Dive into Deep Learning

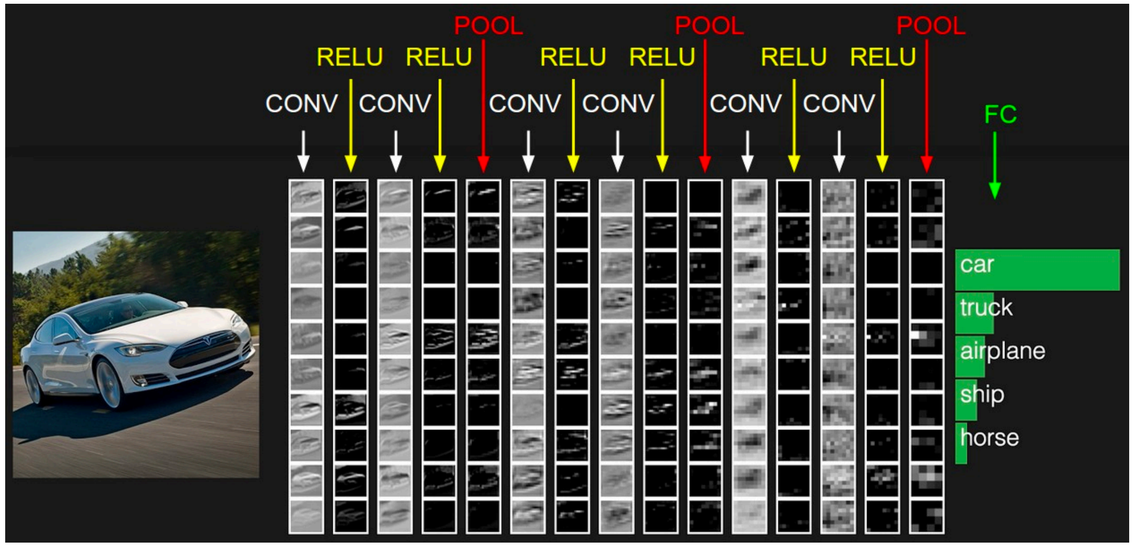
Chapter 6. Convolutional Neural Networks

Wayne L, Nov 1, 2020

CNN

CNN in a nutshell

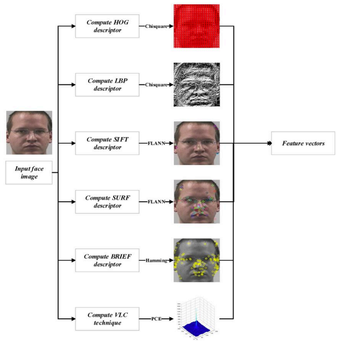
- A typical CNN takes a **raw RGB image** as an input.
- It then applies a series of **non-linear** operations on top of each other.
- These include convolution, activation, matrix multiplication, and pooling (sub-sampling) operations.
- The output of a CNN is a highly non-linear function of the raw RGB image pixels
- Key operations
  - Convolution layer: N-d convolution
  - Fully Connected layer: matrix multiplication
  - Activation layer: sigmoid, tanh, relu
  - Pooling layer: max pool, avg. pool



Before CNN

Traditional method (hand-crafted feature)

- lots options
- domain specific
- more complex
- poor performance



Sun et al.: Facial expression recognition in the wild based on multimodal texture features

