Python

Machine Learning

Chap.6

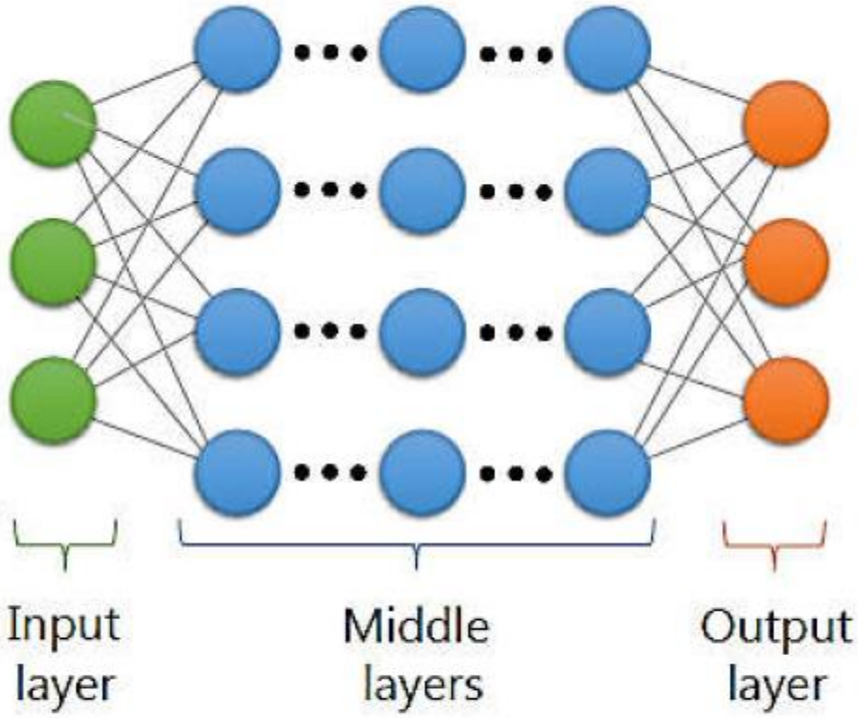
What is Deep Learning?

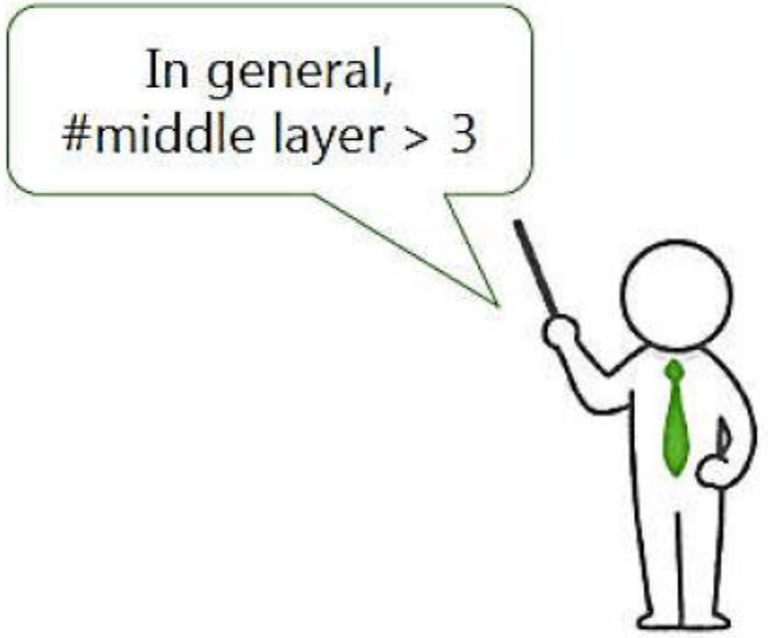
- ***Definition* 1**: A class of machine learning techniques that exploits **many layers of non-linear information processing (via non-linear activation function)** for **feature extraction** and for **pattern analysis** and **classification**.

- MLP seems can do this …?!

Deep Neural Network = MLP with more than one hidden layer?

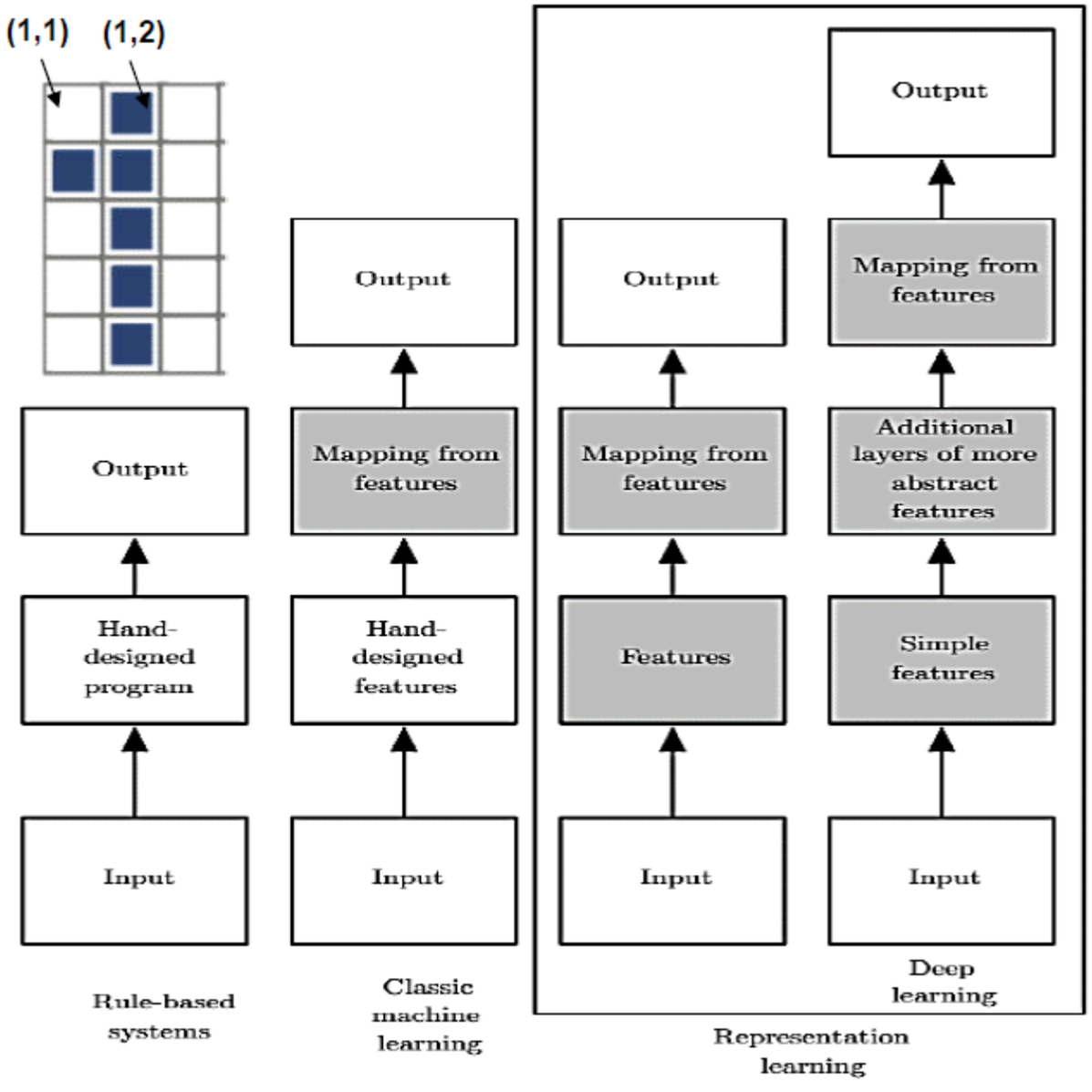
Deep Neural Networks





Deep Neural Network has a lot of middle (hidden) layers!

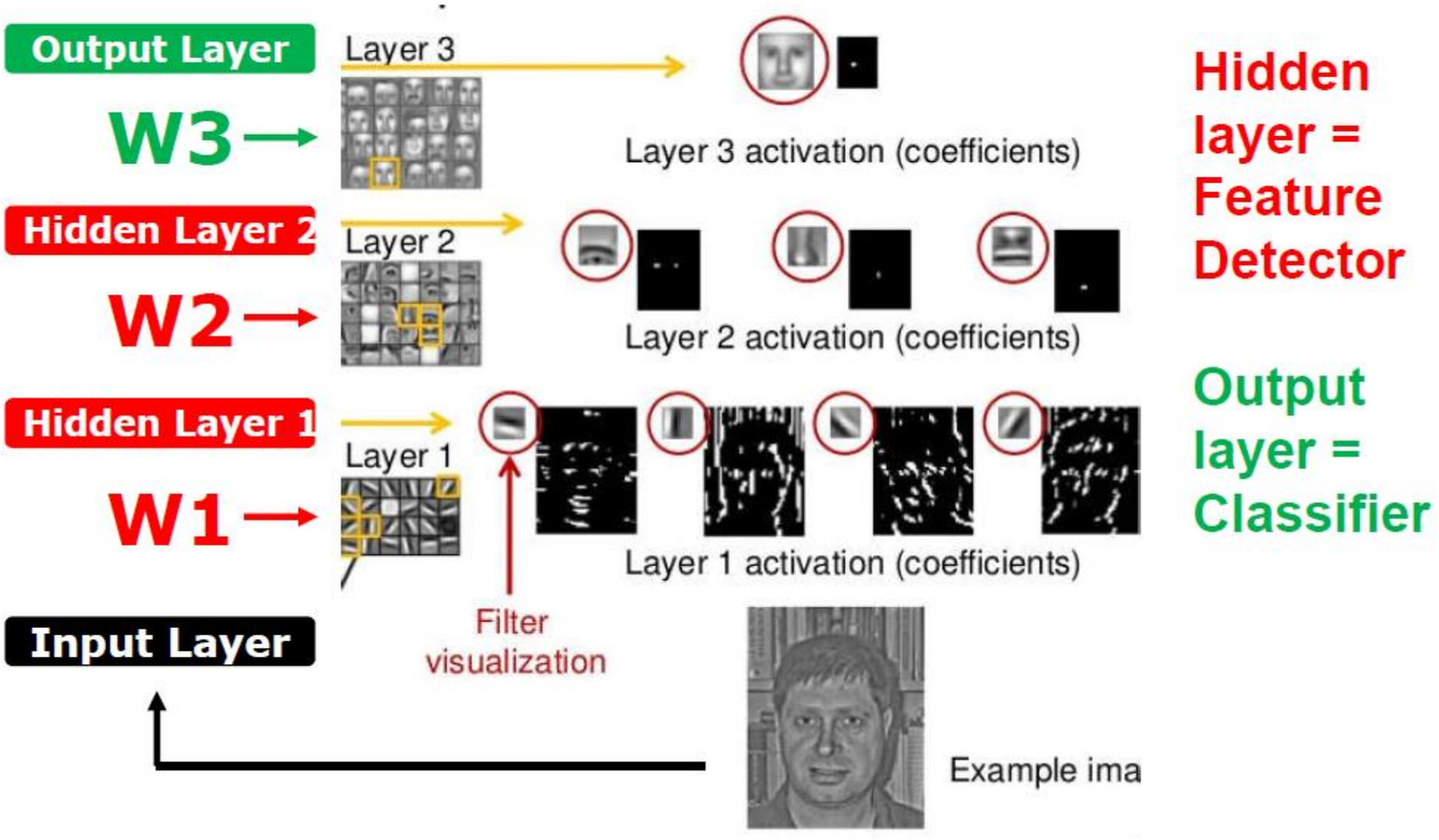
Deep Neural Networks



Shaded boxes indicate components that are able to learn from data.

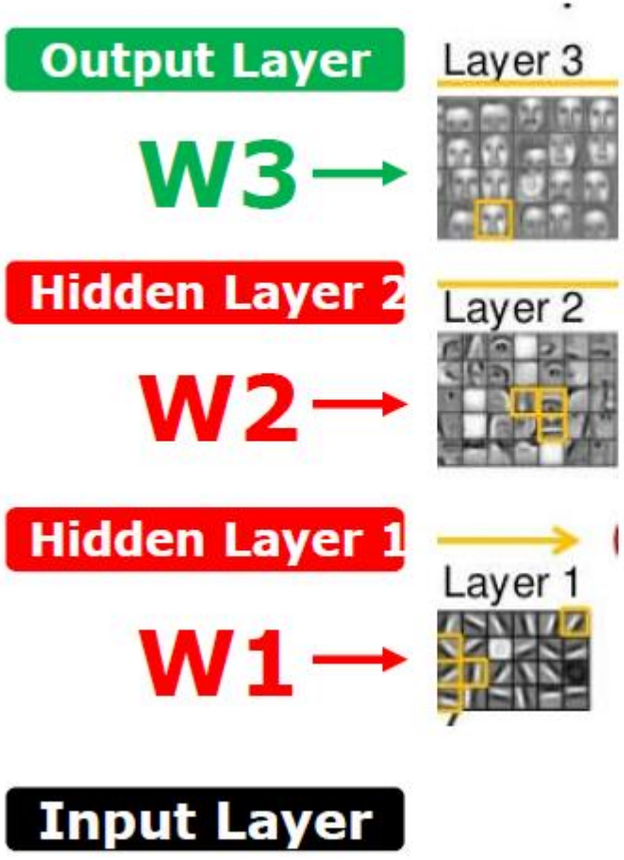
Deep learning = multiple stages representation learning

