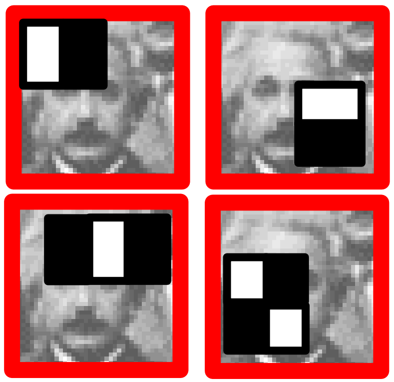
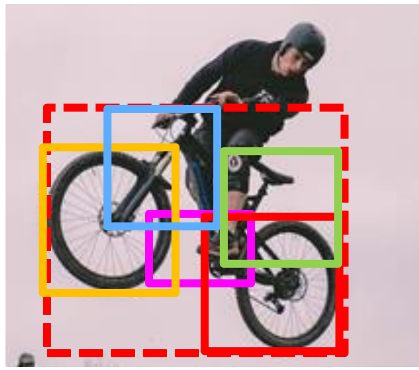
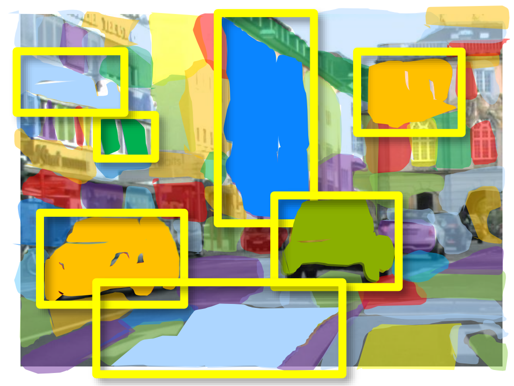
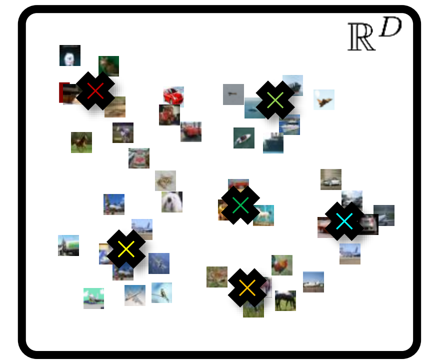
Photogrammetry & Robotics Lab

Machine Learning for Robotics and Computer Vision

Introduction to CNNs

Jens Behley

Last Lecture

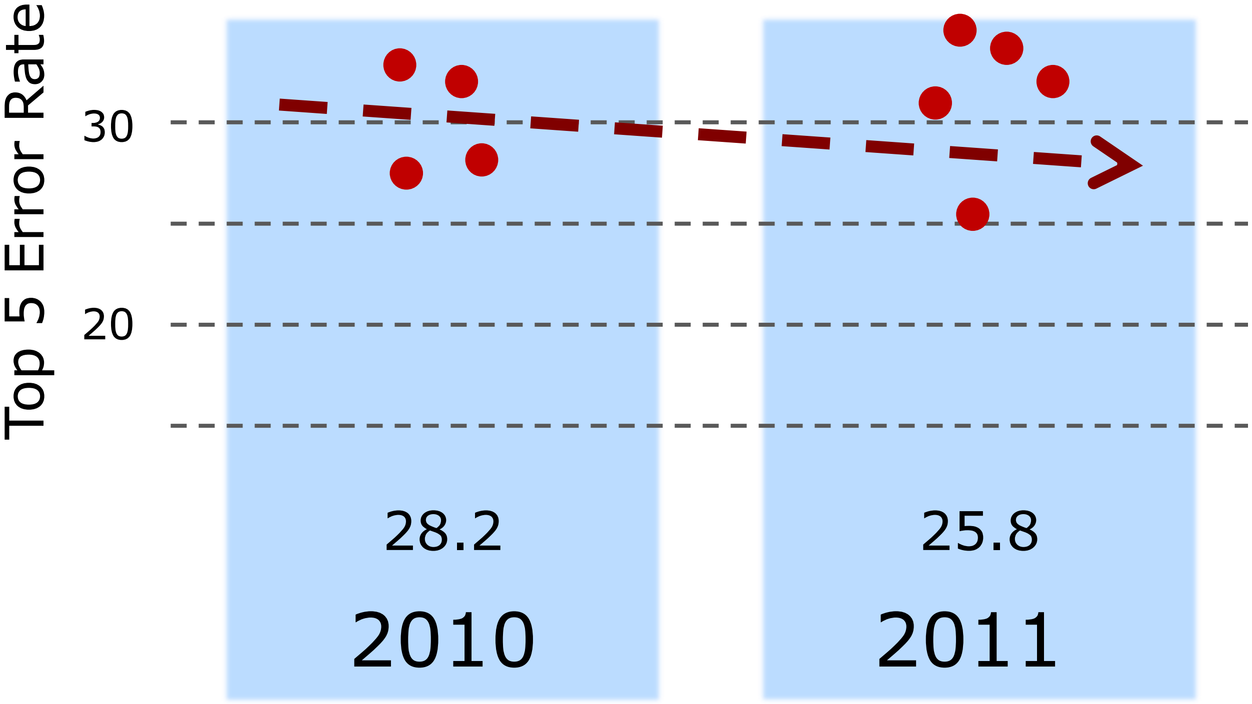





- We looked at a couple of applied ML approaches for object detection & image classification
- Designing better features is the main deal

Recap: Feature Engineering

- Applications to Computer Vision tasks: Extract features and apply supervised learning methods
- Most of the time: designing task-specific features → **feature engineering**

Progress on ImageNet



- Steady progress on ImageNet
- One outlier dramatically improved the error rate in 2012

Deep Learning Ara

MNIST Dataset

German Traffic Sign
Recognition Benchmark

- Convolution Neural Networks (CNN) show promising result on small datasets, like MNIST, CIFAR10, traffic sign classification
- Success on ImageNet showed the prospect of Convolution Neural Networks (CNN) for more complex vision tasks

Why are CNNs successful now?

- Several reasons made progress possible:
  1. Availability of large-scale data (ImageNet, etc.)
  2. Availability of compute capabilities (GPUs)
  3. **Availability of code (and frameworks)!**

- Implementation for most paper available
- Many frameworks made it simple to build and train networks (Caffe, Theano, Torch, etc.)

Deep Learning Frameworks

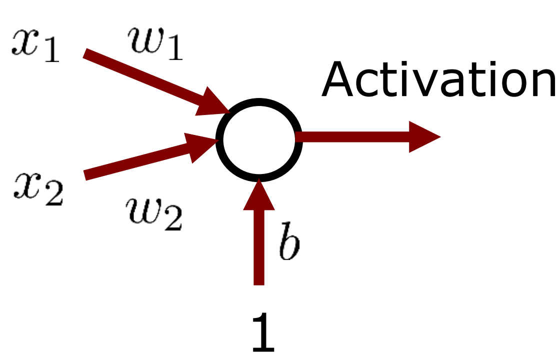
- All operations must be implemented using GPU to get maximum performance
- DL Frameworks available implementing the operations needed (and many more)

TensorFlow

PyTorch

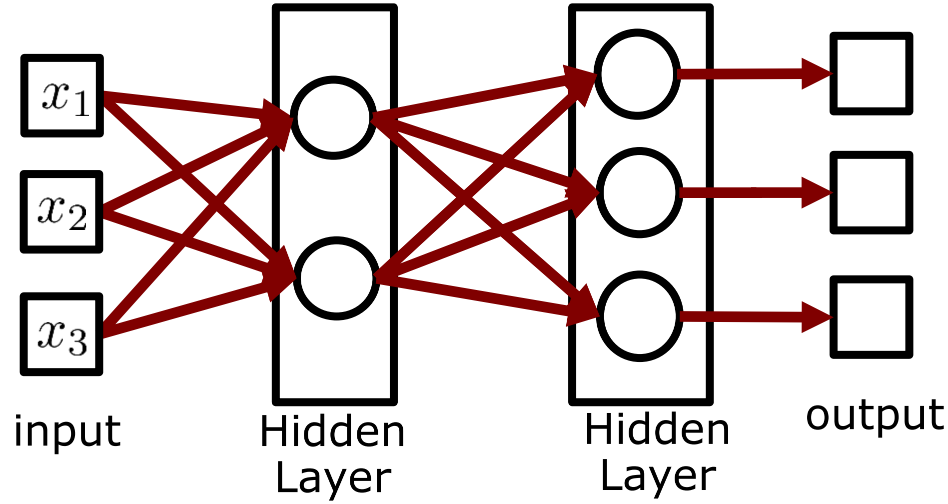
Neural Network

Neuron



$$f(\mathbf{x}) = \sum_i w_i x_i + b$$

2-layer Neural Network



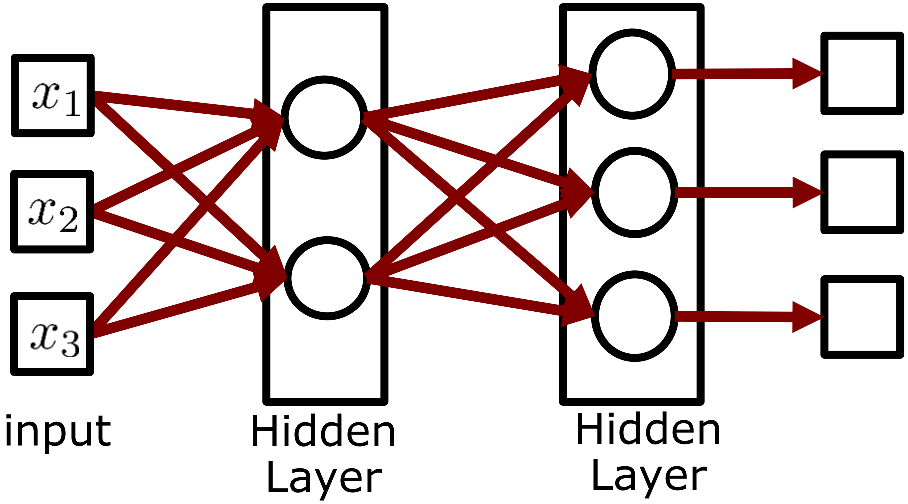
$$f(\mathbf{x}) = \mathbf{W}_2 \sigma(\mathbf{W}_1 \mathbf{x} + b_1) + b_2$$