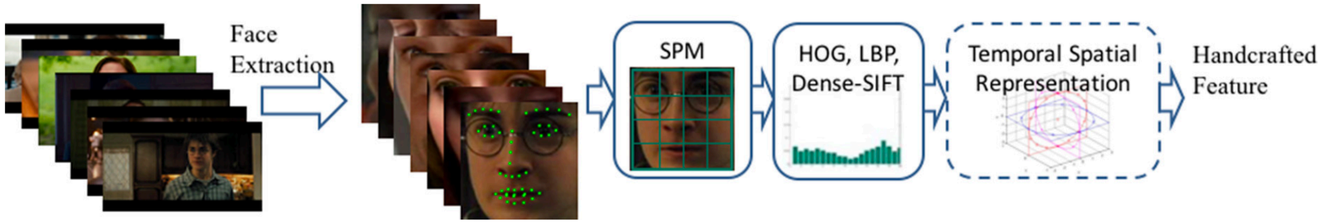
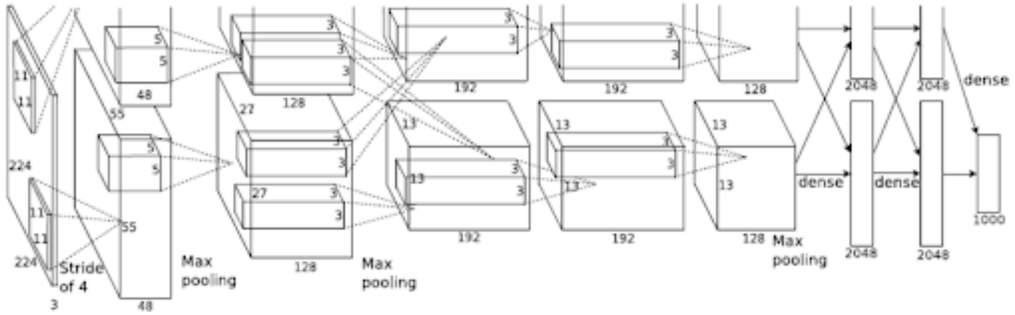
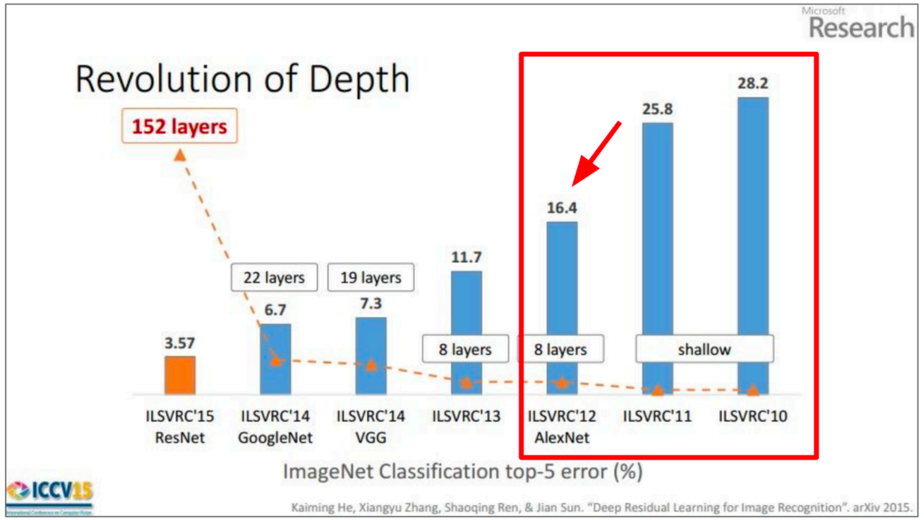


Fig. 4 Pipeline of handcrafted features extraction. The dashed box means that the temporal–spatial representation is only used for AFEW dataset.



Alexnet'12

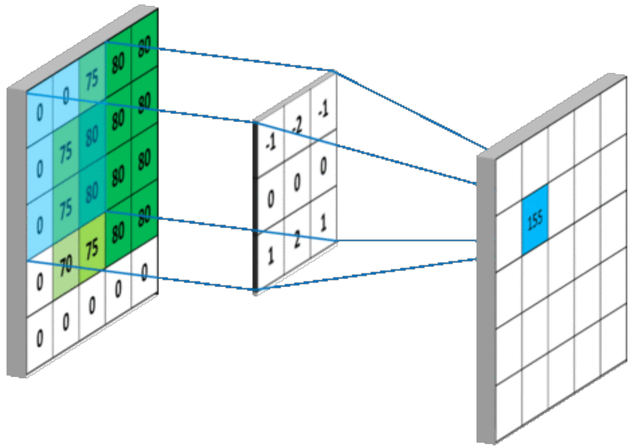


(slide from Kaiming He's recent presentation)