Deep Learning Perspective



Deep Learning Perspective

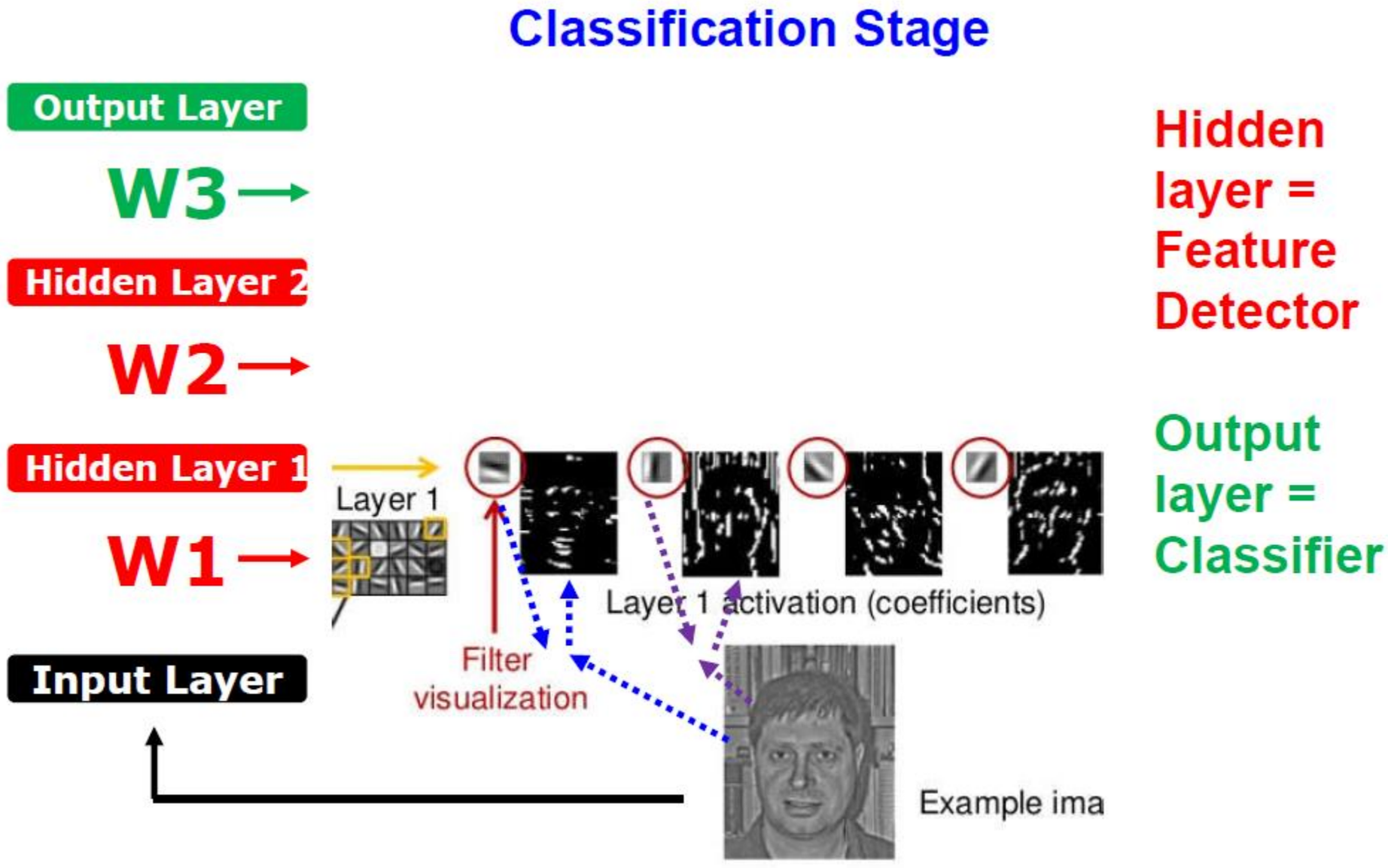
Training Stage



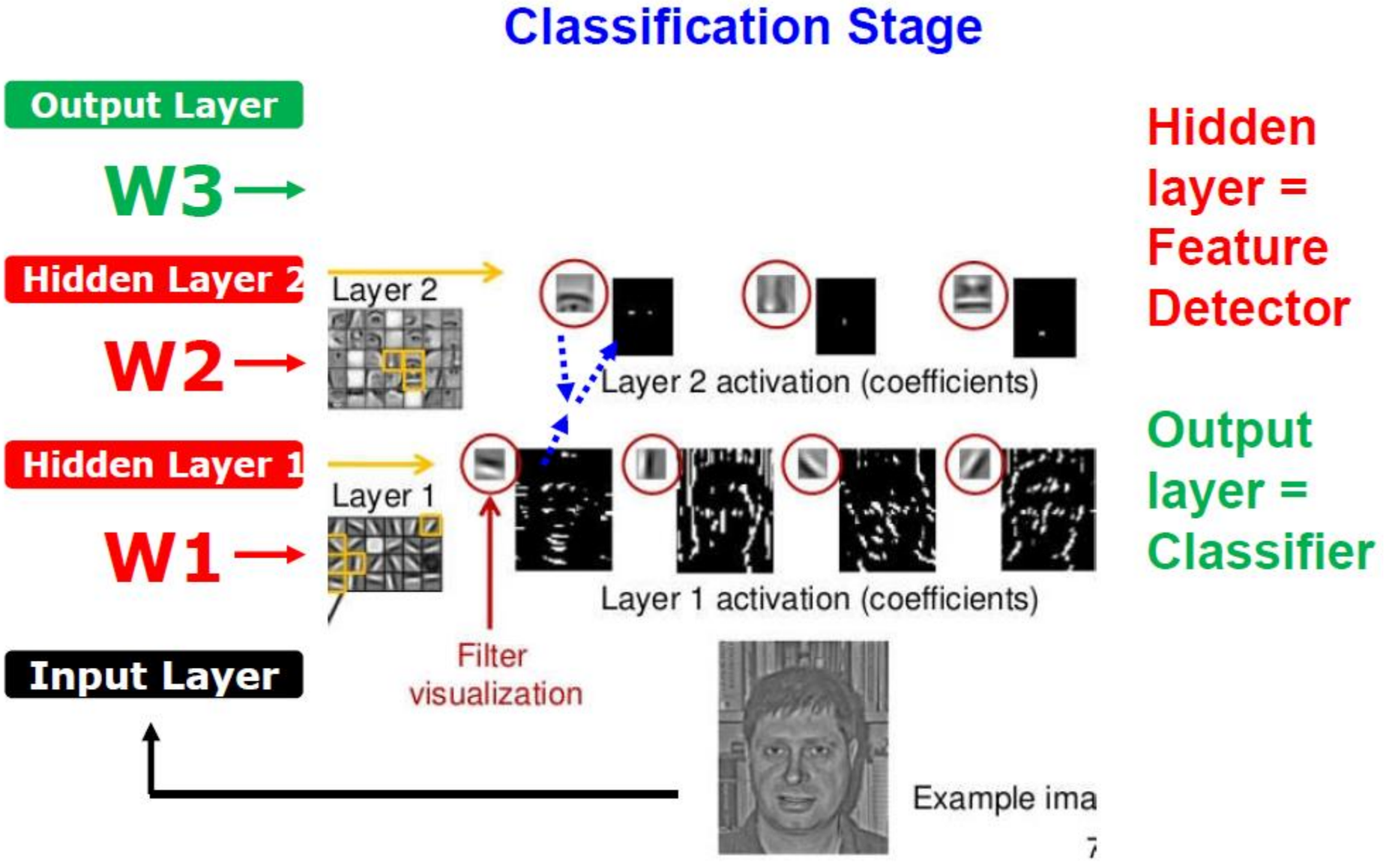
Hidden layer = Feature Detector

Output layer = Classifier

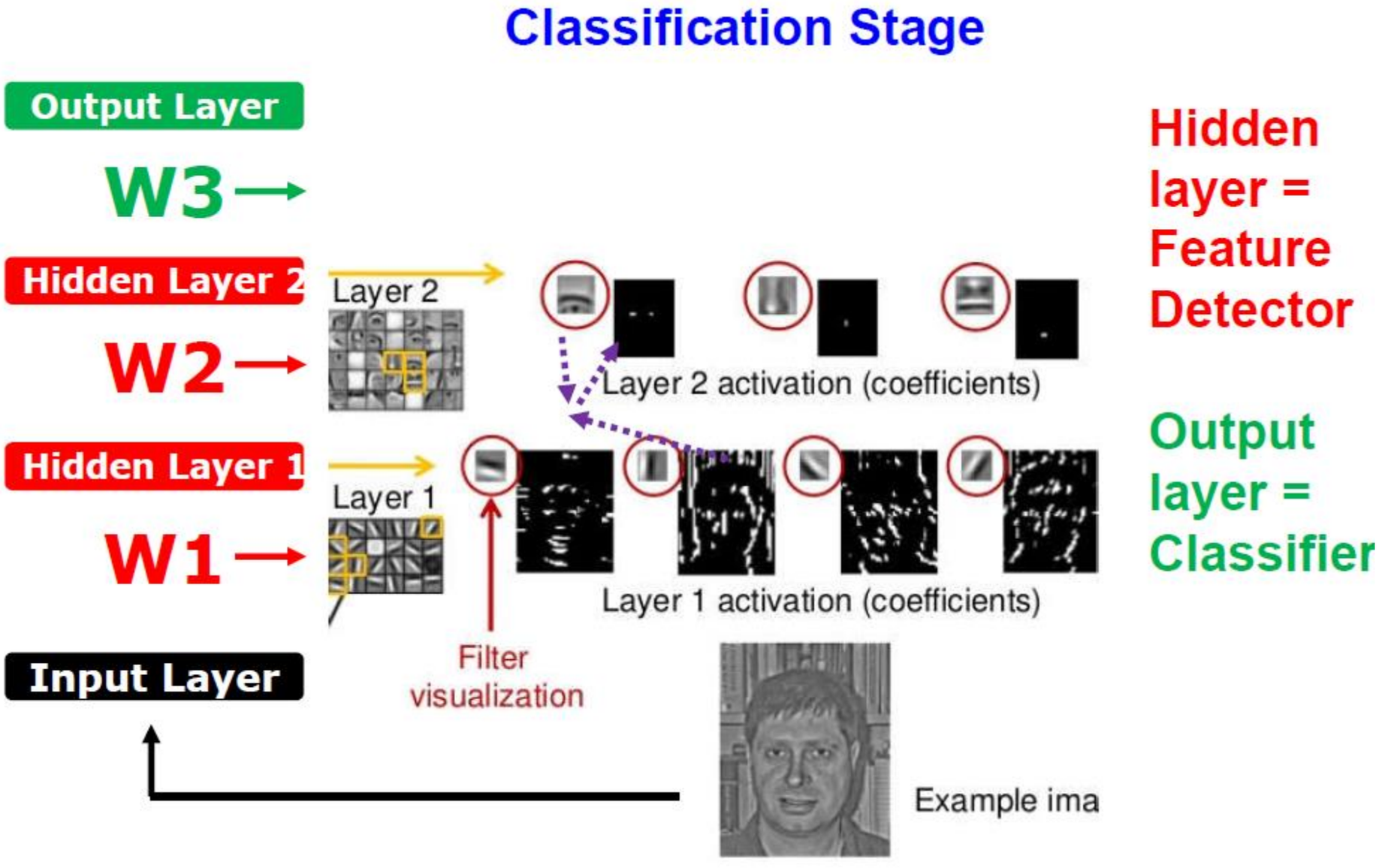
Deep Learning Perspective



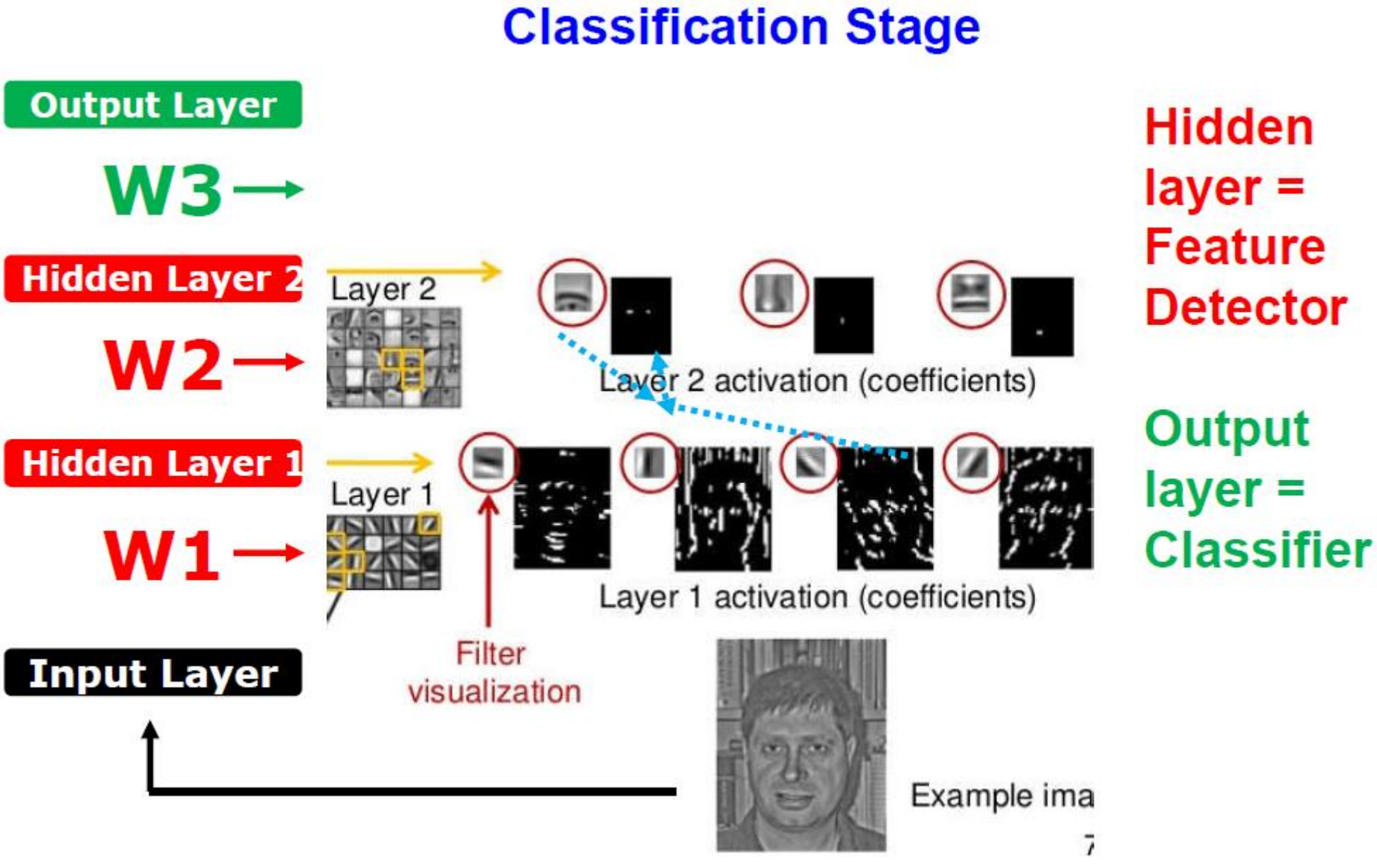
Deep Learning Perspective



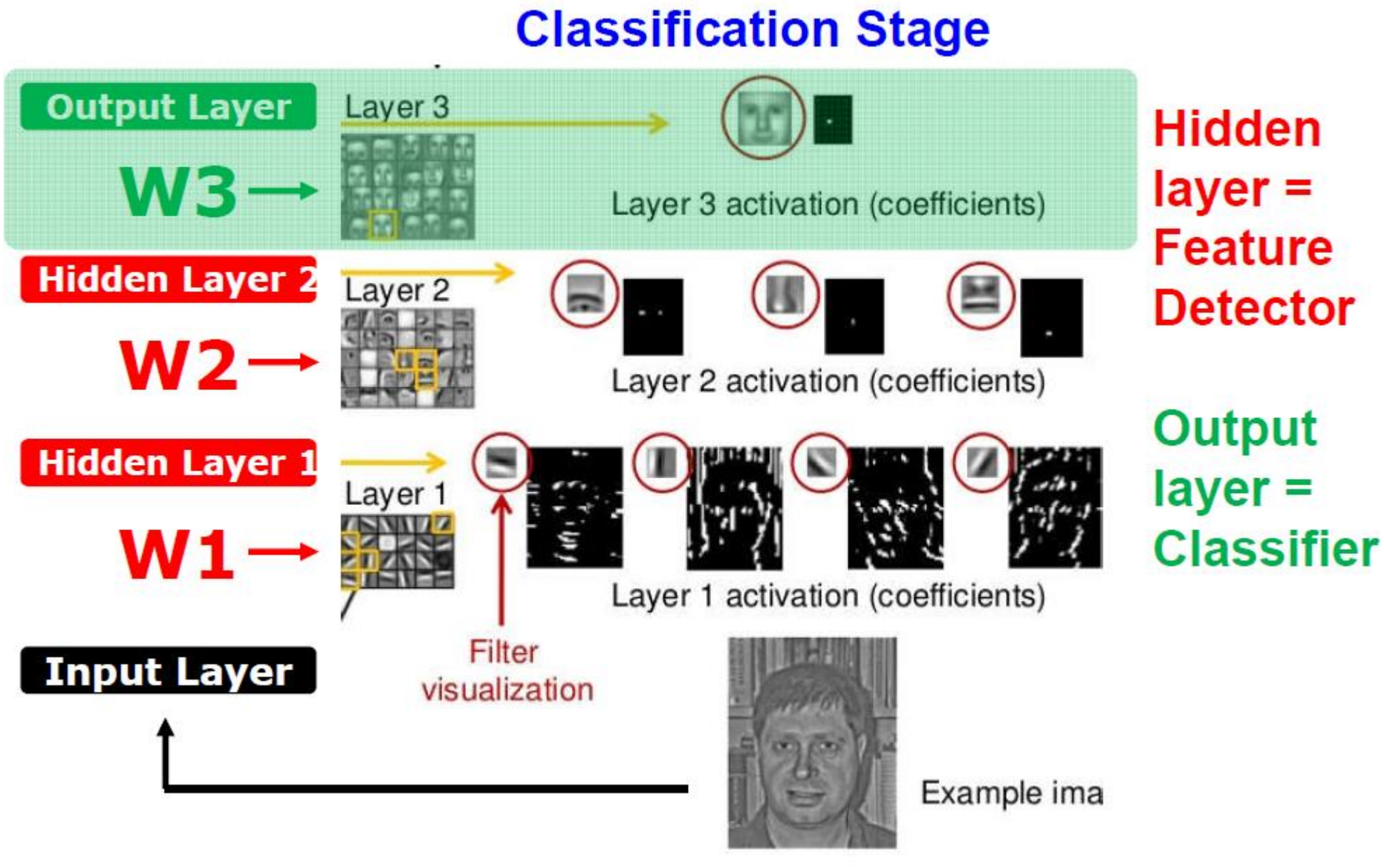
Deep Learning Perspective



Deep Learning Perspective



Deep Learning Perspective



Optimization

Validation data

$$D_N = D_{train} \cup D_{test} \cup D_{valid}$$

- 분할된 데이터의 역할
  - 학습용training
    - 모델(의 매개변수)를 조정하는 데 직접 사용
  - 테스트test
    - 모델의 최종 성능을 측정하는 데 사용
    - **학습에 쓰이지 않음**
  - 검증용validation data
    - 모델의 성능을 중간에 점검할 때 사용
    - 모델을 직접적으로 조정하는 데 쓰이지 않으나 간접적으로 학습에 기여할 수 있음

$$D_{train} = \{(\mathbf{x}^{(d)}, y^{(d)})\}_{d=1}^{A}$$

$$D_{test} = \{(\mathbf{x}^{(d)}, y^{(d)})\}_{d=1}^{B}$$

$$D_{valid} = \{(\mathbf{x}^{(d)}, y^{(d)})\}_{d=1}^{C}$$

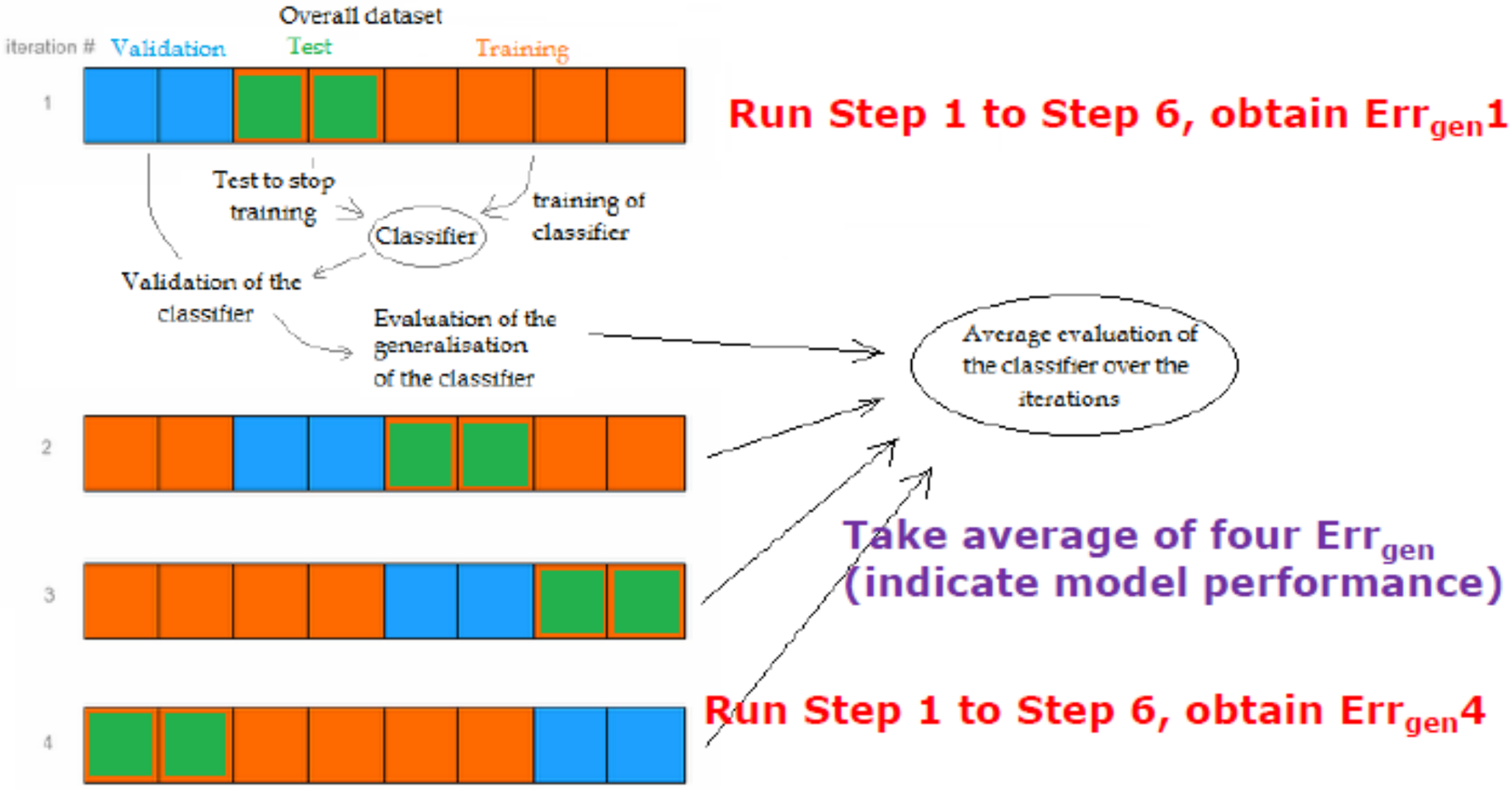
- 데이터를 분할할 때에는 각 데이터가 갖는 **분포가 유사해야** 함

How do you determine performance?

1. Divide the data set into a training set $T_{learning}$ and a test set T_{test} .
2. Subdivide $T_{learning}$ into two subsets: one to train the network $T_{training}$, and one to validate the network $T_{validation}$.
3. Train different network architectures (eg. # of hidden nodes) on $T_{training}$ and evaluate their performance on $T_{validation}$.
4. Select the best network which does not over or underfit.
5. Finally, retrain this network architecture on $T_{learning}$.
6. Test for generalization ability using T_{test}.

- Repeat above process for N times by swapping three subsets (training, validation and test) – N-fold cross validation protocol.

K fold validation



4-fold Cross-validation protocol (4 iterations)
(N-iterations, called N-fold cross validation)

Question

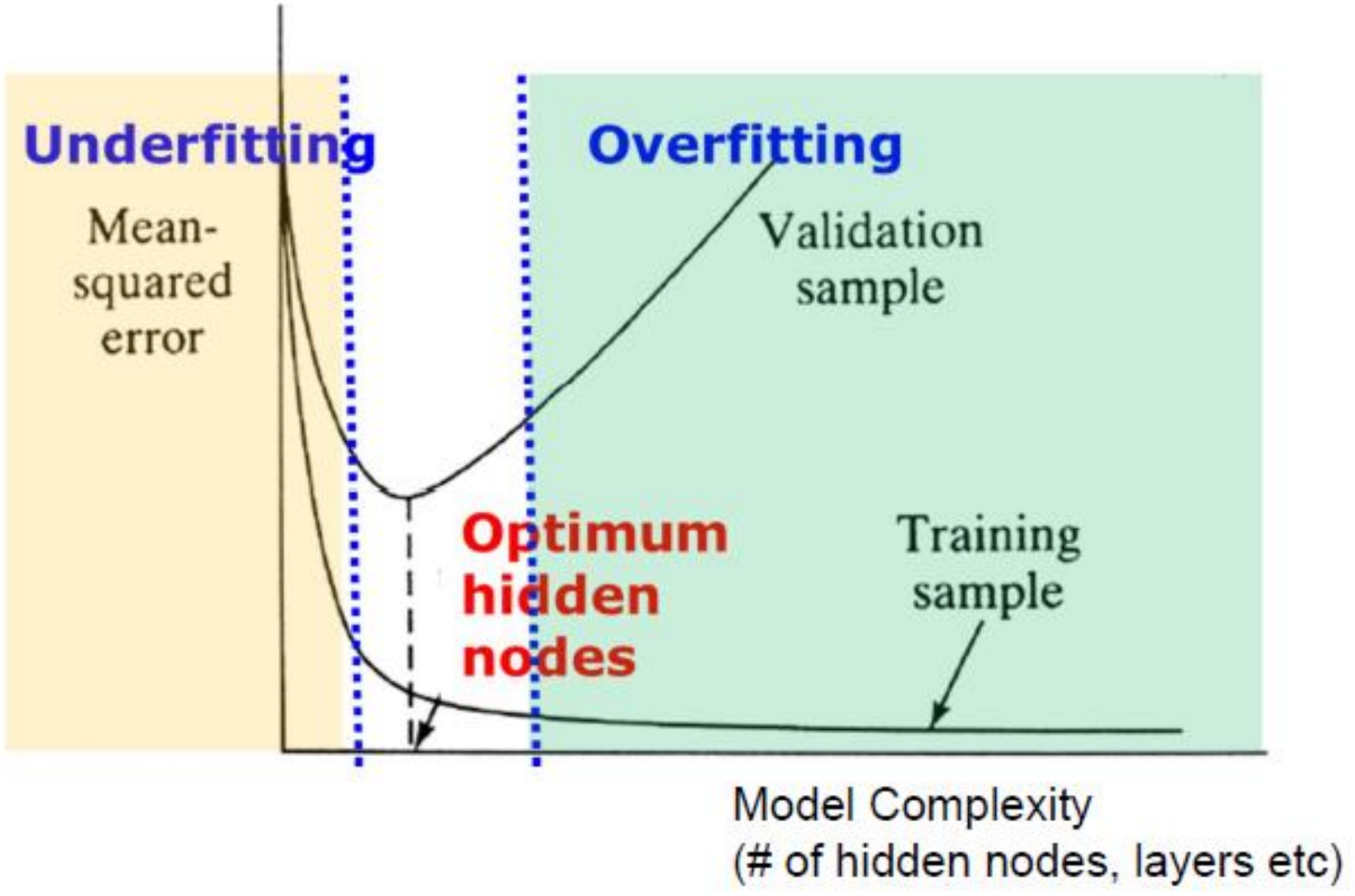
- “But the best hyperparameter can be different for each fold!

- So which value for the hyperparameter do I choose?”
  - None of them, we are not doing model selection (hyperparameter tuning)! We will go for another protocol.

Hyperparameter Search (Model selection) Protocol

1. Use training set and validation set **ONLY**. **Leave testing set alone.**
2. Train different hyperparameters (eg. # of hidden nodes) on $T_{training}$ and evaluate their performance on $T_{validation}$.
3. Record the Err_{val} for each hyperparameter,
4. Reshuffle training and validation set, repeat 2 and 3. (2^{nd} CV)
5. Repeat 4 for n times (n-fold CV)
6. Average the Err_{val} obtained from n-fold, the hyperparameter set that yield smallest average Err_{val} indicate best hyperparameter set.

How to determine the best model?

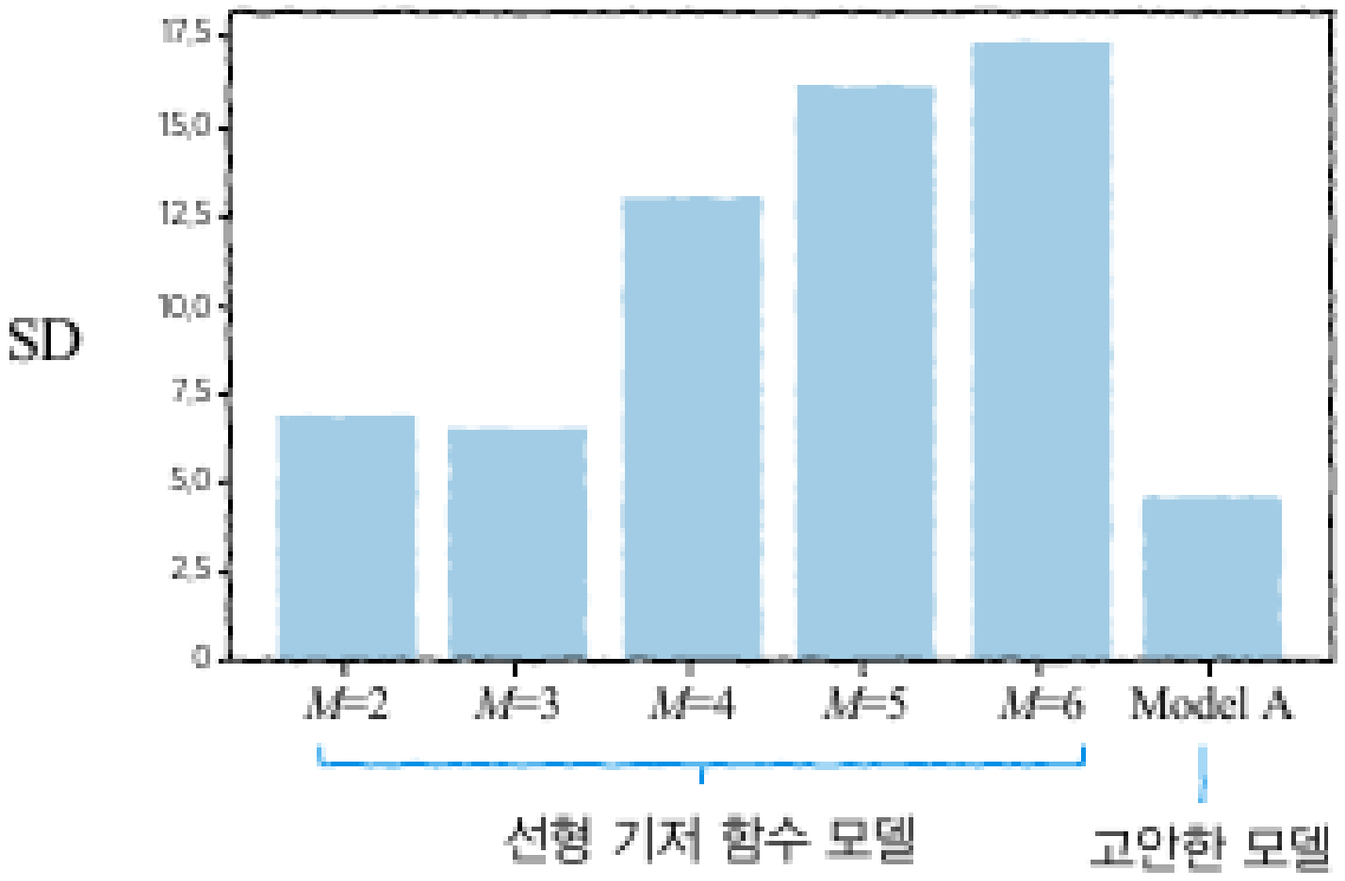


K fold validation

Partition the dataset into learning set and test set
for fold = 1 to n *(CV for Err_{test} estimation)*
  partition learning set to training and validation sets
  for fold = 1 to n *(CV for model selection)*
    for h=1 to H
      * train the model with h^{th} parameters
      * evaluate on validation set
      * record Err_{val}
  & compute the mean of Err_{val} for each parameter value
  & select the model (best parameter value) with lowest Err_{val} ✓
  & use the selected model to evaluate testing set
  & record Err_{test}
report the means of the Err_{test} ✓

(Nested CV)

K fold validation



Hyperparameters Searching Strategies

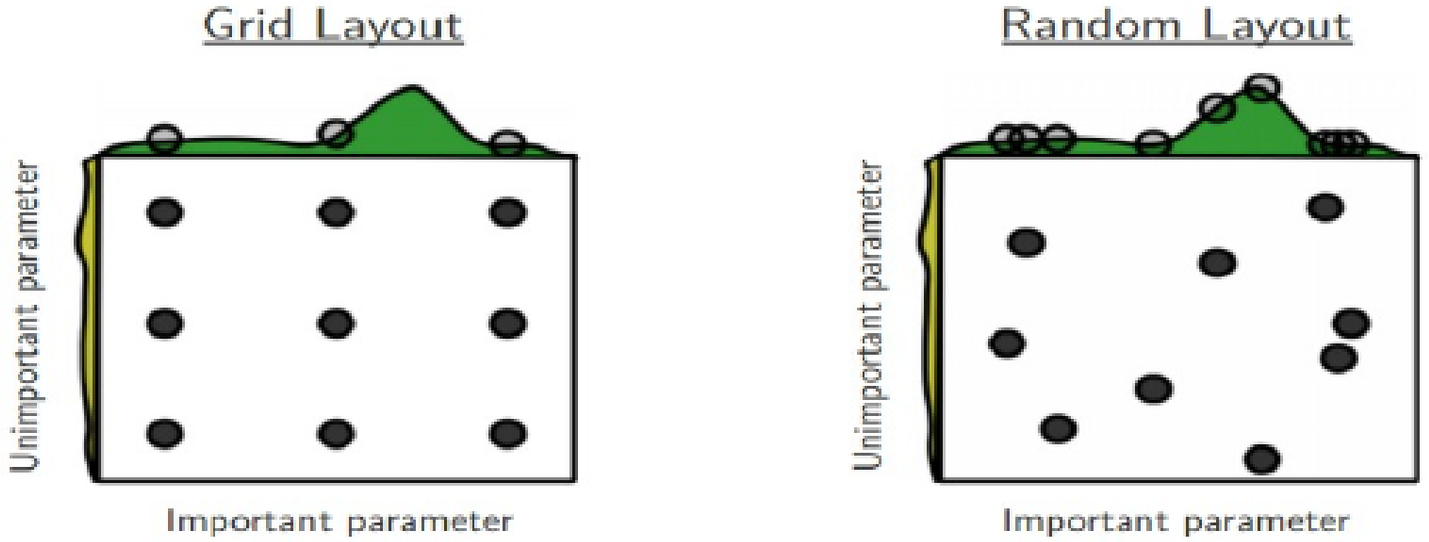
- **Babysitting (aka Trial & Error)**
  - 100% manual method
  - widely adopted by researchers, students, and hobbyists
  - **Workflow:** design a new experiment that follows through all the steps of the learning process (from data collection to feature map visualization), then iterates sequentially on the parameters until runs out time (usually due to a deadline) or motivation.

