7

Main Technics for CNN

7.1 Batch Normalization

7.2 Forward Propagation Batch Normalization

7.3 Softmax Classifier

7.4 Dropout

7.5 Convolution

7.6 Pooling

7.7 Architecture Analysis

7

Main Technics for CNN

7.8 Deconvolution

7.9 YOLO based Detection

7.10 Backpropagation in CNN

7.1 Batch Normalization (BN)

Vanishing/exploding gradient problems:

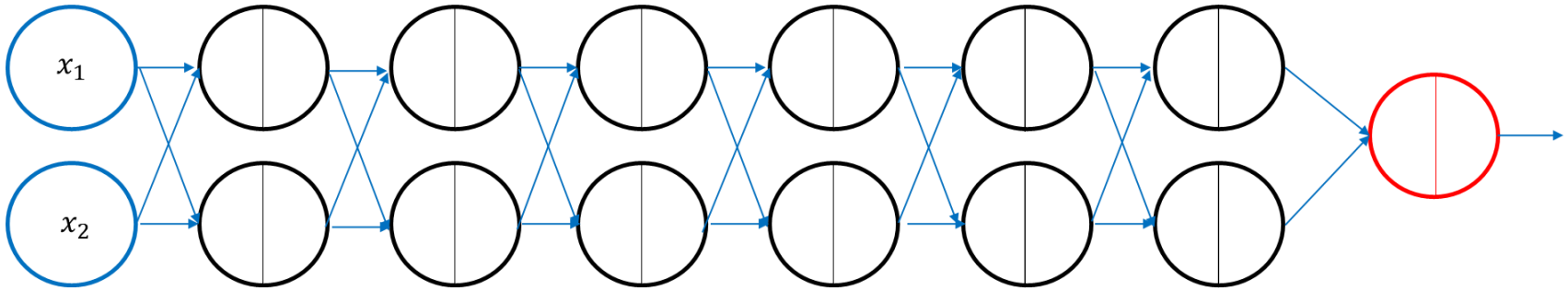


Fig. 7.1 A deep neural network model

From

$\delta_j^{(L)} = (z_j^{(L)} - y^{(j)})$ for the output layer

$\delta_j^{(l)} = \sigma'(z_j^{(l)}) \sum_{i=1}^{S(l+1)} (\theta_{ji}^{(l)})^T \delta_i^{(l+1)}$ for the hidden layer: $l \in \{2, 3, 4, \ldots, L-1\}$

$$\delta_j^{(8)} = (a_j^{(8)} - y^{(j)}) = (z_j^{(8)} - y^{(j)})$$

$$\delta_j^{(7)} = \sigma'(z_j^{(7)}) \sum_{i=1}^{2} (\theta_{ji}^{(7)})^T (z_i^{(8)} - y^{(i)})$$

$$\delta_j^{(6)} = \sigma'(z_j^{(6)}) \sum_{i=1}^{2} (\theta_{ji}^{(6)})^T \sigma'(z_i^{(7)}) \sum_{i=1}^{2} (\theta_{ii}^{(7)})^T (z_i^{(8)} - y^{(i)})$$

$$\delta_j^{(5)} = \sigma'(z_j^{(5)}) \sum_{i=1}^{2} (\theta_{ji}^{(5)})^T \sigma'(z_i^{(6)}) \sum_{i=1}^{2} (\theta_{ii}^{(6)})^T \sigma'(z_i^{(7)}) \sum_{i=1}^{2} (\theta_{ii}^{(7)})^T (z_i^{(8)} - y^{(i)})$$

$$\delta_j^{(4)} = \sigma'(z_j^{(4)}) \sum_{i=1}^{2} (\theta_{ji}^{(4)})^T \sigma'(z_i^{(5)}) \sum_{i=1}^{2} (\theta_{ii}^{(5)})^T \sigma'(z_i^{(6)}) \sum_{i=1}^{2} (\theta_{ii}^{(6)})^T \sigma'(z_i^{(7)}) \sum_{i=1}^{2} (\theta_{ii}^{(7)})^T (z_i^{(8)} - y^{(i)})$$

$$\delta_j^{(3)} = \sigma'(z_j^{(3)}) \sum_{i=1}^{2} (\theta_{ji}^{(3)})^T \sigma'(z_i^{(4)}) \sum_{i=1}^{2} (\theta_i^{(4)})^T \sigma'(z_i^{(5)}) \sum_{i=1}^{2} (\theta_{ii}^{(5)})^T \sigma'(z_i^{(6)}) \sum_{i=1}^{2} (\theta_{ii}^{(6)})^T \sigma'(z_i^{(7)}) \sum_{i=1}^{2} (\theta_{ii}^{(7)})^T (z_i^{(8)} - y^{(i)})$$

$$\delta_j^{(2)} = \boxed{\sigma'(z_j^{(2)})} \sum_{i=1}^{2} (\theta_{ji}^{(2)})^T \boxed{\sigma'(z_i^{(3)})} \sum_{i=1}^{2} (\theta_{ii}^{(3)})^T \boxed{\sigma'(z_i^{(4)})} \sum_{i=1}^{2} (\theta_i^{(4)})^T \boxed{\sigma'(z_i^{(5)})} \sum_{i=1}^{2} (\theta_{ii}^{(5)})^T \boxed{\sigma'(z_i^{(6)})} \sum_{i=1}^{2} (\theta_{ii}^{(6)})^T \boxed{\sigma'(z_i^{(7)})} \sum_{i=1}^{2} (\theta_{ii}^{(7)})^T (z_i^{(8)} - y^{(i)})$$

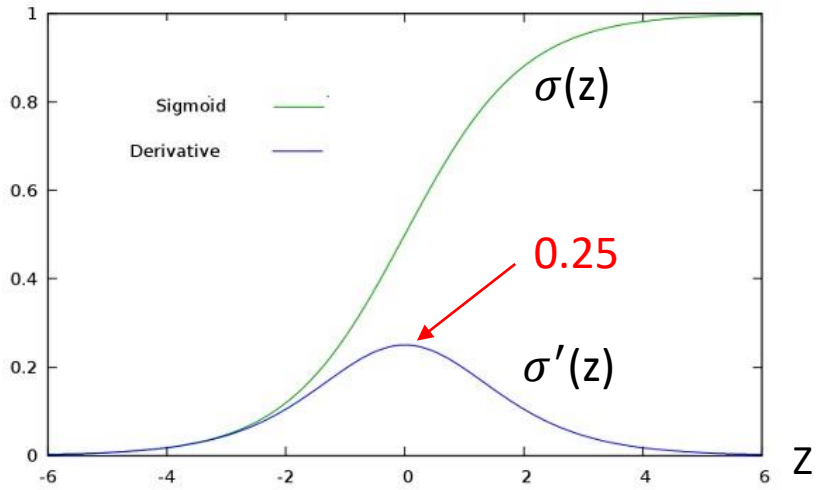


Fig. 7.2 A sigmoid function and its1st derivative

$$\frac{\partial J(\boldsymbol{\Theta})}{\partial \theta_{ji}^{(l)}} = \delta_j^{(l+1)} a_i^{(l)}$$

(1) Vanishing gradient problem occurs when the value of ‘z’s becomes huge or very small (i.e. σ' gets almost ‘0’) and the weights including biases are very small.

(2) Exploding gradient problem occurs when the value of ‘z’s gets close to ’0’ (i.e. σ' gets almost the Max. 0.25) and the weights including biases are huge.

How to solve the vanishing or exploding problems?

(1) Don’t use sigmoid function for hidden layers. Leaky ReLu or ReLu activations are recommended.

(2) You can use gradient clipping to solve gradient exploding for RNN.

(3) Some weight initialization can relieve both problems: Xavier Initialization, He initialization, etc.

(4) Batch normalization prevents both problems by normalizing inputs.

Advantage of BN	Disadvantage of BN
AI system can be less sensitive to initialization and gradient vanishing problem even when using sigmoid or tanh activations.	It makes AI system become more complicated and computation is more expensive.
BN makes GD work well even for the case of large learning rate and learning speed can be improved.	BN depends on batch size a lot and too small size of batch is not recommended.
BN can suppress the possibility of overfitting. The usage of BN with dropout is recommended.	An application of BN to RNN is hard.

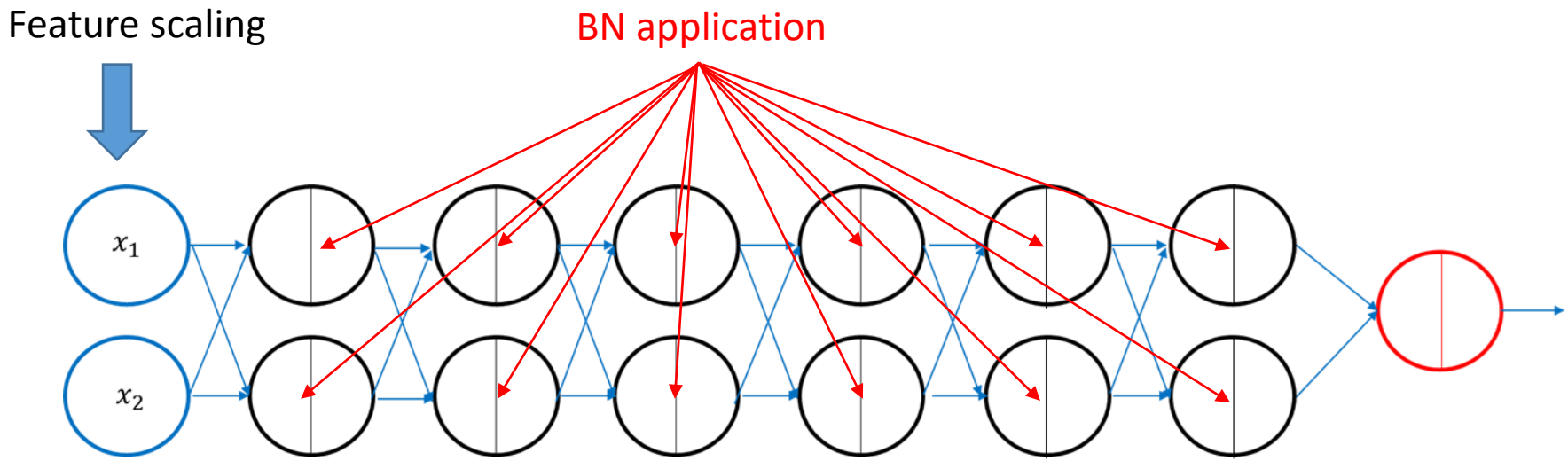


Fig. 7.3 An application of BN to hidden layers

7.2 Forward Propagation with Batch Normalization

- Mini-batch mean: $\mu_j^{(l)} = \frac{1}{m} \sum_{M=1}^{m} z_{j,M}^{(l)}$
- Mini-batch variation: $v_j^{(l)} = \frac{1}{m} \sum_{M=1}^{m} (z_{j,M}^{(l)} - \mu_j^{(l)})^2$
- Mean Normalization: $\hat{z}_{j,M}^{(l)} = \frac{z_{j,M}^{(l)} - \mu_j^{(l)}}{\sqrt{v_j^{(l)} + \epsilon}}$
- Batch Normalization: $\tilde{z}_{j,M}^{(l)} = \gamma\, \hat{z}_{j,M}^{(l)} + \beta$
- $\boldsymbol{A}_M^{(l)} = \sigma\left(\widetilde{\boldsymbol{Z}}_M^{(l)}\right)$, here the subscript 'M' = M-th example number and 'b' = batch number. ϵ = very small value (ex: 10^{-5}) to prevent that the denominator becomes '0'. γ = scaling parameter. β = shifting parameter. σ = sigmoid function.

BN enables to use sigmoid or tanh activations for hidden layers.

Compute $\widetilde{\boldsymbol{Z}}_M^{(l)}$ and $\boldsymbol{A}_M^{(l)}$ for each mini-batch in a given dataset. Repeat the same process (forward propagation) the batch number-times during one epoch.

