Introduction to Convolutional Neural Networks

2018 / 02 / 23

Buzzword: CNN

Convolutional neural networks (CNN, ConvNet) is a class of deep, feed-forward (not recurrent) artificial neural networks that are applied to analyzing visual imagery.

Buzzword: CNN

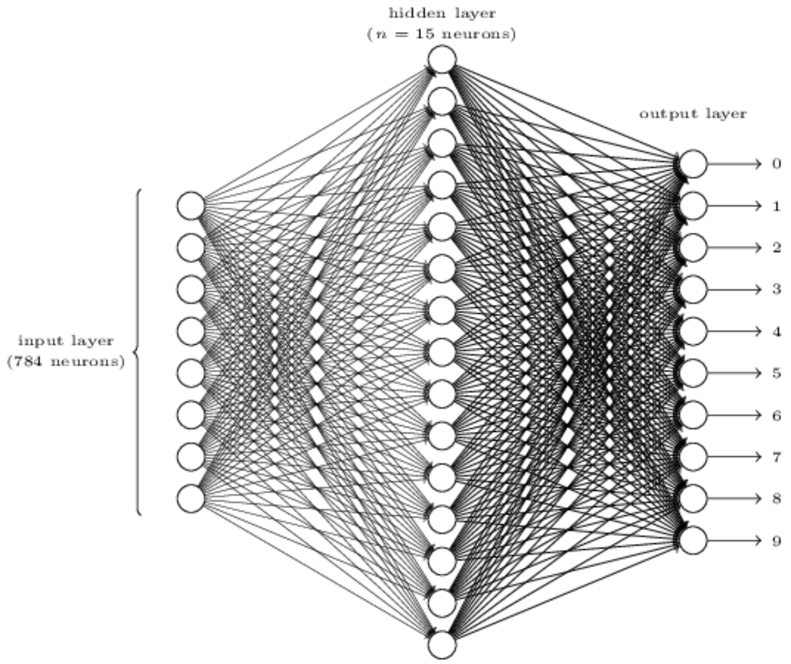
- Convolution

  From wikipedia,

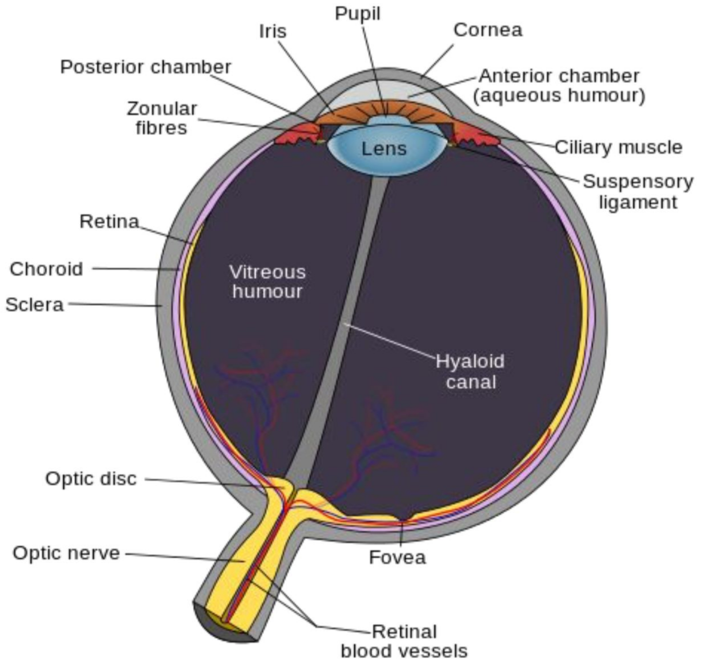
$$\begin{aligned}(f * g)(t) &\stackrel{\text{def}}{=} \int_{-\infty}^{\infty} f(\tau) g(t-\tau)\, d\tau \\ &= \int_{-\infty}^{\infty} f(t-\tau) g(\tau)\, d\tau.\end{aligned}$$