Hyperparameters Searching Strategies

- **Grid Search**
  - Define a grid on *q* dimensions, where each of these maps for an parameter.
    - [hidden_neuron, hidden_layer, learning_rate] (*q*=3)
  - For each dimension, define the range (*p*) of possible values:
    - hidden_neuron = [4, 8, 16, 32, 64, 128, 256] (p=7).
  - Search for all the possible configurations and wait for the results to establish the best one:
    - E.g. *C1* = (4, 1, 0.001) -> acc = 92%, *C2* = (8, 2, 0.05) -> acc = 92.3%, etc...
  - mp^q configurations of search for *p* points, *q* param, m-fold.

Hyperparameters Searching Strategies

- **Random Search**
  - picks the point randomly from the configuration space.
  - Don't use Grid Search if q>3. Instead, use Random Search, which provides a really good baseline for each searching task.



q=2, p=3, m=1 (p^q =9 possible configurations)

http://www.jmlr.org/papers/volume13/bergstra12a/bergstra12a.pdf

Hyperparameters Searching Strategies

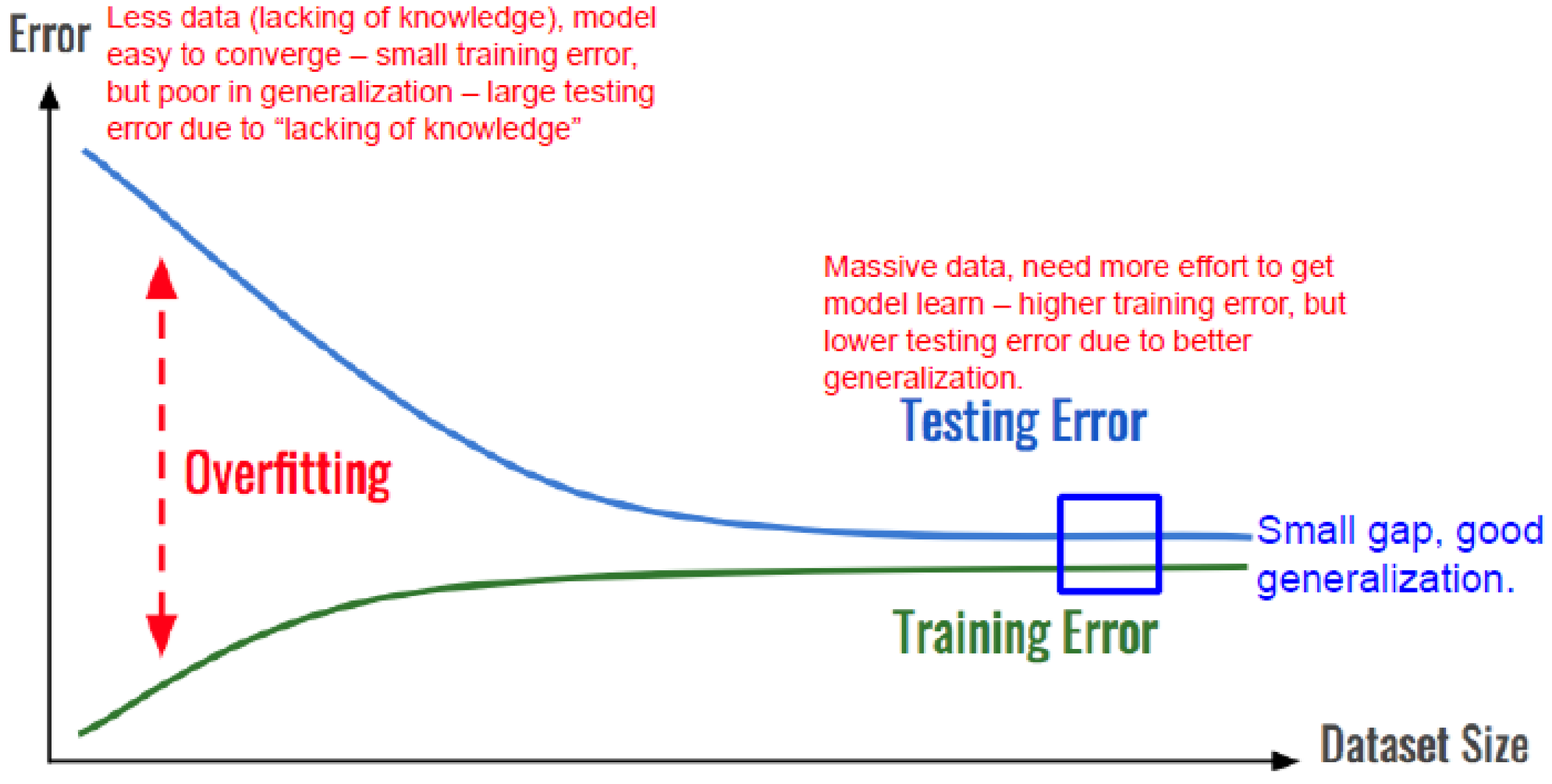
Approach	*ML*	*DL*	*Manual/ Auto*	*Cost*	*Space expl.*	*History*	*Parallel/ Distributed*
Babysitting	👍	👎	🐼	💰	Low	✅	No
Grid	👍	👎	💻	💰 💰 💰 💰	High	🙈	Yes
Random	👍	👍	💻	💰 💰 💰	Medium	🙈	Yes

Practical tricks to deal with generalization

- Overfitting can be reduced via model selection protocol, but it is often insufficient!

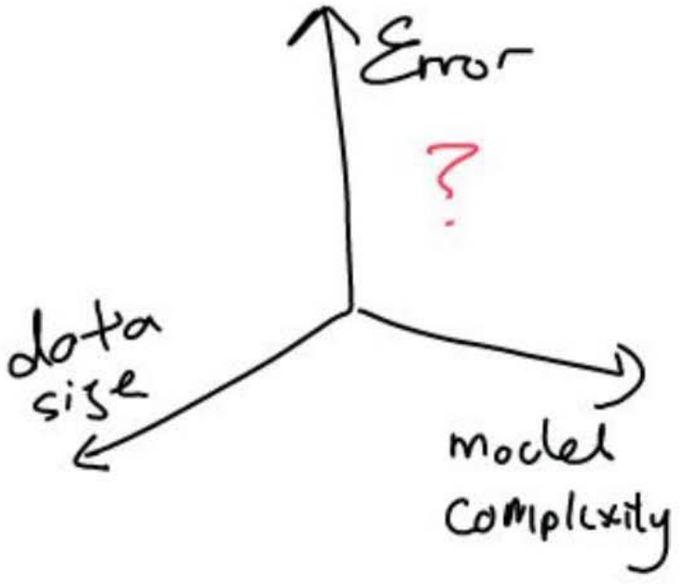
- Any other methods to reduce overfitting problem?
  - Large and representative training data!! (utmost important) – Data Augmentation
  - Early stopping
  - Regularization – L2, L1, Max norm constraint, Dropout etc
  - Ensembles

1. Enrich Training Data



Training err getting larger when data size becomes large.

1. Enrich Training Data



Small data - simple model : Less overfitting

Small data - complex model : overfitting (Conventional ML)

Large data - simple model : underfitting

Large data - complex model : Less overfitting

1. Enrich Training Data

- Use a lot of good quality and representative training data you are less likely to have overfitting!
  - But this could be difficult in practice.

- Create artificial training data to enlarge training data pool – **Data augmentation**

1. Enrich Training Data

- Random mix/combinations of :
  - translation
  - rotation
  - stretching
  - shearing,
  - lens distortions, … (go crazy)

1. Enrich Training Data

- Can augmentation help even if I have lots of data?
  - Yes. It can help to increase the amount of relevant data in your dataset.

Training data: (Left) Brand A, all facing left and (Right) Brand B, all facing right

Testing data: A or B?
True label is A

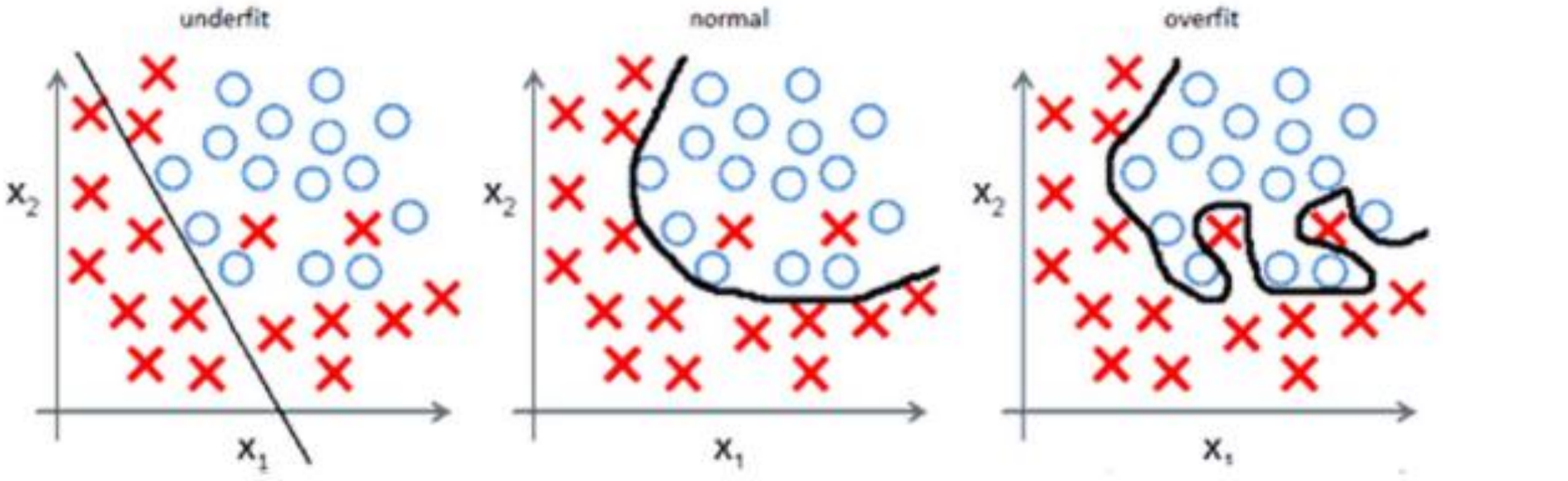
1. Enrich Training Data

- If I use ALL of these techniques, my network would be robust right?
  - If you use it in the right way, then yes!

The first image (from the left) is the original, the second one is flipped horizontally, the third one is rotated by 180 degrees, and the last one is rotated by 90 degrees (clockwise).

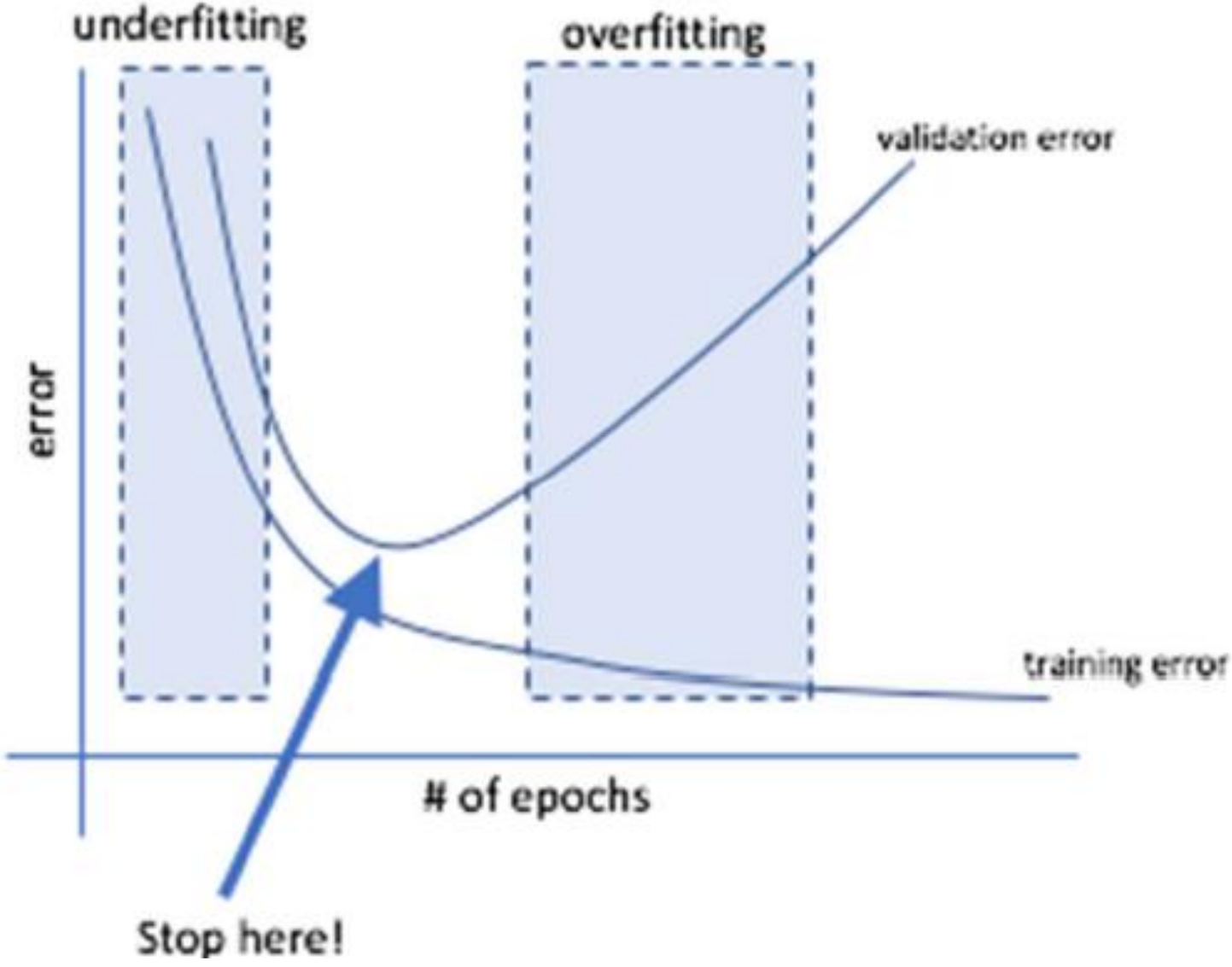
2. Early stopping – When Do We Stop Training?

- Would it be good to press loss function (eg. MSE) to zero in practice?
  - Zero MSE implies Zero training error.



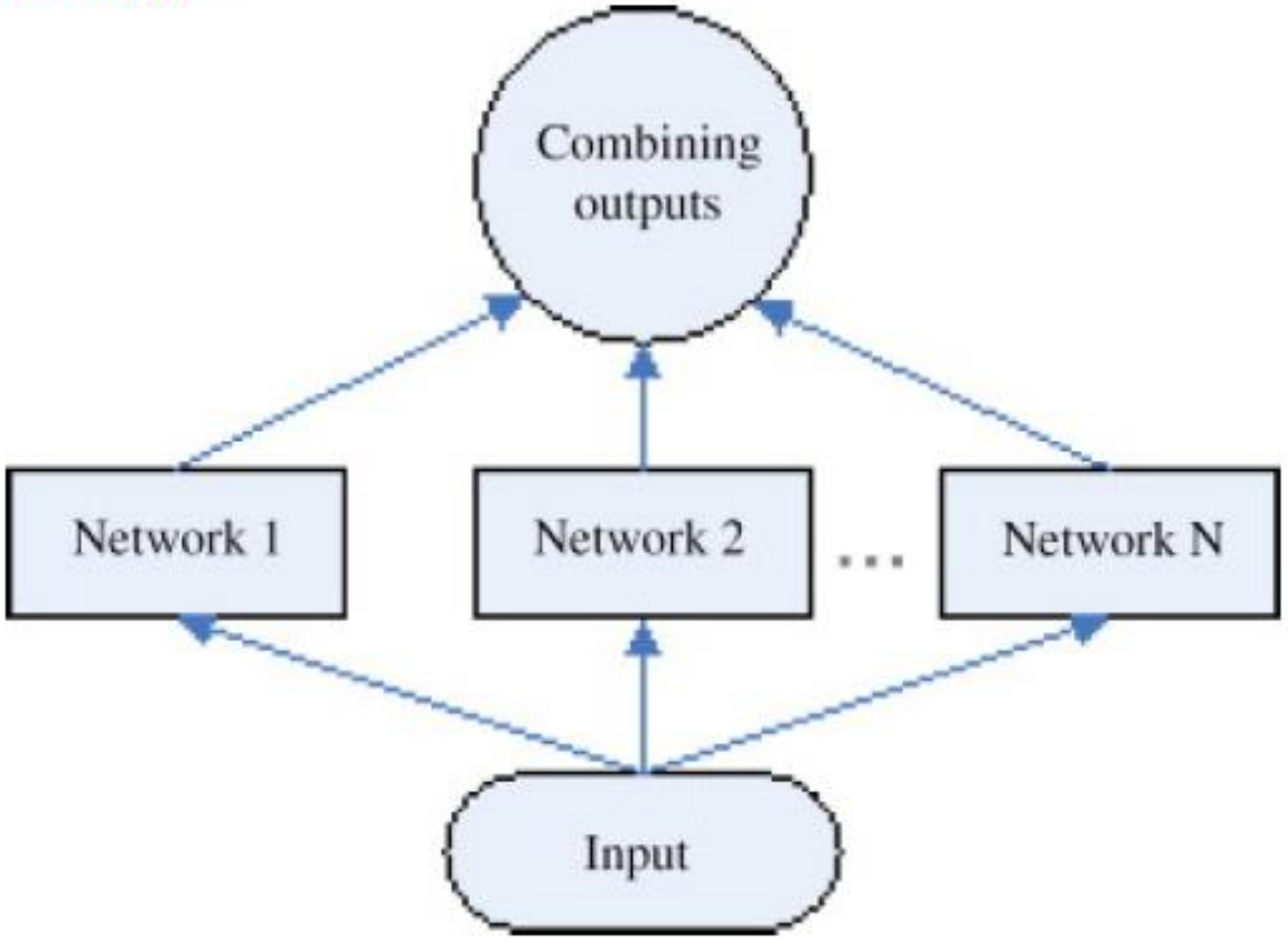
Zero training error (risk for overfitting)

2. Early stopping – When Do We Stop Training?



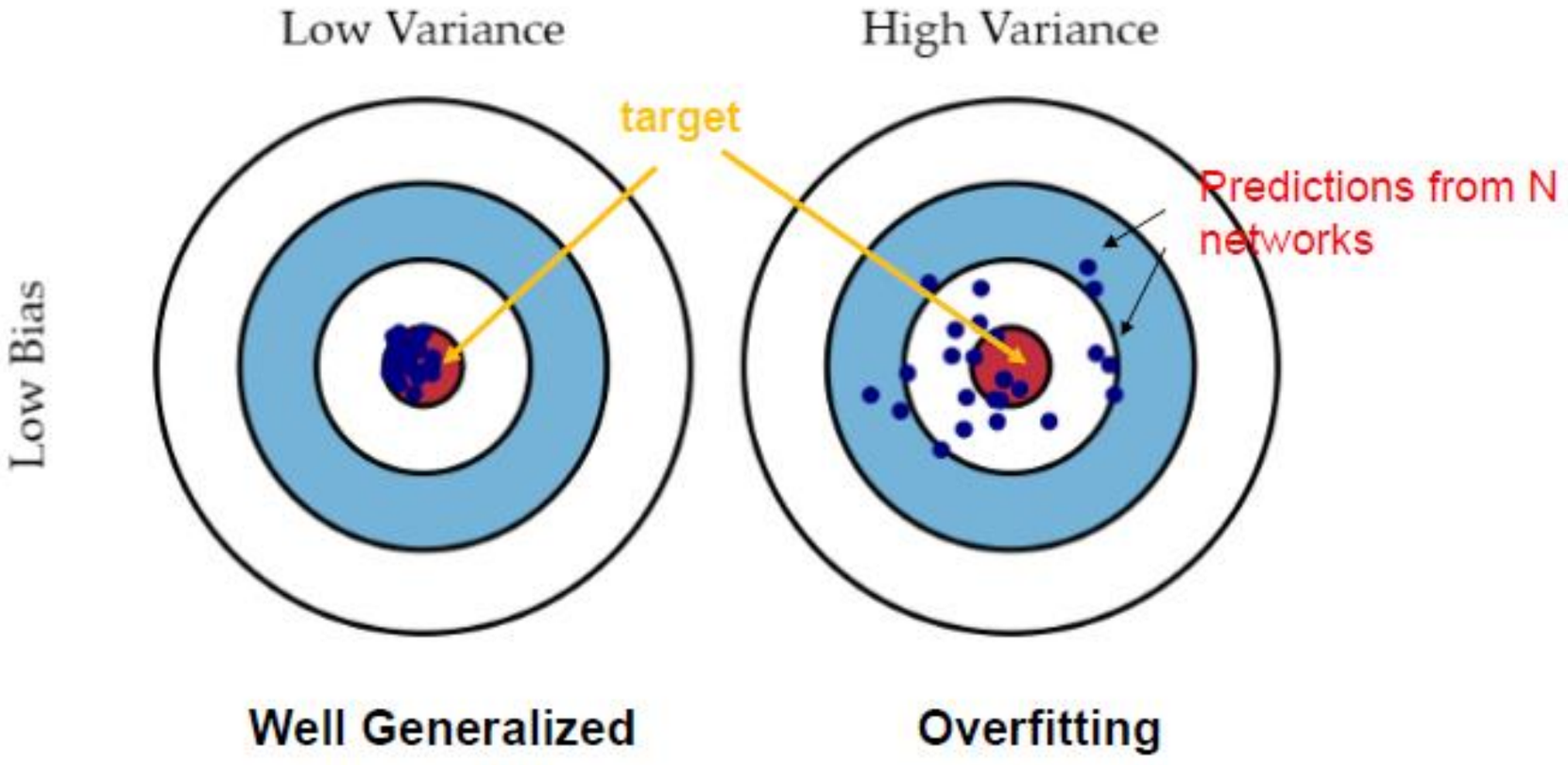
3. Ensembles

- Train multiple models, give input to trained models and take average.



3. Ensembles

- Why Ensemble is useful to reduce overfitting?



3. Ensembles

- **Strategy 1:** Same model, different initialization.
  - Use cross-validation to determine the best hyperparameters, then train multiple models with the best set of hyperparameters but with different random initializations.
  - The danger with this approach is that the variety is only due to initialization.

3. Ensembles

- **Strategy 2:** Top models discovered during cross-validation.
  - Use cross-validation to determine the best hyperparameters, then pick the top few (e.g., 10) models to form the ensemble.