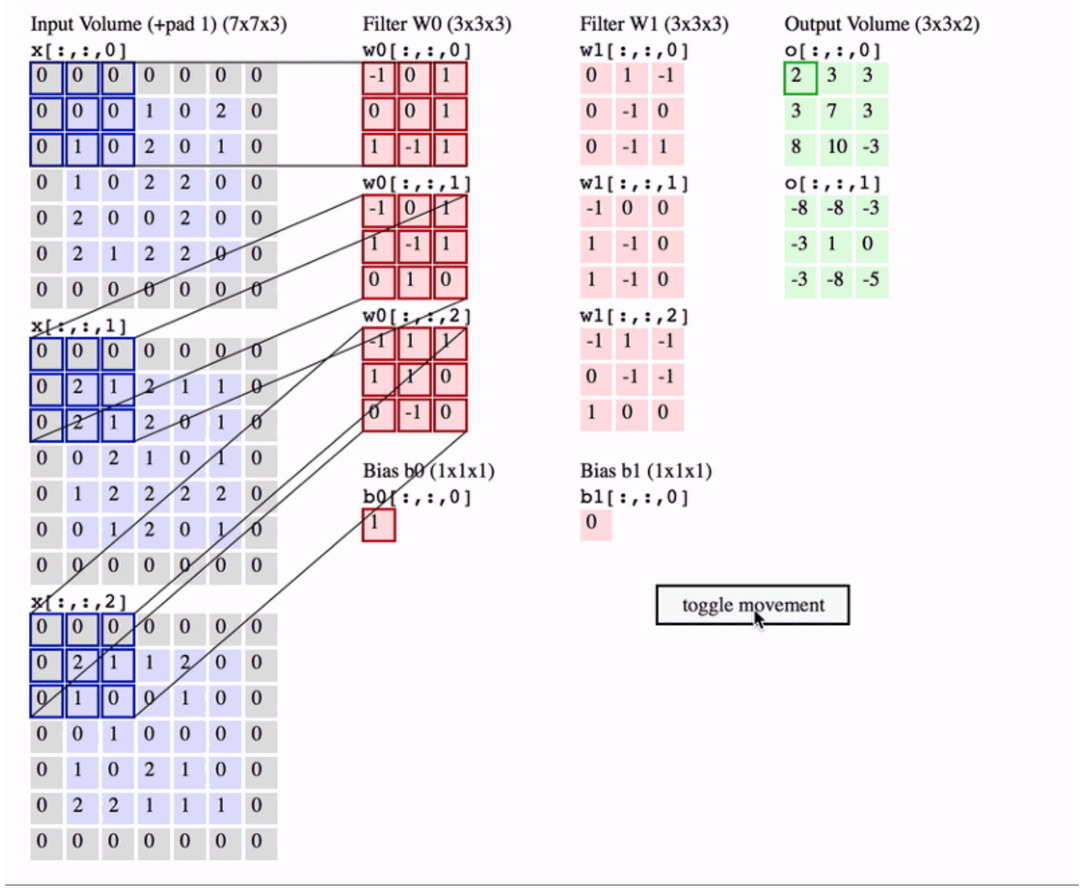
CNN

Multiple {Input & output} Channels

- Multiple channels (e.g., color image: RGB)
  - Shape: (3 x Height x Width)
- Kernel(Filter, Convolution map)