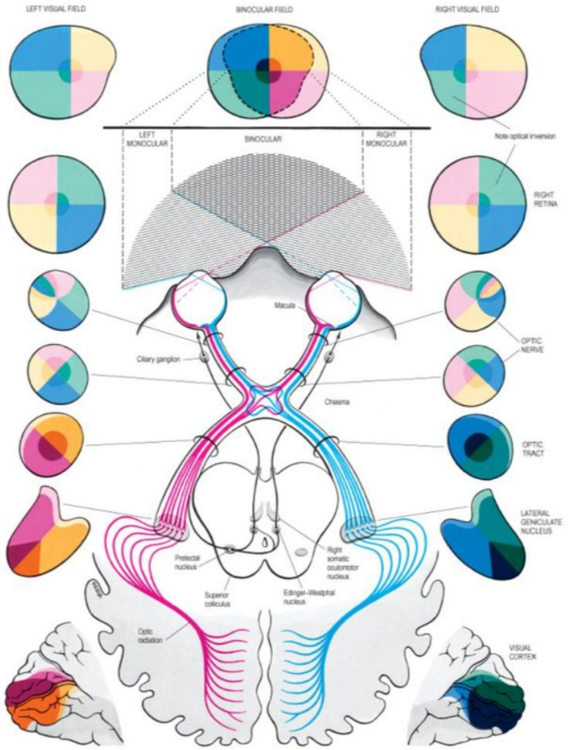
Buzzword: CNN

- Neural Networks

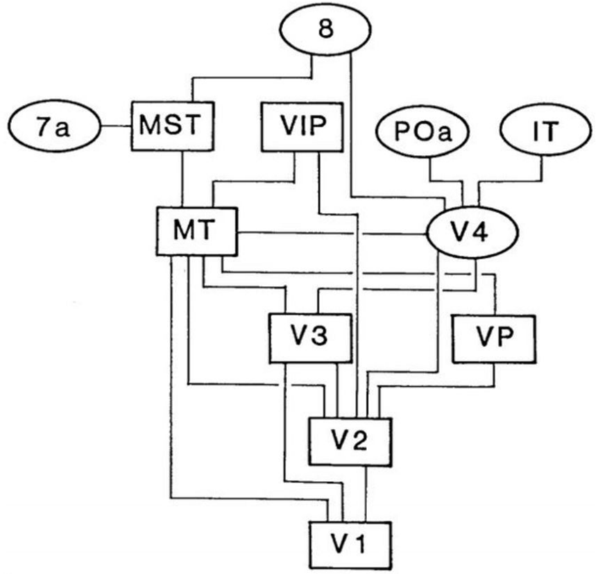


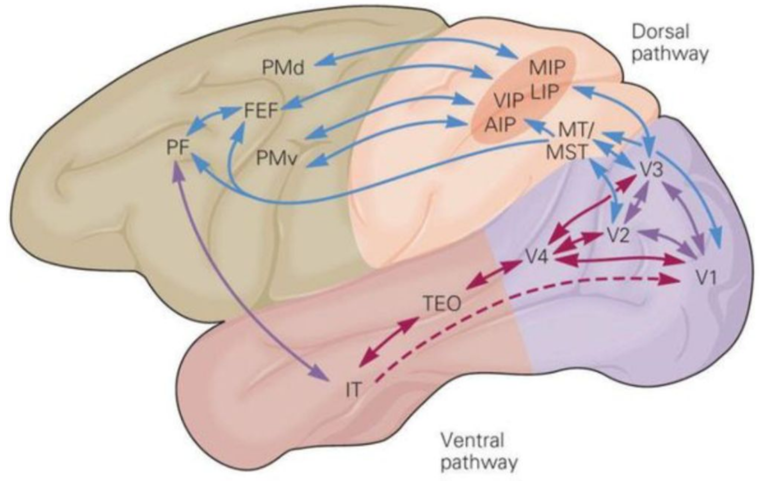
Background: Visual Signal Perception





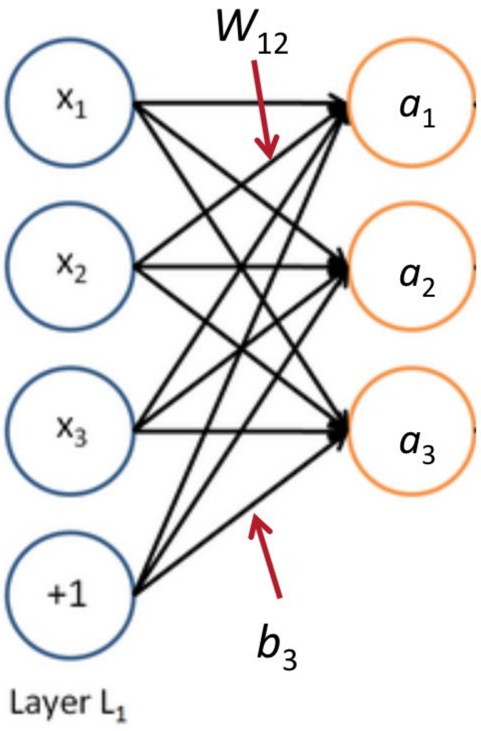
Background: Signal Relay





Starting from V1 primary visual cortex, visual signal is transmitted upwards, becoming more complicated and abstract.

Background: Neural Networks



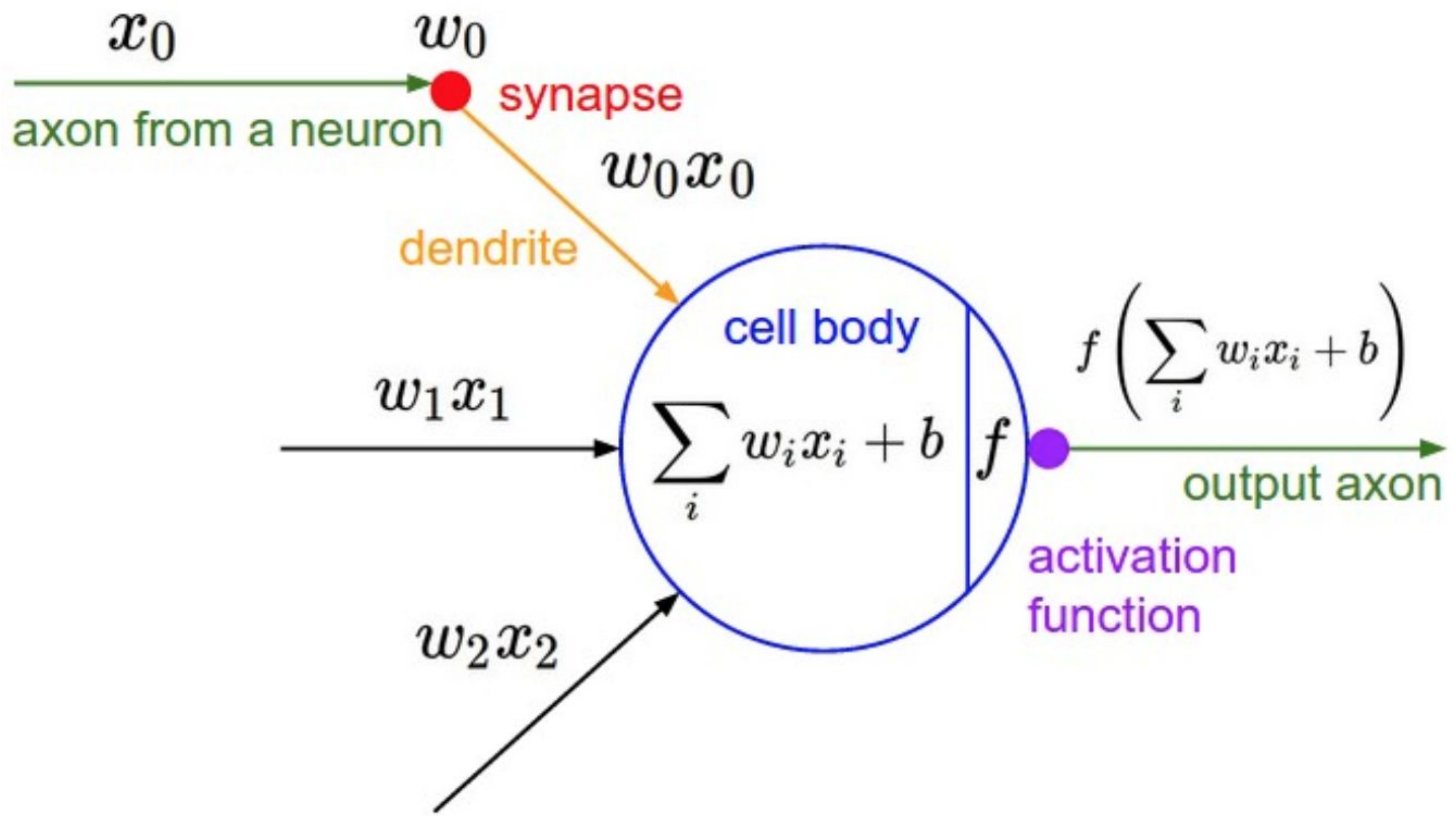
$$a_1 = f(W_{11}x_1 + W_{12}x_2 + W_{13}x_3 + b_1)$$
$$a_2 = f(W_{21}x_1 + W_{22}x_2 + W_{23}x_3 + b_2)$$

etc.

Express the equations in Matrix form, we have

$$z = Wx + b$$

$$a = f(z)$$


x_0
w_0
synapse
axon from a neuron
$w_0 x_0$
dendrite
cell body
$f\left(\sum_i w_i x_i + b\right)$
$w_1 x_1$
$\sum_i w_i x_i + b$
f
output axon
activation function
$w_2 x_2$

Neural Networks for Images

For computer vision, why can’t we just flatten the image and feed it through the neural networks?