    (Recall model selection protocol)
  - This improves the variety of the ensemble but has the danger of including suboptimal models.
  - In practice, this can be easier to perform since it does not require additional retraining of models after cross-validation.

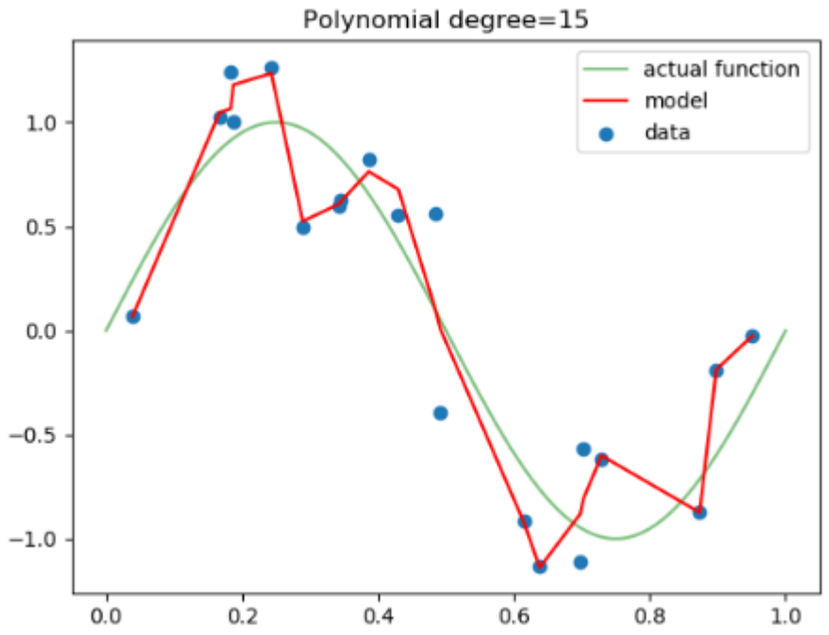
3. Ensembles

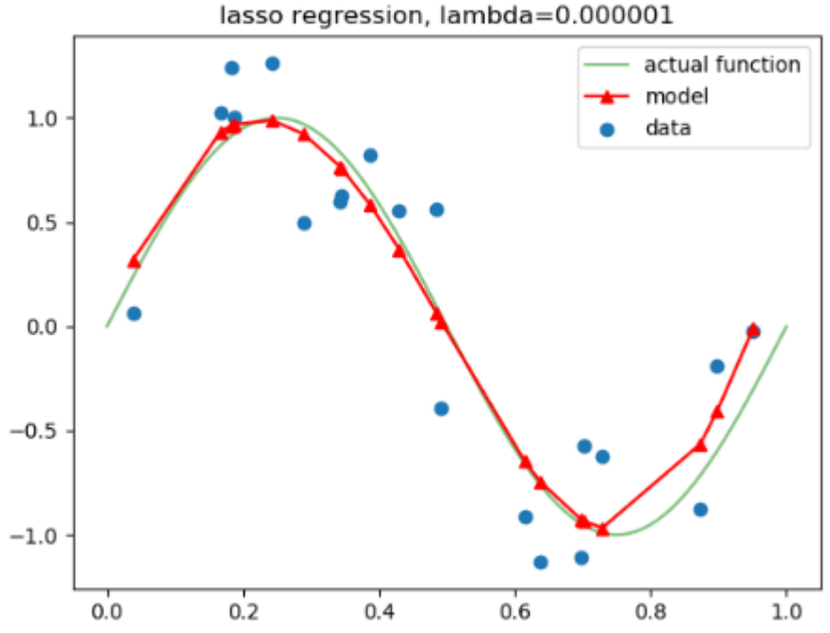
- **Strategy 3:** Different checkpoints of a single model.
  - If training is very expensive, some people have had limited success in taking different checkpoints of a single network over time (for example after every epoch) and using those to form an ensemble.
  - Clearly, this suffers from some lack of variety, but can still work reasonably well in practice.
  - The advantage of this approach is that is very cheap.

4. Regularization

- Regularization is a technique that builds a penalty function (regularizer) into the loss function itself

$$\tilde{\mathcal{E}} = \mathcal{E} + \alpha\Omega$$





4. Regularization– L2

- L2 regularization/Weight Decay – penalize the weight magnitude.
- α is a weight regularization parameter.
- A weight decay regularizer needs to treat both input-hidden and hidden-output weights differently in order to work well

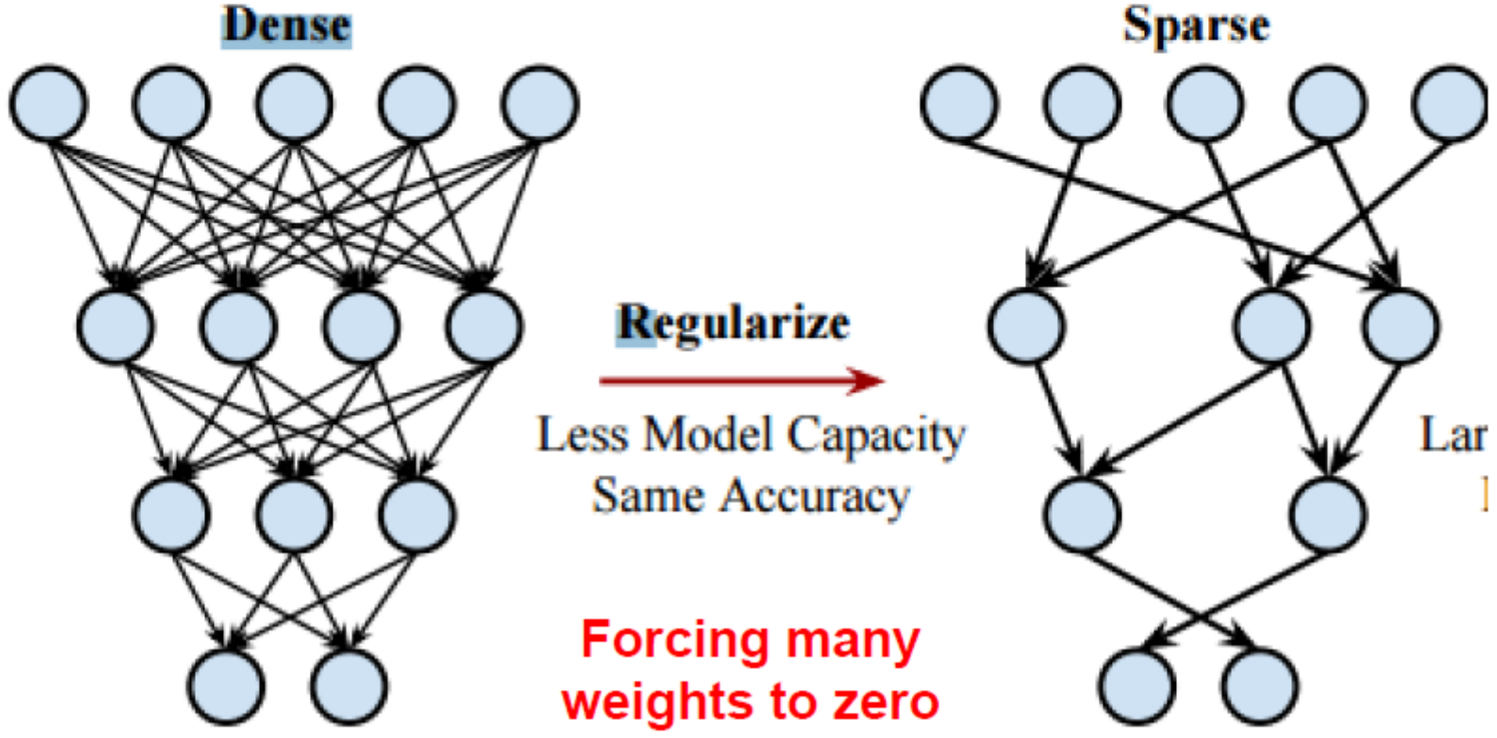
$$\tilde{\mathcal{E}} = \mathcal{E} + \alpha\Omega$$
$$= \mathcal{E} + \alpha\left(\frac{1}{2}\sum_i w_i^2\right)$$
$$= \mathcal{E} + \boldsymbol{\alpha}|\boldsymbol{w}|_2$$

$$\Omega = \frac{\alpha_1}{2}\sum_i\sum_h w_{ih}^2 + \frac{\alpha_2}{2}\sum_h\sum_j w_{hj}^2$$

4. Regularization– L1

- L1 Regularization - impose sparsity constraint,
  - Sparsity is useful to reduce the number of weights in network.

$$\tilde{\mathcal{E}} = \mathcal{E} + \alpha|w|_1$$



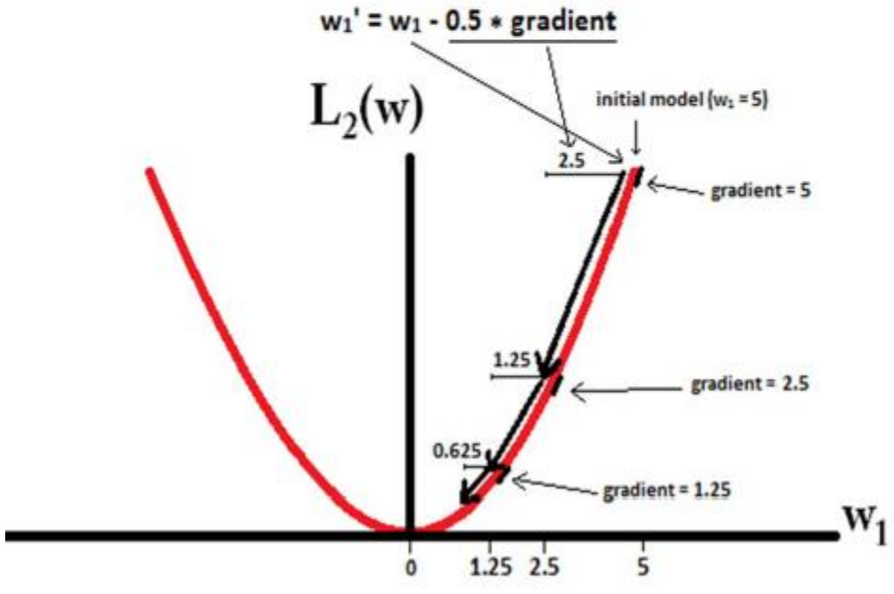
L2 vs L1

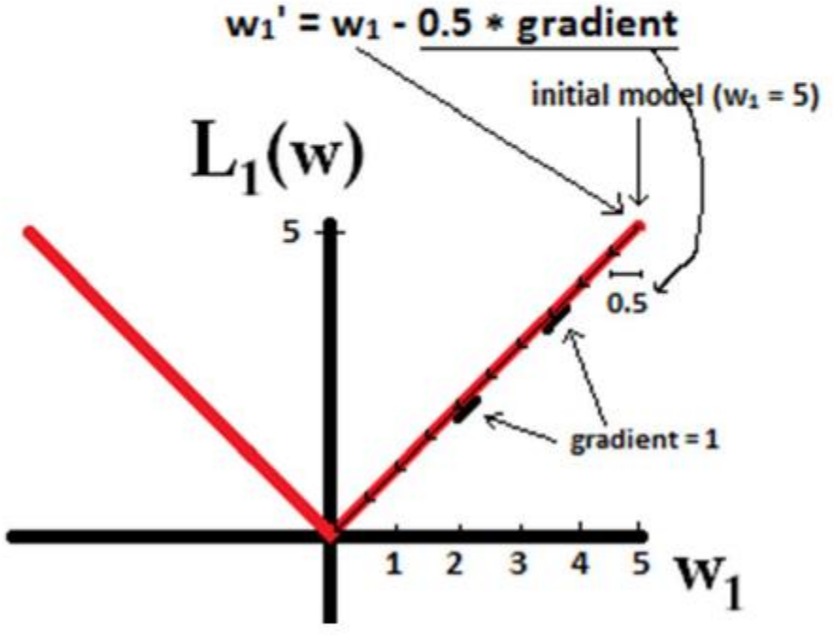
$$L_1(w) = \Sigma_i |w_i|.$$

$$\frac{dL_1(w)}{dw} = sign(w),$$

$$L_2(w) = \frac{1}{2}\Sigma_i w_i^2$$

$$\frac{dL_2(w)}{dw} = w$$



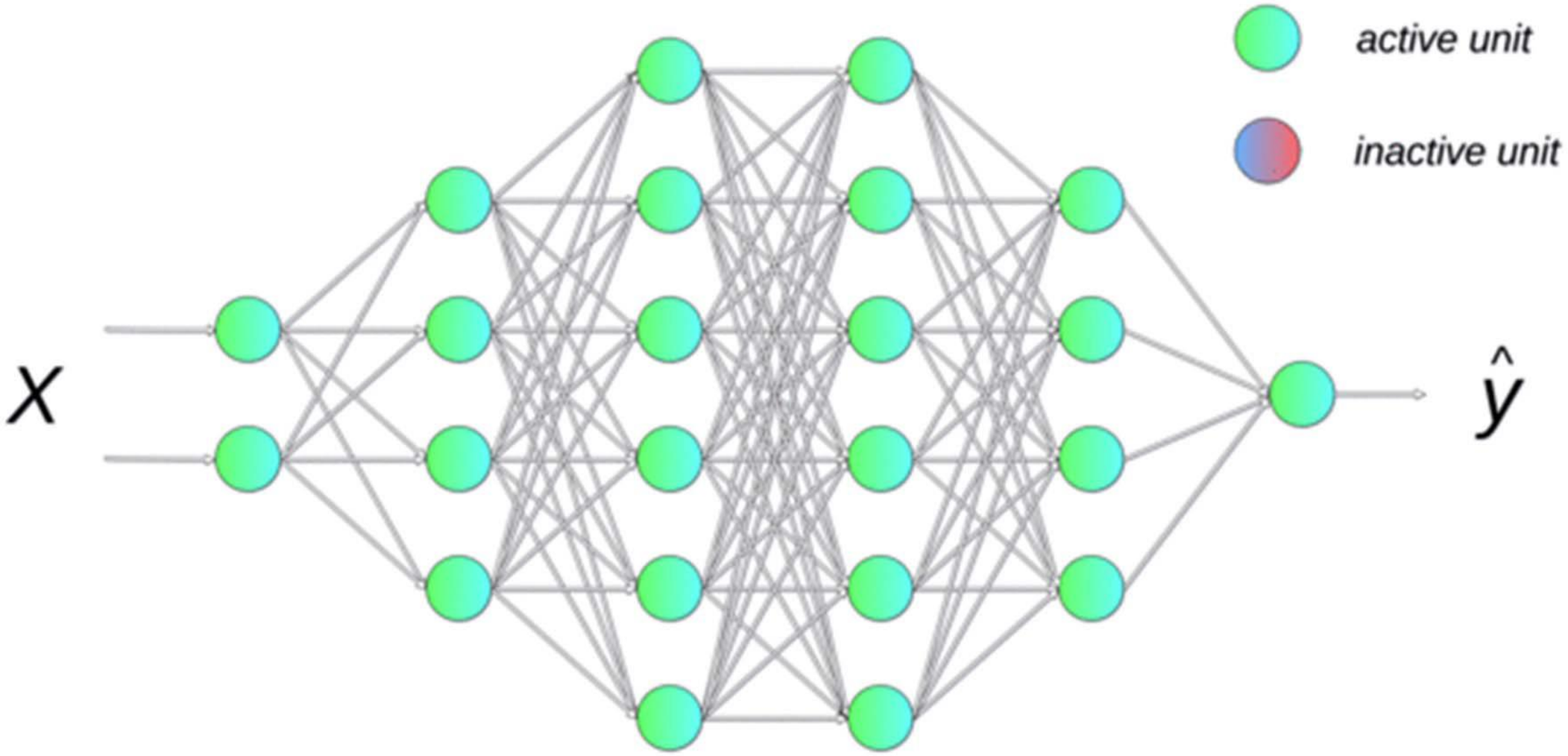


Fix step size

4. Regularization– Dropout

- Set a limit for weight magnitude, $|w|<\tau$ (Max norm constraint)
- Each time we present a training data, we randomly **omit** each hidden node or input nodes (not eliminate) with probability p=0.5 – **don't update their weights**.
- When next data come in, randomly omit some hidden nodes or input nodes again and repeat this for the rest of data.

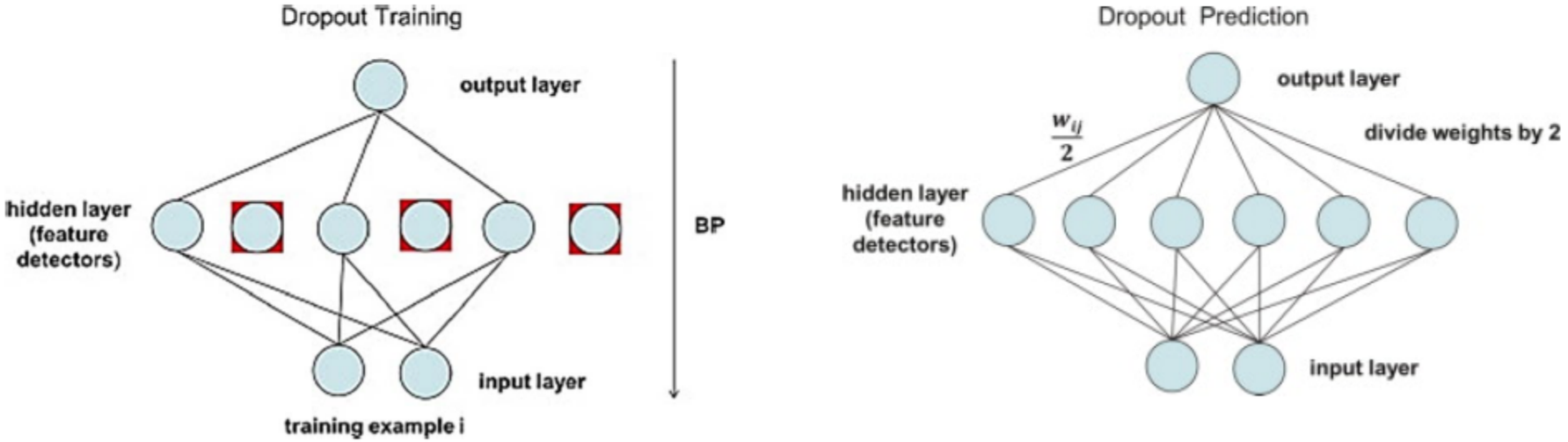
4. Regularization– Dropout



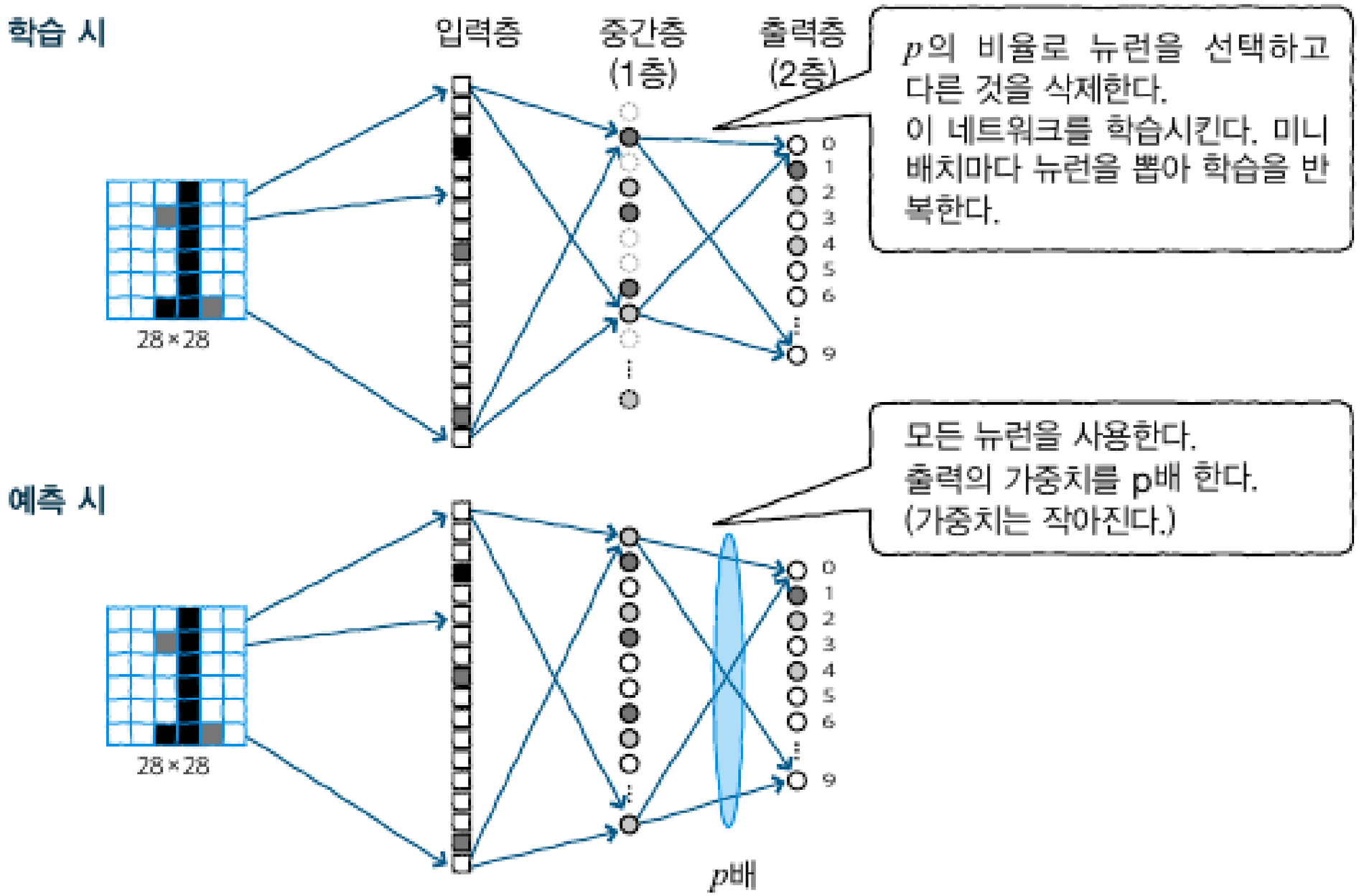
$p^{[0]} = 0.0$ $p^{[1]} = 0.0$ $p^{[2]} = 0.5$ $p^{[3]} = 0.0$ $p^{[4]} = 0.25$

4. Regularization– Dropout

- During prediction, there is no dropout applied.
- Use all nodes (no more omitted nodes), scaled the weights with *p*.
- In practice, the value of dropout ratio *p*=0.5 is a reasonable default, but this can be tuned on validation data.



4. Regularization– Dropout



5. Weight Initialization

- The magnitude of initial weights can affect the network convergence very significantly.