padding: same



-75%

2x2 receptive fields

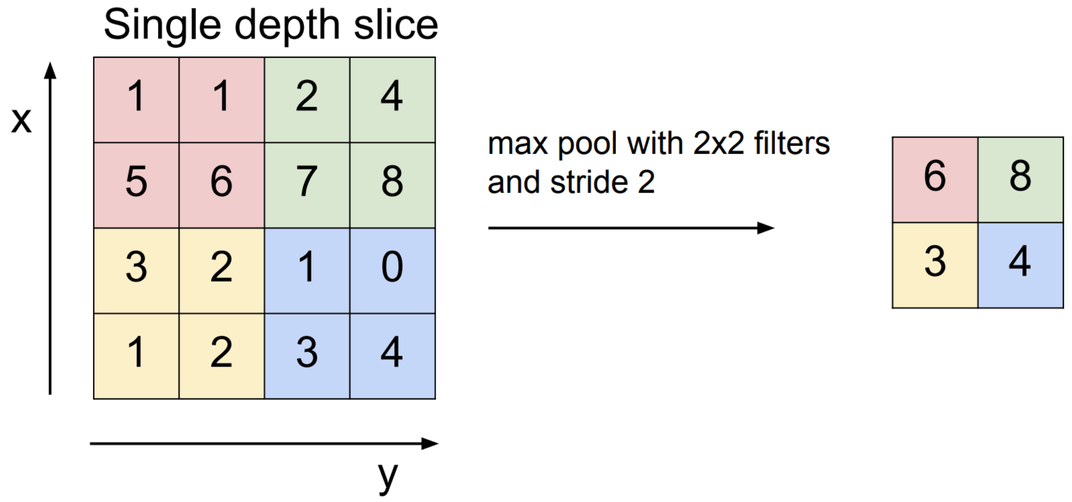
CNN

Pooling

- No parameters (there is no kernel)
- deterministic
- robustness

```
from d2l import torch as d2l
import torch
from torch import nn

def pool2d(X, pool_size, mode='max'):
    p_h, p_w = pool_size
    Y = torch.zeros((X.shape[0] - p_h + 1, X.shape[1] - p_w + 1))
    for i in range(Y.shape[0]):
        for j in range(Y.shape[1]):
            if mode == 'max':
                Y[i, j] = X[i: i + p_h, j: j + p_w].max()
            elif mode == 'avg':
                Y[i, j] = X[i: i + p_h, j: j + p_w].mean()
    return Y
```



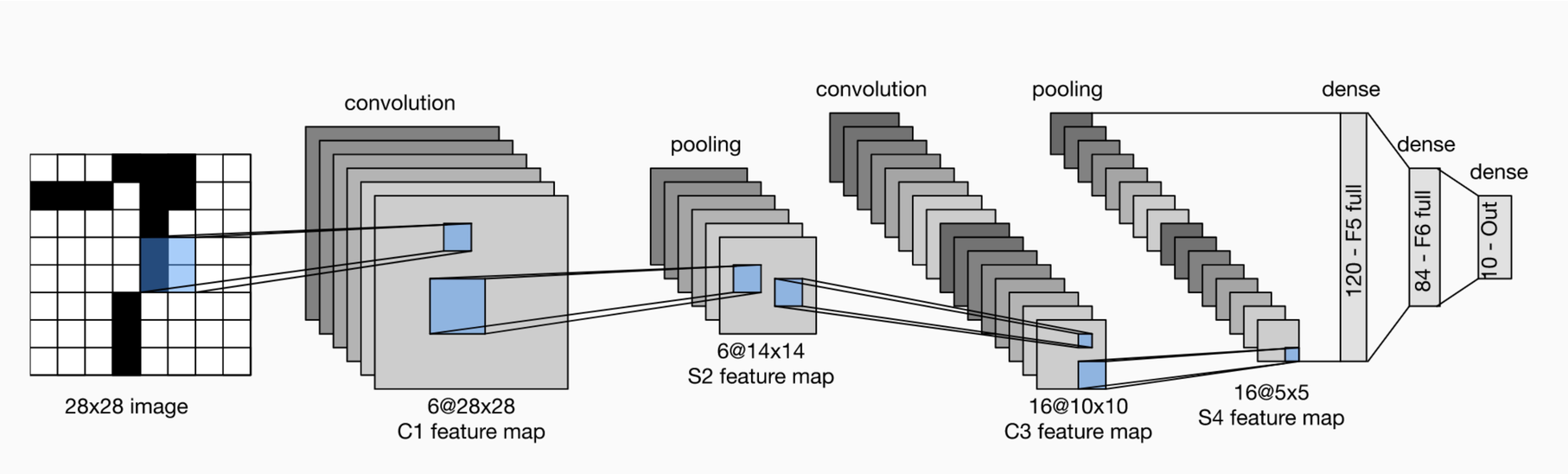
input_x = torch.tensor([[1,1,2,4,],[5,6,7,8],[3,2,1,0],[1,2,3,4]])
pool2d(input_x (2,2))

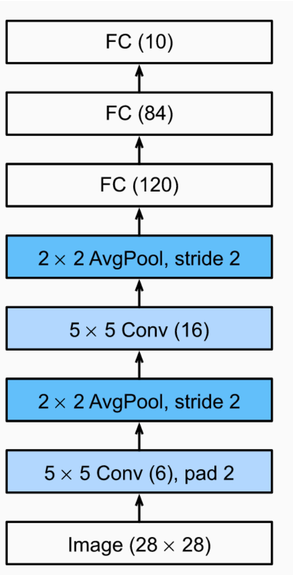
tensor([[6., 8.],[3., 4.]])

CNN

LeNet

- The first successful applications of convolutional networks were developed by Yann LeCun in 1990's. Of these, the best known is the LeNet architecture that was used to read zip codes, digits, etc.



