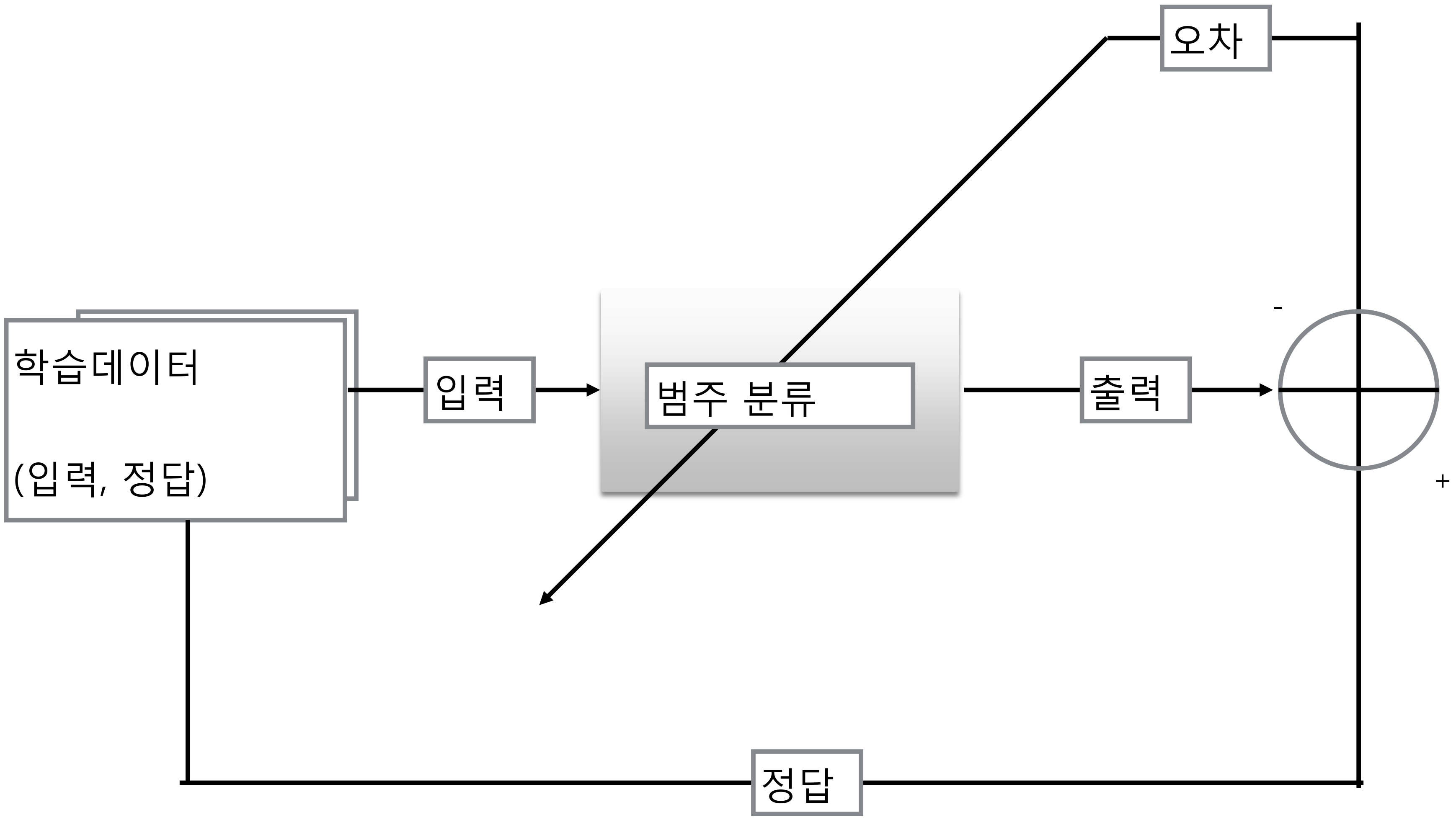
딥러닝 첫걸음

6. 컨벌루션 신경망

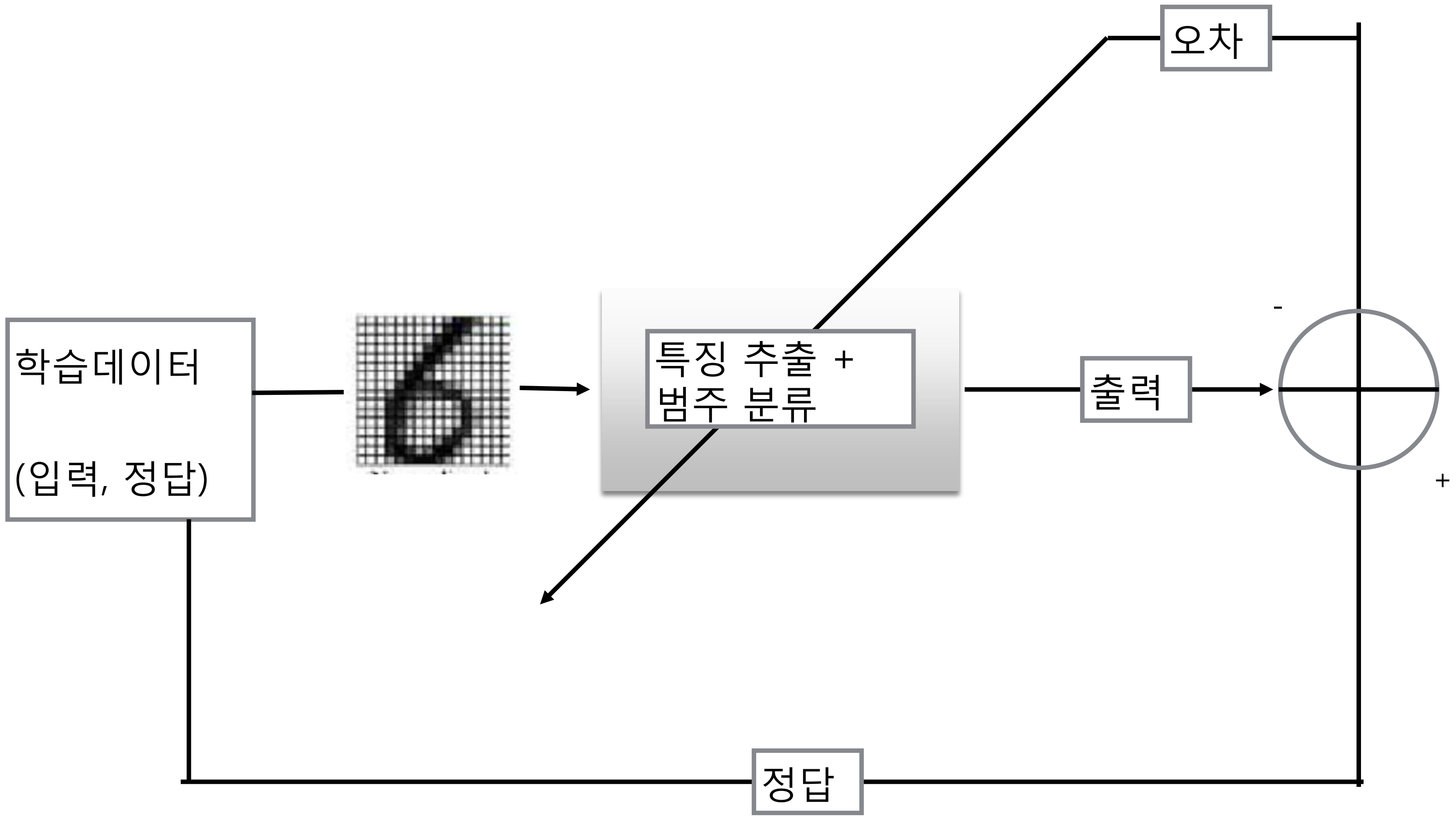
(CNN: Convolution Neural Network)

