

## Lecture 14: Convolutional Neural Networks

Lecturer: Swaprava Nath

Scribe(s): SG27 &amp; SG28

**Disclaimer:** These notes aggregate content from several texts and have not been subjected to the usual scrutiny deserved by formal publications. If you find errors, please bring to the notice of the Instructor.

## 14.1 Introduction

Convolutional Neural Networks are extensively used in image processing tasks such as object detection and image classification, where the input data is typically large in size. Convolution provides a way to reduce the size of the input without compromising the information contained within it.

For instance, consider a coloured image of size  $1024 \times 1024$ . Typically a coloured image is made of 3 colour channels corresponding to red, green and blue colours in the RGB colourspace, and each of these channels is of size  $1024 \times 1024$ . Thus, the number of pixels in the input image is  $1024 \times 1024 \times 3$ .

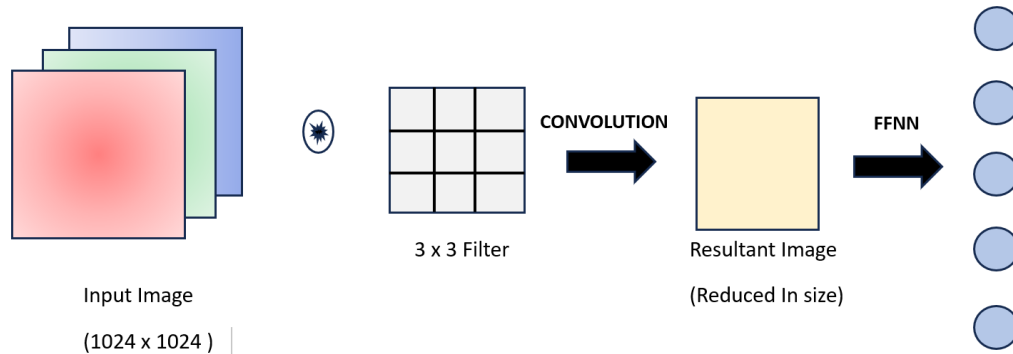


Figure 14.1: A Basic Convolutional Neural Network

In a traditional feedforward neural network, each of these pixels would correspond to a node in the input layer, resulting in an input size of the order of  $10^6$ . If we were to have 1000 neurons in the first hidden layer, the number of parameters needed to train our model would blow up to  $3 \times 10^9$ , becoming very computationally heavy and might even lead to overfitting.

In practice, specially in image processing, we do not need to fine grain these parameters as we are more concerned about edges and object boundaries instead of the information held in each pixel. This is where CNNs step in. The heart of the convolution process involved is to reduce the image size without losing the data stored at the pixel level.

## 14.2 Convolution Operation

The convolution operation between two functions  $f$  and  $g$  (denoted by  $f * g$ ) is defined as:

$$f(x) * g(x) = \int_{-\infty}^{\infty} f(t)g(x-t)dt$$

Convolution essentially combines the information of the two input functions by measuring the overlap between them at every point and integrating the product over the entire range of possible shifts. This operation is often visualized as sliding one function ( $g$ ) over another ( $f$ ), multiplying them at each point of overlap, and summing up the results.

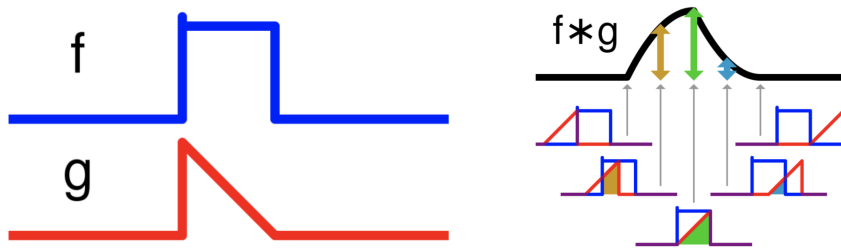


Figure 14.2: Convolution Operation

Convolution is widely used in signal processing for bandpass filtering and noise reduction. This operation is particularly effective for filtering because it allows for the extraction of specific frequency components from a signal while suppressing others. For instance, in bandpass filtering, convolution kernel enables the isolation of frequency bands of interest by convolving the input signal with a suitable filter kernel.

