Deep Learning Implementation & Acceleration - Introduction

February 20, 22 2023

Eunhyeok Park

Class Overview

- Goal:
  - Study the advanced optimization topics for deep learning applications
  - How to optimize power / performance / accuracy (PPA)?

- Understanding optimization techniques for CNNs
  - GPU and HW acceleration
  - Quantization, pruning, and low rank approximation
  - Conditional execution
  - Network architecture search
  - And other optimization methods

Lecture and Scoring

- No textbook
  - Slides will be provided via PLMS

- There will be several paper review classes
  - Lecture on key concepts (class1) → Paper review (by students) → Understanding the papers in more detail (class2)
  - Selected papers will be assigned as homework
  - Or you can select papers on your own if there is an interest or related paper

- Scoring
  - Mid-term (40%) and final exam (40%)
  - Paper Review, Q&A(10%)
  - Attendance (10%)

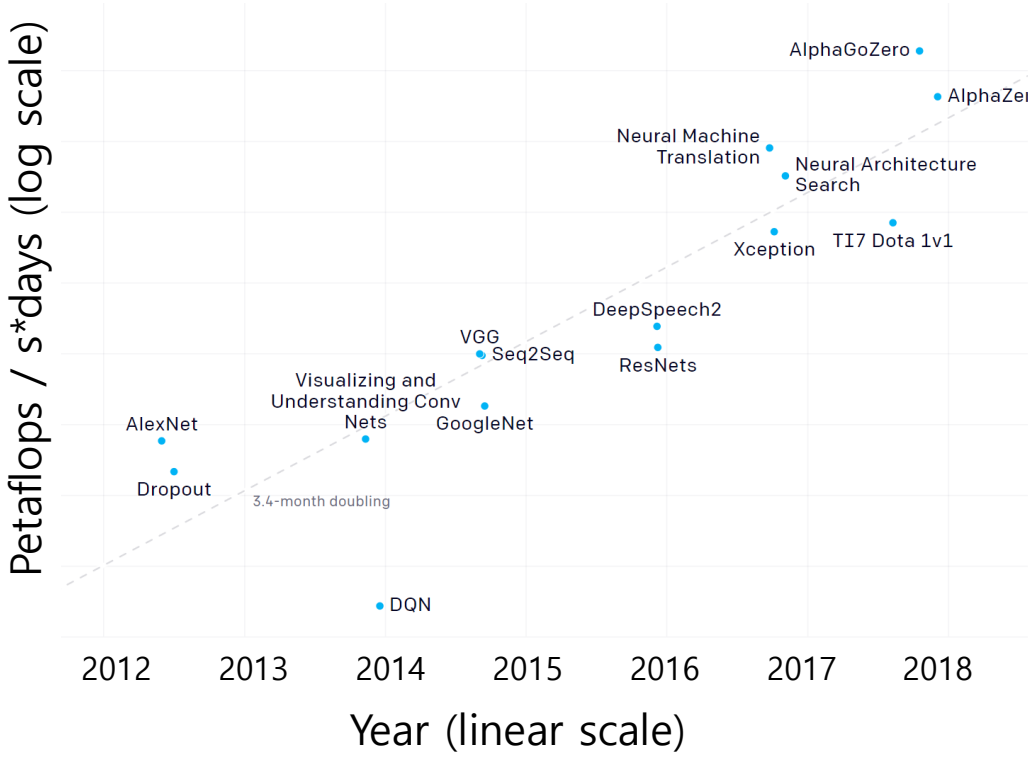
Schedule

- 1 Class introduction, Deep learning basics (convolutional neural network)
- 2 Representative CNNs
- 3 GPU architecture and CNN Acceleration
- 4 Object detection case analysis
- 5 Transfer learning and Online training
- 6 Selective & conditional execution
- 7 Introduction to reinforcement learning
- **8 Mid-term exam, Q&A for mid-term**
- 9 Neural Architecture search #1
- 10 Neural Architecture search #2
- 11 Quantization
- 12 Pruning
- 13 Network compression
- 14 Hardware accelerators for CNN #1
- 15 Hardware accelerators for CNN #2
- **16 Final exam , Q&A for final and conclusion**

Importance of Optimization

Importance of Efficient AI

The annual trend of the computation growth



We're releasing an analysis showing that since 2012, the amount of compute used in the largest AI training runs has been increasing exponentially with a 3.4-month doubling time (by comparison, Moore's Law had a 2-year doubling period).[1] Since 2012, this metric has grown by more than 300,000x (a 2-year doubling period would yield only a 7x increase). Improvements in compute have been a key component of AI progress, so as long as this trend continues, it's worth preparing for the implications of systems far outside today's capabilities.

Training the Model

GPT-3 is trained using next word prediction, just the same as its GPT-2 predecessor. To train models of different sizes, the batch size is increased according to number of parameters, while the learning rate is decreased accordingly. For example, GPT-3 125M use batch size 0.5M and learning rate of 6.0×10^{-4}, where GPT-3 175B uses batch size 3.2M and learning rate of 0.6×10^{-4}.

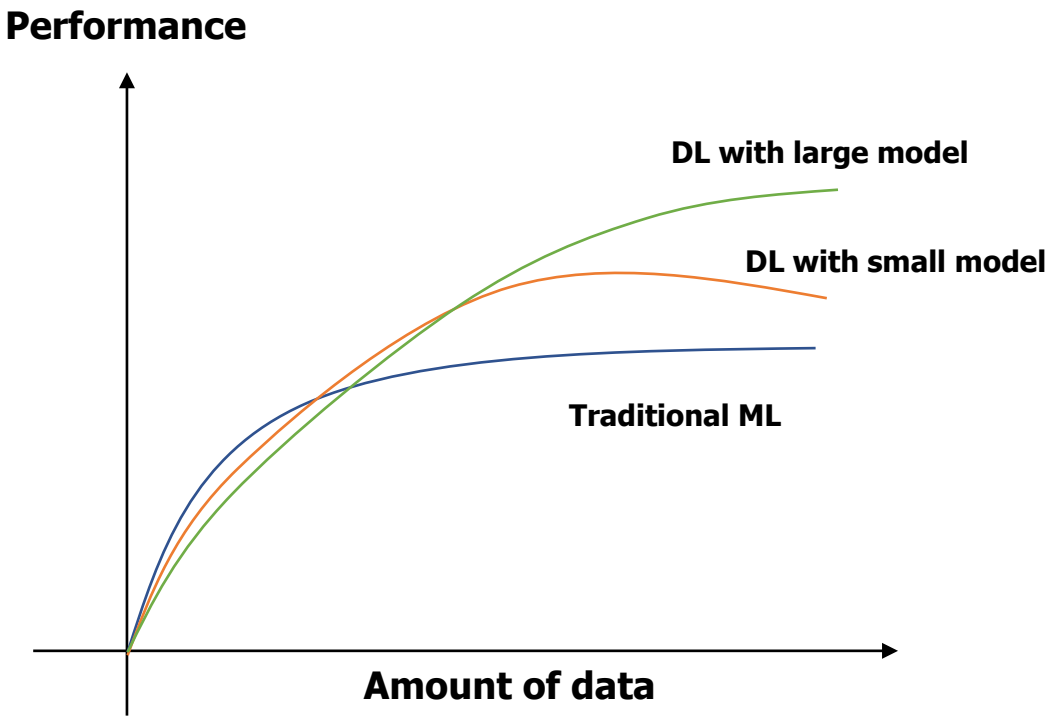
We are waiting for OpenAI to reveal more details about the training infrastructure and model implementation. But to put things into perspective, GPT-3 175B model required 3.14E23 FLOPS of computing for training. Even at theoretical 28 TFLOPS for V100 and lowest 3 year reserved cloud pricing we could find, this will take 355 GPU-years and cost $4.6M for a single training run. Similarly, a single RTX 8000, assuming 15 TFLOPS, would take 665 years to run.

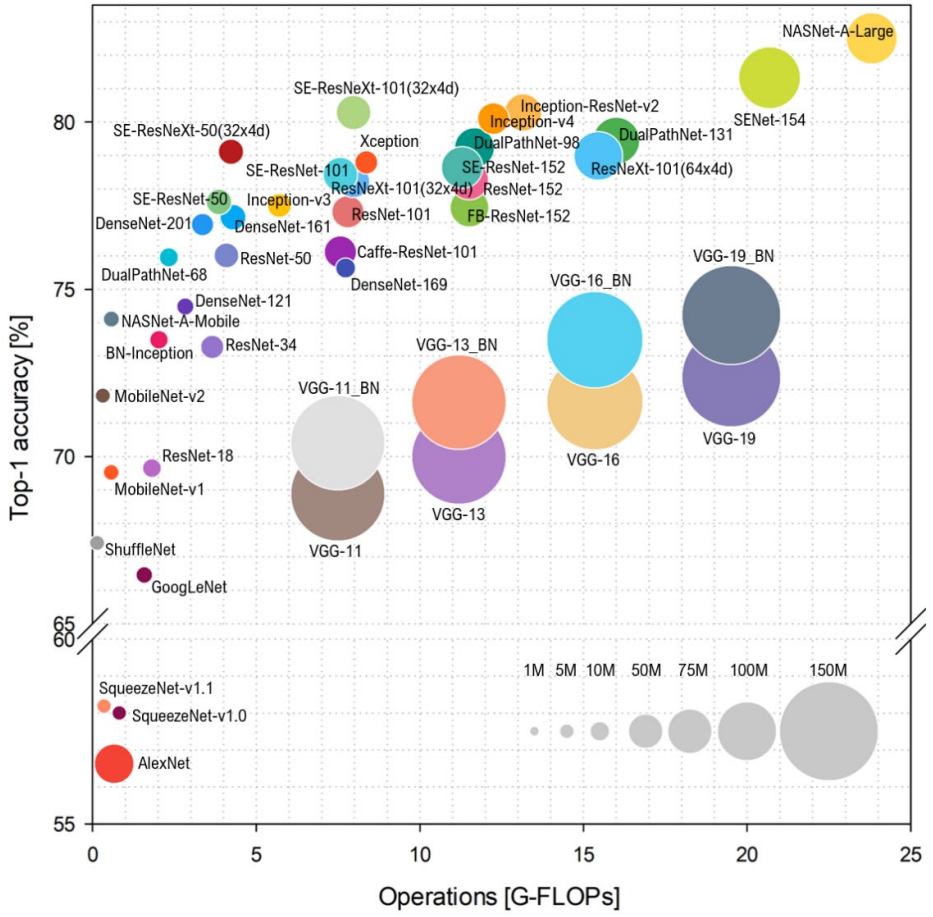
Time is not the only enemy. The 175 Billion parameters needs 175 × 4 = 700GB memory to store in FP32 (each parameter needs 4 Bytes). This is one order of magnitude larger than the maximum memory in a single GPU (48 GB of Quadro RTX 8000). To train the larger models without running out of memory, the OpenAI team uses a mixture of model parallelism within each matrix multiply and model parallelism across the layers of the network. All models were trained on V100 GPU's on the part of a high-bandwidth cluster provided by Microsoft.

In fact, The size of SOTA language model increases by at least a factor of 10 every year: BERT-Large (2018) has 355M parameters, GPT-2 (early 2019) reaches 1.5B, T5 (late 2019) further streches to 11B, GPT-3 (mid-2020) finally gets to 175B. The progress of the sizes of language models clearly outpace the growth of GPU memory. This implies that for NLP, the days of "embarrassingly parallel" is coming to the end, and model parallelization is going to be indispensable for researching SOTA language models.

https://openai.com/blog/ai-and-compute/

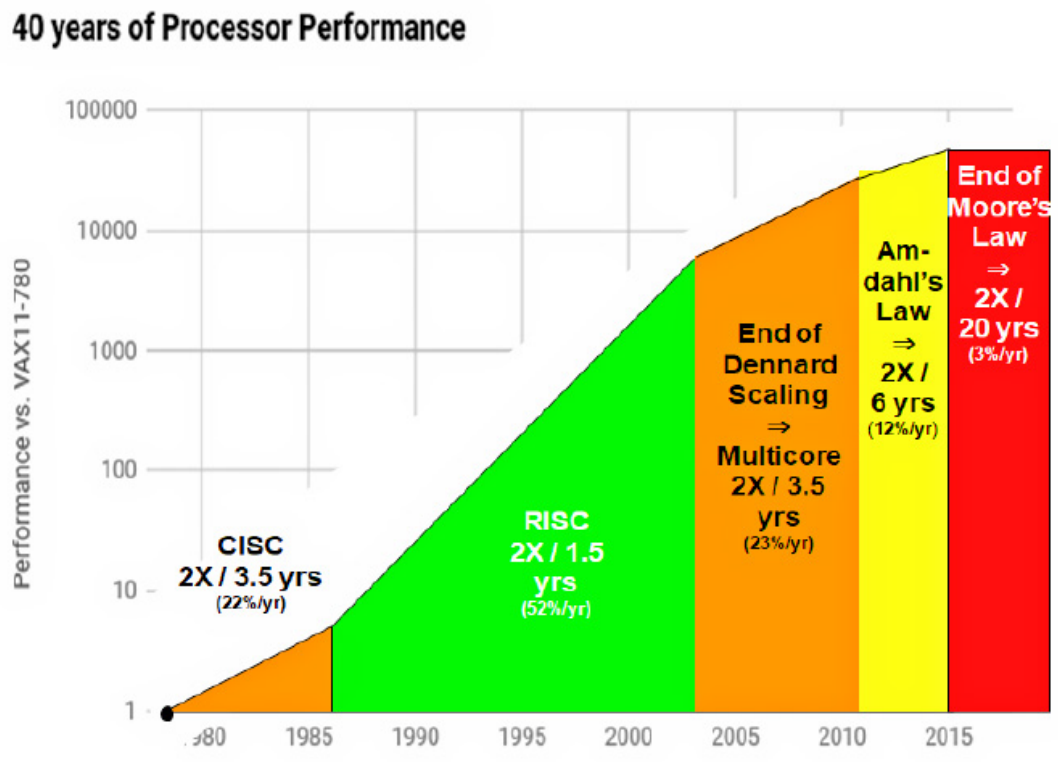
Model sizes tend to increase

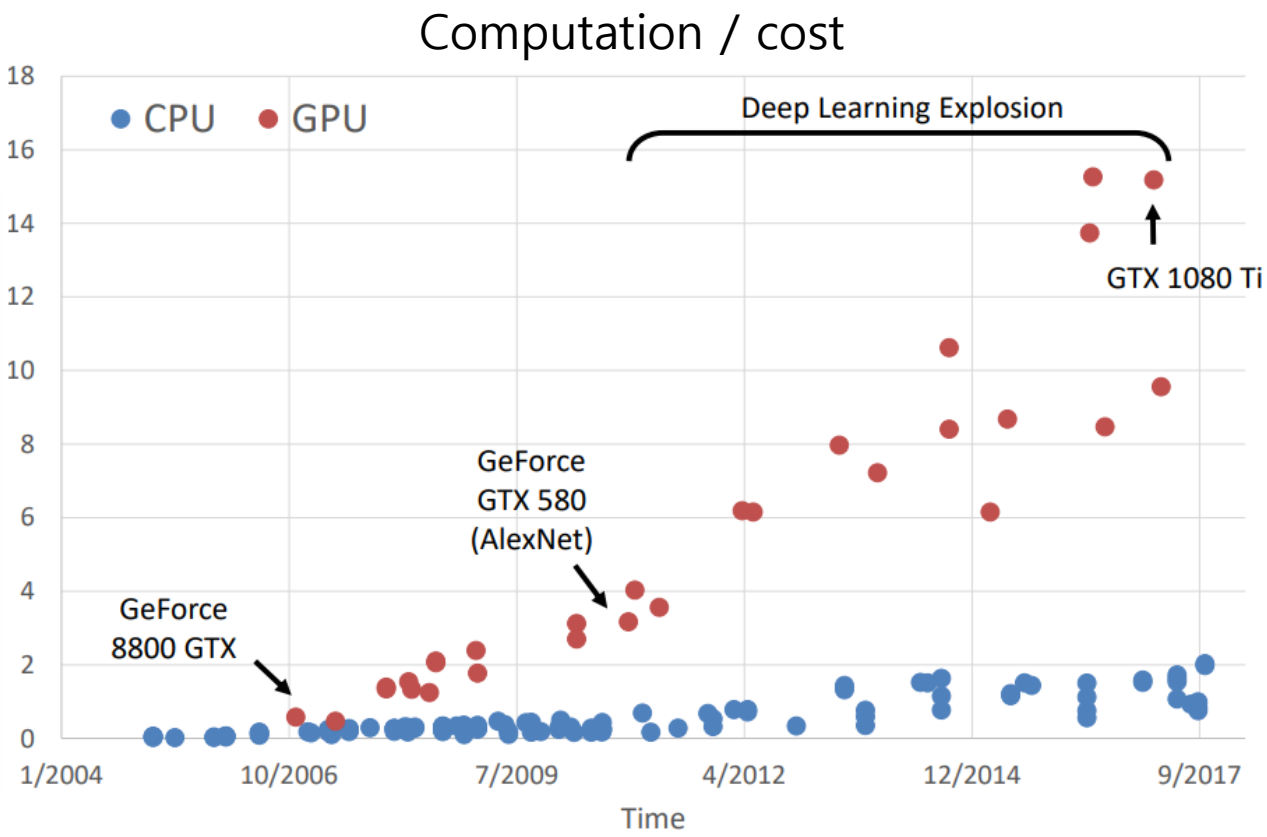




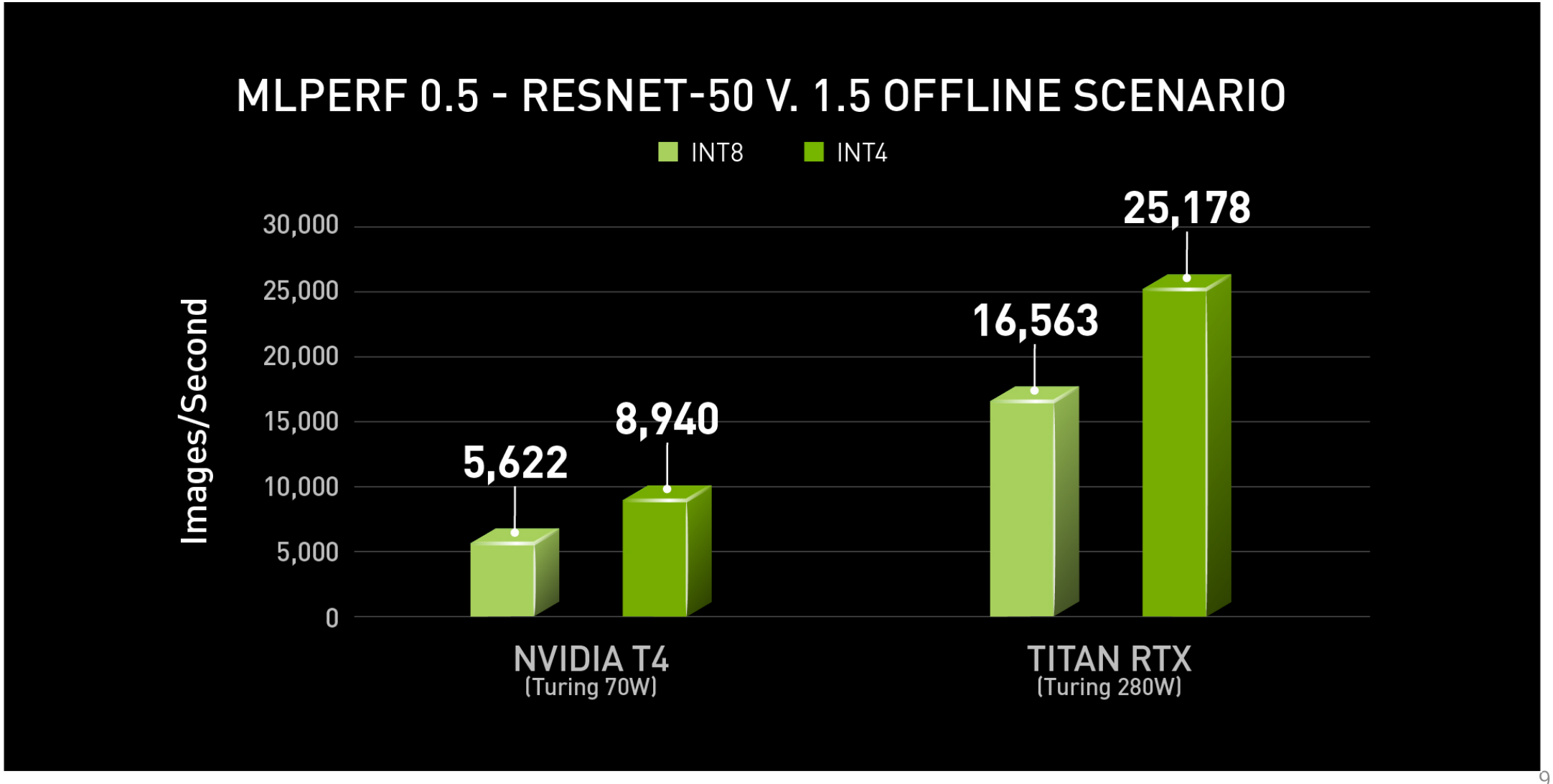
Benchmark Analysis of Representative Deep Neural Network Architectures

Limitation of HW Performance





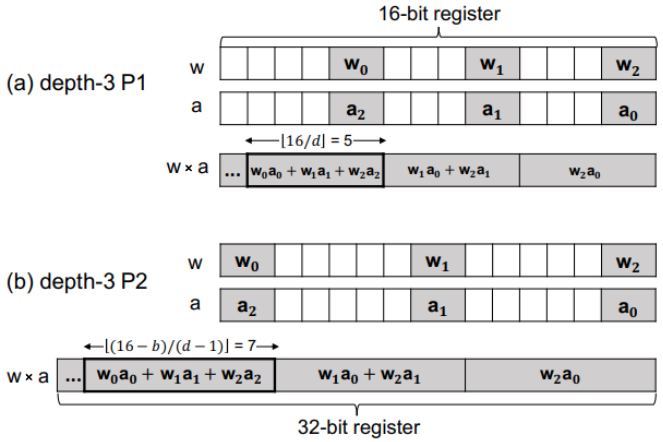
Benefit of Quantization on Real Hardware



https://devblogs.nvidia.com/int4-for-ai-inference/

Benefit of Quantization on Real Hardware - 2

- Performance benefit of ULPPack



Model	FP32	W8A8	W4A4	W3A3	W2A2
ResNet18	1.7	2.6	3.4	3.9	4.0
ResNet34	0.9	1.3	1.8	2.1	2.2
ResNet50	0.7	1.2	1.6	1.8	1.9
InceptionV3	1.0	1.6	2.2	2.5	2.6
GoogleNet	1.8	2.8	3.5	3.9	4.0
ShuffleNetV2	2.2	14.3	14.9	15.9	16.4
MobileNetV2	0.8	11.1	12.8	13.7	14.3
Speedup/FP32	1.0×	2.7×	3.3×	3.7×	3.9×
Speedup/W8A8	-	1.0×	1.2×	1.4×	1.5×

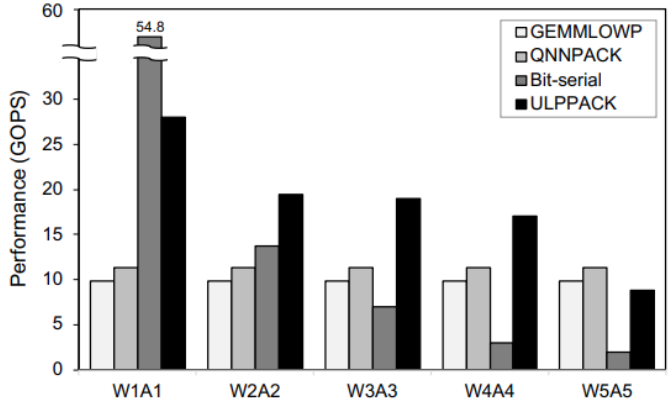
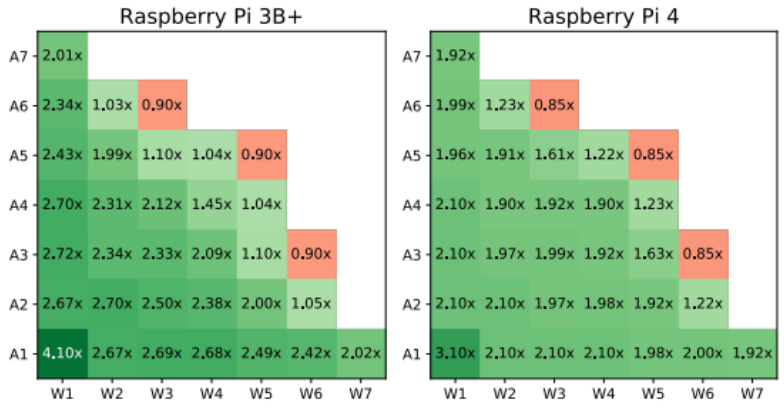
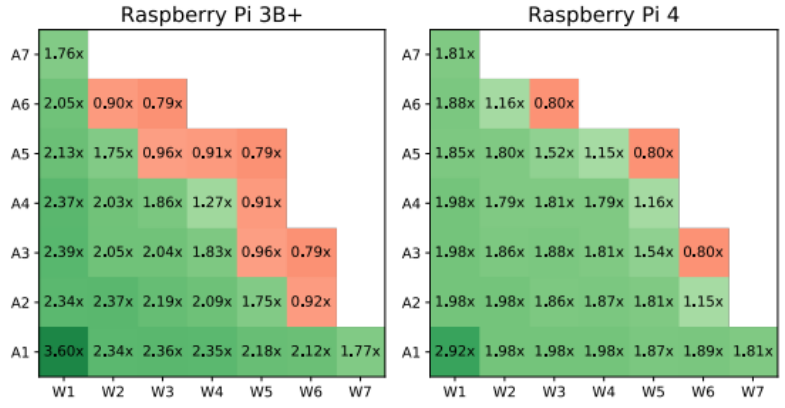


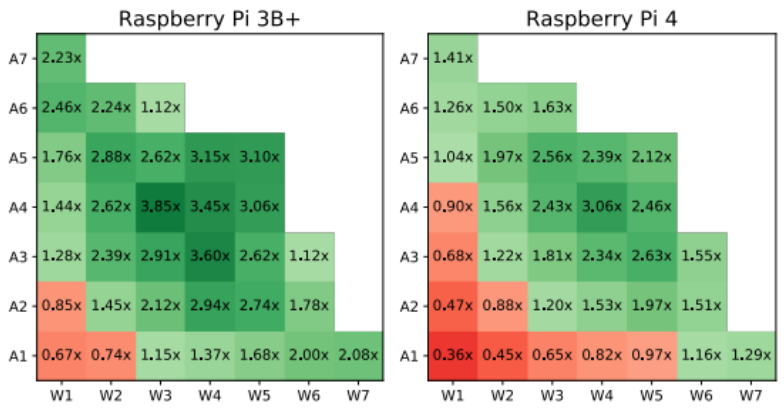
Figure 9. Throughput of 512×512×512 matrix multiplication across varying bit-widths.