CNN: 개념

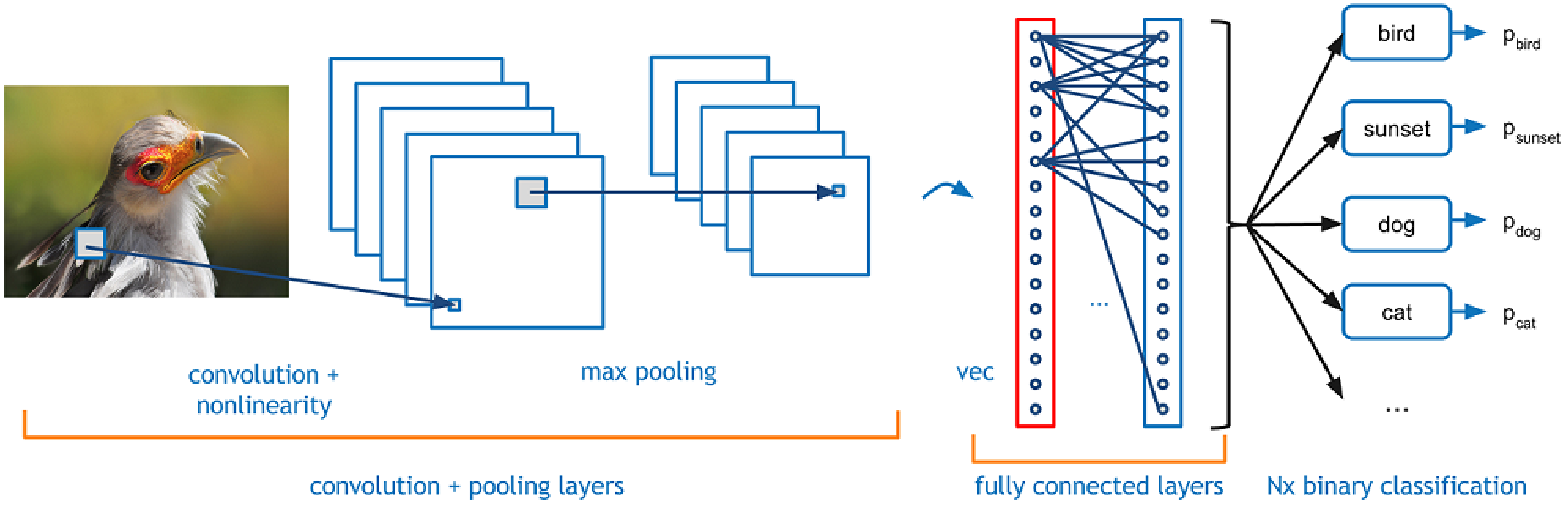
신경망 지도학습



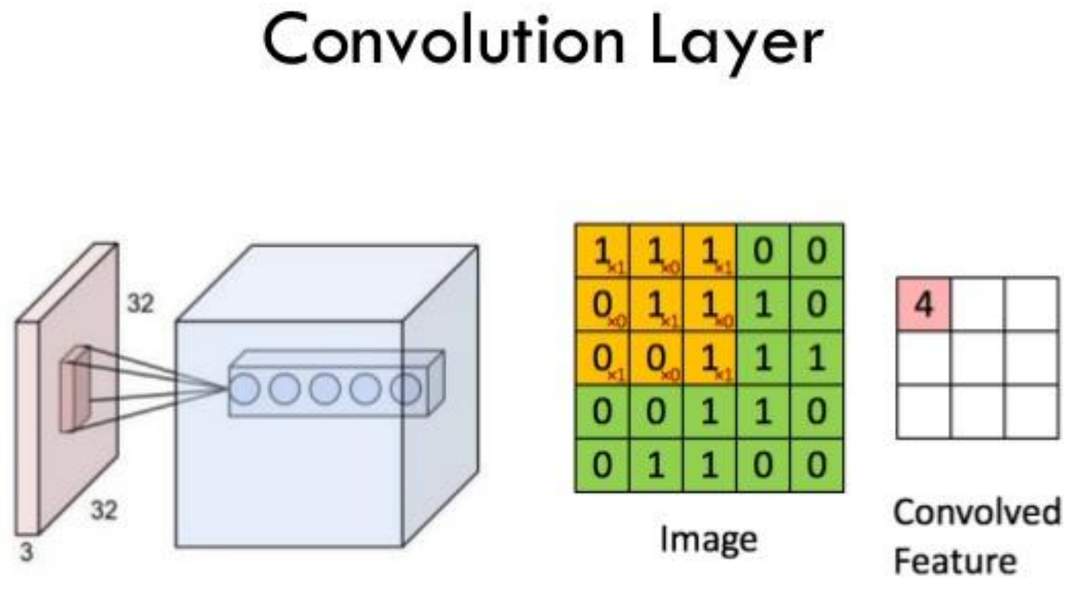
CNN 지도학습



특징추출: Convolution + Pooling
범주분류: Fully Connected layer



Convolution : Filtering

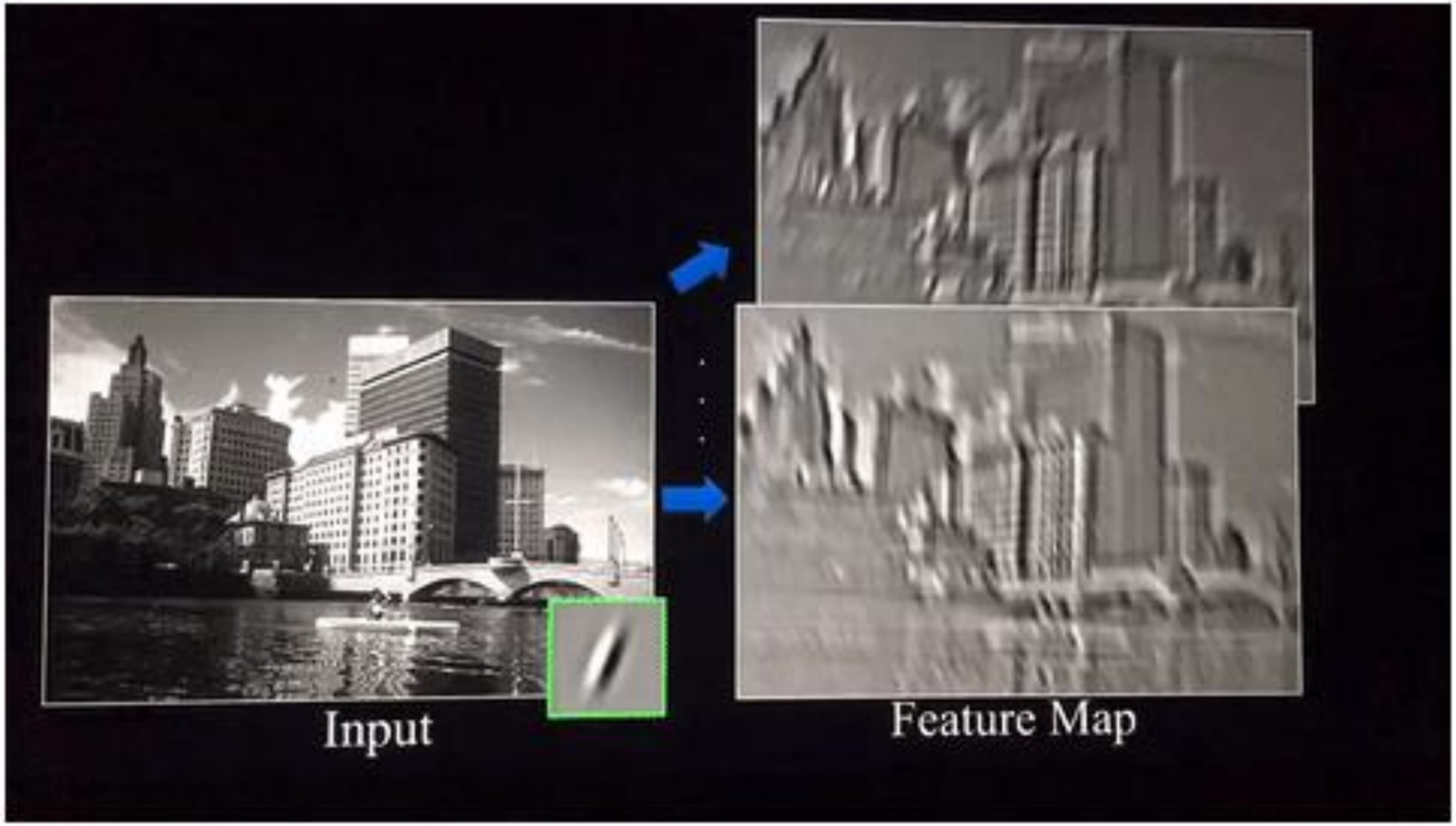


Andrej Karpathy and Fei-Fei. CS231n: Convolutional Neural Networks for Visual Recognition http://cs231n.github.io/convolutional-networks

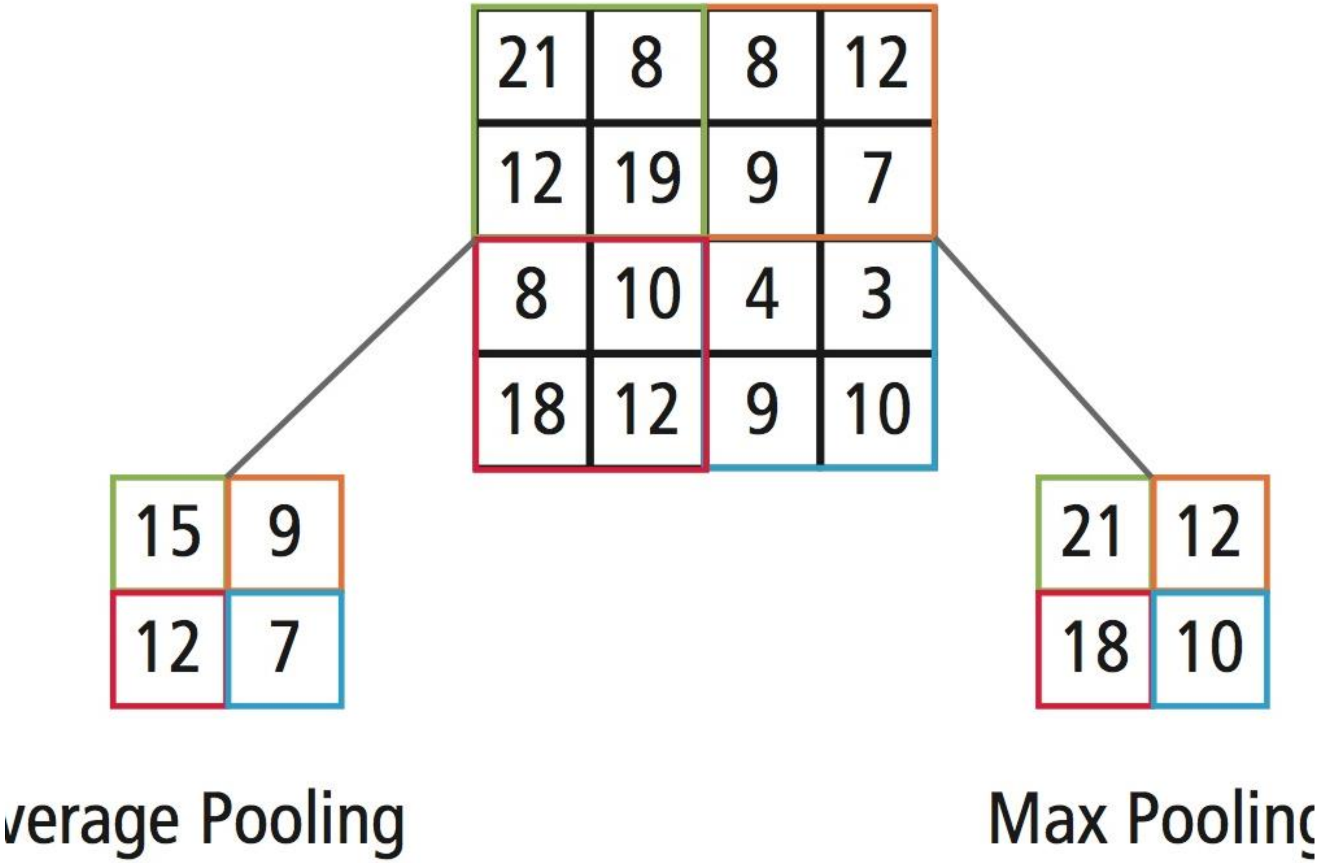
Yoshua Bengio, Ian Goodfellow and Aaron Courville. Deep Learning // An MIT Press

- 필터: 투입자료의 특성을 강조
- 하나의 레이어에서는 하나의 필터만 사용
  - 하나의 가중치가 여러 노드에 적용
- 여러개의 레이어를 깔아서 여러 필터를 적용(Depth)
  - 그림을 그 그림을 구성하는 여러가지 요소로 분해