In our case of image classification, we will use the matrix form of convolution wherein we apply filters to the input image by convolving them with appropriate kernel matrices. This involves sliding the filter matrix over the input image, computing element-wise multiplications, summing these products, and aggregating the results to generate the output image.

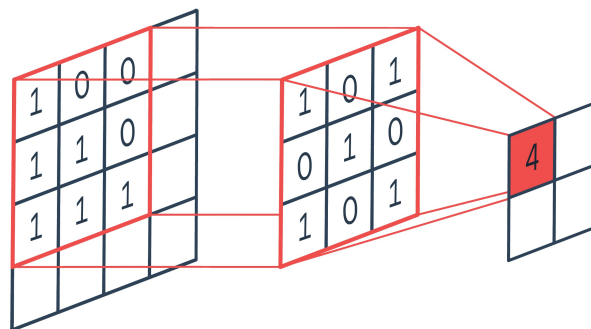


Figure 14.3: Matrix Convolution

It can be easily verified that if the input image is of size  $(n \times n)$  and the applied filter is  $(f \times f)$ , then the output image formed is of size  $(n-f+1) \times (n-f+1)$ .

### 14.3 Examples based on grayscale

The RGB model uses 8 bits each – from 0 to 255 – for red, green and blue colors at each pixel. Each color also has values ranging from 0 to 255. This translates into millions of colors – 16,777,216 possible colors to be precise.

In grayscale images, a pixel value between 0 and 255 represents the brightness of the pixel. 0 is black and 255 is white. The values between 0 and 255 are shades of gray, with values closer to 0 being darker and values closer to 255 being lighter.

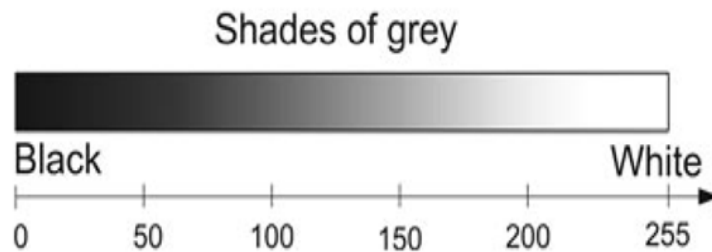


Figure 14.4: Grayscale image [1]

The grayscale image matrix  $f$  represents the intensity values of pixels in a 5x5 image.

$$\text{Grayscale Image Matrix - (f)} = \begin{bmatrix} 1 & 20 & 20 & 20 & 1 \\ 20 & 1 & 1 & 1 & 20 \\ 20 & 1 & 1 & 1 & 20 \\ 1 & 20 & 1 & 20 & 1 \\ 1 & 1 & 20 & 1 & 1 \end{bmatrix}$$

### Vertical Edge Detection

$$\text{Vertical Edge Detector - (g)} = \begin{bmatrix} 1 & 0 & -1 \\ 1 & 0 & -1 \\ 1 & 0 & -1 \end{bmatrix}$$

The vertical edge detector matrix  $g$  is a 3x3 kernel designed to detect vertical edges in an image. When the convolution operation  $f * g$  is applied, the resulting output matrix represents the response of the image to the vertical edge detector.

The output matrix is given by:

$$\text{Output matrix } f * g = \begin{bmatrix} 19 & 0 & -19 \\ 38 & 0 & -38 \\ 0 & 0 & 0 \end{bmatrix}$$

The values in the output matrix indicate the degree of change in intensity, with positive values representing edges and negative values representing the opposite direction of the edges. In this specific case, the matrix shows that there is a strong response to vertical edges present in the input image.