(a) Speedups over GEMMLOWP



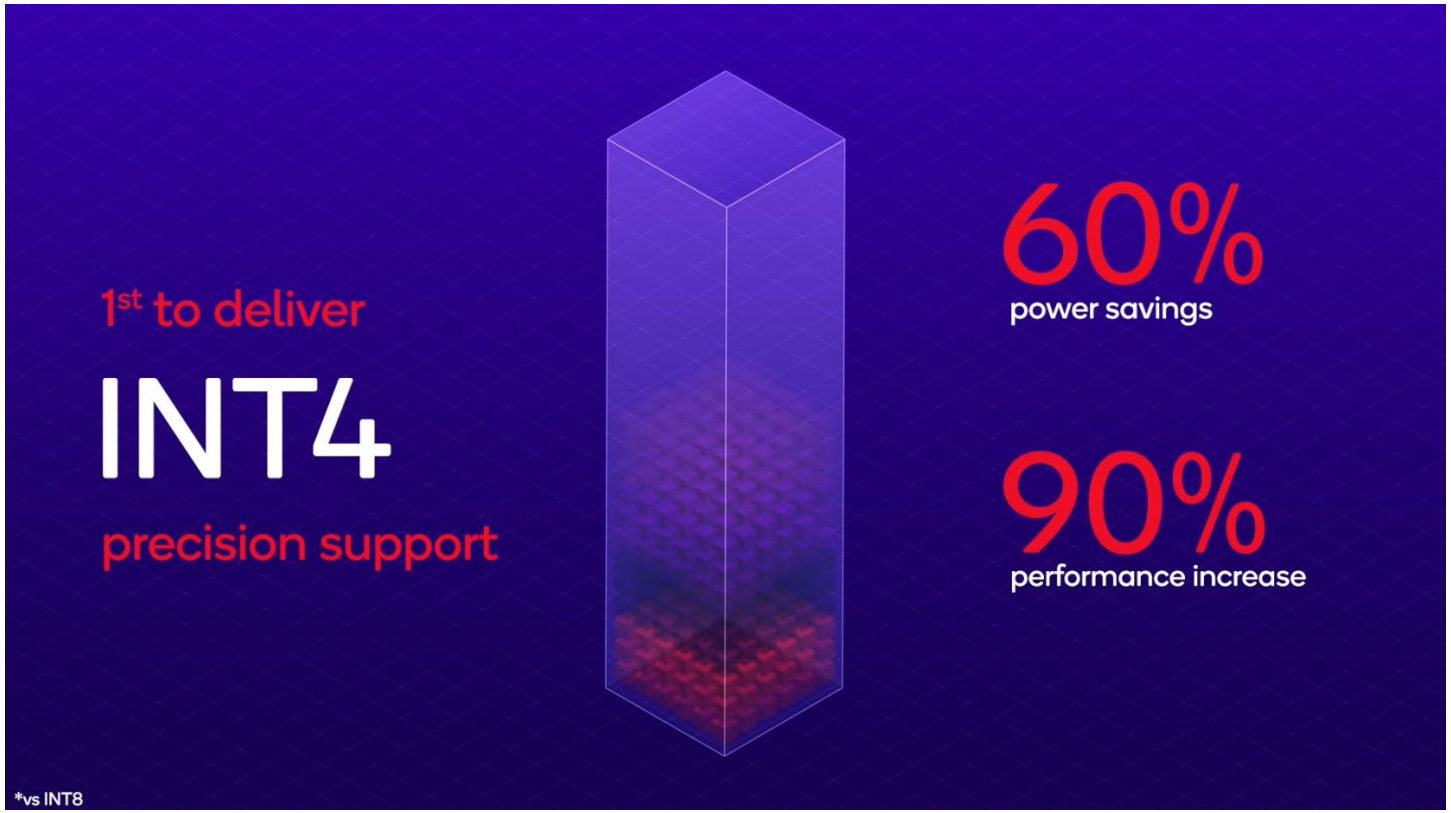
(b) Speedups over QNNPACK



(c) Speedups over Bit-serial

Benefit of Quantization on Real Hardware - 3

- Snapdragon 8 Gen 2

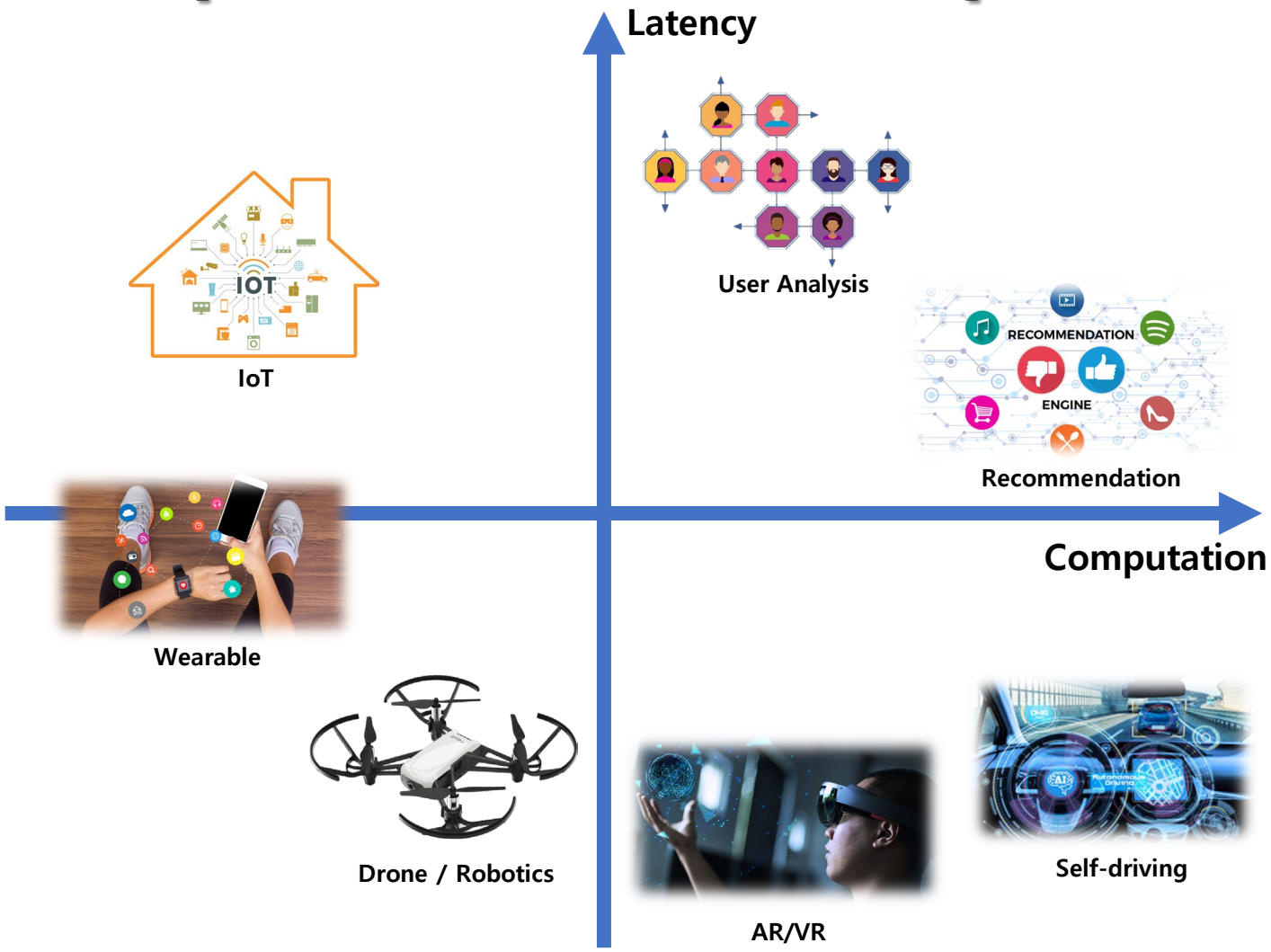


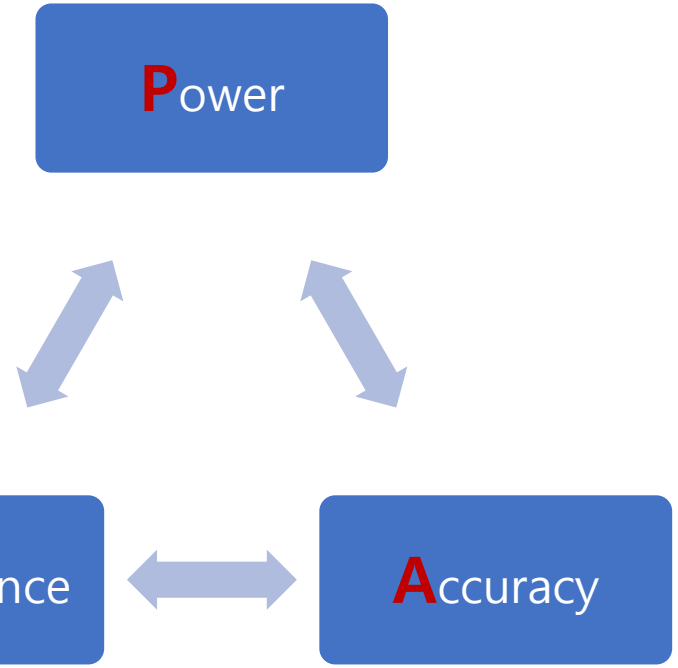
Groundbreaking AI

Our fastest, most advanced Qualcomm® AI Engine ever puts a world of possibilities at your fingertips. Communicate with friends or colleagues across the globe with multi-language translation and transcription, or capture premium content with AI Cinematic video. And now, the Qualcomm® Sensing Hub features dual-AI processors for the first time, powering exciting new experiences like direct-to-app voice assistance for convenient control of your favorite apps.

- Up to 4.35x faster AI performance[1]
- First Snapdragon® Mobile Platform with INT4 precision support for 60% better performance/watt
- First Snapdragon Mobile Platform with Micro Tile Inferencing

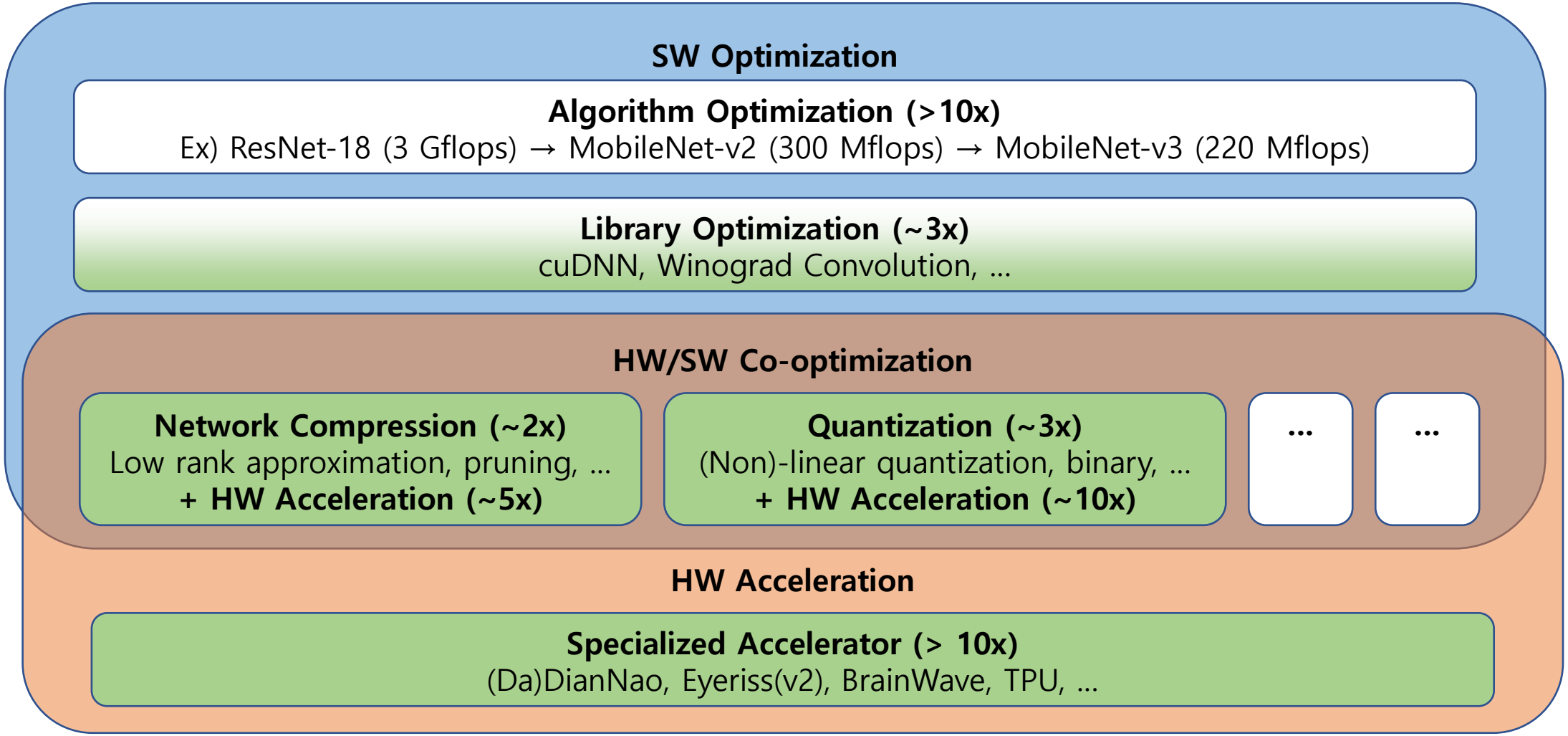
Optimization of Deep Learning





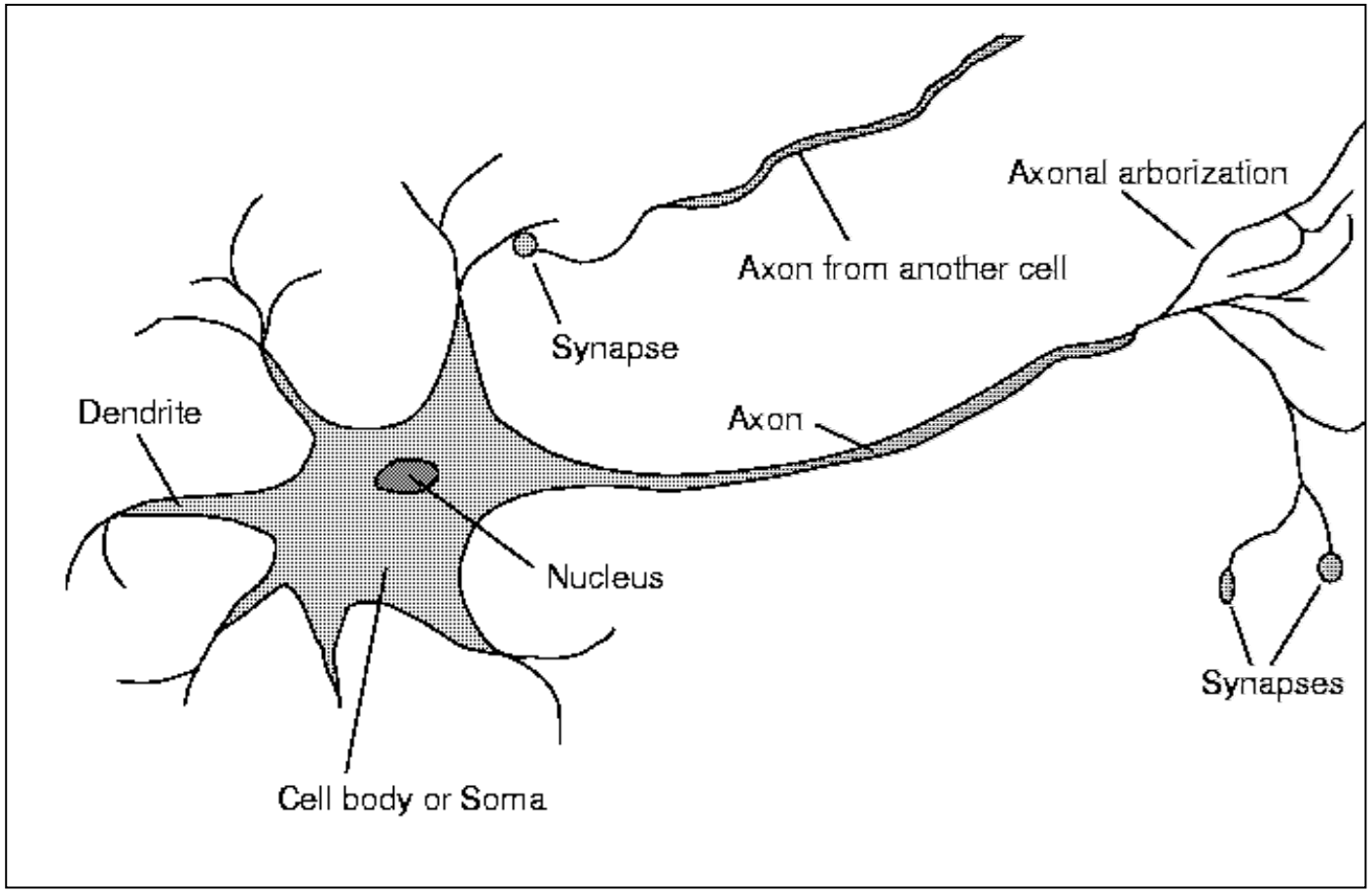
Optimizing PPA of AI considering
the characteristics of application and device

Optimization Methods



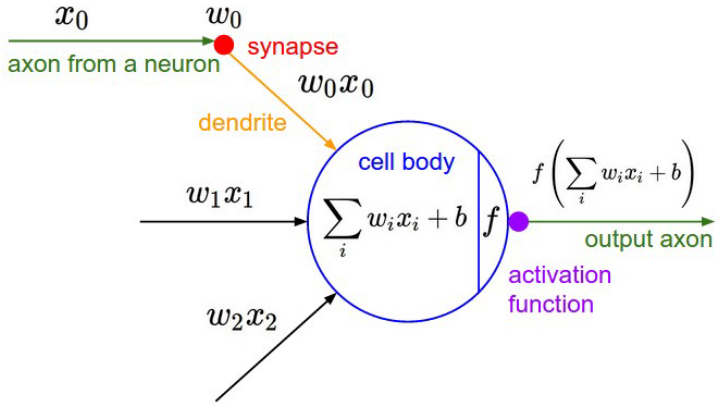
Basics of Convolution Neural Network

Neural Networks



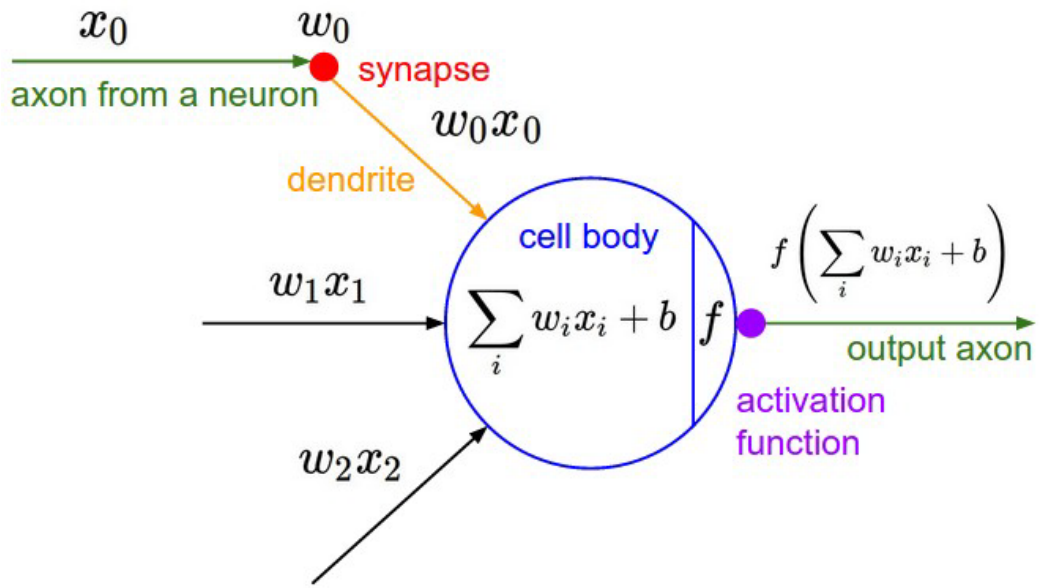
Human brain
- ~100 billion neurons (10^{11})
- ~100,000 connections / neuron

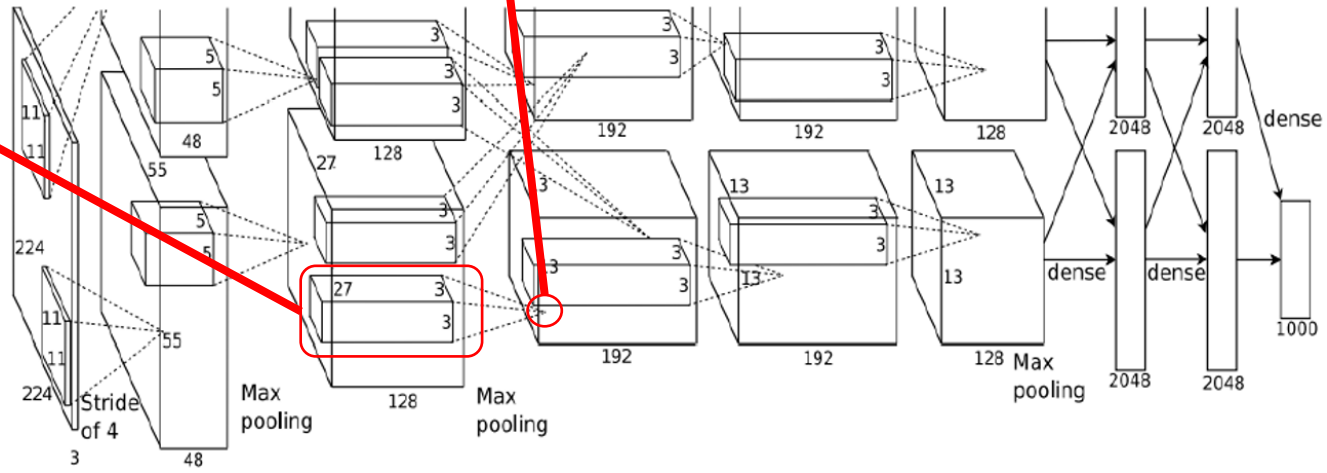
McCulloch and Pitts introduced first artificial neural network in 1943



[cs231n]
[Graham Kendall]

Neuron in Neural Network

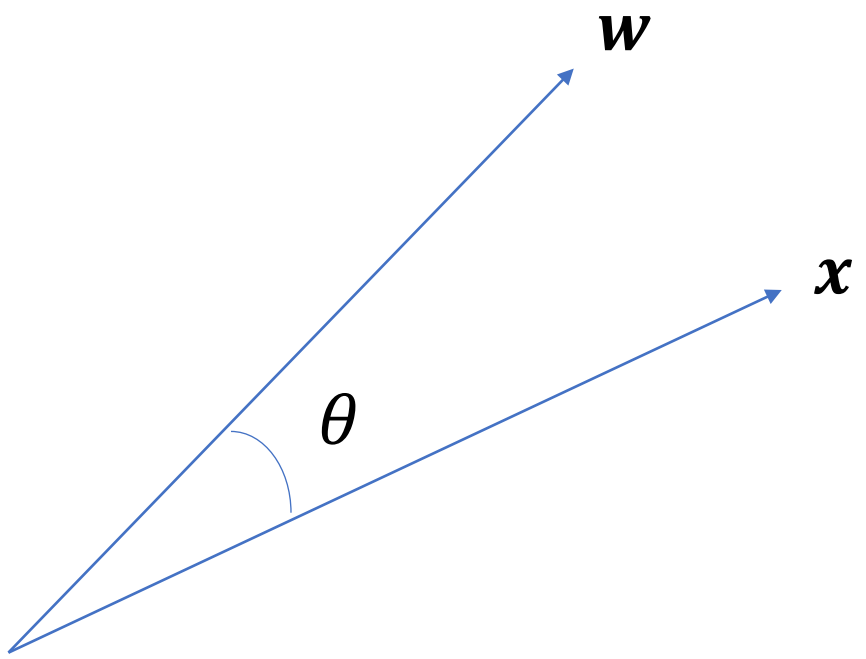




Neural Network = Feature Extractor

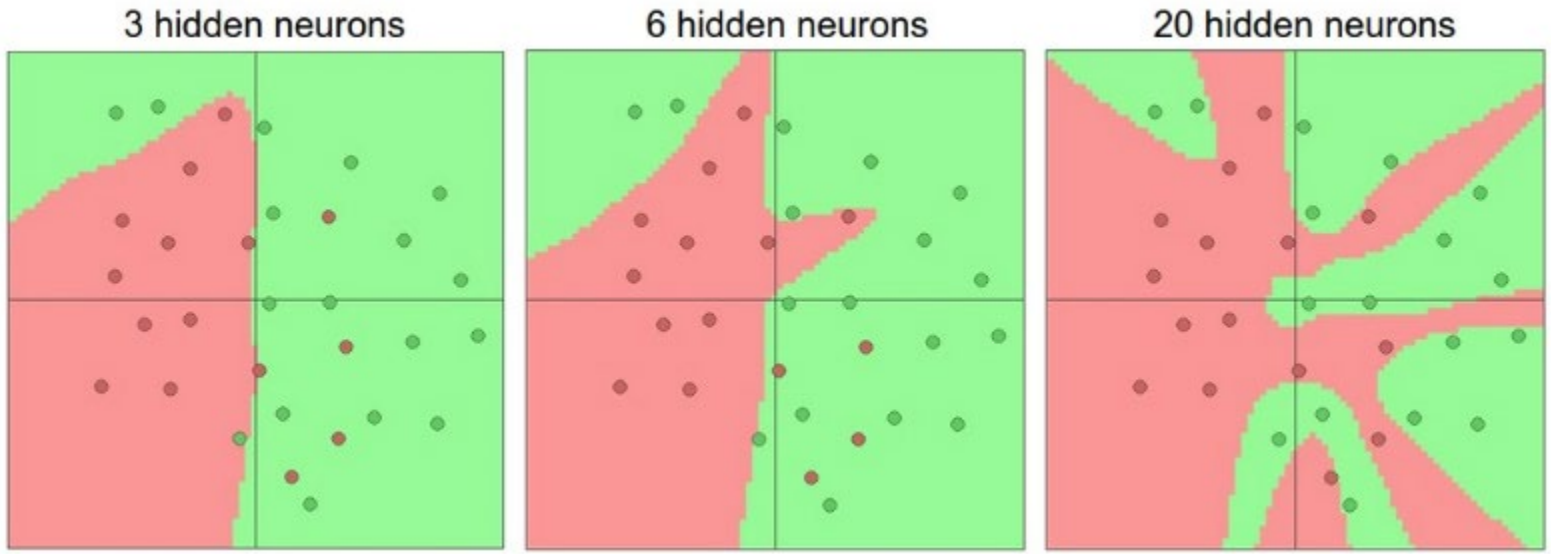
- Weighted sum of activation and weight can be seen as cosine similarity with scaling

- $Cosine\ similarity = \cos(\theta) = \frac{A \cdot B}{||A||||B||}$
- $v = w \cdot x = ||w||||x|| \cos(\theta)$