Convolution 효과

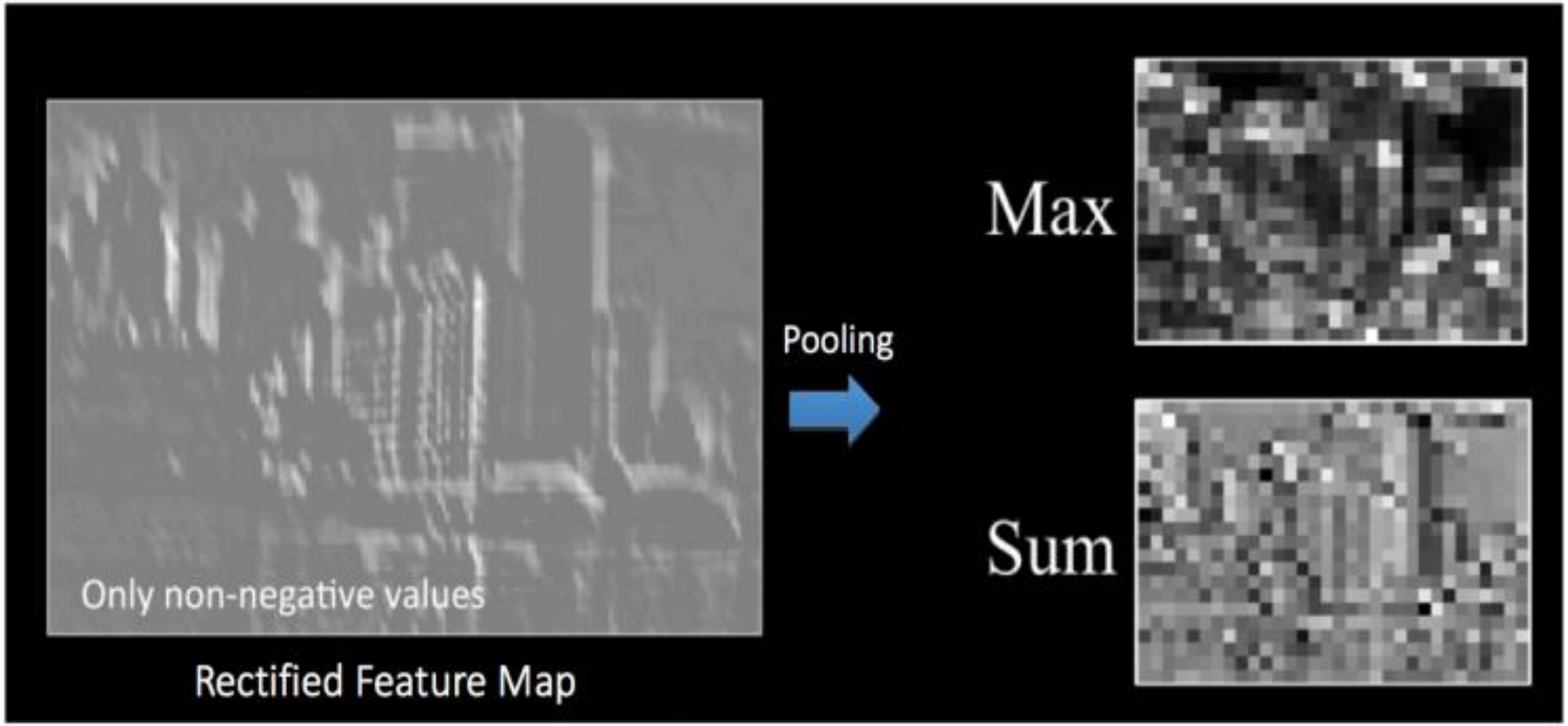


Convolution : Pooling

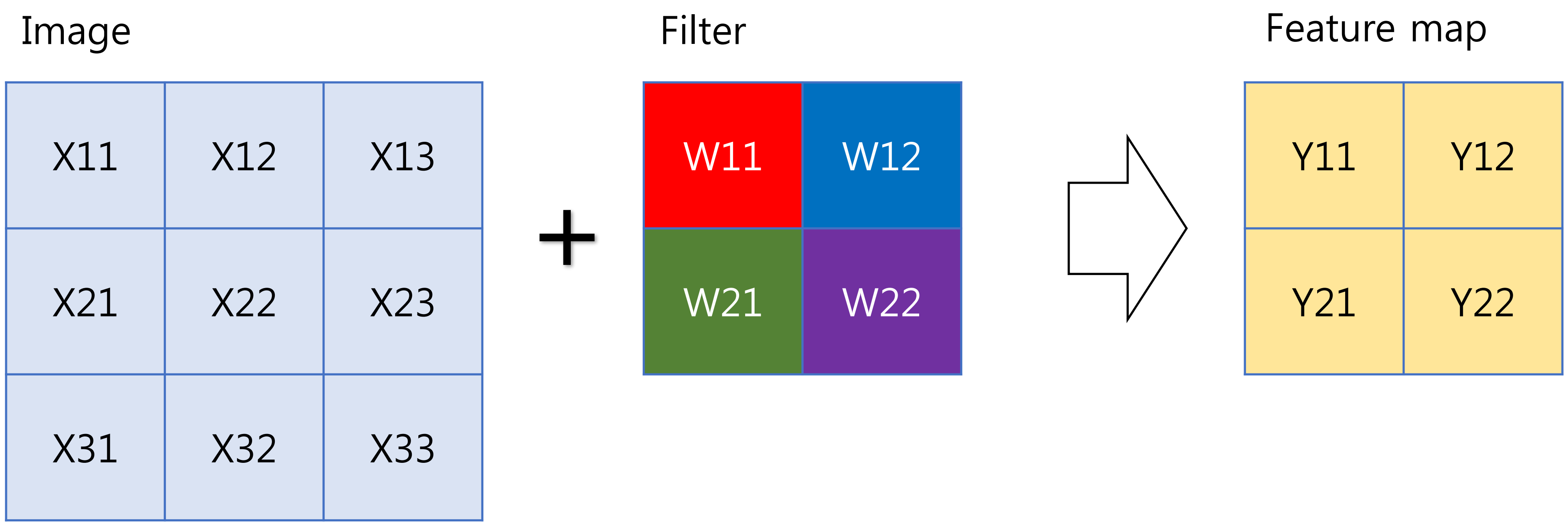


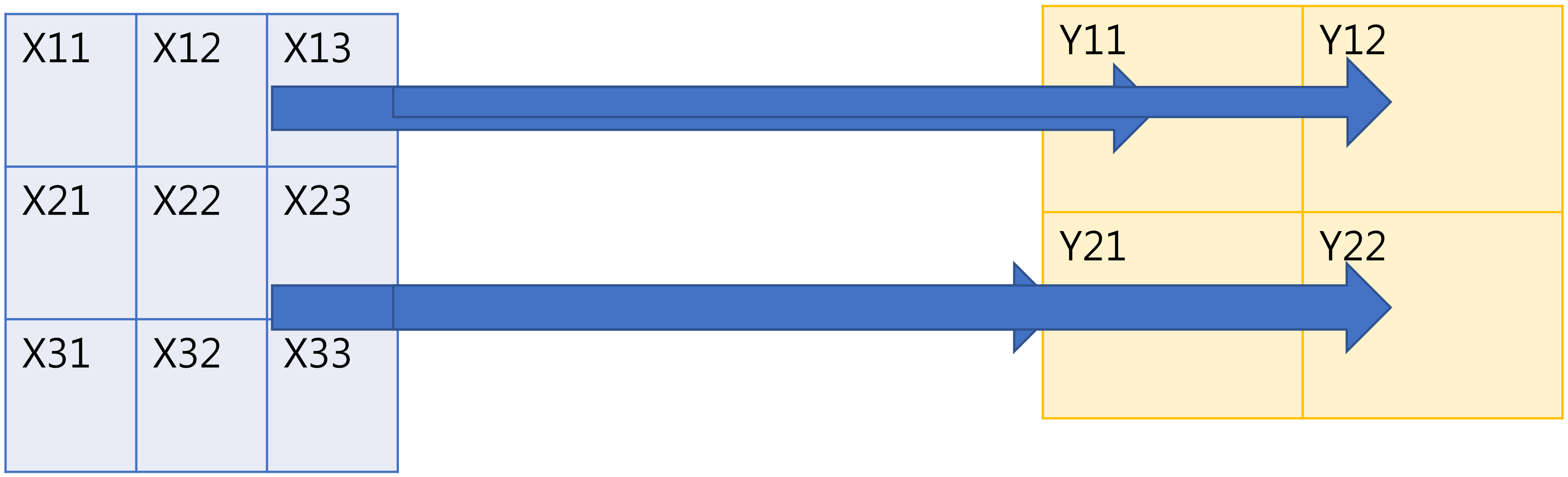
- 풀링: 여러 셀의 정보를 종합
    - 최대값/평균값을 주로 사용
- 풀링의 기능
    - 정보량 축소
    - 과적합 방지
    - robustness 작은 차이에는 둔감하게
    - scale invariant 크기에 큰 영향을 받지 않게

Pooling 효과


Only non-negative values
Rectified Feature Map
Pooling
Max
Sum

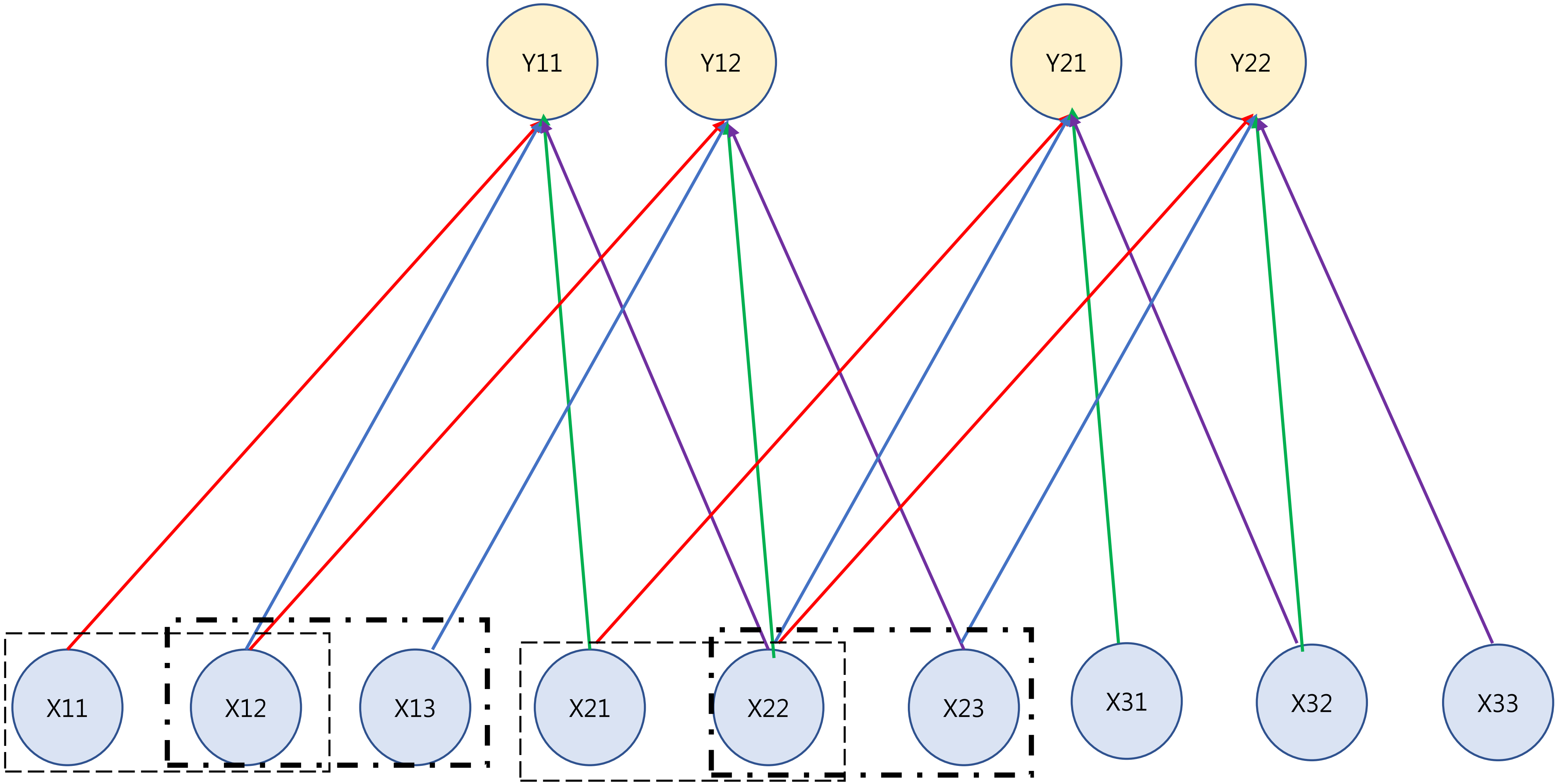
Convolution: Example





$$
\begin{aligned}
y_{11} &= W_{11}X_{11} + W_{12}X_{12} + W_{21}X_{21} + W_{22}X_{22} \\
y_{12} &= W_{11}X_{12} + W_{12}X_{13} + W_{21}X_{22} + W_{22}X_{23} \\
y_{21} &= W_{11}X_{21} + W_{12}X_{22} + W_{21}X_{31} + W_{22}X_{32} \\
y_{22} &= W_{11}X_{22} + W_{12}X_{23} + W_{21}X_{32} + W_{22}X_{33}
\end{aligned}
$$

1. 하나의 가중치가 여러번 쓰인다.
2. 모든 node가 다 연결되지 않는다.
3. 한 번 안 쓰인 node는 계속 안 쓰인다.