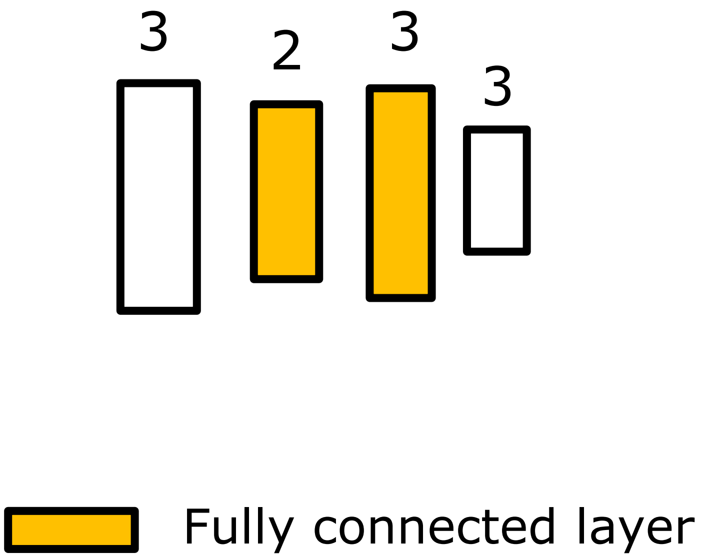
- Composition of functions with non-linear activation function $\sigma(\cdot)$ in between
- Each neuron takes all inputs into account → **fully-connected** layer

(Loose) biological inspiration

- Artificial neurons are inspired by biological neurons in the brain that transmit information
- Therefore, neural nets are often **wrongly** attributed as “artificial” brains
- **Beware:**
  - Current neural nets are much, much simpler then the brain
  - Real neurons and activations are much more complicated!
  - Neural nets can do amazing things, but this is not intelligence

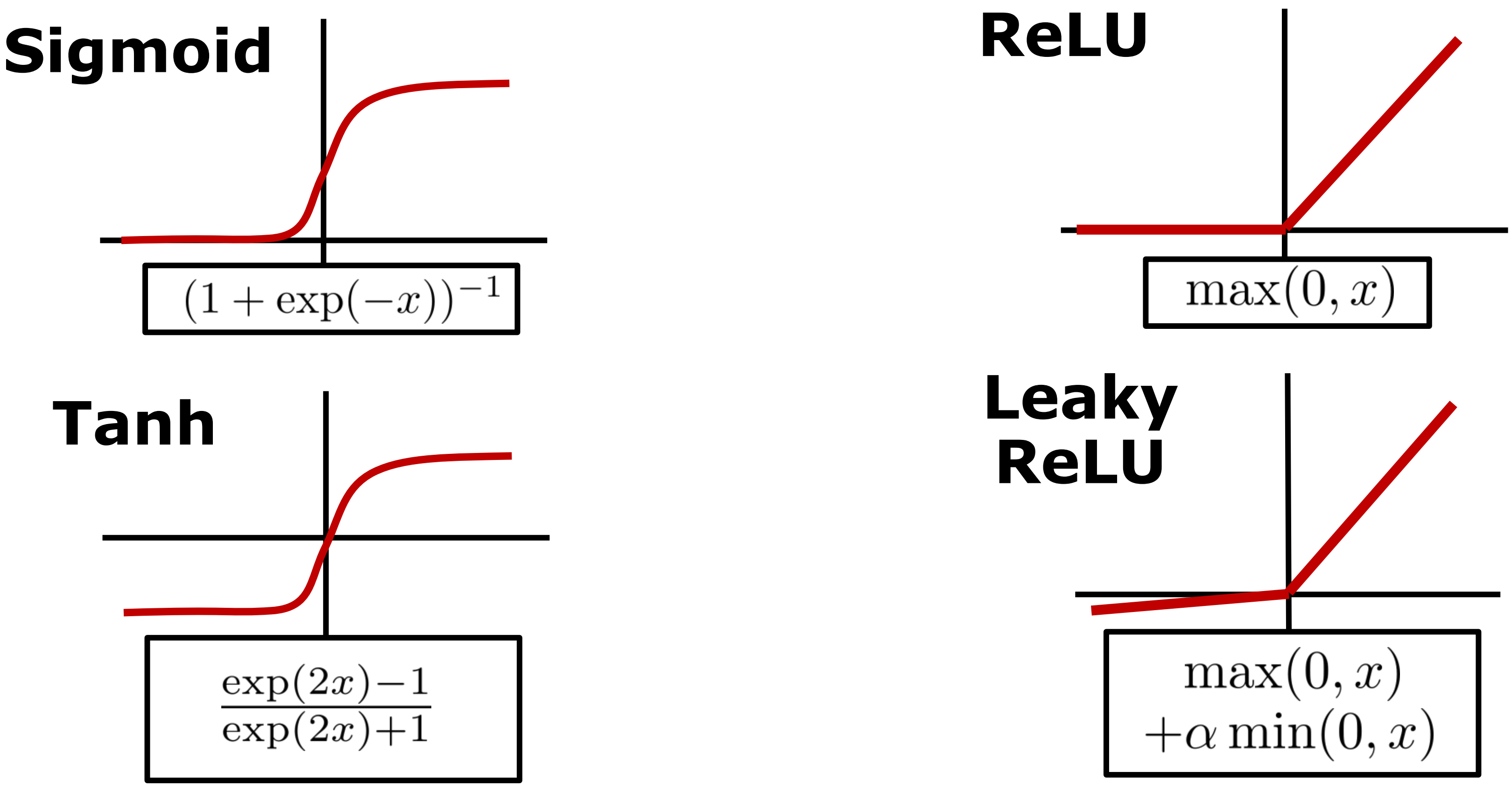
Alternative Visualization





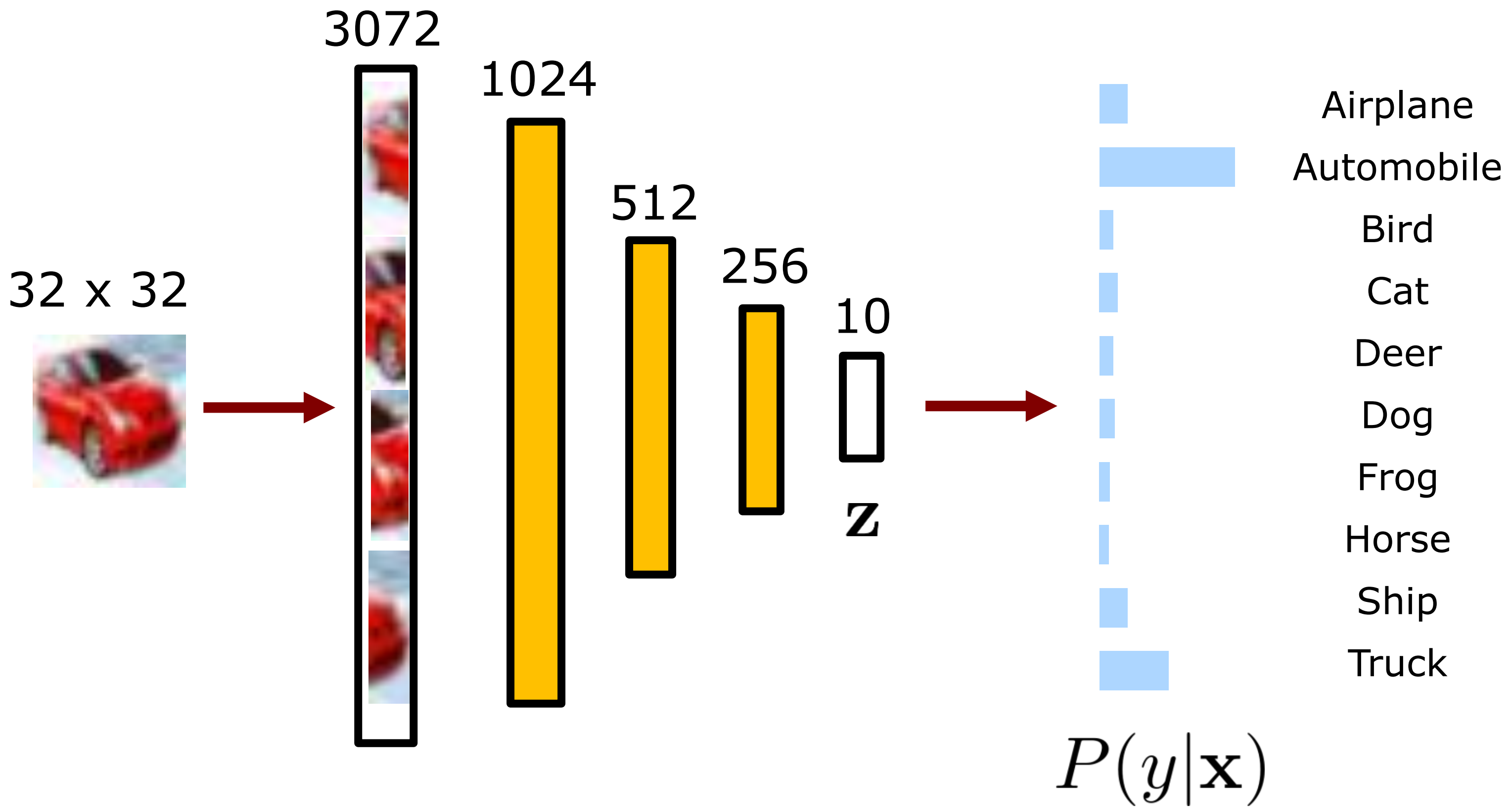
- Common way of visualization: show just stack of layers with output dimensions
- **Multi Layer Perceptron**

Activation function



- Common activation functions
- **Rectified Linear Units (ReLU)** good default for most problems

Example: Image Classification



- Output of neural network are scores for softmax resulting in $P(y|\mathbf{x})$
- **Output:** $\mathbf{z} \in \mathbb{R}^C$ with C number of classes

Image from CIFAR10

Fully-connected Layer: Summary

- **Input size:** N
- **Output size:** M
- **Parameters:** MN + M biases

LINEAR

Fully connected layer in PyTorch

CLASS `torch.nn.Linear(in_features, out_features, bias=True)` [SOURCE]