Representation Power ~ Network Complexity

- Larger, especially, deeper network
  can give better representation power

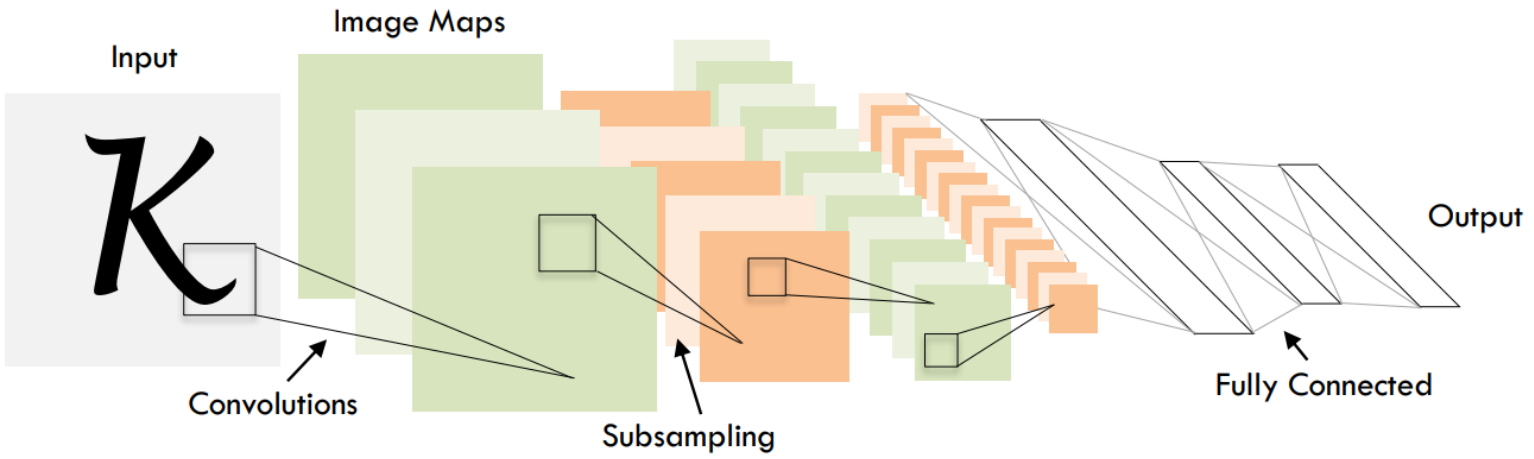


[CS231n]
[Sunjoo Yoo]

Convolutional Neural Network (CNN)

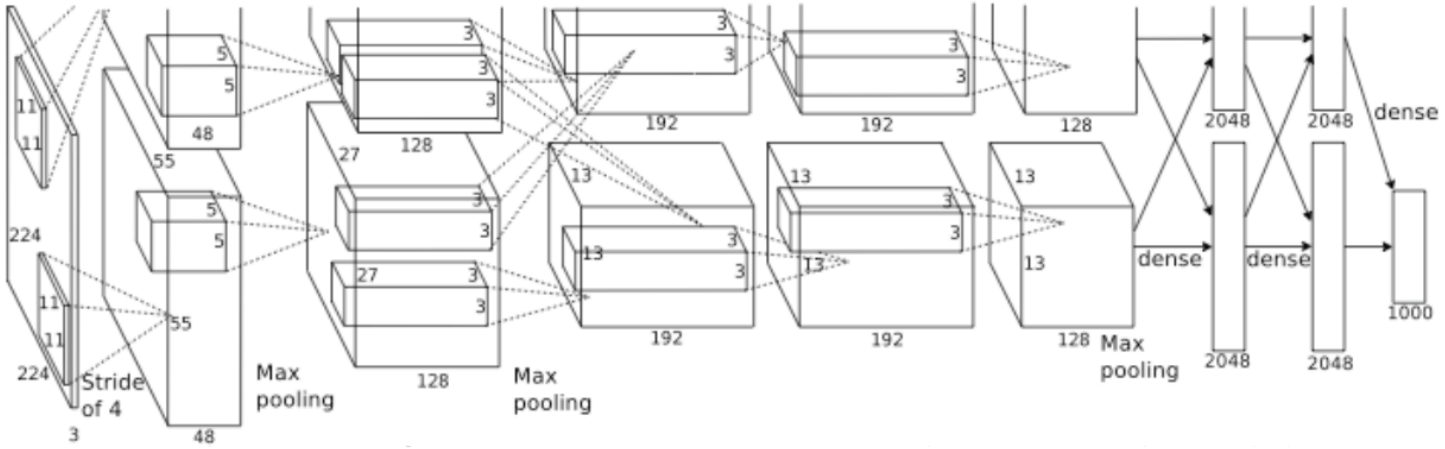
1998

LeCun et al.



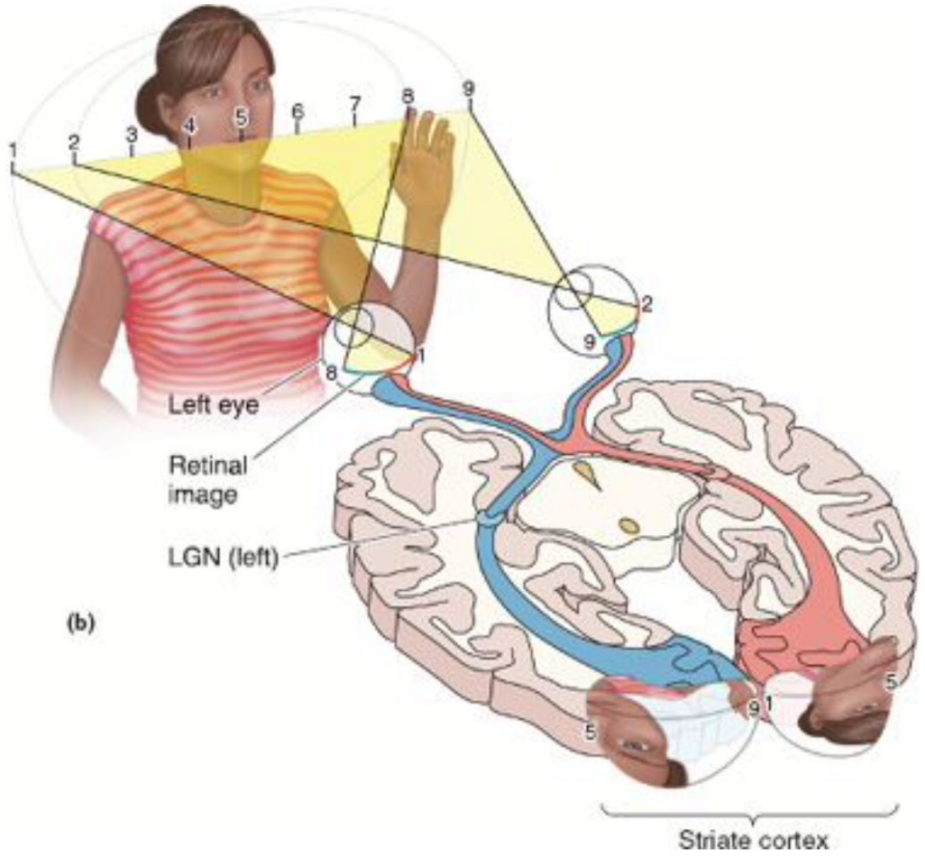
2012

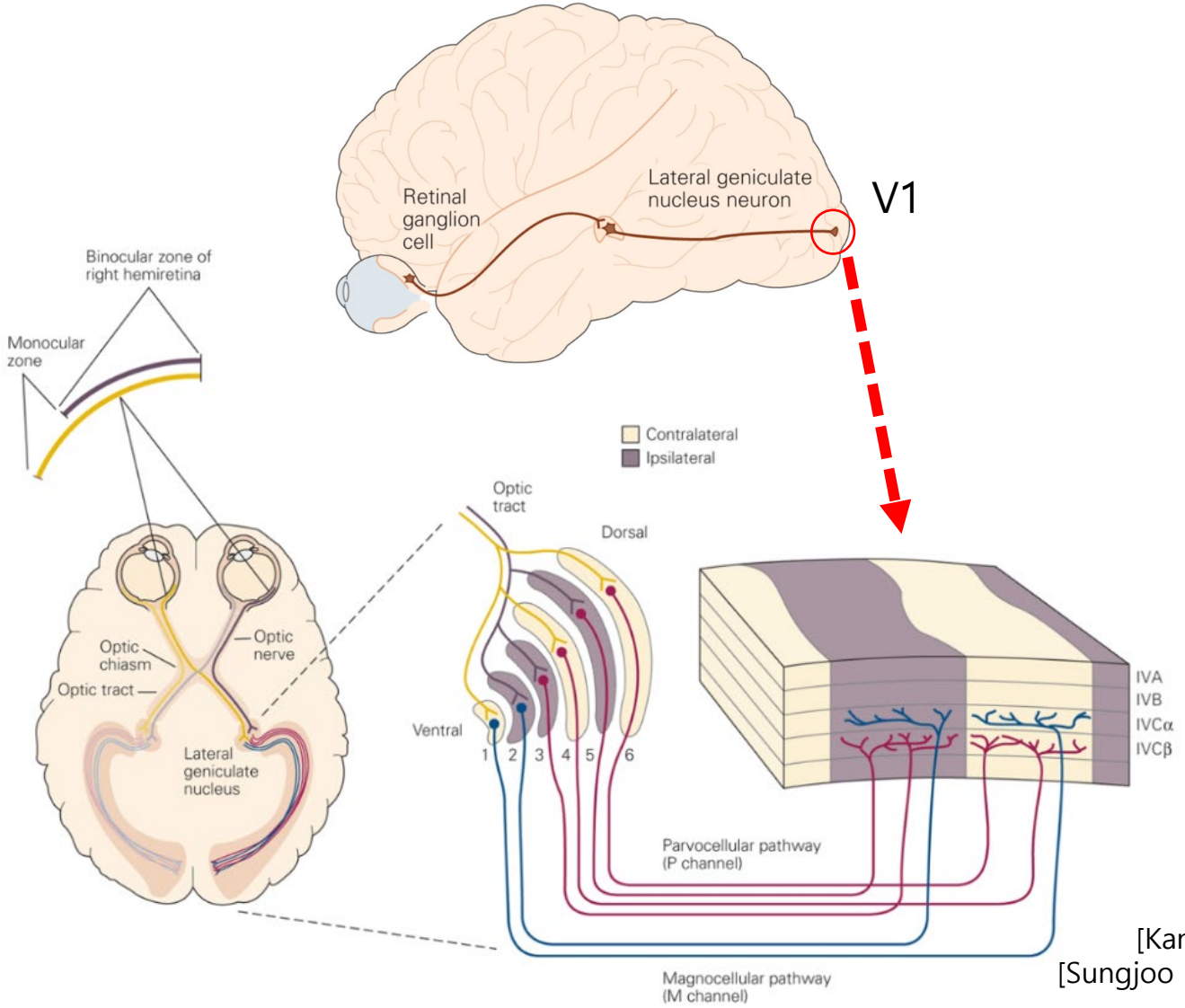
Krizhevsky et al.



[CS231n]

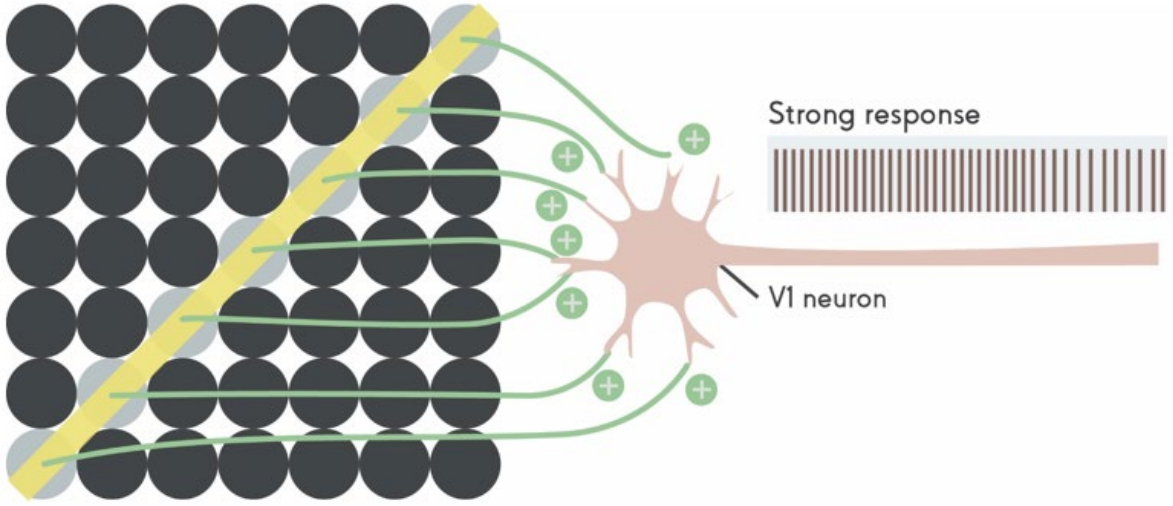
Retina → LGN → Primary Visual Cortex (V1)

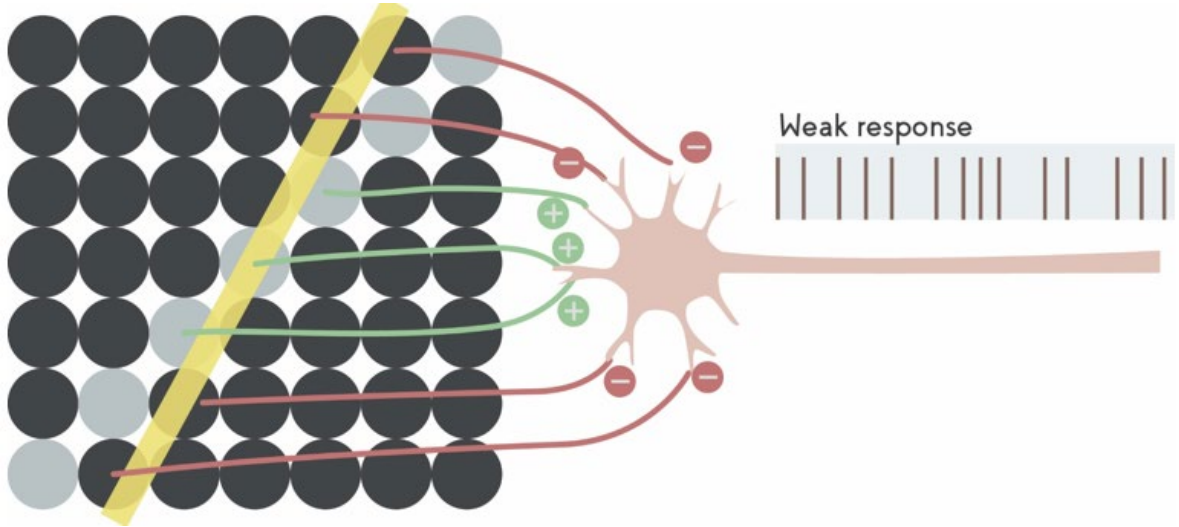


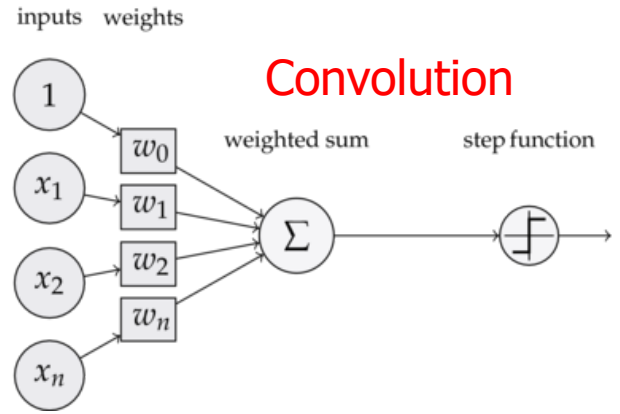


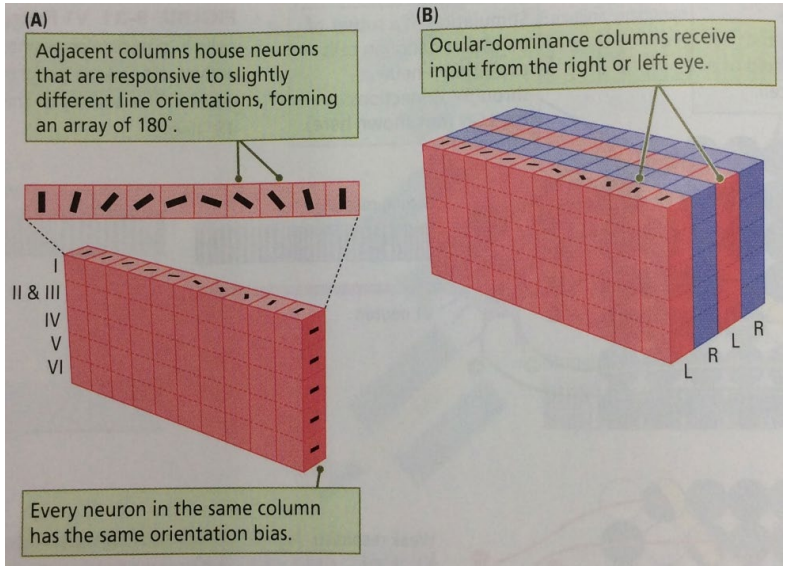
[Kandel]
[Sungjoo Yoo]

Line Detection in V1 ~ Convolution



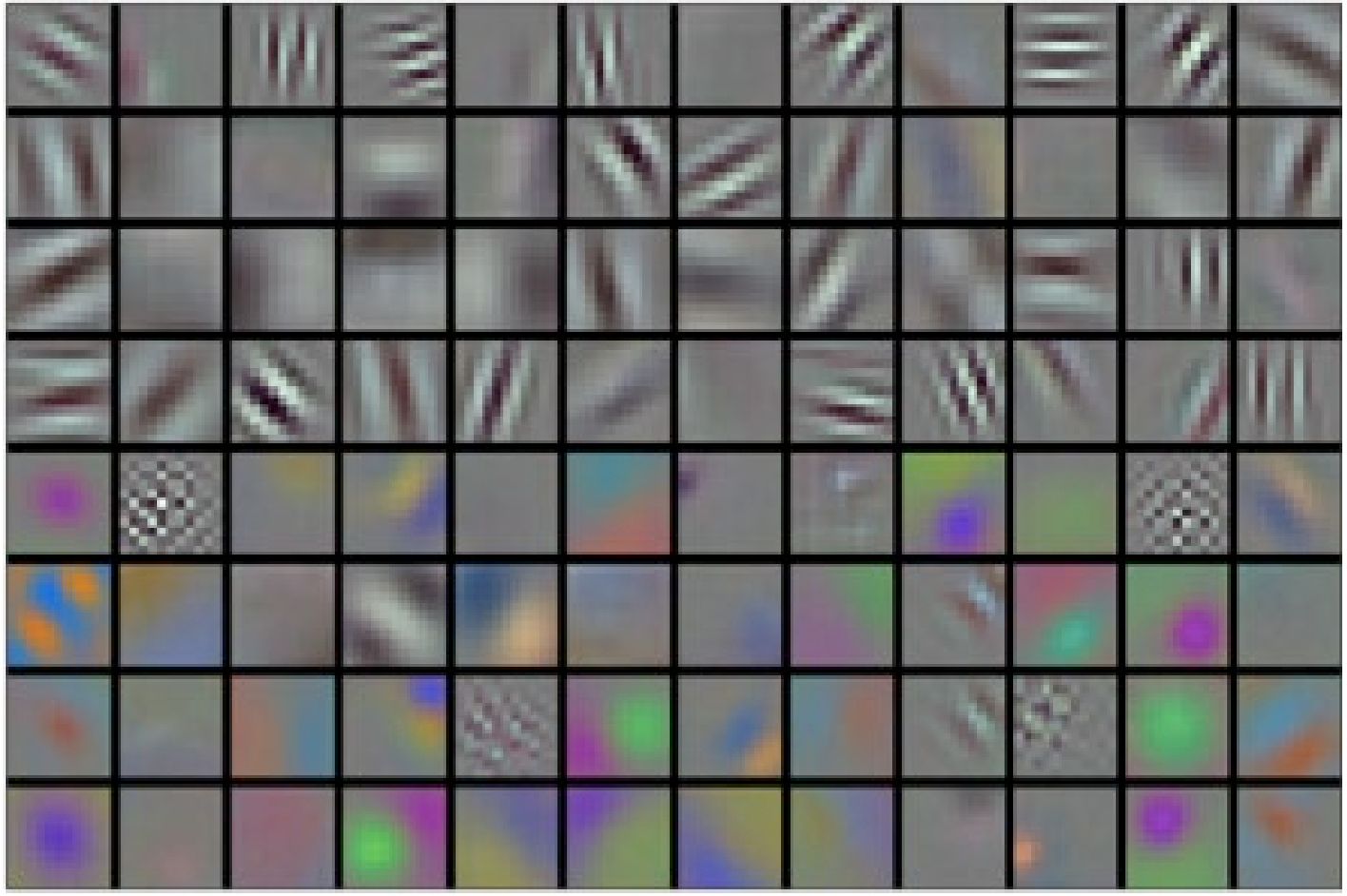






[Kolb_Whishaw]
[Sungjoo Yoo]

What CNN learns

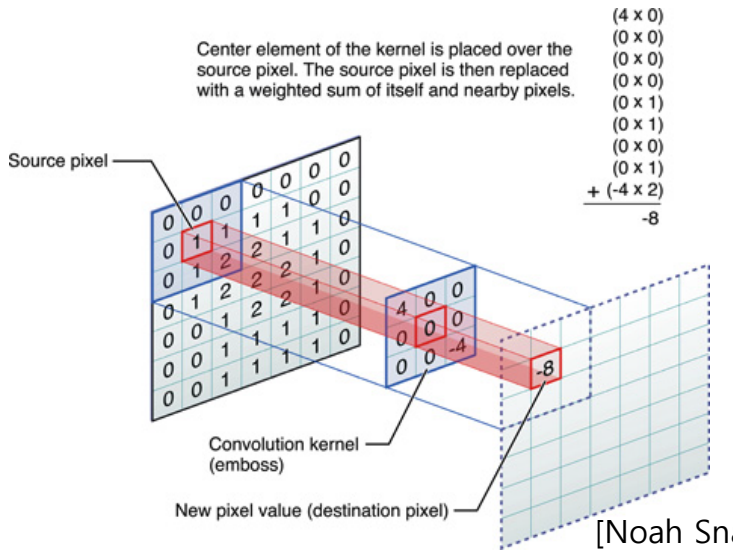


Convolution

- Convolution is **commutative** and **associative**

$$G = H * F$$

$$G[i,j] = \sum_{u=-k}^{k} \sum_{v=-k}^{k} H[u,v]F[i-u, j-v]$$



[Noah Snavely]

Linear filters: examples

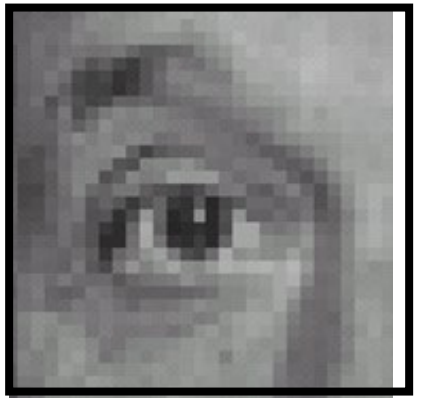
Original

$$*\begin{bmatrix}0&0&0\\0&1&0\\0&0&0\end{bmatrix}=$$

Identical image

Original

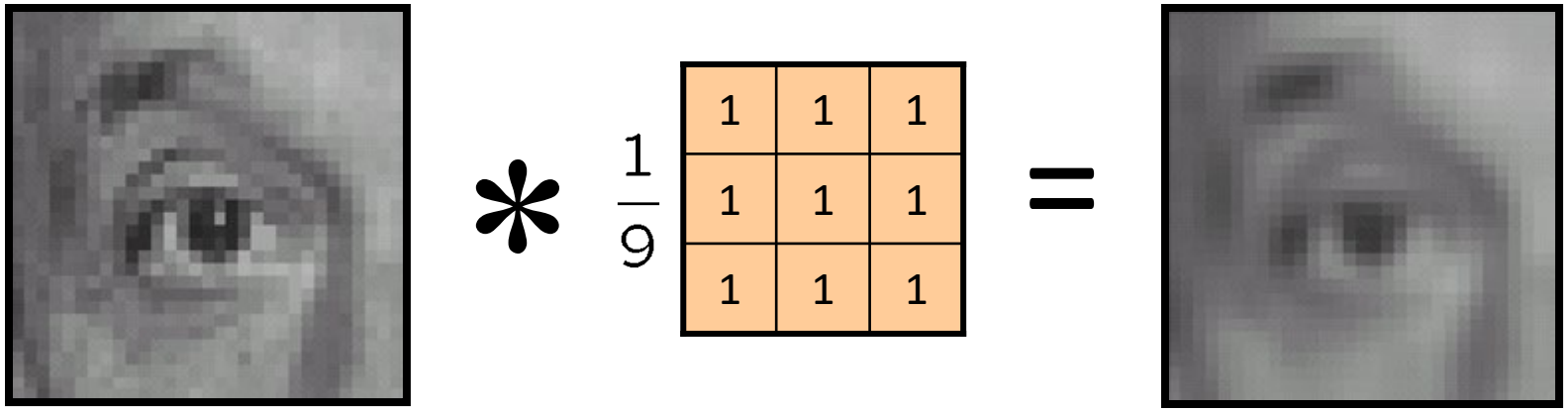
$$*\begin{bmatrix}0&0&0\\1&0&0\\0&0&0\end{bmatrix}=$$



Shifted left by 1 pixel

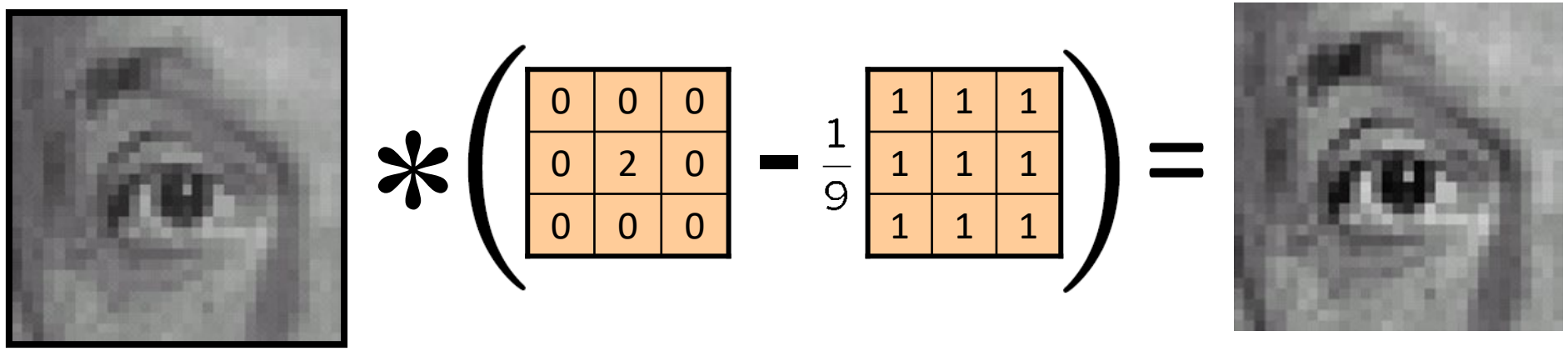
[D. Lowe]

Linear filters: examples



Original

Blur (with a mean filter)