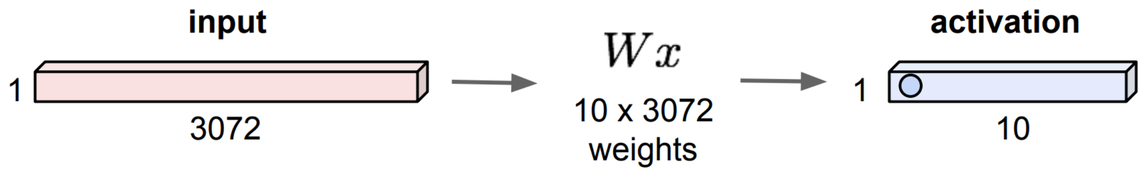


5x5 kernel average pooling sigmoid

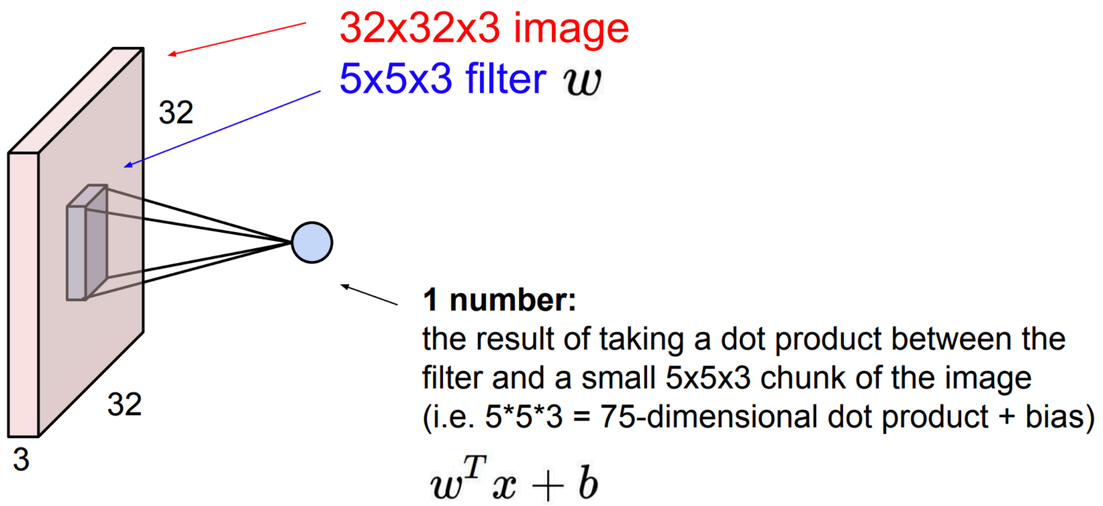
FC vc Conv

Why Conv is better?

