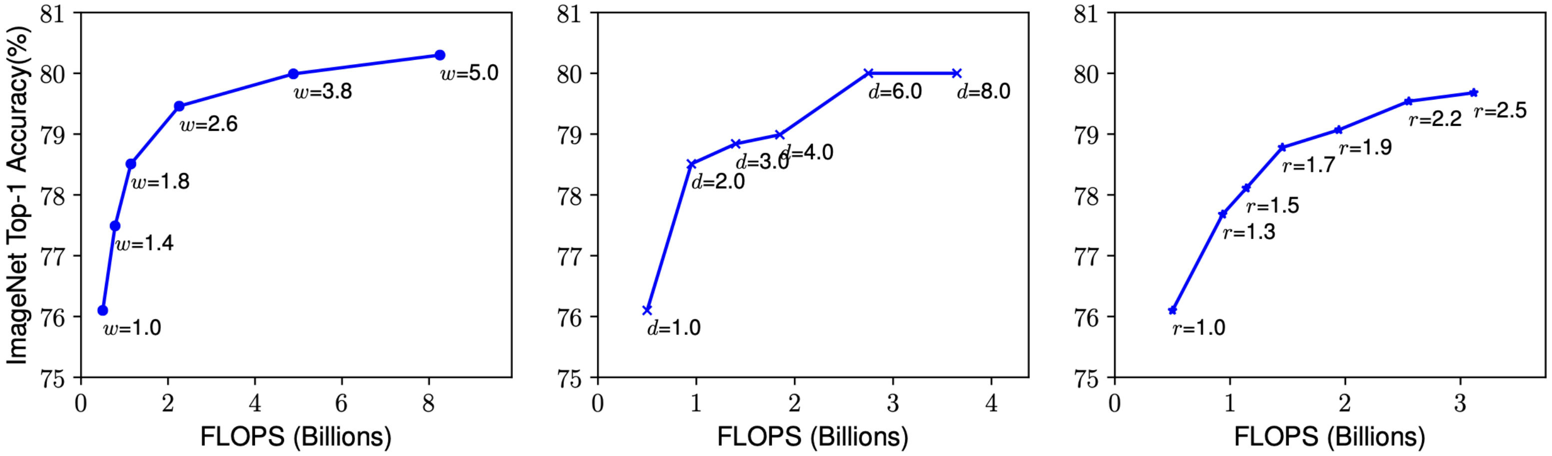
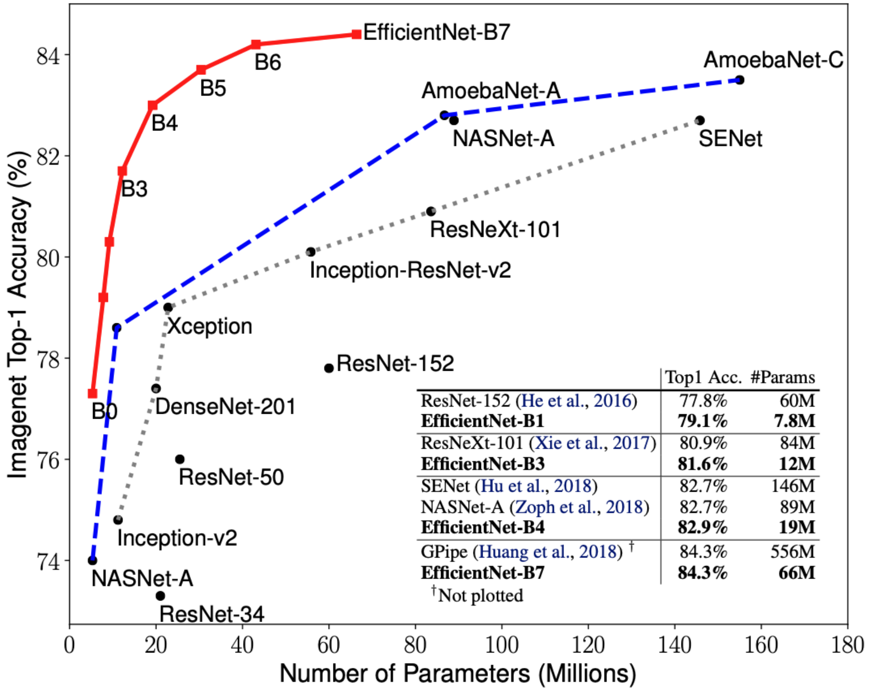


Figure 3. **Scaling Up a Baseline Model with Different Network Width (w), Depth (d), and Resolution (r) Coefficients.** Bigger networks with larger width, depth, or resolution tend to achieve higher accuracy, but the accuracy gain quickly saturate after reaching 80%, demonstrating the limitation of single dimension scaling. Baseline network is described in Table 1.

https://arxiv.org/abs/1905.11946

The Deep Learning Revolution

- EfficientNet (2019): Balanced scaling of network dimensions
  - Compound scaling
    - Scales all three dimensions together
    - Depth $d = \alpha^{\phi}, \alpha \geq 1$
    - Width $w = \beta^{\phi}, \beta \geq 1$
    - Resolution $r = \gamma^{\phi}, \gamma \geq 1$
    - Such that $\alpha \cdot \beta^2 \cdot \gamma^2 \approx 2$
    - Defined by a grid search
  - https://arxiv.org/abs/1905.11946



	Top1 Acc.	#Params
ResNet-152 (He et al., 2016)	77.8%	60M
EfficientNet-B1	**79.1%**	**7.8M**
ResNeXt-101 (Xie et al., 2017)	80.9%	84M
EfficientNet-B3	**81.6%**	**12M**
SENet (Hu et al., 2018)	82.7%	146M
NASNet-A (Zoph et al., 2018)	82.7%	89M
EfficientNet-B4	**82.9%**	**19M**
GPipe (Huang et al., 2018) †	84.3%	556M
EfficientNet-B7	**84.3%**	**66M**

†Not plotted

Figure 1. **Model Size vs. ImageNet Accuracy.** All numbers are for single-crop, single-model. Our EfficientNets significantly outperform other ConvNets. In particular, EfficientNet-B7 achieves new state-of-the-art 84.3% top-1 accuracy but being 8.4x smaller and 6.1x faster than GPipe. EfficientNet-B1 is 7.6x smaller and 5.7x faster than ResNet-152. Details are in Table 2 and 4.

https://arxiv.org/abs/1905.11946

Transformers

- Vision Transformers (2020): Incorporating transformer architecture
  - Treats images as sequences of patches
  - Uses self-attention instead of convolutions
  - Outperforms CNNs for many tasks
  - https://arxiv.org/abs/2010.11929