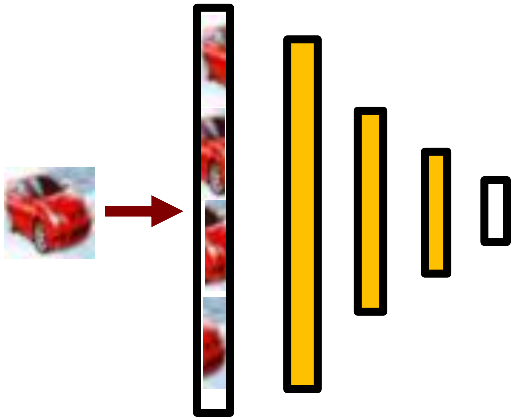
Applies a linear transformation to the incoming data: $y = xA^T + b$

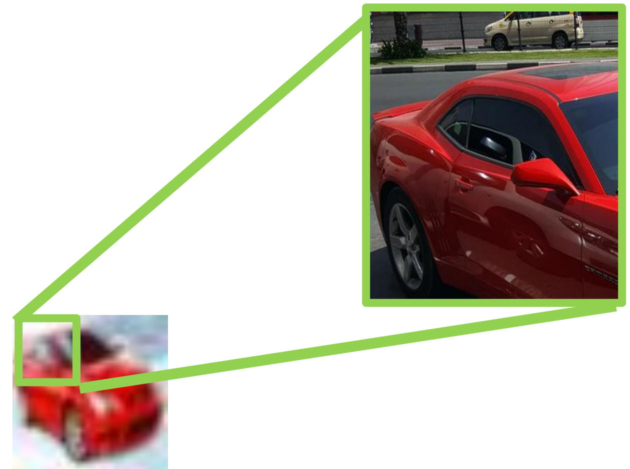
This module supports TensorFloat32.

Parameters

- **in_features** – size of each input sample
- **out_features** – size of each output sample
- **bias** – If set to `False`, the layer will not learn an additive bias. Default: `True`

Structure of Images

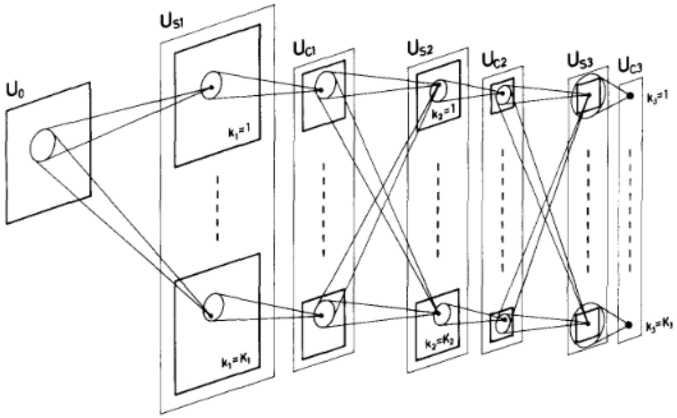


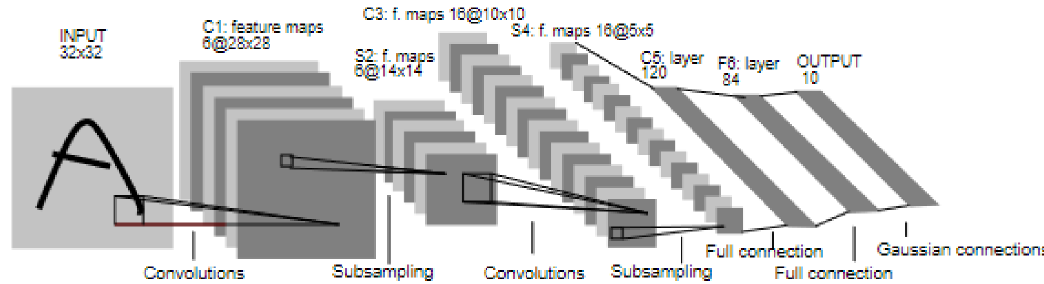


- Converting images into vector removes neighborhood structure
- Network should use this **inductive bias** that pixel neighborhood is important
- → **Convolutional Neural Network (CNN)**

Image from CIFAR10, Photo by Syed Hussaini on Unsplash

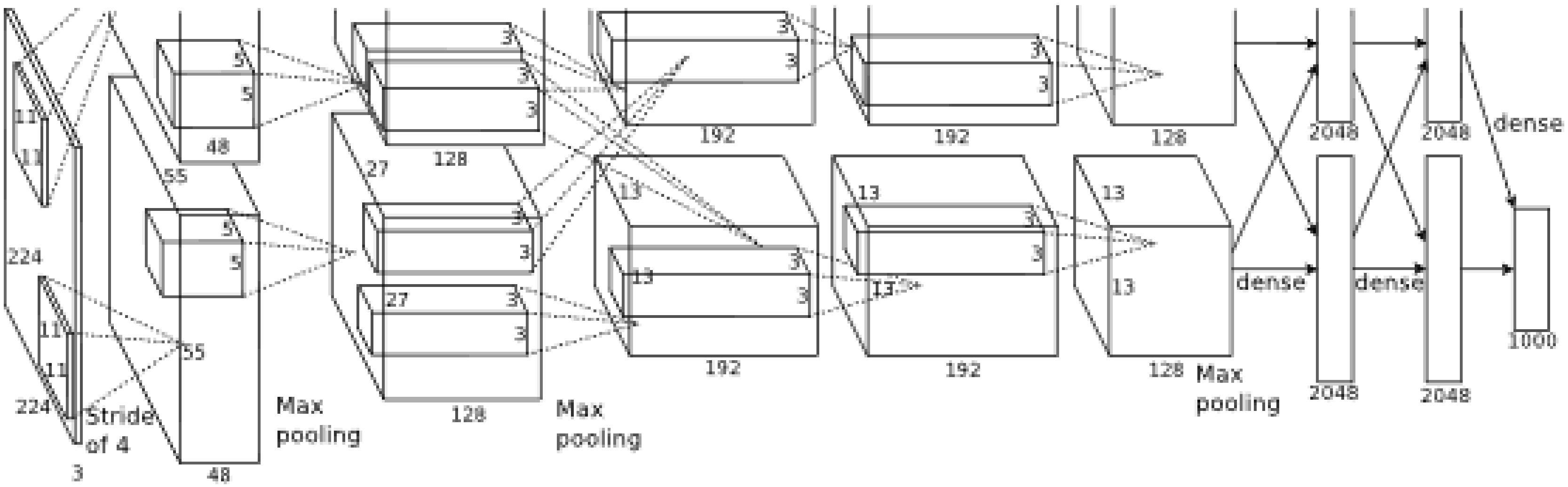
A bit of history of CNNs





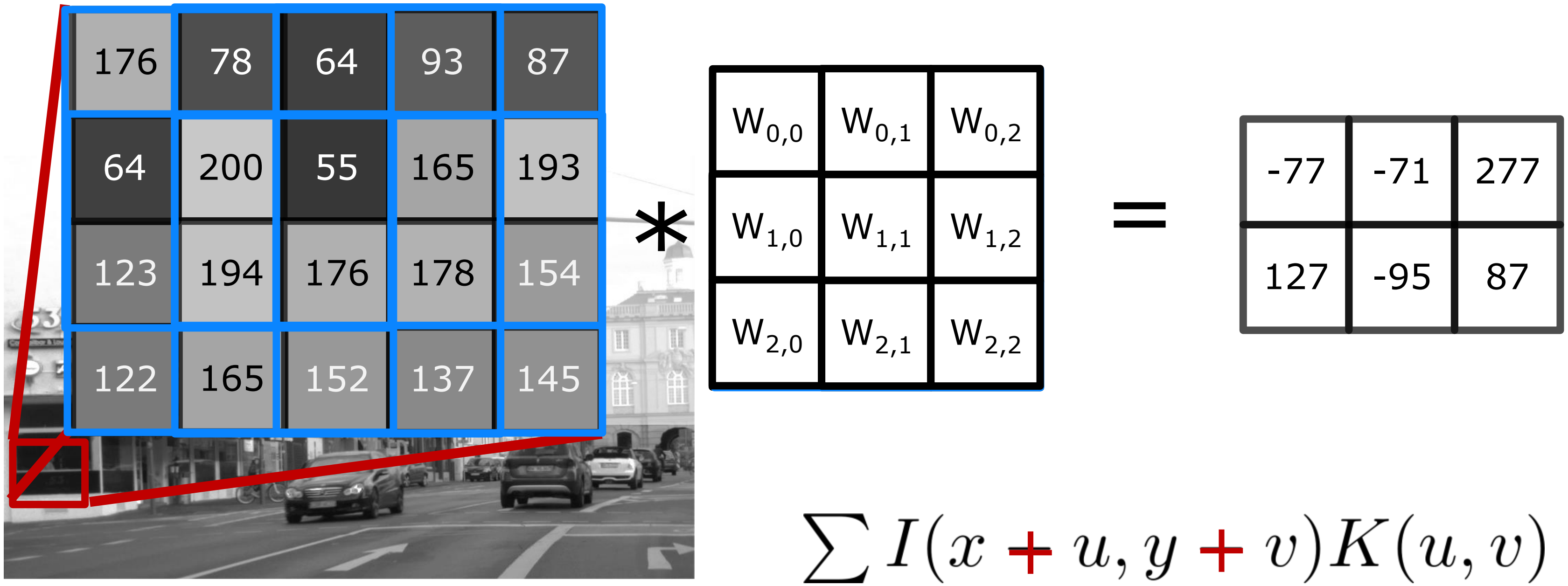
- 1980: Neocognitron (Fukushima)
- 1986: Backpropagation (Rumelhart, Hinton & Williams): Practical way to train a neural network and compute gradients
- 1998: CNN for handwritten digits (LeCun), commercially used for handwritten checks

Building block of CNNs



- Convolutional networks are build from
  - Convolutional Layers
  - Max Pooling operations
  - Fully-connected Layers

Convolution Neural Network



$$\sum I(x+u, y+v)K(u,v)$$

- Convolution “slides” kernel/filter K over image I
- Trivia: Most DL frameworks use cross-correlation instead