- Fully connected layer
  - loss spatial information
  - massive weights



- Convolution layer
  - preserve spatial structure
  - extremely small weights



CNN

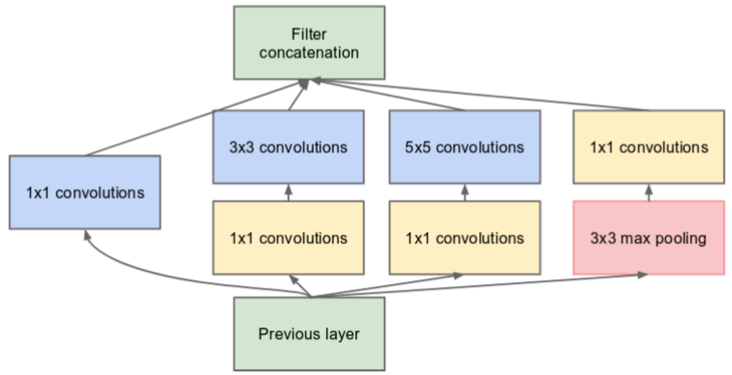
1 x 1 Convolution layer

Managing channel

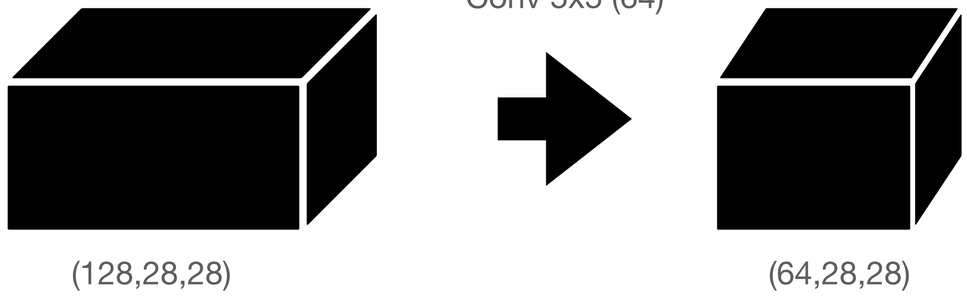
Efficient

Non-linearity

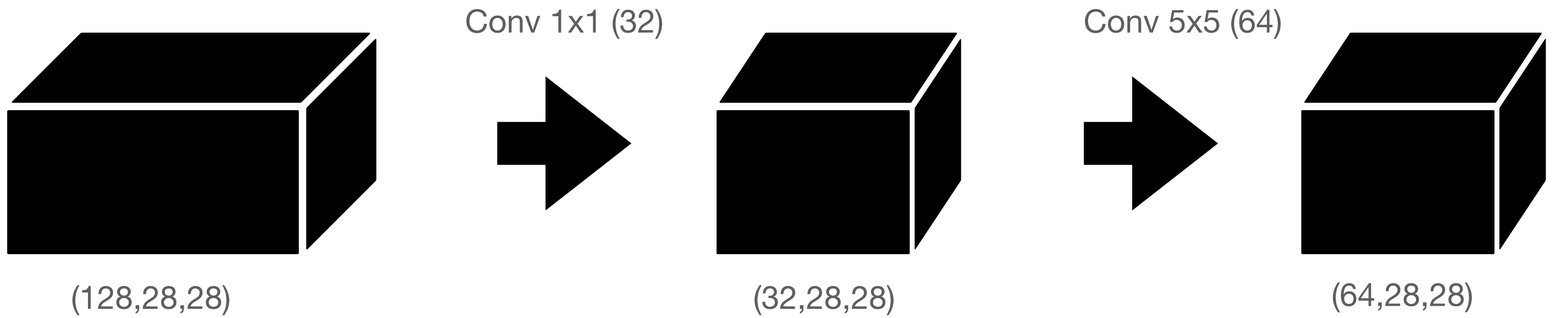
GoogLeNet (Inception module)



(b) Inception module with dimension reductions



# of params: 28*28*64*5*5*128 = **160M**



total: 44.8M

# of params: 28*28*32*1*1*128 = 4.8M

# of params: 28*28*64*5*5*32 = 40M

CNN

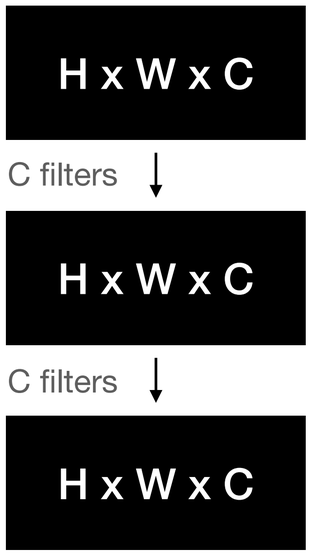
The power of small filters

one Conv with 7x7 filters

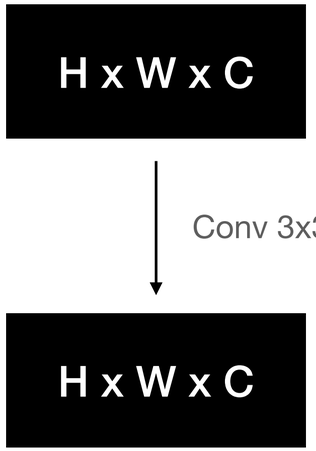
(7*7*C)*1*C = 49C^2

three Conv with 3x3 filters

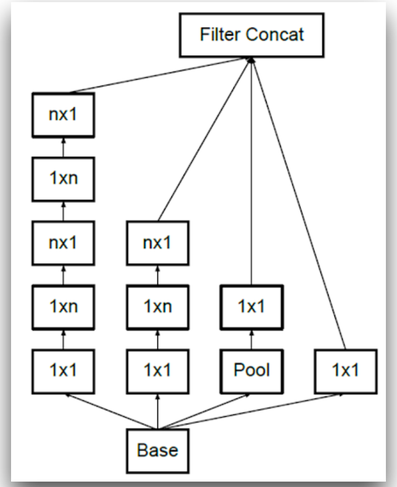
(3*3*C)*3*C = 27C^2



6C^2



9C^2



num_weights = (filter_width * filter_height * num_channel) * num_filters

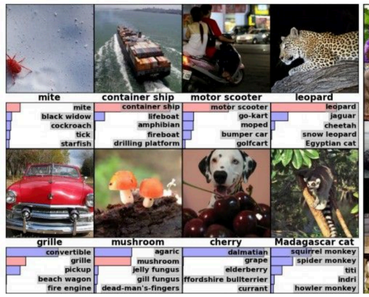
More non-linearity
fewer params
less compute

Why not try 1x1?

