Notes on Convolutional Neural Networks

Resources taken from the lecture notes of Andrew Ng and the book Deep Learning by Goodfellow, I., Bengio, Y., & Courville, A.[*]

Md. Azizul Hakim[†]
Undergraduate Student
Dept. of Computer Science & Engineering
University of Liberal Arts Bangladesh (ULAB)

October 20, 2022

Convolutional networks, also known as convolutional neural networks or CNNs, are a specialized kind of neural network for processing data that has a known, grid-like topology.

Convolutional networks are simply neural networks that use convolution in place of general matrix multiplication in at least one of their layers.

1 The Convolution Operation

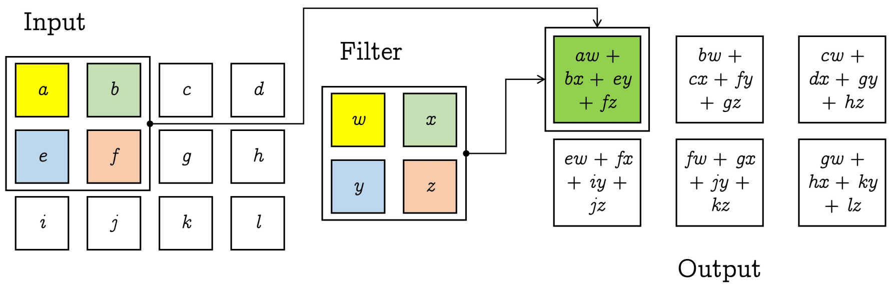

Figure 1: The convolution operation: iteration 1

[*]Supervised by: Dr. Muhammad Abul Hasan

[†]azizul.hakim.cse@ulab.edu.bd

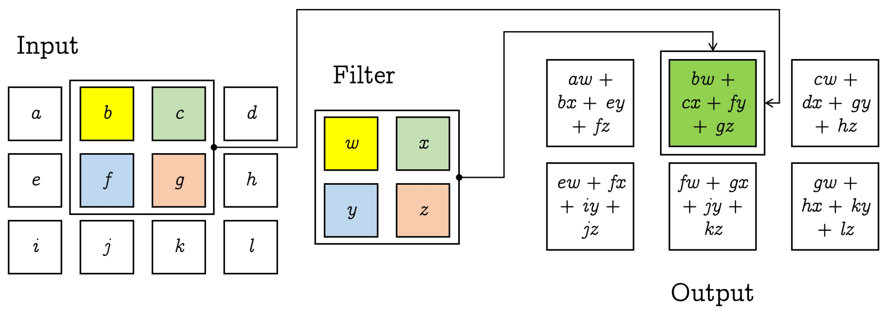

Figure 2: The convolution operation: iteration 2

Convolution Operation: We place the first element of the filter on top of the first element of the image. It overlaps the filter with a portion of the image. Then we perform element wise multiplication and get the summation of all the multiplied values. It becomes the first element of the output image. We perform the same task by stepping right and below and create a new output image.

Convolution operation is one of the fundamental building blocks of a convolutional neural network. This operation can be well-demonstrated using an example of edge detection. It is discussed in the Section 2.

2 Edge Detection

Let's take a sample image as example,

$$\begin{bmatrix} 10 & 10 & 10 & 0 & 0 & 0 \\ 10 & 10 & 10 & 0 & 0 & 0 \\ 10 & 10 & 10 & 0 & 0 & 0 \\ 10 & 10 & 10 & 0 & 0 & 0 \\ 10 & 10 & 10 & 0 & 0 & 0 \\ 10 & 10 & 10 & 0 & 0 & 0 \end{bmatrix}$$

The above gray scale image has a one vertical line at the very center of the image. The color of the left half of the image can be thought as white and the color of the right half of the image can be thought as gray.

The two most common edge detecting filter kernels are: Sobel filter & Scharr filter.

$$\text{The Sobel filter: } \begin{bmatrix} 1 & 0 & -1 \\ 2 & 0 & -2 \\ 1 & 0 & -1 \end{bmatrix} \;\&\; \begin{bmatrix} 1 & 2 & 1 \\ 0 & 0 & 0 \\ -1 & -2 & -1 \end{bmatrix}.$$

And, the Scharr filter: $\begin{bmatrix} 3 & 0 & -3 \\ 10 & 0 & -10 \\ 3 & 0 & -3 \end{bmatrix}$ & $\begin{bmatrix} 3 & 10 & 3 \\ 0 & 0 & 0 \\ -3 & -10 & -3 \end{bmatrix}$.