Pooling: Example

Image

X11	X12	X13	X14
X21	X22	X23	X24
X 31	X32	X33	X34
X41	X42	X43	X44

+

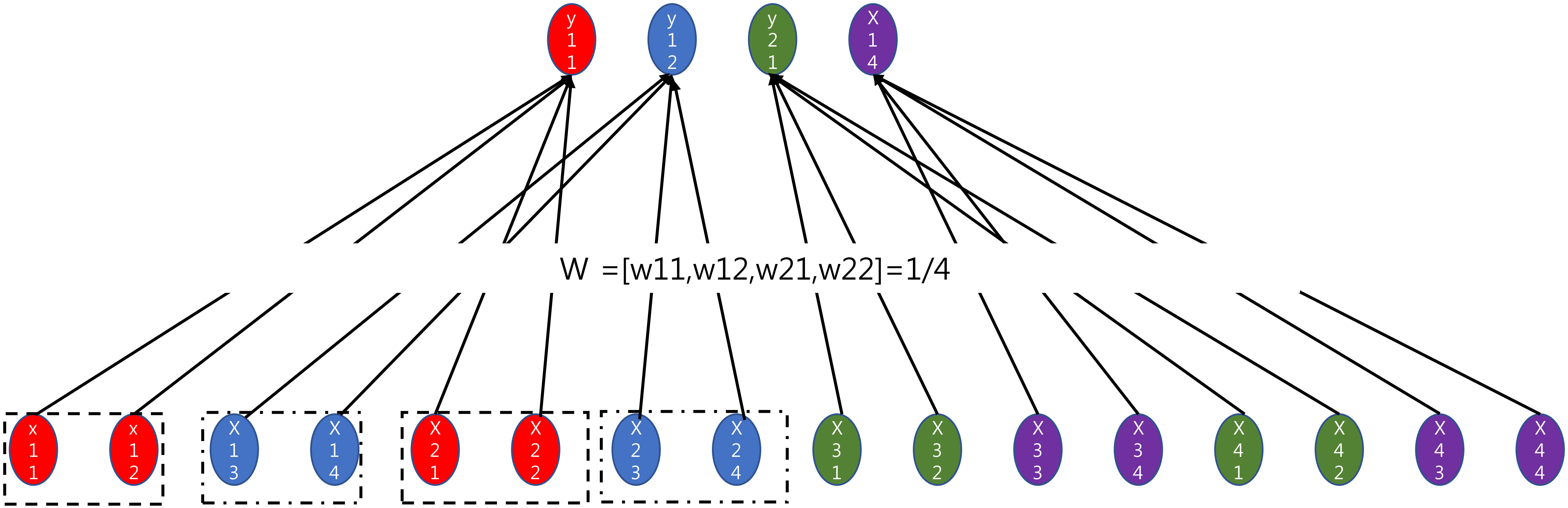
Filter

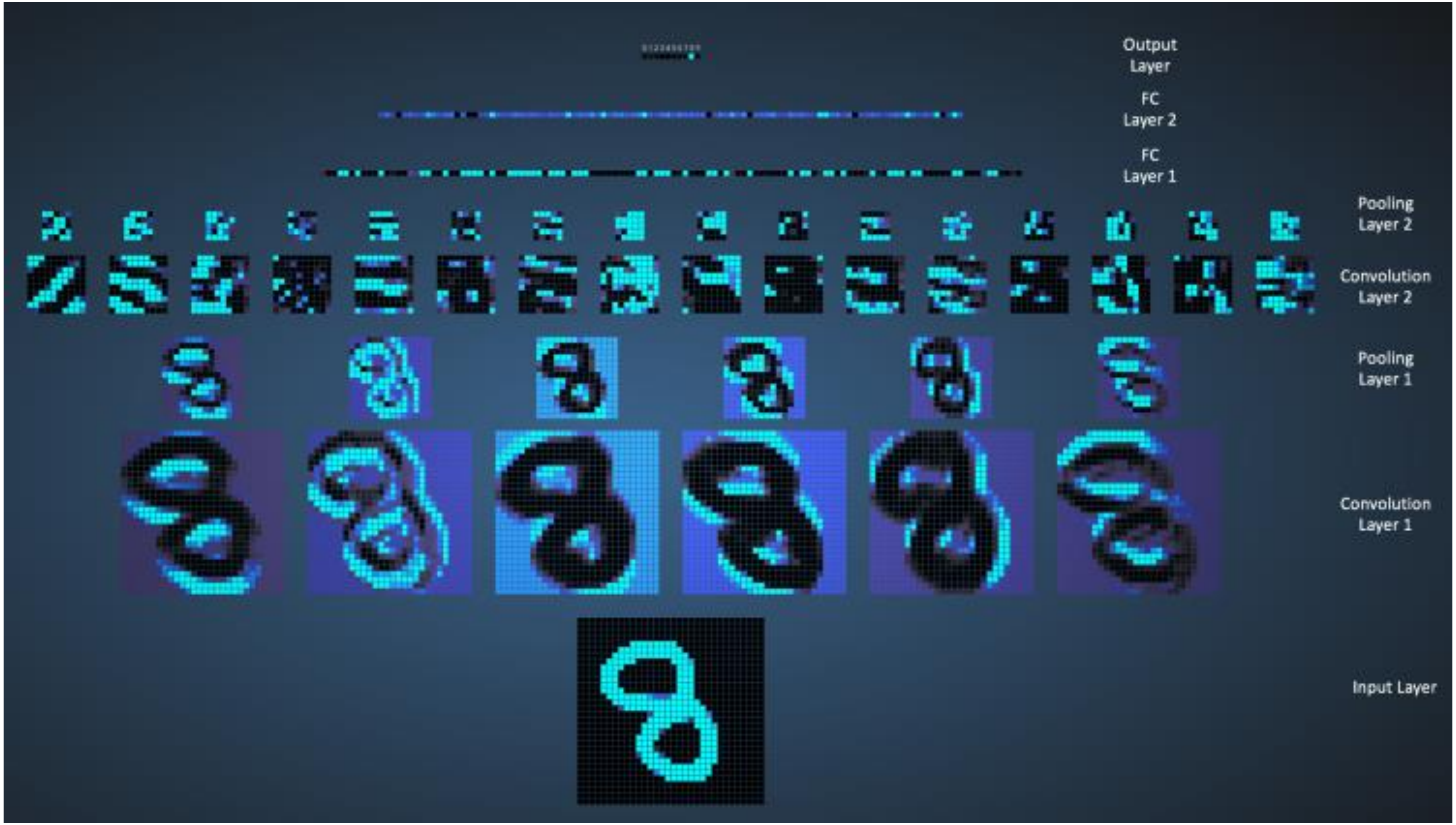
W11 =1/4	W12 =1/4
W21 =1/4	W22 =1/4

Pooled

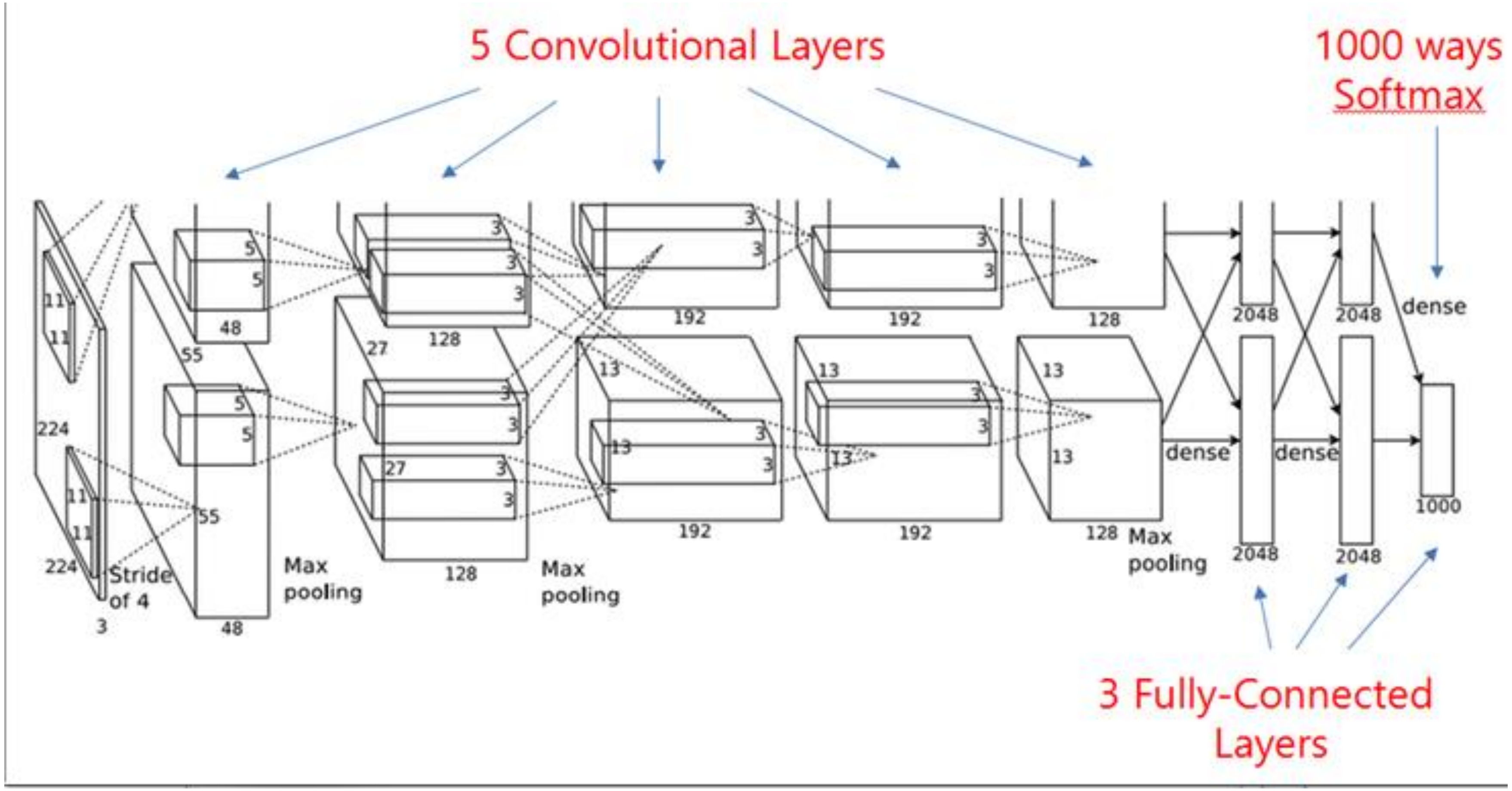
Y11 = [X11+X12+X21+X22] /4	Y12 = [X13+X14+X23+X24] /4
Y21 = [X31+X32+X41+X42] /4	Y22 = [X33+X34+X43+X44] /4

1. 가중치는 변하지 않는다.
2. 모든 node가 다 연결되지 않는다.
3. 한 번 안 쓰인 node는 계속 안 쓰인다.
4. 필터가 건너 뛰면서 적용된다. (Stride)




Output Layer
FC Layer 2
FC Layer 1
Pooling Layer 2
Convolution Layer 2
Pooling Layer 1
Convolution Layer 1
Input Layer

어느정도 복잡해 질 수 있을까?(Alexnet)



Krizhevsky, Alex, Ilya Sutskever, and Geoffrey E. Hinton. "Imagenet classification with deep convolutional neural networks." *Advances in neural information processing systems*. 2012.

2. CNN: math

Convolution: padding /Stride

- (zero) Padding : Image 주위에 all 0 행,렬을 더하여 확장
  - Valid convolution : zero padding이 없는 convolution
  - Full convolution : Image 정보가 포함된 가장 큰 convolution
  - Same convolution: Image와 크기가 같은 convolution
- Stride: Filter를 간격을 두고 적용

VALID

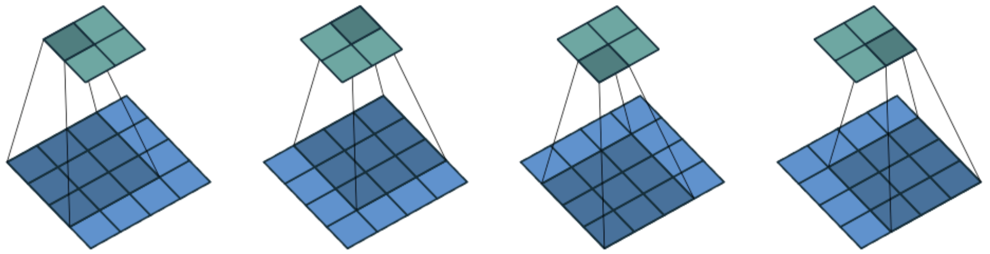


Figure 2.1: (No padding, no strides) Convolving a 3 × 3 kernel over a 4 × 4 input using unit strides (i.e., $i = 4$, $k = 3$, $s = 1$ and $p = 0$).

SAME

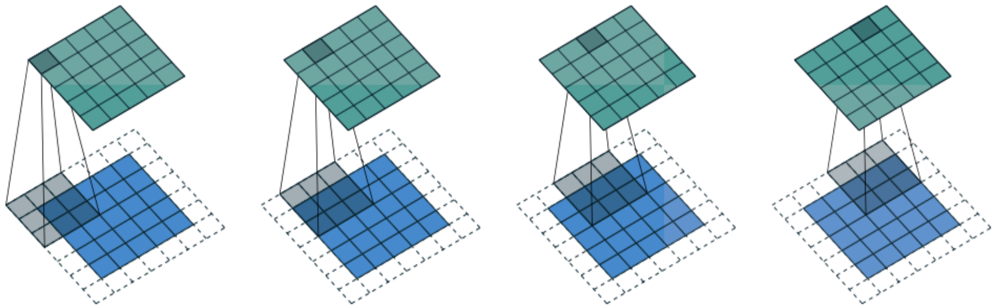


Figure 2.3: (Half padding, no strides) Convolving a 3 × 3 kernel over a 5 × 5 input using half padding and unit strides (i.e., $i = 5$, $k = 3$, $s = 1$ and $p = 1$).

FULL

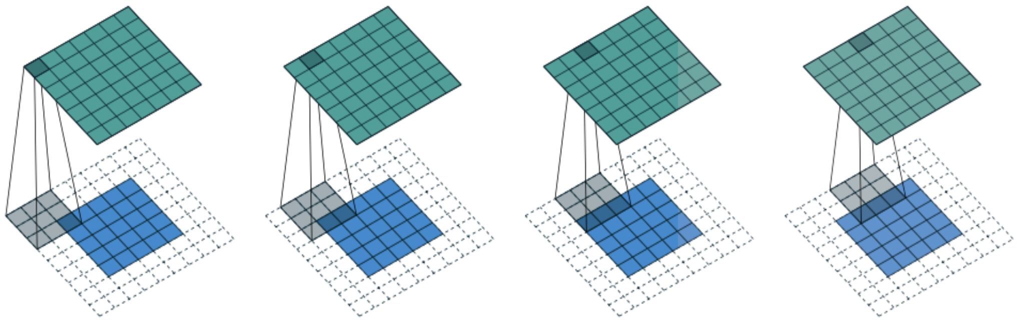


Figure 2.4: (Full padding, no strides) Convolving a 3 × 3 kernel over a 5 × 5 input using full padding and unit strides (i.e., $i = 5$, $k = 3$, $s = 1$ and $p = 2$).