Translation Equivariance

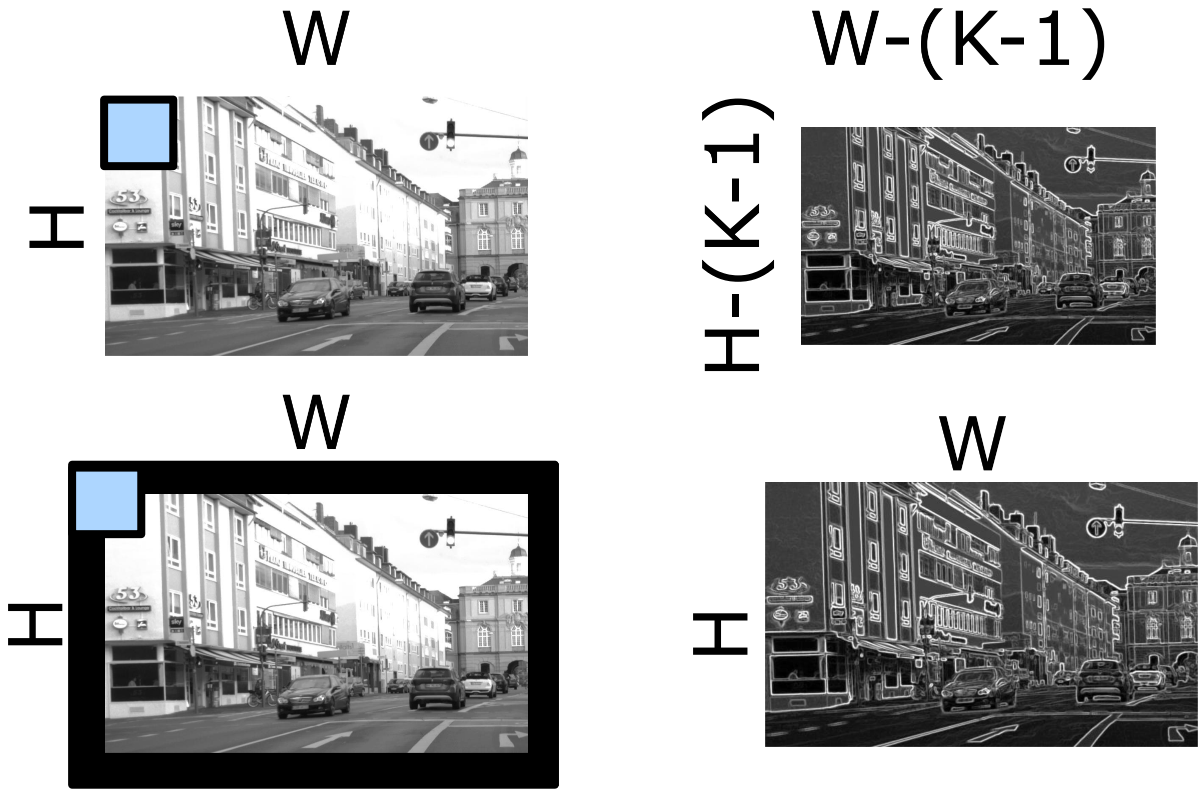
- Same kernel applied everywhere
- Translation of image will translates feature map → **equivariance** of convolution

Convolution Neural Network

Padding

No Padding

Zero Padding
P=(K-1)/2



- Convolution on valid location leads to reduced size of feature map
- Padding adds border values

Convolution Neural Network

Stride

- Stride specifies spacing between evaluation of kernels
- Stride > 1 reduces size of feature map

Convolution Neural Network

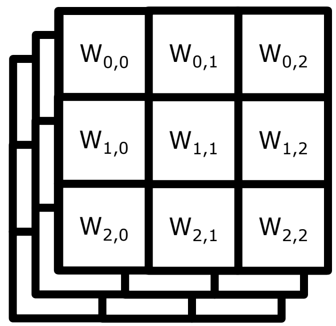
Dilation

- Dilation specifies spacing between entries of the kernel
- Stride > 1 reduces size of feature map

Convolution Neural Network
Multiple Input Channels

W

H

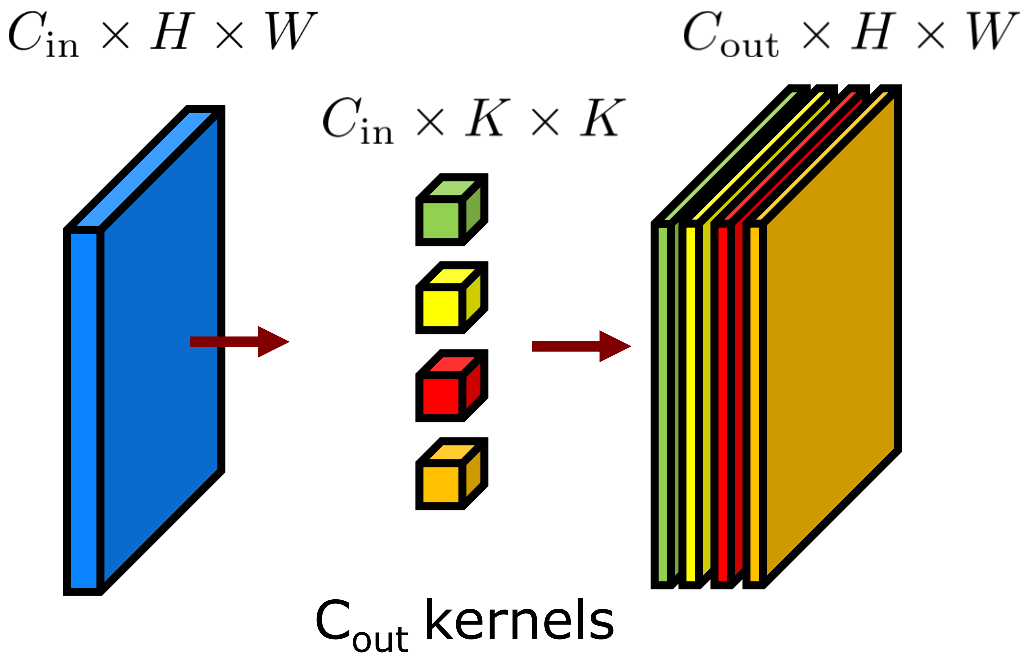


$\mathbf{3} \times H \times W$ $\mathbf{3} \times 3 \times 3$ $\mathbf{1} \times H \times W$

(with zero padding)

- For multi-channel input convolutional filter has also as many channels
- Produces still one activation map

Convolutional Layer



- Use multiple kernels to produce C_{out} maps
- Non-linear activation function applied element-wise on convolution results

Convolutional Layer: Summary

Hyperparameter:

- K kernel size
- P padding
- S stride
- D dilation

- **Input size:** $C_{\text{in}} \times H \times W$
- **Output size:** $C_{\text{out}} \times H' \times W'$

$$H' = \left\lfloor \frac{H+2P-D(K-1)-1}{S} + 1 \right\rfloor \quad W' = \left\lfloor \frac{W+2P-D(K-1)-1}{S} + 1 \right\rfloor$$

- **Parameters:** $C_{\text{out}} C_{\text{in}} K^2$ + C_{out} biases

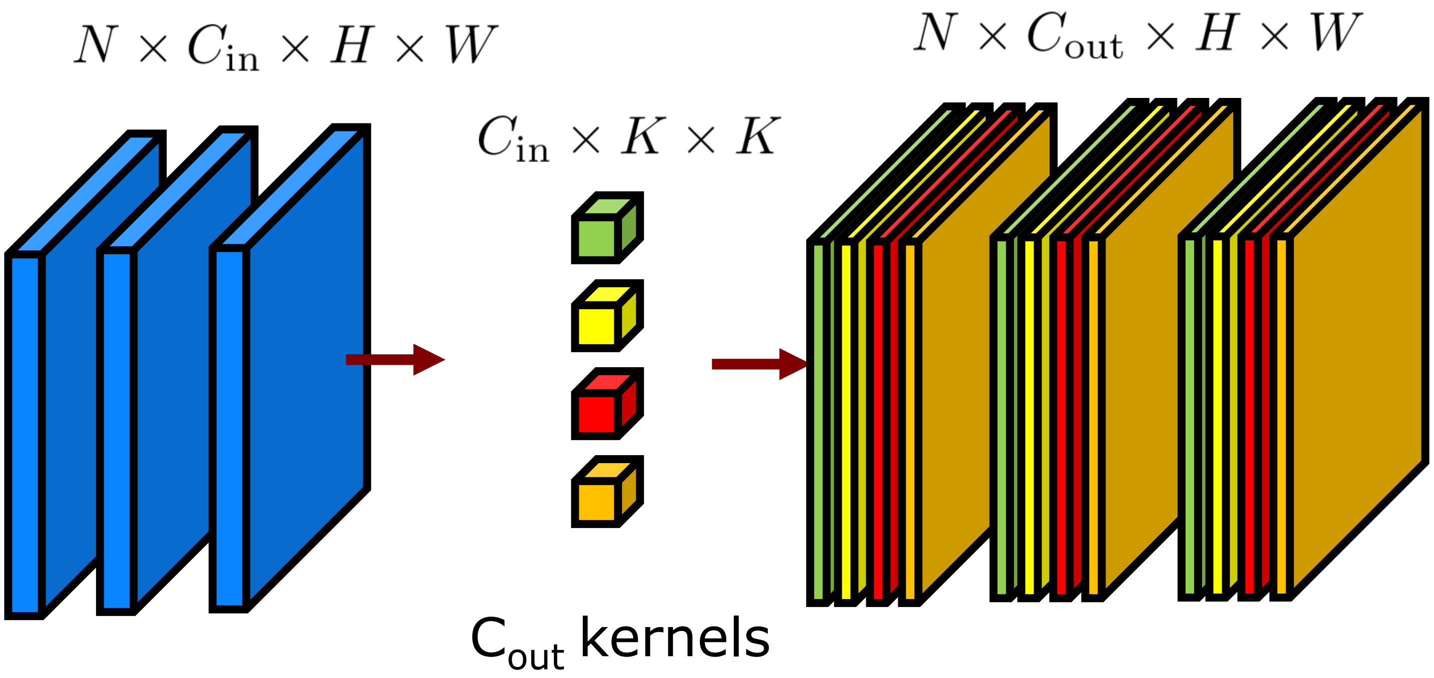
- **Common:**
  - K = **1**, **3**, 5, 7
  - P = (K-1)/2 → input = output size
  - S = 1, 2

Channel-first vs. Channel-last

- Organizing tensors in different ways possible
- Main conventions:
  - Channel-first (PyTorch): $C \times H \times W$
  - Channel-last (Tensorflow): $H \times W \times C$

- Ensure that input images are channel-first before applying convolutions in PyTorch
- See `ToTensor()` in `torchvision.transform` that converts PIL image to a tensor

ConvLayer on Batch of Images



- Usually all operations are applied on a batch of images (multiple images)

ConvLayer in PyTorch

CONV2D

CLASS `torch.nn.Conv2d(in_channels, out_channels, kernel_size, stride=1, padding=0, dilation=1, groups=1, bias=True, padding_mode='zeros')` [SOURCE]

Applies a 2D convolution over an input signal composed of several input planes.