Therefore, by convolving the grayscale image with the vertical edge detector, we have successfully detected and highlighted the vertical edges in the original image.

## Horizontal Edge Detection

Now, let's consider using a horizontal edge detector matrix to detect horizontal edges in the grayscale image matrix  $f$ .

$$\text{Horizontal Edge Detector} - (g') = \begin{bmatrix} 1 & 1 & 1 \\ 0 & 0 & 0 \\ -1 & -1 & -1 \end{bmatrix}$$

When we convolve the grayscale image matrix  $f$  with the horizontal edge detector matrix  $g'$ , we obtain the following output matrix:

$$\text{Output matrix } f * g' = \begin{bmatrix} 19 & 57 & 19 \\ 0 & -38 & 0 \\ 0 & -19 & 0 \end{bmatrix}$$

## In coloured Images

These are some examples of convolution in coloured images.

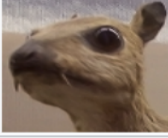
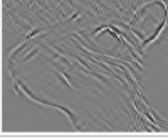
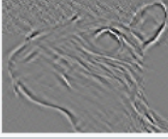
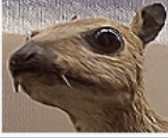


Operation	Kernel $\omega$	Image result $g(x,y)$
<b>Identity</b>	$\begin{bmatrix} 0 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 0 \end{bmatrix}$	
<b>Ridge or edge detection</b>	$\begin{bmatrix} 0 & -1 & 0 \\ -1 & 4 & -1 \\ 0 & -1 & 0 \end{bmatrix}$	
	$\begin{bmatrix} -1 & -1 & -1 \\ -1 & 8 & -1 \\ -1 & -1 & -1 \end{bmatrix}$	
<b>Sharpen</b>	$\begin{bmatrix} 0 & -1 & 0 \\ -1 & 5 & -1 \\ 0 & -1 & 0 \end{bmatrix}$	
<b>Box blur</b> (normalized)	$\frac{1}{9} \begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix}$	
<b>Gaussian blur 3 x 3</b> (approximation)	$\frac{1}{16} \begin{bmatrix} 1 & 2 & 1 \\ 2 & 4 & 2 \\ 1 & 2 & 1 \end{bmatrix}$	

Figure 14.5: Convolution filters output on a coloured image [2]

## 14.4 Need for Multiple Filters in CNNs

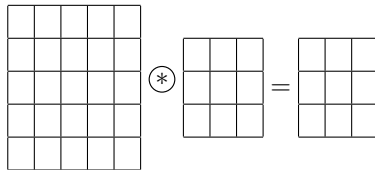
In Convolutional Neural Networks (CNNs), the use of multiple filters is crucial for:

1. **Feature Hierarchy:** Different filters capture features at varying abstraction levels, forming a hierarchical representation.
2. **Variability Handling:** Filters with diverse orientations and scales handle variations in object orientation and size.
3. **Spatial Hierarchies:** Filters learn spatial hierarchies, capturing details and relationships across different scales.
4. **Class-Specific Features:** Filters specialize in extracting features relevant to specific object classes.
5. **Increased Model Capacity:** Multiple filters enhance the network's capacity, allowing it to model a broader range of features.
6. **Regularization and Generalization:** Diverse filters act as regularization, preventing overfitting and improving generalization.

## 14.5 Tools Used In CNNs

### 14.5.1 Padding

Consider the example below. Here we are using 5 X 5 grid and a 3 X 3 filter.

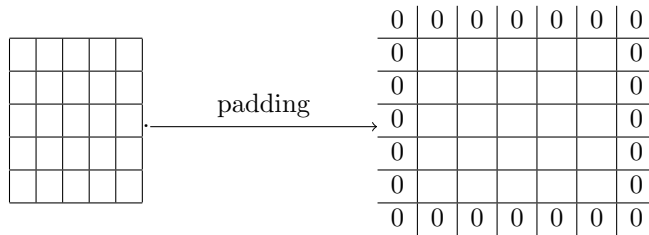


Every pixel affects some pixels of the final grid. But the corner pixels appear lesser number of times. For example, the top left pixel affects only one pixel in the final grid. But the pixels in middle affects much more pixel. For example, the centre most pixel affects all the nine pixels of the final grid. To be clear, when we say "X affects Y" we mean that value of X is used in the calculation of the value of Y.