Stride = 1 인 경우
VALID, SAME, FULL
Convolution

No padding + Stride

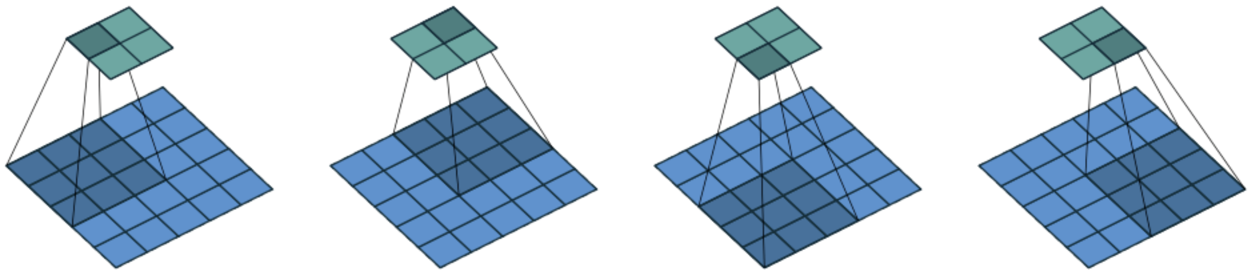


Figure 2.5: (No zero padding, arbitrary strides) Convolving a 3×3 kernel over a 5×5 input using 2×2 strides (i.e., $i = 5$, $k = 3$, $s = 2$ and $p = 0$).

Stride >1 인 경우

Padding and Stride

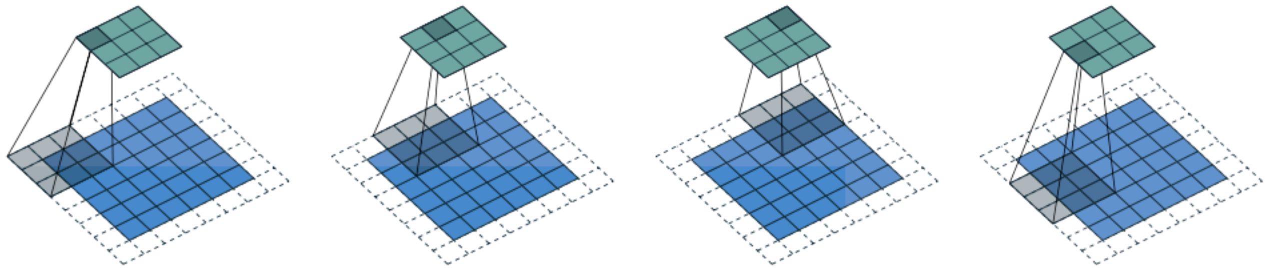


Figure 2.7: (Arbitrary padding and strides) Convolving a 3×3 kernel over a 6×6 input padded with a 1×1 border of zeros using 2×2 strides (i.e., $i = 6$, $k = 3$, $s = 2$ and $p = 1$). In this case, the bottom row and right column of the zero padded input are not covered by the kernel.

Cross Correlation. Convolution

$$X \in M^{m \times n}, W \in M^{k_1 \times k_2}, p = \text{padding}, s = \text{stride}$$

Cross-Correlation

$$Y_{i,j} = \sum_{a=1}^{k_1} \sum_{b=1}^{k_2} X_{1+s(i-1)+(a-1),1+s(j-1)+(b-1)} W_{a,b}$$

$$1 \le i \le \left\lfloor \frac{m - k_1 + 2p}{s} \right\rfloor + 1 \qquad \lfloor \quad \rfloor = \text{floor}$$

$$1 \le j \le \left\lfloor \frac{n - k_2 + 2p}{s} \right\rfloor + 1$$

Convolution

$$Y_{i,j} = \sum_{a=1}^{k} \sum_{b=1}^{k} X_{i+s(i-1)-(a-1),j+s(i-1)-(b-1)} W_{a,b}$$

Convolution: Cross-Correlation 을 180도 회전한 Filter를 이용하여 수행

No padding. Stride =1

$$X \in M^{m \times n}, W \in M^{k_1 \times k_2}, p = 0, s = 1$$

Cross-Correlation

$$Y_{i,j} \;=\; \sum_{a=1}^{k_1}\sum_{b=1}^{k_2} X_{i+(a-1),j+(b-1)} W_{a,b}$$

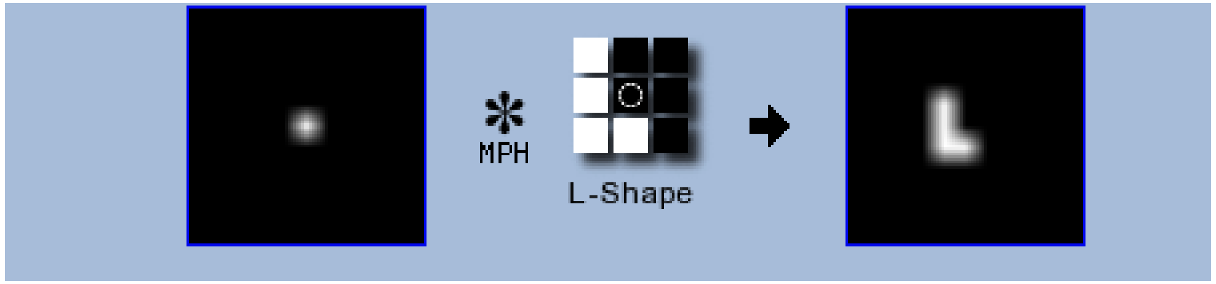
$$1 \le i \le m - k_1 + 1$$

$$1 \le j \le n - k_2 + 1$$

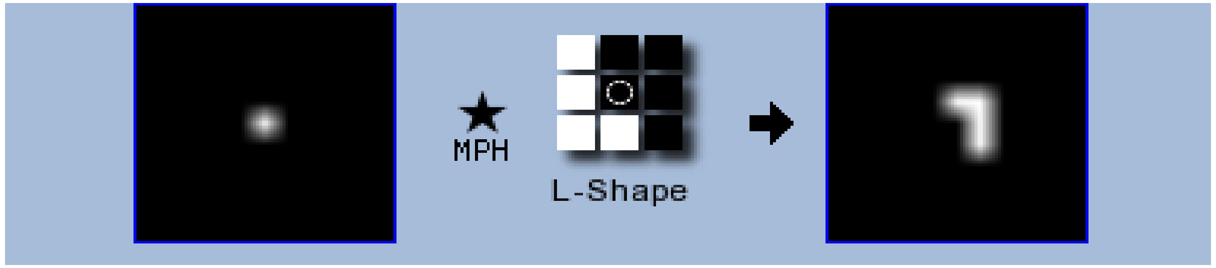
Convolution

$$Y_{i,j} \;=\; \sum_{a=1}^{k}\sum_{b=1}^{k} X_{i-(a-1),j-(b-1)} W_{a,b}$$

Convolution



Cross-Correlation



http://www.imagemagick.org/Usage/convolve/#convolve_vs_correlate

Convolution size : Example

Ex. Convolution

$$m = n = 3, k_1 = k_2 = 2, p = 0, s = 1$$
$$\rightarrow \quad i = j \leq \left[\frac{m - k_1 + 2p}{s} \right] + 1$$
$$= \quad \left[\frac{3 - 2 + 2 \times 0}{1} \right] + 1 = 2$$

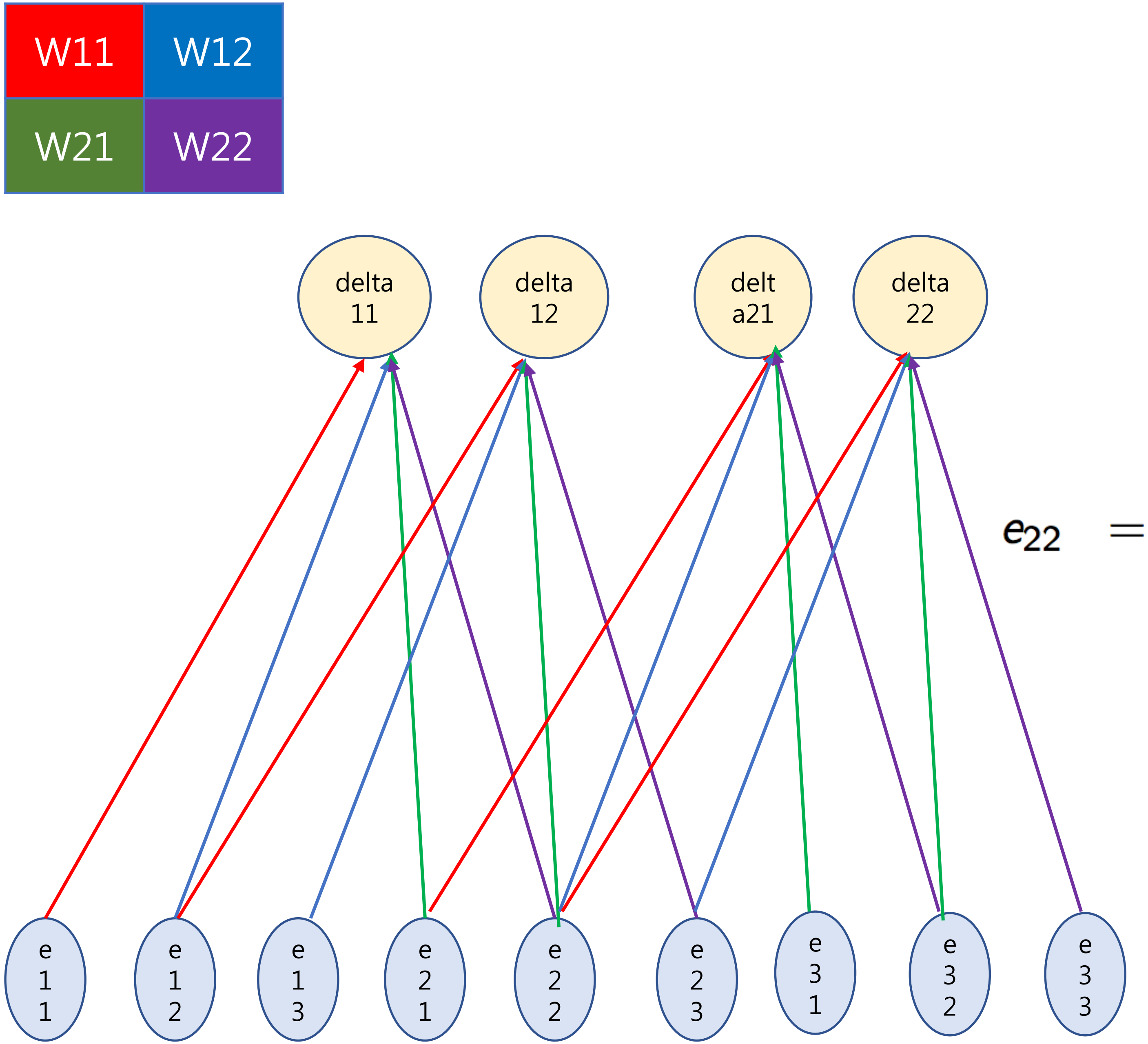
Ex. Pooling

$$m = n = 4, k_1 = k_2 = 2, p = 0, s = 2$$
$$\rightarrow \quad i = j \leq \left[\frac{m - k_1 + 2p}{s} \right] + 1$$
$$= \quad \left[\frac{4 - 2 + 2 \times 0}{2} \right] + 1 = 2$$

역전파 : Convolution/Pooling

- Backpropagation from conv layer to image layer : ‘Convolution’
  - Convolution : Image = delta, filter = ‘flipped’ weight
    - 180도 회전한 필터를 delta 에 적용시키는 Convolution
  - Backpropagation for ‘valid’ convolution => ‘Full’ convolution