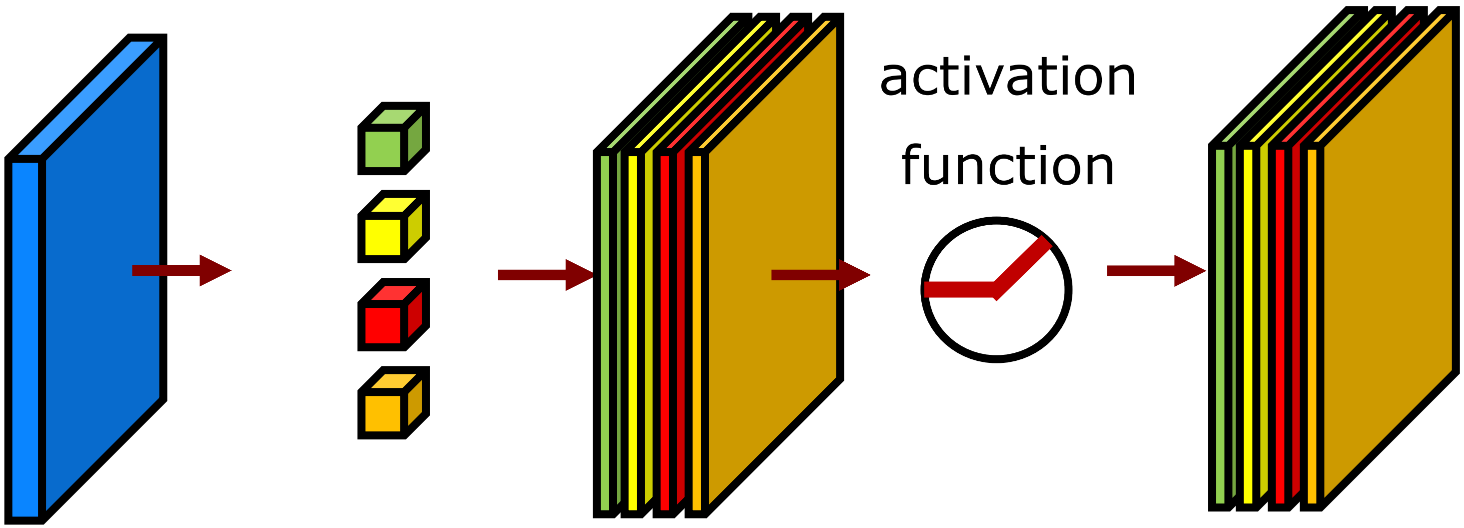
In the simplest case, the output value of the layer with input size (N, C_{in}, H, W) and output $(N, C_{\text{out}}, H_{\text{out}}, W_{\text{out}})$ can be precisely described as:

$$\text{out}(N_i, C_{\text{out}_j}) = \text{bias}(C_{\text{out}_j}) + \sum_{k=0}^{C_{\text{in}}-1} \text{weight}(C_{\text{out}_j}, k) \star \text{input}(N_i, k)$$

where $\star$ is the valid 2D cross-correlation operator, N is a batch size, C denotes a number of channels, H is a height of input planes in pixels, and W is width in pixels.

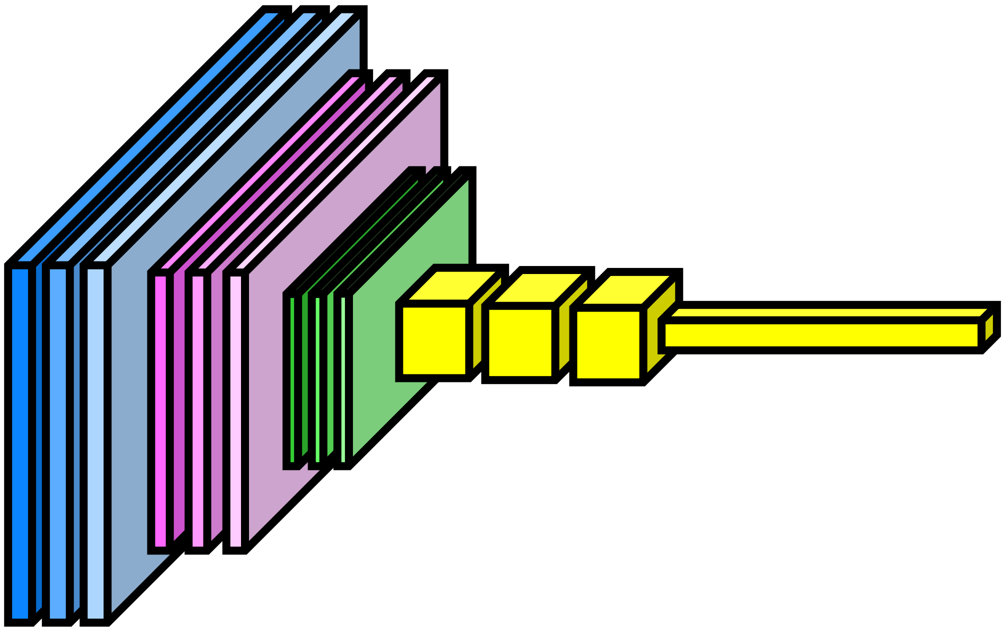
- Also Conv1d & Conv3d available
- LazyConv2d infers parameters from first forward pass

ConvLayer + Activation Function



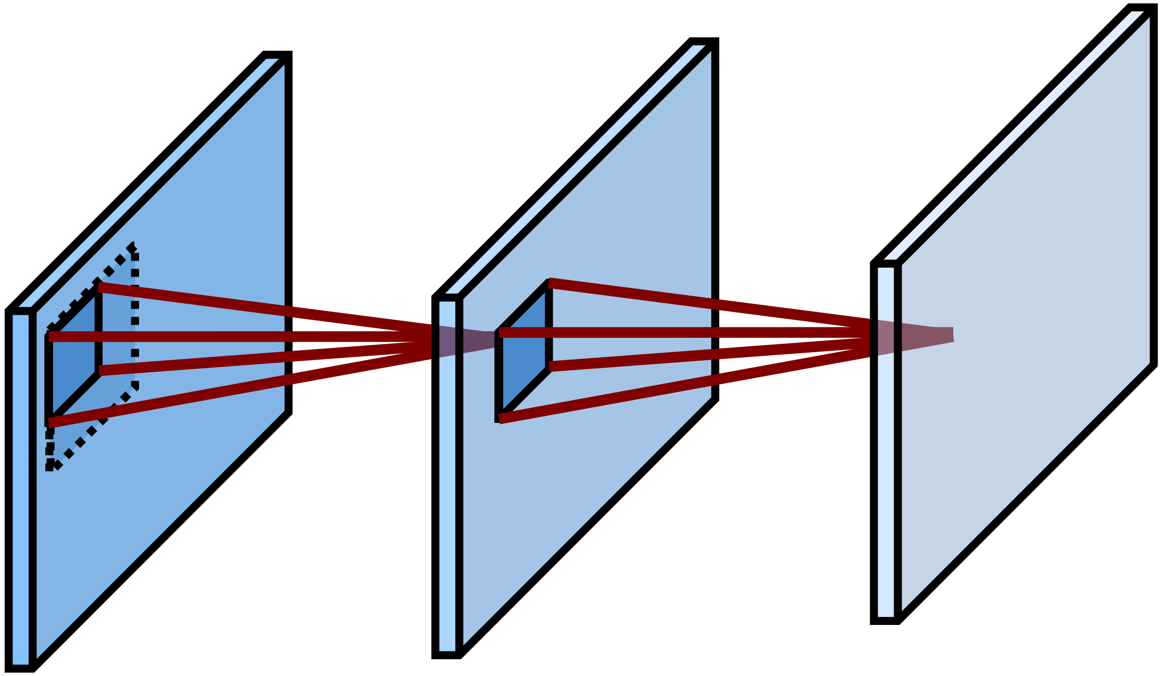
- Activation function (such as ReLU) applied after each convolutional layer
- Usually only implicit in the graphical representation

Convolution Neural Network



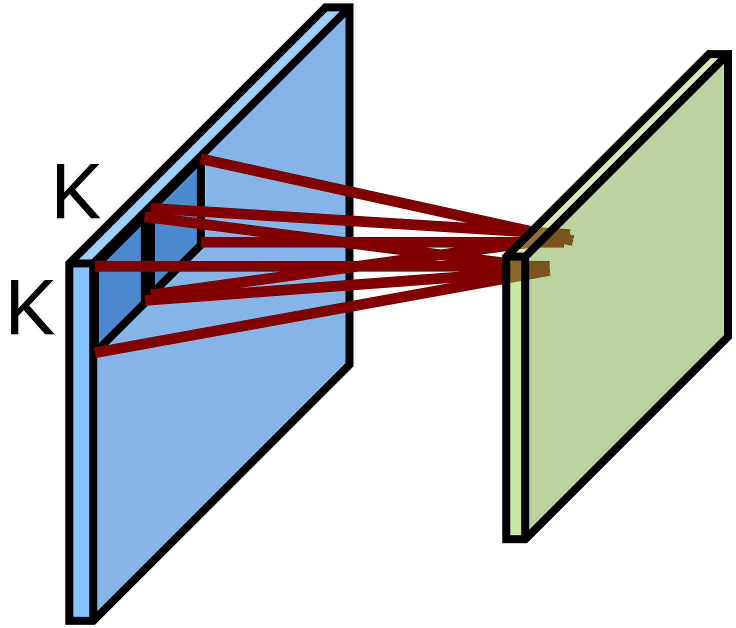
- Stack of convolutional layers
- Pooling layer to increase receptive field of layers and provide translation invariance