Since this, the information about the corner pixels is not captured as much as of the pixels in the mid. To avoid this we use the technique called "Padding".

Formally, padding is an operation which takes a grid and a natural number as inputs and outputs another grid which is basically the original grid with a few layers of zeros surrounding that grid. The number of layers depends on the second input.

Consider the example below.



Here we have padded only one layer of zeros.

If suppose p layers of zeros have been added then the size of the new grid is given by

$$(n + 2p) * (n + 2p)$$

Further, suppose that the size of the filter used for convolution is f then the size of the final output after padding with p layers and convolution with this filter is

$$((n + 2p) - f + 1) * ((n + 2p) - f + 1)$$

For

$$((n + 2p) - f + 1) = n$$

$$2p = f - 1$$

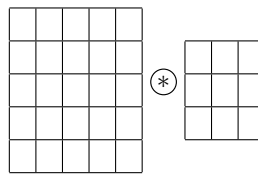
$$p = \frac{(f - 1)}{2}, \text{ where } f \text{ is odd}$$

### 14.5.2 Striding

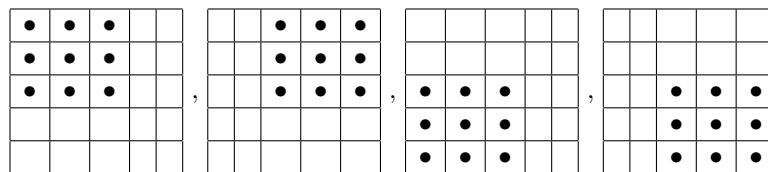
Often the image pixel values changes gradually. Given this we don't need to apply the filter to consecutive blocks, instead we can skip some of the blocks which is what we call as "striding". This way we can decrease the size of the grid obtained after convolution while at the same time we are fetching the information the filter was supposed to fetch.

Formally, stride is the amount of shift of the filter between two consecutive evaluations. It should be noted that striding has to be applied both vertically and horizontally. What does this mean will become clear with the following example.

Here we are going to use convolution operation with the grid size  $5 \times 5$  and the filter size  $3 \times 3$  but this time with stride being 2.



The different blocks that are considered for convolution are:



As we can see the vertical shifts as well as the horizontal shifts are by 2 units. So this time the resultant grid will be  $2 \times 2$ .

The standard value for stride is 1. Even in the Python libraries the default value is 1. We can also generalize the size of the resultant grid as follow.

If the grid size is  $n_1 \times n_2$  and the filter is square grid with size  $f$  and the stride is  $s$  then the resultant grid will be of size

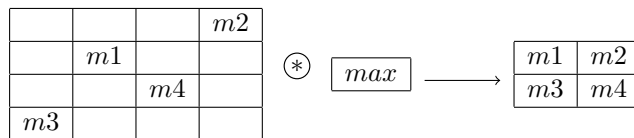
$$\left(\left\lfloor \frac{(n_1 - f)}{s} \right\rfloor + 1\right) \times \left(\left\lfloor \frac{(n_2 - f)}{s} \right\rfloor + 1\right)$$

While striding we consider only those evaluations where the filter fully lies in the grid, which is why the floor has been taken.

### 14.5.3 Max Pooling Layer

The Max Pooling Layer is used to reduce the size of the input. In this, we take only the max valued pixel as per grayscale at every evaluation so that, our image sharpens/contrast increase.

Even after padding and striding of the input image we may get a filtered image that is large, so we use the max pooling layer to reduce the size.