- Backpropagation from pooling to ‘before pooling’ layer
  - Maximum pooling: 직전 은닉층에서 최대값이었던 node에 할당
    - 직전 은닉층 최대값 node를 저장
  - Average pooling =직전 은닉층에서 pooling 되었던 node에 평균값을 할당



$$e_{11} = W_{11}\delta_{11}$$

$$e_{12} = W_{12}\delta_{11} + W_{11}\delta_{12}$$

$$e_{13} = W_{12}\delta_{12}$$

$$e_{21} = W_{21}\delta_{11} + W_{11}\delta_{21}$$

$$e_{22} = W_{22}\delta_{11} + W_{21}\delta_{12} + W_{12}\delta_{21} + W_{11}\delta_{22}$$

$$e_{23} = W_{22}\delta_{12} + W_{12}\delta_{22}$$

$$e_{31} = W_{21}\delta_{21}$$

$$e_{32} = W_{22}\delta_{21} + W_{21}\delta_{22}$$

$$e_{33} = W_{22}\delta_{22}$$

Backpropagation: Convolution

Delta in L layer + padding

0	0	0	0
0	d11	d12	0
0	d21	d22	0
0	0	0	0

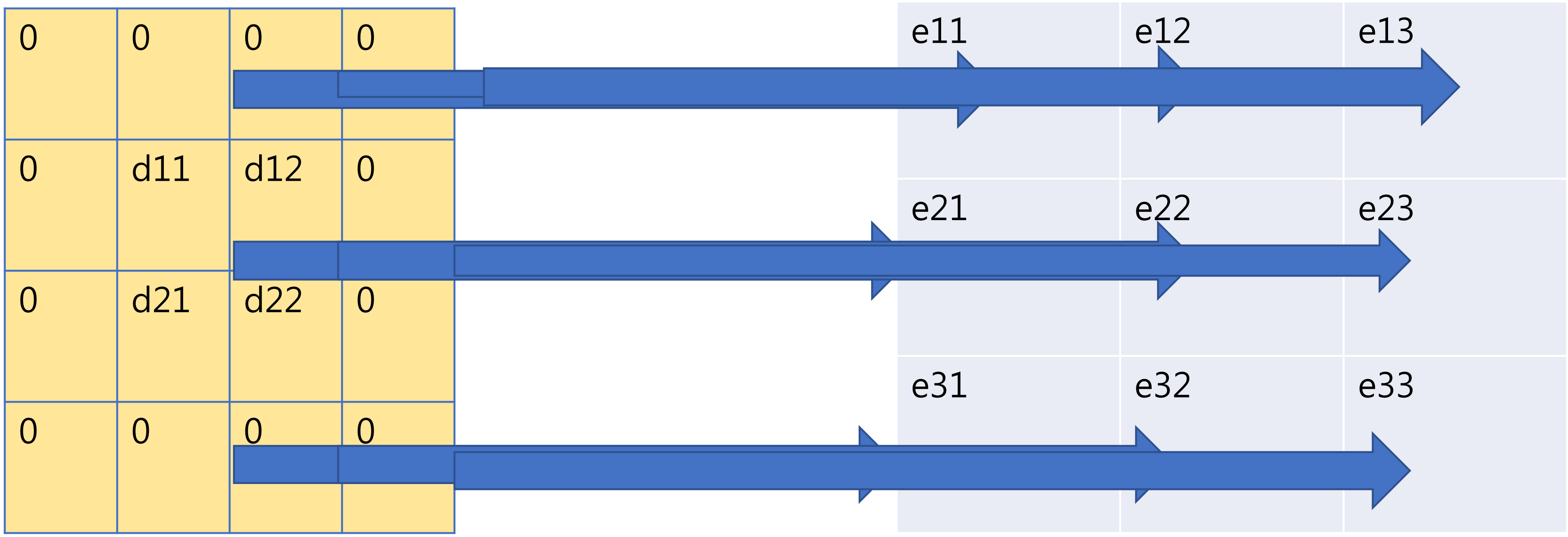
+

Filter + Rotation

W22	W21
W12	W11

Error in L-1 layer

e11	e12	e13
e21	e22	e23
e31	e32	e33



$$e_{11} = W_{11}\delta_{11}$$

$$e_{12} = W_{12}\delta_{11} + W_{11}\delta_{12}$$

$$e_{13} = W_{12}\delta_{12}$$

$$e_{21} = W_{21}\delta_{11} + W_{11}\delta_{21}$$

W22	W21
W12	W11

$$e_{22} = W_{22}\delta_{11} + W_{21}\delta_{12} + W_{12}\delta_{21} + W_{11}\delta_{22}$$

$$e_{23} = W_{22}\delta_{12} + W_{12}\delta_{22}$$

$$e_{31} = W_{21}\delta_{21}$$

$$e_{32} = W_{22}\delta_{21} + W_{21}\delta_{22}$$

$$e_{33} = W_{22}\delta_{22}$$

Pooling: Backpropagation

Image

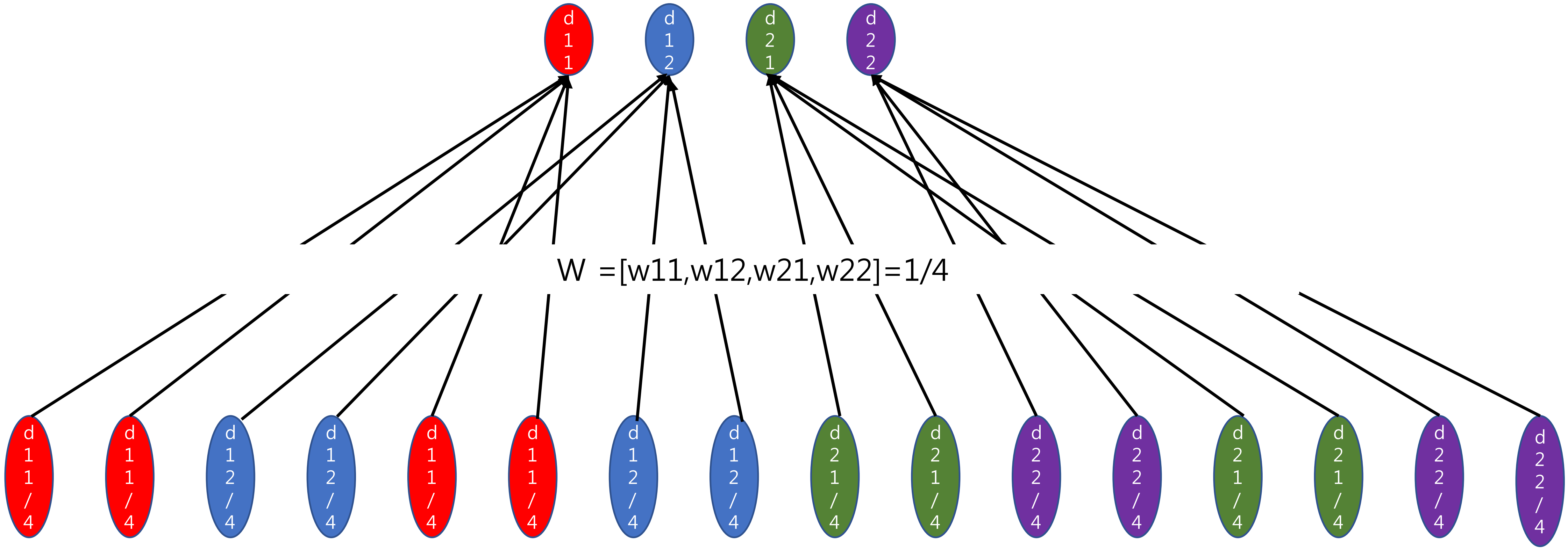
E11 = Delta11/4	E12 = Delta11/4	E13 = Delta12/4	E14 = Delta12/4
E21 = Delta11/4	E22 = Delta11/4	E23 = Delta12/4	E24 = Delta12/4
E 31 = Delta21/4	E32 = Delta21/4	E33 = Delta22/4	E34 = Delta22/4
E41 = Delta21/4	E42 = Delta21/4	E43 = Delta22/4	X44 = Delta22/4

Filter

W11 =1/4	W12 =1/4
W21 =1/4	W22 =1/4

Pooled

delta11	delta12
delta21	delta22


d11
d12
d21
d22
W =[w11,w12,w21,w22]=1/4
d11/4
d11/4
d12/4
d12/4
d11/4
d11/4
d12/4
d12/4
d21/4
d21/4
d22/4
d22/4
d21/4
d21/4
d22/4
d22/4

Kronecker Product

$$\begin{bmatrix} a & b \\ c & d \end{bmatrix} \otimes \begin{bmatrix} a^* & c^* \\ b^* & d^* \end{bmatrix} = \begin{bmatrix} a\begin{bmatrix} a^* & c^* \\ b^* & d^* \end{bmatrix} & b\begin{bmatrix} a^* & c^* \\ b^* & d^* \end{bmatrix} \\ c\begin{bmatrix} a^* & c^* \\ b^* & d^* \end{bmatrix} & d\begin{bmatrix} a^* & c^* \\ b^* & d^* \end{bmatrix} \end{bmatrix}$$

$$= \begin{bmatrix} aa^* & ac^* & ba^* & bc^* \\ ab^* & ad^* & bb^* & bd^* \\ ca^* & cc^* & da^* & dc^* \\ cb^* & cd^* & db^* & dd^* \end{bmatrix}$$

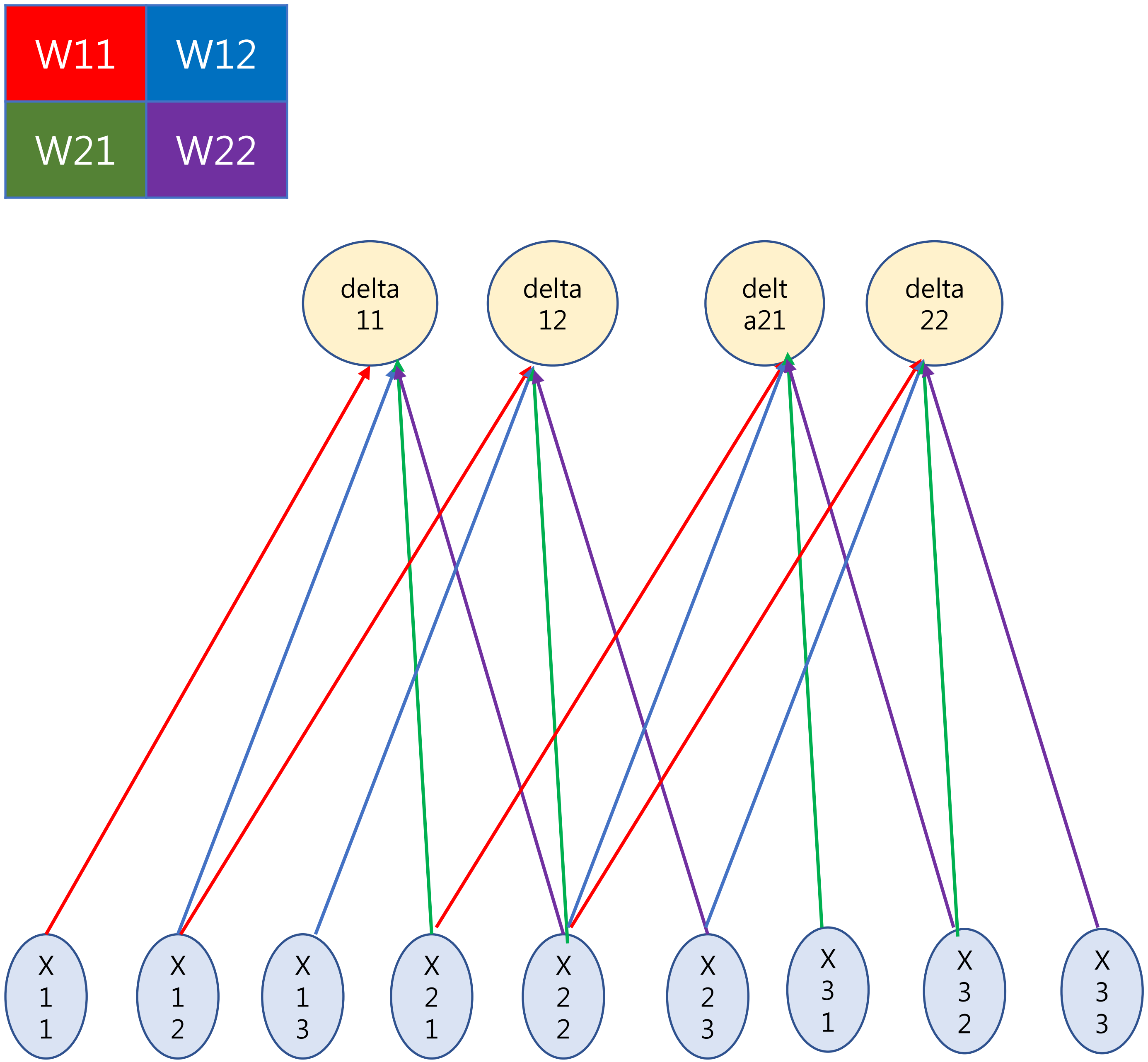
가중치 조정: Convolution

- Convolution layer 가중치 조정 : Convolution
  - 하나의 가중치가 여러번 사용: 개별 node 조정치의 합
  - Image 가 입력자료 이고, 필터가 Convolution층의 delta 인 Convolution의 Feature map

$$\Delta W_{i,j} = \sum_{a=1}^{k_1} \sum_{b=1}^{k_2} X_{i+(a-1),j+(b-1)} \delta_{a,b}$$

$$1 \le i \le m - k_1 + 1$$

$$1 \le j \le n - k_2 + 1$$



$$\Delta W_{11} = \delta_{11}X_{11} + \delta_{12}X_{12} + \delta_{21}X_{21} + \delta_{22}X_{22}$$

$$\Delta W_{12} = \delta_{11}X_{12} + \delta_{12}X_{13} + \delta_{21}X_{22} + \delta_{22}X_{23}$$