Example: suppose that there are 4 classes, horse (m=1), dog (m=2), cat (m=3) and monkey (m=4). Every single picture has the dimension, $1 \times 2 \times 3 = 6$ (pixels). Implement a forward propagation with patch normalization. Here $\epsilon = 0.00001, \gamma$ = 0.1 and β = 0.00005

horse (m=1)

dog (m=2)

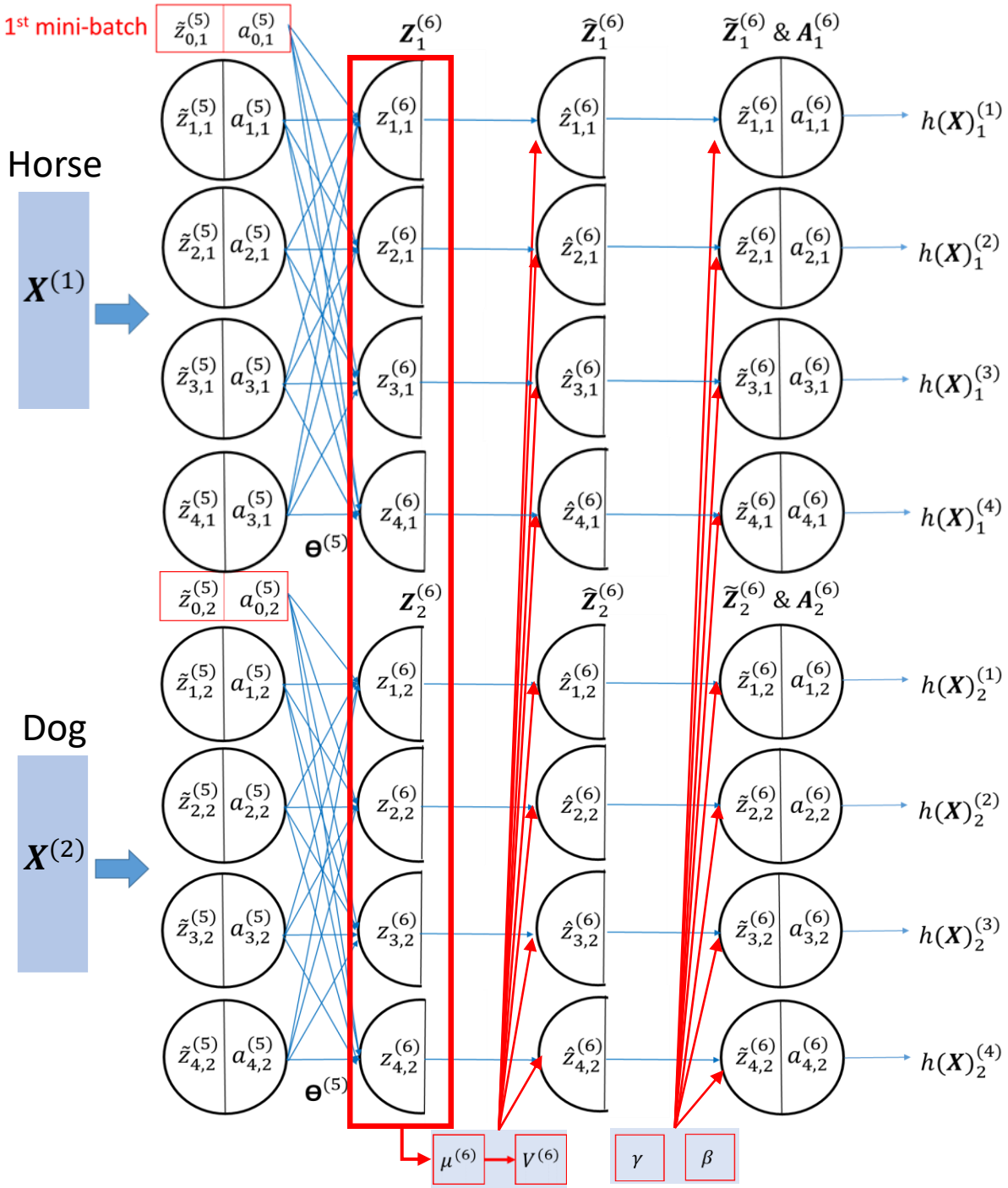
cat (m=3)

monkey (m=4)

Table 7.1 A input display with two mini-batches

M	X_1	X_2	X_3	X_4	X_5	X_6	Y
1	$x_1^{(1)}$	$x_2^{(1)}$	$x_3^{(1)}$	$x_4^{(1)}$	$x_5^{(1)}$	$x_6^{(1)}$	$Y_1^{(1)}=\begin{Bmatrix}1\\0\\0\\0\end{Bmatrix}$
2	$x_1^{(2)}$	$x_2^{(2)}$	$x_3^{(2)}$	$x_4^{(2)}$	$x_5^{(2)}$	$x_6^{(2)}$	$Y_2^{(2)}=\begin{Bmatrix}0\\1\\0\\0\end{Bmatrix}$ 1st mini-batch
3	$x_1^{(3)}$	$x_2^{(3)}$	$x_3^{(3)}$	$x_4^{(3)}$	$x_5^{(3)}$	$x_6^{(3)}$	$Y_3^{(3)}=\begin{Bmatrix}0\\0\\1\\0\end{Bmatrix}$
4	$x_1^{(3)}$	$x_2^{(3)}$	$x_3^{(3)}$	$x_4^{(3)}$	$x_5^{(3)}$	$x_6^{(3)}$	$Y_4^{(4)}=\begin{Bmatrix}0\\0\\0\\1\end{Bmatrix}$ 2nd mini-batch

Here, there are 2 mini-batches and $\boldsymbol{X}^{(M)}=\{x_1^{(M)}, x_2^{(M)}, x_3^{(M)}, x_4^{(M)}, x_5^{(M)}, x_6^{(M)}\}$: every single mini-batch has '2×6= 12' number of features (= batch size is 12).



$$\begin{bmatrix} 0.01 & -0.05 & 0.1 & 0.05 \\ 0.7 & 0.2 & 0.05 & 0.16 \\ 0.0 & -0.45 & -0.2 & 0.03 \\ 0.02 & 0.01 & 0.14 & 0.01 \end{bmatrix} \begin{Bmatrix} -15 \\ 22 \\ -44 \\ 56 \end{Bmatrix} + \begin{Bmatrix} 0.0 \\ 0.2 \\ -0.3 \\ 0.1 \end{Bmatrix} = \begin{Bmatrix} -2.85 \\ 0.86 \\ 0.28 \\ -5.58 \end{Bmatrix}$$

$\Theta^{(5)}$ $\quad A_1^{(5)}$ $\quad B_1^{(5)}$ $\quad Z_1^{(6)}$

$$\begin{bmatrix} 0.01 & -0.05 & 0.1 & 0.05 \\ 0.7 & 0.2 & 0.05 & 0.16 \\ 0.0 & -0.45 & -0.2 & 0.03 \\ 0.02 & 0.01 & 0.14 & 0.01 \end{bmatrix} \begin{Bmatrix} -13 \\ 20 \\ -34 \\ 50 \end{Bmatrix} + \begin{Bmatrix} 0.0 \\ 0.2 \\ -0.3 \\ 0.1 \end{Bmatrix} = \begin{Bmatrix} -2.03 \\ 1.4 \\ -1.0 \\ -4.22 \end{Bmatrix}$$

$\Theta^{(5)}$ $\quad A_2^{(5)}$ $\quad B_2^{(5)}$ $\quad Z_2^{(6)}$

Z vectors with no application of BN

Fig. 7.4 An application of BN to the hidden layer, l = 6.

Mini-batch mean

- $\mu_j^{(6)} = \frac{1}{m} \sum_{M=1}^{m} z_{j,M}^{(6)}$: $\mu_1^{(6)} = \frac{1}{2}(z_{1,1}^{(6)} + z_{1,2}^{(6)}) = \frac{1}{2}$ (-2.85 – 2.03) = -2.44,

$\mu_2^{(6)} = \frac{1}{2}(z_{2,1}^{(6)} + z_{2,2}^{(6)}) = \frac{1}{2}$ (0.86 + 1.40) = 1.13,

$\mu_3^{(6)} = \frac{1}{2}(z_{3,1}^{(6)} + z_{3,2}^{(6)}) = \frac{1}{2}$ (0.28 – 1.00) = -0.36

$\mu_4^{(6)} = \frac{1}{2}(z_{4,1}^{(6)} + z_{4,2}^{(6)}) = \frac{1}{2}$ (-5.58 – 4.22) = -4.90

Mini-batch variation

- $v_j^{(6)} = \frac{1}{m} \sum_{M=1}^{m} (z_{j,M}^{(6)} - \mu_j^{(6)})^2$: $v_1^{(6)} = \frac{1}{2} \{(z_{1,1}^{(6)} - \mu_1^{(6)})^2 + (z_{1,2}^{(6)} - \mu_1^{(6)})^2\} = \frac{1}{2} \{(-2.85 + 2.44)^2 + (-2.03 + 2.44)^2\} = 0.1681$

$v_2^{(6)} = \frac{1}{2} \{(z_{2,1}^{(6)} - \mu_2^{(6)})^2 + (z_{2,2}^{(6)} - \mu_2^{(6)})^2\} = \frac{1}{2} \{(0.86 - 1.13)^2 + (1.40 - 1.13)^2\} = 0.0729$

$v_3^{(6)} = \frac{1}{2} \{(z_{3,1}^{(6)} - \mu_3^{(6)})^2 + (z_{3,2}^{(6)} - \mu_3^{(6)})^2\} = \frac{1}{2} \{(0.28 + 0.36)^2 + (-1.00 + 0.36)^2\} = 0.4096$

$v_4^{(6)} = \frac{1}{2} \{(z_{4,1}^{(6)} - \mu_4^{(6)})^2 + (z_{4,2}^{(6)} - \mu_4^{(6)})^2\} = \frac{1}{2} \{(-5.58 + 4.90)^2 + (-4.22 + 4.90)^2\} = 0.4624$

Mean Normalization

- $\hat{Z}_{j,M}^{(6)} = \frac{z_{j,M}^{(6)} - \mu_j^{(6)}}{\sqrt{v_j^{(6)} + \epsilon}}$:

$\hat{Z}_{1,1}^{(6)} = \frac{z_{1,1}^{(6)} - \mu_1^{(6)}}{\sqrt{v_1^{(6)} + \epsilon}} = \frac{-2.85 - (-2.44)}{\sqrt{0.1681 + 0.00001}}$ =-0.99997, $\hat{Z}_{2,1}^{(6)} = \frac{z_{2,1}^{(6)} - \mu_2^{(6)}}{\sqrt{v_2^{(6)} + \epsilon}} = \frac{0.86 - 1.13}{\sqrt{0.0729 + 0.00001}}$ =-0.999931,

$\hat{Z}_{3,1}^{(6)} = \frac{z_{3,1}^{(6)} - \mu_3^{(6)}}{\sqrt{v_3^{(6)} + \epsilon}} = \frac{0.28 - (-0.36)}{\sqrt{0.4096 + 0.00001}}$ = 0.999988, $\hat{Z}_{4,1}^{(6)} = \frac{z_{4,1}^{(6)} - \mu_4^{(6)}}{\sqrt{v_4^{(6)} + \epsilon}} = \frac{-5.58 - (-4.90)}{\sqrt{0.4624 + 0.00001}}$ =-0.999989

$\hat{Z}_{1,2}^{(6)} = \frac{z_{1,2}^{(6)} - \mu_1^{(6)}}{\sqrt{v_1^{(6)} + \epsilon}} = \frac{-2.03 - (-2.44)}{\sqrt{0.1681 + 0.00001}}$ =0.99997, $\hat{Z}_{2,2}^{(6)} = \frac{z_{2,2}^{(6)} - \mu_2^{(6)}}{\sqrt{v_2^{(6)} + \epsilon}} = \frac{1.40 - 1.13}{\sqrt{0.0729 + 0.00001}}$ =0.999931,

$\hat{Z}_{3,2}^{(6)} = \frac{z_{3,2}^{(6)} - \mu_3^{(6)}}{\sqrt{v_3^{(6)} + \epsilon}} = \frac{-1.00 - (-0.36)}{\sqrt{0.4096 + 0.00001}}$ =-0.999988, $\hat{Z}_{4,2}^{(6)} = \frac{z_{4,2}^{(6)} - \mu_4^{(6)}}{\sqrt{v_4^{(6)} + \epsilon}} = \frac{-4.22 - (-4.90)}{\sqrt{0.4624 + 0.00001}}$ =0.999989

Batch Normalization

- $\tilde{z}_{j,M}^{(6)} = \gamma\, \hat{z}_{j,M}^{(6)} + \beta$:

$\tilde{z}_{1,1}^{(6)} = \gamma\, \hat{z}_{1,1}^{(6)} + \beta = 0.1 \cdot (-0.99997) + 0.00005 = -0.099947$

$\tilde{z}_{2,1}^{(6)} = \gamma\, \hat{z}_{2,1}^{(6)} + \beta = 0.1 \cdot (-0.999931) + 0.00005 = -0.0999431$

$\tilde{z}_{3,1}^{(6)} = \gamma\, \hat{z}_{3,1}^{(6)} + \beta = 0.1 \cdot (0.999988) + 0.00005 = 0.100049$

$\tilde{z}_{4,1}^{(6)} = \gamma\, \hat{z}_{4,1}^{(6)} + \beta = 0.1 \cdot (-0.999989) + 0.00005 = -0.0999489$

$\tilde{z}_{1,2}^{(6)} = \gamma\, \hat{z}_{1,2}^{(6)} + \beta = 0.1 \cdot (0.99997) + 0.00005 = 0.100047$

$\tilde{z}_{2,2}^{(6)} = \gamma\, \hat{z}_{2,2}^{(6)} + \beta = 0.1 \cdot (0.999931) + 0.00005 = 0.100043$

$\tilde{z}_{3,2}^{(6)} = \gamma\, \hat{z}_{3,2}^{(6)} + \beta = 0.1 \cdot (-0.999988) + 0.00005 = -0.0999488$

$\tilde{z}_{4,2}^{(6)} = \gamma\, \hat{z}_{4,2}^{(6)} + \beta = 0.1 \cdot (0.999989) + 0.00005 = 0.100049$

$\boldsymbol{A}_M^{(6)} = \sigma\left(\widetilde{\boldsymbol{Z}}_M^{(6)}\right)$:

$$\boldsymbol{A}_1^{(6)} = \sigma\left(\widetilde{\boldsymbol{Z}}_1^{(6)}\right) = \begin{Bmatrix} a_{1,1}^{(6)} \\ a_{2,1}^{(6)} \\ a_{3,1}^{(6)} \\ a_{4,1}^{(6)} \end{Bmatrix} = \begin{Bmatrix} h(\boldsymbol{X})_1^{(1)} \\ h(\boldsymbol{X})_1^{(2)} \\ h(\boldsymbol{X})_1^{(3)} \\ h(\boldsymbol{X})_1^{(4)} \end{Bmatrix} = \sigma\left(\begin{Bmatrix} -0.099947 \\ -0.0999431 \\ 0.1000490 \\ -0.0999489 \end{Bmatrix}\right) = \begin{Bmatrix} 0.475034 \\ 0.475035 \\ 0.524991 \\ 0.475034 \end{Bmatrix}$$