Receptive field



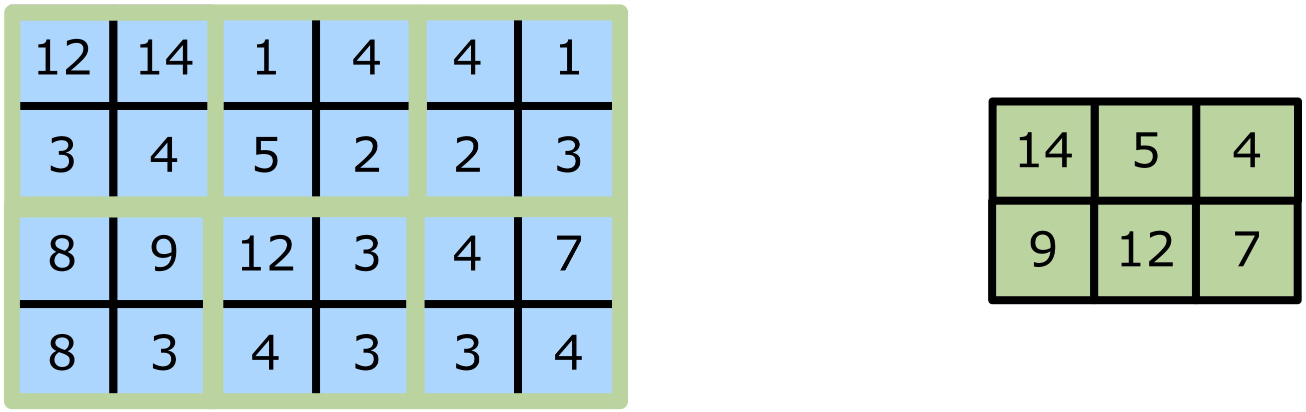
- Location in deeper layers take inputs of window of earlier layers
- Deeper layers "see" more from earlier layers

Pooling Layer



- Pooling layers increase the receptive field & aggregates information
- Translation invariance to small shifts
- Common: max pooling, average pooling

Example: Max Pooling



2 × 2 max pooling, stride 2

- Compute maximum in each region
- Common to have non-overlapping windows

Max Pooling in PyTorch

MAXPOOL2D

CLASS `torch.nn.MaxPool2d(kernel_size, stride=None, padding=0, dilation=1, return_indices=False, ceil_mode=False)` [SOURCE]

Applies a 2D max pooling over an input signal composed of several input planes.

In the simplest case, the output value of the layer with input size (N, C, H, W), output (N, C, H_{out}, W_{out}) and `kernel_size` (kH, kW) can be precisely described as:

$$out(N_i, C_j, h, w) = \max_{m=0,\ldots,kH-1} \max_{n=0,\ldots,kW-1} \text{input}(N_i, C_j, \text{stride}[0] \times h + m, \text{stride}[1] \times w + n)$$

If `padding` is non-zero, then the input is implicitly zero-padded on both sides for `padding` number of points. `dilation` controls the spacing between the kernel points. It is harder to describe, but this link has a nice visualization of what `dilation` does.

- AvgPool2d for average pooling

Pooling: Summary

Hyperparameter:

- K kernel size
- P padding
- S stride
- D dilation

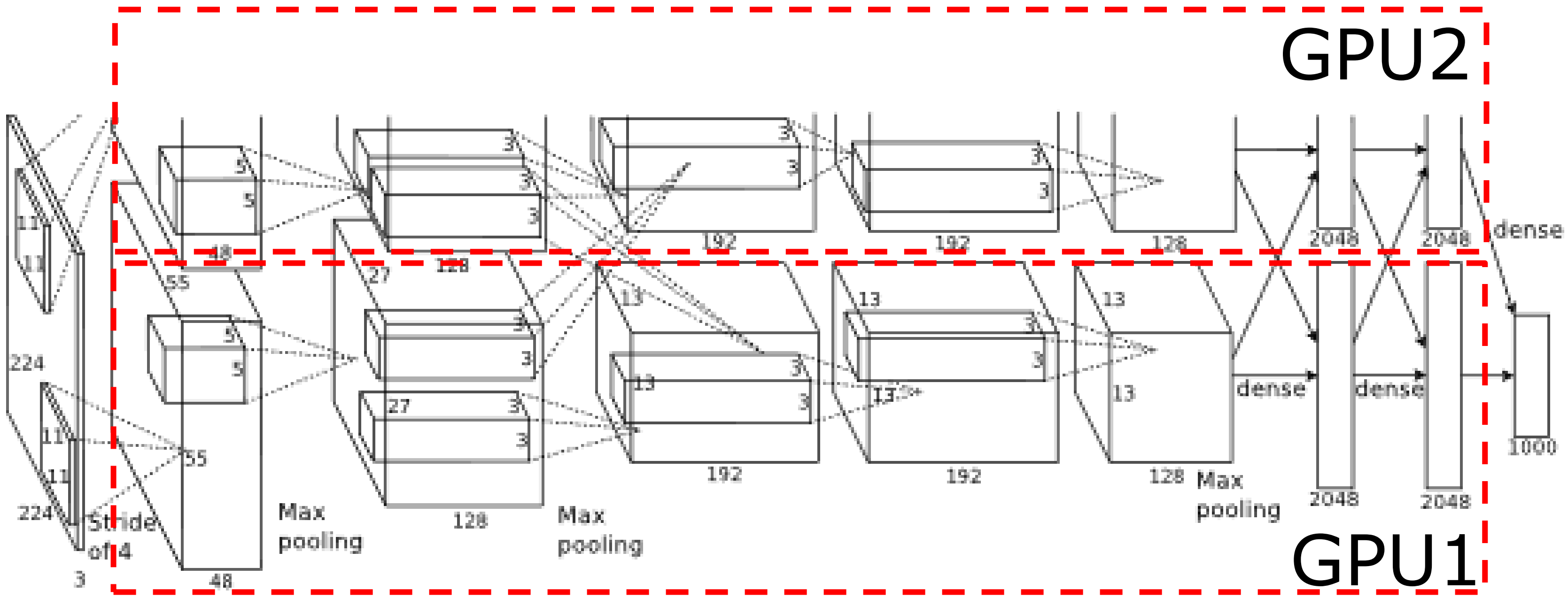
- **Input size:** $C_{\text{in}} \times H \times W$
- **Output size:** $C_{\text{out}} \times H' \times W'$

$$H' = \left\lfloor \frac{H+2P-D(K-1)-1}{S} + 1 \right\rfloor \quad W' = \left\lfloor \frac{W+2P-D(K-1)-1}{S} + 1 \right\rfloor$$

- **Parameters: 0**

- **Common:**
  - K=2, S=2 (non-overlapping)
  - K=3, S=2 (overlapping) in AlexNet

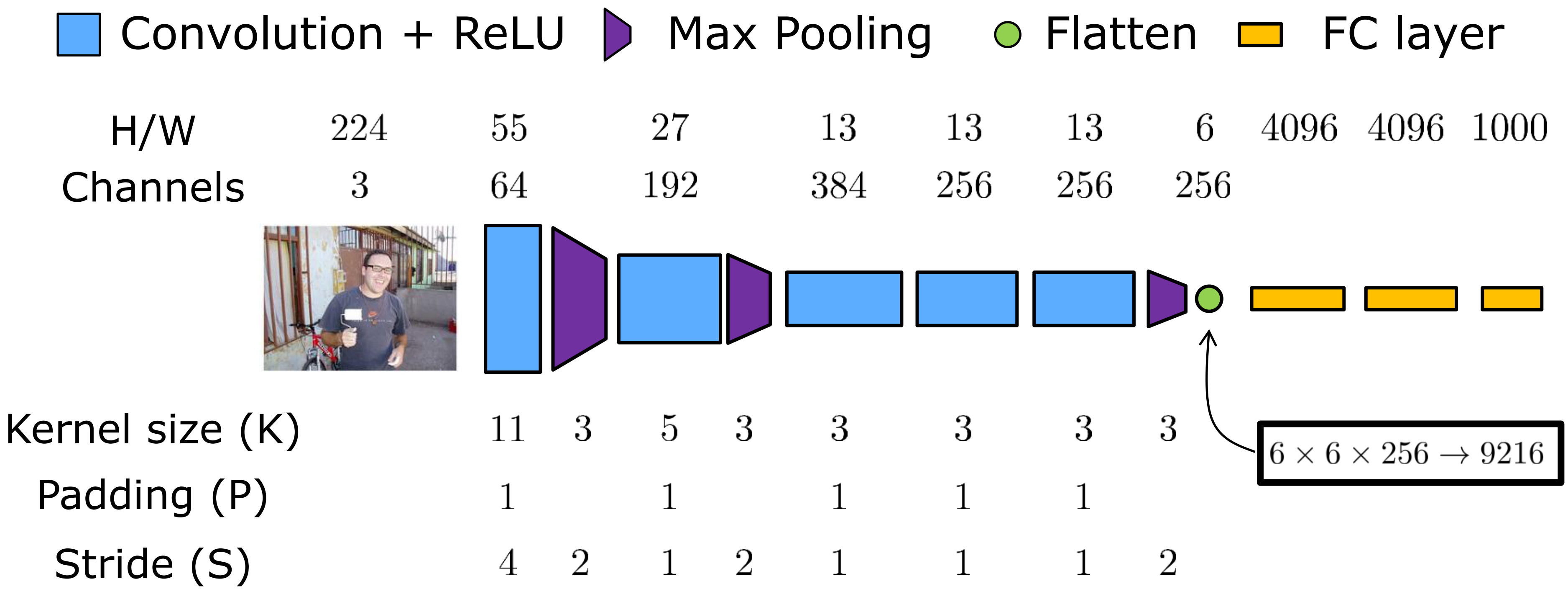
AlexNet Structure



- 5 convolutions, 3 max pooling, 3 fully-connected layers
- Split across 2 GPUs due to memory constraints

[Krizhevsky, 2012]

AlexNet Structure*

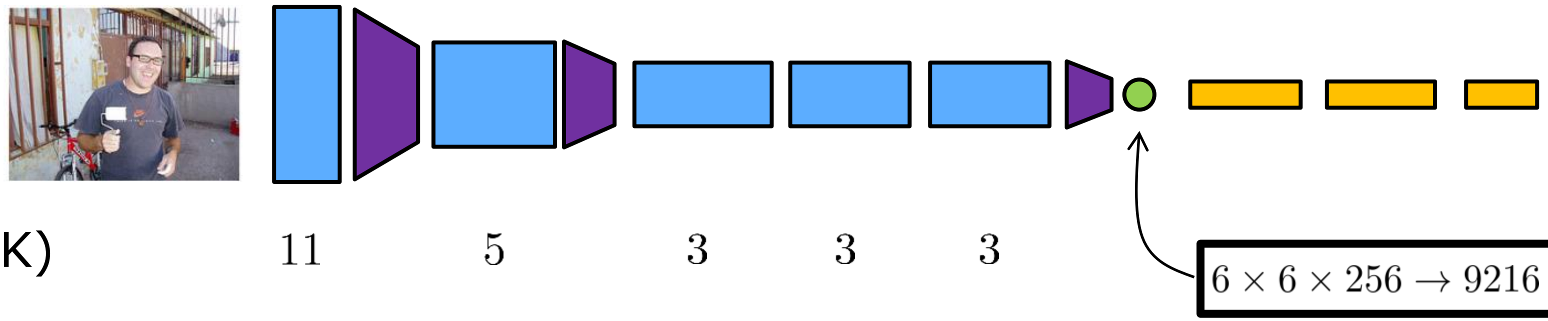


- Here, single network version (similar to PyTorch version of AlexNet)
- Overall, 57 Million parameters (but where?)