CNN

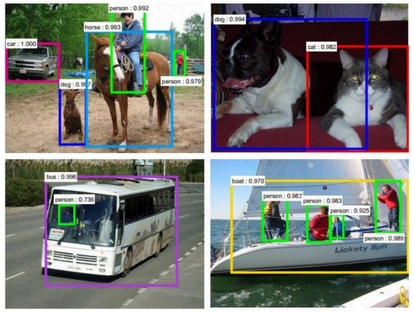
ConvNets are everywhere

Classification

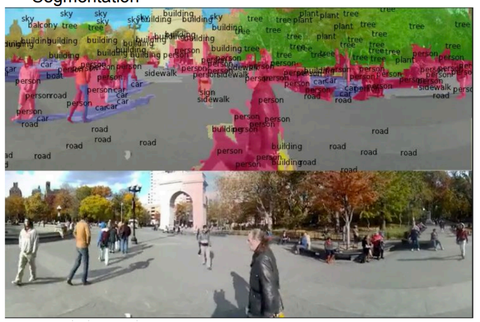


Retrieval

Detection

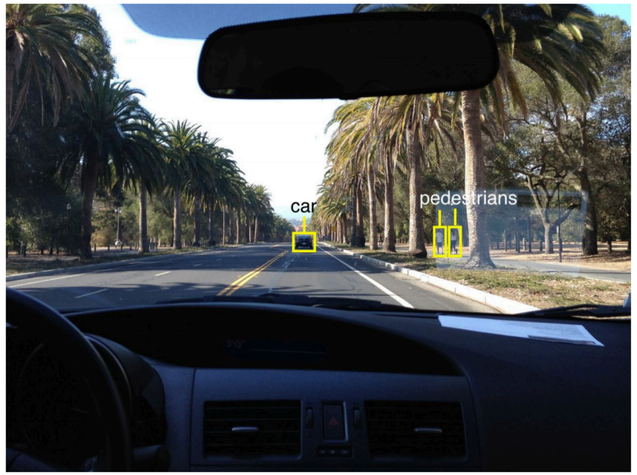


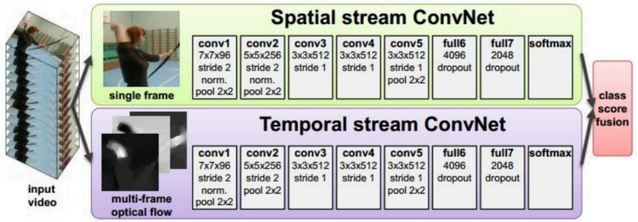
Segmentation

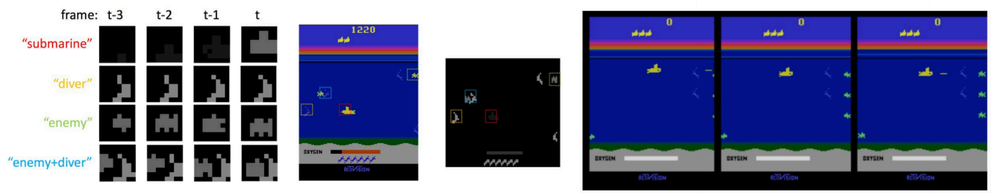


A white teddy bear sitting in the grass

A man riding a wave on top of a surfboard







CNN

summary

- ConvNets stack CONV, POOL, FC layers
- Historically architectures looked like
  [(CONV-RELU)*N - POOL?] * M - (FC-RELU)*K, SOFTMAX
  where N is usually up to ~5, M is large, 0 <= k <= 2
- but recent advances such as ResNet/GoogLeNet have challenged this paradigm
- ConvNetJS demo: training on CIFAR-10
  https://cs.stanford.edu/people/karpathy/convnetjs/demo/cifar10.html