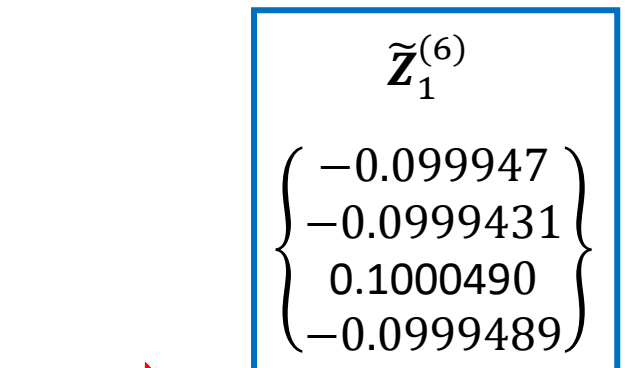
$$\boldsymbol{A}_2^{(6)} = \sigma\left(\widetilde{\boldsymbol{Z}}_2^{(6)}\right) = \begin{Bmatrix} a_{1,2}^{(6)} \\ a_{2,2}^{(6)} \\ a_{3,2}^{(6)} \\ a_{4,2}^{(6)} \end{Bmatrix} = \begin{Bmatrix} h(\boldsymbol{X})_2^{(1)} \\ h(\boldsymbol{X})_2^{(2)} \\ h(\boldsymbol{X})_2^{(3)} \\ h(\boldsymbol{X})_2^{(4)} \end{Bmatrix} = \sigma\left(\begin{Bmatrix} 0.100047 \\ 0.100043 \\ -0.0999488 \\ 0.100049 \end{Bmatrix}\right) = \begin{Bmatrix} 0.524991 \\ 0.524990 \\ 0.475034 \\ 0.524991 \end{Bmatrix}$$

$$z_1^{(6)}$$

$$\begin{Bmatrix} -2.85 \\ 0.86 \\ 0.28 \\ -5.58 \end{Bmatrix}$$

$$z_2^{(6)}$$

$$\begin{Bmatrix} -2.03 \\ 1.4 \\ -1.0 \\ -4.22 \end{Bmatrix}$$

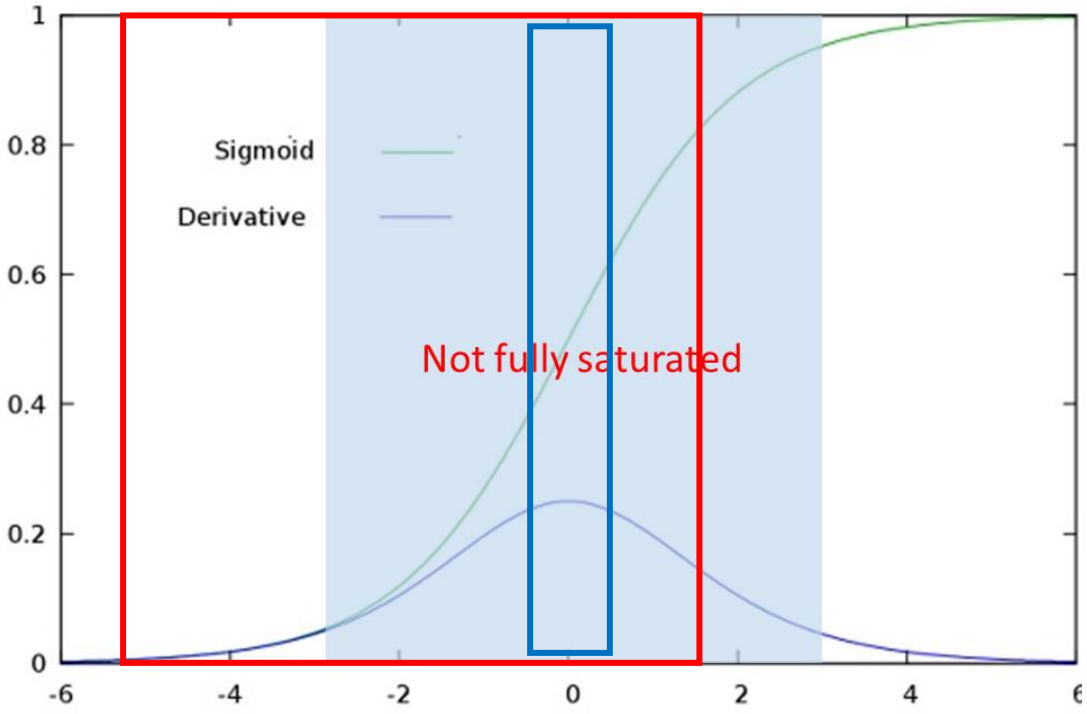


$$\tilde{z}_1^{(6)}$$

$$\begin{Bmatrix} -0.099947 \\ -0.0999431 \\ 0.1000490 \\ -0.0999489 \end{Bmatrix}$$

$$\tilde{z}_2^{(6)}$$

$$\begin{Bmatrix} 0.100047 \\ 0.100043 \\ -0.0999488 \\ 0.100049 \end{Bmatrix}$$



$\tilde{z}^{(6)}$or $z^{(6)}$

Fig. 7.5 A sigmoid and its derivative fo $\tilde{z}^{(l)}$or $z^{(6)}$

Here $\tilde{z}^{(l)}$ is the batch normalized of '$z^{(l)}$,

7.3 Softmax Classifier

- Softmax classifier includes Softmax activation function and Cross-Entropy Loss (Logistic Regress Loss).
- Softmax, also known as normalized exponential function, is a function that takes input as a vector of real numbers, and normalizes it into a probability distribution consisting of probabilities proportional to the exponentials of the input numbers. That is, prior to applying softmax, some vector components could be negative, or greater than one and might not sum to 1 but after applying softmax, each component will be in the interval [0,1], and the components will add up to 1, so that they can be interpreted as probabilities.

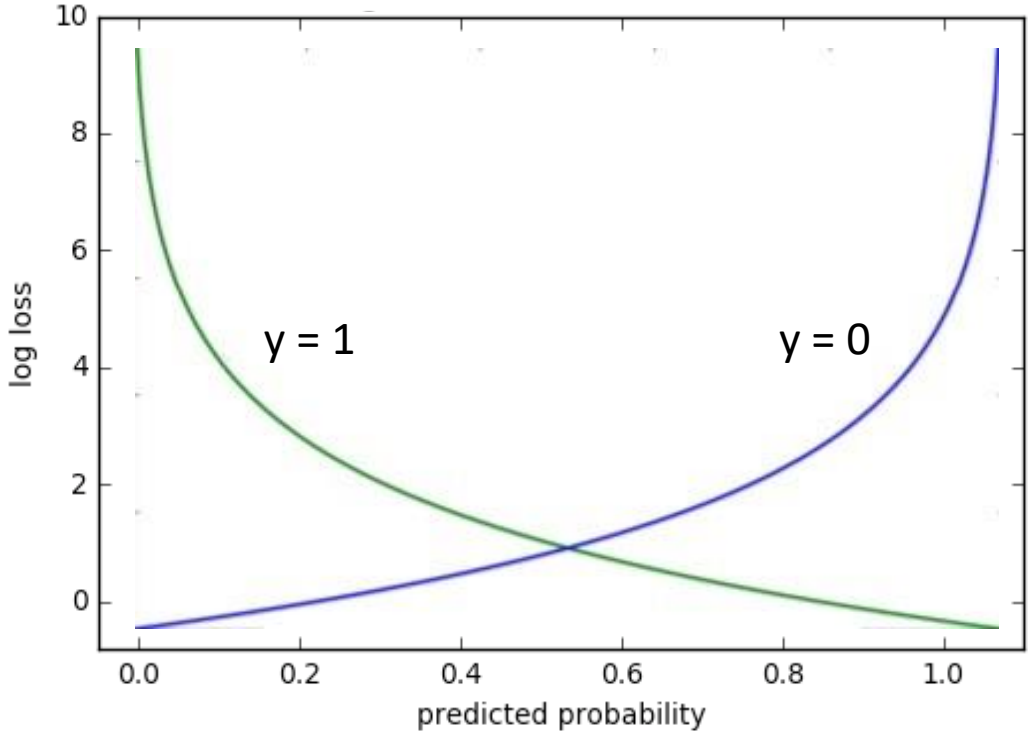


Fig 7.6 Cross-entropy loss

- Cross-Entropy Loss (Logistic Regression Loss)

  measures the performance of a classification model whose output is a probability between 0 and 1. The loss increases as the predicted probability diverges from the actual label. For example, suppose the predicted probability is vey close to 0 (that is, the neural model fails to give exact predicting probability) when the actual label is 1.0. This case would be bad and the result in a loss value is high while the predicting probability of close to 1.0 gives small loss (see Fig. 7.4)

Softmax Function:

The standard Softmax function, g is defined by the formula

$$a_j^{(l)} = g(z_j^{(l)}) = \frac{exp^{(z_j^{(l)})}}{\sum_{i=1}^{S(l)} exp^{(z_i^{(l)})}} \text{ for j = 1, 2, 3 ……, } S(l) \text{ and } \boldsymbol{Z}^{(l)} = \{z_1^{(l)}, z_2^{(l)}, z_3^{(l)} …., z_{S(l)}^{(l)}\} \in \boldsymbol{R}^{S(l)}$$

The Softmax function, g after applying BN for ‘b’-th mini-batch can be

$$a_{j,M}^{(l)} = g(\tilde{z}_{j,M}^{(l)}) = \frac{exp^{(\tilde{z}_{j,M}^{(l)})}}{\sum_{i=1}^{S(l)} exp^{(\tilde{z}_{i,M}^{(l)})}} \text{ for j = 1, 2, 3 ……, } S(l) \text{ and } \widetilde{\boldsymbol{Z}}^{(l)} = \{\tilde{z}_{1,M}^{(l)}, \tilde{z}_{2,M}^{(l)}, \tilde{z}_{3,M}^{(l)} …., \tilde{z}_{S(l),M}^{(l)}\} \in \boldsymbol{R}^{S(l)}$$

Example: Compute $g(\tilde{z}_{j,M}^{(l)})$ to get the predicted probability for each class (horse, dog, cat and monkey)

$\tilde{z}_{j,M}^{(6)} = \gamma\, \hat{z}_{j,M}^{(6)} + \beta$:

$\tilde{z}_{1,1}^{(6)} = \gamma\, \hat{z}_{1,1}^{(6)} + \beta = 0.1 \cdot (-0.99997) + 0.00005 = -0.099947$

$\tilde{z}_{2,1}^{(6)} = \gamma\, \hat{z}_{2,1}^{(6)} + \beta = 0.1 \cdot (-0.999931) + 0.00005 = -0.0999431$

$\tilde{z}_{3,1}^{(6)} = \gamma\, \hat{z}_{3,1}^{(6)} + \beta = 0.1 \cdot (0.999988) + 0.00005 = 0.100049$

$\tilde{z}_{4,1}^{(6)} = \gamma\, \hat{z}_{4,1}^{(6)} + \beta = 0.1 \cdot (-0.999989) + 0.00005 = -0.0999489$

$\tilde{z}_{1,2}^{(6)} = \gamma\, \hat{z}_{1,2}^{(6)} + \beta = 0.1 \cdot (0.99997) + 0.00005 = 0.100047$

$\tilde{z}_{2,2}^{(6)} = \gamma\, \hat{z}_{2,2}^{(6)} + \beta = 0.1 \cdot (0.999931) + 0.00005 = 0.100043$

$\tilde{z}_{3,2}^{(6)} = \gamma\, \hat{z}_{3,2}^{(6)} + \beta = 0.1 \cdot (-0.999988) + 0.00005 = -0.0999488$

$\tilde{z}_{4,2}^{(6)} = \gamma\, \hat{z}_{4,2}^{(6)} + \beta = 0.1 \cdot (0.999989) + 0.00005 = 0.100049$

$$a_{j,M}^{(l)} = g(\tilde{z}_{j,M}^{(l)}) = \frac{exp^{(\tilde{z}_{j,M}^{(l)})}}{\sum_{i=1}^{S(l)} exp^{(\tilde{z}_{i,M}^{(l)})}} :$$

The probability, 28.9% of 'Cat' is the highest.

$$a_{1,1}^{(6)} = \frac{exp^{(\tilde{z}_{1,1}^{(6)})}}{(exp^{(\tilde{z}_{1,1}^{(6)})} + exp^{(\tilde{z}_{2,1}^{(6)})} + exp^{(\tilde{z}_{3,1}^{(6)})} + exp^{(\tilde{z}_{4,1}^{(6)})})} = \frac{exp(-0.099947)}{(exp\,(-0.099947) + exp\,(-0.0999431) + \exp\,(0.100049) + \exp(-0.0999489))} = 0.236888$$ Horse

$$a_{2,1}^{(6)} = \frac{exp^{(\tilde{z}_{2,1}^{(6)})}}{(exp^{(\tilde{z}_{1,1}^{(6)})} + exp^{(\tilde{z}_{2,1}^{(6)})} + exp^{(\tilde{z}_{3,1}^{(6)})} + exp^{(\tilde{z}_{4,1}^{(6)})})} = \frac{exp(-0.0999431)}{(exp\,(-0.099947) + exp\,(-0.0999431) + \exp\,(0.100049) + \exp(-0.0999489))} = 0.236889$$ Dog

$$a_{3,1}^{(6)} = \frac{exp^{(\tilde{z}_{3,1}^{(6)})}}{(exp^{(\tilde{z}_{1,1}^{(6)})} + exp^{(\tilde{z}_{2,1}^{(6)})} + exp^{(\tilde{z}_{3,1}^{(6)})} + exp^{(\tilde{z}_{4,1}^{(6)})})} = \frac{exp(0.100049)}{(exp\,(-0.099947) + exp\,(-0.0999431) + \exp\,(0.100049) + \exp(-0.0999489))} = 0.289335$$ Cat

$$a_{4,1}^{(6)} = \frac{exp^{(\tilde{z}_{4,1}^{(6)})}}{(exp^{(\tilde{z}_{1,1}^{(6)})} + exp^{(\tilde{z}_{2,1}^{(6)})} + exp^{(\tilde{z}_{3,1}^{(6)})} + exp^{(\tilde{z}_{4,1}^{(6)})})} = \frac{exp(-0.0999489)}{(exp\,(-0.099947) + exp\,(-0.0999431) + \exp\,(0.100049) + \exp(-0.0999489))} = 0.236888$$ Monkey

$$a_{1,2}^{(6)} = \frac{exp^{(\tilde{z}_{1,2}^{(6)})}}{(exp^{(\tilde{z}_{1,2}^{(6)})} + exp^{(\tilde{z}_{2,2}^{(6)})} + exp^{(\tilde{z}_{3,2}^{(6)})} + exp^{(\tilde{z}_{4,2}^{(6)})})} = \frac{exp(0.100047)}{(exp\,(0.100047) + exp\,(0.100043) + \exp\,(-0.0999488) + \exp(0.100049))} = 0.261867$$ Horse

$$a_{2,2}^{(6)} = \frac{exp^{(\tilde{z}_{2,2}^{(6)})}}{(exp^{(\tilde{z}_{1,2}^{(6)})} + exp^{(\tilde{z}_{2,2}^{(6)})} + exp^{(\tilde{z}_{3,2}^{(6)})} + exp^{(\tilde{z}_{4,2}^{(6)})})} = \frac{exp(0.100043)}{(exp\,(0.100047) + exp\,(0.100043) + \exp\,(-0.0999488) + \exp(0.100049))} = 0.261866$$ Dog

$$a_{3,2}^{(6)} = \frac{exp^{(\tilde{z}_{3,2}^{(6)})}}{(exp^{(\tilde{z}_{1,2}^{(6)})} + exp^{(\tilde{z}_{2,2}^{(6)})} + exp^{(\tilde{z}_{3,2}^{(6)})} + exp^{(\tilde{z}_{4,2}^{(6)})})} = \frac{exp(-0.0999488)}{(exp\,(0.100047) + exp\,(0.100043) + \exp\,(-0.0999488) + \exp(0.100049))} = 0.214399$$ Cat

$$a_{4,2}^{(6)} = \frac{exp^{(\tilde{z}_{4,2}^{(6)})}}{(exp^{(\tilde{z}_{1,2}^{(6)})} + exp^{(\tilde{z}_{2,2}^{(6)})} + exp^{(\tilde{z}_{3,2}^{(6)})} + exp^{(\tilde{z}_{4,2}^{(6)})})} = \frac{exp(0.100049)}{(exp\,(0.100047) + exp\,(0.100043) + \exp\,(-0.0999488) + \exp(0.100049))} = 0.261868$$ Monkey