Neural Networks for Images

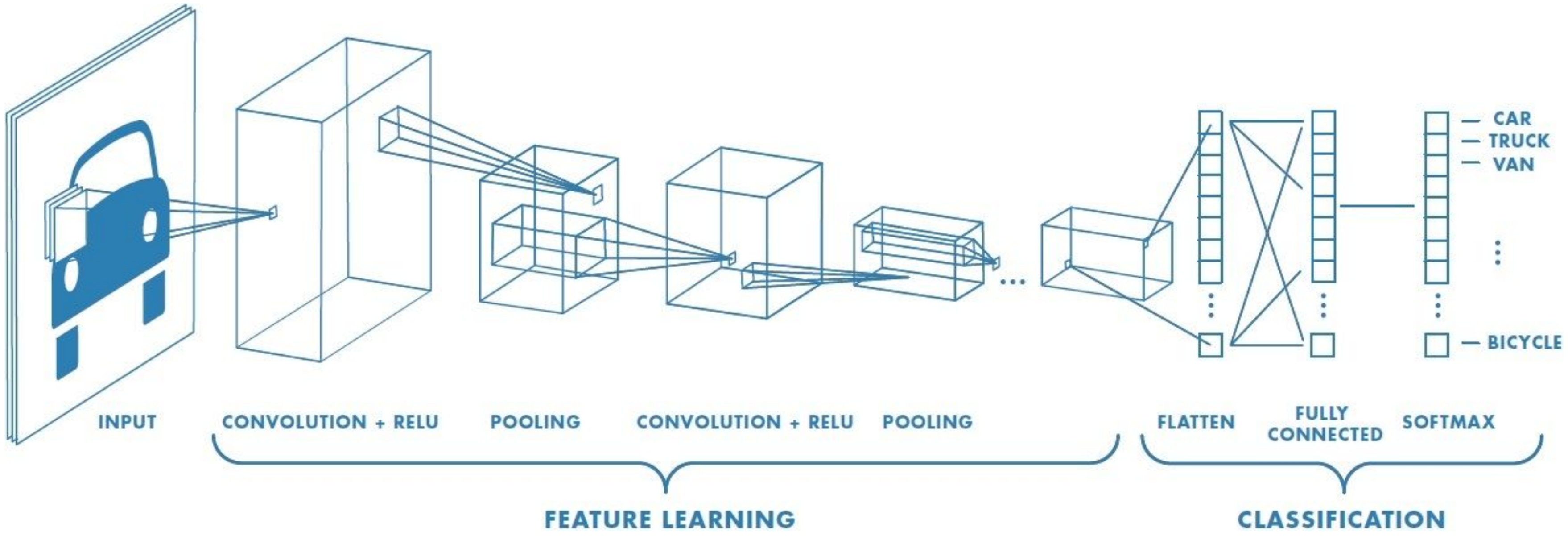
Images are high-dimensional vectors. It would take a huge amount of parameters to characterize the network.

Convolutional Neural Networks

To address this problem, bionic convolutional neural networks are proposed to reduced the number of parameters and adapt the network architecture specifically to vision tasks.

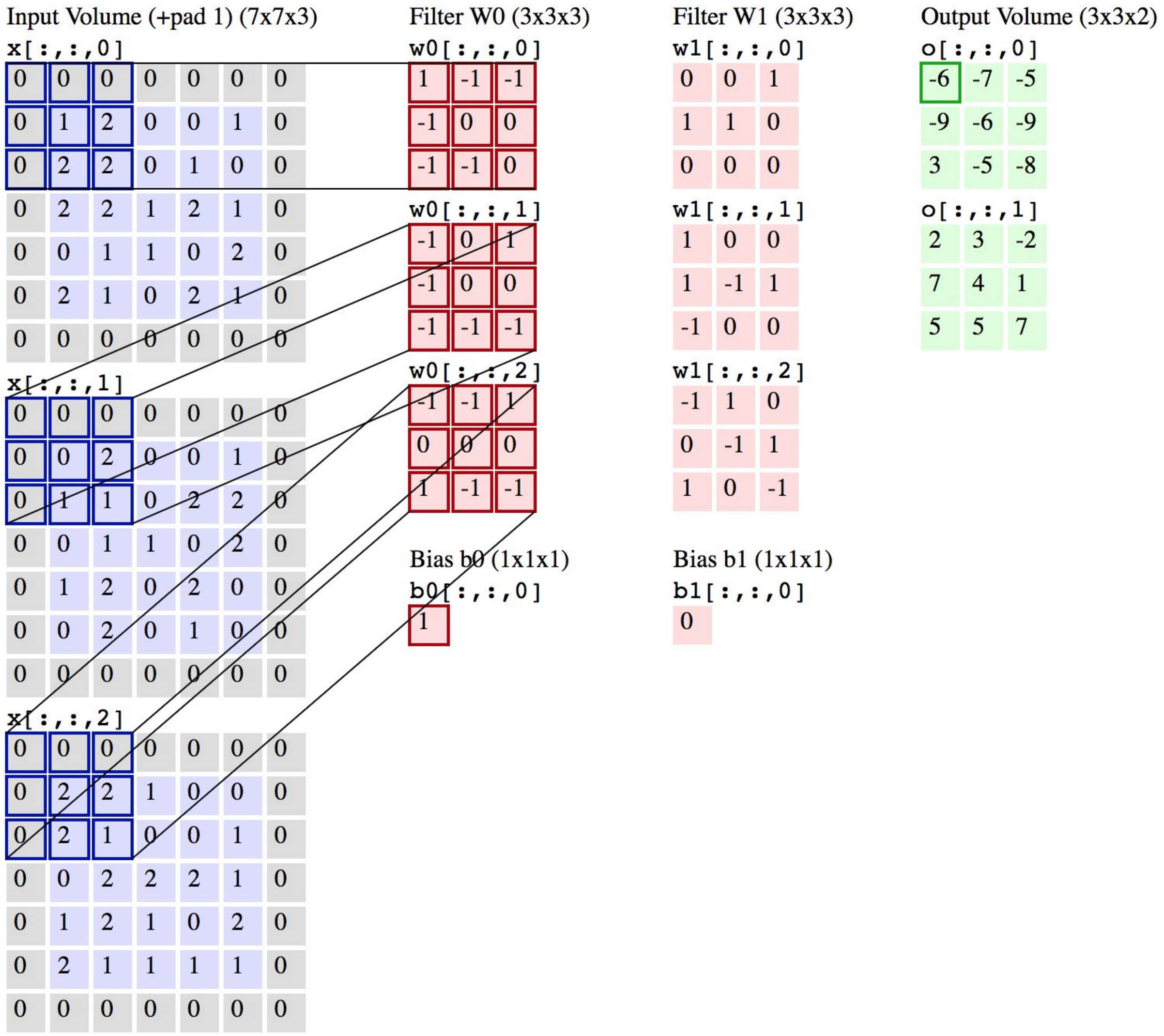
Convolutional neural networks are usually composed by a set of layers that can be grouped by their functionalities.