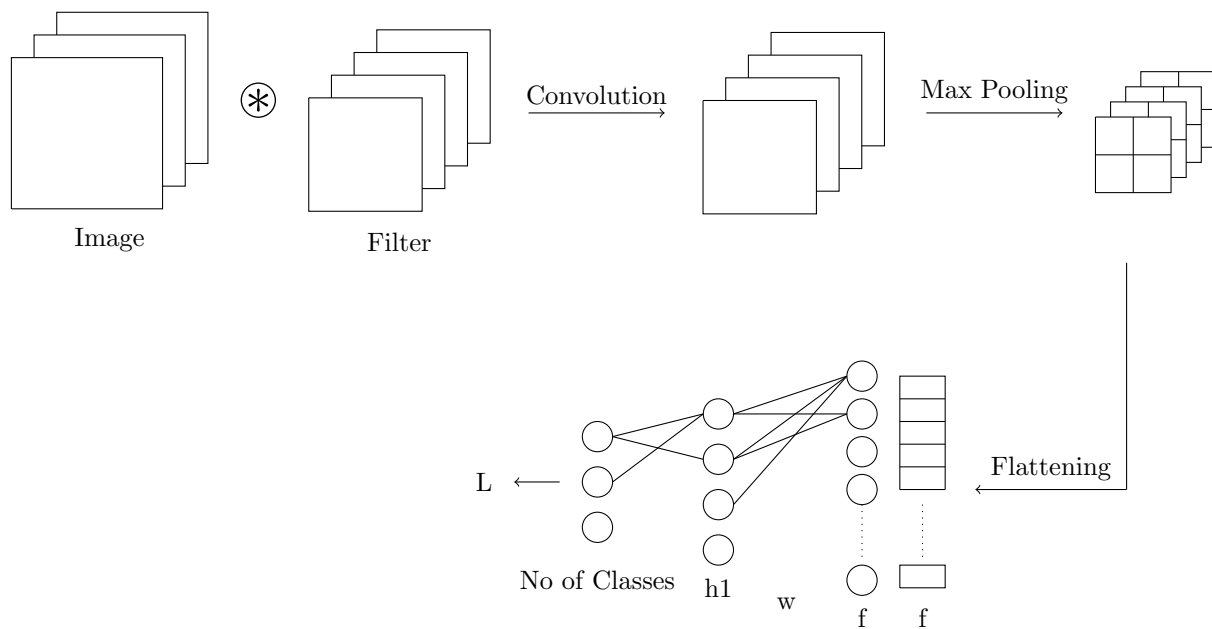
Here, our image is of grid size 4 x 4 and filter is of size 2 x 2 with a stride of 2 and the filtered image is of size 2 x 2. So, in max pooling, we consider only the max value in every evaluation (in every f x f region). In the above example, m1 is max in the first 2 x 2 grid, and similarly m2, m3, and m4 in their respective 2 x 2 grids. Therefore, our 4 x 4 input image is reduced to a 2 x 2 grid which is sharpened as we are considering the brightest pixels only.

In Max Pooling, the typical value of stride is the side of the filter size. And one of the main advantages is that this does not have any parameters.

Another alternative is to take the average of all pixels in every evaluation (in every f x f region).

### 14.6 Neural Network Architecture

The Neural Network Architecture used to classify images using these features is called the **Convolutional Neural Network (CNN)**.