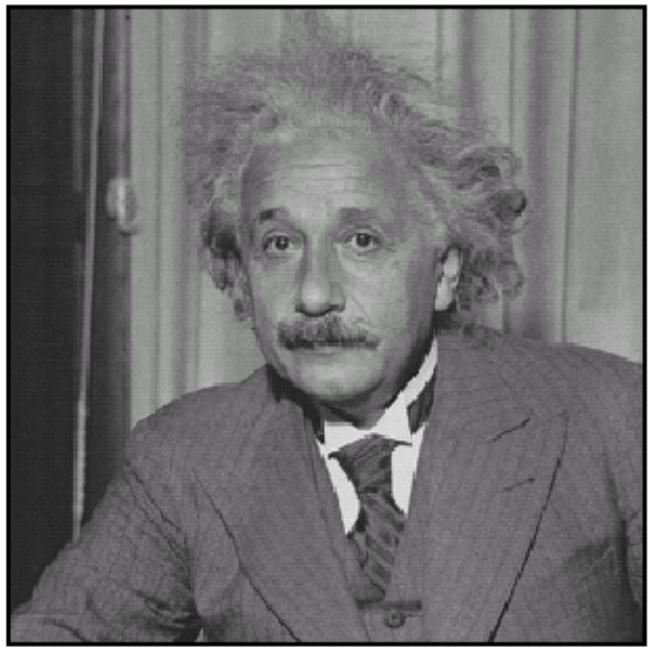


Original

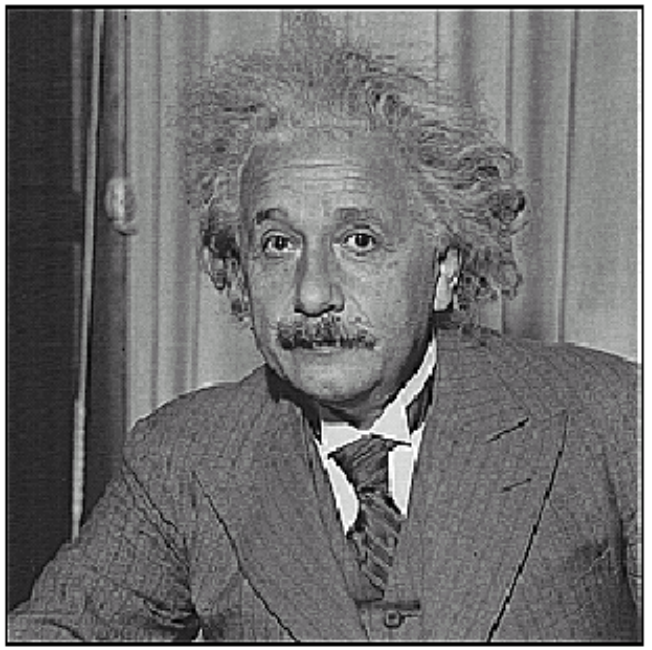
Sharpening filter
(accentuates edges)

[D. Lowe]

Sharpening



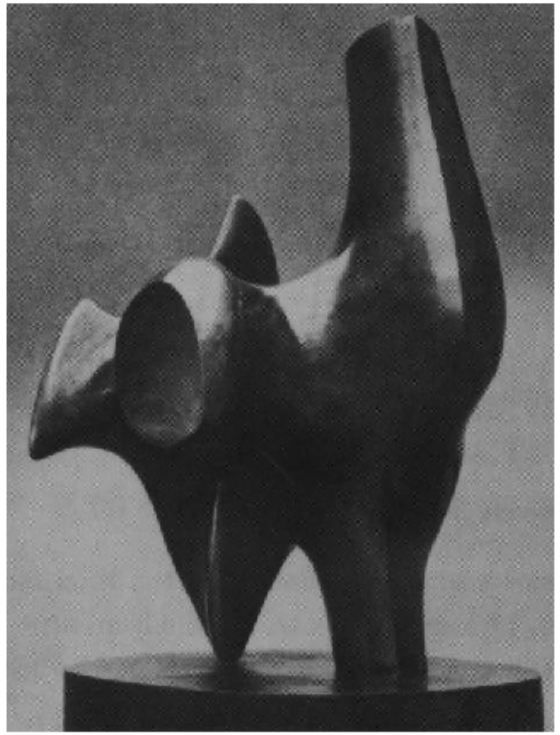
before

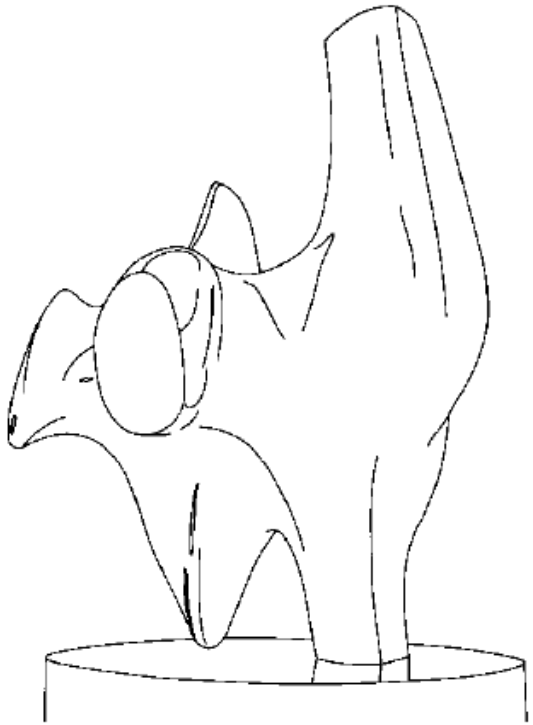


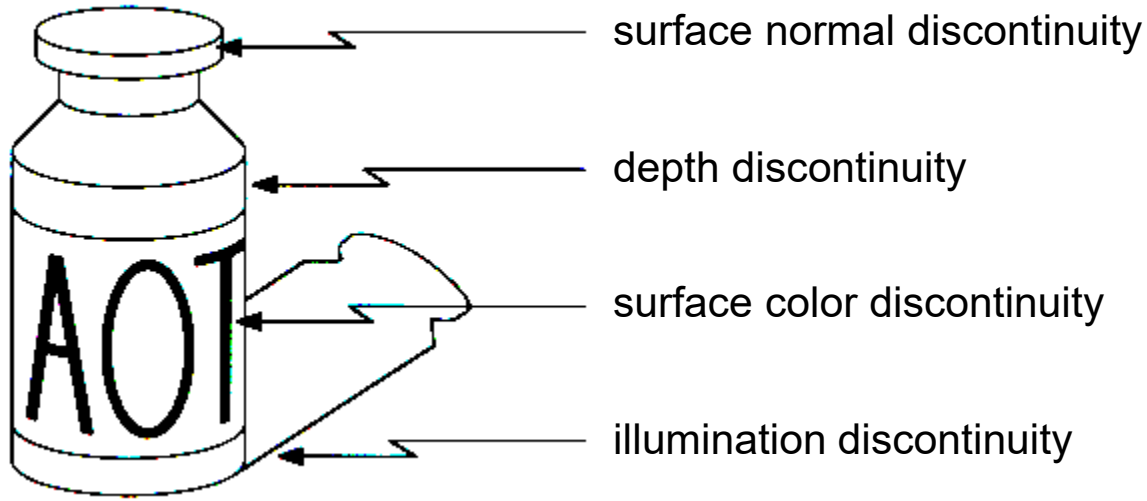
after

[D. Lowe]

Edge detection



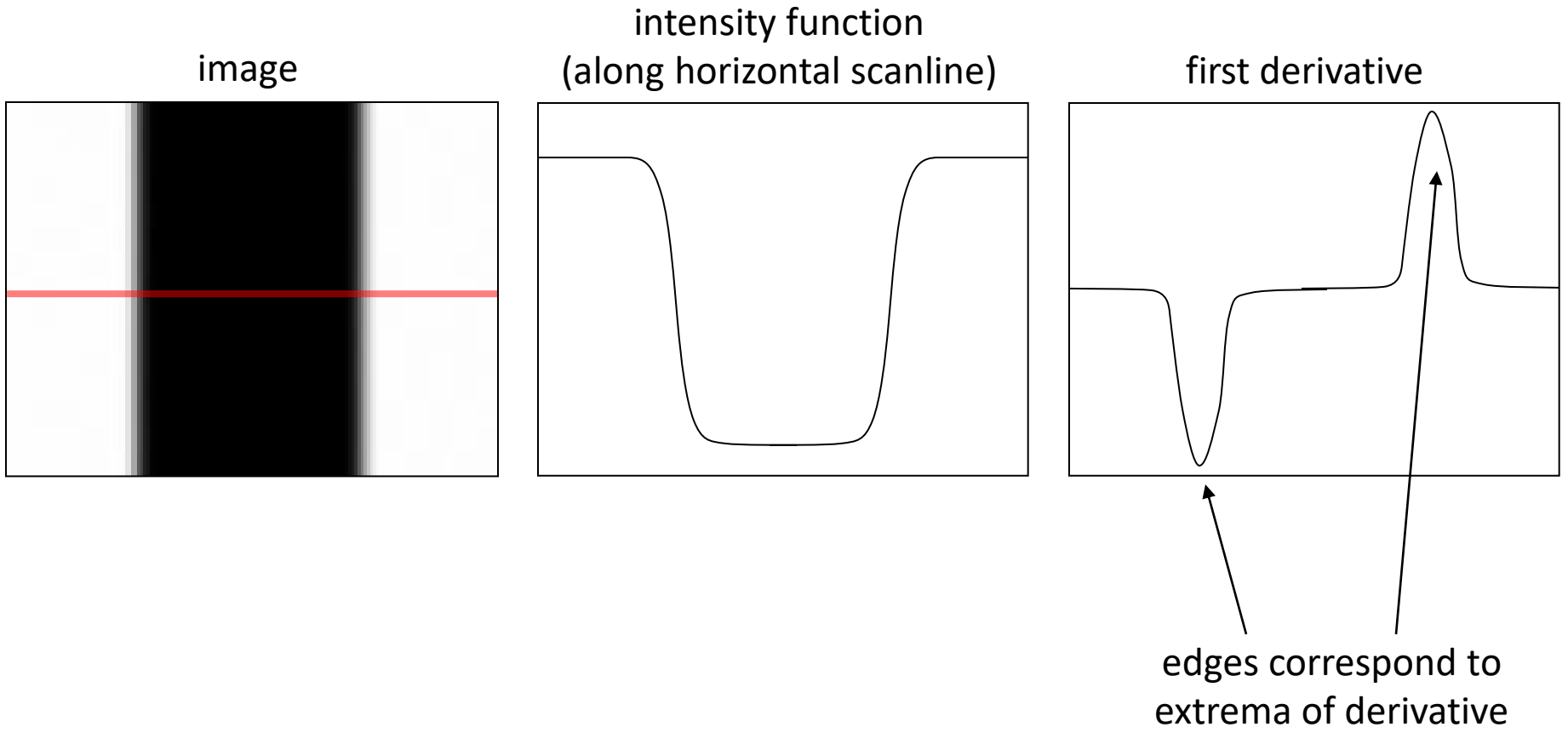




[Noah Snavely]

Characterizing edges

- An edge is a place of rapid change in the image intensity function



[L. Lazebnik]
[Noah Snavely]

Image derivatives

- How can we differentiate a *digital* image F[x,y]?
  - Option 1: reconstruct a continuous image, *f*, then compute the derivative
  - Option 2: take discrete derivative (finite difference)

$$\frac{\partial f}{\partial x}[x, y] \approx F[x+1, y] - F[x, y]$$

How would you implement this as a linear filter?

$\frac{\partial f}{\partial x}$:

1	-1	

H_x

$\frac{\partial f}{\partial y}$:

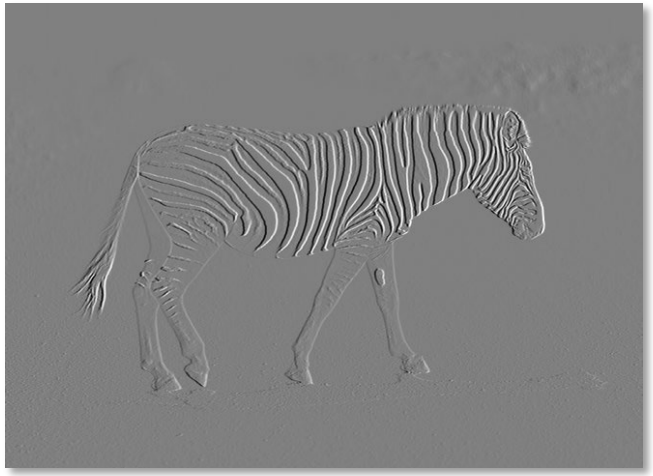
	-1	
	1	

H_y

[S. Seitz]
[Noah Snavely]

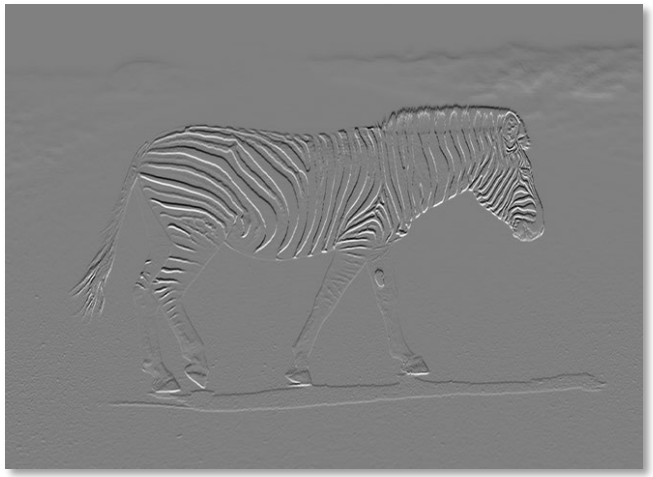
Image gradient

f



$\frac{\partial f}{\partial x}$

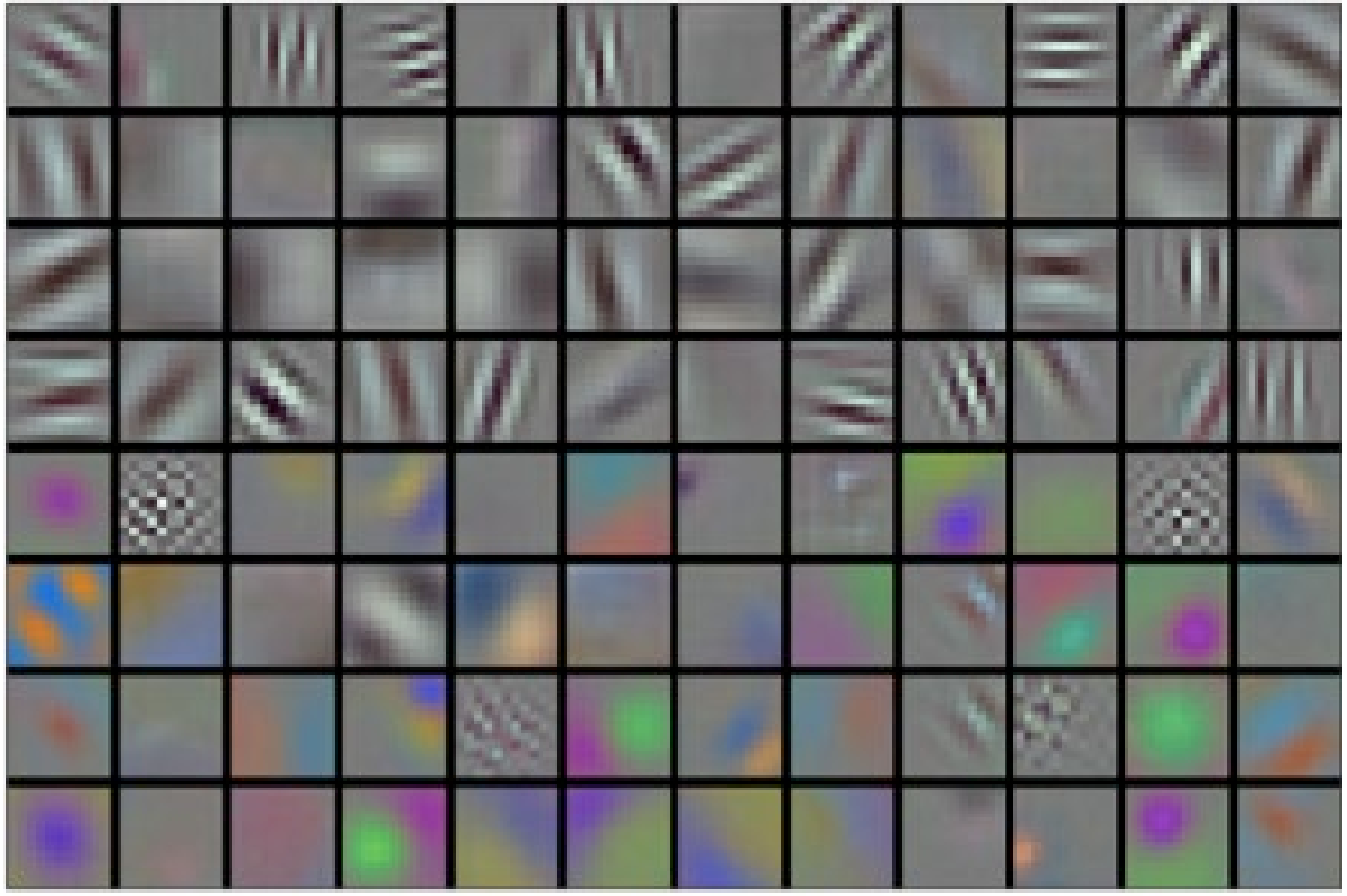
$\|\nabla f\| = \sqrt{(\frac{\partial f}{\partial x})^2 + (\frac{\partial f}{\partial y})^2}$



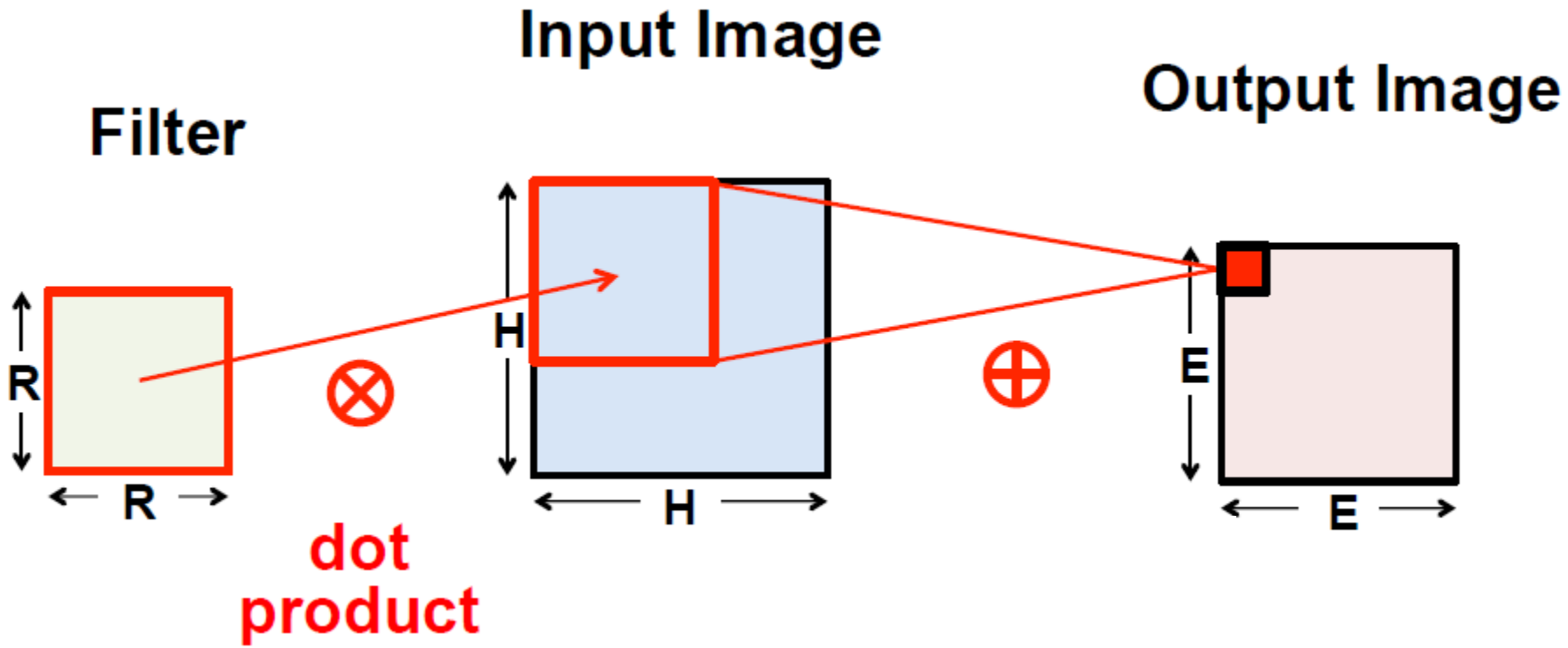
$\frac{\partial f}{\partial y}$

[Noah Snavely]

What CNN learns



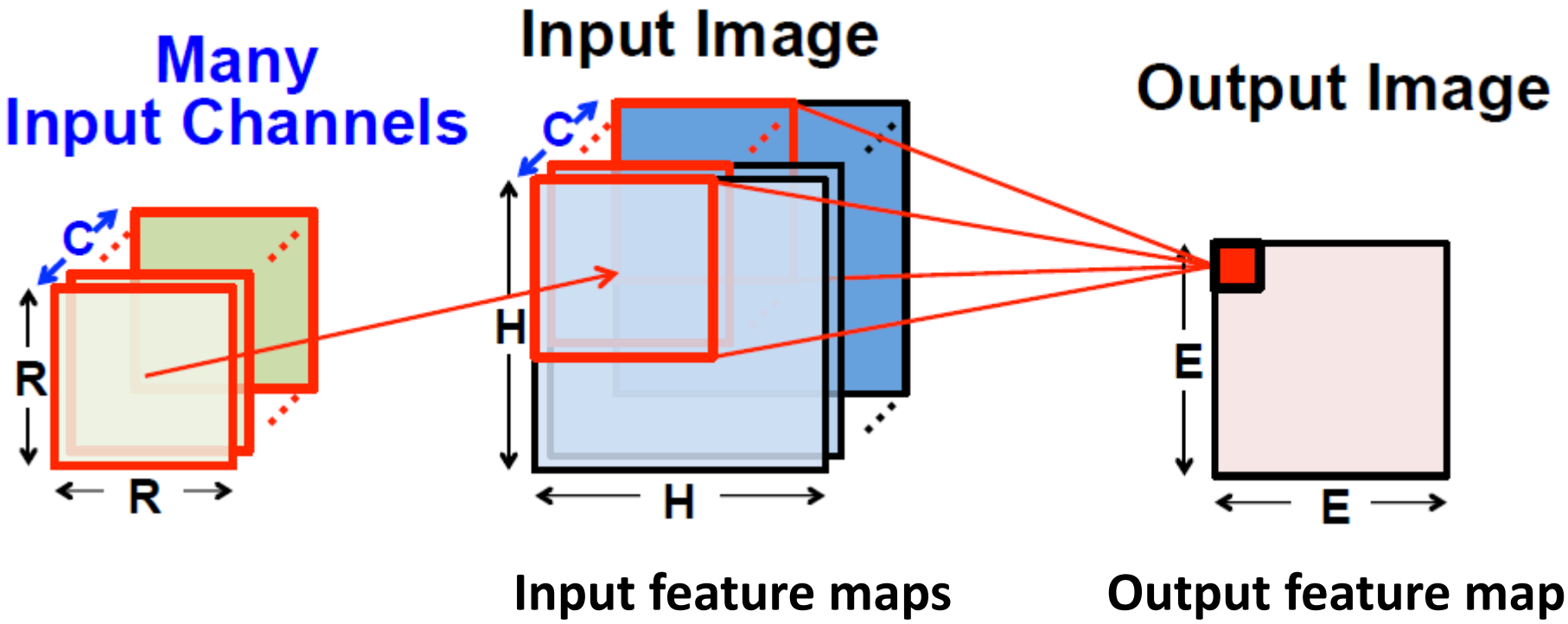
Convolution: 2D Input Case



[Chen, 2016]

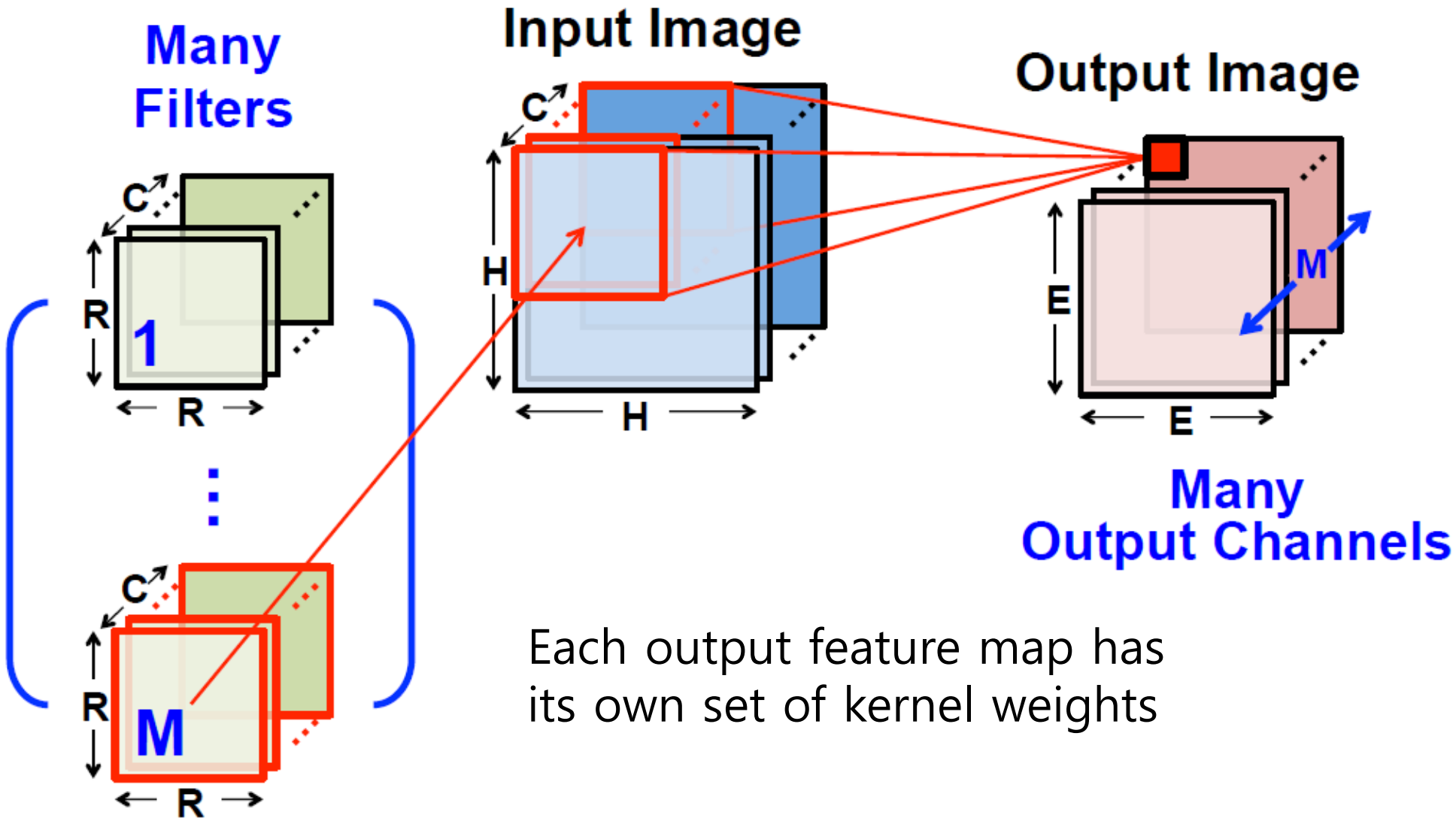
Convolution: 3D Input Case

- Each input feature map has its own set of kernel weights



[Chen, 2016]