* [Krizhevsky, 2014]

Number of Parameters

H/W	224	55	27	13	13	13	6	4096	4096	1000
Channels	3	64	192	384	256	256	256			
Kernel size (K)		11	5	3	3	3				



Conv1: 64*3*11*11 + 64 = 23,296

Conv2: 192*64*5*5 + 192 = 307,392

Conv3: 384*192*3*3 + 384 = 663,936

Conv4: 256*384*3*3 + 256 = 884,992

Conv5: 256*256*3*3 + 256 = 590,080

FC1: 9,216*4,096 + 4,096 = **37,752,832**

FC2: 4,096*4,096 + 4,096 = 16,777,216

FC3: 4,096*1,000 + 1,000 = 4,097,000

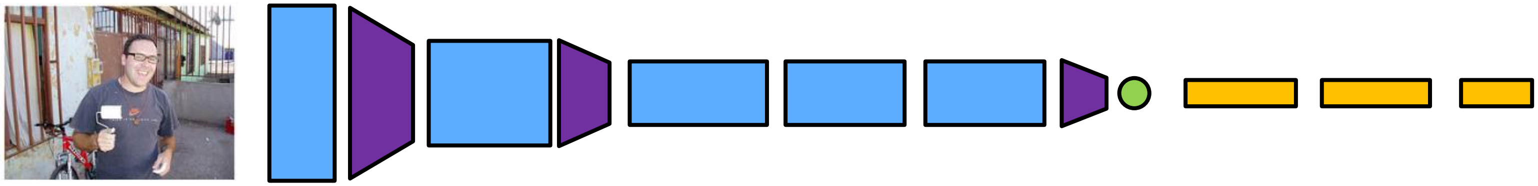
57,409,444

#Parameters

Convolution:
$C_{\text{out}} C_{\text{in}} K^2 + C_{\text{out}}$

Fully connected:
$MN + M$

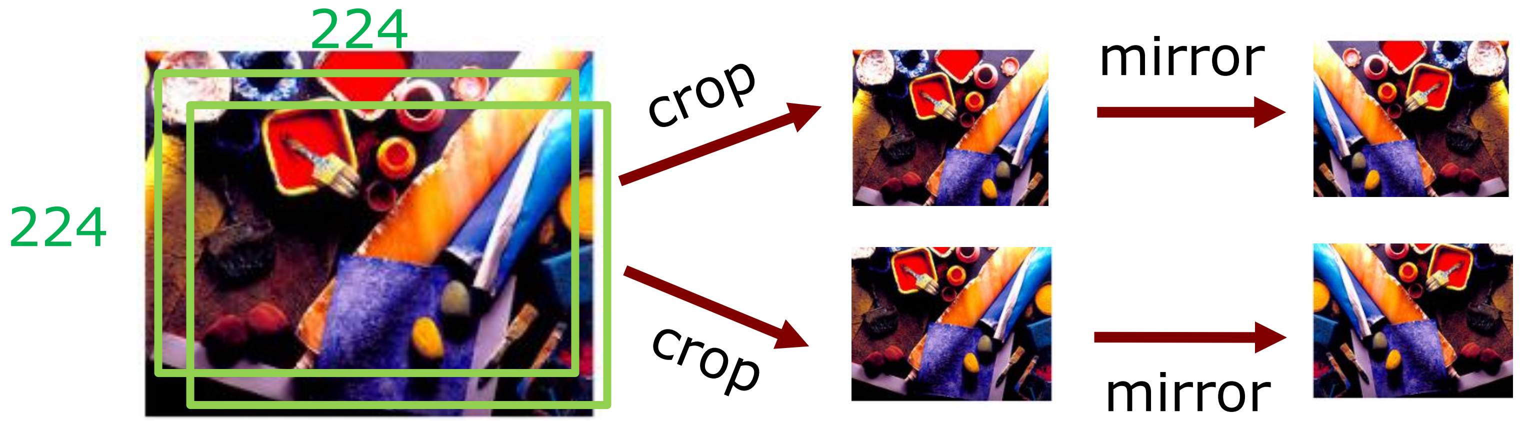
Design Decisions



- Experience and trial-and-error
- Most parameter in fully-connected layers

- Modern variants of CNNs:
  - Certain design patterns
  - Single fully connected layer

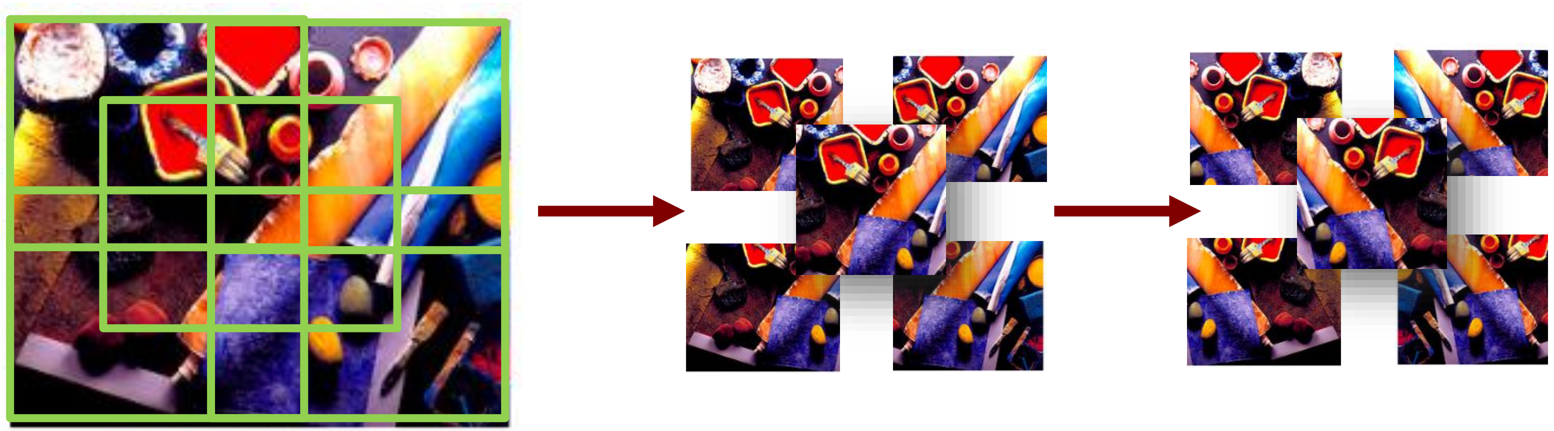
Additional Parts - Training



- **Data Augmentation**
  - Random crops while training
  - Horizontal reflection
  - Color variation by adding random values along principle components
- **Dropout**
  - Randomly set neurons to zero while training
  - Dropout in first two fully connected layers

[Krizhevsky, 2012]

Additional Parts - Testing



- Average over 5 crops with reflection at test time
- Average over ensemble of 5 CNNs trained with different initializations

[Krizhevsky, 2012]