The convolution operation over the image with a filter kernel can produce an edge detected version of the original image.

2.1 Vertical Edge Detection

The filter kernel to detect vertical edges is $\begin{bmatrix} 1 & 0 & -1 \\ 1 & 0 & -1 \\ 1 & 0 & -1 \end{bmatrix}$.

We can detect vertical edges by performing convolution operation in the taken image with the filter kernel. The operation would be as follows,

$$\begin{bmatrix} 10 & 10 & 10 & 0 & 0 & 0 \\ 10 & 10 & 10 & 0 & 0 & 0 \\ 10 & 10 & 10 & 0 & 0 & 0 \\ 10 & 10 & 10 & 0 & 0 & 0 \\ 10 & 10 & 10 & 0 & 0 & 0 \\ 10 & 10 & 10 & 0 & 0 & 0 \end{bmatrix} * \begin{bmatrix} 1 & 0 & -1 \\ 1 & 0 & -1 \\ 1 & 0 & -1 \end{bmatrix} = \begin{bmatrix} 0 & 30 & 30 & 0 \\ 0 & 30 & 30 & 0 \\ 0 & 30 & 30 & 0 \\ 0 & 30 & 30 & 0 \end{bmatrix}$$

2.2 Horizontal Edge Detection

The filter kernel to detect horizontal edges is $\begin{bmatrix} 1 & 1 & 1 \\ 0 & 0 & 0 \\ -1 & -1 & -1 \end{bmatrix}$.

We can detect horizontal edges by performing convolution operation in the taken image with the filter kernel. The operation would be as follows,

$$\begin{bmatrix} 10 & 10 & 10 & 0 & 0 & 0 \\ 10 & 10 & 10 & 0 & 0 & 0 \\ 10 & 10 & 10 & 0 & 0 & 0 \\ 10 & 10 & 10 & 0 & 0 & 0 \\ 10 & 10 & 10 & 0 & 0 & 0 \\ 10 & 10 & 10 & 0 & 0 & 0 \end{bmatrix} * \begin{bmatrix} 1 & 1 & 1 \\ 0 & 0 & 0 \\ -1 & -1 & -1 \end{bmatrix} = \begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix}$$

Examine the output image. As the image has no horizontal edges, we didn't find any edges in the output image as well!

2.3 Vertical and Horizontal Edge Detection

Let's take a new image example so that it covers both vertical and horizontal edges. The following image has both vertical and horizontal edges.

$$\begin{bmatrix} 10 & 10 & 10 & 0 & 0 & 0 \\ 10 & 10 & 10 & 0 & 0 & 0 \\ 10 & 10 & 10 & 0 & 0 & 0 \\ 0 & 0 & 0 & 10 & 10 & 10 \\ 0 & 0 & 0 & 10 & 10 & 10 \\ 0 & 0 & 0 & 10 & 10 & 10 \end{bmatrix}$$

Let's say we want to detect *horizontal edges* on the above image. So, the convolution operation for the task would be as follows:

$$\begin{bmatrix} 10 & 10 & 10 & 0 & 0 & 0 \\ 10 & 10 & 10 & 0 & 0 & 0 \\ 10 & 10 & 10 & 0 & 0 & 0 \\ 0 & 0 & 0 & 10 & 10 & 10 \\ 0 & 0 & 0 & 10 & 10 & 10 \\ 0 & 0 & 0 & 10 & 10 & 10 \end{bmatrix} * \begin{bmatrix} 1 & 1 & 1 \\ 0 & 0 & 0 \\ -1 & -1 & -1 \end{bmatrix} = \begin{bmatrix} 0 & 0 & 0 & 0 \\ 30 & 10 & -10 & -30 \\ 30 & 10 & -10 & -30 \\ 0 & 0 & 0 & 0 \end{bmatrix}$$

Similarly, we can detect *vertical edges* from the image as well. The convolution operation for the task would be as follows:

$$\begin{bmatrix} 10 & 10 & 10 & 0 & 0 & 0 \\ 10 & 10 & 10 & 0 & 0 & 0 \\ 10 & 10 & 10 & 0 & 0 & 0 \\ 0 & 0 & 0 & 10 & 10 & 10 \\ 0 & 0 & 0 & 10 & 10 & 10 \\ 0 & 0 & 0 & 10 & 10 & 10 \end{bmatrix} * \begin{bmatrix} 1 & 0 & -1 \\ 1 & 0 & -1 \\ 1 & 0 & -1 \end{bmatrix} = \begin{bmatrix} 0 & 30 & 30 & 0 \\ 0 & 10 & 10 & 0 \\ 0 & -10 & -10 & 0 \\ 0 & -30 & -30 & 0 \end{bmatrix}$$

3 Padding

Padding is the process of expanding an image. We use *zero padding* most commonly. It adds layers of zeros around the border of an image.