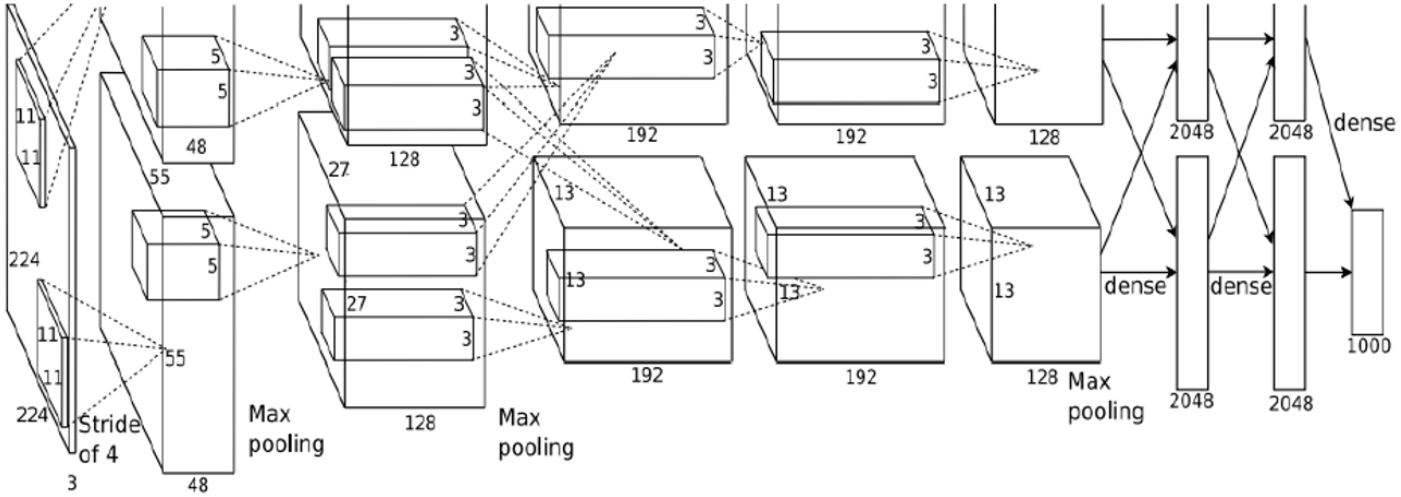
Convolution: 3D Input / 3D Output

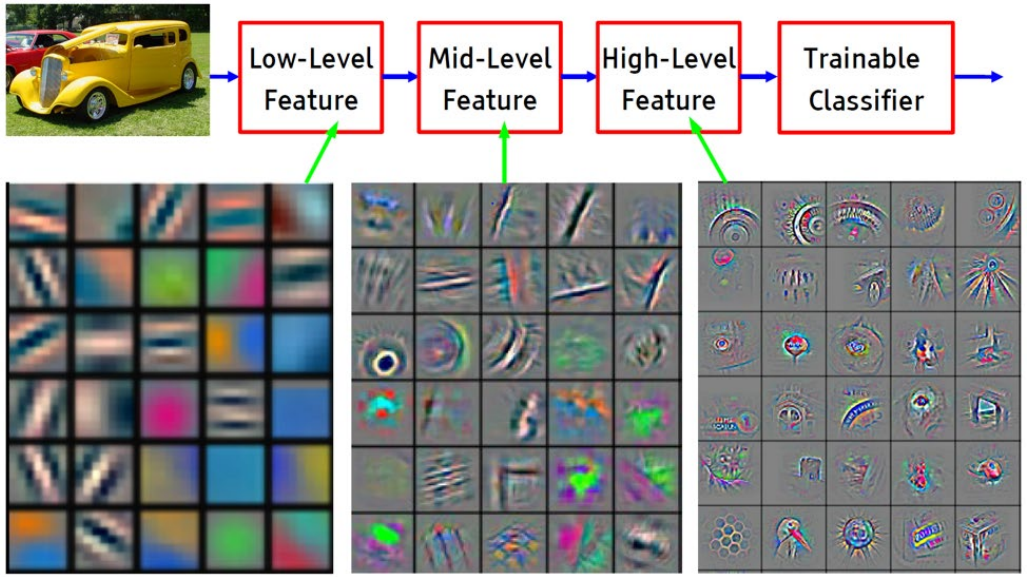


Each output feature map has its own set of kernel weights

[Chen, 2016]

Low Level to High Level Features

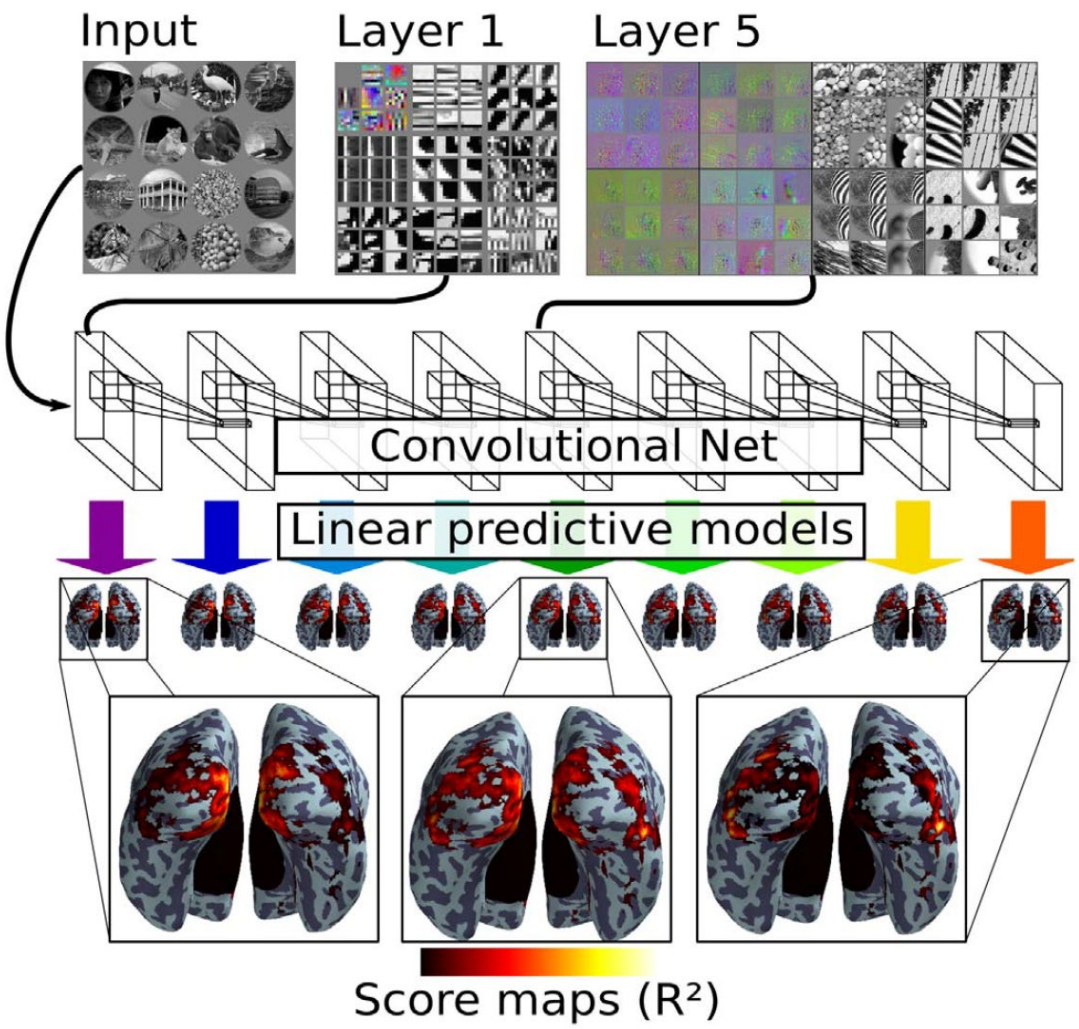


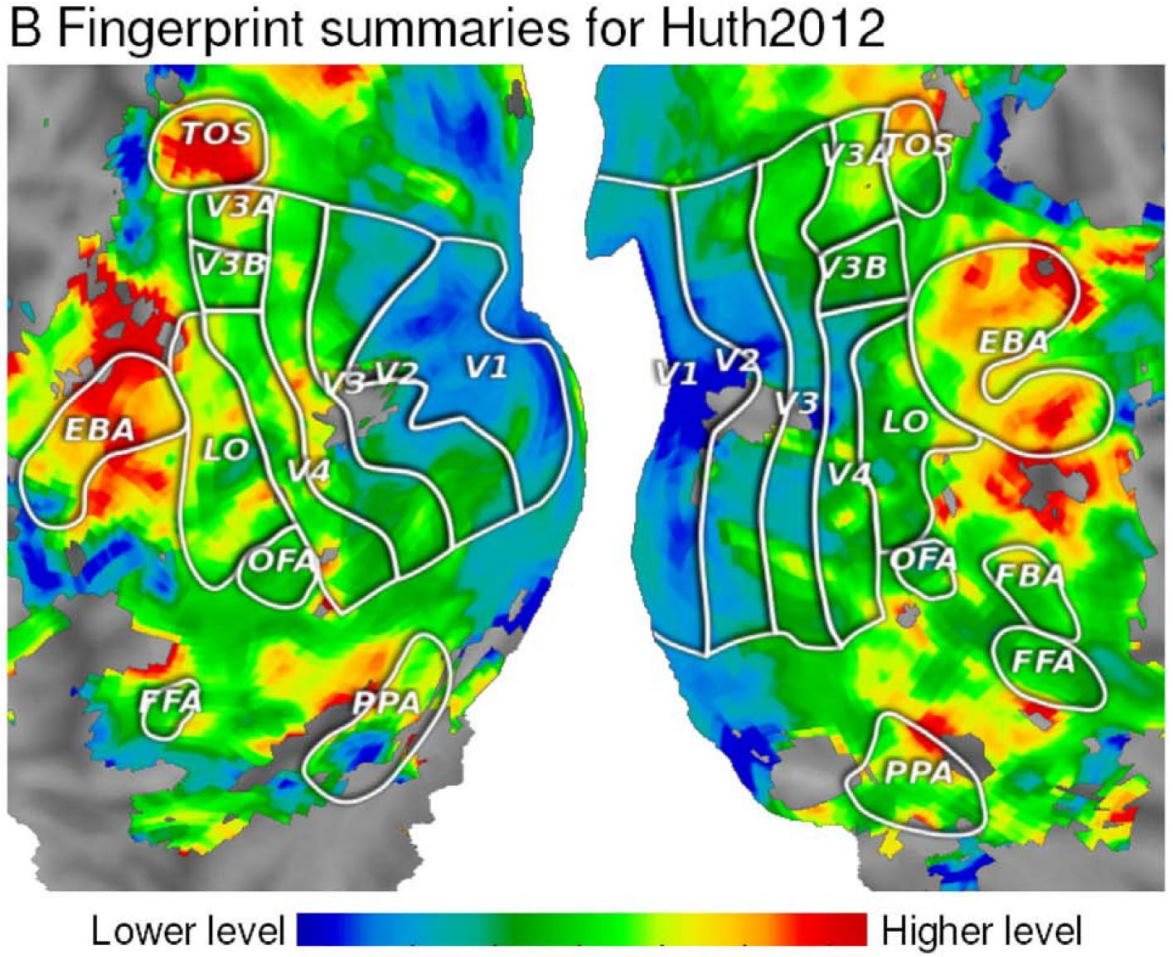


Feature visualization of convolutional net trained on ImageNet from [Zeiler & Fergus 2013]

[LeCun]

Correlation between CNN and Brain

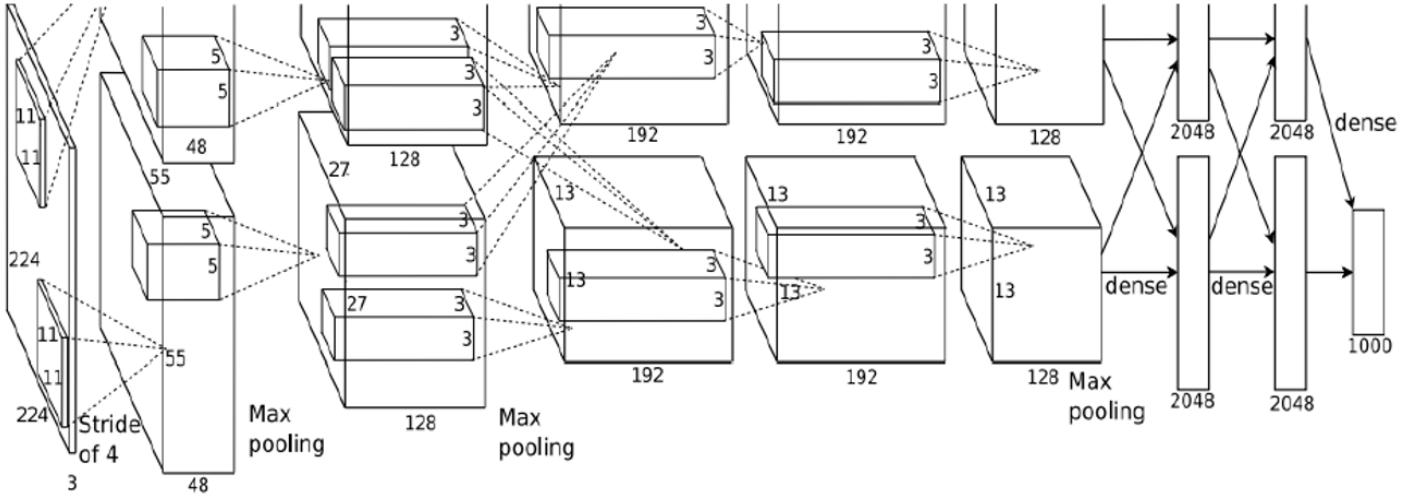


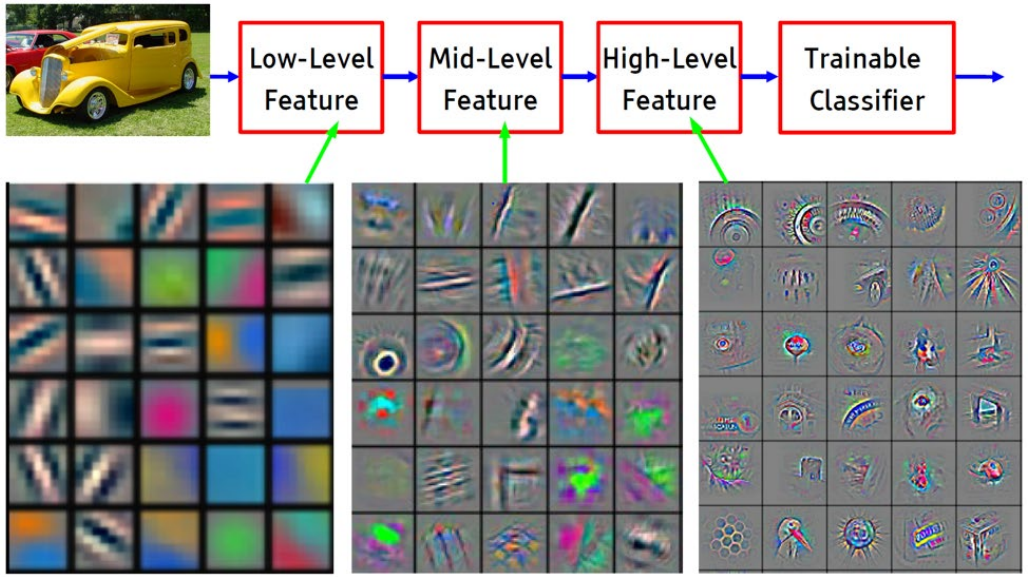


[Michael Eickenberga, et al. 2017]

Neural Network Training

How to Train Feature Extractor?



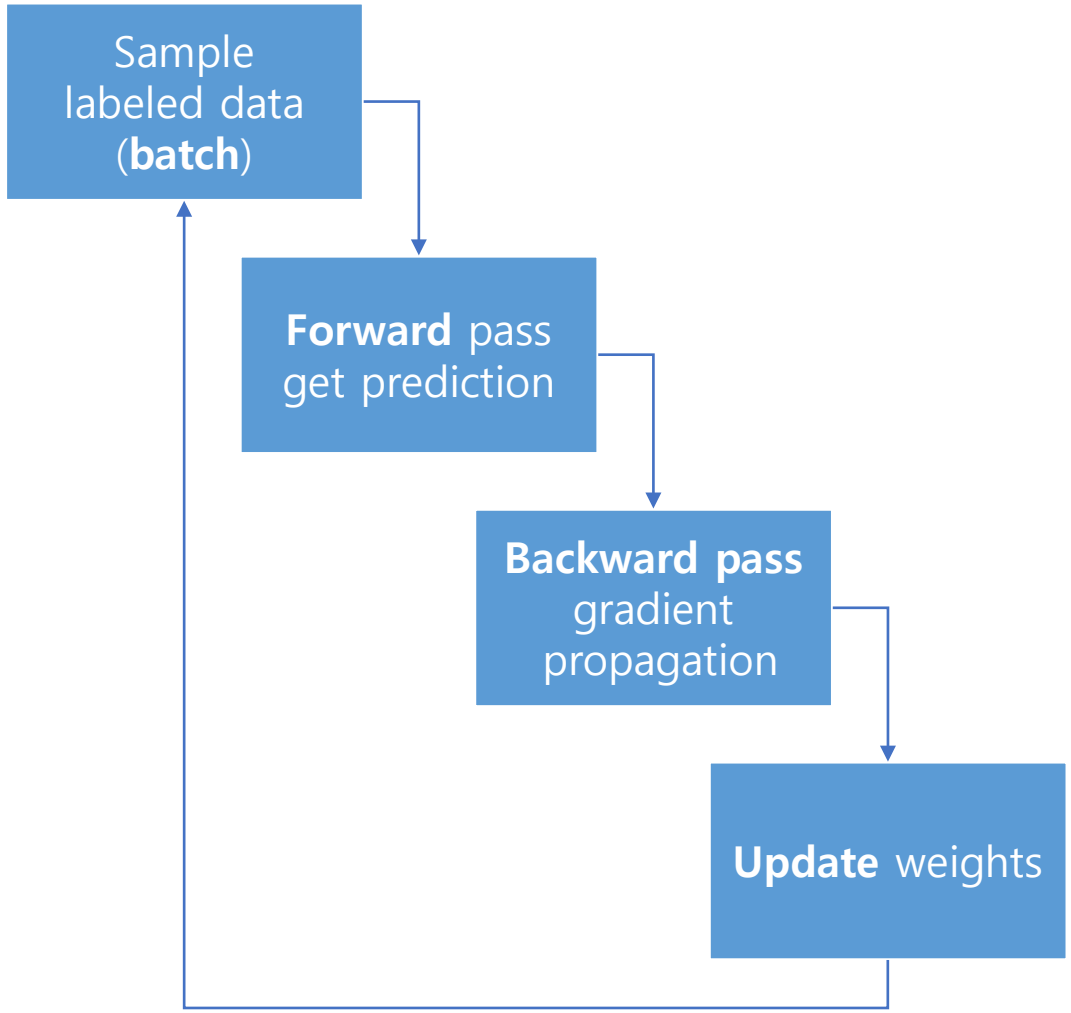


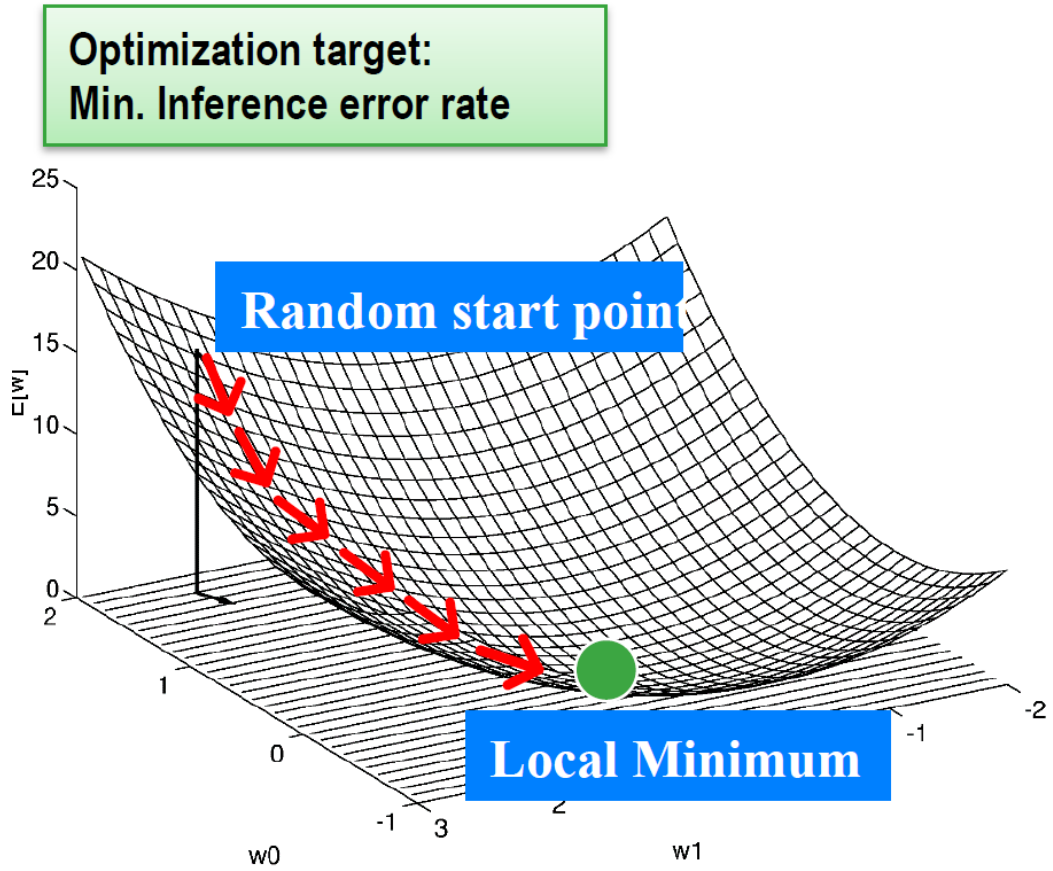
Feature visualization of convolutional net trained on ImageNet from [Zeiler & Fergus 2013]

- Neural Network = feature extractor
- Extract High-level feature by staking multiple layers of feature extractor
- How to train the weight of the feature extractor close to the optimal point?
  - ResNet-18 ~ 11.2M parameters

[LeCun]

Training Neural Network





[Cong, 2015]

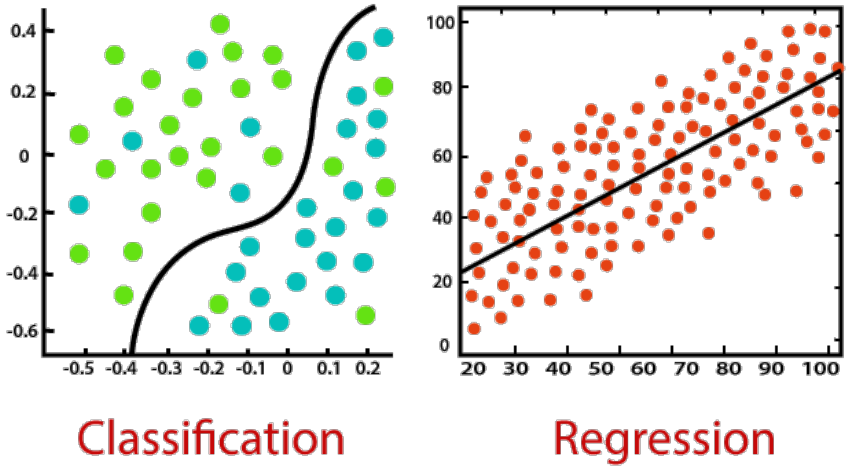
Loss Function Minimization

- Loss function?
  - A **loss function** is a mapping function that is defined to express how well we have solved the problem into a single real number
  - The better an algorithm solves, the lower the loss has.