The probability, 26.1688% of 'MONKEY' is the highest.

Interpretation:

- For the first case, the highest probability, 28.9% says that the input ($\boldsymbol{X}^{(1)}$='HORSE') is 'CAT', but it is not true. Tunning weights will finally give the highest probability of 'HORSE', and the probability of the others will get lower and lower over tuning iteration.
- For the second case, the probability, 26.1688% says that ($\boldsymbol{X}^{(2)}$= 'DOG') is 'MONKEY' but it is not true either. It must be fixed with a bunch of backpropagations.

Cross-Entropy Loss:

The Cross-Entropy Loss (Logistic Regression Loss) function is

$$J(\boldsymbol{\Theta}) = -\frac{1}{m}\sum_{M=1}^{m}\sum_{j=1}^{S(l)} y_{j,c}^{(M)} \log(h(\boldsymbol{X}^{(M)})_j)$$

$$a_{1,1}^{(6)} = 0.236888$$
$$a_{2,1}^{(6)} = 0.236889$$
$$a_{3,1}^{(6)} = 0.289335$$
$$a_{4,1}^{(6)} = 0.236888$$
$$a_{1,2}^{(6)} = 0.261867$$
$$a_{2,2}^{(6)} = 0.261866$$
$$a_{3,2}^{(6)} = 0.214399$$
$$a_{4,2}^{(6)} = 0.261868$$

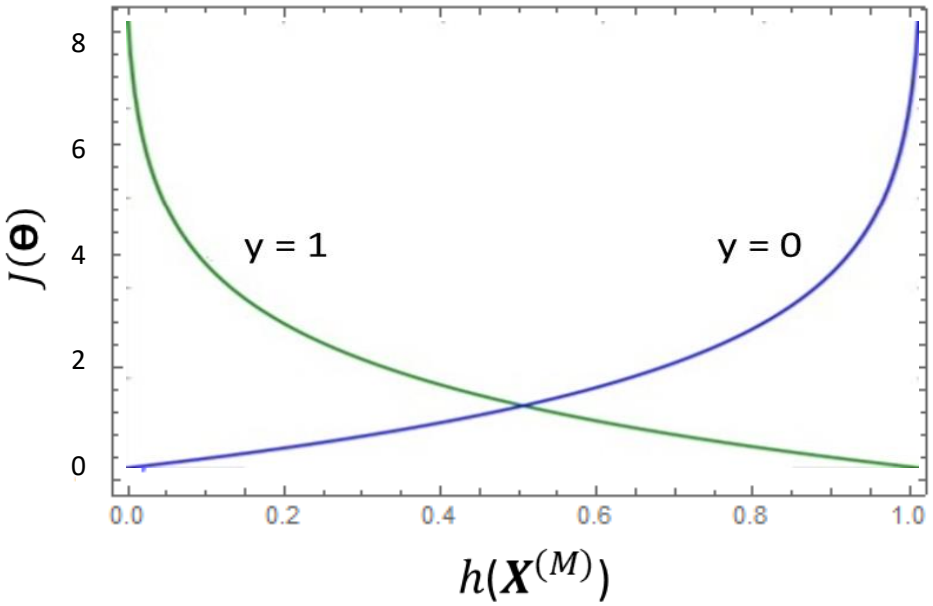


Fig 7.7 Cross-entropy loss

Example: compute the Cross-Entropy Loss for the 1st mini-batch

$$J(\boldsymbol{\Theta}) = -\frac{1}{m}\sum_{M=1}^{m}\sum_{j=1}^{S(l)} y_{j,c}^{(M)} \log(h(\boldsymbol{X}^{(M)})_j)$$

$$= -\frac{1}{2}\sum_{M=1}^{2}\sum_{j=1}^{4} y_{j,c}^{(M)} \log(h(\boldsymbol{X}^{(M)})_j)$$

$$= -\frac{1}{2}\left(y_{1,1}^{(1)}\log(h(\mathbf{X})_1^{(1)}) + y_{2,1}^{(1)}\log(h(\mathbf{X})_2^{(1)}) + y_{3,1}^{(1)}\log(h(\mathbf{X})_3^{(1)}) + y_{4,1}^{(1)}\log(h(\mathbf{X})_4^{(1)}) + y_{1,2}^{(2)}\log(h(\mathbf{X})_1^{(2)}) + y_{2,2}^{(2)}\log(h(\mathbf{X})_2^{(2)}) + y_{3,2}^{(2)}\log\left(h(\mathbf{X})_3^{(2)}\right) + y_{4,2}^{(2)}\log(h(\mathbf{X})_4^{(2)})\right)$$

$$= -\frac{1}{2}\left(1\cdot\log(0.236888) + 0\cdot\log(0.236889) + 0\cdot\log(0.289335) + 0\cdot\log(0.236888)\right) + \left(0\cdot\log(0.261867) + 1\cdot\log(0.261866) + 0\cdot\log(0.214399) + 0\cdot\log(0.261868)\right)$$

$$= 1.39005$$

$log(h(\boldsymbol{X}^{(M)})_j)$ approaches 0 when $h(\boldsymbol{X}^{(M)})_j$ becomes 1.0. That makes the red boxes get 0. It indicates $J(\boldsymbol{\Theta})$ becomes 0, i.e. the cross-entropy loss works well as a cost function .

- Cross-Entropy Loss is also related to and often confused with Logistic Regression Loss, called Log Loss. Although the two measures are derived from a different source, when used as loss functions for classification models, both measures calculate the same quantity and can be used interchangeably.

7.4 Dropout

What is Dropout in Neural Networks?

Dropout refers to ignoring units (i.e. neurons) during the training phase of certain set of neurons which is chosen at random. More technically, at each training stage, individual nodes are either dropped out of the net with probability 1-p or kept retaining with probability p, so that a reduced network (thinned network) is left; incoming and outgoing edges to a dropped-out node are also removed.

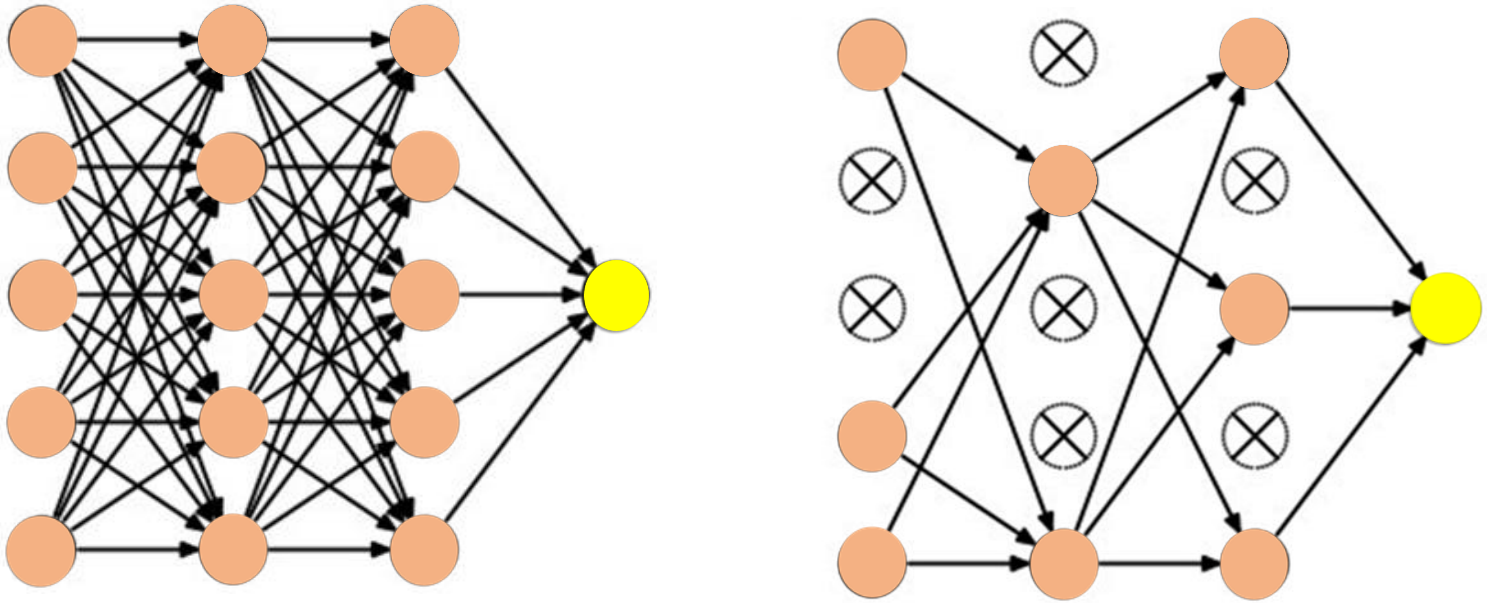


Fig 7.8 A standard neural network (left) and a thinned network with dropout (right)

Dropout Effects?

- It is one of the good methods to prevent ‘Overfitting’.
- It forces a neural network to learn more robust features that are useful in conjunction with many different random models of the other neurons: It can give an effect to combine a lot of thinned neural networks (2^N models can be available when the original neural network has N hidden layers).
- It roughly doubles the number of iterations until convergence. However, training time for each epoch is less.

Dropout Description

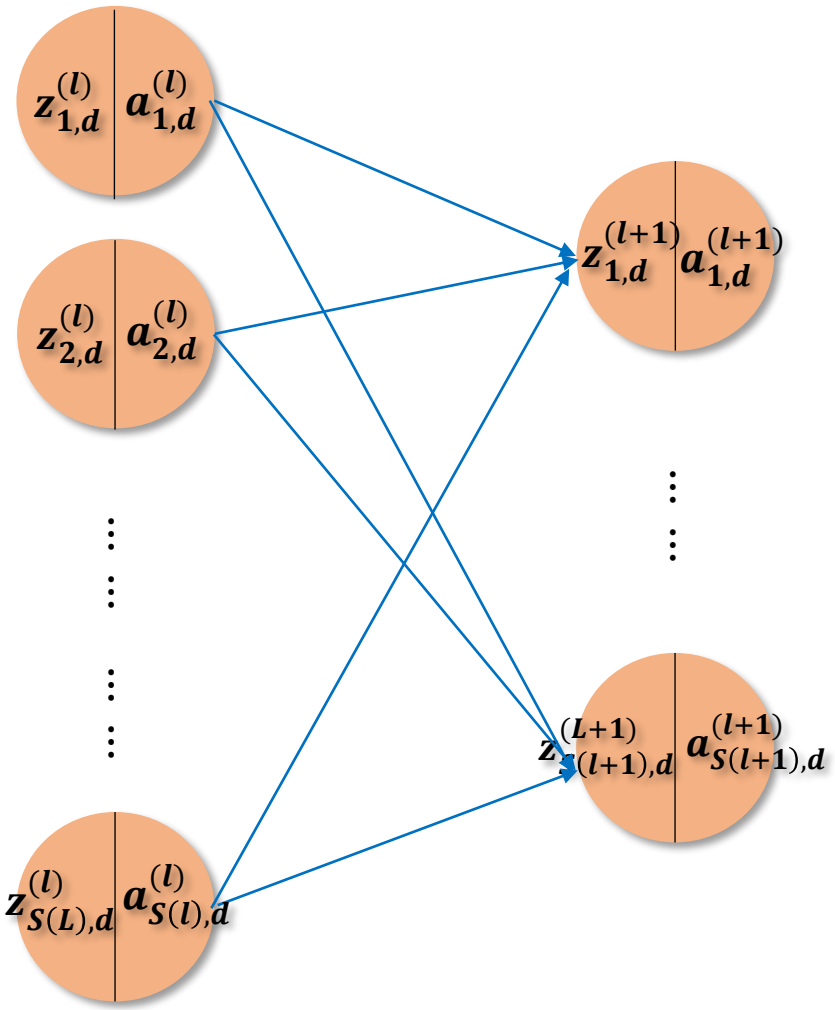


Fig 7.9 Training phase

$$a_{j,d}^{(l)} = r_j \cdot a_j^{(l)} \text{ for } a_j^{(l)} = \sigma(z_j^{(l)})$$

$$z_{j,d}^{(l+1)} = \sum_{i=0}^{S(l)} \theta_{ji}^{(l)} \cdot a_{i,d}^{(l)}$$

$$a_{j,d}^{(l+1)} = \sigma(z_{j,d}^{(l+1)})$$

We make an element-wise multiplication with a mask in which each element is a random variable following the Bernoulli distribution. Here, $r_j \sim Bernoulii(p)$: the random variable, r_j follows Bernoulli distribution of probability. '1-p' is dropout probability (rate) and 'p' is retaining probability (rate). The subscript 'd' means 'Dropout' is applied.