Let's say, we have this image: $\begin{bmatrix} 1 & 3 \\ 4 & 2 \end{bmatrix}$.

A zero padded version of this image would be: $\begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & 1 & 3 & 0 \\ 0 & 4 & 2 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix}$.

Padding resolves two of the very important problems in convolutional neural networks. They are: 1) shrinking output image, and 2) throwing away information from the edges of the image.

The main benefits of padding are:

- It allows you to use a CONV layer without necessarily shrinking the height and width of the volumes. This is important for building deeper networks, since otherwise the height/width would shrink as you go to deeper layers. An important special case is the *same convolution*, in which the height/width is exactly preserved after one layer.
- It helps us keep more of the information at the border of an image. Without padding, very few values at the next layer would be affected by pixels at the edges of an image.

3.1 Valid & Same Convolutions

Valid Convolutions: If we convolve an image without using any padding on the image, this convolution would be called as *valid convolutions.*

If a $n \times n$ image convolves with a $f \times f$ filter, the output image would be $(n - f + 1) \times (n - f + 1)$.

Same Convolutions: If we convolve an image with such a padding so that the output image has the same height and width as the input size, the convolution would be called as *same convolutions.*

If a $n \times n$ image having a padding, p convolves with a $f \times f$ filter, the output image would be $(n + 2p - f + 1) \times (n + 2p - f + 1)$. And, padding, p would be equal to $\frac{f-1}{2}$.

4 Strided Convolution

Let's examine the following operation. Here we have done our usual convolution operation that we have shown in the above examples. But, instead of taking one step to the right (one step to the below as well), we have taken two steps. And, generated the final output image. The stride in this case would be, $s = 2$. We can calculate the size of the output image using equation (1) shown in the Section 5.

$$\begin{bmatrix} 2 & 3 & 7 & 4 & 6 & 2 & 9 \\ 6 & 6 & 9 & 8 & 7 & 4 & 3 \\ 3 & 4 & 8 & 3 & 8 & 9 & 7 \\ 7 & 8 & 3 & 6 & 6 & 3 & 4 \\ 4 & 2 & 1 & 8 & 3 & 4 & 6 \\ 3 & 2 & 4 & 1 & 9 & 8 & 3 \\ 0 & 1 & 3 & 9 & 2 & 1 & 4 \end{bmatrix} * \begin{bmatrix} 3 & 4 & 4 \\ 1 & 0 & 2 \\ -1 & 0 & 3 \end{bmatrix} = \begin{bmatrix} 91 & 100 & 88 \\ 69 & 91 & 117 \\ 44 & 72 & 74 \end{bmatrix}$$

5 Summary of Convolutions

If we want to convolve a $n \times n$ image with a $f \times f$ filter (we are using padding, p and stride, s), the output image size would be,

$$\left\lfloor \frac{n+2p-f}{s} + 1 \right\rfloor \times \left\lfloor \frac{n+2p-f}{s} + 1 \right\rfloor \tag{1}$$

The equation (1) represents the output image size. The image that we have used in the previous section is of 7×7 dimension ($n = 7$). The filter we have used is of 3×3 dimension ($f = 3$). No padding was used ($p = 0$). And, stride was, $s = 2$. Using equation (1), we can calculate the output image size.

$$\begin{aligned} \text{Output image size} &\Rightarrow \left\lfloor \frac{n+2p-f}{s} + 1 \right\rfloor \times \left\lfloor \frac{n+2p-f}{s} + 1 \right\rfloor \\ &\Rightarrow \left\lfloor \frac{7+2\times 0-3}{2} + 1 \right\rfloor \times \left\lfloor \frac{7+2\times 0-3}{2} + 1 \right\rfloor \\ &\Rightarrow \left\lfloor \frac{4}{2} + 1 \right\rfloor \times \left\lfloor \frac{4}{2} + 1 \right\rfloor \\ &\Rightarrow 3 \times 3 \end{aligned}$$

6 Convolution Over Volume

If we want to detect features in RGB image, we have to consider convolution over three dimensional volumes. We will have an image of 3 channels and we will need a filter with 3 channels as well. Consider Figure 3.