Sample Architecture



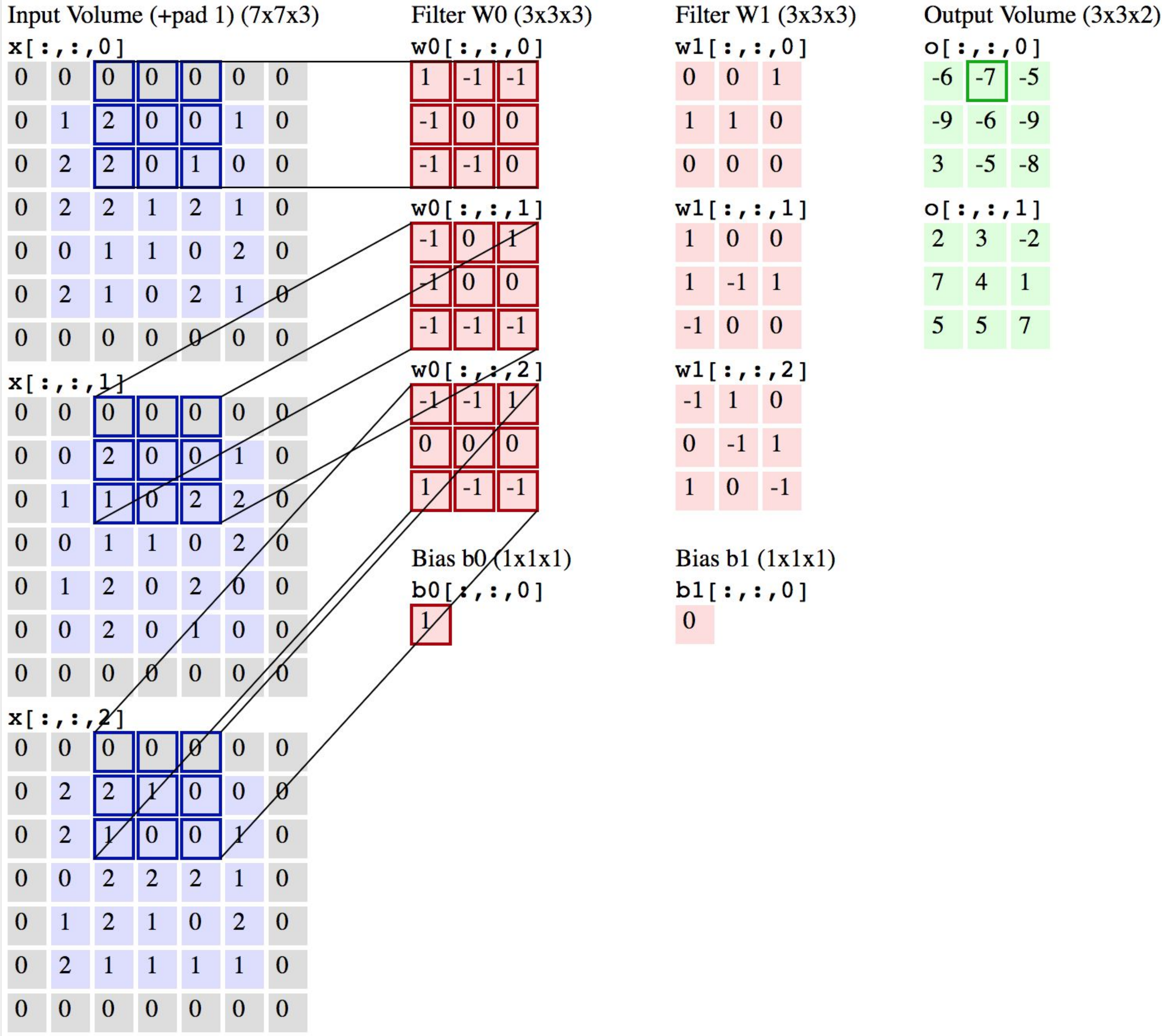
Convolution Layer

- The process is a 2D convolution on the inputs.
- The “dot products” between weights and inputs are “integrated” across “channels”.
- Filter weights are shared across receptive fields. The filter has same number of layers as input volume channels, and output volume has same “depth” as the number of filters.