Bernoulli (p)

$$p(r_j) = \begin{cases} p & , r_j = 1 \\ q = 1 - p & , r_j = 0 \end{cases} \quad \text{or} \quad p(r_j) = p^{r_j} \times (1 - p)^{1 - r_j}$$

Ex: the drop-out probability, ‘1-p’ = 0.2 means that two out of ten Bernoulli's trials gives $r_j = 0$.

Dropout Description

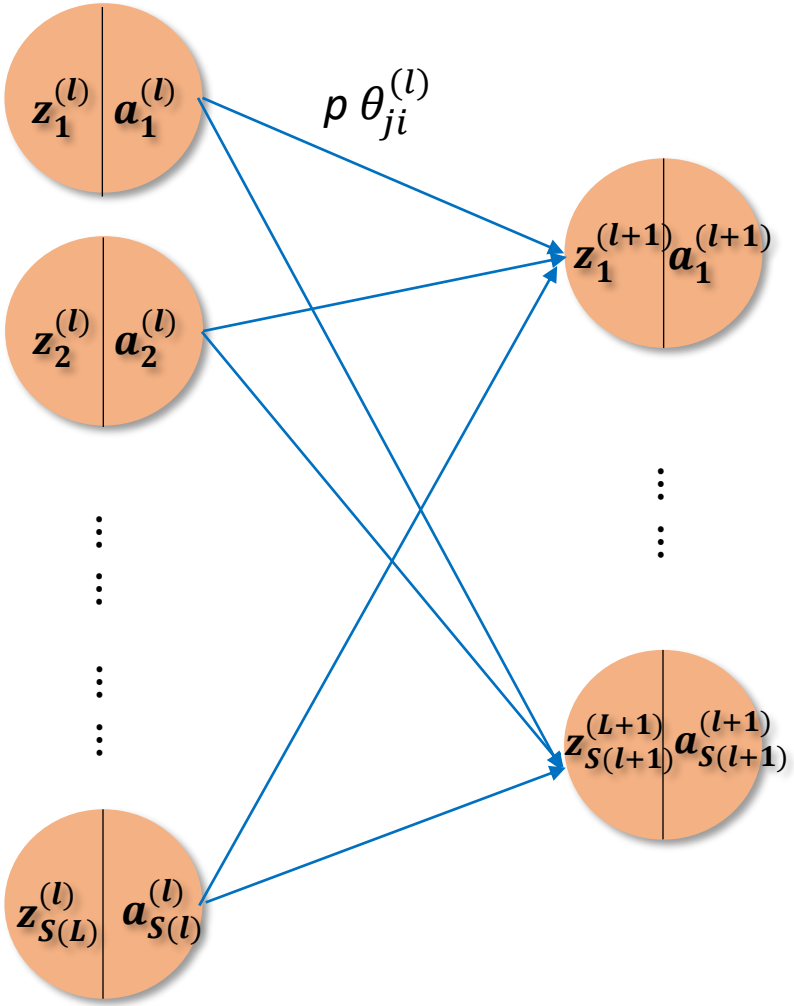


Fig 7.10 Testing phase

$$z_j^{(l+1)} = p\ \sum_{i=0}^{S(l)} (\theta_{ji}^{(l)} \cdot a_i^{(l)})$$

$$a_j^{(l+1)} = \sigma(z_j^{(l+1)})$$

To compensate the revival units for the additional information compared to the training phase,

they are weighted by the multiplication of the retaining probability, ‘p’

Note that:

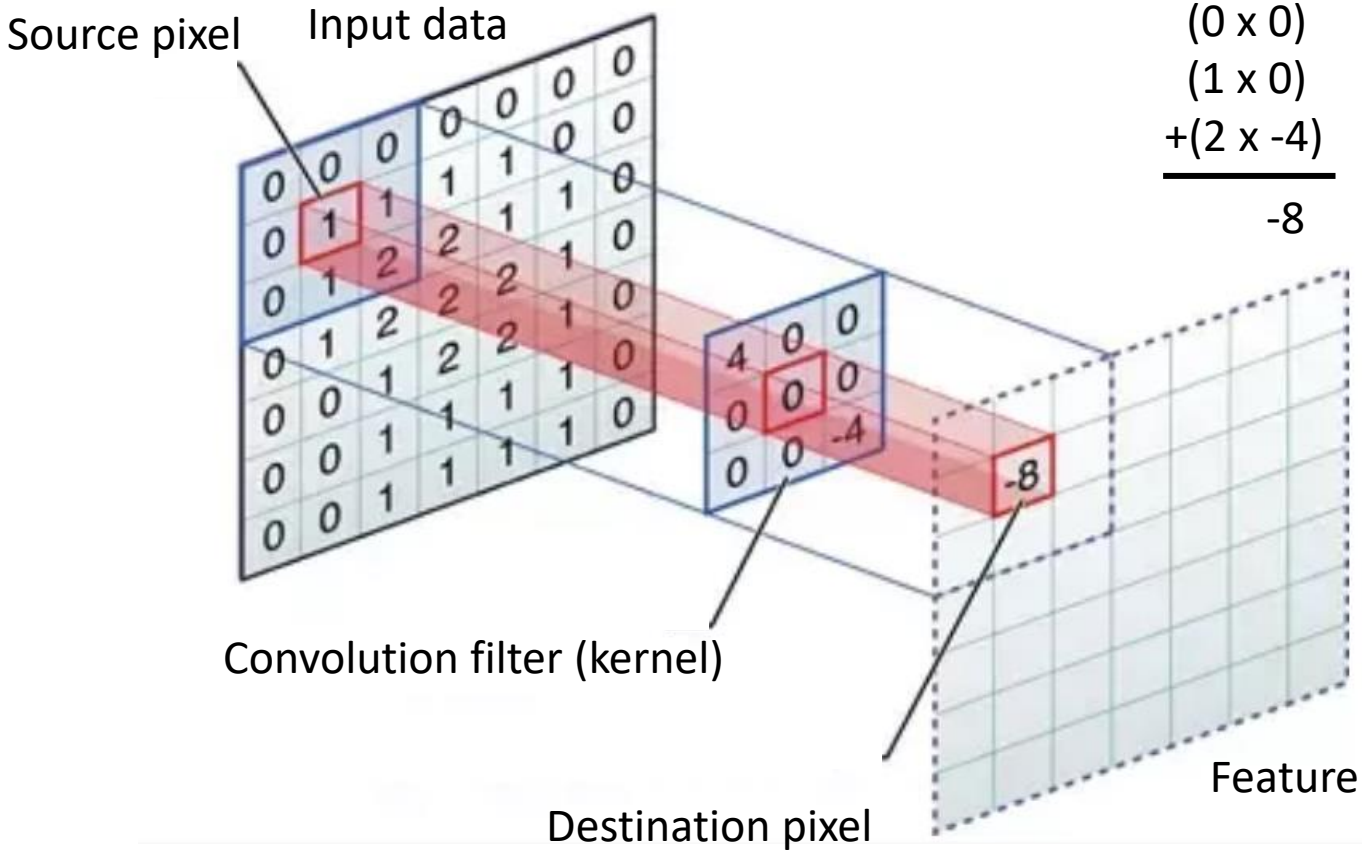
- For each hidden layer for each training sample (but not for an output layer), ignore activations or features randomly only during training iteration (forward and back propagations).
- Different 'Droptout' probabilities, '1-p' can be applied, respectively, to an input layer and hidden layers.
- **All activations MUST BE ACTIVE for testing, but the revival activations should be WEIGHTED by the retaining rate, 'p' (to account for the missing activations during training).**

7.5 Convolution

Fig. 7.11 RGB Input (3 x 5 x 5)

				162	159	163	162	164
				157	159	160	161	163
		165	168	170	170	169	160	162
		162	166	169	170	169	162	163
154	157	159	159	157	168	167	160	162
151	155	158	159	157	160	163		
146	149	153	158	159	163	164		
146	149	153	158	154				
143	143	155	156	145				

What is convolution in CNN?



(0 x 4)
(0 x 0)
(0 x 0)
(0 x 0)
(1 x 0)
(1 x 0)
(0 x 0)
(1 x 0)
+(2 x -4)
-8

Convolution refers to the mathematical combination of two functions to produce a third function. It merges two sets of information. In the case of a CNN, the convolution is performed on the input data (features) with the use of a filter (kernel) to then produce a feature map.

Fig 7.12 2D Convolution process

Toggle Movement

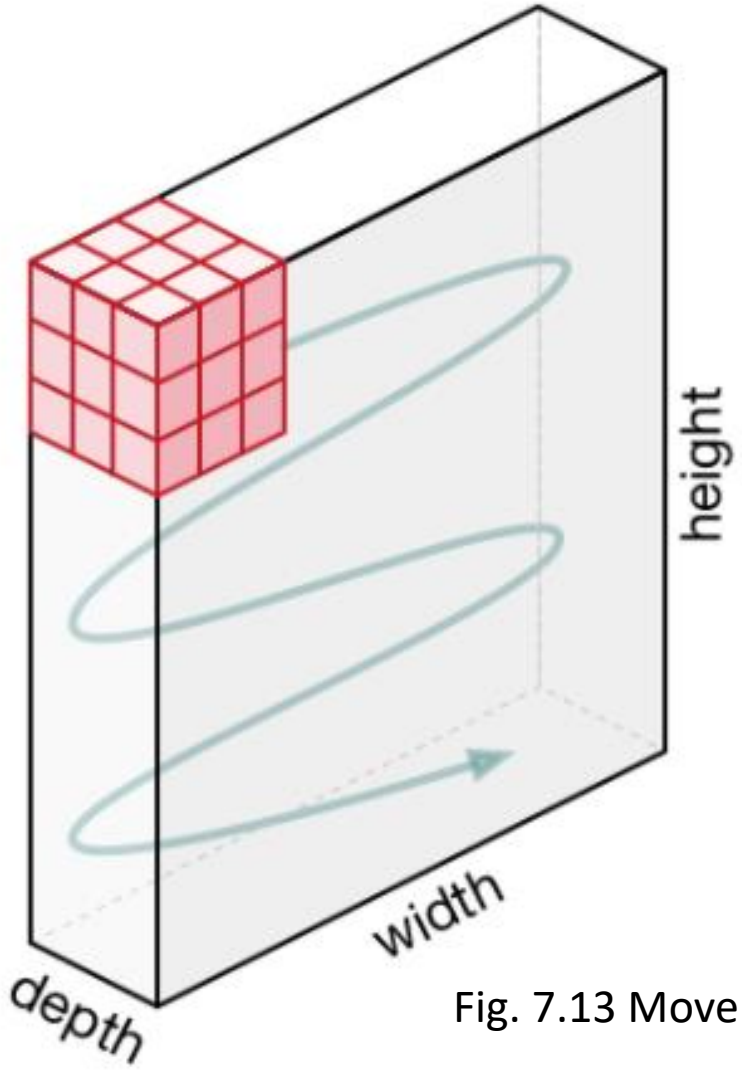


Fig. 7.13 Movement of a 3D filter on a 3D input

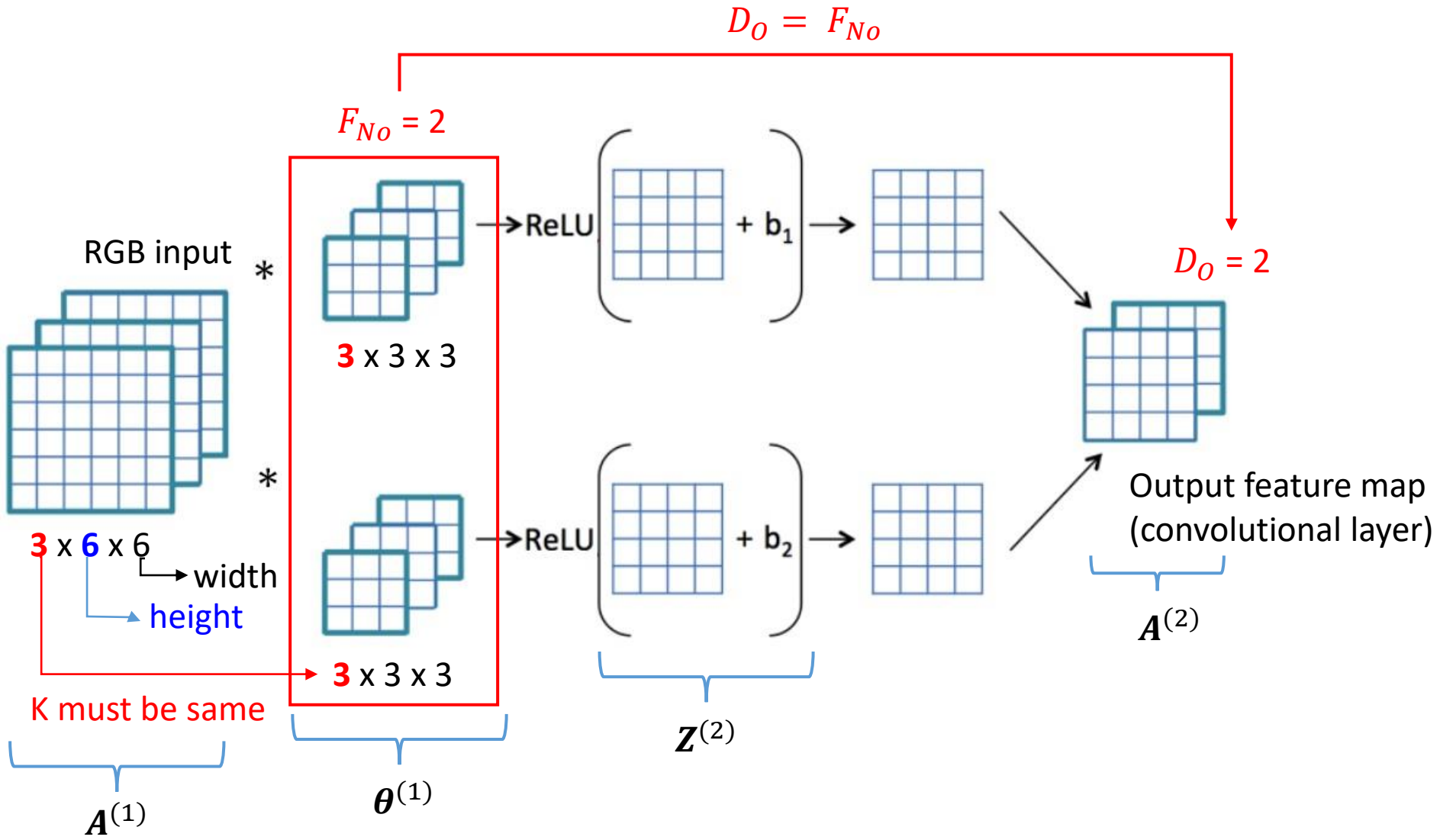


Fig 7.14 3D convolution scheme

How to define the size of output ?