We have a 6×6 2D image of 3 channels. So, basically, we have a $6 \times 6 \times 3$ volume. And, we have a $3 \times 3 \times 3$ filter to convolve over that 3D image. From the filter, we are seeing that we have basically 27 values. Similar to the 2D convolution, we can put that filter on top of the first element of the

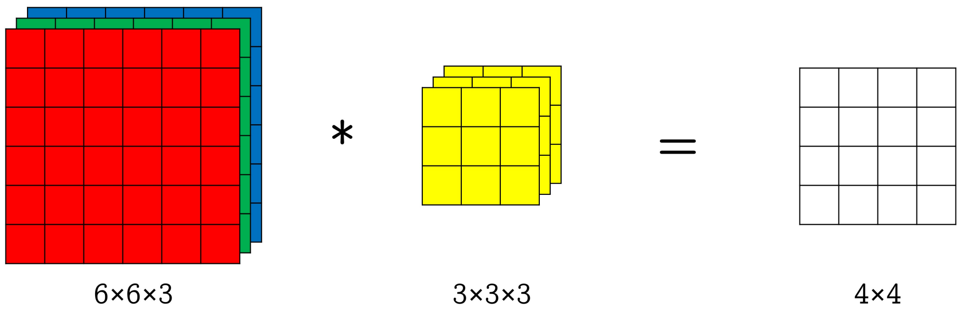

Figure 3: Convolutions on RGB image

3D image (Figure 4). It will overlap the 3×3 portion of the first matrix. And, as it also has 3 channels, it will automatically overlaps over to all the 3 channels of the original image to match all the 27 values. Then, similar to the 2D convolution, we can perform element wise multiplication and get the summation of those values. This will give us the first element value of the output image. We continue the operation by taking steps to the right and below to tabulate the 4×4 output image. Figure 4 summarizes the operation.

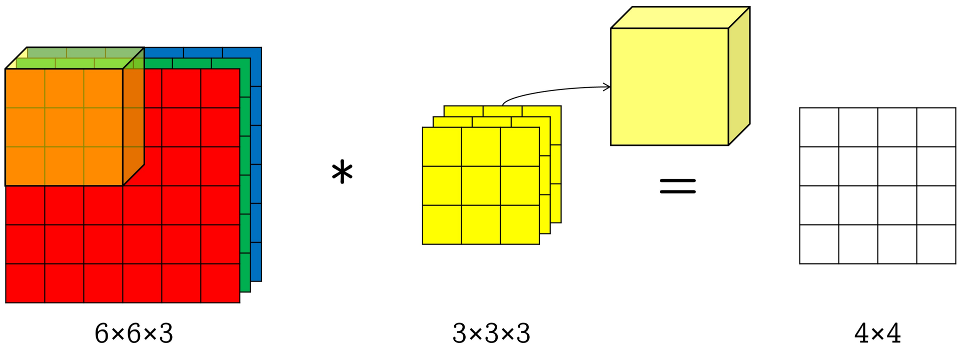

Figure 4: The convolution operation

If we convolve a $(n \times n \times n_c)$ image with a $(f \times f \times n_c)$ filter, we will get a $(n - f + 1) \times (n - f + 1)$ output image.

6.1 Use of Multiple Filters

We can use multiple filters over an RGB image. Consider Figure 5, we have two $3 \times 3 \times 3$ filters. If we convolve it with a $6 \times 6 \times 3$ image, we will get a $4 \times 4 \times 2$ volume.

So, to summarize, if we convolve a $(n \times n \times n_c)$ image with n'_c number of $(f \times f \times n_c)$ filters, we will get a $(n - f + 1) \times (n - f + 1) \times n'_c$ output image volume. Here, n_c = number of channels.