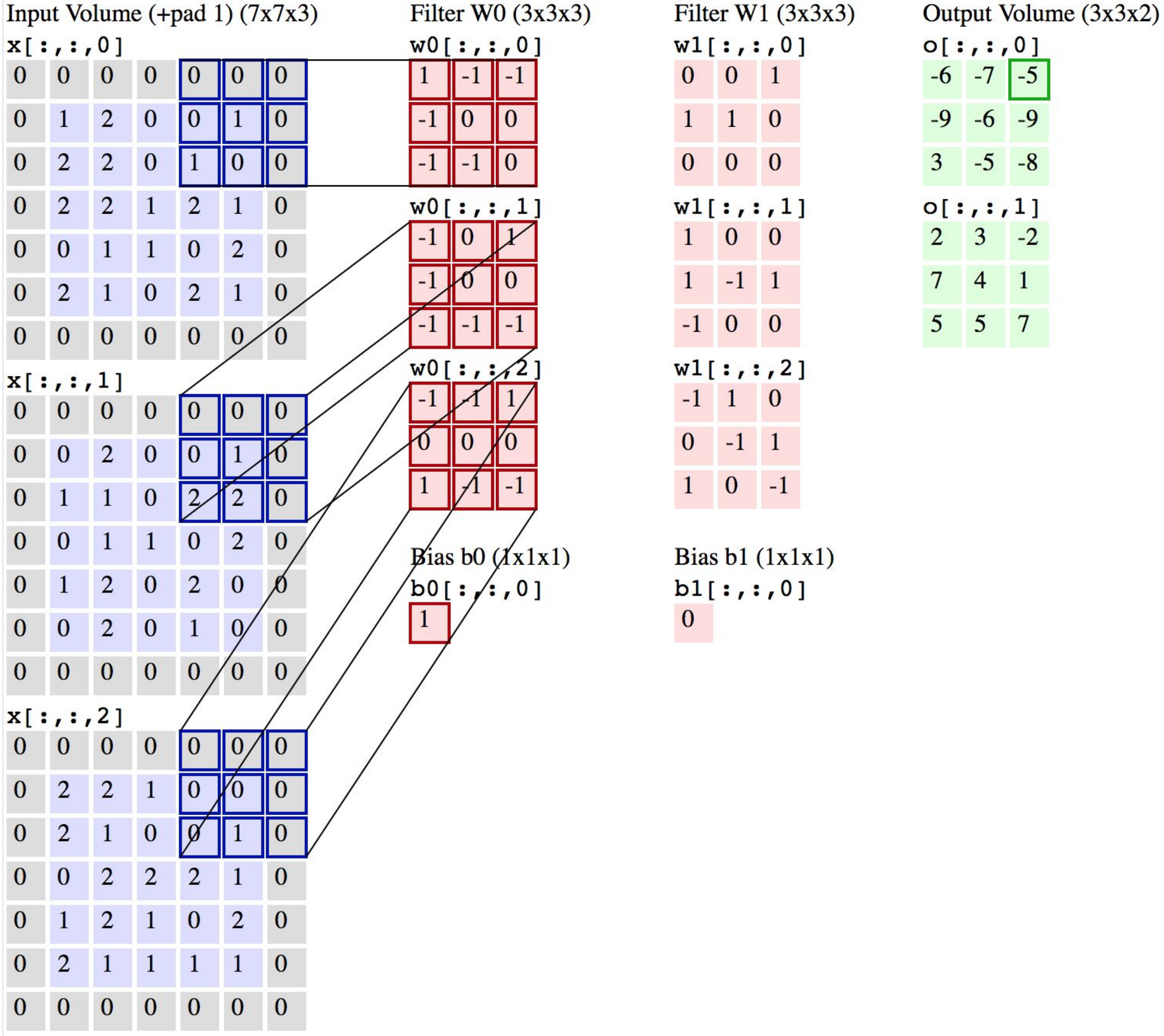
Convolution Layer

- The process is a 2D convolution on the inputs.
- The "dot products" between weights and inputs are "integrated" across "channels".
- Filter weights are shared across receptive fields. The filter has same number of layers as input volume channels, and output volume has same "depth" as the number of filters.



Convolution Layer

- The process is a 2D convolution on the inputs.
- The "dot products" between weights and inputs are "integrated" across "channels".
- Filter weights are shared across receptive fields. The filter has same number of layers as input volume channels, and output volume has same "depth" as the number of filters.



Input Volume (+pad 1) (7x7x3)

x[:,:,0]

0	0	0	0	0	0	0
0	1	2	0	0	1	0
0	2	2	0	1	0	0
0	2	2	1	2	1	0
0	0	1	1	0	2	0
0	2	1	0	2	1	0
0	0	0	0	0	0	0

x[:,:,1]

0	0	0	0	0	0	0
0	0	2	0	0	1	0
0	1	1	0	2	2	0
0	0	1	1	0	2	0
0	1	2	0	2	0	0
0	0	2	0	1	0	0
0	0	0	0	0	0	0

x[:,:,2]

0	0	0	0	0	0	0
0	2	2	1	0	0	0
0	2	1	0	0	1	0
0	0	2	2	2	1	0
0	1	2	1	0	2	0
0	2	1	1	1	1	0
0	0	0	0	0	0	0

Filter W0 (3x3x3)

w0[:,:,0]

1	-1	-1
-1	0	0
-1	-1	0

w0[:,:,1]

-1	0	1
-1	0	0
-1	-1	-1

w0[:,:,2]

-1	-1	1
0	0	0
1	-1	-1

Bias b0 (1x1x1)

b0[:,:,0]

1

Filter W1 (3x3x3)

w1[:,:,0]

0	0	1
1	1	0
0	0	0

w1[:,:,1]

1	0	0
1	-1	1
-1	0	0

w1[:,:,2]

-1	1	0
0	-1	1
1	0	-1

Bias b1 (1x1x1)

b1[:,:,0]

0

Output Volume (3x3x2)

o[:,:,0]

-6	-7	-5
-9	-6	-9
3	-5	-8

o[:,:,1]

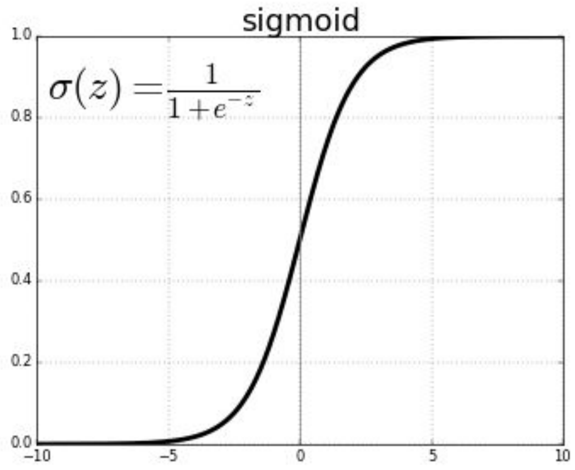
2	3	-2
7	4	1
5	5	7

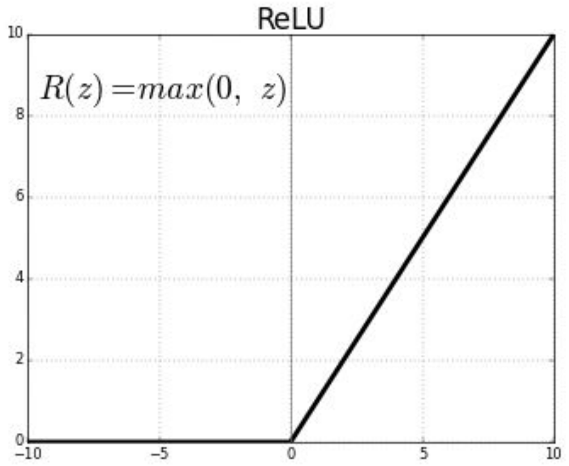
Convolution Layer

- Accepts a volume of size $W_1 \times H_1 \times D_1$
- Requires four hyperparameters:
  - Number of filters K,
  - their spatial extent F,
  - the stride S,
  - the amount of zero padding P.
- Produces a volume of size $W_2 \times H_2 \times D_2$ where:
  - $W_2 = (W_1 - F + 2P)/S + 1$
  - $H_2 = (H_1 - F + 2P)/S + 1$ (i.e. width and height are computed equally by symmetry)
  - $D_2 = K$

Activation Layer

- Used to increase non-linearity of the network without affecting receptive fields of conv layers
- Prefer ReLU, results in faster training
- LeakyReLU addresses the vanishing gradient problem





Other types:
Leaky ReLU, Randomized Leaky ReLU, Parameterized ReLU
Exponential Linear Units (ELU), Scaled Exponential Linear Units
Tanh, hardtanh, softtanh, softsign, softmax, softplus...