W_i (Input channel width)= 5

H_i (Input channel height)= 5

D_i (Input map depth)= 3

P (Zero padding) = 1, F_s (Filter size) = 3 x 3, S (Stride) = 2

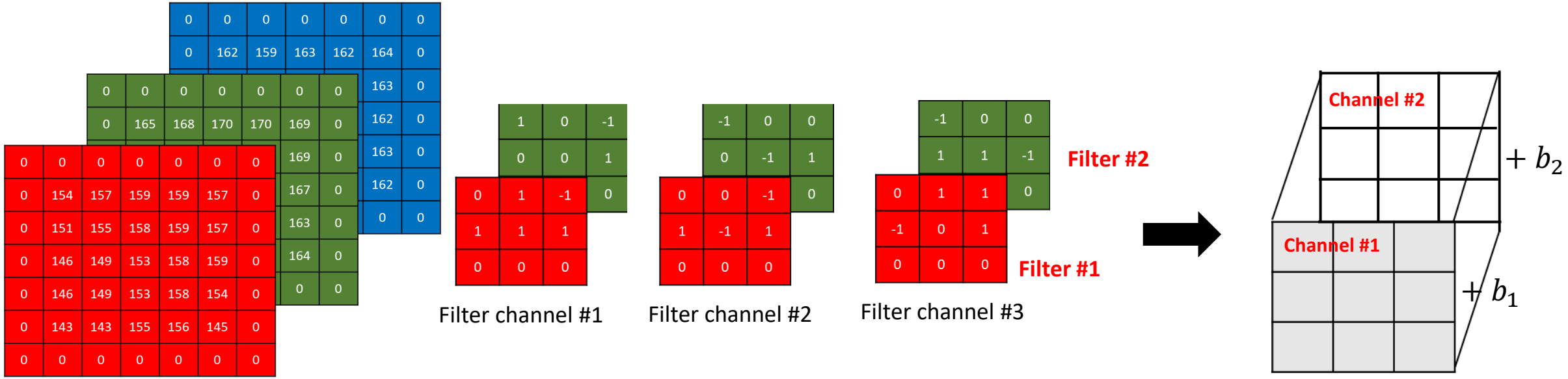
K = No. of channels of inputs or filters, F_{No} = No. of filters of outputs.

$$W_o = \frac{W_i + 2P - F_s}{S} + 1$$

$$H_o = \frac{H_i + 2P - F_s}{S} + 1$$

$D_O = F_{No}$

Creating an output convolution layer



RGB Input (3 x 5 x 5) with P = 1

Filter channel #1 Filter channel #2 Filter channel #3

Output Convolutional layer

Fig 7.15 3D convolution process

Here $*$ means convolution of two matrices and b is bias.

Performing 3D convolutions using 3D filters (1)

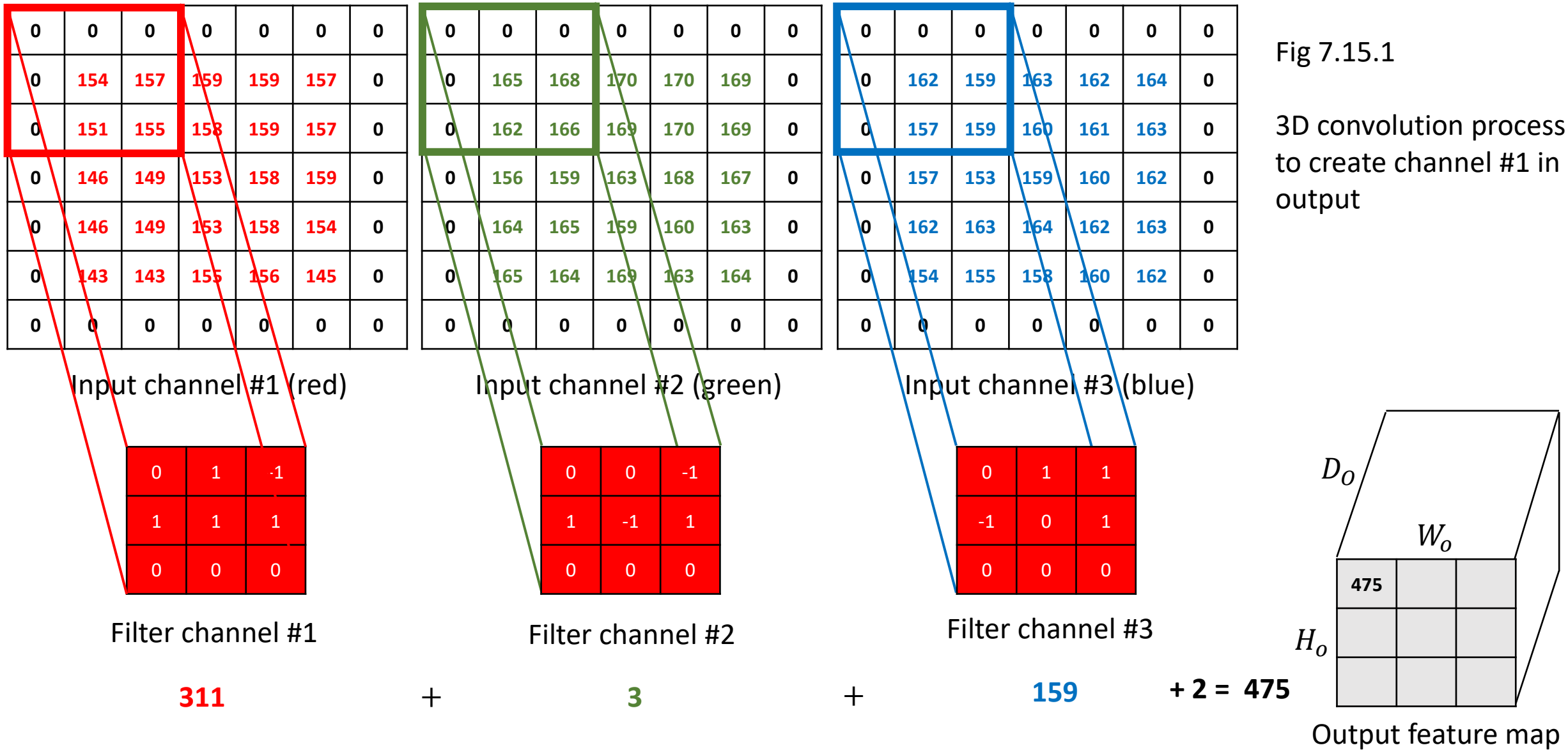


Fig 7.15.1

3D convolution process to create channel #1 in output

Performing 3D convolutions using 3D filters (2)

0	0	0	0	0	0	0
0	154	157	159	159	157	0
0	151	155	158	159	157	0
0	146	149	153	158	159	0
0	146	149	153	158	154	0
0	143	143	155	156	145	0
0	0	0	0	0	0	0

Input channel #1 (red)

0	0	0	0	0	0	0
0	165	168	170	170	169	0
0	162	166	169	170	169	0
0	156	159	163	168	167	0
0	164	165	159	160	163	0
0	165	164	169	163	164	0
0	0	0	0	0	0	0

Input channel #2 (green)

0	0	0	0	0	0	0
0	162	159	163	162	164	0
0	157	159	160	161	163	0
0	157	153	159	160	162	0
0	162	163	164	162	163	0
0	154	155	158	160	162	0
0	0	0	0	0	0	0

Input channel #3 (blue)

0	1	-1
1	1	1
0	0	0

Filter channel #1

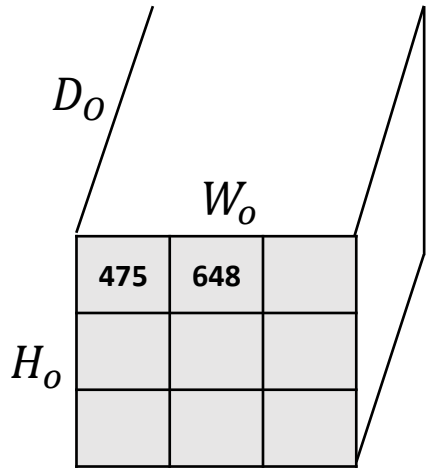
0	0	-1
1	-1	1
0	0	0

Filter channel #1

0	1	1
-1	0	1
0	0	0

Filter channel #1

475 + **168** + **3** **+ 2 = 648**



Output feature map

Fig 7.15.2

3D convolution process to create channel #1 in output

Performing 3D convolutions using 3D filters (3)

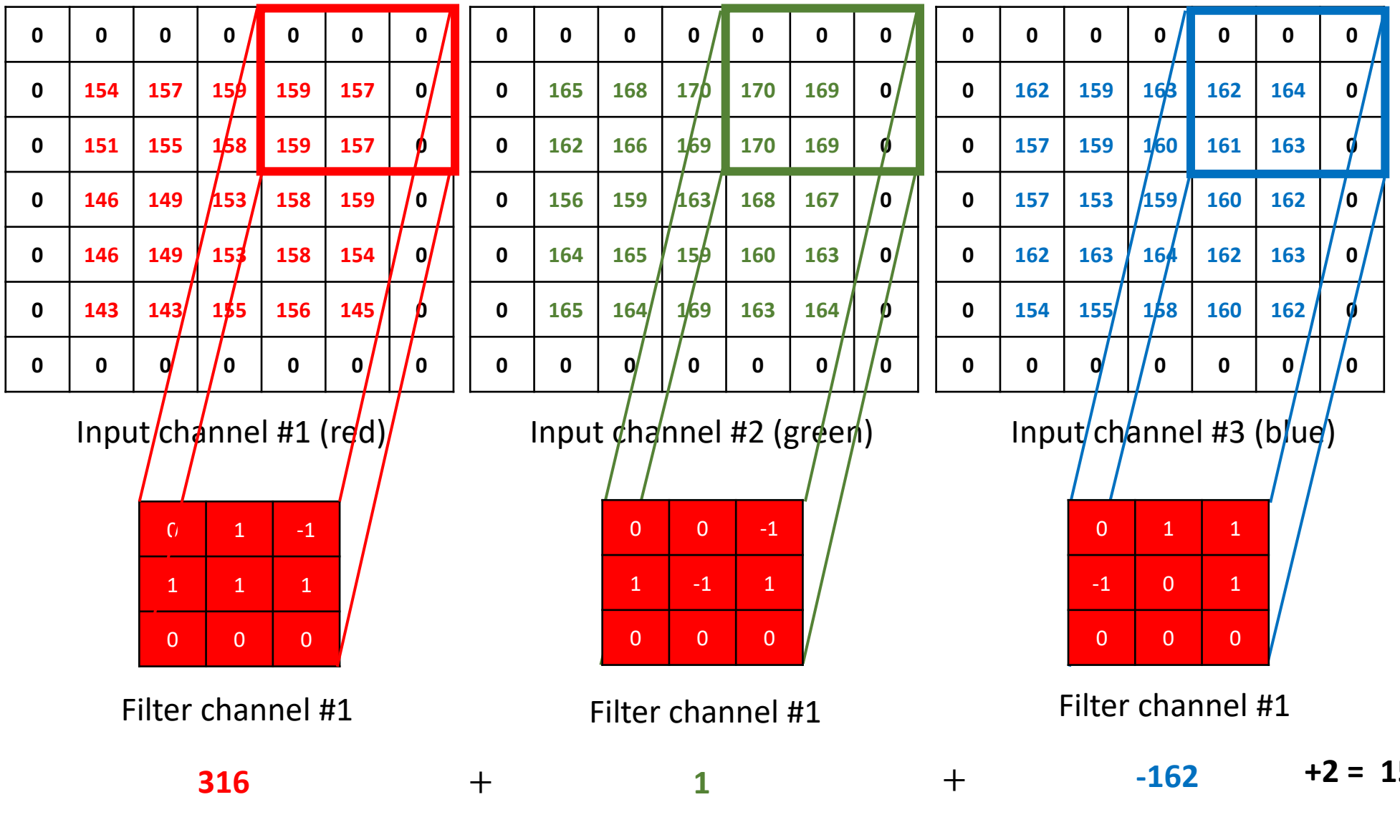


Fig 7.15.3

3D convolution process to create channel #1 in output

Performing 3D convolutions using 3D filters (4)