  - Design a neural network for specific task = design an appropriate loss function

- We must update the weights in the direction of the loss value becoming smaller

Loss Function Example

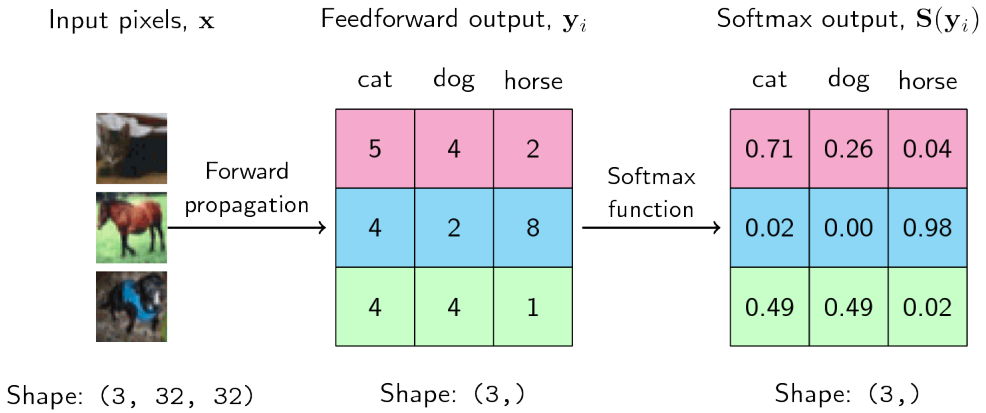


- Regression vs Classification
  - Regression:
    - The process of distinguishing the data into continuous real values

    - Example: predict pressure of machine from sensor data
      - Dataset: $(x_i, y_i),\ x_i \in R^n,\ y_i \in R,\ i \in (0, 1, \ldots, k-1, k)$
      - To find out the weight w s.t. minimize error between y and $y' = f(x)$

      - Loss function: L2 norm
        - $L = \frac{1}{2}\Sigma_i |y - y'|^2 = \frac{1}{2}\Sigma_i |y - f(x)|^2$

Loss Function Example



- Regression vs Classification
  - Classification
    - the process of separating the data into multiple categorical classes

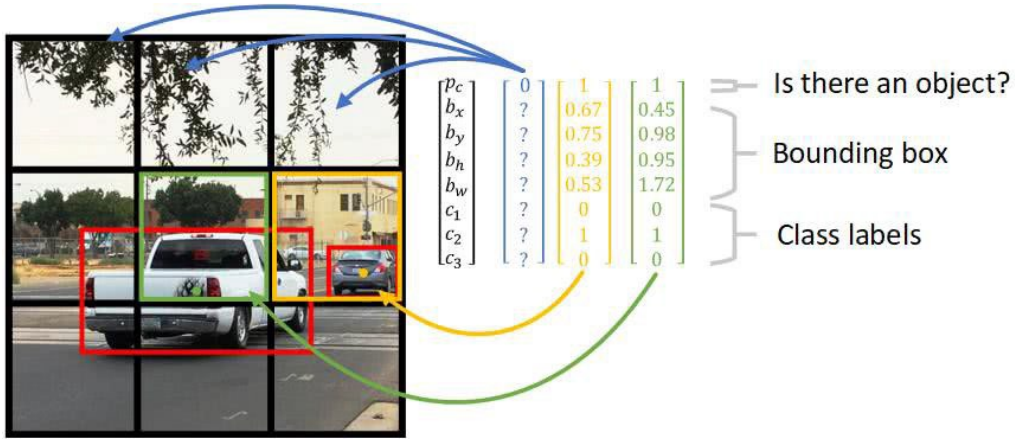
    - Example: image categorization
      - $Dataset$: (x_i, y_i), $x_i \in R^n$, $y_i \in [0,1]^C = [0,0,..,1,..,0]$, $i \in (0,1,\dots,k-1,k)$
      - To find out function $f: R^n \rightarrow R^C$ which maps image to class C

      1 for target class

      - Loss function: cross entropy with softmax
        - Softmax: $p^j(x) = Softmax^j(f(x)) = \frac{e^{f^j(x)}}{\Sigma_k e^{f^k(x)}}$
        - Cross entropy: $L = -\Sigma_i y_i^j \log p^j(x_i)$

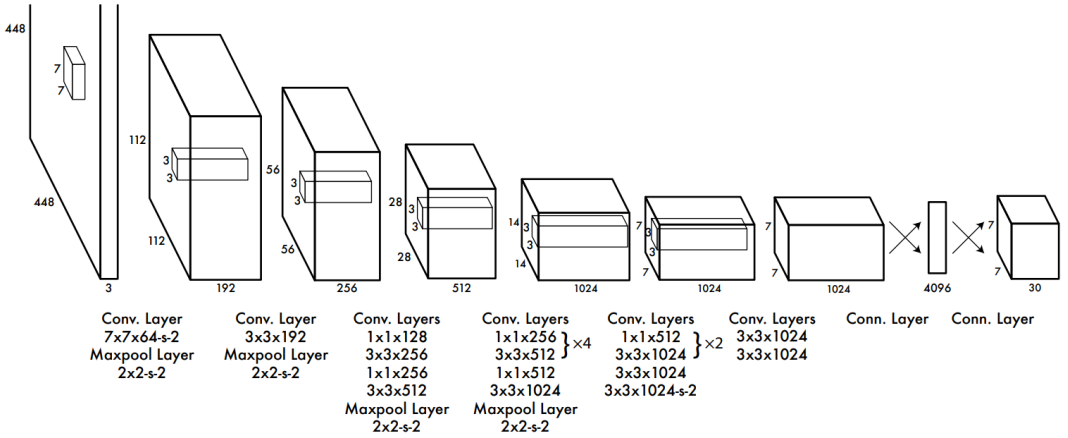
[https://ljvmiranda921.github.io/notebook/2017/08/13/softmax-and-the-negative-log-likelihood/]

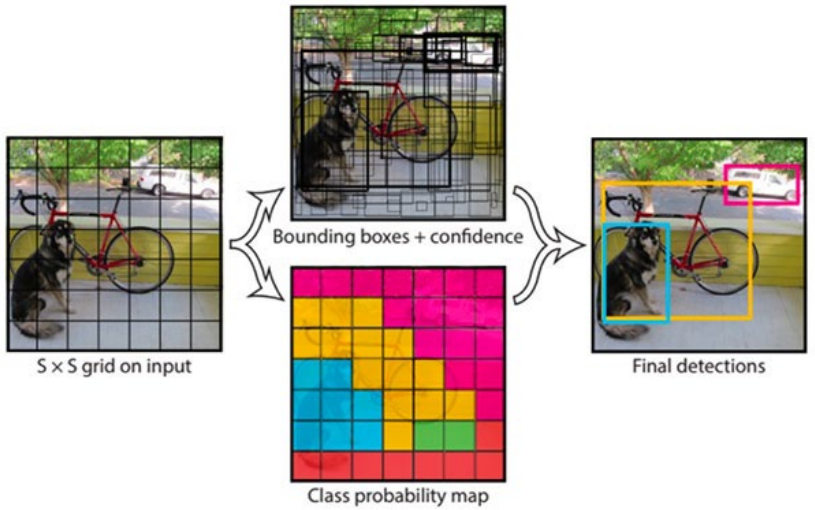
Loss Function Example



- Object Detection
  - Classification + Regression task
    - Object class + object location

    - Network generates 1000s of prediction
    - Solid bounding box is generated by merging the bounding boxes
    - Loss function: $L \approx L_{class} + \alpha \cdot L_{box}$

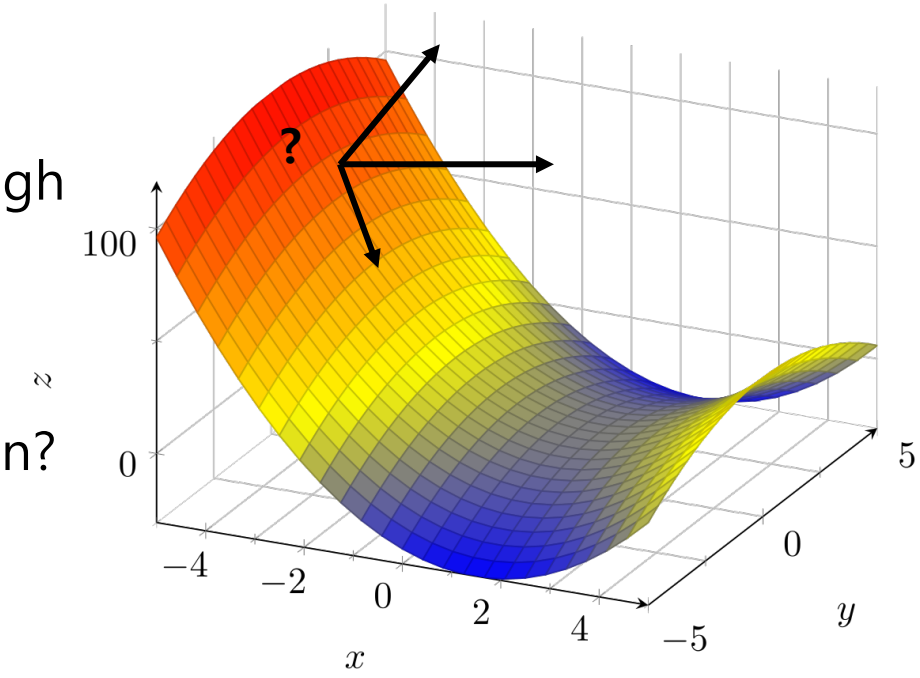




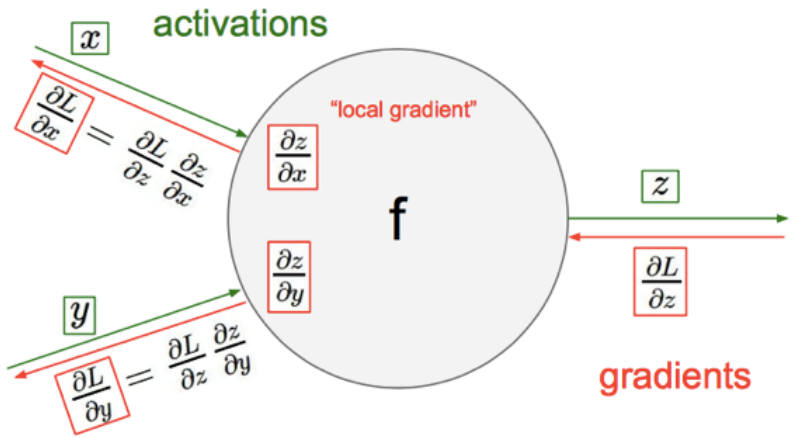
[Yolo]

Gradient Descent with Back Propagation

- Gradient Descent?
  - **Gradient descent** is an optimization algorithm used to find a local minimum of a differentiable space by iteratively update the parameters toward steepest descent.
  - Steepest descent = negative of the gradient
- After the initialization, output loss of the network is high
  - To minimize the loss, we should update network parameters toward the negative gradient of loss
  - How could we get the gradient of the loss function?
    - Back Propagation

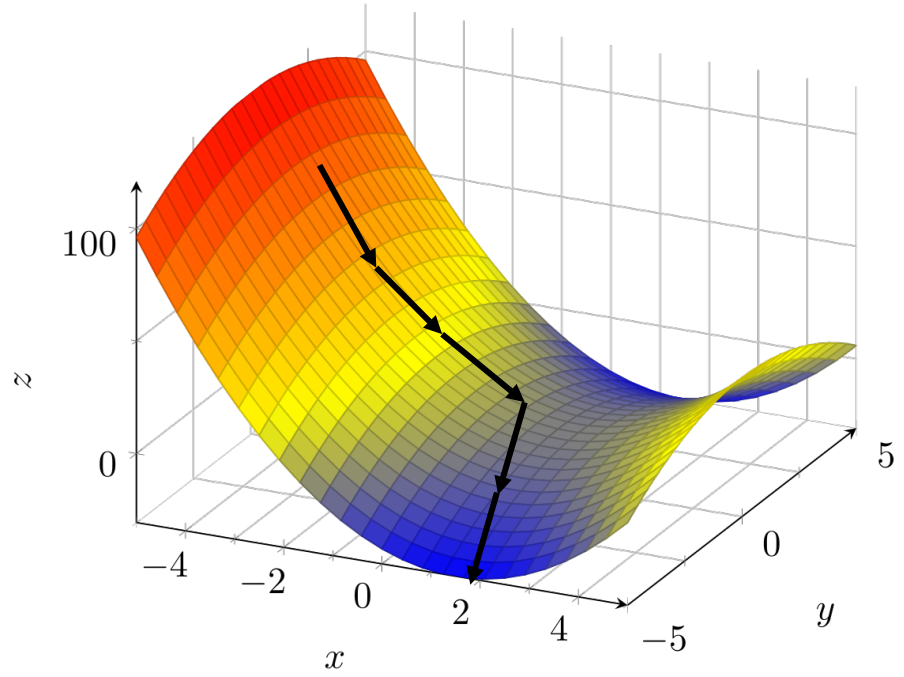


Back Propagation

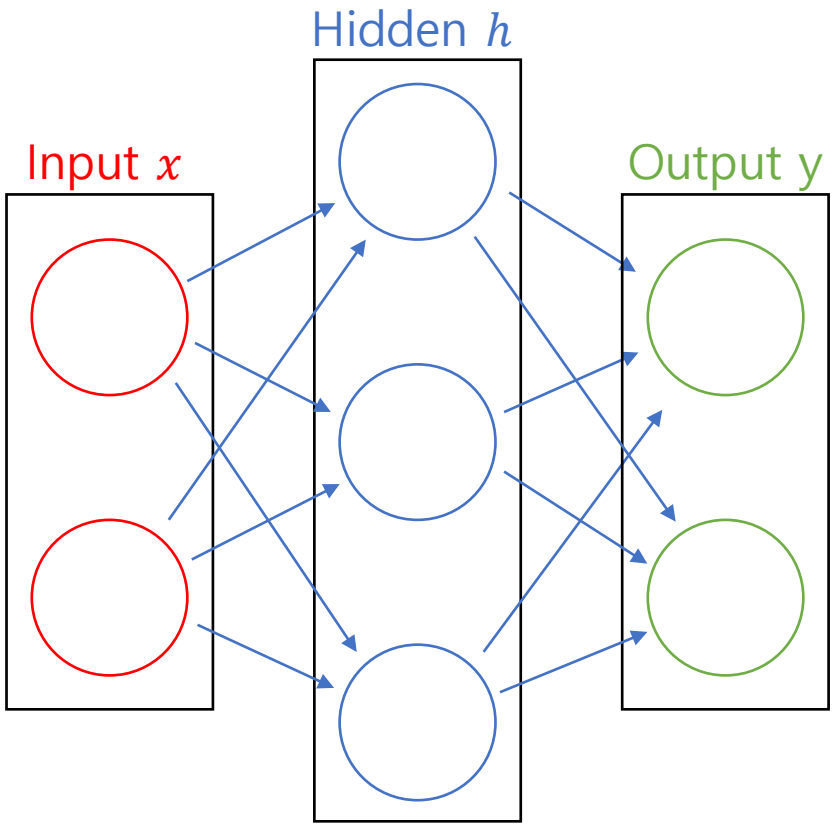


- Back Propagation
  1. Perform forward operation through the entire network
  2. Get the loss value and calculate the local gradient to minimize loss value
  3. Propagating gradient from loss to input progressively to get the gradient of weight

- How to calculate the gradient of weight?
  - Recursively apply chain rule though each node



Weight Update ~ Propagated Error*Activation

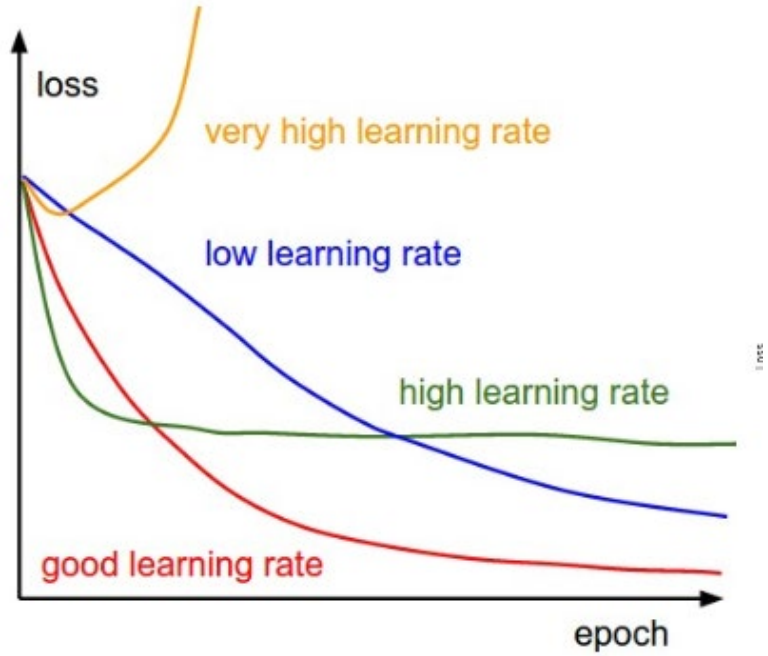


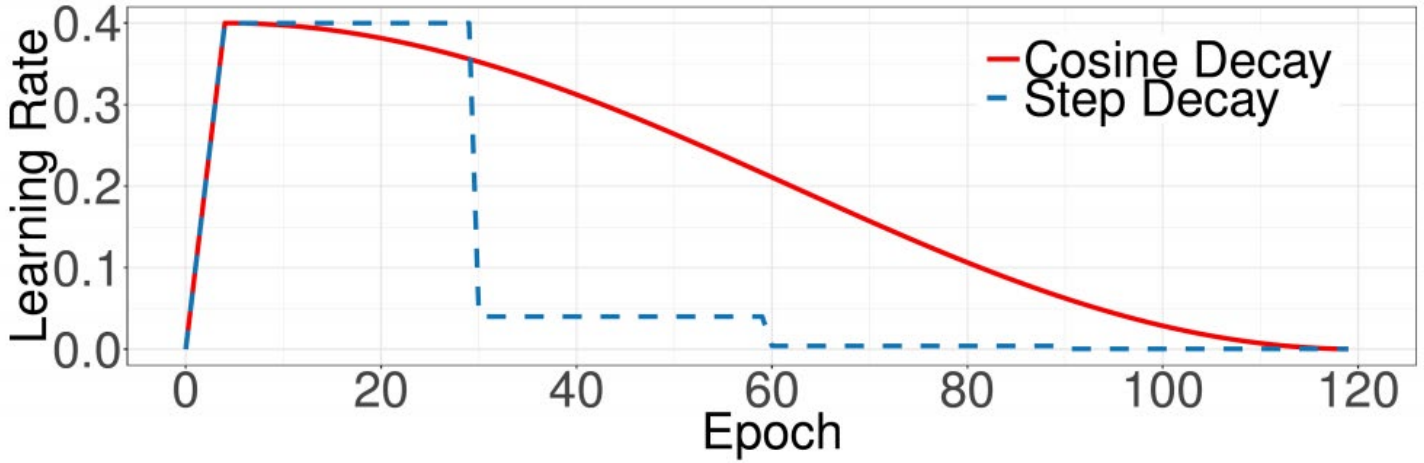
$$\begin{pmatrix} \text{Weight} \\ \text{correction} \\ \Delta w_{ji}(n) \end{pmatrix} = \begin{pmatrix} \text{learning-} \\ \text{rate parameter} \\ \eta \end{pmatrix} \times \begin{pmatrix} \text{local} \\ \text{gradient} \\ \delta_j(n) \end{pmatrix} \times \begin{pmatrix} \text{input signal} \\ \text{of neuron } j, \\ y_i(n) \end{pmatrix}$$

Learning Rate

$$\begin{pmatrix} \text{Weight} \\ \text{correction} \\ \Delta w_{ji}(n) \end{pmatrix} = \begin{pmatrix} \text{learning-} \\ \text{rate parameter} \\ \eta \end{pmatrix} \times \begin{pmatrix} \text{local} \\ \text{gradient} \\ \delta_j(n) \end{pmatrix} \times \begin{pmatrix} \text{input signal} \\ \text{of neuron } j, \\ y_i(n) \end{pmatrix}$$

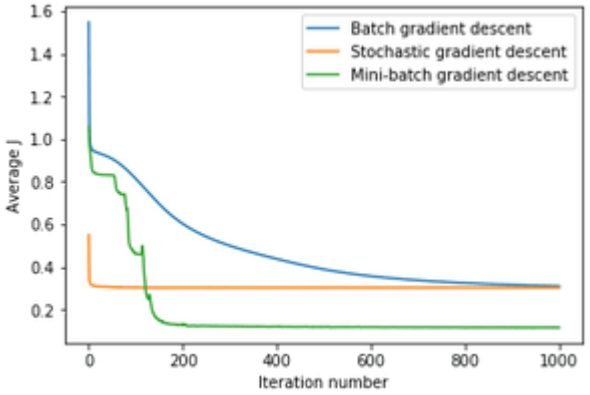
- Small learning rate, slow and smooth change in *w*
- Too large learning rate, *w* can be unstable
- Step decay or cosine decay with warmup is often used



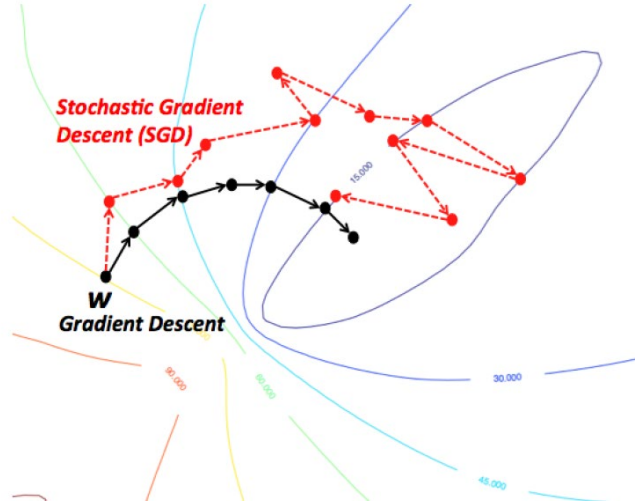


[CS231n]
[Tong, 2018]

Training with Mini-Batch



- Batch Gradient Descent
  - Get gradient for batched data (all training set)
  - Expensive when the number of training data is large
  - Smooth loss surface → slow convergence

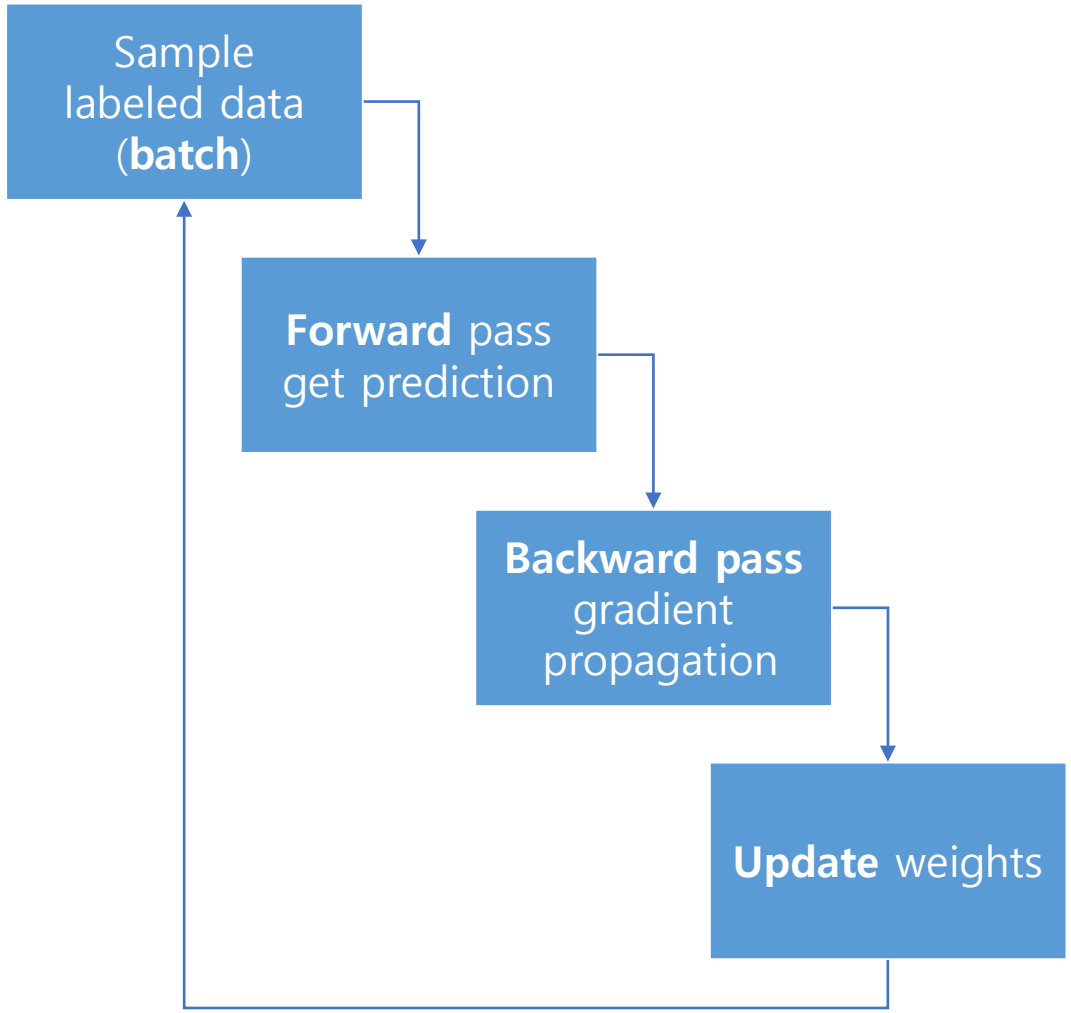


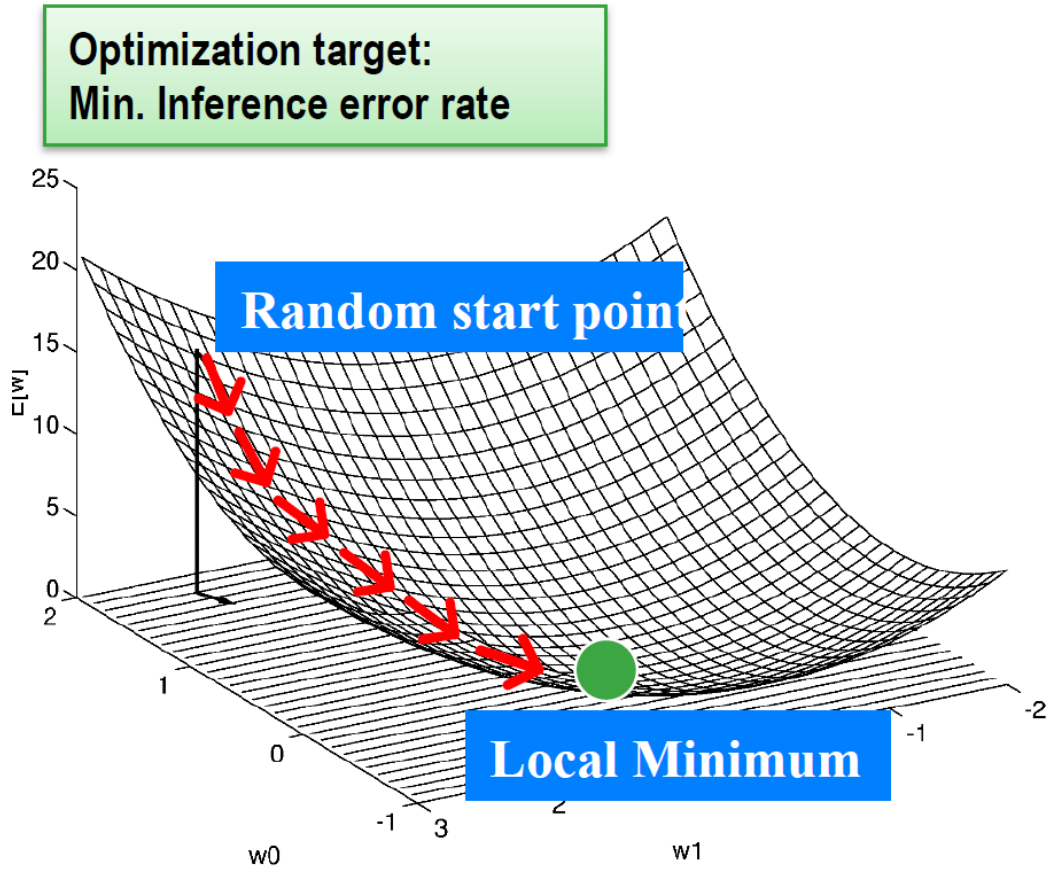
- Stochastic Gradient Descent
  - Get gradient for single randomly-sampled data
  - Oscillation near the optima due to random noise
  - A small amount of computation → low device utilization

- Mini Batch Gradient Descent
  - Get gradient for randomly-sampled 10~1000s of data
  - Fast convergence with efficient computation

[https://adventuresinmachinelearning.com/stochastic-gradient-descent/]
[https://wikidocs.net/3413

Training Neural Network





[Cong, 2015]

SGD with Momentum

- SGD may have trouble near
  - Plateau
  - Saddle point
  - Local minima

- Momentum is introduced to speed up the learning process
  - Accumulate the moving average of previous gradients decaying exponentially
  - $m_t = \alpha m_{t-1} + (1-\alpha)\frac{\partial L}{\partial w}, w_t = w_{t-1} - \eta m_t$