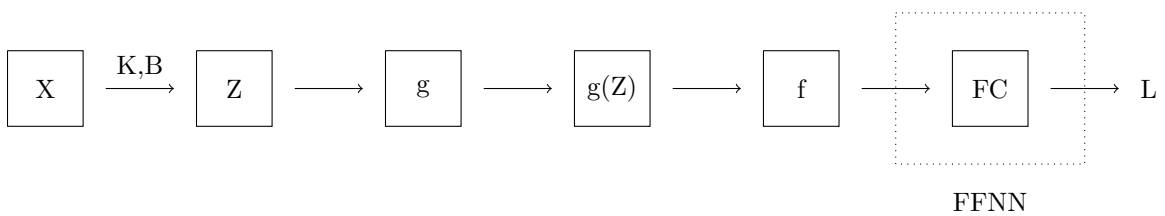
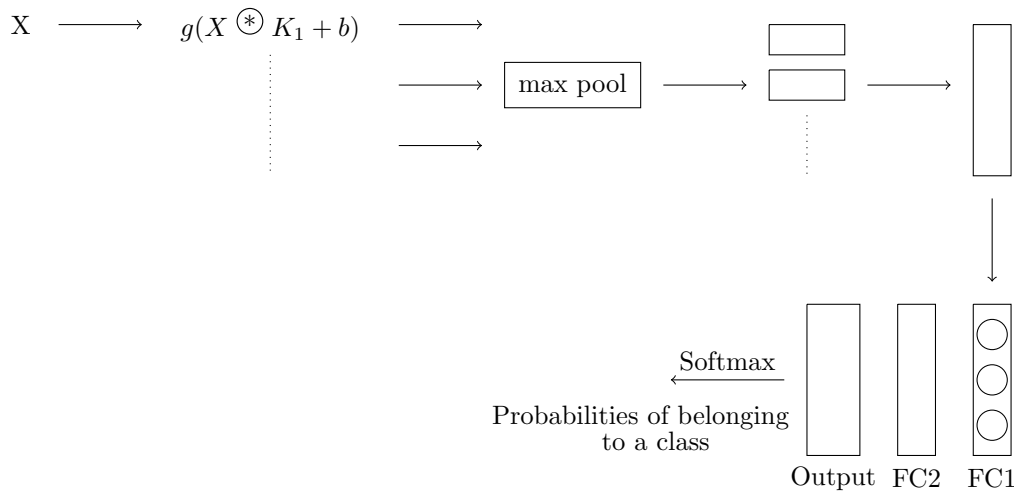
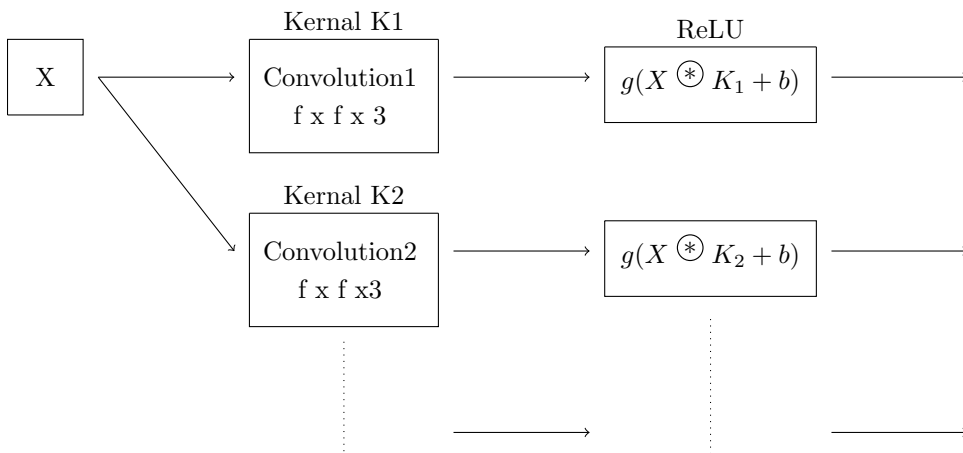


Therefore, after using all these features we obtain various nodes, after which we have a FC layer FFNN. By using Forward pass and Backward pass we can find the weights for this FC layer by gradient descent which minimizes L.

Filters in a neural network can be having specific properties to classify features, such as detecting horizontal edges, vertical edges, circles, and so on. These filters themselves are also set via Backward Pass as these are parameters in the CNN as well.