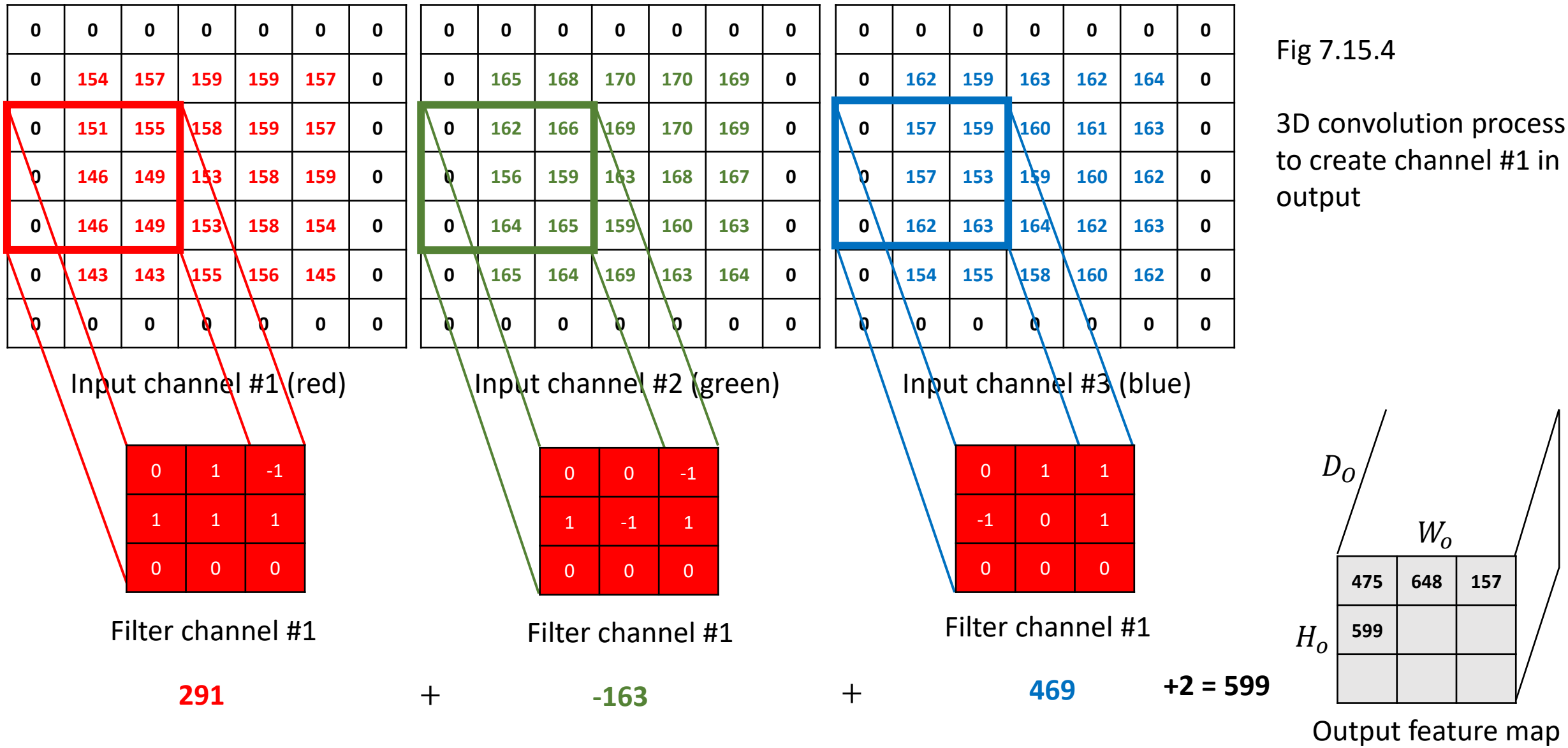


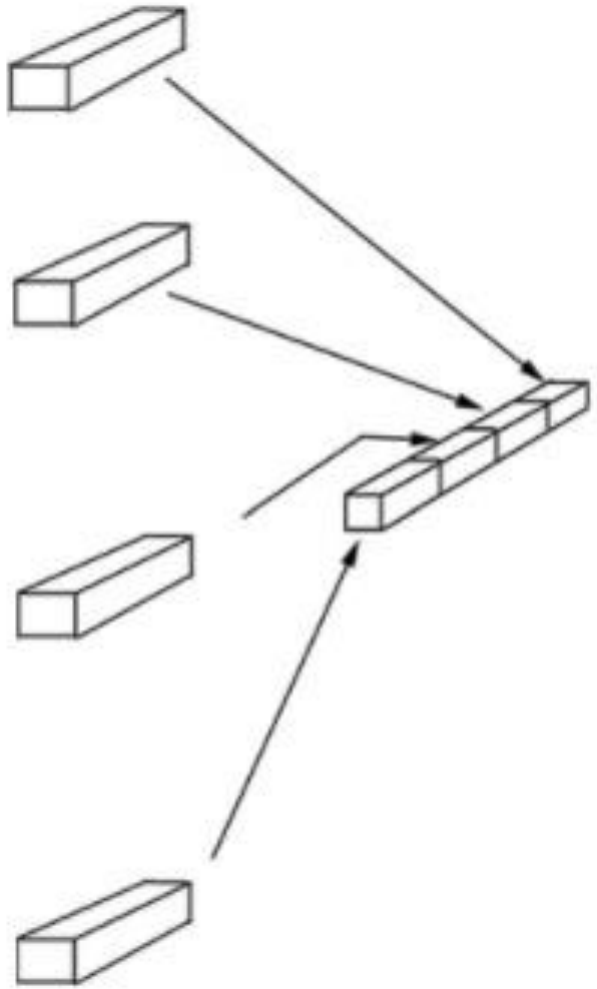
Fig 7.15.4

3D convolution process to create channel #1 in output

Padding Effects

- Padding increases the contribution of the pixels at the border of the original image by bringing them into the middle of the padded image. Thus, information on the borders is preserved as well as the information in the middle of the image. It allows us to design deeper networks without the loss of information of the original image.
- Network architectures need to concatenate or add convolutional layers with the different size of filters 3x3, 5x5 or 11 x 11 etc. and it wouldn't be possible with no padding because dimensions wouldn't match.

Concatenation



Addition

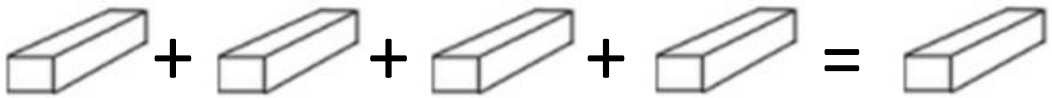


Fig 7.16 Concatenation and addition

7.6 Pooling

- Pooling is the sub-sampling technique to down sample the detection of feature in feature maps.
- A limitation of feature map output of Convolutional layers is that they record the precise position of features in input image. It means that small movement in the position of features in image will result in a different feature map. In other words, they are highly sensitive to the location of features in input. One of the approaches to address the sensitivity is Pooling technique. This is where a lower resolution version of an input signal is created that still contains the large or important structural elements without the fine detail that may not be as useful to the task.
- Pooling layer should be placed after Convolutional layers.

Example of Pooling:

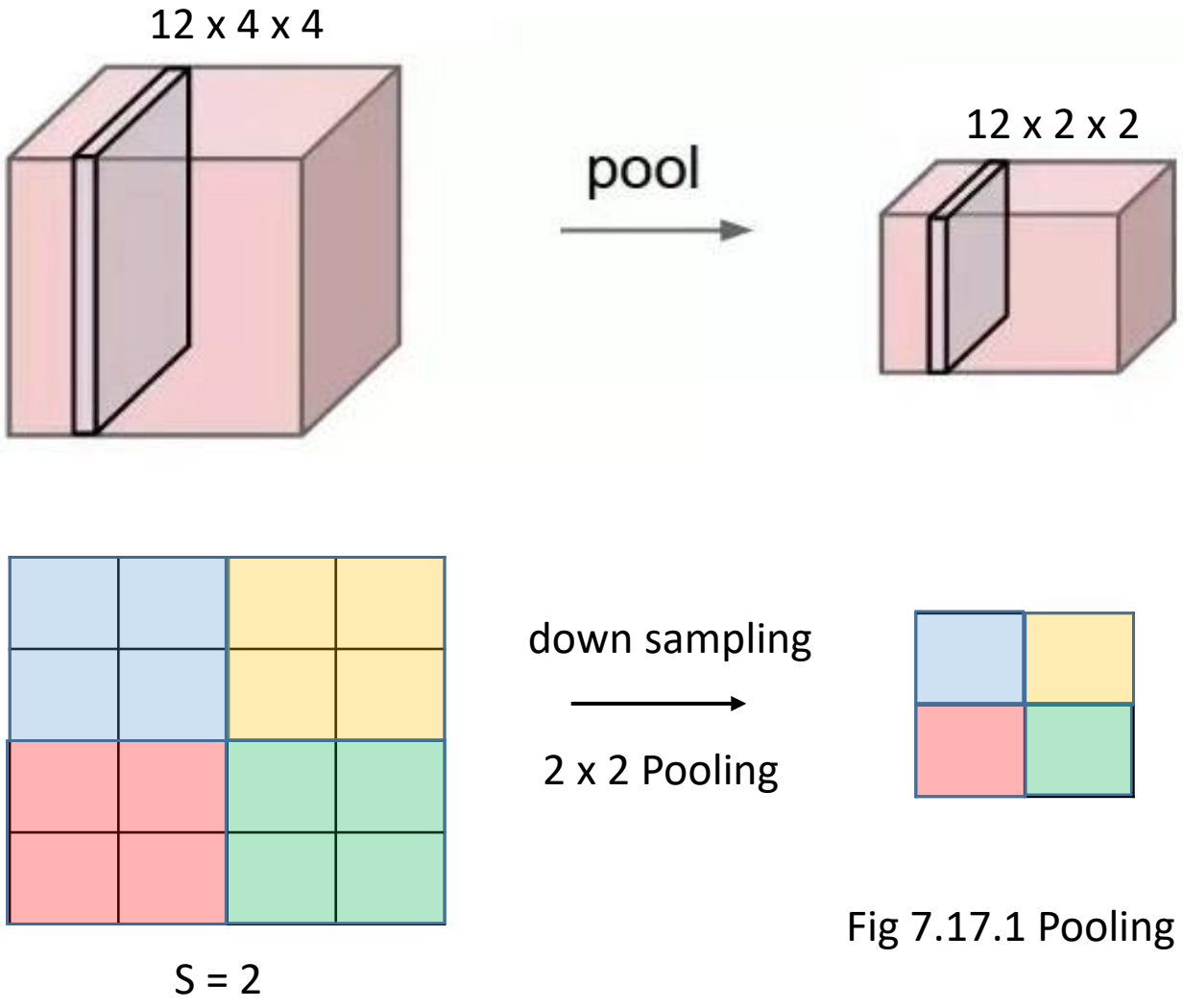


Fig 7.17.1 Pooling process

Max Pooling: The maximum pixel value of the batch (filter) is selected.

Min Pooling: The minimum pixel value of the batch is selected.

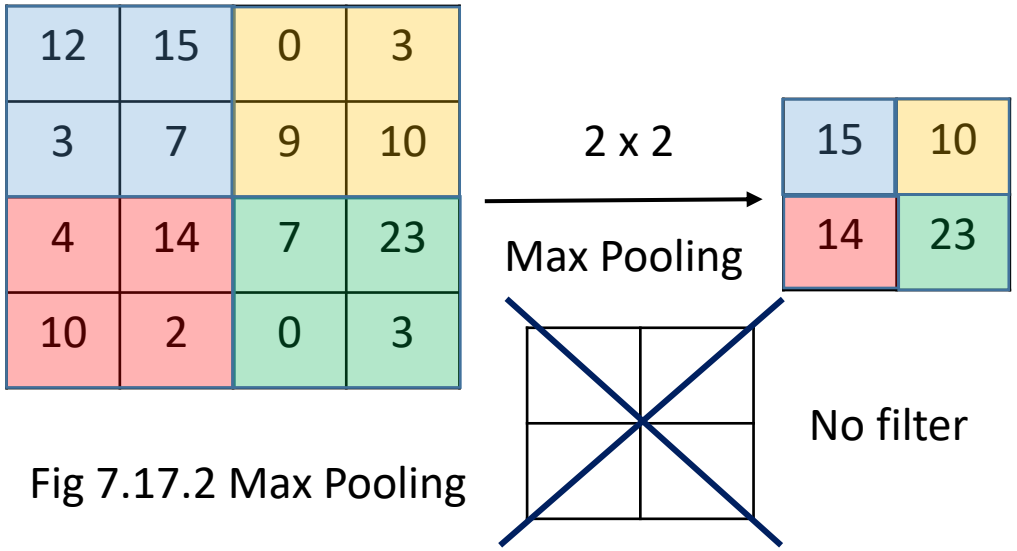


Fig 7.17.2 Max Pooling

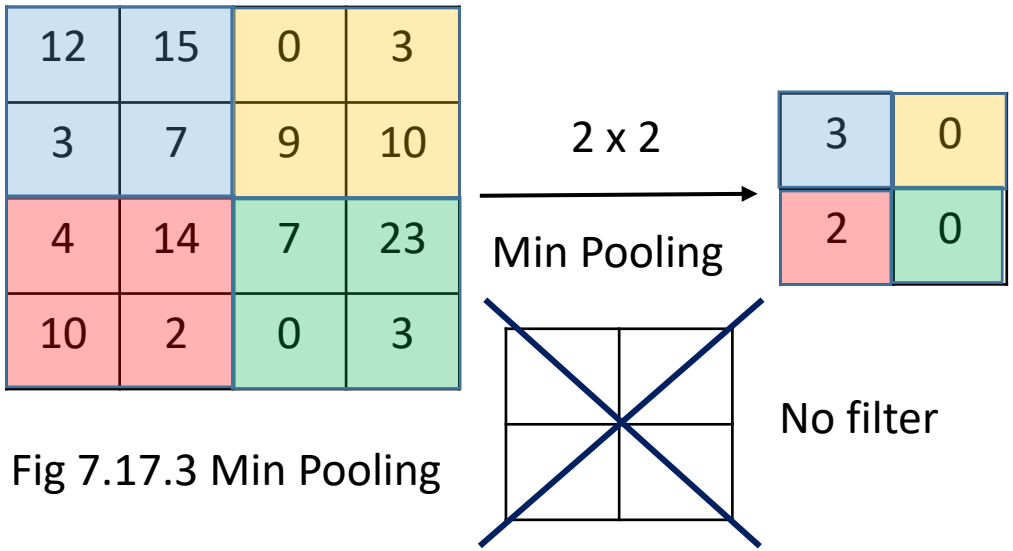


Fig 7.17.3 Min Pooling

Max Pooling selects the brighter pixels from the image. It is useful when the background of the image is dark and we are interested in only the lighter pixels of the image. Similarly, Min Pooling is useful in the reverse case.

