Many deep learning courses have contents like
- fully-connected networks
- its optimization problem
- its gradient (back propagation)
- ...
- other types of networks (e.g., CNN)
- ...

If I am a student of such courses, after seeing the significant differences of CNN from fully-connected networks, I wonder how the back propagation can be done.
The problem is that back propagation for CNN seems to be very complicated.
So fewer people talk about details.
Here we try to give a clear explanation.
Consider two layers $m$ and $m + 1$. The variables between them are $W^m$ and $b^m$, so we aim to calculate

$$
\frac{\partial f}{\partial W^m} = \frac{1}{C} W^m + \frac{1}{l} \sum_{i=1}^{l} \frac{\partial \xi_i}{\partial W^m},
$$

(1)

$$
\frac{\partial f}{\partial b^m} = \frac{1}{C} b^m + \frac{1}{l} \sum_{i=1}^{l} \frac{\partial \xi_i}{\partial b^m}.
$$

(2)

Note that (1) is in a matrix form.
Following past developments such as Vedaldi and Lenc (2015), it is easier to transform them to a vector form for the derivation.
For the convolutional layers, recall that

\[ S^{m,i} = W^m \text{mat}(P^m_{\phi}P^m_{\text{pad}} \text{vec}(Z^{m,i}))_{h^m h^m d^m \times a^m_{\text{conv}} b^m_{\text{conv}}} + \phi(\text{pad}(Z^{m,i})) \]

\[ b^m_{1} \begin{bmatrix} 1 \\ a^m_{\text{conv}} b^m_{\text{conv}} \end{bmatrix} \]

\[ Z^{m+1,i} = \text{mat}(P^m_{\text{pool}} \text{vec}(\sigma(S^{m,i}))_{d^{m+1} \times a^{m+1} b^{m+1}}, \quad (3) \]
We have

\[ \text{vec}(S^{m,i}) = \text{vec}(W^m \phi(\text{pad}(Z^{m,i}))) + \text{vec}(b^m 1^{\top}_{a_{\text{conv}} b_{\text{conv}}}) \]

\[ = (\mathcal{I}_{a_{\text{conv}} b_{\text{conv}}} \otimes W^m) \text{vec}(\phi(\text{pad}(Z^{m,i}))) + (1^{\top}_{a_{\text{conv}} b_{\text{conv}}} \otimes I_{d^{m+1}}) b^m \]

(4)

\[ = (\phi(\text{pad}(Z^{m,i}))^T \otimes I_{d^{m+1}}) \text{vec}(W^m) + (1^{\top}_{a_{\text{conv}} b_{\text{conv}}} \otimes I_{d^{m+1}}) b^m , \]

(5)
where $\mathcal{I}$ is an identity matrix. For example,

$$\mathcal{I}a_{\text{conv}}^m b_{\text{conv}}^m$$

is an

$$a_{\text{conv}}^m b_{\text{conv}}^m \times a_{\text{conv}}^m b_{\text{conv}}^m$$

identity matrix. Eqs. (4) and (5) are respectively from

$$\text{vec}(AB) = (\mathcal{I} \otimes A)\text{vec}(B)$$  \hspace{1cm} (6)

$$= (B^T \otimes \mathcal{I})\text{vec}(A), \hspace{1cm} (7)$$
Here \( \otimes \) is the Kronecker product.

What’s the Kronecker product? If 

\[ A \in \mathbb{R}^{m \times n} \]

then

\[ A \otimes B = \begin{bmatrix} a_{11}B & \cdots & a_{1n}B \\ \vdots & & \vdots \\ a_{m1}B & \cdots & a_{mn}B \end{bmatrix}, \]

a much bigger matrix
Vector Form V

- For the fully-connected layers,

\[ s^{m,i} = W^m z^{m,i} + b^m \]

\[ = (I_1 \otimes W^m) z^{m,i} + (1_1 \otimes I_{n_{m+1}}) b^m \]

\[ = (\overline{z^{m,i}}^T \otimes I_{n_{m+1}}) \text{vec}(W^m) + (1_1 \otimes I_{n_{m+1}}) b^m, \]

where (8) and (9) are from (6) and (7), respectively.

- An advantage of using (4) and (8) is that they are in the same form.
Further, if for fully-connected layers we define
\[ \phi(\text{pad}(z^{m,i})) = I_{n_m} z^{m,i}, \quad L^c < m \leq L + 1, \]
then (5) and (9) are in the same form.

Thus we can derive the gradient of convolutional and fully-connected layers together