Softmax

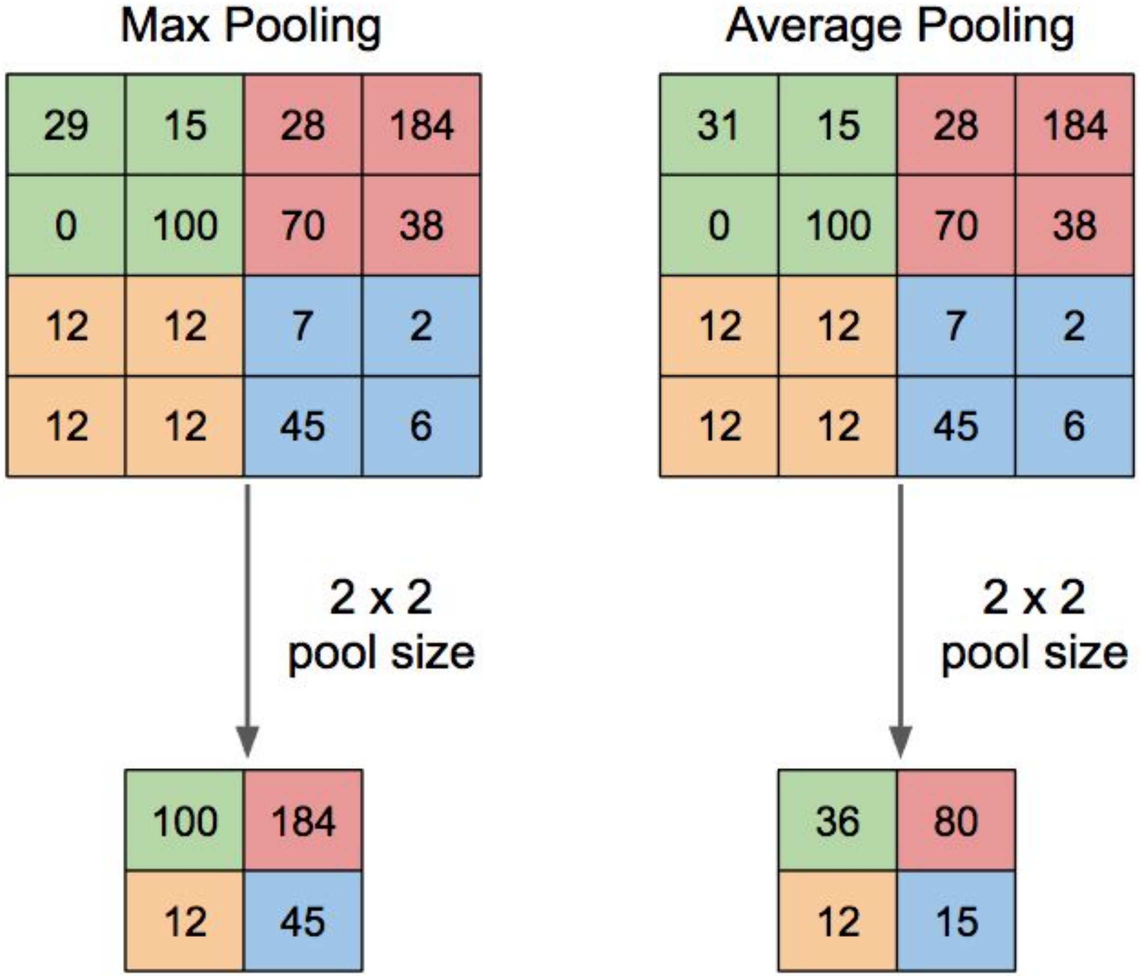
- A special kind of activation layer, usually at the end of FC layer outputs
- Can be viewed as a fancy normalizer (a.k.a. Normalized exponential function)
- Produce a discrete probability distribution vector
- Very convenient when combined with cross-entropy loss

$$P(y = j \mid \mathbf{x}) = \frac{e^{\mathbf{x}^\mathsf{T}\mathbf{w}_j}}{\sum_{k=1}^{K} e^{\mathbf{x}^\mathsf{T}\mathbf{w}_k}}$$

Given sample vector input **x** and weight vectors {$\mathbf{w}_j$}, the predicted probability of y = j

Pooling Layer

- Convolutional layers provide activation maps.
- Pooling layer applies non-linear downsampling on activation maps.
- Pooling is aggressive (discard info); the trend is to use smaller filter size and abandon pooling

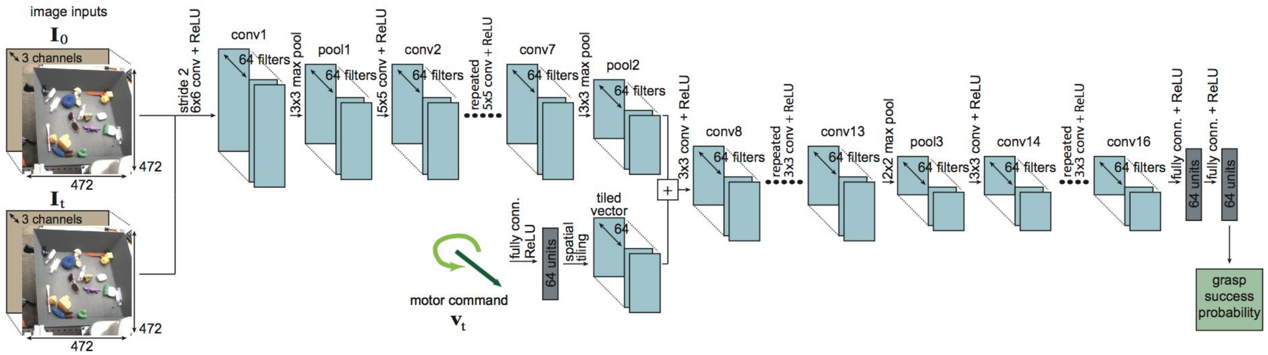


Pooling Layer

- Accepts a volume of size $W_1 \times H_1 \times D_1$
- Requires two hyperparameters:
  - their spatial extent F,
  - the stride S,
- Produces a volume of size $W_2 \times H_2 \times D_2$ where:
  - $W_2 = (W_1 - F)/S + 1$
  - $H_2 = (H_1 - F)/S + 1$
  - $D_2 = D_1$
- Introduces zero parameters since it computes a fixed function of the input
- Note that it is not common to use zero-padding for Pooling layers

FC Layer

- Regular neural network
- Can view as the final learning phase, which maps extracted visual features to desired outputs
- Usually adaptive to classification/encoding tasks
- Common output is a vector, which is then passed through softmax to represent confidence of classification
- The outputs can also be used as “bottleneck”



In above example, FC generates a number which is then passed through a sigmoid to represent grasp success probability

Loss Layer

- L1, L2 loss
- Cross-Entropy loss (works well for classification, e.g., image classification)
- Hinge Loss
- Huber Loss, more resilient to outliers with smooth gradient
- Minimum Squared Error (works well for regression task, e.g., Behavioral Cloning)