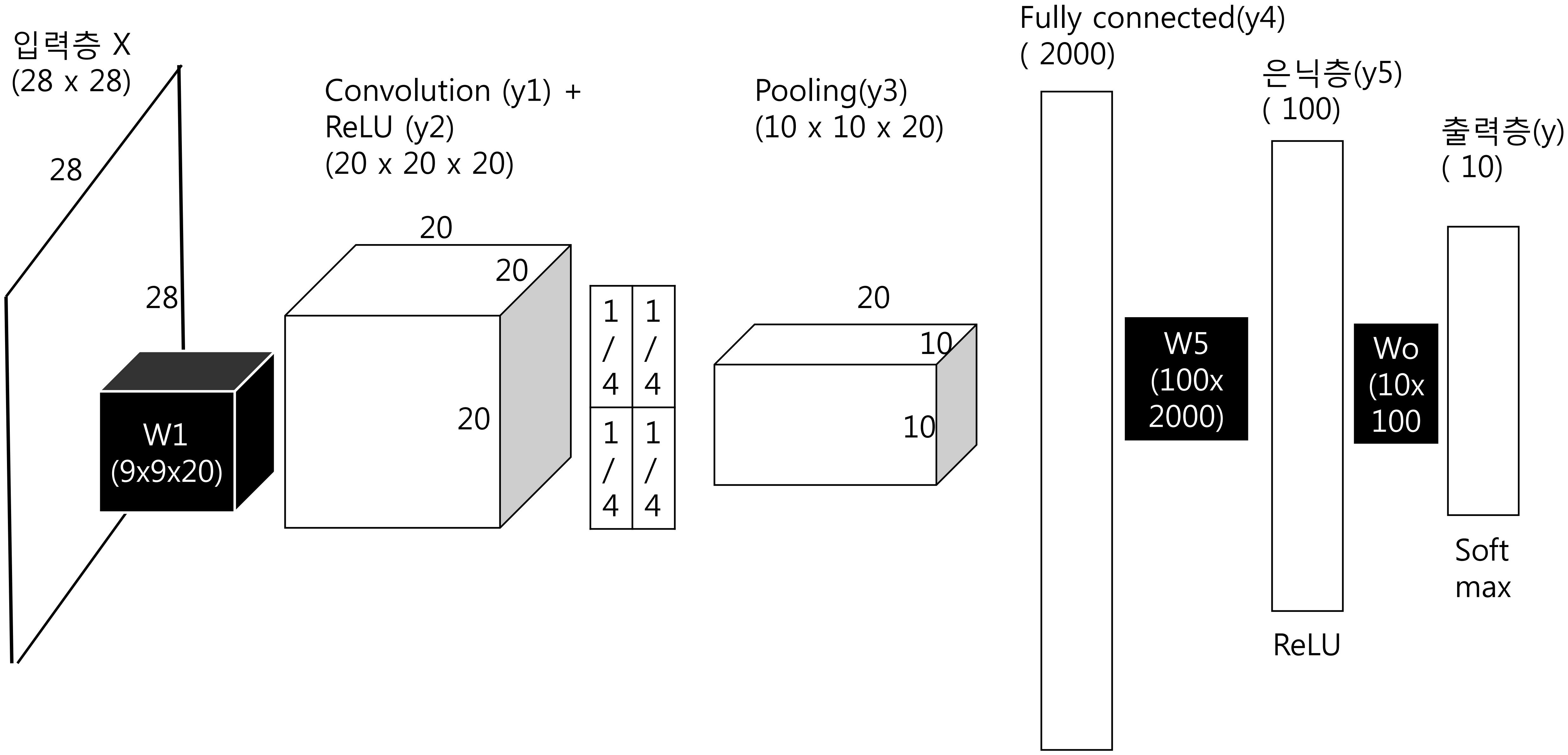
$$\Delta W_{21} = \delta_{11}X_{21} + \delta_{12}X_{22} + \delta_{21}X_{31} + \delta_{22}X_{32}$$

$$\Delta W_{22} = \delta_{11}X_{22} + \delta_{12}X_{23} + \delta_{21}X_{32} + \delta_{22}X_{33}$$

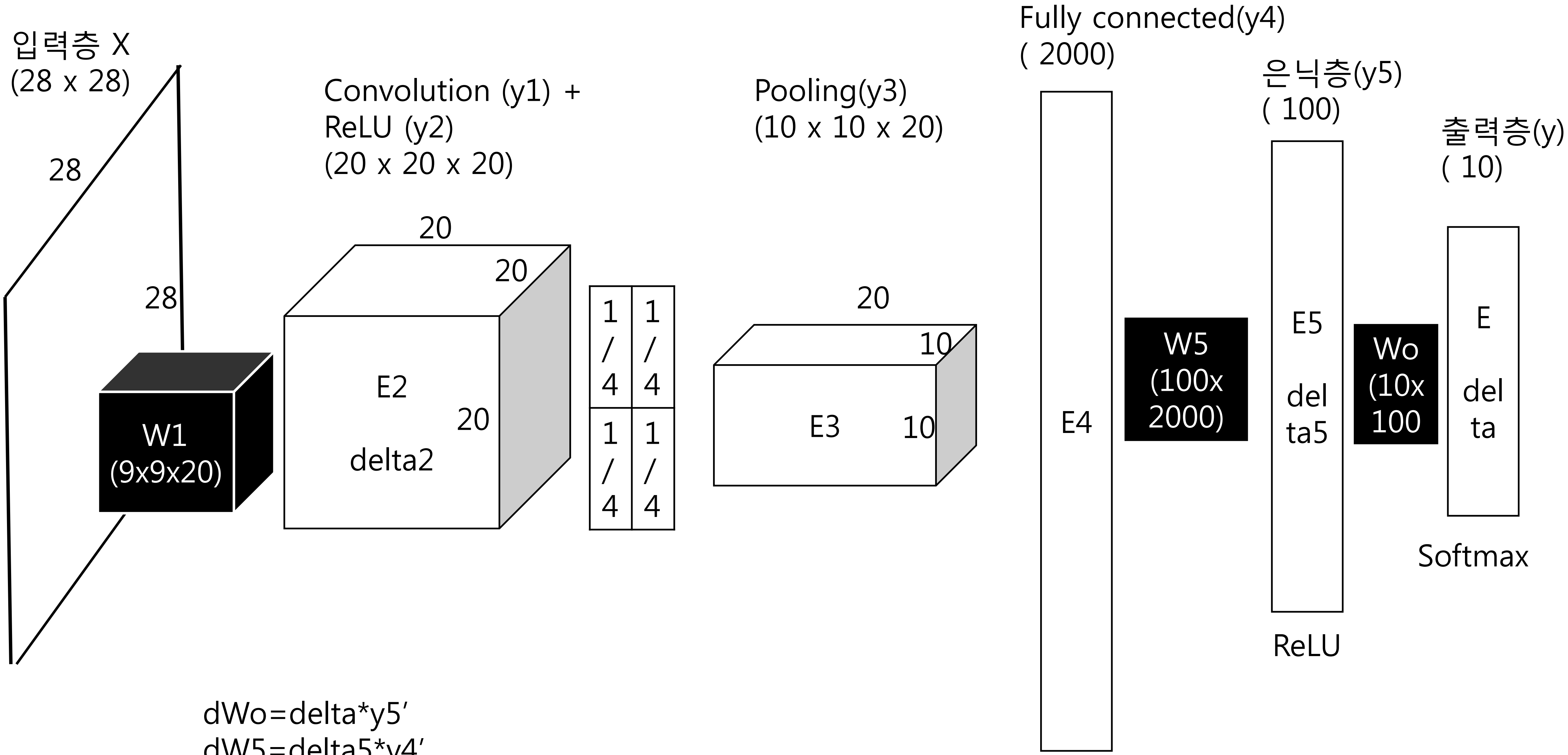
CNN: Example

MNIST data set +CNN

- MNIST data set : 0-9 손글씨
  - 학습 6만개, 검증 1만개로 구성
  - 교재: 검증 1만개 중 8천개를 학습자료, 2천개를 검증자료로 사용
- CNN: 1 개 Convolution, 1개 Pooling, 1개 은닉층, 1개 출력층
  - 입력: 28 x 28 matrix
  - Convolution : 9 x 9 필터 20개 (9 x 9 x 20) => 20 x 20 x 20
    - Activation : ReLU
  - Pooling: 2 x 2 필터 20개 (2 x 2 x 20) => 10 x 10 x 20
  - 은닉층 : 100 node/ Activation: ReLU
  - 출력층: 10 node/ Activation: Softmax
  - 가중치: W1(입력→ Convolution), W5(Pooling → 은닉), Wo(은닉 → 출력)



입력층 X
(28 x 28)
28
28
W1
(9x9x20)
Convolution (y1) +
ReLU (y2)
(20 x 20 x 20)
20
20
20
1/4
1/4
1/4
1/4
Pooling(y3)
(10 x 10 x 20)
20
10
10
Fully connected(y4)
(2000)
W5
(100x
2000)
은닉층(y5)
(100)
ReLU
Wo
(10x
100
출력층(y)
(10)
Soft
max



dWo=delta*y5′
dW5=delta5*y4′
dW1=conv(X,delta2)

Convolution: Conv.m/Conv.r

Conv.m

```
function y=Conv(x,W)
[wrow,wcol,numFilters]=size(W);
[xrow,xcol,~]=size(x);
```

1. Convolution Size

```
yrow=xrow-wrow+1;
ycol=xcol-wcol+1;

y=zeros(yrow,ycol,numFilters);
```

2. Convolution calculation

```
for k=1:numFilters

filter=W(:,:,k);
filter=rot90(squeeze(filter),2);
y(:,:,k)=conv2(x,filter,'valid');
end

end
```

Conv.r

```
Conv=function(x,W){
wrow=dim(W)[1]
wcol=dim(W)[2]
numFilters=dim(W)[3]
xrow=dim(x)[1]
xcol=dim(x)[2]
```

1. Convolution Size

```
yrow=xrow-wrow+1
ycol=xcol-wcol+1

y=array(data=rep(0,yrow*ycol*numFilters),dim=c(yrow,ycol,numFilters))
```

```
# convolution with unfilpped filter
for (k in 1:numFilters){
  filter=W[,,k]
  filter=matrix(filter,nrow=wrow,ncol=wcol)
  for (k1 in 1:yrow){
    for(k2 in 1:ycol){
      xij=x[(k1:(k1+wrow-1)),(k2:(k2+wcol-1))]
      y[k1,k2,k]=sum(xij*filter)
    }
  }
}
return(y)
}
```

Pooling: Pool.m/Pool.r

Pool.m

```
function y = Pool(x)

1. Size of Pooled image

[xrow, xcol,numFilters]=size(x);

y=zeros(xrow/2,xcol/2,numFilters);


2. Convolution

for k=1:numFilters

filter=ones(2)/(2*2);

image=conv2(x(:,:,k),filter,'valid');

3. Discard inbetween (Not included b/c stride>1)

y(:,:,k)=image(1:2:end,1:2:end);%pick every other elements of
convolution (skip =1)

end


end
```