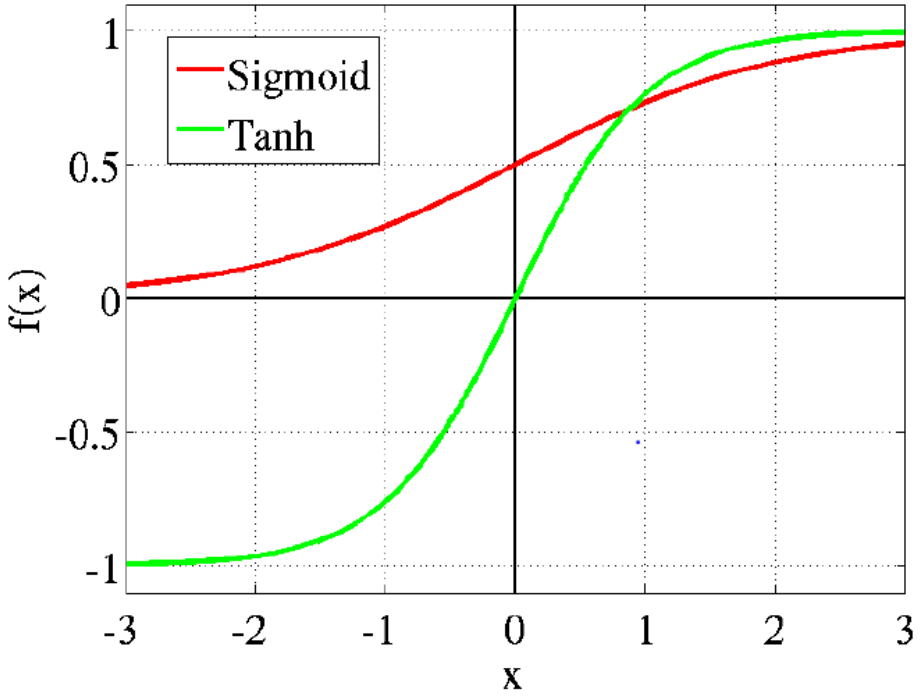
Activation Function

- Non-linearity is a key-concept of a neural network
  - Without non-linearity, stacked layers could be merged into one layer
  - Ex) 2-layer MLP without non-linear function

- We insert activation function or layer between feature extractor
  - Sigmoid, tanh was frequently used

  - Vanishing gradient problem?
    - Use ReLU instead of Sigmoid or Tanh
    - But, sigmoid and tanh are still very important and often used

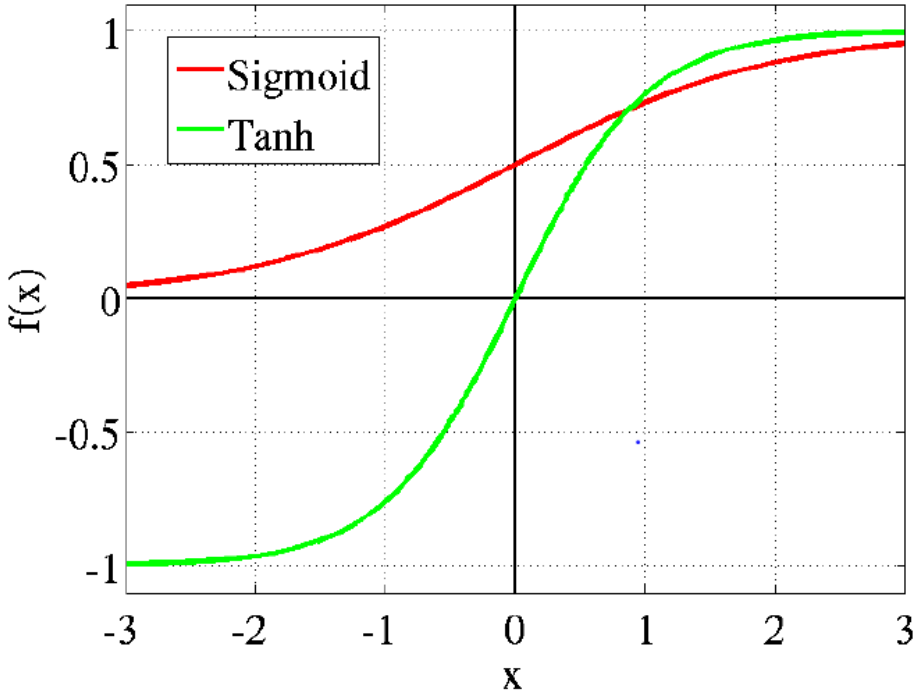
Sigmoid and Tanh



- Sigmoid maps real value to [0, 1]
  - $y = \frac{e^x}{e^x+1}$
  - Useful to emulate probability or activity
- Tanh maps real value to [-1, 1]
  - $y = 2 * sigmoid(x) - 1$
  - Zero-centered
- Both operations have saturation region
  - There is almost no difference when input is large

Problem of Vanishing Gradient

- Diminishing delta away from the output layer



$$\delta_j(n) = \varphi_j'(v_j(n)) \sum_k \delta_k(n) w_{kj}(n), \qquad \text{neuron } j \text{ is hidden}$$

$$\sigma(x) = \frac{e^x}{e^x + 1}$$

$$\sigma'(\cdot) \leq \frac{1}{4}$$

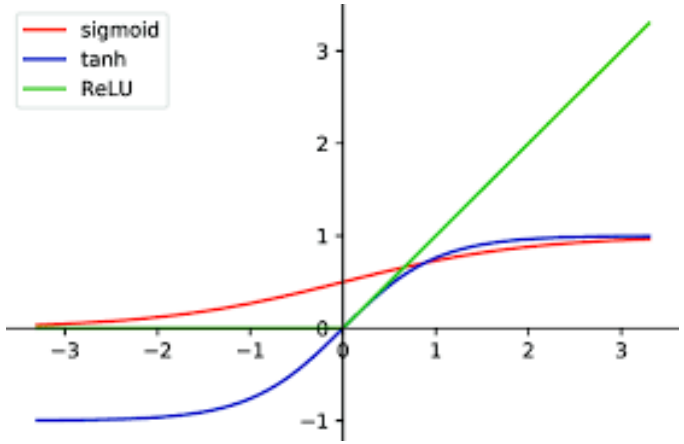
At each layer,
local gradient decreases
by more than ¼ times

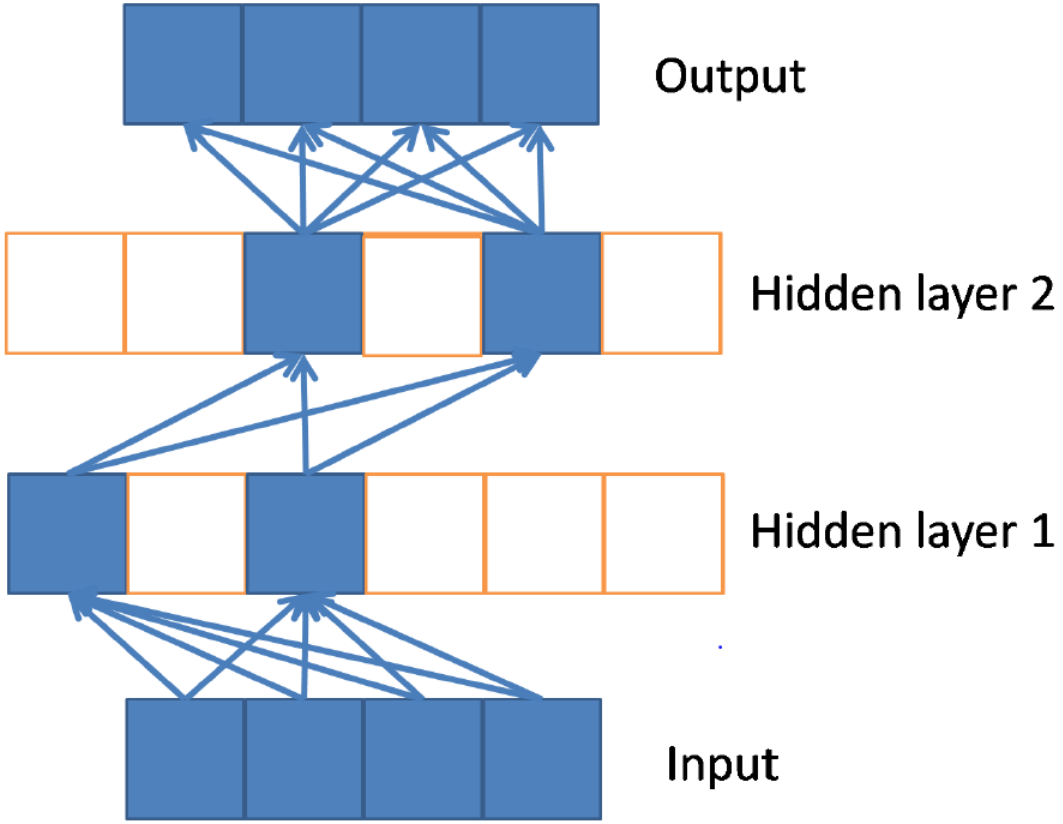
$$\Delta w_{ji}(n) = \eta \delta_j(n) y_i(n)$$

[Sungjoo Yoo]

Rectified Linear Unit (ReLU)

- ReLU vs Sigmoid/Tanh
  - Non-saturation form
  - No vanishing-gradient problem
  - Faster implementation





[J. Kim, KAIST]
[Sungjoo Yoo]

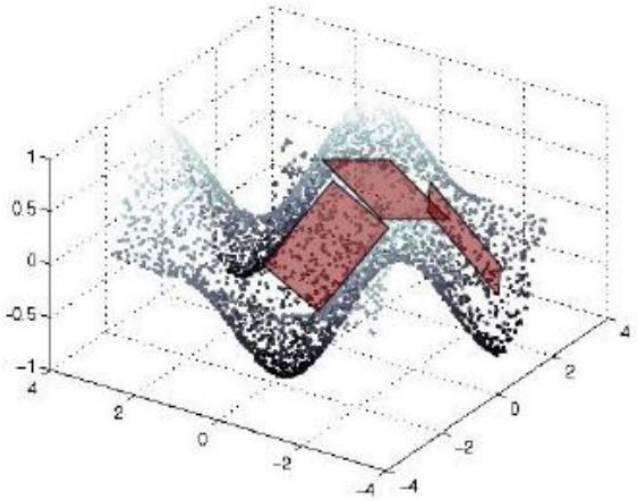
ReLU for Piece-wise Linear Approximation possibly on Manifold

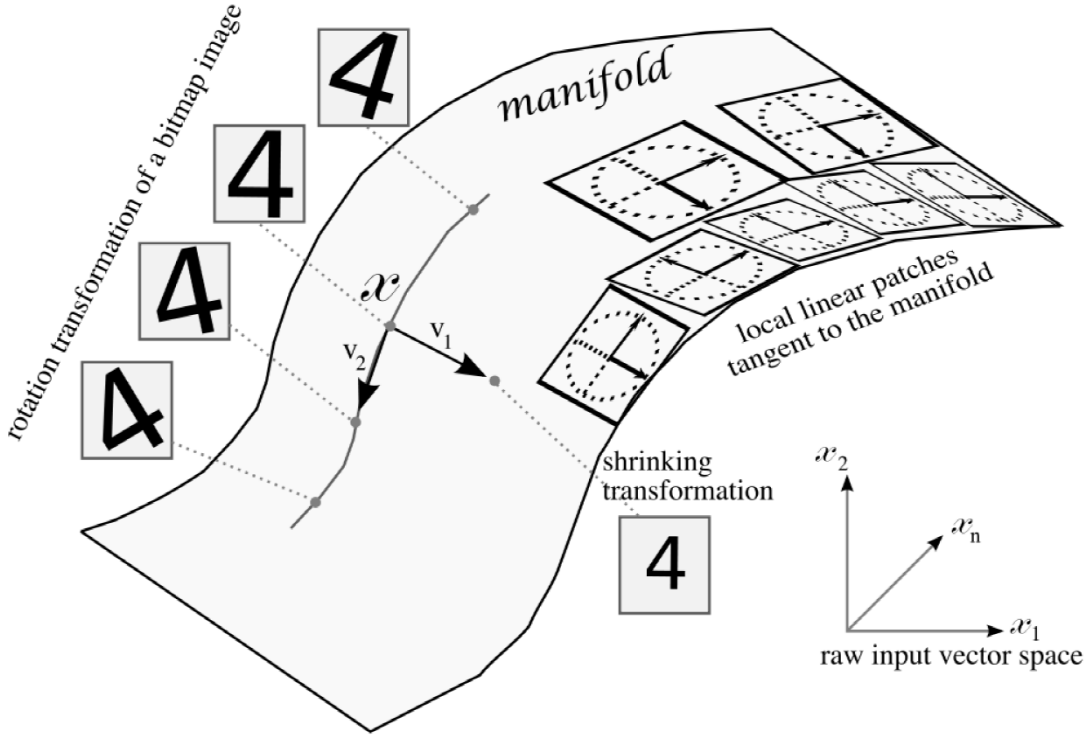
Question: Why can't the mapping between layers be linear?

Answer: Because composition of linear functions is a linear function.
Neural network would reduce to (1 layer) logistic regression.

Question: What do ReLU layers accomplish?

Answer: Piece-wise linear tiling: mapping is locally linear.

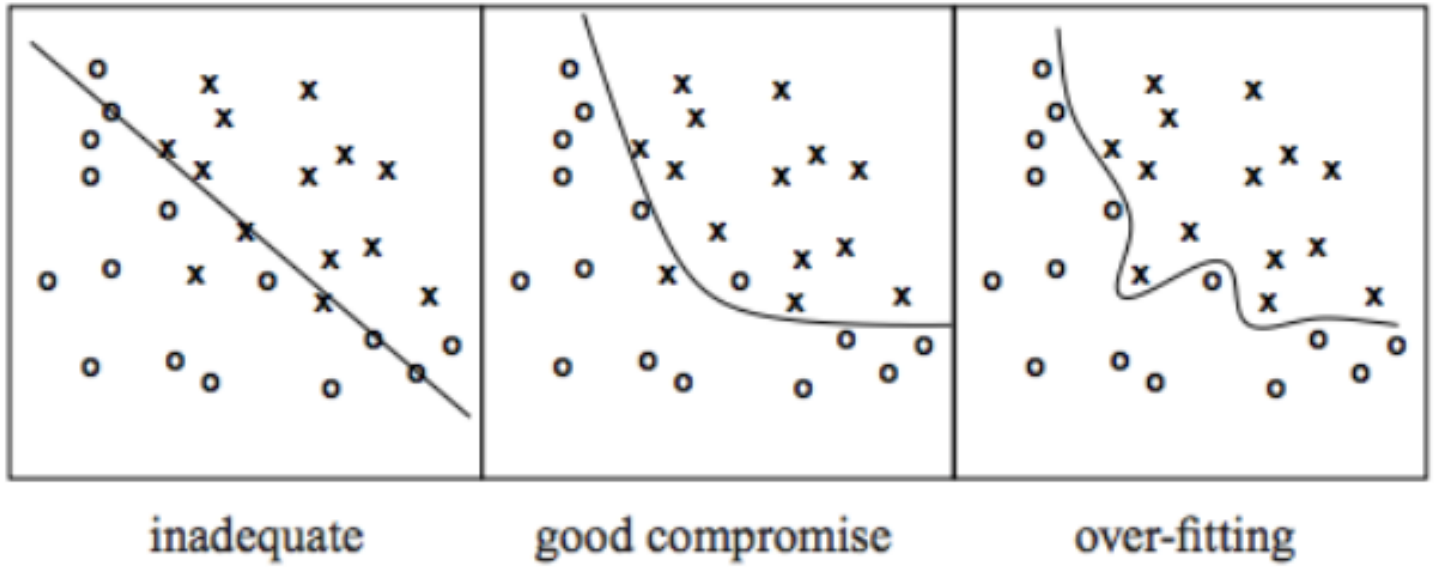


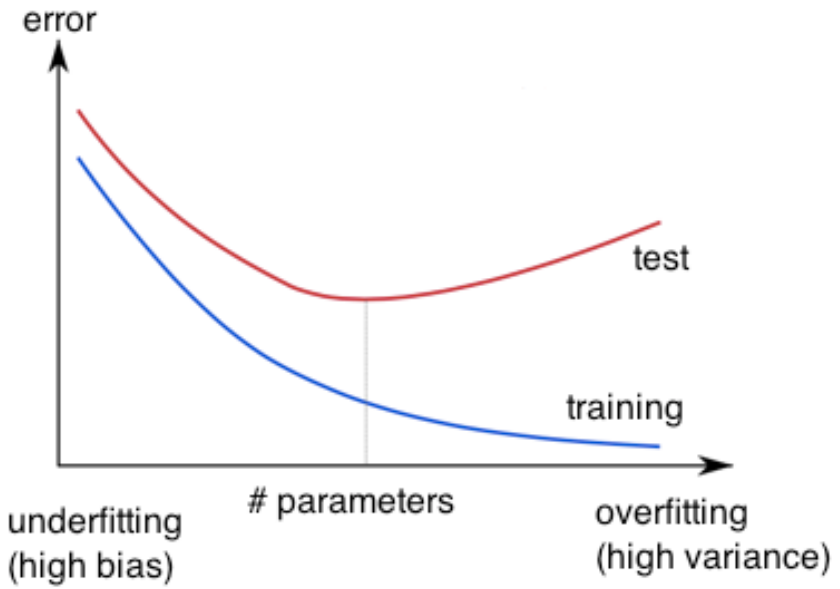


[Bengio, 2014]

Overfitting

- Training data may contain noise & mislabeled data
- Learned network may show 100% accuracy for the training data, even outliers (noise)
- But it could fail to generalize to new examples (test data)

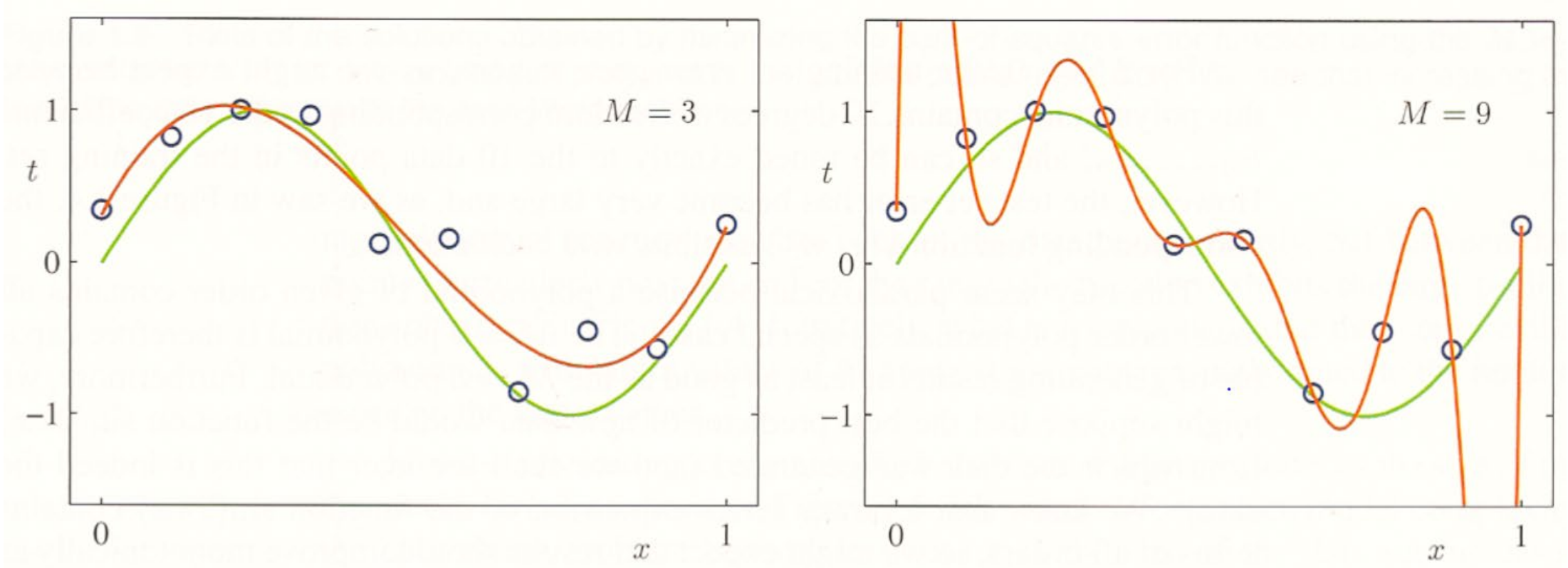




[http://wiki.bethanycrane.com/overfitting-of-data]
[https://www.neuraldesigner.com/images/learning/selection_error.svg]

Overfitting - 2

- 3rd order approximation vs 9th order approximation



[Bishop, 2006]

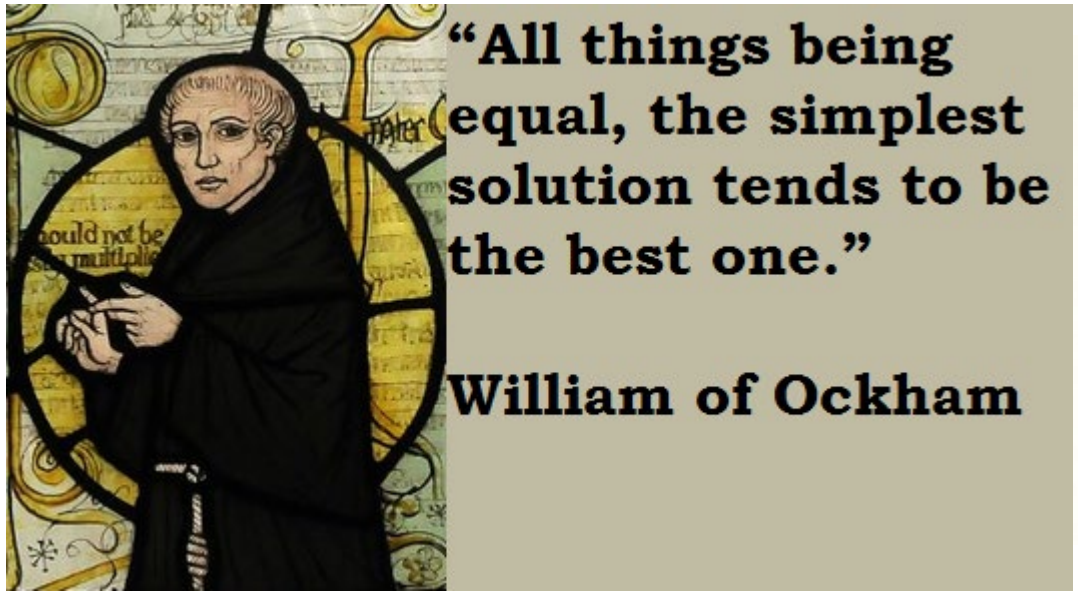
Overfitting - Solution

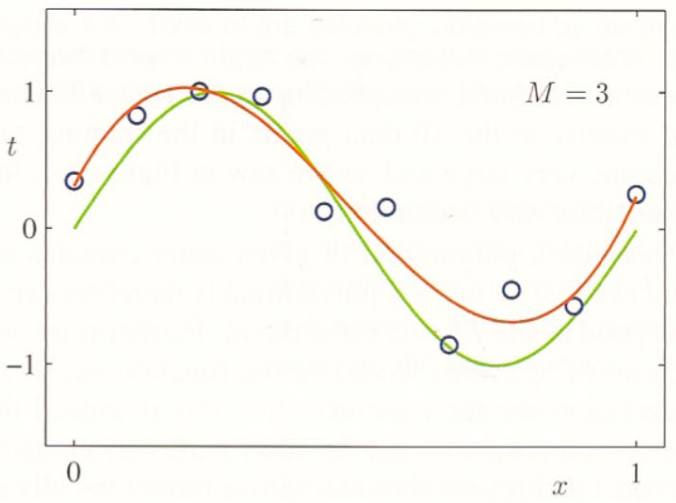
- Problem?
  - The degree of freedom of the parameter is much greater than the amount of information in the learning data.
  - When the amount of learning data is not sufficient

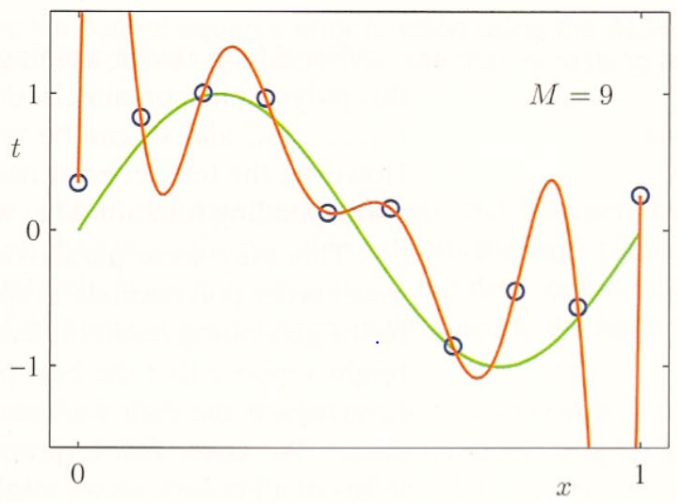
- Solutions
  - Selecting a simple model, i.e., Occam's Razor
  - Regularization
  - Drop out
  - Larger training set

Occam's Razor

- Use 3rd order approximation rather than 9th order one



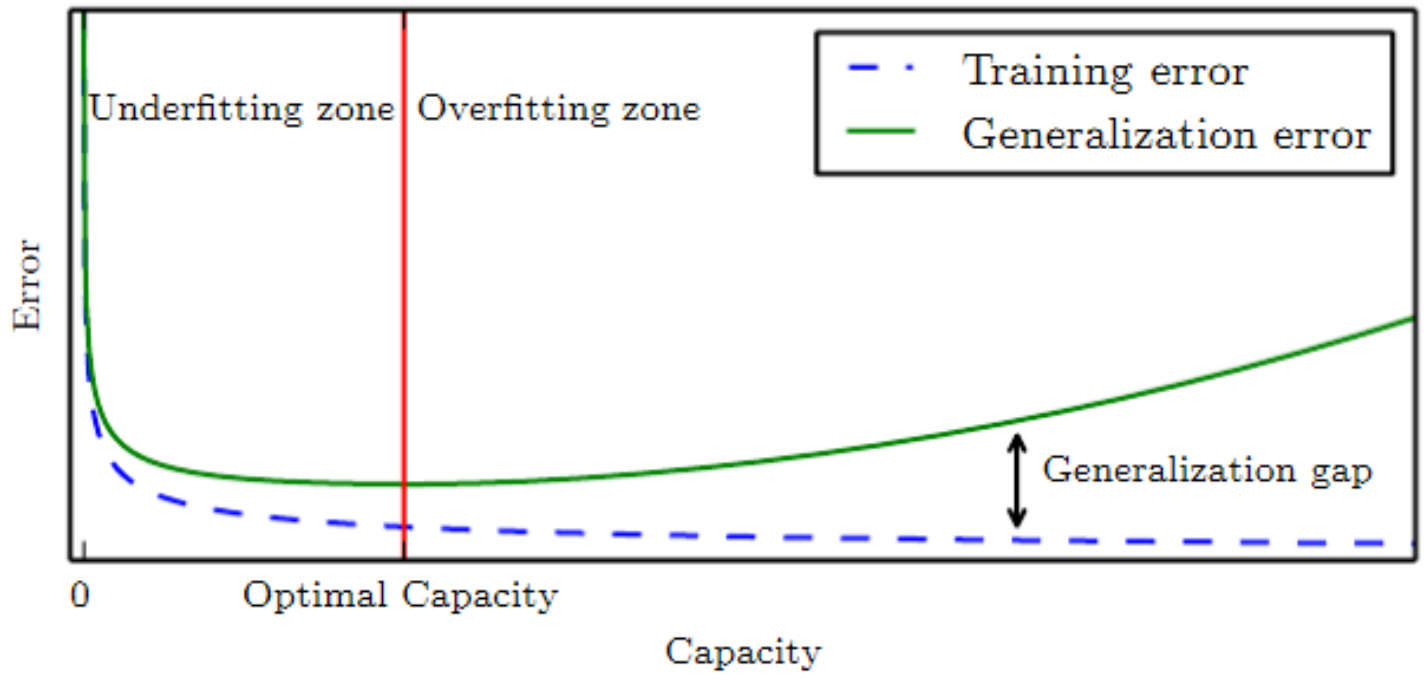




[Bishop, 2006]
[Sungjoo Yoo]