Average Pooling: The average value of all the pixels in the batch is selected.

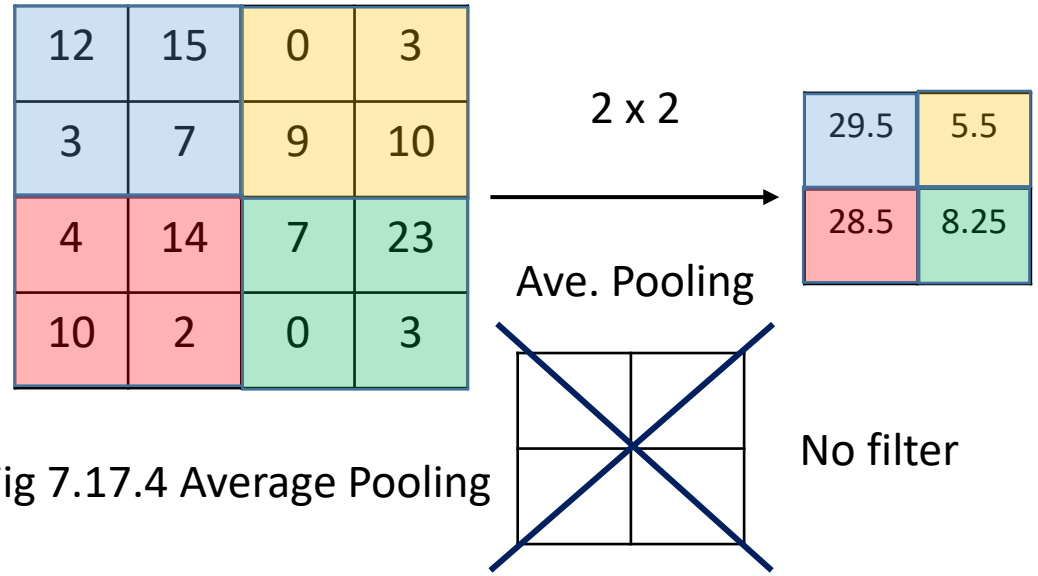


Fig 7.17.4 Average Pooling

Average pooling method smooths out the image and hence the sharp features may not be identified when it is used.

7.7 Architecture Analysis

LeNet-5 is the convolutional network designed for handwritten numbers (0 – 9) recognition.

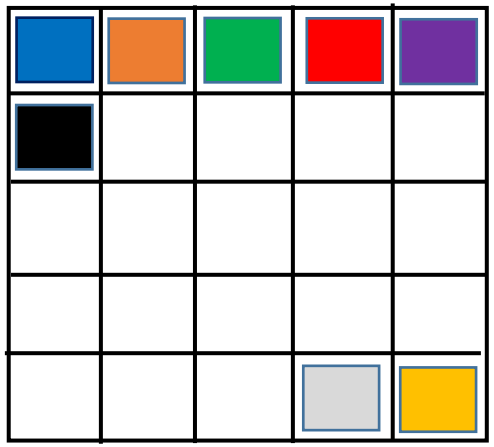
Table 7.2 LeNet-5 Architecture

* Non-Zero Padding

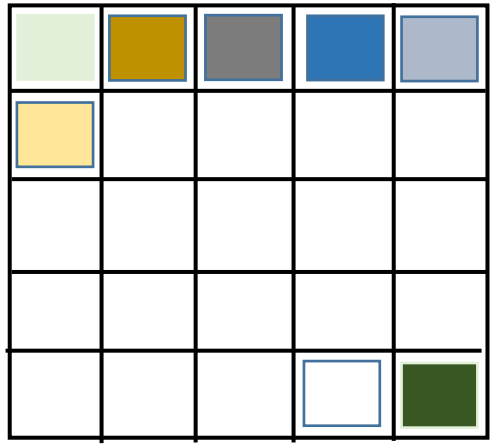
Filter shape (K x H_f x W_f) (F_{NO})	Stride (S)	Feature map shape (D_O x H x W)	Activation size	Activation type	Layer Type
-	-	1 x 32 x 32	1,024	-	Input image
1 x 5 x 5 (F_{NO}=6)	1	6 x 28 x 28	4,704	tanh	Conv. 1
-	2	6 x 14 x 14	1,176	tanh	Av. PL 1 (2x2 pooling)
6 x 5 x 5 (F_{NO}=16)	1	16 x 10 x 10	1,600	tanh	Conv. 2
-	2	16 x 5 x 5	400	tanh	Av. PL 2 (2x2 pooling)
-	-	-	400	tanh	Flattening
-	-	-	120	tanh	FC 1
-	-	-	84	tanh	FC 2
-	-	-	10	Softmax	FC 3

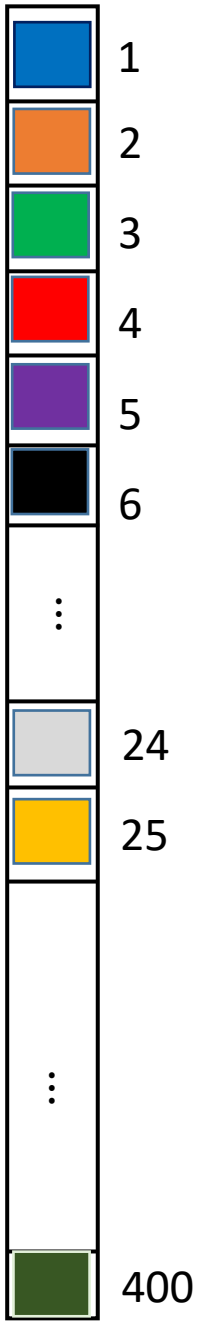
Flattening:

Channel #1

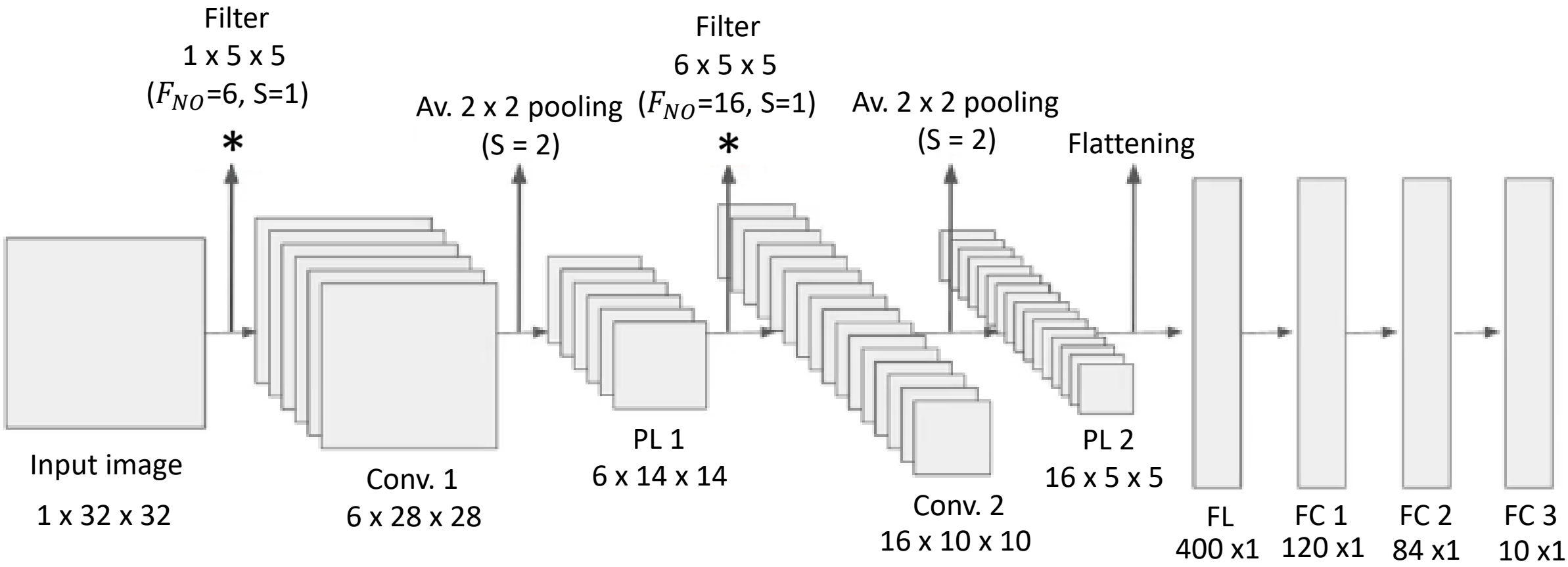


Channel #16





Visualization of LeNet-5 architecture



7.8 Deconvolution (Transposed convolution)

What is deconvolution in CNN?

Deconvolution is an intensive image processing algorithm used to reverse the effects of convolution on recorded data. In CNN, convolution reduces the size of the feature map. However, deconvolution, on the other hand, works by increasing the size of the feature map.

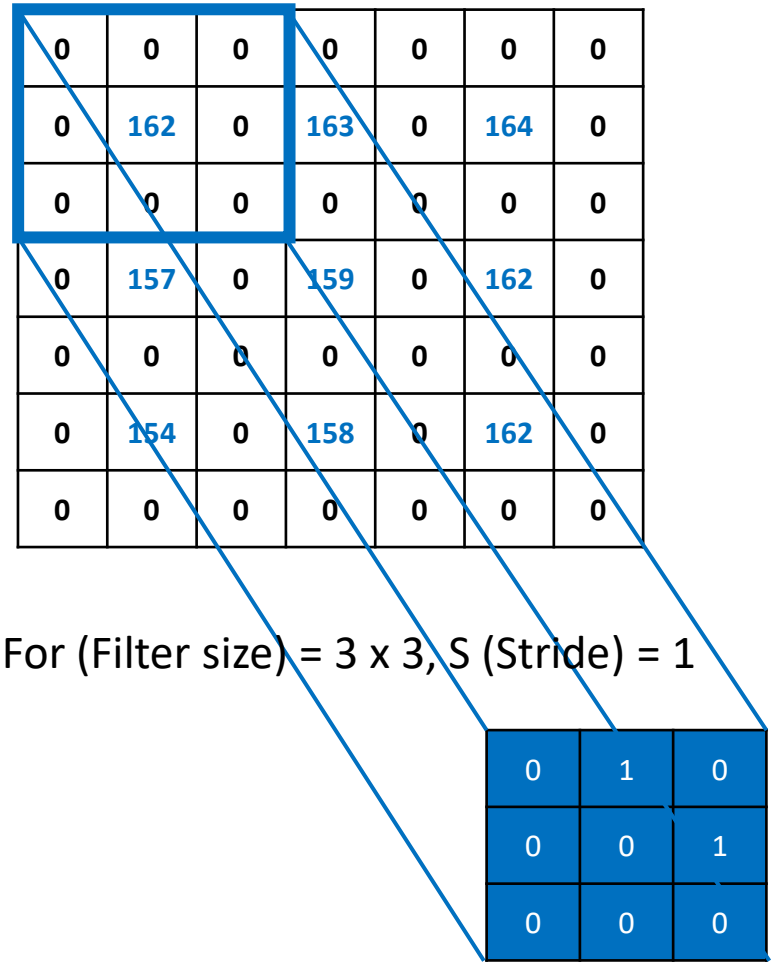
What is the purpose of deconvolution?

It is utilized for **improving the contrast and resolution of digital images captured in the microscope** and can be also mainly used for **semantic segmentation**. .

- Deconvolution is used when you want to recover the same size as the input by reversing the final result of normal CNN, in other word, is more efficient than the normal convolution operation in the case when you want to go back from convolved values to the original ones (opposite direction):

  - Original Image → Convolution → Result
  - Result → Deconvolution → Original Image

- Sometimes you can store some important values along the convolution path and reuse it when going back.



For (Filter size) = 3 x 3, S (Stride) = 1

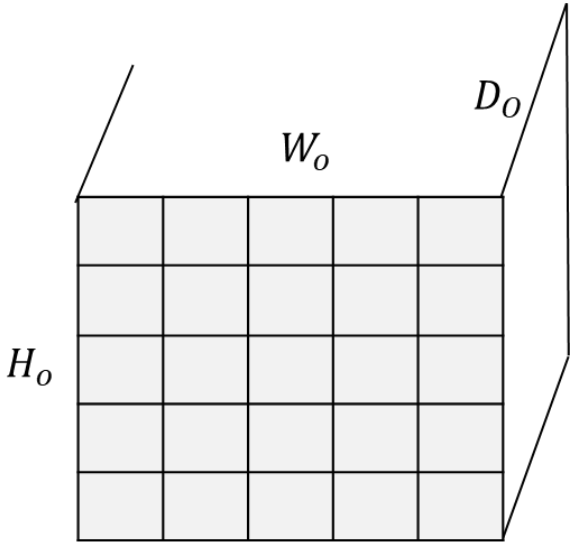
Fig 7.18 Deconvolution process

Deconvolution works in the following way:

(1) Add zero-padding around each pixel.

(2) Convolution operation is performed on this padding.

It is called 'Upsampling' 3 x 3 → 5 x 5



Output feature map