- Solution
  - Xavier Initialization, He Initialization
  - Layerwise Pre-training

5. Weight Initialization

- Not all 0's
- Challenging issue
- Hinton et al. (2006) "A Fast Learning Algorithm for Deep Belief Nets"
  - Restricted Boatman Machine (RBM)

5. Weight Initialization

- No need to use complicated RBM for weight initializations
- Simple methods are OK
  - **Xavier initialization**: X. Glorot and Y. Bengio, "Understanding the difficulty of training deep feedforward neural networks," in International conference on artificial intelligence and statistics, 2010
  - **He's initialization**: K. He, X. Zhang, S. Ren, and J. Sun, "Delving Deep into Rectifiers: Surpassing Human-Level Performance on ImageNet Classification," 2015

5. Weight Initialization

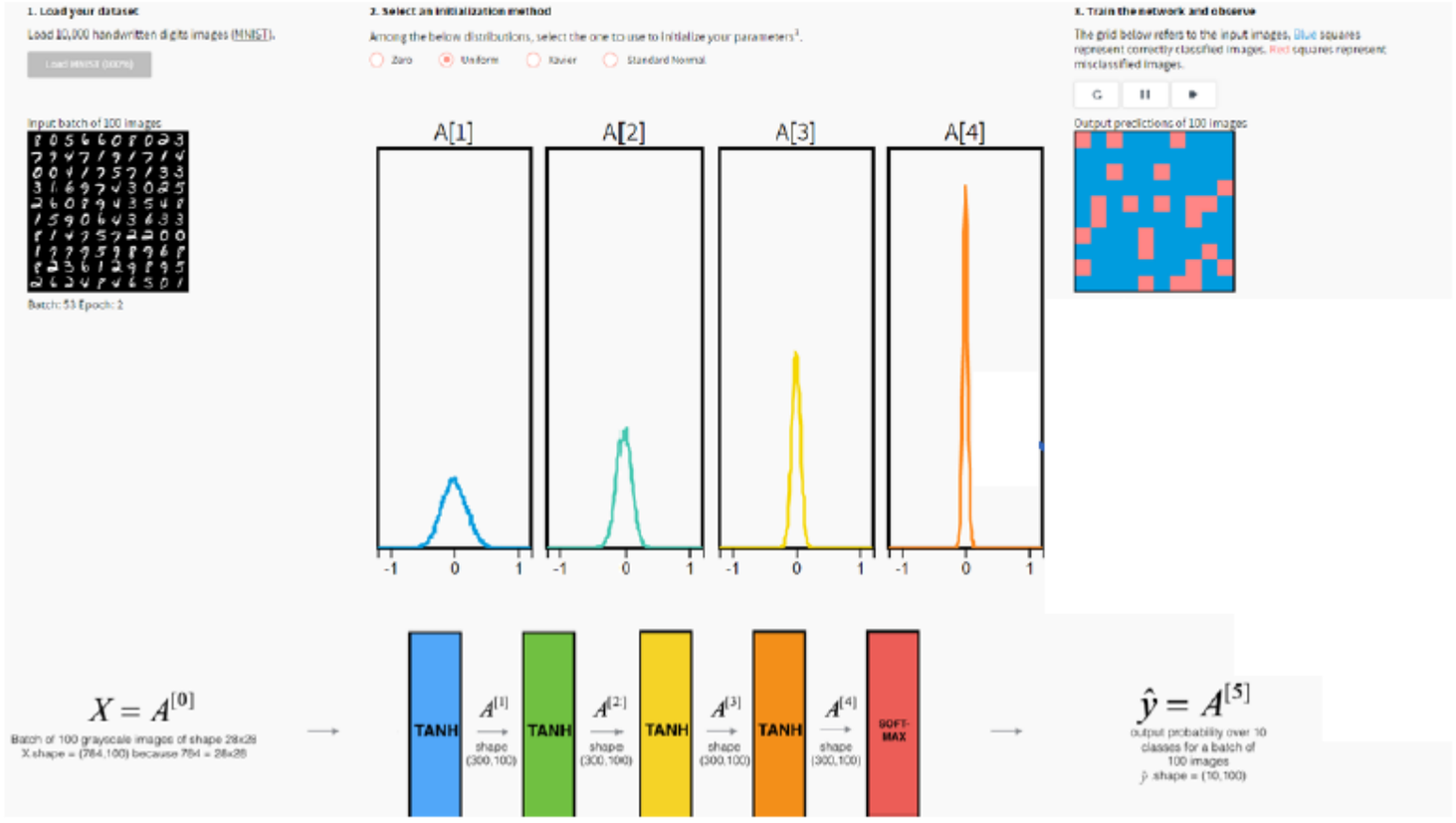
- Makes sure the weights are 'just right', not too small, not too big
- Using number of input (fan_in) and output (fan_out)

```
# Xavier initialization
# Glorot et al. 2010
W = np.random.randn(fan_in, fan_out)/np.sqrt(fan_in)

# He et al. 2015
W = np.random.randn(fan_in, fan_out)/np.sqrt(fan_in/2)
```

5. Weight Initialization

- Xavier Initialization
  - It suggests that variance of the Gaussian distribution of weights for each neuron should depend on the number of inputs to the layer, ie. wi ~ G(0, $1/N_{in}$), Nin = # of input neurons to the current neuron.

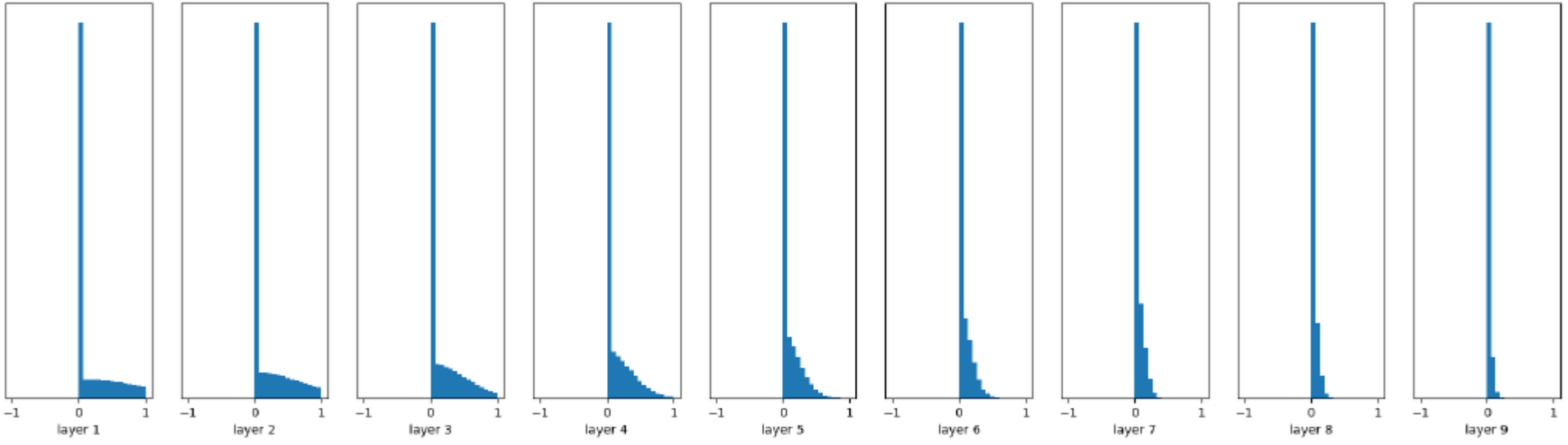


https://www.deeplearning.ai/ai-notes/initialization/#III

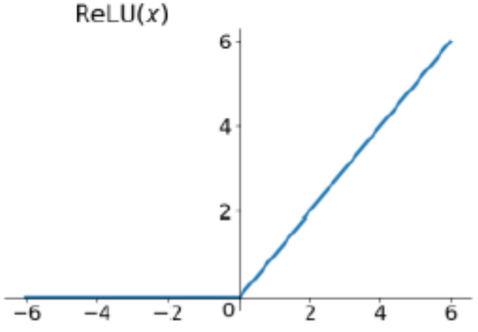
5. Weight Initialization

- Xavier Initialization

Neuron output values at each hidden layer

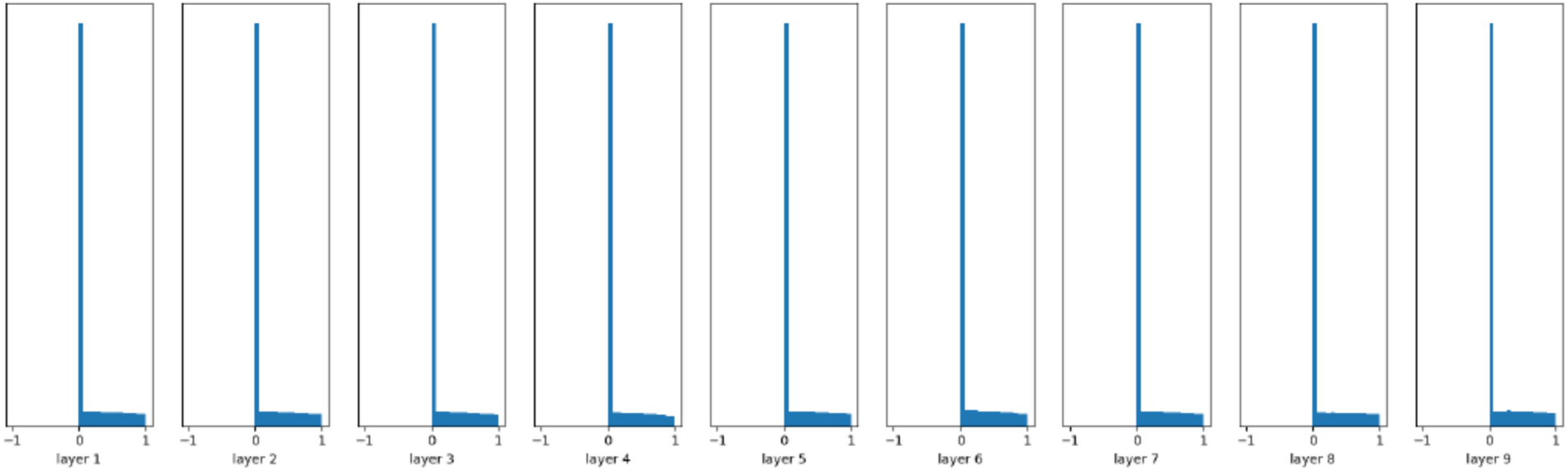


Activation function: ReLU()



5. Weight Initialization

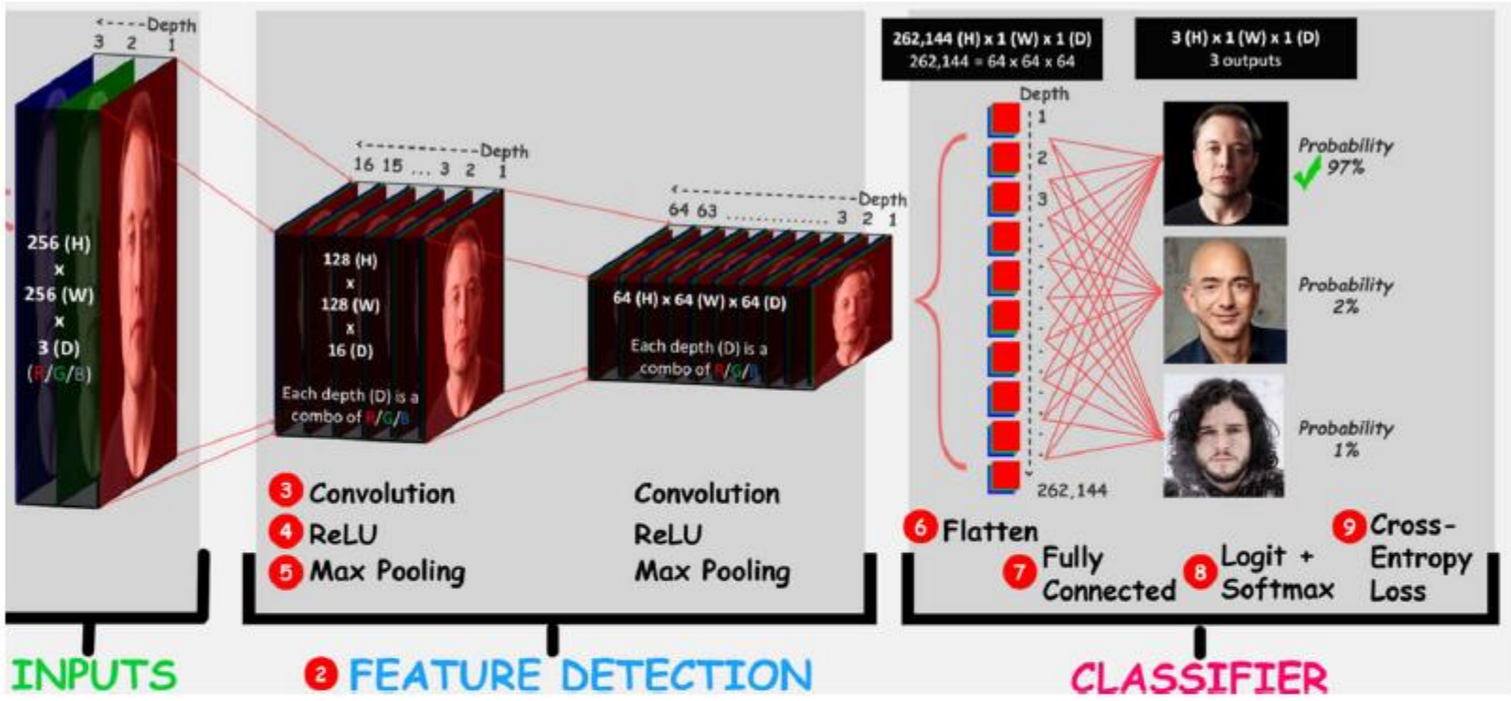
- He Initialization
  - It suggests that variance of the Gaussian distribution of weights for each neuron should depend on the number of inputs to the layer, ie. $w_i \sim G(0, 2/N_{in})$.



Activation function: ReLU()

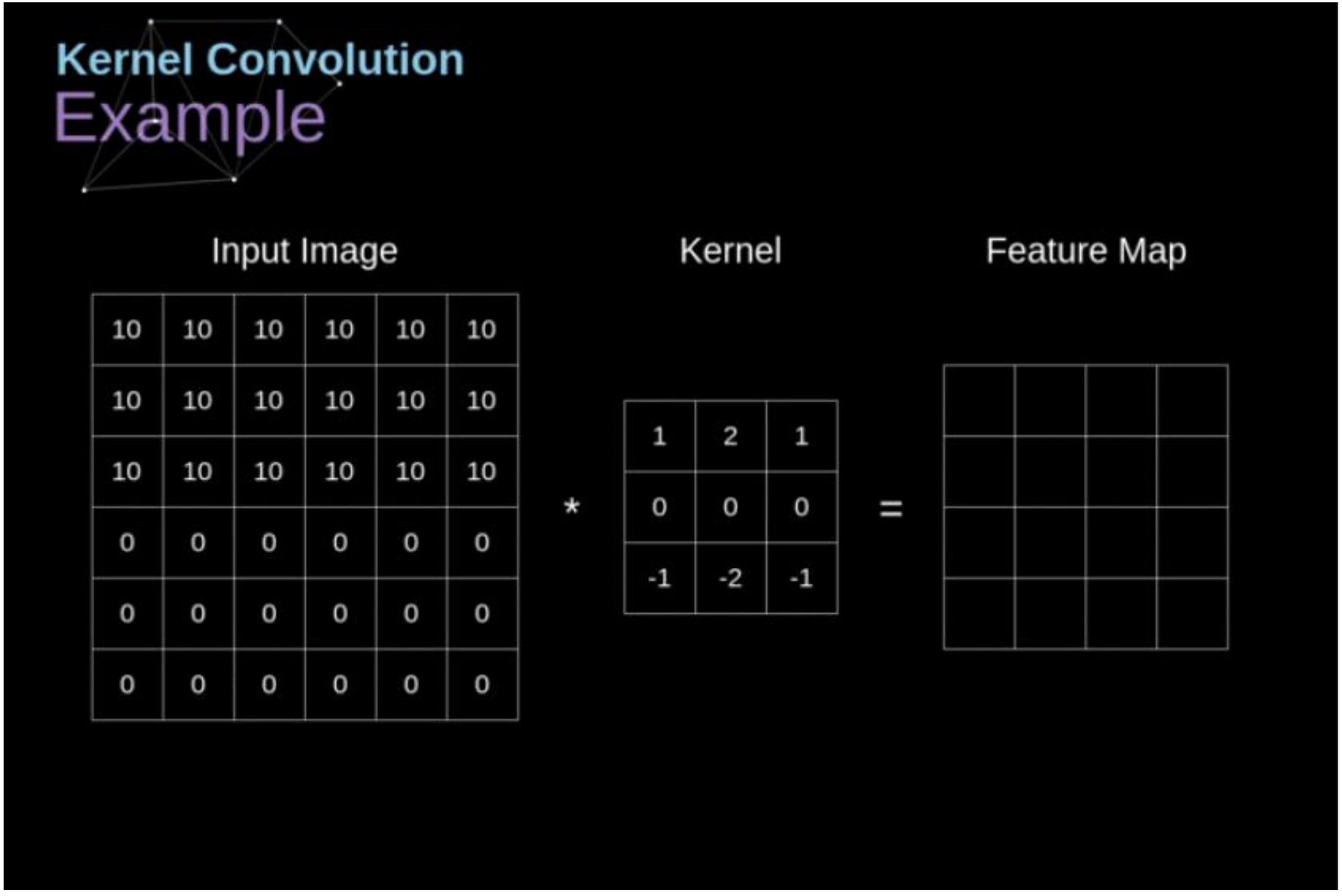
Convolution Neural Network

Overview of ConvNets



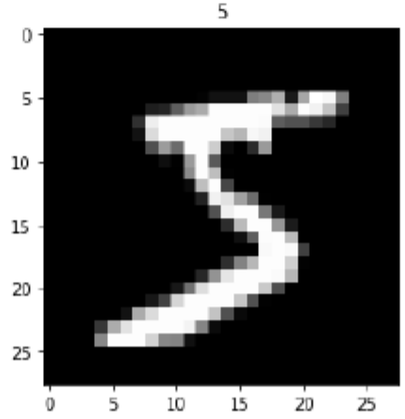
- A special case of MLP
- Commonly apply to visual data (image, video, 3D object etc) or 2D array input.
- Unified feature extractor and classifier in one network.
- Filter kernel = weights in between layers (to be learned from BP)
- Feature map = filtered outputs (corresponds to neurons in MLP)

Convolution



Convolution Neural Network (CNN)

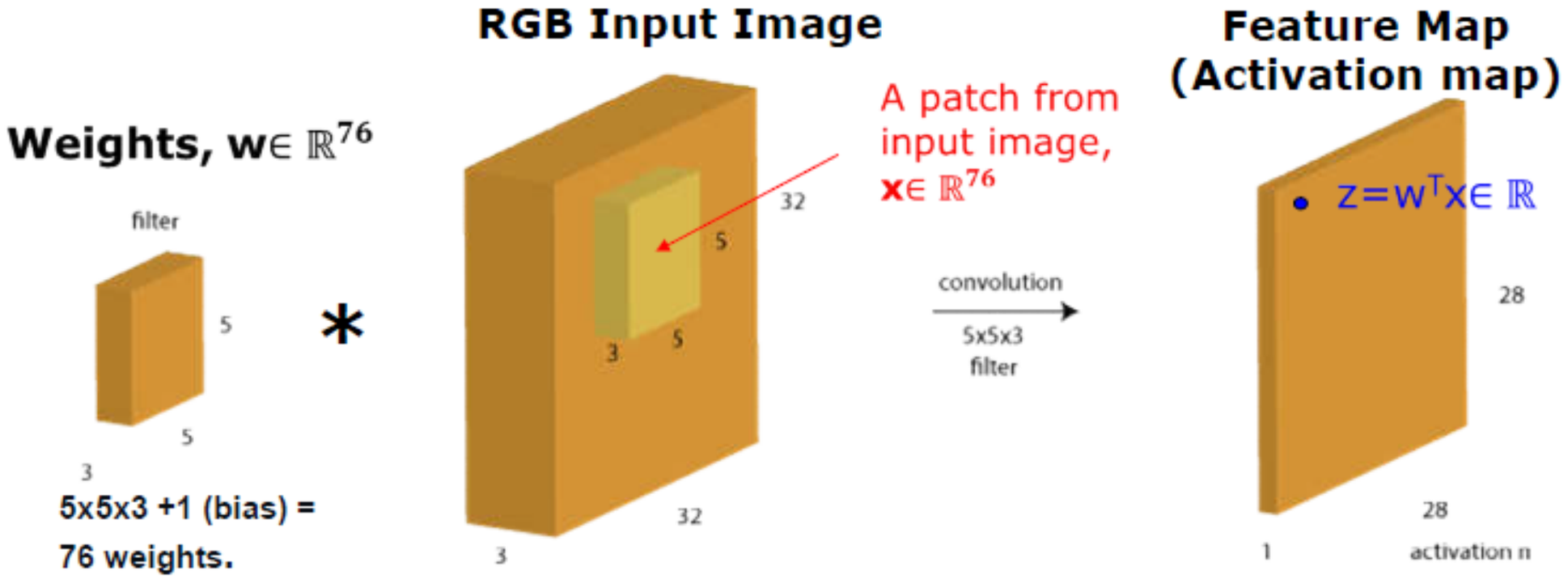
- 이미지 데이터의 특성
  - (일반적으로) 이미지 하나를 구성하는 특성값feature value이 많음
    - 최소한 픽셀 개수만큼의 특성값이 존재
  - 픽셀 하나는 **인접한 다른 픽셀들과의 상관관계correlation**가 존재함
  - 약간의 늘림, 회전, 확대/축소 등이 이미지의 **핵심 정보를 바꾸지 않음**
  - MNIST의 예시
    - 28×28 = 784 개의 흑백 픽셀로 구성된 0부터 9까지의 숫자 이미지
    - 뉴런 구조가 (784+1)-100-10인 다층 퍼셉트론으로 분류 문제를 학습하는 경우 가중치 79500개가 필요함



Convolution Neural Network (CNN)

- 이미지 데이터의 특성
  - 컬러 이미지(RGB)의 예시
    - 가로 500 픽셀 × 세로 500 픽셀 = 총 250000 픽셀
    - 픽셀 하나당 R, G, B 특성값이 존재: 이미지 하나당 특성값 750000개
    - 이미지 속의 동물을 강아지, 고양이, 사람 중 하나로 분류하는 데이터를 학습하는 경우 뉴런 구조가 750000 – 100 – 3 인 다층 퍼셉트론을 쓴다고 하더라도 약 **8천만개에 달하는 가중치**를 학습해야 함
    - **요즘 스마트폰으로 찍는 사진을 쓴다면??**

Convolution Neural Network (CNN)



One convolution yield one scalar output, filter scanning is carried out yield 28 x 28 feature map

Convolution Neural Network (CNN)

- CNN의 작동 원리
  - Local receptive field

0	0	0	1	1	1	0	0	0
0	0	1	1	0	1	1	0	0
0	0	1	0	0	0	1	0	0
0	0	0	0	0	0.8	1	0	0
0	0	0	0	1	1	0	0	0
0	0	0	0	1	0	0	0	0
0	0	0	0	1	0	0	0	0
0	0	0	0	0	0	0	0	0
0	0	0	0	1	0	0	0	0

학습 대상 이미지
(9 * 9 * 1)

1	0	0
0	1	0
0	0	1

b_1

0	0	1
0	1	0
1	0	0

b_2

0	0	0
1	1	1
0	0	0

b_3

0	1	0
0	1	0
0	1	0

b_4

3*3 필터 4개
필터당 대응되는 bias 4개

Convolution Neural Network (CNN)

- CNN의 작동 원리
  - Local receptive field – convolution

1	0	0	1	1	1	0	0	0
0	1	0	1	0	1	1	0	0
0	0	1	0	0	0	1	0	0
0	0	0	0	0	0.8	1	0	0
0	0	0	0	1	1	0	0	0
0	0	0	0	1	0	0	0	0
0	0	0	0	1	0	0	0	0
0	0	0	0	0	0	0	0	0
0	0	0	0	1	0	0	0	0

1	0	0
0	1	0
0	0	1

b_1

$1 + b_1$						

필터의 각 가중치와 이미지의 픽셀값을 곱하고 bias를 더하여 feature map에 저장

Convolution Neural Network (CNN)

- CNN의 작동 원리
  - Local receptive field – convolution

0	1	0	0	1	1	0	0	0
0	0	1	0	0	1	1	0	0
0	0	0	1	0	0	1	0	0
0	0	0	0	0	0.8	1	0	0
0	0	0	0	1	1	0	0	0
0	0	0	0	1	0	0	0	0
0	0	0	0	1	0	0	0	0
0	0	0	0	0	0	0	0	0
0	0	0	0	1	0	0	0	0