$$H(p,q) = -\sum_x p(x)\,\log q(x)$$

Binary case

$$-y\log\hat{y} - (1-y)\log(1-\hat{y})$$

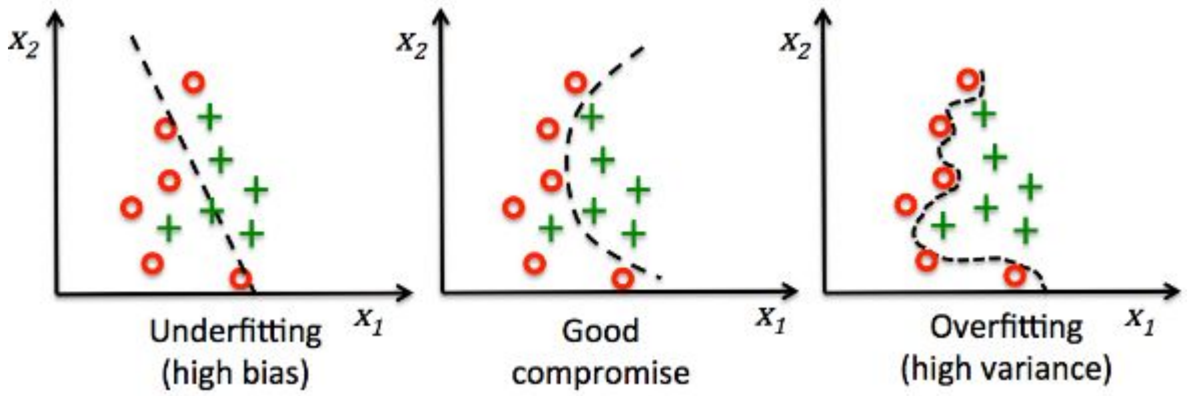
General case

$$-\sum_i p_i \log q_i$$

Regularization

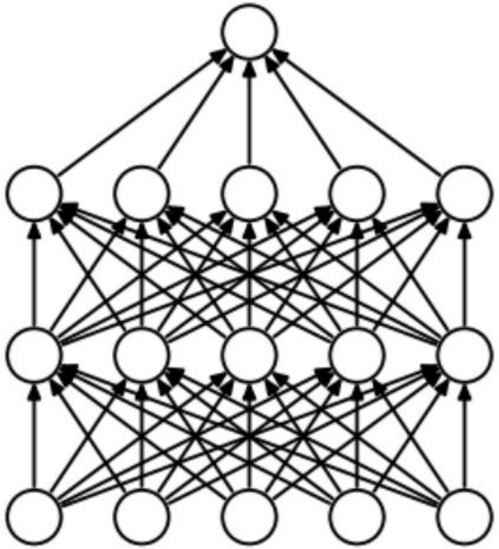
- L1 / L2
- Dropout
- Batch norm
- Gradient clipping
- Max norm constraint

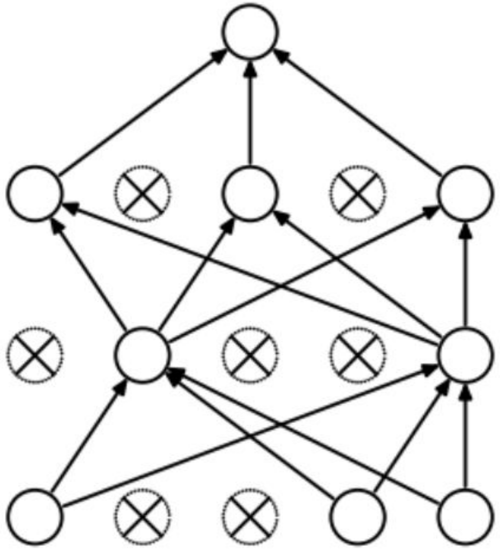
To prevent overfitting with huge amount of training data



Dropout

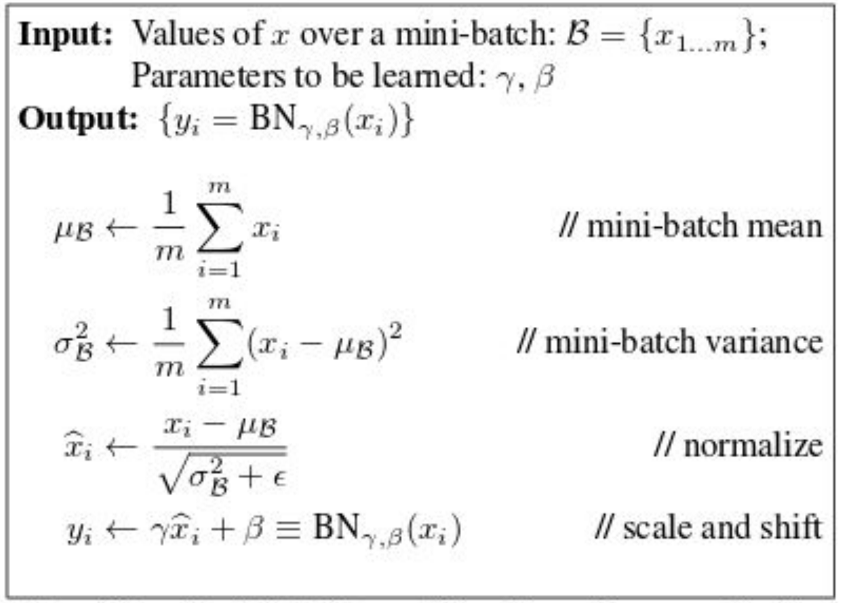
- During training, randomly ignore activations by probability ***p***
- During testing, use all activations but scale them by ***p***
- Effectively prevent overfitting by reducing correlation between neurons


(a) Standard Neural Net


(b) After applying dropout.

Batch Normalization

- Makes networks robust to bad initialization of weights
- Usually inserted right before activation layers
- Reduce covariance shift by normalizing and scaling inputs
- The scale and shift parameters are trainable to avoid losing stability of the network



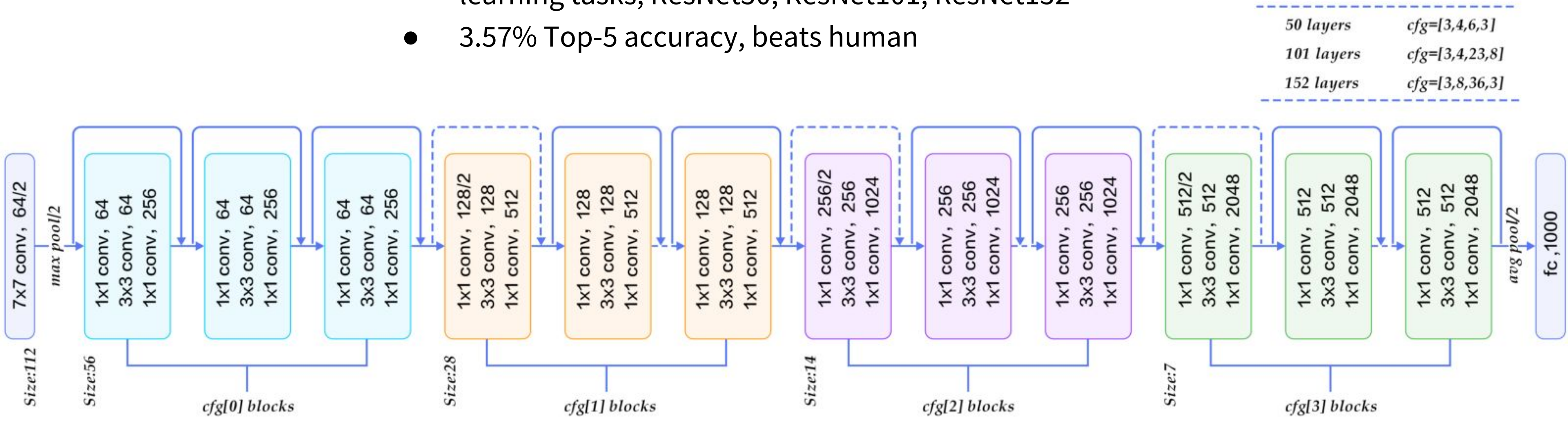
```
Input: Values of $x$ over a mini-batch: $\mathcal{B} = \{x_{1...m}\}$;
       Parameters to be learned: $\gamma, \beta$
Output: $\{y_i = \mathrm{BN}_{\gamma,\beta}(x_i)\}$

    $\mu_\mathcal{B} \leftarrow \frac{1}{m}\sum_{i=1}^{m} x_i$                          // mini-batch mean
    $\sigma_\mathcal{B}^2 \leftarrow \frac{1}{m}\sum_{i=1}^{m}(x_i - \mu_\mathcal{B})^2$        // mini-batch variance
    $\widehat{x}_i \leftarrow \frac{x_i - \mu_\mathcal{B}}{\sqrt{\sigma_\mathcal{B}^2 + \epsilon}}$                    // normalize
    $y_i \leftarrow \gamma\widehat{x}_i + \beta \equiv \mathrm{BN}_{\gamma,\beta}(x_i)$          // scale and shift
```