Pool.r

```
Pool=function(x){
xrow=dim(x)[1]
xcol=dim(x)[2]
numFilters=dim(x)[3]

#1. Size of Pooled image
yrow=xrow-2+1
ycol=xcol-2+1
ybig=array(data=rep(0,yrow*ycol*numFilters),dim=c(yrow,ycol,numFilters))

# 2. Convolution

for (k in 1:numFilters){
  filter=matrix(rep(1/4,4),nrow=2,ncol=2)
  for (k1 in 1:yrow){
    for(k2 in 1:ycol){
      xij=x[(k1:(k1+2-1)),(k2:(k2+2-1)),k]
      ybig[k1,k2,k]=sum(xij*filter)
    }
  }

}

# 3. Discard inbetween (Not included b/c stride>1)

pickr=seq(from=1,to=yrow,by=2)
pickc=seq(from=1,to=ycol,by=2)
y=ybig[pickr,pickc,]
return(y)
}
```

Training: MNISTConv.m/MNISTconv.r

MNISTConv.m

```
function[W1,W5,Wo]=MnistConv(W1,W5,Wo,X,D)
   alpha=0.01;
   beta=0.95;
    momentum1=zeros(size(W1));
  momentum5=zeros(size(W5));
   momentumo=zeros(size(Wo));
N=length(D);
   bsize=100;
   blist=1:bsize:(N-bsize+1);
   %One epoch loop

for batch=1:length(blist)
     dW1=zeros(size(W1));
     dW5=zeros(size(W5));
       dWo=zeros(size(Wo));
```

MNISTConv.r

```
MnistConv=function(W1,W5,Wo,X,D){
  alpha=0.01
  beta=0.95
 momentum1=array(data=rep(0,prod(dim(W1))),dim=dim(W1))
 momentum5=array(data=rep(0,prod(dim(W5))),dim=dim(W5))
  momentumo=array(data=rep(0,prod(dim(Wo))),dim=dim(Wo))


  N=length(D)
  #define batch size
  bsize=100
  blist=seq(from=1,by=bsize,to=(N-bsize+1))
  ## one epoch loop
  for (batch in 1:length(blist)){
   dW1=array(data=rep(0,prod(dim(W1))),dim=dim(W1))
   dW5=array(data=rep(0,prod(dim(W5))),dim=dim(W5))
     dWo=array(data=rep(0,prod(dim(Wo))),dim=dim(Wo))
```

MNISTConv.m

```
%mini-batch loop
   begin=blist(batch);
for k=begin:begin+bsize-1
      %forward pass
      x=X(:,:,k);
      y1=Conv(x,W1);
      y2=ReLU(y1);
      y3=Pool(y2);
      y4=reshape(y3,[],1);
      v5=W5*y4;
      y5=ReLU(v5);
      v=Wo*y5;
      y=Softmax(v);

      %one hot encoding
      d=zeros(10,1);
      d(sub2ind(size(d),D(k),1))=1;
```

MNISTConv.r

```
begin=blist[batch]
##one minibatch loop
   for (k in (begin:(begin+bsize-1))){
     ###forward
     x=X[,,k]
     y1=Conv(x,W1)
     y2=ReLU(y1)
     y3=Pool(y2)
     y4=matrix(as.vector(y3),length(y3),1)
     v5=W5%*%y4
     y5=ReLU(v5)
     v=Wo%*%y5
     y=Softmax(v)
     ### One hot encoding (set d[4]=1 for D[k]=4))
     d=rep(0,10)
     d[D[k]]=1
```

MNISTConv.m

```
%Backprop
     e=d-y;
     delta=e;

     e5=Wo'*delta;
     delta5=(y5>0).*e5;

     e4=W5'*delta5;
     e3=reshape(e4,size(y3)); %2000x 1 to 10-10-20

     e2=zeros(size(y2)); % 20-20-20
     W3=ones(size(y2))/(2*2); % same weight for pooling layer

     for c=1:20
       e2(:,:,c)=kron(e3(:,:,c),ones([2 2])).*W3(:,:,c); %expand from
10x10 to 20x20
     end

     delta2=(y2 >0).*e2;

     delta_x=zeros(size(W1)); %(same as y2 layer derivation)
```

MNISTConv.r

```
### Backprop
     e=d-y
     delta=e


     e5=t(Wo)%*%delta
     delta5=(y5>0)*e5


     e4=t(W5)%*%delta5
     e3=array(e4,dim=dim(y3))


     e2=array(rep(0,prod(dim(y2))),dim=dim(y2))
     W3=array(rep(1,prod(dim(y2))),dim=dim(y2))/4
     library(pracma)
     for (c in (1:20)){
     e2[,,c]=kron(e3[,,c],matrix(rep(1,4),nrow=2,ncol=2))*W3[,,c]
#expand from 10x10 to 20x20
   }
     delta2=(y2>0)*e2
     delta_x=array(rep(0,prod(dim(W1))),dim=dim(W1))
```

MNISTConv.m

```
%Backprop

for c =1:20
        delta_x(:,:,c)=conv2(x(:,:),rot90(delta2(:,:,c)),'valid');
    end
```

MNISTConv.r

```
### Backprop


 for (c in (1:20)){
      wrow_c=dim(delta2)[1]
      wcol_c=dim(delta2)[2]
      xrow_c=dim(x)[1]
      xcol_c=dim(x)[2]
      yrow_c=xrow_c-wrow_c+1
      ycol_c=xcol_c-wcol_c+1
      conv_c=matrix(rep(0,yrow_c*ycol_c),nrow=yrow_c,ncol=ycol_c)
      filter=delta2[,,c]
      for (k1 in 1:yrow_c){
        for(k2 in 1:ycol_c){
          xij=x[(k1:(k1+wrow_c-1)),(k2:(k2+wcol_c-1))]
          conv_c[k1,k2]=sum(xij*filter)
        }
        }
      delta_x[,,c]=conv_c
      }
```

MNISTConv.m

```
%Weight adumstment
          dW1=dW1+delta_x;
          dW5=dW5+delta5*y4';
          dWo=dWo+delta*y5';
     end
  dW1=dW1/bsize;
  dW5=dW5/bsize;
   dWo=dWo/bsize;


  momentum1=alpha*dW1+beta*momentum1;
  W1=W1+momentum1;
  momentum5=alpha*dW5+beta*momentum5;
  W5=W5+momentum5;
   momentumo=alpha*dWo+beta*momentumo;
   Wo=Wo+momentumo;
end


end
```

MNISTConv.r

```
### Weight adumstment
       #update weight
      dW1=dW1+delta_x
      dW5=dW5+delta5%*%t(matrix(y4))
      dWo=dWo+delta%*%t(y5)
    }
    #end of minibatch
    dW1=dW1/bsize
    dW5=dW5/bsize
    dWo=dWo/bsize
    momentum1=alpha*dW1+beta*momentum1
    W1=W1+momentum1
    momentum5=alpha*dW5+beta*momentum5
    W5=W5+momentum5
    momentumo=alpha*dWo+beta*momentumo
    Wo=Wo+momentumo
  }
#end of epoch
  return(list("W1"=W1,"W5"=W5,"Wo"=Wo))
}
```

Test: TestMNISTconv.m

```
clear all
Images=loadMNISTImages('t10k-images.idx3-ubyte');
Images=reshape(Images,28,28,[]);
Labels=loadMNISTLabels('t10k-labels.idx1-ubyte');
Labels(Labels==0)=10;
W1=1e-2*randn([9 9 20]);
W5=(2*rand(100,2000)-1)*sqrt(6)/sqrt(360+2000);
Wo=(2*rand(10,100)-1)*sqrt(6)/sqrt(10+100);
X=Images(:,:,1:8000);
D=Labels(1:8000);
for epoch=1:3
  epoch
  [W1, W5, Wo]=MnistConv(W1, W5, Wo, X, D);
end
X=Images(:,:,8001:10000);
D=Labels(8001:10000);
acc=0;
```

```
N=length(D);
for k=1:N
x=X(:,:,k);
y1=Conv(x,W1);
y2=ReLU(y1);
y3=Pool(y2);
y4=reshape(y3,[],1);
v5=W5*y4;
y5=ReLU(v5);
v=Wo*y5;
y=Softmax(v);
[~,i]=max(y);
if i==D(k);
acc=acc+1;
end
end
acc=acc/N;
acc
```

TestConv .r

```
print('begin time')
print(Sys.time())

#clear all
rm(list=ls())
library(pracma)
source("Pool.r")
source("Conv.r")
source("ReLU.r")
source("Softmax.r")
source("MnistConv.r")
load('M.test.Rdata')
Images.D=test$x
Images=array(data=rep(0,length(Images.D)),dim=c(28,28,10000))
for (i in 1:10000){
  Images[,,i]=t(matrix(Images.D[i,],nrow=28,ncol=28))/255
}
Labels=test$y
Labels[Labels==0]=10
set.seed(12345)
```

```
W1=1e-2*array(rnorm(9*9*20),dim=c(9,9,20))
W5=(2*matrix(runif(100*2000),nrow=100,ncol=2000)-
1)*sqrt(6)/sqrt(360+2000)
Wo=(2*matrix(runif(10*100), nrow=10,ncol=100)-1)*sqrt(6)/sqrt(10+100)
X=Images[,,1:8000]
D=Labels[1:8000]


for (epoch in (1:3)){
  print("epoch=")
  print(epoch)
  Result=MnistConv(W1, W5, Wo, X, D)
  W1=Result$W1
  W5=Result$W5
  Wo=Result$Wo
}
X=Images[,,8001:10000]
D=Labels[8001:10000]
acc=0
N=length(D)
```

TestConv .r

```
for (k in (1:N)){
x=X[,,k]
y1=Conv(x,W1)
y2=ReLU(y1)
y3=Pool(y2)
y4=matrix(as.vector(y3),length(y3),1)
v5=W5%*%y4
y5=ReLU(v5)
v=Wo%*%y5
y=Softmax(v)
yk=which.max(y)
if (yk==D[k]){
   acc=acc+1
}
}
acc=acc/N
print("acc=")
print(acc)
```

MXNET을 사용하면......

```
require(mxnet)

train <- read.csv('data/train.csv', header=TRUE)
test <- read.csv('data/test.csv', header=TRUE)
train <- data.matrix(train)
test <- data.matrix(test)
train.x <- train[,-1]
train.y <- train[,1]
train.x <- t(train.x/255)
test <- t(test/255)

train.array <- train.x dim(train.array) <- c(28, 28, 1,
ncol(train.x))
test.array <- test dim(test.array) <- c(28, 28, 1, ncol(test))

data <- mx.symbol.Variable('data')

# first conv
conv1 <- mx.symbol.Convolution(data=data, kernel=c(5,5),
num_filter=20)
tanh1 <- mx.symbol.Activation(data=conv1,
act_type="tanh")
pool1 <- mx.symbol.Pooling(data=tanh1, pool_type="max",
kernel=c(2,2), stride=c(2,2))

# second conv
conv2 <- mx.symbol.Convolution(data=pool1, kernel=c(5,5),
num_filter=50)
tanh2 <- mx.symbol.Activation(data=conv2,
act_type="tanh")
pool2 <- mx.symbol.Pooling(data=tanh2, pool_type="max",
kernel=c(2,2), stride=c(2,2))

# first fullc
flatten <- mx.symbol.Flatten(data=pool2)
fc1 <- mx.symbol.FullyConnected(data=flatten,
num_hidden=500)
tanh3 <- mx.symbol.Activation(data=fc1, act_type="tanh")

# second fullc
fc2 <- mx.symbol.FullyConnected(data=tanh3,
num_hidden=10)

# loss
lenet <- mx.symbol.SoftmaxOutput(data=fc2)

devices <- mx.cpu()

mx.set.seed(0)
tic <- proc.time()
model <- mx.model.FeedForward.create(lenet,
X=train.array, y=train.y, ctx=device.gpu, num.round=5,
array.batch.size=100, learning.rate=0.05,
momentum=0.9, wd=0.00001,
eval.metric=mx.metric.accuracy,
epoch.end.callback=mx.callback.log.train.metric(100))

preds <- predict(model, test.array)
```