1	0	0
0	1	0
0	0	1

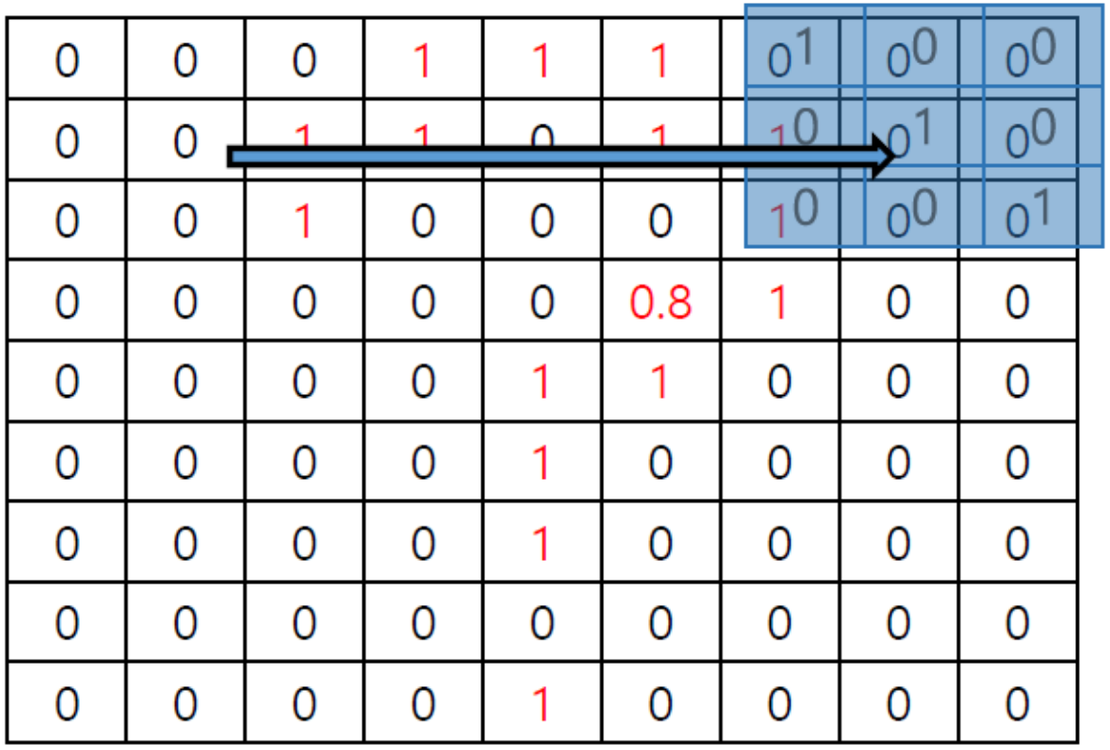
b_1

$1+b_1$	$1+b_1$					

필터의 각 가중치와 이미지의 픽셀값을 곱하고 bias를 더하여 feature map에 저장

Convolution Neural Network (CNN)

- CNN의 작동 원리
  - Local receptive field – convolution



$1+b_1$	$1+b_1$	$1+b_1$	$1+b_1$	$3+b_1$	$2+b_1$	$0+b_1$

필터의 각 가중치와 이미지의 픽셀값을 곱하고 bias를 더하여 feature map에 저장

Convolution Neural Network (CNN)

- CNN의 작동 원리
  - Local receptive field – convolution

0	0	0	1	1	1	0	0	0
0	0	1	1	0	1	1	0	0
0	0	1	0	0	0	1	0	0
0	0	0	0	0	0.8	1	0	0
0	0	0	0	1	1	0	0	0
0	0	0	0	1	0	0	0	0
0	0	0	0	1	0	01	00	00
0	0	0	0	0	0	00	01	00
0	0	0	0	1	0	00	00	01

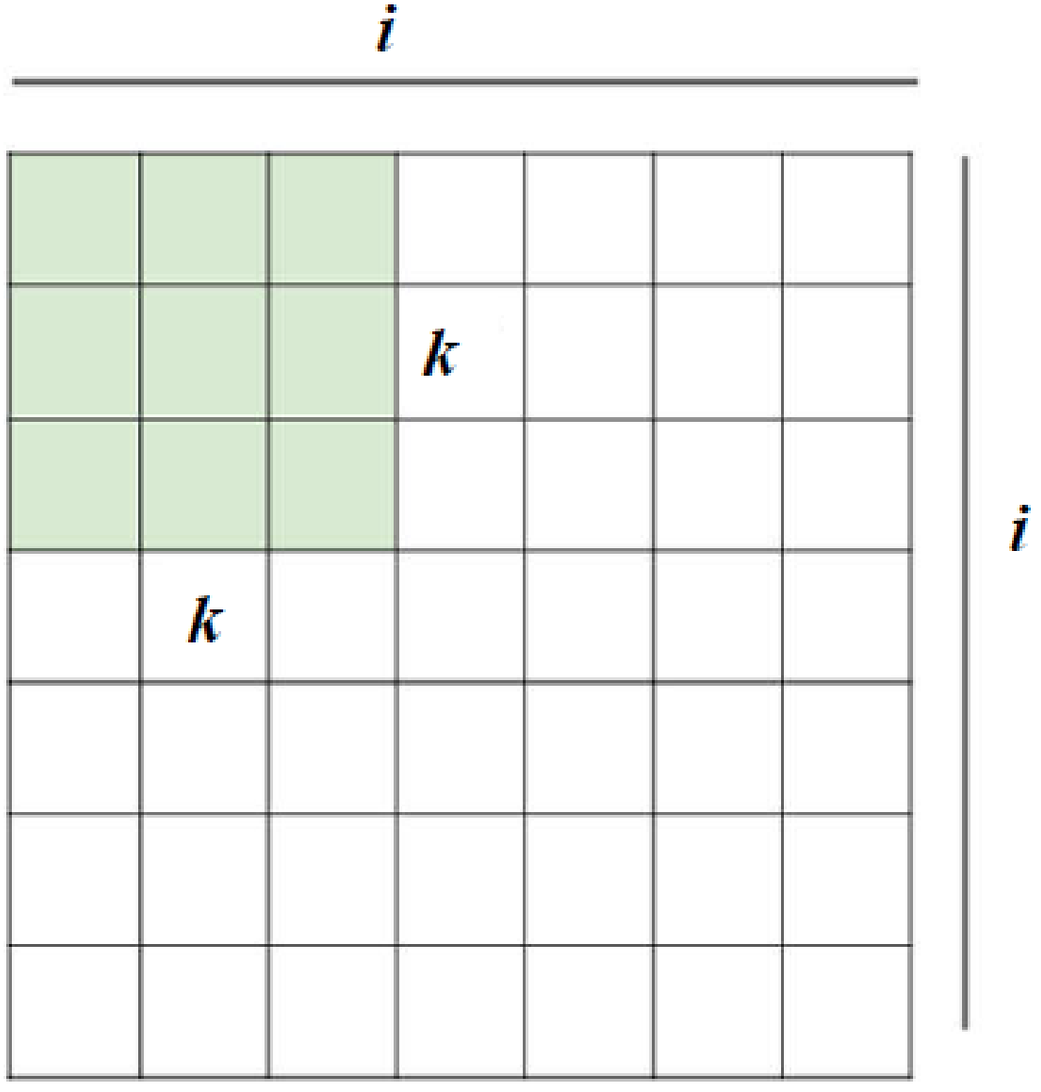
1	0	0
0	1	0
0	0	1

b_1

$1+b_1$	$1+b_1$	$1+b_1$	$1+b_1$	$3+b_1$	$2+b_1$	$0+b_1$
$0+b_1$	$1+b_1$	$1+b_1$	$1.8+b_1$	$1+b_1$	$2+b_1$	$1+b_1$
$0+b_1$	$0+b_1$	$2+b_1$	$1+b_1$	$0.8+b_1$	$1+b_1$	$1+b_1$
$0+b_1$	$0+b_1$	$1+b_1$	$1+b_1$	$1+b_1$	$0.8+b_1$	$1+b_1$
$0+b_1$	$0+b_1$	$1+b_1$	$1+b_1$	$1+b_1$	$1+b_1$	$0+b_1$
$0+b_1$	$0+b_1$	$0+b_1$	$1+b_1$	$1+b_1$	$0+b_1$	$0+b_1$
$0+b_1$	$0+b_1$	$1+b_1$	$0+b_1$	$1+b_1$	$0+b_1$	$0+b_1$

필터의 각 가중치와 이미지의 픽셀값을 곱하고 bias를 더하여 feature map에 저장

Convolution Neural Network (CNN)



$$o = \left\lfloor \frac{i - k}{s} \right\rfloor + 1.$$

Example:
For filter size k =3,
input size i=7,
stride s=1

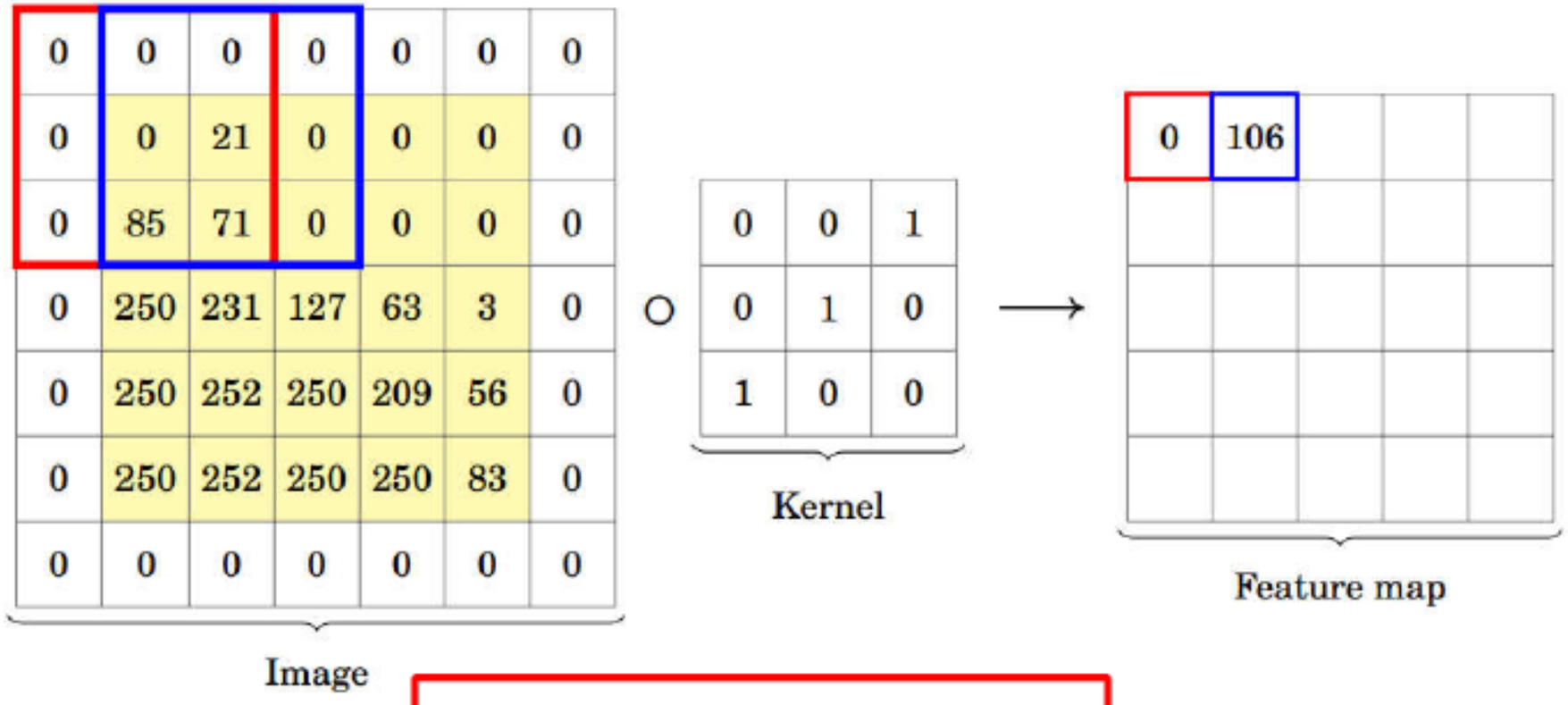
Output feature map size,
o = (7-3)/1 + 1 = 5

Convolution Neural Network (CNN)

- Why would you opt stride > 1?
  - PROS: Makes your model faster by having less calculations and fewer calculations to store in memory.
  - CONS: You lose information about the picture because you would skip pixels and potentially miss out on seeing a pattern.

- How to maintain the feature map output size as input size?
  - Zero Padding

Convolution Neural Network (CNN)

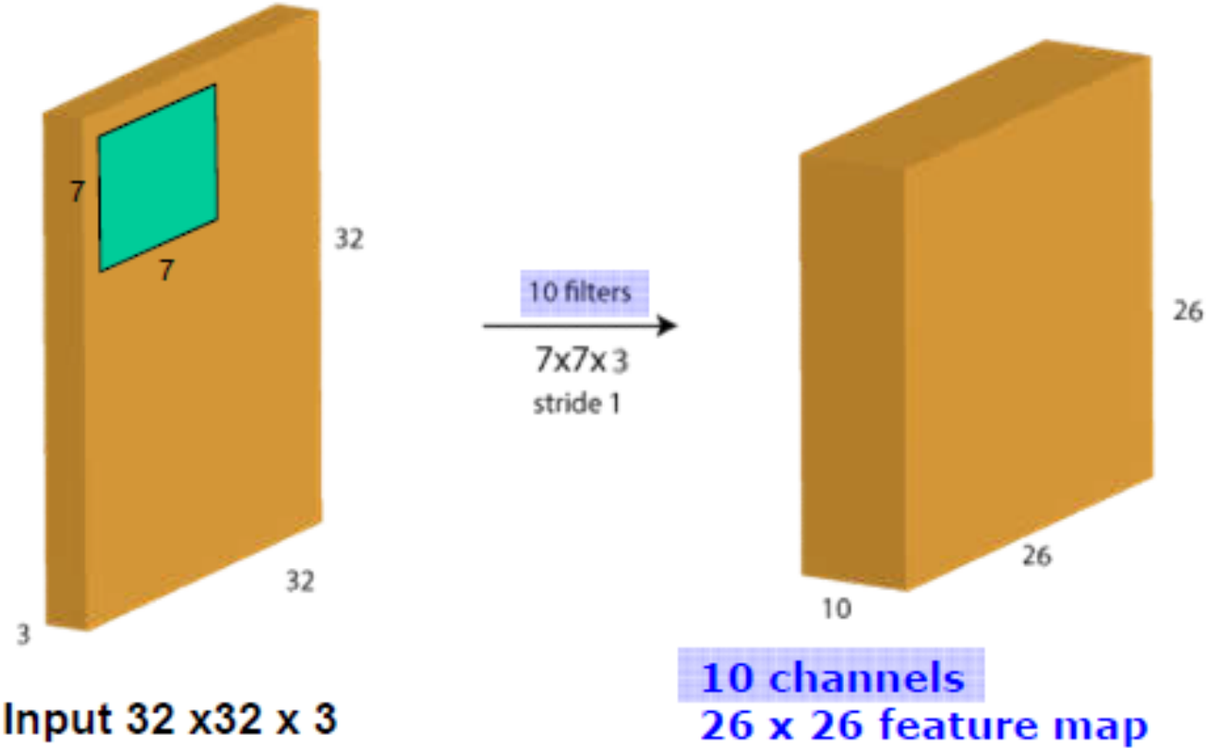


$$o = \left\lfloor \frac{i + 2p - k}{s} \right\rfloor + 1.$$

Zero padding (p=0) is not a MUST for ConvNets but recommended

Convolution Neural Network (CNN)

Apply 7x7x3 filters yield 26x26 output, repeat 10 times yield 26x26x10 feature maps



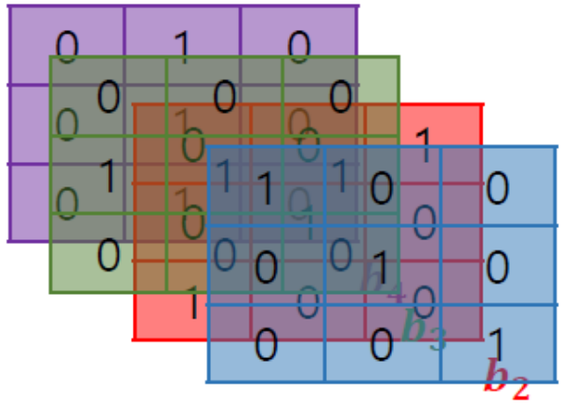
Input 32 x32 x 3

10 channels
26 x 26 feature map

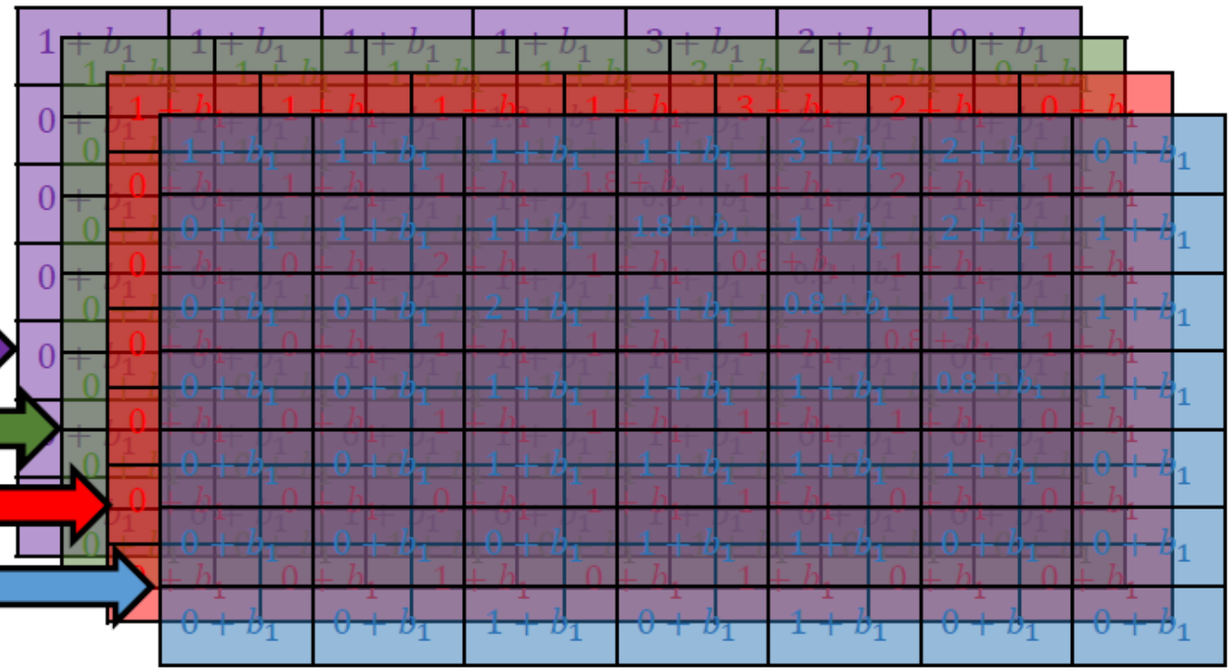
* Valid convolution is assumed in this example

Convolution Neural Network (CNN)

- CNN의 작동 원리
  - Local receptive field – convolution



0	0	0	1	1	1	0	0	0
0	0	1	1	0	1	1	0	0
0	0	1	0	0	0	1	0	0
0	0	0	0	0	0.8	1	0	0
0	0	0	0	1	1	0	0	0
0	0	0	0	1	0	0	0	0
0	0	0	0	1	0	0	0	0
0	0	0	0	0	0	0	0	0
0	0	0	0	1	0	0	0	0

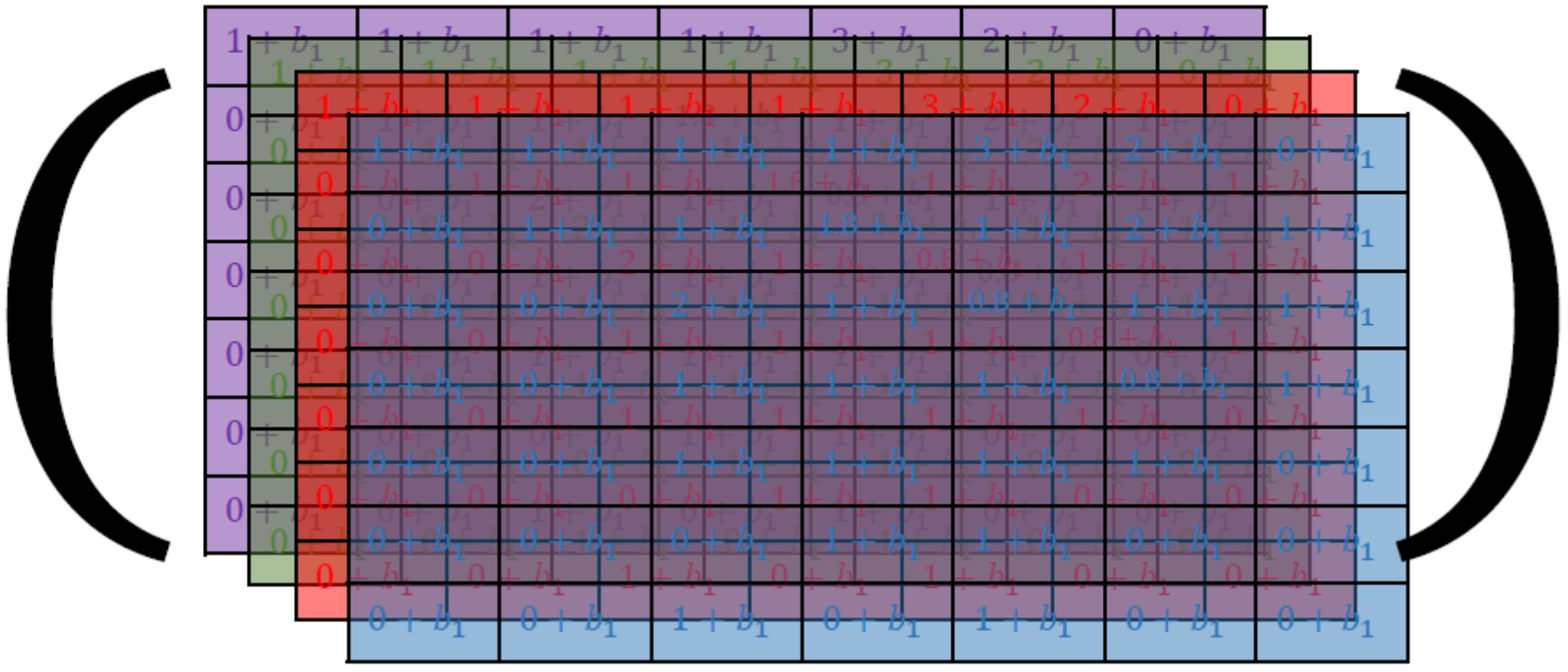


이미지에 주어진 필터들을 적용하여 feature map들을 생성 (7 * 7 * 4)

Convolution Neural Network (CNN)