ReLU activation function is used in CNNs, denoted as  $g(.)$  henceforth.





$$Z = X \circledast K + B$$

filter/kernel

$$K^{(t+1)} \leftarrow K^{(t)} - \eta * \frac{\partial L}{\partial K} \Big|_t$$

$$B^{(t+1)} \leftarrow B^{(t)} - \eta * \frac{\partial L}{\partial B} \Big|_t$$

## 14.7 Gradients for back propagation

Let  $X$  represent the input image,  $K$  represent the kernel,  $B$  represent the scalar bias,  $Z$  represent the result of convolution operation. Note that here the convolution is with standard stride of 1.

$$Z = X \circledast K + B$$

Suppose we know the gradient of Loss with respect to  $Z$ . How to calculate the gradient of Loss with respect to  $X$ ,  $K$  and  $B$ ?

To answer this question we would like to know how infinitesimal changes affect entries of  $Z$ . For the following discussion,  $\frac{\partial L}{\partial W}$  represent the matrix where each entry being equal to the gradient of  $L$  with respect to the corresponding entry in  $W$ .

Let's consider  $K$  first. To calculate the gradient of  $L$  wrt  $K$  we need the gradient of each entry of  $Z$  wrt each entry of  $K$ . Let's consider the entry  $K_{i,j}$ . As we know that to calculate each entry of  $Z$ , we basically keep the filter over some specific block of the grid and do the convolution which is followed by addition of bias. So for each entry in  $Z$ , say  $Z_{c,d}$ , there is a specific element of  $X$  which is multiplied with  $K_{i,j}$ . Using the definition of convolution we can see that the element is also numerically equal to  $\frac{\partial Z_{c,d}}{\partial K_{i,j}}$ .

Suppose  $X_{a_i,j,b_i,j}$  is the coefficient in the case of  $Z_{0,0}$ . Now consider  $Z_{c,d}$ . To calculate this we must have moved the filter  $d$  rightwards and  $c$  downwards because of which the element that is below  $K_{i,j}$  will change from  $X_{a_i,j,b_i,j}$  to  $X_{a_i,j+c,b_i,j+d}$ . Moreover,  $a_{i,j} = i$  and  $b_{i,j} = j$  because for evaluation of  $Z_{0,0}$  we place the kernel at the top left corner. Now,

$$\begin{aligned}\frac{\partial L}{\partial K_{i,j}} &= \sum_{c,d} \frac{\partial L}{\partial Z_{c,d}} \frac{\partial Z_{c,d}}{\partial K_{i,j}} \\ \frac{\partial L}{\partial K_{i,j}} &= \sum_{c,d} \frac{\partial L}{\partial Z_{c,d}} X_{a_i,j+c,b_i,j+d} \\ \frac{\partial L}{\partial K_{i,j}} &= \sum_{c,d} \frac{\partial L}{\partial Z_{c,d}} X_{i+c,j+d}\end{aligned}$$

The above equation can be equivalently written as

$$\frac{\partial L}{\partial K} = X \circledast \frac{\partial L}{\partial Z}$$

Let's consider  $B$ . Since  $B$  here is a scalar which is directly added to the result of convolution of  $X$  and  $K$ , so

$$\frac{\partial Z_{c,d}}{\partial B} = 1$$

Hence

$$\frac{\partial L}{\partial B} = \sum_{c,d} \frac{\partial Z_{c,d}}{\partial B} \frac{\partial L}{\partial Z_{c,d}} = \sum_{c,d} \frac{\partial L}{\partial Z_{c,d}}$$

Let  $\bar{K}$  represent the kernel obtained by rotating  $K$  by  $180^\circ$ . Let  $\frac{\partial L}{\partial \bar{Z}}$  represent the matrix obtained by padding the matrix  $\frac{\partial L}{\partial Z}$  with  $\text{size}(K) - 1$  layers. Then

$$\frac{\partial L}{\partial X} = \frac{\partial L}{\partial Z} \circledast \bar{K}$$

The proof of the above result is left as an exercise for the reader.