Selecting Appropriate Model Size

- Selecting a proper model size is important for accuracy and cost
- Small model – low capacity – underfitting
- Large model – high capacity – overfitting

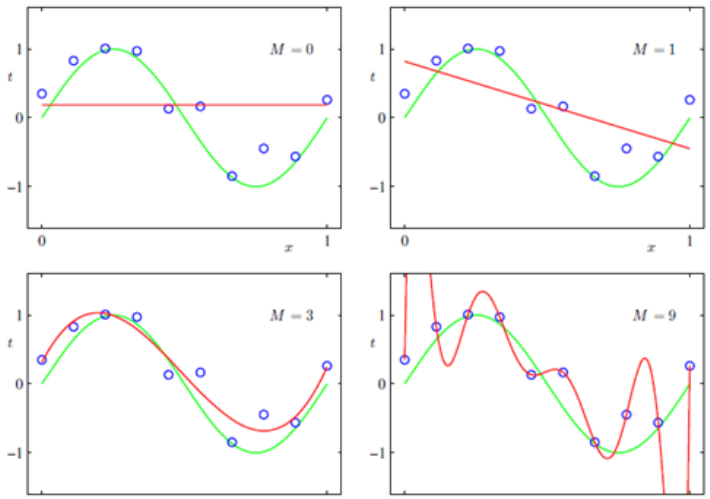


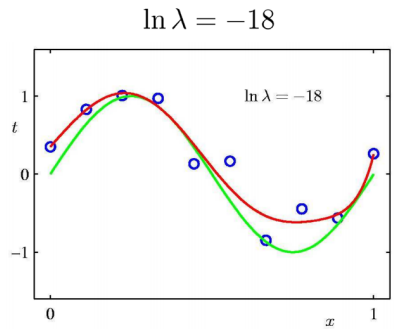
[Deep Learning Book, Ch5]

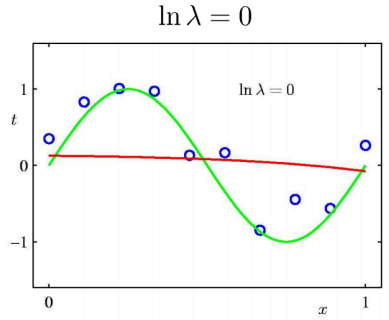
Regularization

- Observation: Coefficients of overfitting function tend to have large magnitude

- Regularization (L2 case)
  - Add an additional term to loss function
  - $L = L_{class} + \lambda w^2$
  - A suitable adjustment of λ is needed







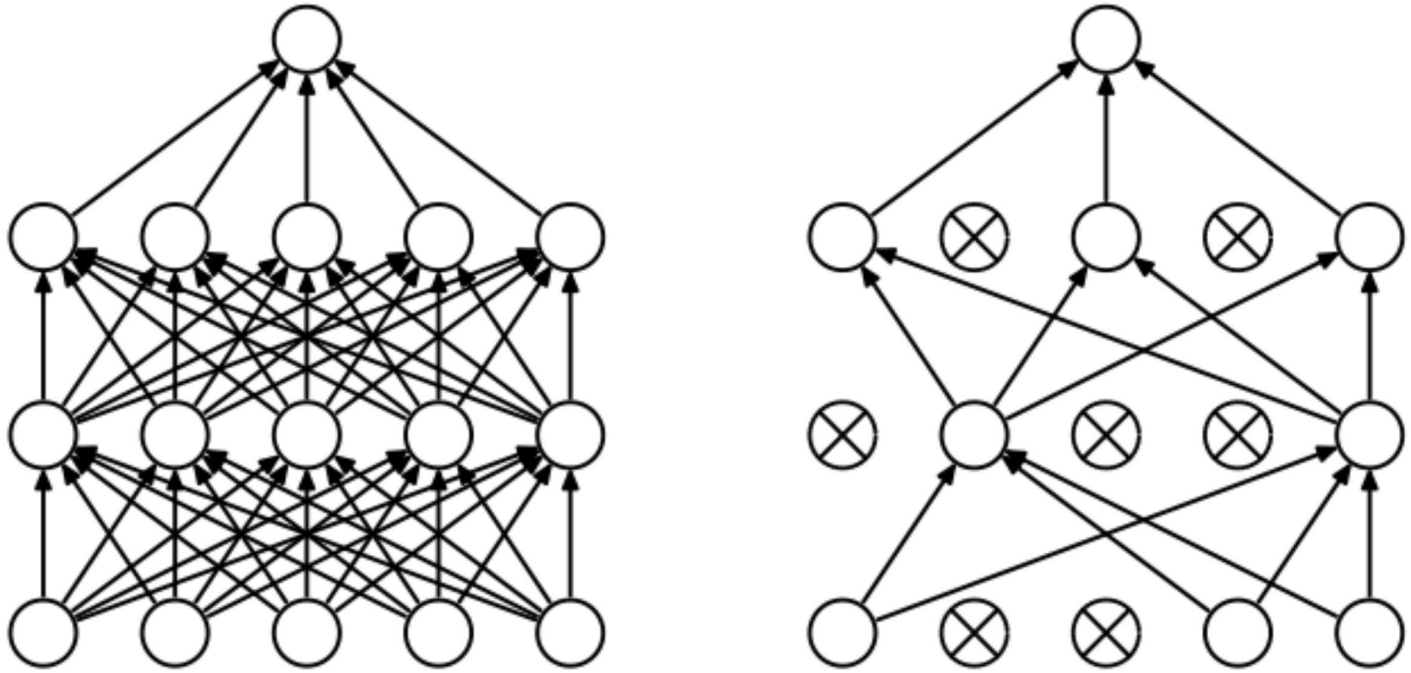
	$M=0$	$M=1$	$M=6$	$M=9$
$w_0^\star$	0.19	0.82	0.31	0.35
$w_1^\star$		-1.27	7.99	232.37
$w_2^\star$			-25.43	-5321.83
$w_3^\star$			17.37	48568.31
$w_4^\star$				-231639.30
$w_5^\star$				640042.26
$w_6^\star$				-1061800.52
$w_7^\star$				1042400.18
$w_8^\star$				-557682.99
$w_9^\star$				125201.43

	$\ln\lambda=-\infty$	$\ln\lambda=-18$	$\ln\lambda=0$
$w_0^\star$	0.35	0.35	0.13
$w_1^\star$	232.37	4.74	-0.05
$w_2^\star$	-5321.83	-0.77	-0.06
$w_3^\star$	48568.31	-31.97	-0.05
$w_4^\star$	-231639.30	-3.89	-0.03
$w_5^\star$	640042.26	55.28	-0.02
$w_6^\star$	-1061800.52	41.32	-0.01
$w_7^\star$	1042400.18	-45.95	-0.00
$w_8^\star$	-557682.99	-91.53	0.00
$w_9^\star$	125201.43	72.68	0.01

[Bishop, 2006]

Dropout Effect [Hinton, 2012]

- Subset of neurons are disabled and not trained



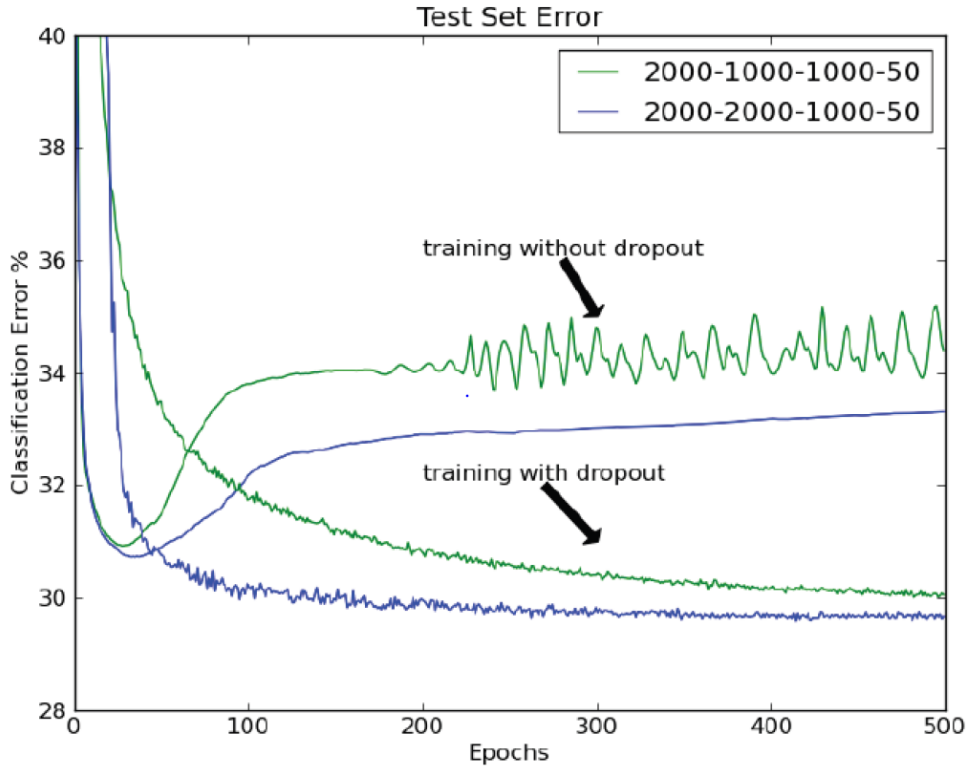
(a) Standard Neural Net

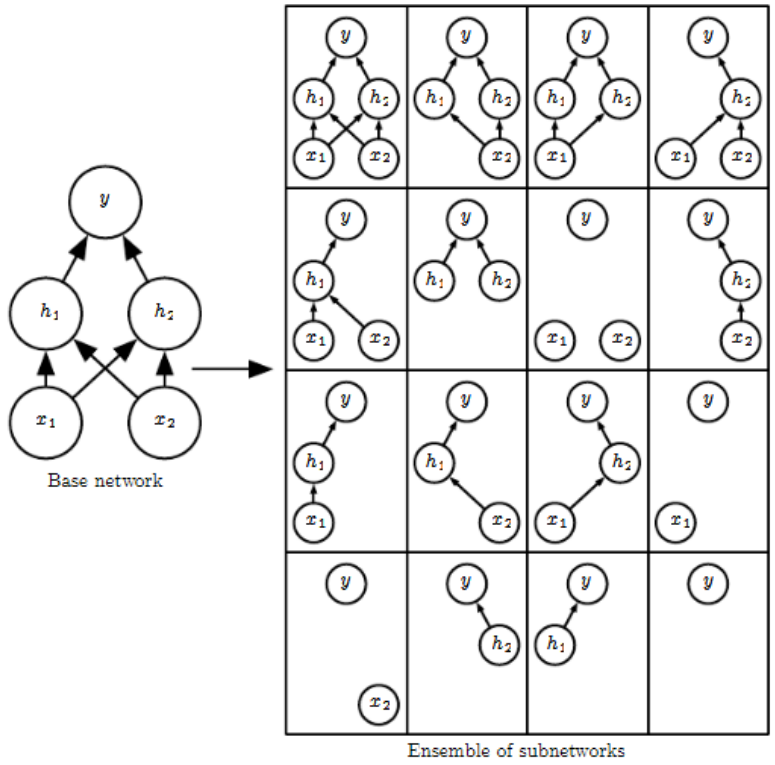
(b) After applying dropout.

[J. Kim, KAIST]

Dropout Effect [Hinton, 2012]

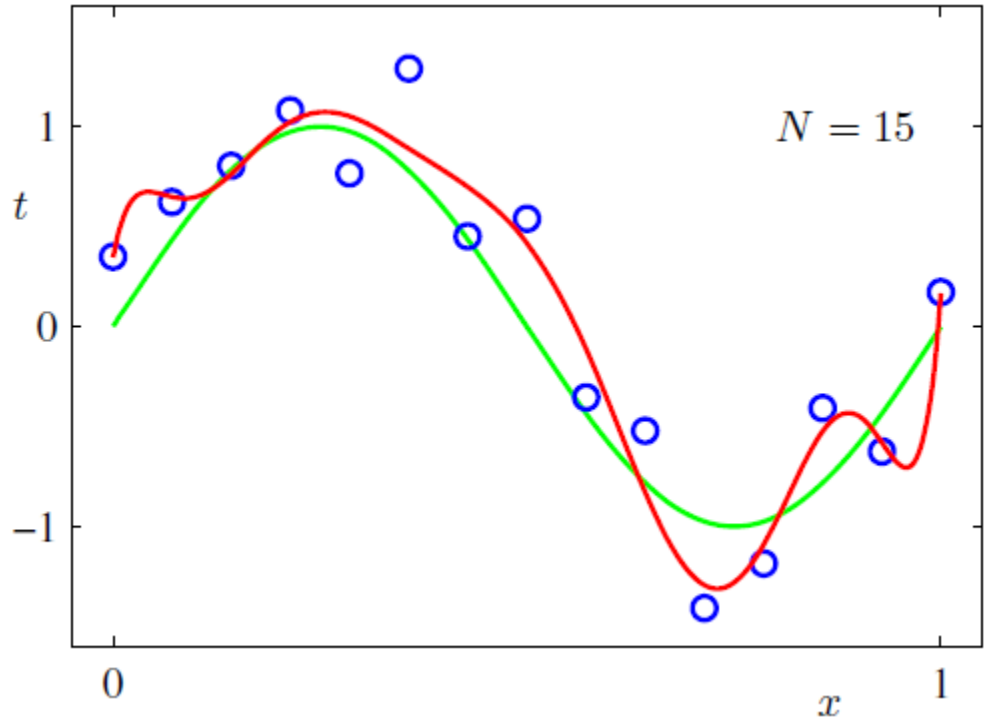
- Subset of neurons are disabled and not trained
- Dropout can be seen as ensemble learning

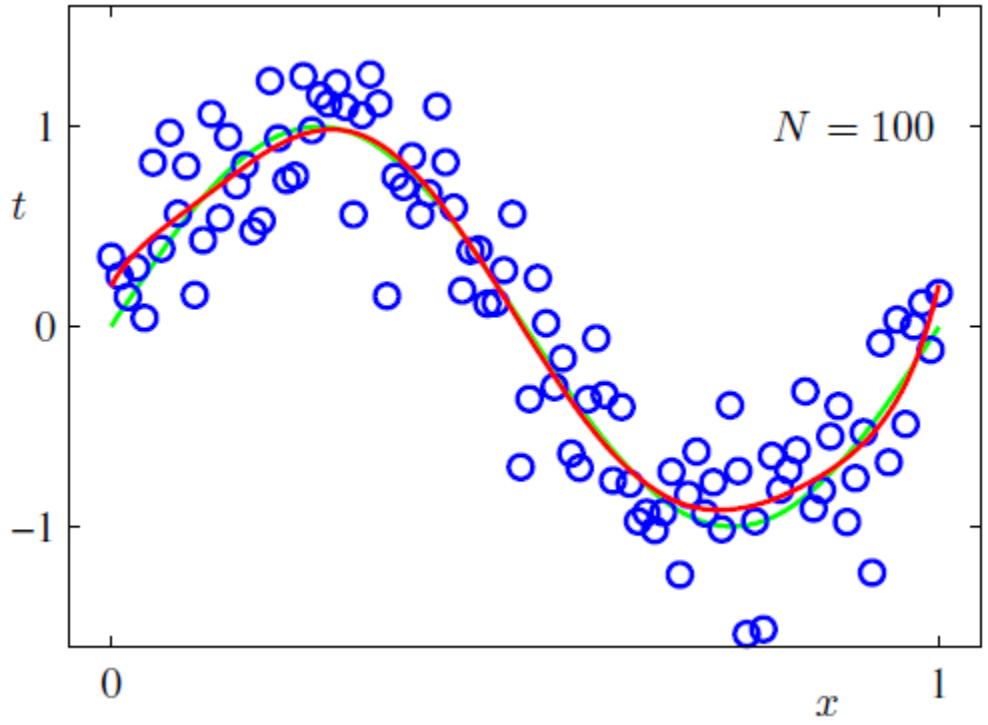




[Deep Learning Book, Ch7]

Increasing Training Set Size Mitigates Overfitting





[Bishop, 2006]

Large Training Data: ImageNet

http://image-net.org/synset?wnid=n02119789

- ImageNet
  - Over 15 million labeled high-resolution images
  - Roughly 22,000 categories
  - Collected from the web
  - Labeled by human labelers using Amazon's Mechanical Turk crowd-sourcing tool.
- ImageNet Large-Scale Visual Recognition Challenge (ILSVRC)
  - Uses a subset of imageNet
    - 1000 categories
    - 1.2 million training images
    - 50,000 validation images
    - 150,000 test images
  - Report two error rates:
    - Top-1 and top-5

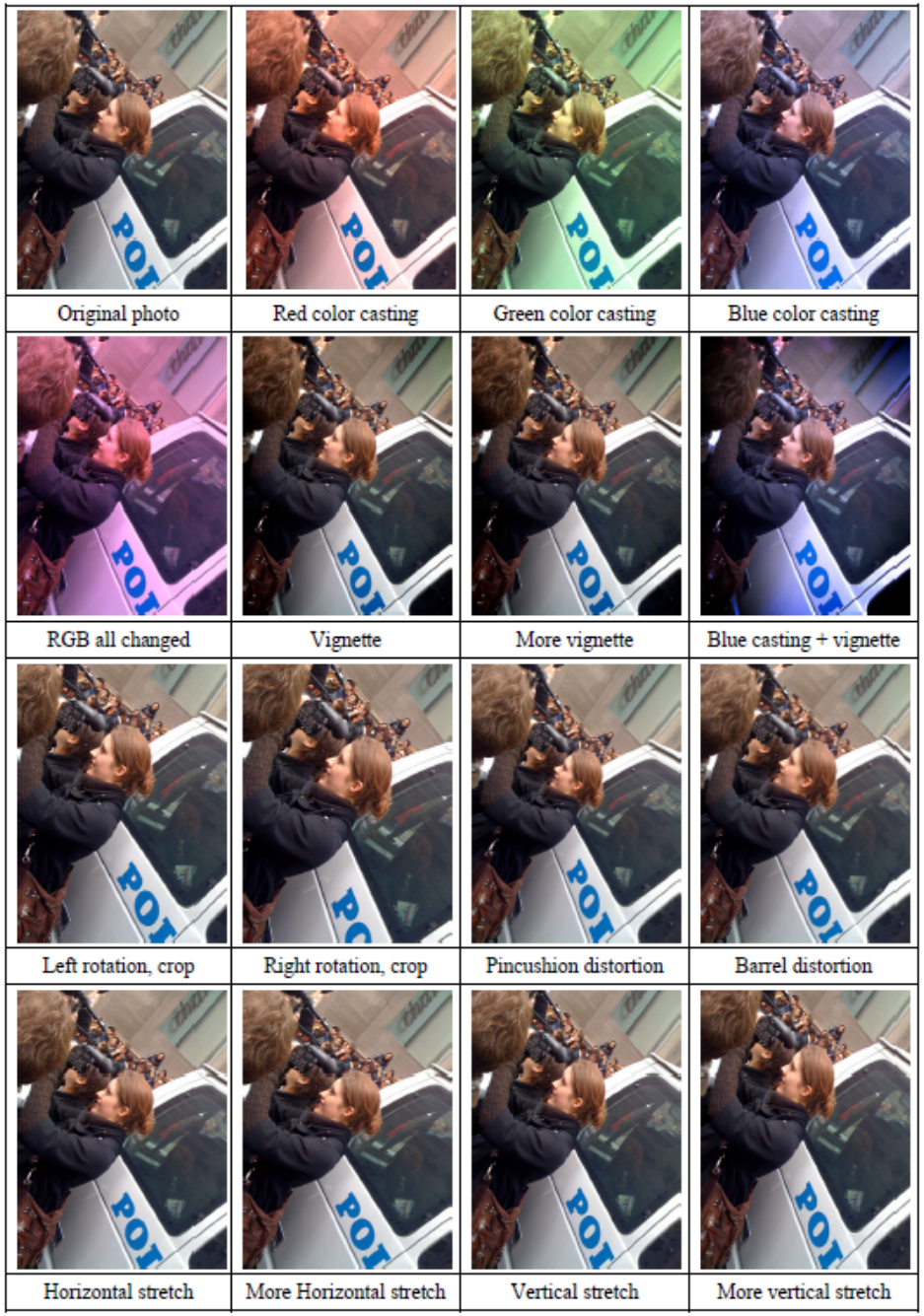
[Sungjoo Yoo]

Data Augmentation

- Data augmentation is a method to increase the effective size or diversity of training set using various transformation, such as rotation, flip, etc..

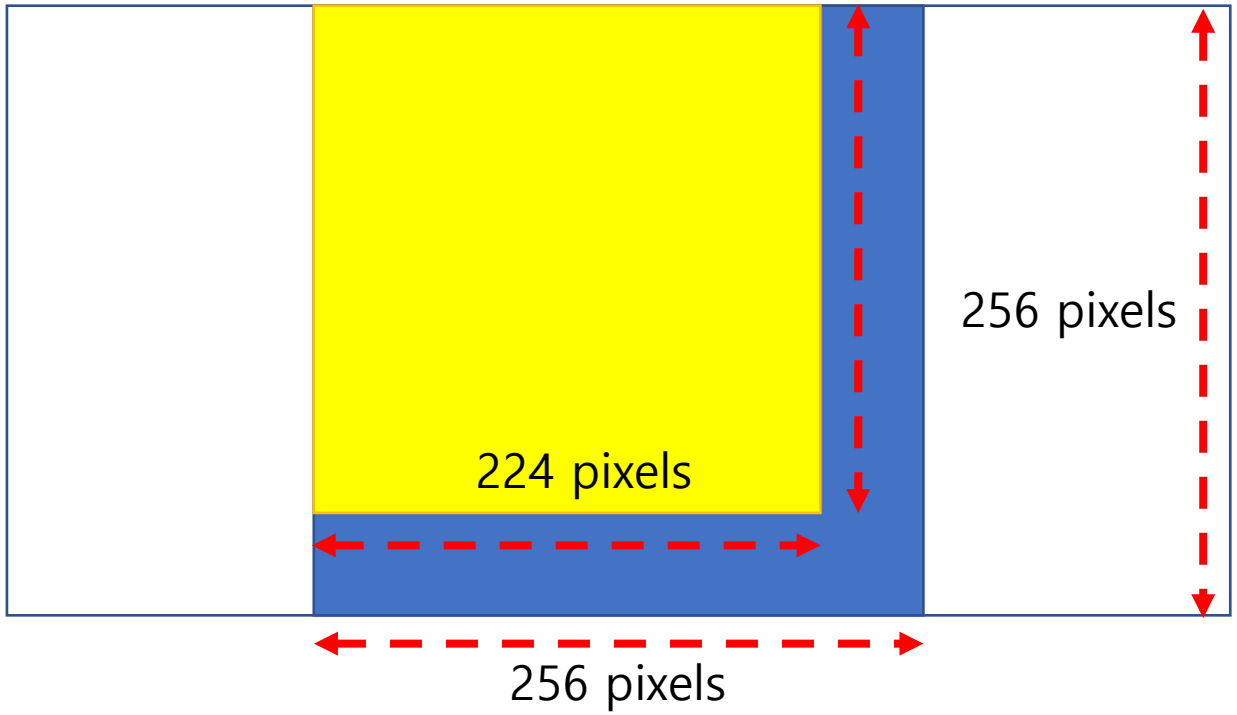
- Important for improving network performance

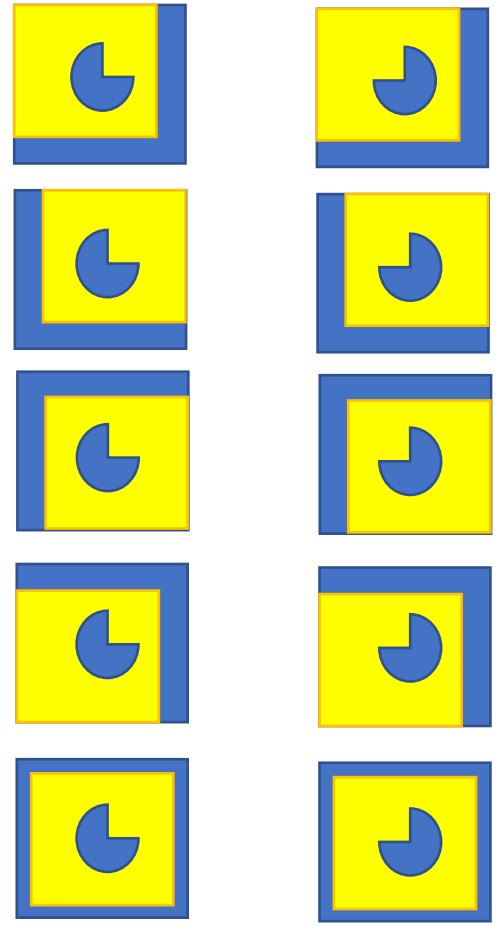
Operation Name	Description	Range of magnitudes
ShearX(Y)	Shear the image along the horizontal (vertical) axis with rate *magnitude*.	[-0.3,0.3]
TranslateX(Y)	Translate the image in the horizontal (vertical) direction by *magnitude* number of pixels.	[-150,150]
Rotate	Rotate the image *magnitude* degrees.	[-30,30]
AutoContrast	Maximize the the image contrast, by making the darkest pixel black and lightest pixel white.	
Invert	Invert the pixels of the image.	
Equalize	Equalize the image histogram.	
Solarize	Invert all pixels above a threshold value of *magnitude*.	[0,256]
Posterize	Reduce the number of bits for each pixel to *magnitude* bits.	[4,8]
Contrast	Control the contrast of the image. A *magnitude*=0 gives a gray image, whereas *magnitude*=1 gives the original image.	[0.1,1.9]
Color	Adjust the color balance of the image, in a manner similar to the controls on a colour TV set. A *magnitude*=0 gives a black & white image, whereas *magnitude*=1 gives the original image.	[0.1,1.9]
Brightness	Adjust the brightness of the image. A *magnitude*=0 gives a black image, whereas *magnitude*=1 gives the original image.	[0.1,1.9]
Sharpness	Adjust the sharpness of the image. A *magnitude*=0 gives a blurred image, whereas *magnitude*=1 gives the original image.	[0.1,1.9]
Cutout [12, 69]	Set a random square patch of side-length *magnitude* pixels to gray.	[0,60]
Sample Pairing [24, 68]	Linearly add the image with another image (selected at random from the same mini-batch) with weight *magnitude*, without changing the label.	[0, 0.4]



Data Augmentation for inference

- Data augmentation is also used for inference to maximize accuracy
- Ex) Data Augmentation in AlexNet
  - Given an original image, first resize it to fit 256 pixel to the smaller side
  - Crop 10 224x224 patches (4 corner and 1 center crops)
  - Prediction based on the average of 10 softmax outputs





[Sungjoo Yoo]