- CNN의 작동 원리
  - Local receptive field – 활성함수 적용

$ReLU$

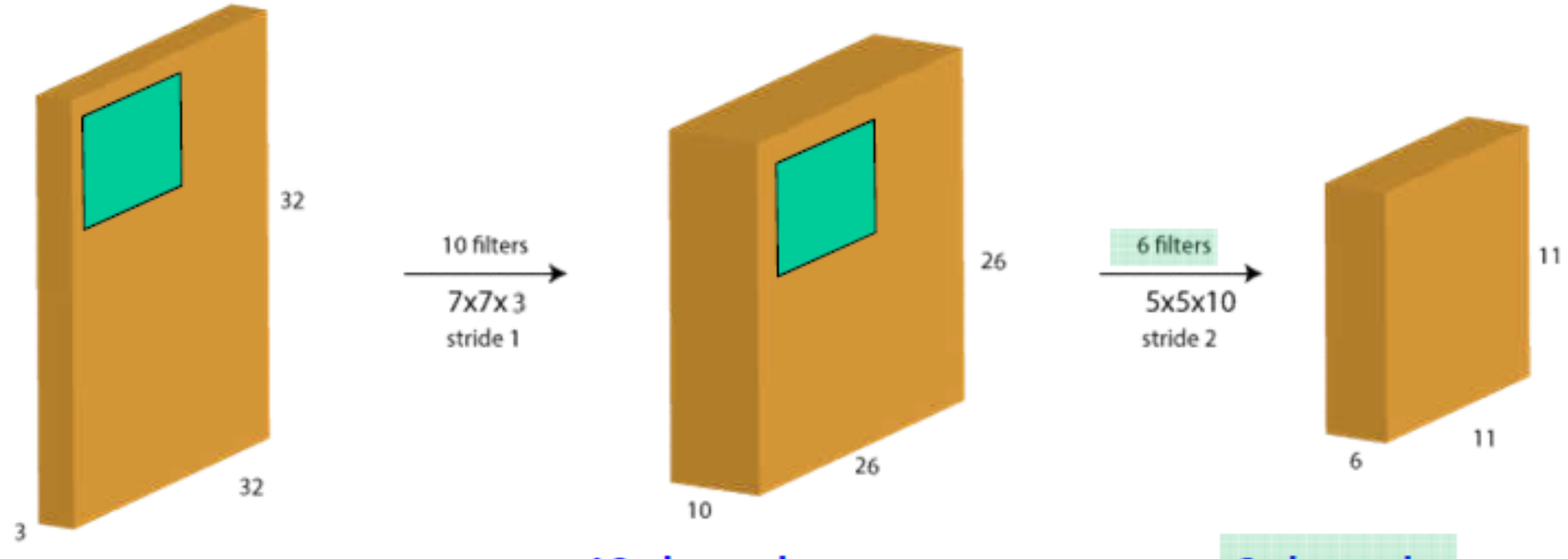


생성된 feature map의 값들에 활성함수를 적용

Convolution Neural Network (CNN)

Apply 7x7x3 filters yield 26x26 output, repeat 10 times yield 26x26x10 feature maps

Apply 6x6x10 filters yield 11x11 output, repeat 6 times yield 11x11x6 feature maps



Input 32 x32 x 3

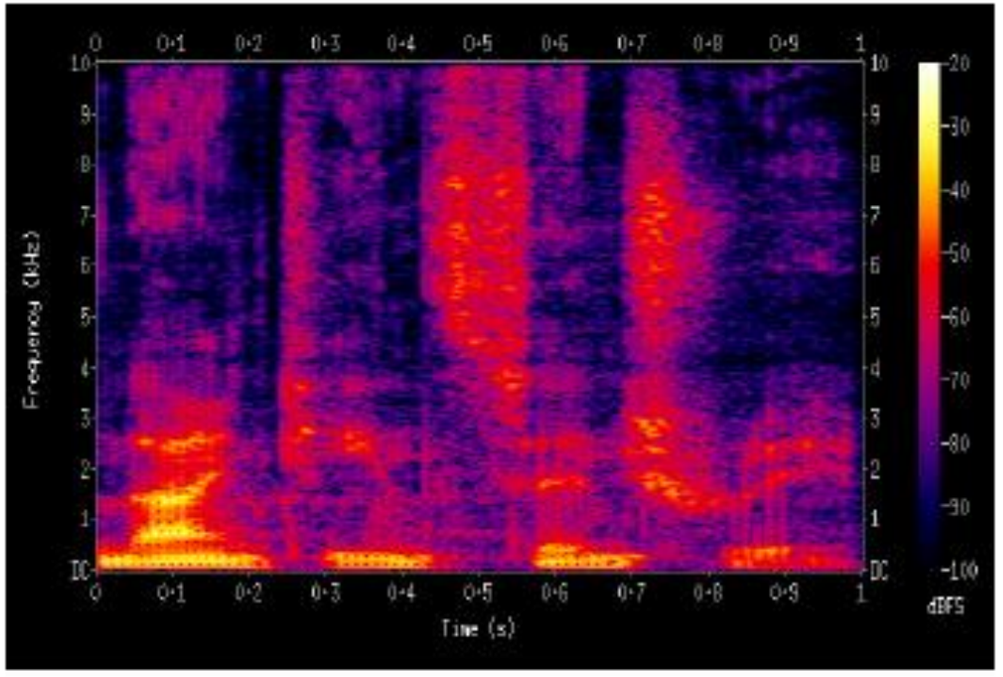
10 channels
26 x 26 feature map

6 channels
11x11 feature map

* Valid convolution is assumed in this example

Convolution Neural Network (CNN)

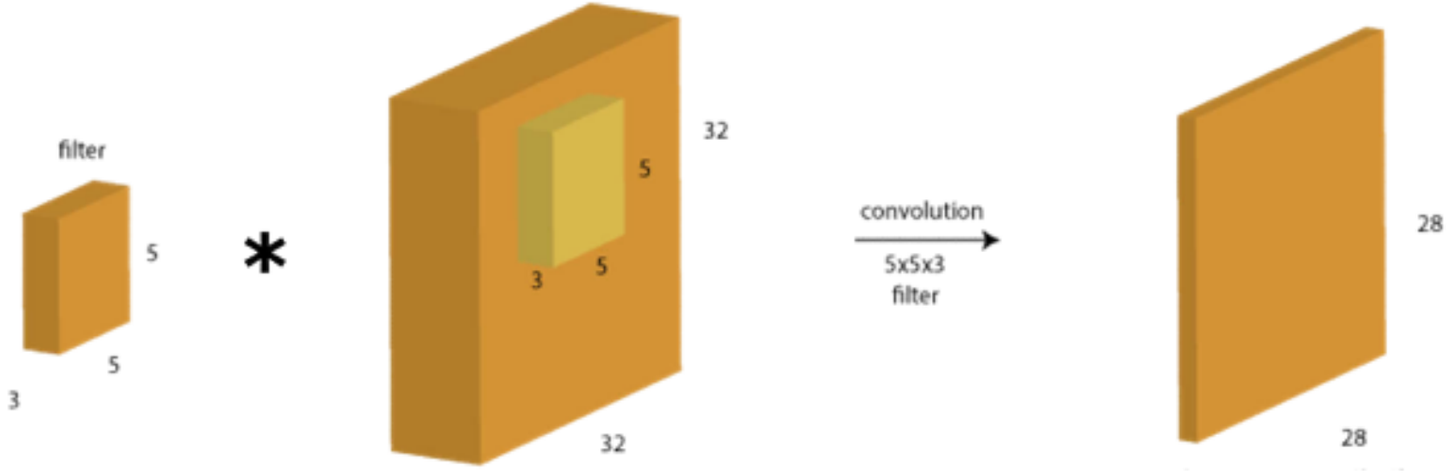
- **Local Connectivity** – image is locally correlated.



- **Weight Sharing** – overcome enormous weights problem.

1: Local Connectivity

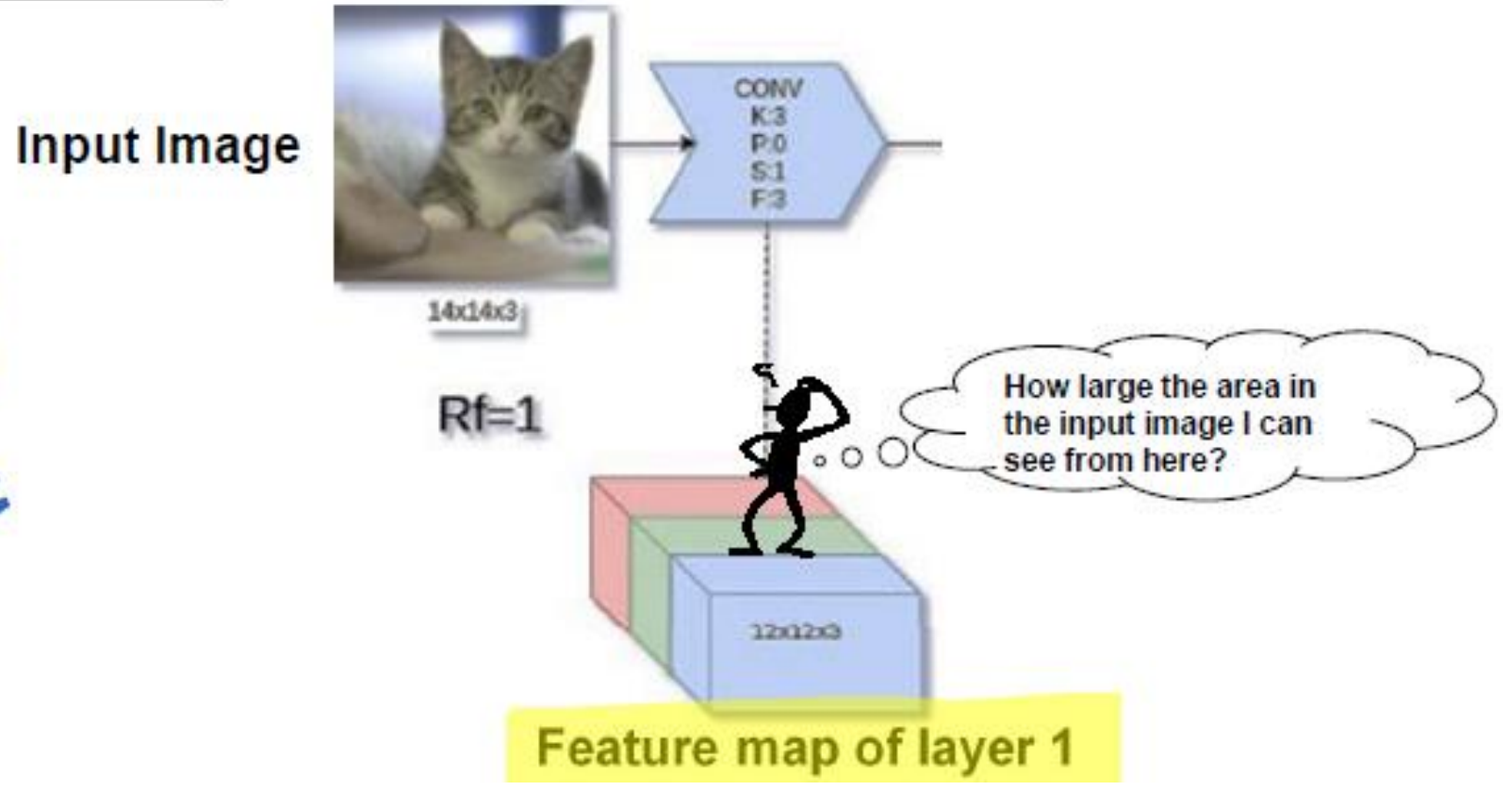
- Let input image size 32 x 32 x 3, filter size 5 x 5 x 3
- **MLP**: Weights are **fully connected**, ie. 32x32x3 weights
- **ConvNets**: One neuron will connect to 5x5x3 chunk and only have 5x5x3 weights. The connection is
  - Local in space (5x5 inside 32 x 32)
  - But full in depth (all 3 depth channels)



1: Receptive Field (RF) of ConvNets

- Local connectivity of ConvNets induces RF notion.

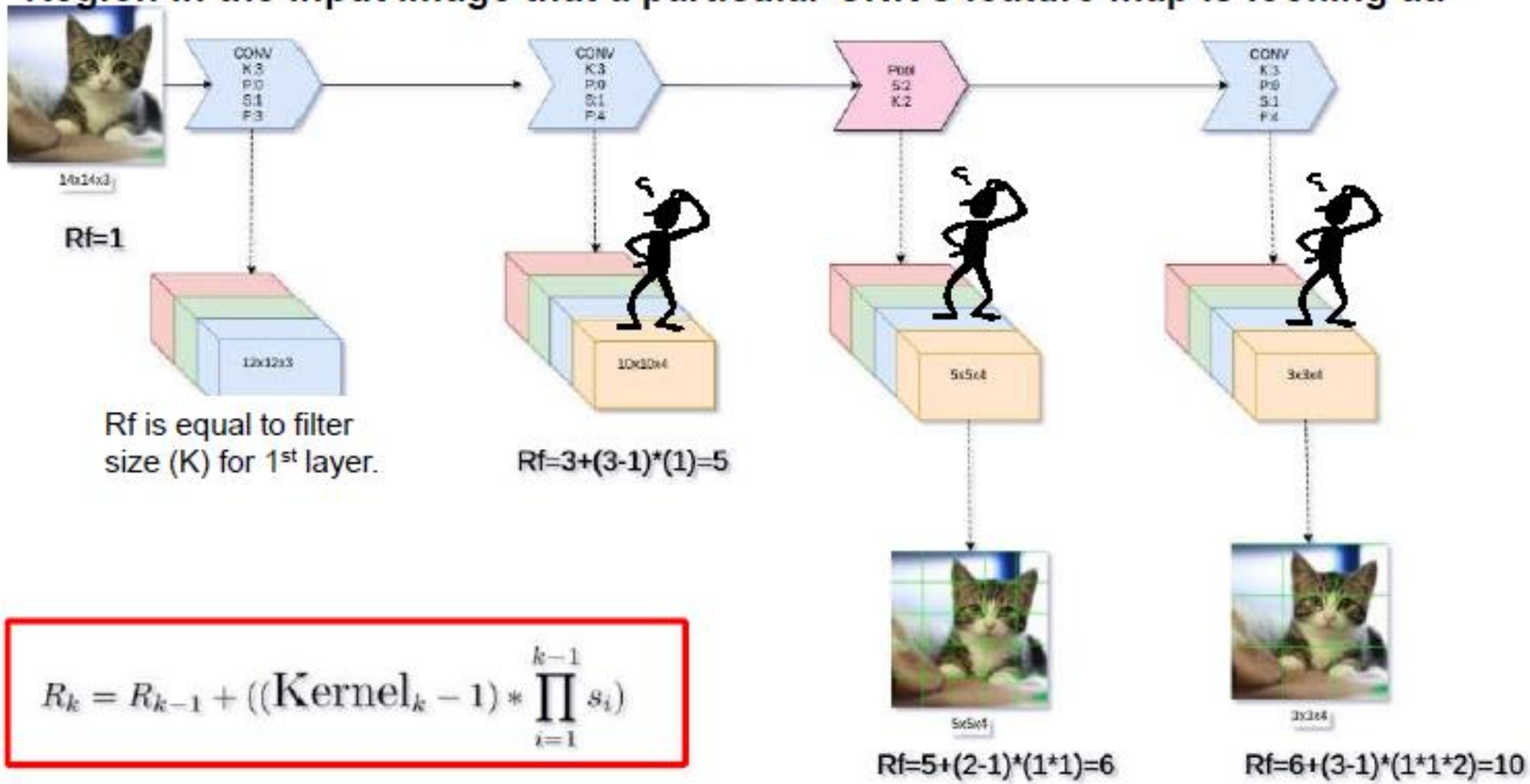
- The RF = how much a convolution window "see" on it's input image or part of the image that is visible to one filter at a time.



filter size = K
padding = P
stride = S
channel = F

1: Receptive Field (RF) of ConvNets

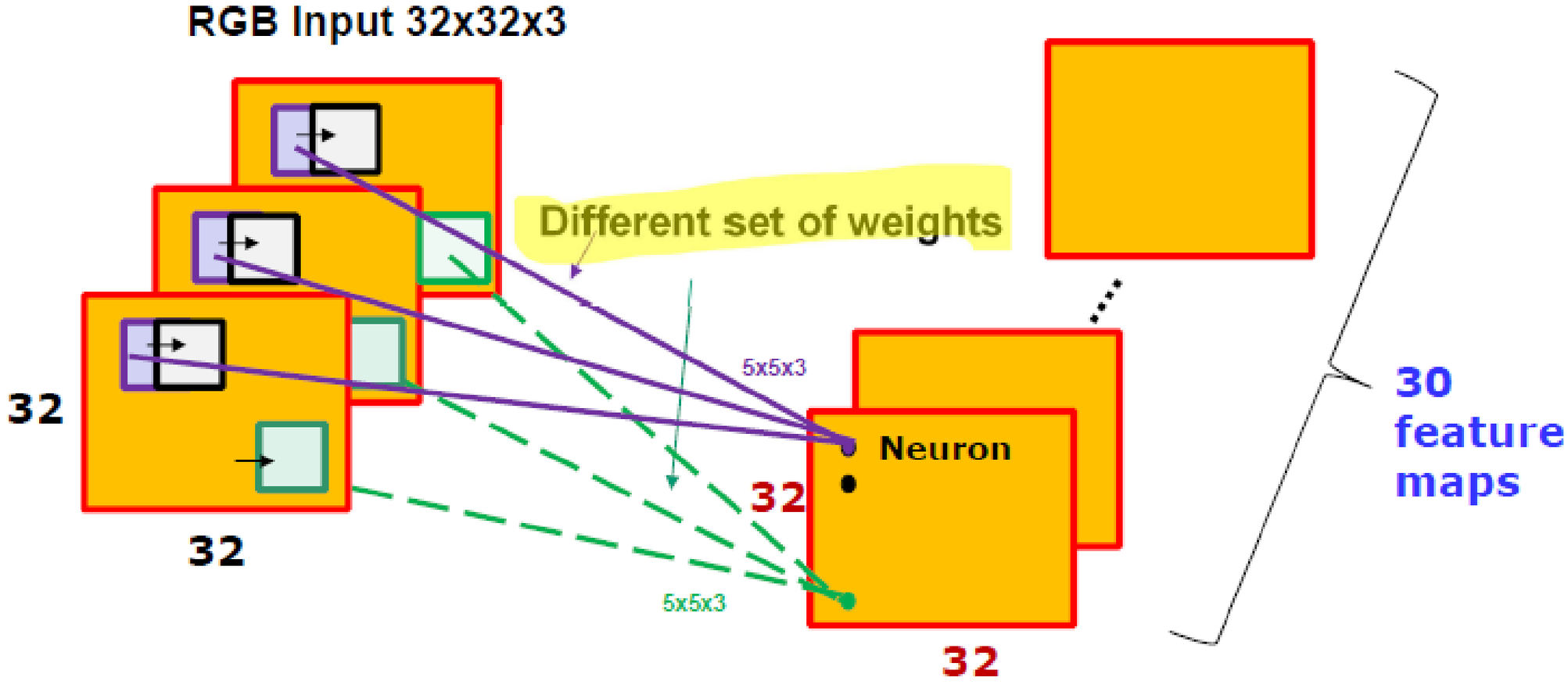
How the later feature map "see" the image?
Region in the input image that a particular CNN's feature map is looking at.



$$R_k = R_{k-1} + ((\text{Kernel}_k - 1) * \prod_{i=1}^{k-1} s_i)$$

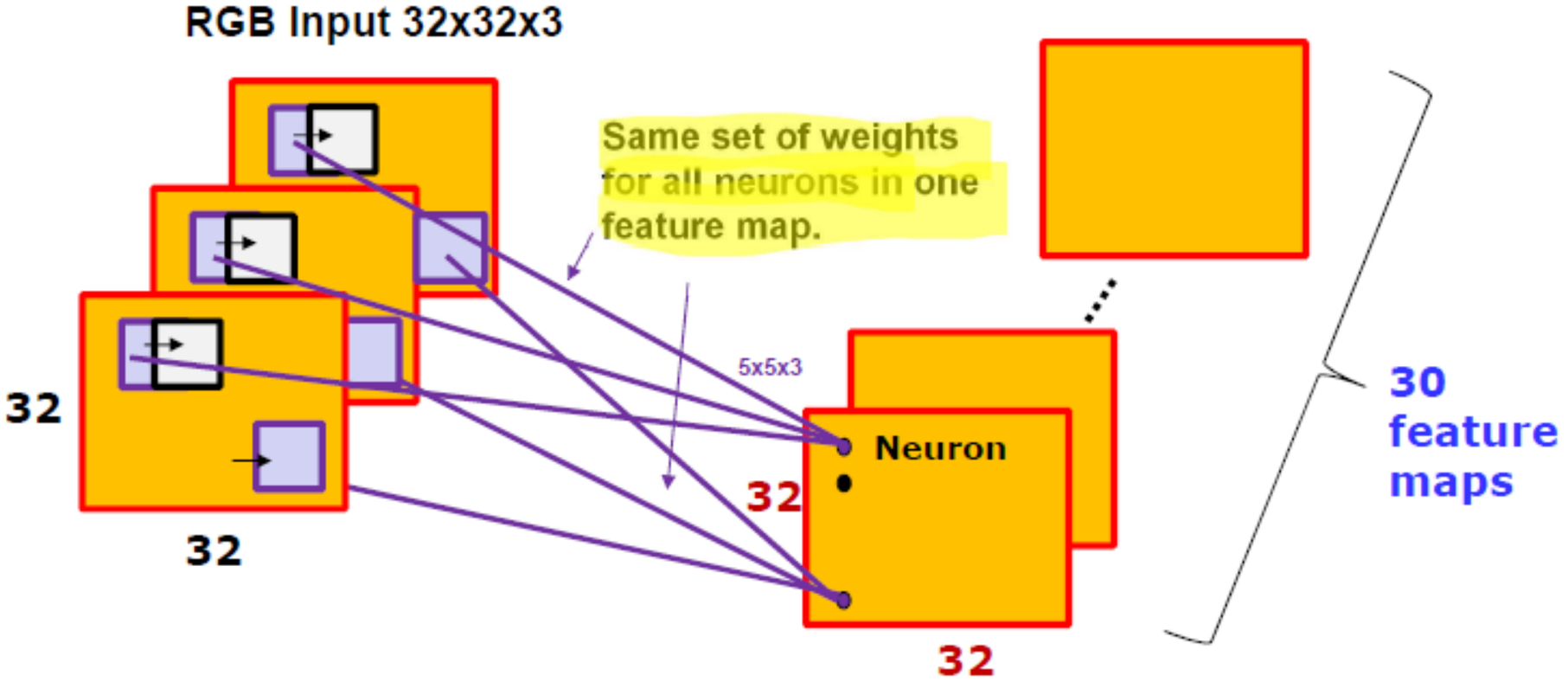
Further Reading: W. Luo, Y. Li, R. Urtasun, and R. Zemel, "Understanding the Effective Receptive Field in Deep Convolutional Neural Networks," Jan. 2017.

2: Weight Sharing



MLP way of setup: different weights for every single neuron
Total # weights to be learned = (5x5x3) x (32 x 32 x 30)
= 75 x 30720 = 2.3 millions!

Convolution Neural Network (CNN)

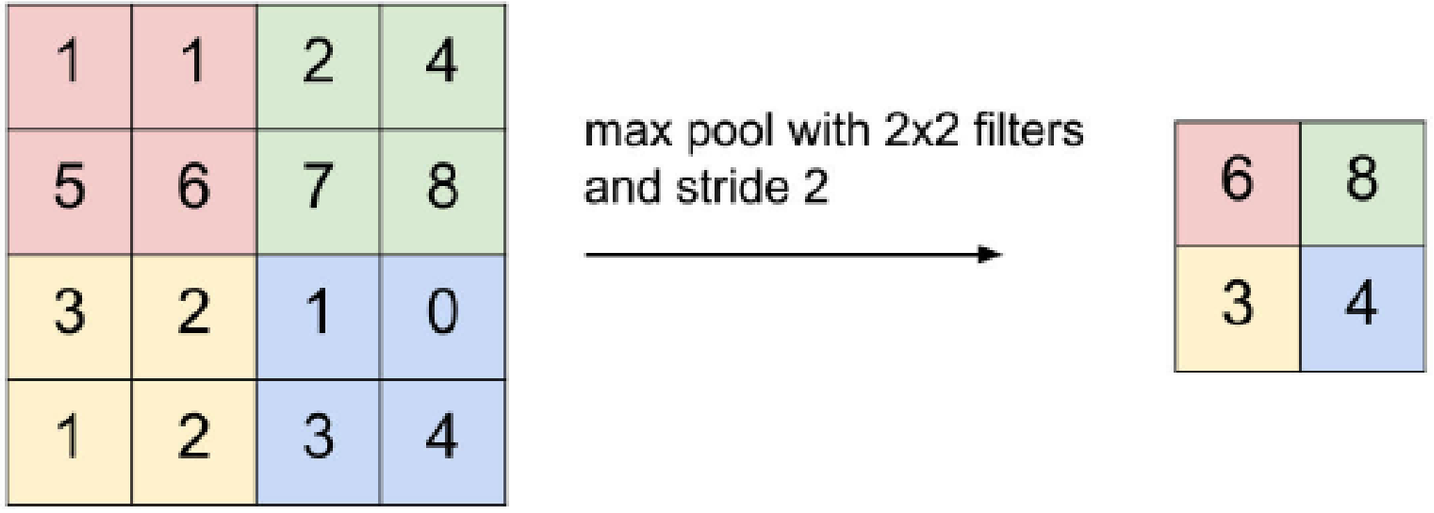


ConvNet weight sharing: same weight for every single feature map

Total # weights to be learned = (5x5x3) x 30

= 75 x 30 = 2250.