Algorithm 1: Batch Normalizing Transform, applied to activation x over a mini-batch.

Example: ResNet

- Residual Network, by Kaiming He (2015)
- Heavy usage of “skip connections” which are similar to RNN Gated Recurrent Units (GRU)
- Commonly used as visual feature extractor in all kinds of learning tasks, ResNet50, ResNet101, ResNet152
- 3.57% Top-5 accuracy, beats human



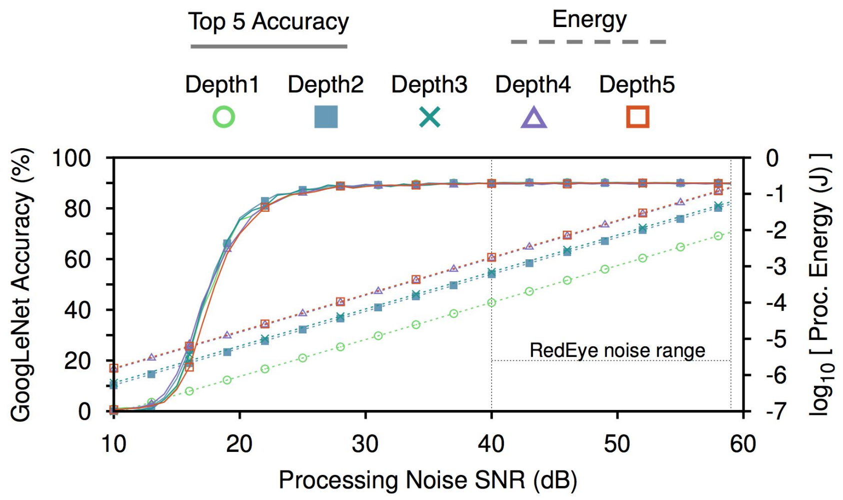
Applications

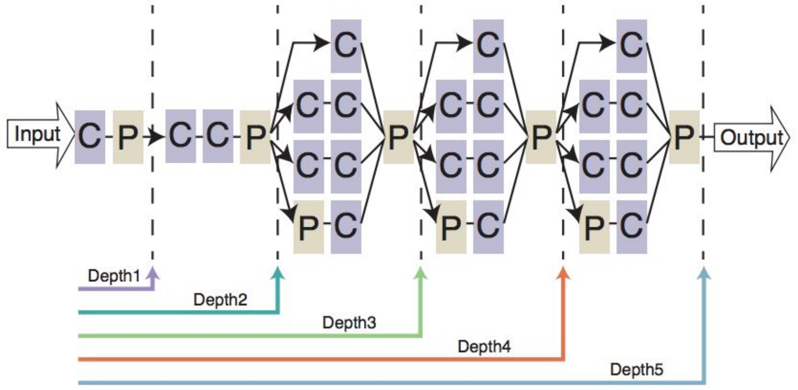
Can be viewed as a fancy feature extractor, just like SIFT, SURF, etc

CNN & Vision: RedEye

https://roblkw.com/papers/likamwa2016redeye-isca.pdf

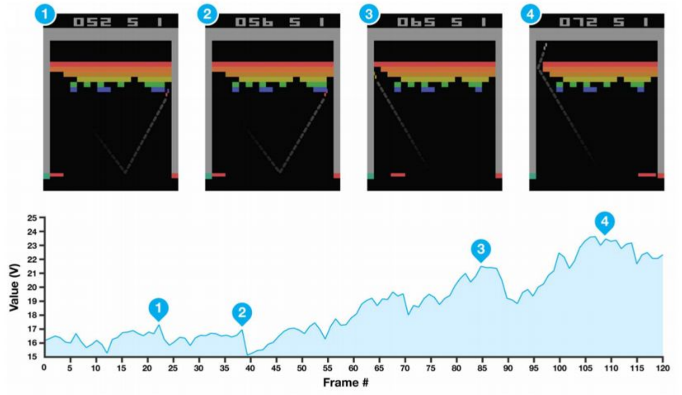
Two keys: Add noise, save bandwidth/energy

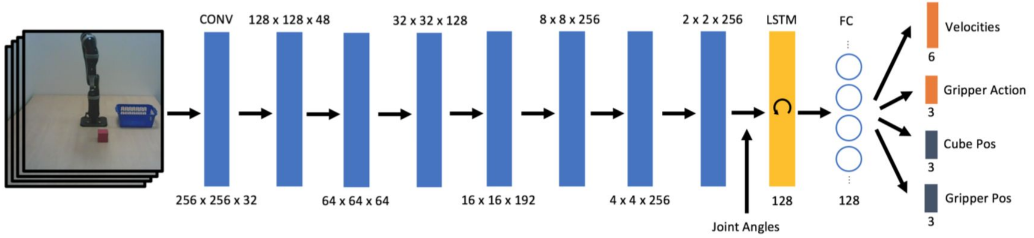




CNN & Robotics: RL Example

Usually used with Multi-Layer Perceptron (MLP, can be viewed as a fancy term for non-trivial neural networks) for policy networks.





Software 2.0

Quotes from Andrej Karpathy:

Software 1.0 is what we're all familiar with—it is written in languages such as Python, C++, etc. It consists of explicit instructions to the computer written by a programmer. By writing each line of code, the programmer is identifying a specific point in program space with some desirable behavior.

Software 2.0 is written in neural network weights. No human is involved in writing this code because there are a lot of weights (typical networks might have millions). Instead, we specify some constraints on the behavior of a desirable program (e.g., a dataset of input output pairs of examples) and use the computational resources at our disposal to search the program space for a program that satisfies the constraints.

Is CNN the Answer?

Capsule?

End2End is not the right way?