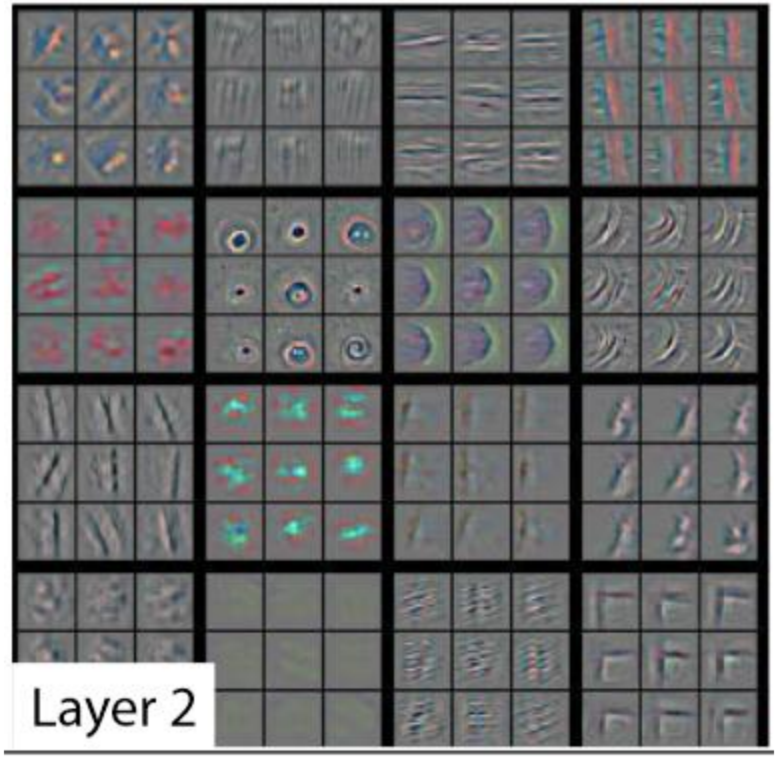
Learned Filters

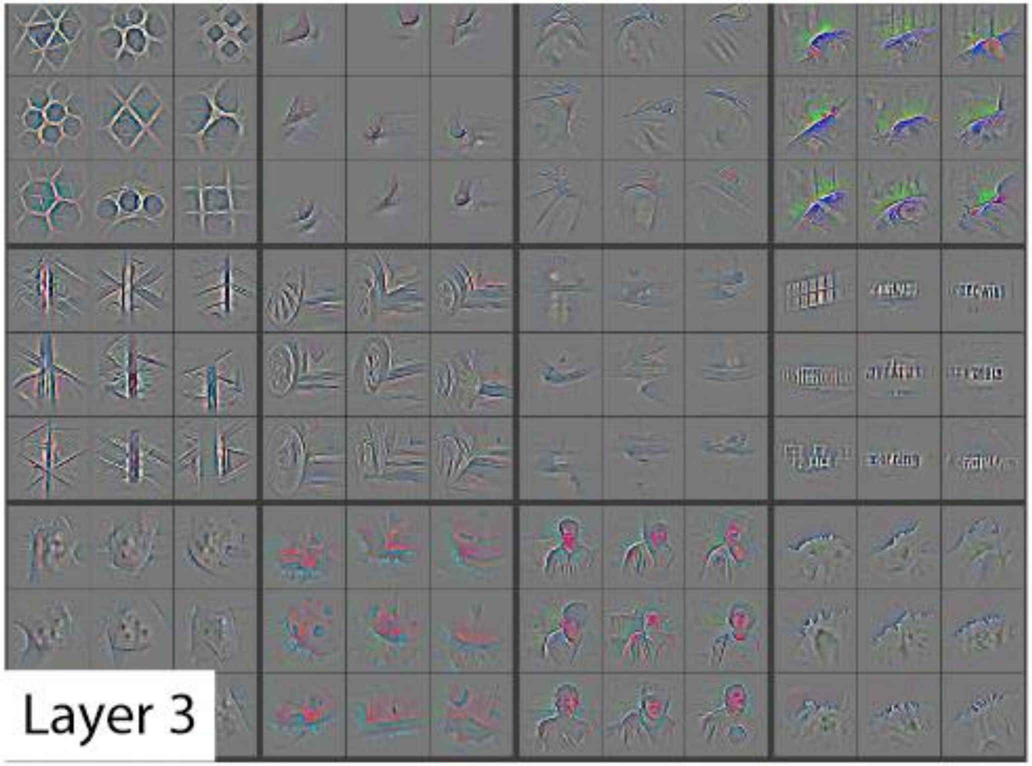
first layer kernels

- First layer learns “edge features”
- Low-level vision features

[Krizhevsky, 2012]

Deeper Layers

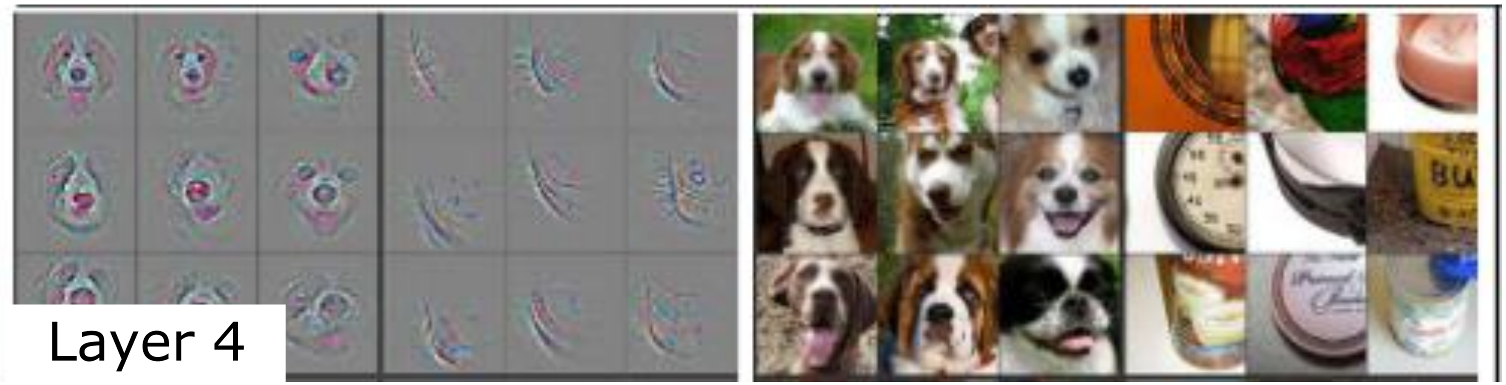




- Activation maps generated by passing images through network and inverting the convolution
- Higher layers learn texture features

[Zeiler, 2013]

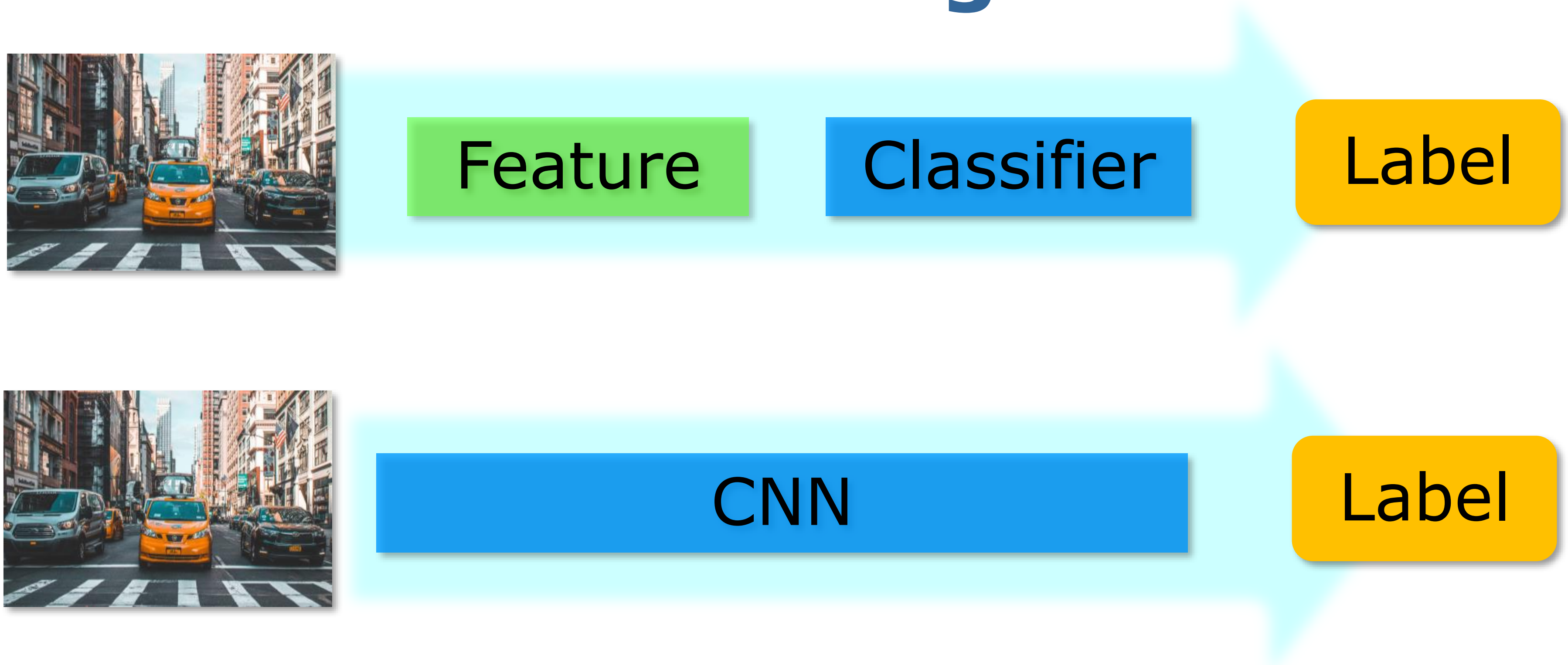
Deeper Layers



- Later layers react to parts and locations
- Aggregation of high level concepts

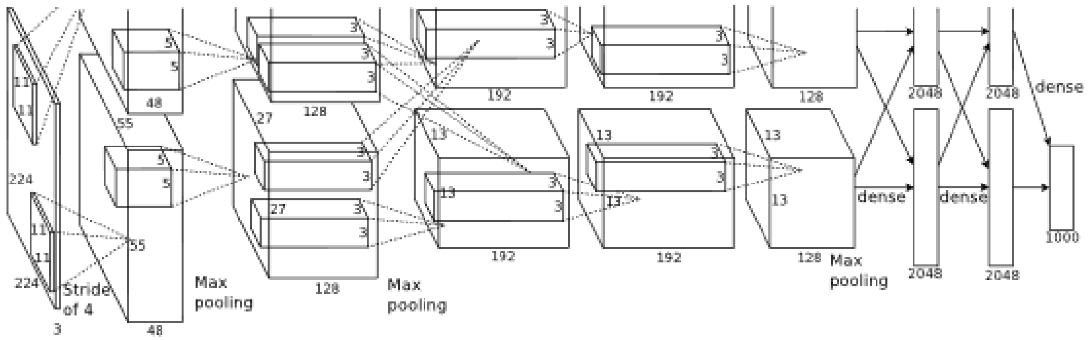
[Zeiler, 2013]

End-to-End Learning



- Traditional pipeline: Features-engineering
- **Now:** end-to-end learning of features and classifier

Summary



- Breakthrough of CNNs on ImageNet
- Main enabler: More Data & more compute
- CNNs learn strong features in end-to-end fashion
- Most ML for vision tasks nowadays tackled using CNNs

References

- Fukushima, „Neocognitron: A self-organizing neural network model for mechanism of pattern recognition unaddected by shift in position", Biological Cybernetics, 36(4): 193-202, 1980.
- Krizhevsky et al. „ImageNet Classification with Deep Convolutional Neural Networks", NeurIPS, 2012.
- Krizhevsky, "One weird trick for parallelizing convolutional neural networks", arxiv:1404.5997, 2014.
- LeCun et al. "Gradient-Based Learning Applied to Document Recognition", Proc. of the IEEE, 1998.
- Rumelhart et al. „Learning representations by back-propagating errors." Nature, 323, p. 533-536, 1986
- Zeiler & Fergus. "Visualizing and Understanding Convolutional Networks", ECCV, 2014.

See you next week!