Max Pooling



$$o = \left\lfloor \frac{i - k}{s} \right\rfloor + 1.$$

Example:
Input size, i = 4
Pooling window size, k = 2
Stride, s = 2

Output size, o = (4 – 2)/2 + 1 = 2

Convolution Neural Network (CNN)

- CNN의 작동 원리
  - Subsampling(pooling) – 2*2 max pooling의 사례

$1+b_1$	$1+b_1$	$1+b_1$	$1+b_1$	$3+b_1$	$2+b_1$	$0+b_1$
$0+b_1$	$1+b_1$	$1+b_1$	$1.8+b_1$	$1+b_1$	$2+b_1$	$1+b_1$
$0+b_1$	$0+b_1$	$2+b_1$	$1+b_1$	$0.8+b_1$	$1+b_1$	$1+b_1$
$0+b_1$	$0+b_1$	$1+b_1$	$1+b_1$	$1+b_1$	$0.8+b_1$	$1+b_1$
$0+b_1$	$0+b_1$	$1+b_1$	$1+b_1$	$1+b_1$	$1+b_1$	$0+b_1$
$0+b_1$	$0+b_1$	$0+b_1$	$1+b_1$	$1+b_1$	$0+b_1$	$0+b_1$
$0+b_1$	$0+b_1$	$1+b_1$	$0+b_1$	$1+b_1$	$0+b_1$	$0+b_1$

$1+b_1$			

Pooling이 적용되는 범위 안에서 최댓값을 대표값으로 선정

Convolution Neural Network (CNN)

- CNN의 작동 원리
  - Subsampling(pooling) – 2*2 max pooling의 사례

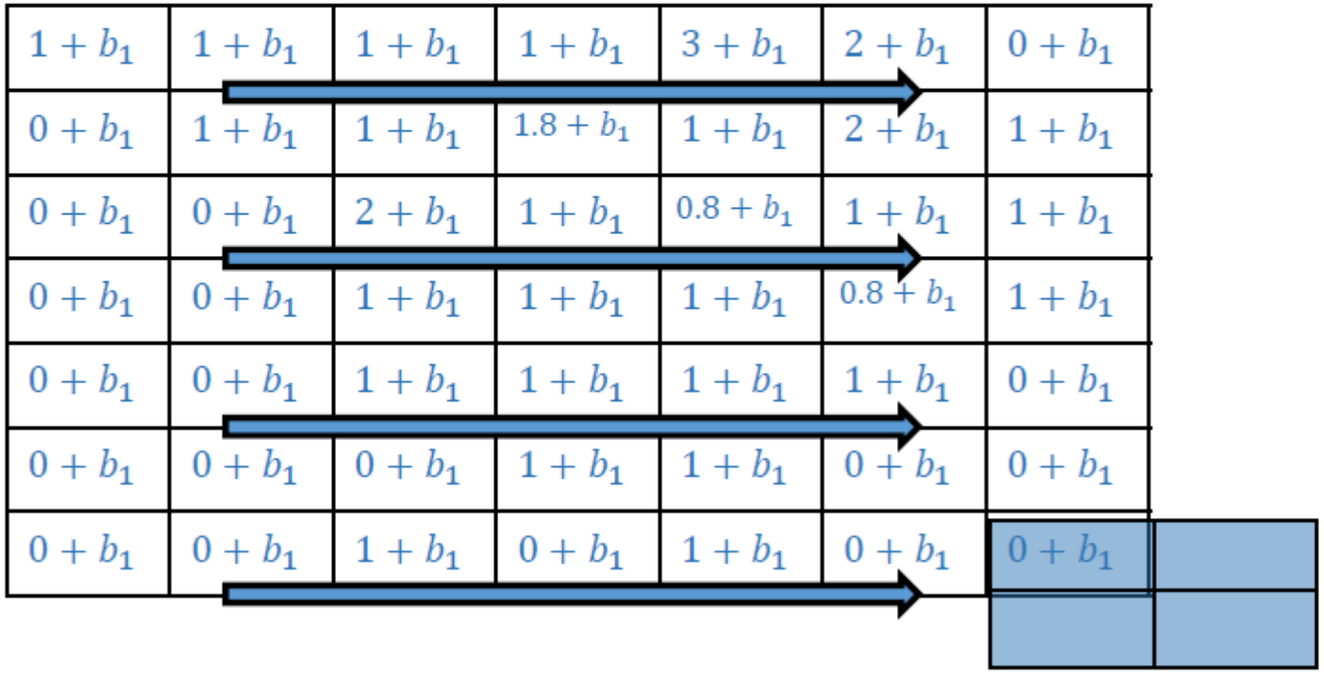
$1+b_1$	$1+b_1$	$1+b_1$	$1+b_1$	$3+b_1$	$2+b_1$	$0+b_1$
$0+b_1$	$1+b_1$	$1+b_1$	$1.8+b_1$	$1+b_1$	$2+b_1$	$1+b_1$
$0+b_1$	$0+b_1$	$2+b_1$	$1+b_1$	$0.8+b_1$	$1+b_1$	$1+b_1$
$0+b_1$	$0+b_1$	$1+b_1$	$1+b_1$	$1+b_1$	$0.8+b_1$	$1+b_1$
$0+b_1$	$0+b_1$	$1+b_1$	$1+b_1$	$1+b_1$	$1+b_1$	$0+b_1$
$0+b_1$	$0+b_1$	$0+b_1$	$1+b_1$	$1+b_1$	$0+b_1$	$0+b_1$
$0+b_1$	$0+b_1$	$1+b_1$	$0+b_1$	$1+b_1$	$0+b_1$	$0+b_1$

$1+b_1$	$1.8+b_1$		

Pooling이 적용되는 범위 안에서 최댓값을 대표값으로 선정

Convolution Neural Network (CNN)

- CNN의 작동 원리
  - Subsampling(pooling) – 2*2 max pooling의 사례

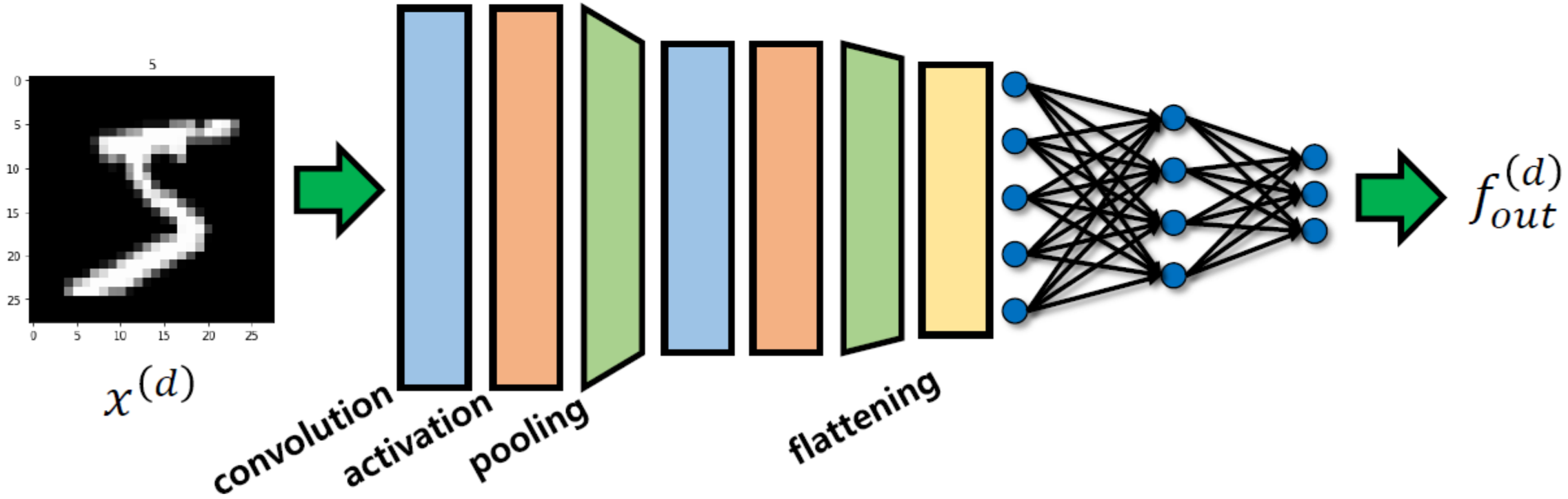


$1 + b_1$	$1.8 + b_1$	$3 + b_1$	$1 + b_1$
$0 + b_1$	$2 + b_1$	$1 + b_1$	$1 + b_1$
$0 + b_1$	$1 + b_1$	$1 + b_1$	$0 + b_1$
$0 + b_1$	$1 + b_1$	$1 + b_1$	$0 + b_1$

Pooling이 적용되는 범위 안에서 최댓값을 대표값으로 선정

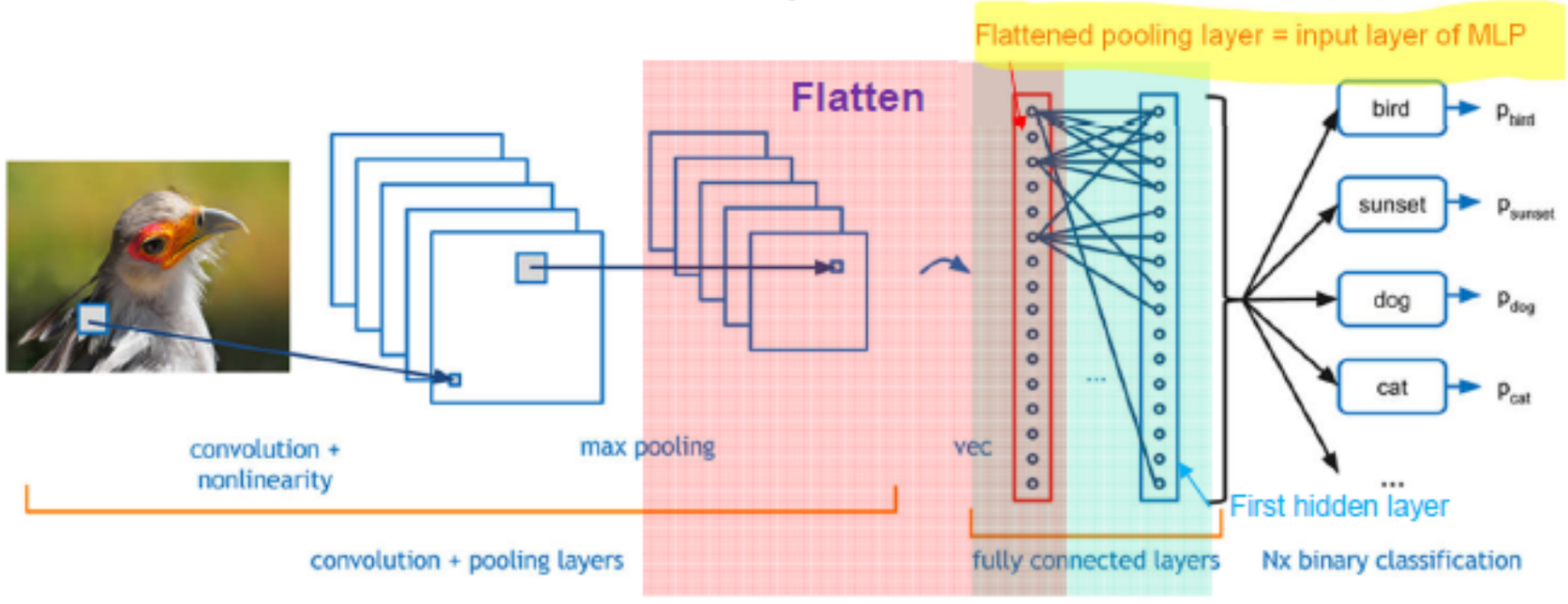
Convolution Neural Network (CNN)

- CNN 모델 구조
  - 일반적으로 **(컨볼루션층 – 활성함수(ReLU) – pooling)**이 한 묶음
    - 한 번의 Pooling 이전에 (컨볼루션층 – 활성함수) 구조가 반복되기도 함
  - CNN으로 처리한 $(height, width, channel)$ 크기의 (3차원) 결과 데이터를 $(height \times width \times channel)$ 차원의 1차원 데이터로 변환(flattening)하여 다층퍼셉트론에 넘김



Convolution Neural Network (CNN)

- FC layer started right after the last convolution or pooling layer.
- Last convolution layer (or pooling layer) in the network after flattened = input layer of MLP, then followed by FC layers.
- Last FC layer = output layer of ConvNet, will contain as many neurons as the number of classes to be predicted.



Convolution Neural Network (CNN)

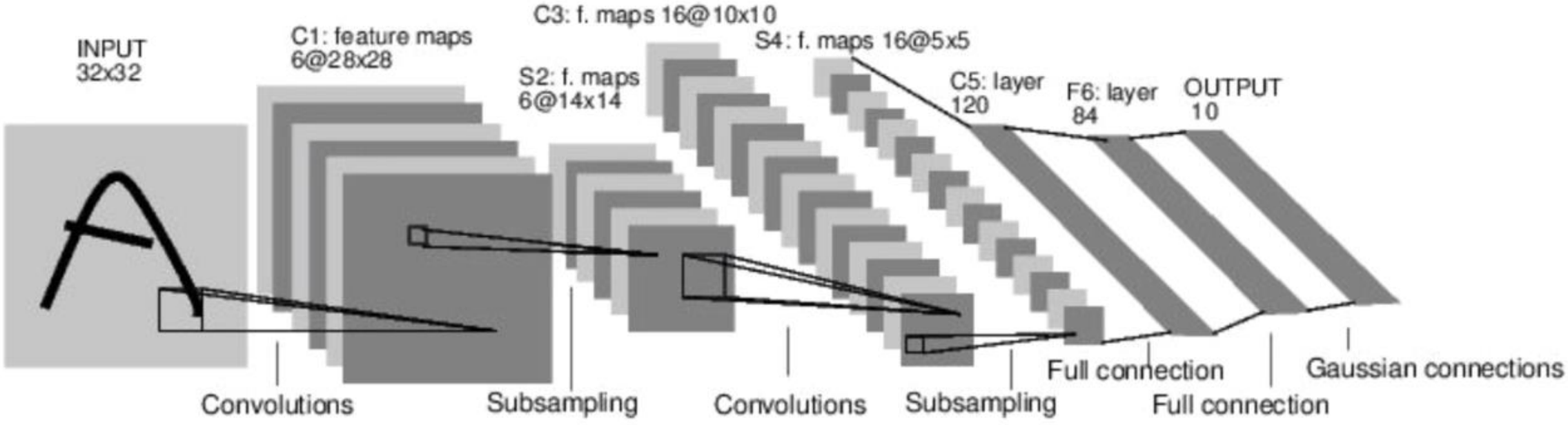
- 컨볼루션층의 구조
  - 주어진 조건
    - 입력 데이터 크기가 (가로, 세로, **깊이1**) 이고
    - (가로, 세로, **깊이1**) 크기의 필터가 feature map을 **깊이2** 개 생성
  - 가중치의 개수
    - (가로 * 세로 * **깊이1** * **깊이2**)
    - 입력 데이터 크기가 (128, 128, 1)일 때 (3, 3, 1) 크기의 필터를 64개 사용하여 깊이가 1인 feature map을 64개 생성하는 경우
      (3 * 3 * 1 * 64) + 64 = (9 * 64) + 64 = 640개의 가중치가 필요함
    - 다층 퍼셉트론을 사용하여 128*128 = 16384픽셀의 이미지를 100개의 뉴런으로 압축하는 경우 (16384+1) * 100 = 1638500개의 가중치를 학습해야 함

Convolution Neural Network (CNN)

- CNN의 특징
  - Local receptive field를 사용
    - 아래층의 값들이 **윗층의 일부에만 연결**됨
    - 두 층 사이의 뉴런들이 완전히 연결된 다층퍼셉트론보다 적은 수의 가중치를 사용함
  - 가중치 공유weight sharing
    - 같은 가중치가 같은 층에서 **여러 번 사용**됨
    - 더 적은 수의 가중치를 사용하면서도 이미지에서 나타나는 패턴의 작은 변형에 악영향을 덜 받게 됨
  - 하위추출subsampling을 수행
    - 아래층에서 추출한 데이터 중 일부만을 윗층에 전달
    - 가중치 공유와 같은 장점을 가짐

Convolution Neural Network (CNN)

[LeCun et al., 1998]

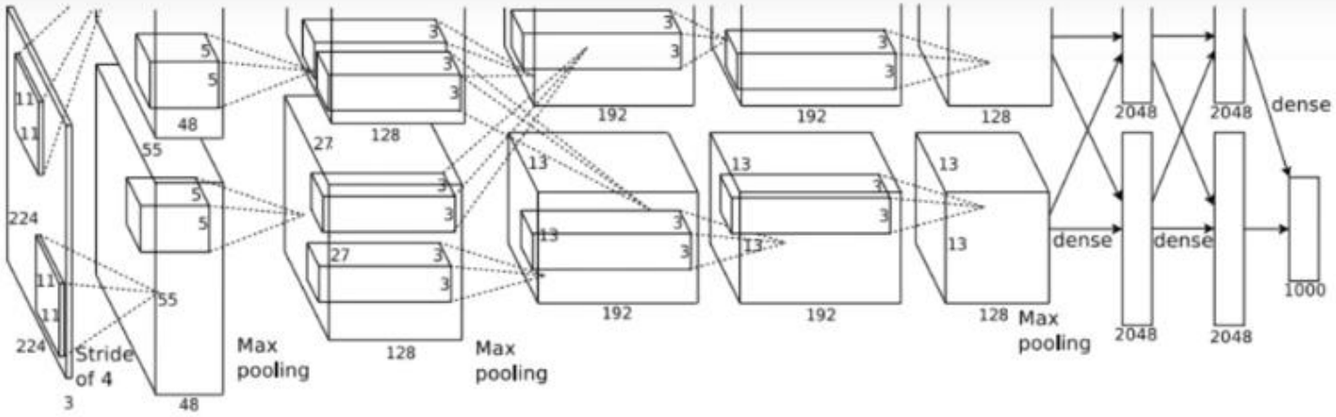


Conv filters were 5x5, applied at stride 1
Subsampling (Pooling) layers were 2x2 applied at stride 2
i.e. architecture is [CONV-POOL-CONV-POOL-CONV-FC]

Convolution Neural Network (CNN)

Case Study: AlexNet

[Krizhevsky et al. 2012]



Input: 227x227x3 images

First layer (CONV1): 96 11x11 filters applied at stride 4

=>

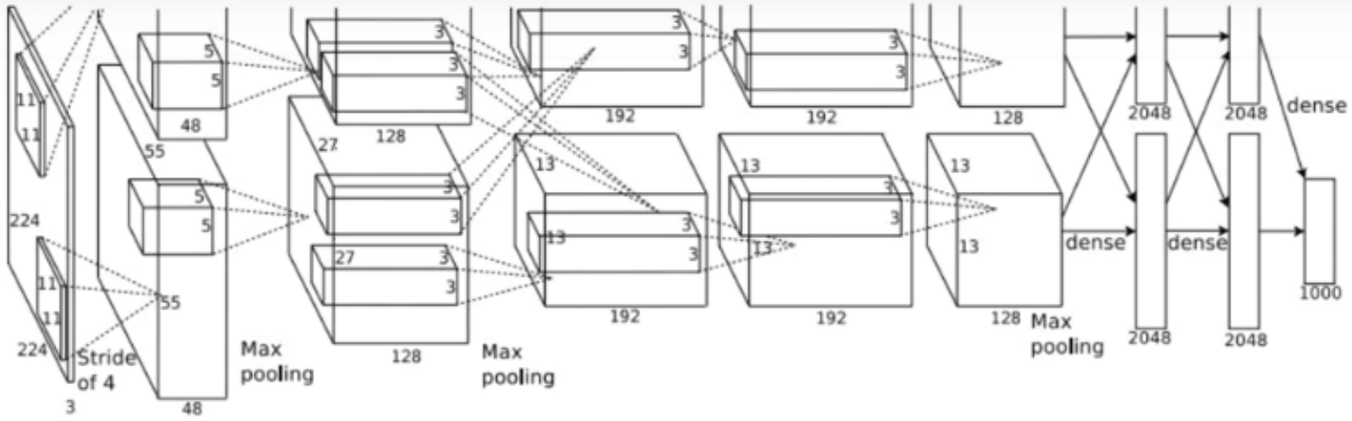
Output volume **[55x55x96]**

Parameters: (11*11*3)*96 = **35K**

Convolution Neural Network (CNN)

Case Study: AlexNet

[Krizhevsky et al. 2012]



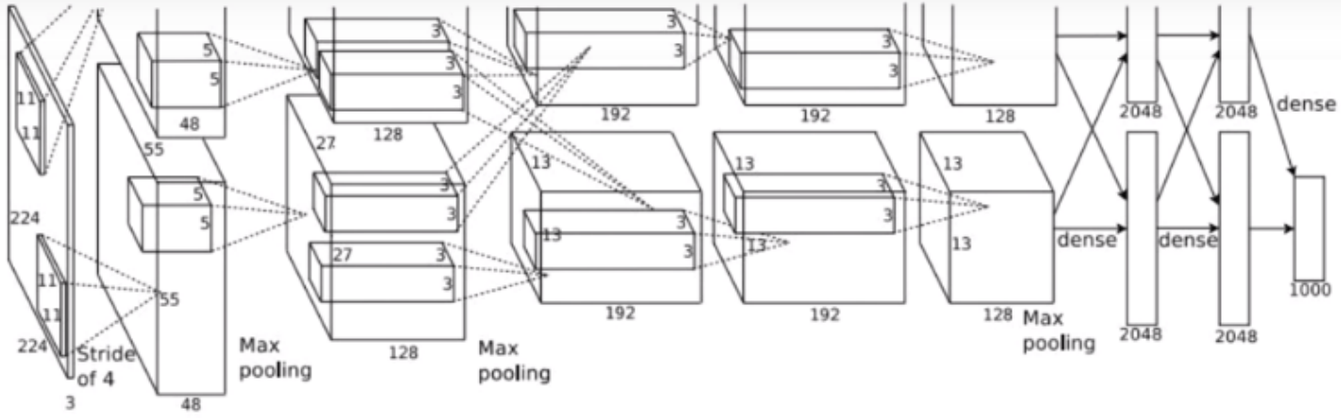
Input: 227x227x3 images
After CONV1: 55x55x96

Second layer (POOL1): 3x3 filters applied at stride 2
Output volume: 27x27x96
Parameters: 0!

Convolution Neural Network (CNN)

Case Study: AlexNet

[Krizhevsky et al. 2012]



Full (simplified) AlexNet architecture:
[227x227x3] INPUT
[55x55x96] CONV1: 96 11x11 filters at stride 4, pad 0
[27x27x96] MAX POOL1: 3x3 filters at stride 2
[27x27x96] NORM1: Normalization layer
[27x27x256] CONV2: 256 5x5 filters at stride 1, pad 2
[13x13x256] MAX POOL2: 3x3 filters at stride 2
[13x13x256] NORM2: Normalization layer
[13x13x384] CONV3: 384 3x3 filters at stride 1, pad 1
[13x13x384] CONV4: 384 3x3 filters at stride 1, pad 1
[13x13x256] CONV5: 256 3x3 filters at stride 1, pad 1
[6x6x256] MAX POOL3: 3x3 filters at stride 2
[4096] FC6: 4096 neurons
[4096] FC7: 4096 neurons
[1000] FC8: 1000 neurons (class scores)

THANK YOU