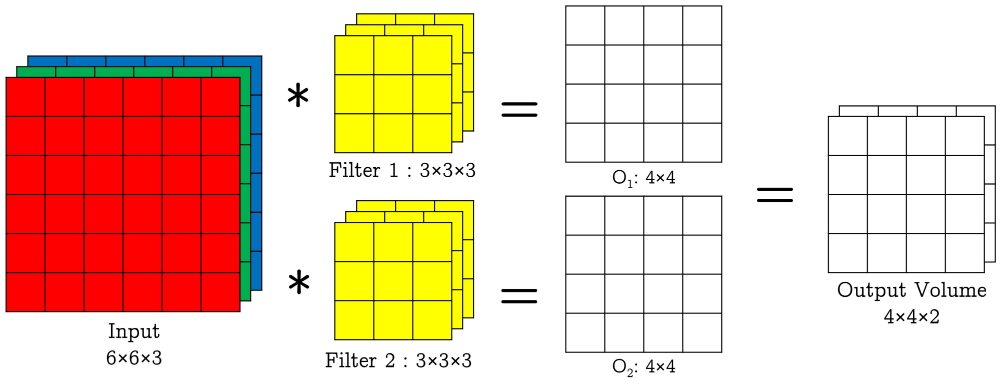

Figure 5: The convolution operation with multiple filters

7 One Layer of a Convolutional Network

Get back to the Figure 5, we have got two 4×4 output matrices (O_1, O_2) after performing convolution with the original image. Now, let's do some more stuff.

First, add a bias (a real number) to O_1, O_2. Then, apply a *ReLU* non-linearity to it. So, we are going to get, $ReLU(O_1 + b)$ and $ReLU(O_2 + b)$. And, finally, after stacking the outputs, we will get the final $4 \times 4 \times 2$ output volume. This computational process of getting a $4 \times 4 \times 2$ output from the $6 \times 6 \times 3$ input is one layer of convolutional neural network.

7.1 Analogy with a Neural Network

Recall the neural network activation at layer 1 of a neural network (Figure 6),

$$z^{[1]} = W^{[1]} \cdot a^{[0]} + b^{[1]}$$
$$a^{[1]} = g(z^{[1]})$$

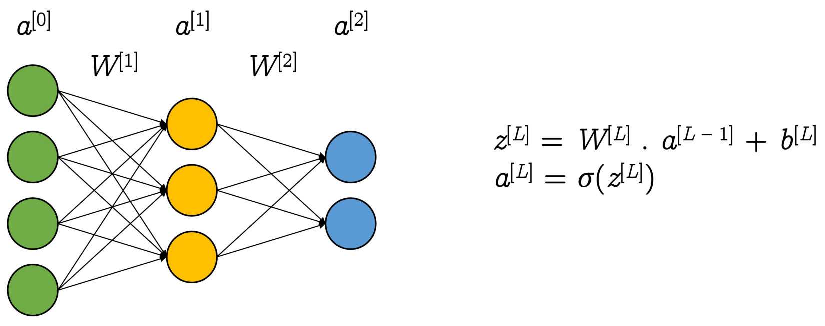
$a^{[0]}$
$a^{[1]}$
$a^{[2]}$
$W^{[1]}$
$W^{[2]}$
$z^{[L]} = W^{[L]} \, . \, a^{[L-1]} + b^{[L]}$
$a^{[L]} = \sigma(z^{[L]})$

Figure 6: Neural network representation

Here, $W^{[1]}$ is the layer 1 weight matrix, $a^{[0]}$ is the input layer which is equals to X, and $b^{[1]}$ is the layer 1 bias unit. $g()$ is the sigmoid activation function. Let's compare the representation with the Figure 5.

The input volume acts similarly to the $a^{[0]}$ or the input layer of the neural network. Filter 1 and filter 2 combinedly acts as the weight, $W^{[1]}$. And, the output volume is activation of layer 1, $a^{[1]}$. The process of getting the output volume is already described in the previous section. It is completely similar to the sigmoid activation of a neural network.

7.2 Notational Convention

If layer l is a convolution layer:

$$\begin{aligned}
\text{Input} &: n_H^{[l-1]} \times n_W^{[l-1]} \times n_c^{[l-1]} \\
\text{Output} &: n_H^{[l]} \times n_W^{[l]} \times n_c^{[l]} \\
f^{[l]} &= \text{filter size} \\
p^{[l]} &= \text{padding} \\
s^{[l]} &= \text{stride} \\
n_c^{[l]} &= \text{number of filters} \\
\text{Each filter is} &: f^{[l]} \times f^{[l]} \times n_c^{[l-1]} \\
\text{Activations} &: a^{[l]} \rightarrow n_H^{[l]} \times n_W^{[l]} \times n_c^{[l]} \\
\text{Weights} &: f^{[l]} \times f^{[l]} \times n_c^{[l-1]} \times n_c^{[l]} \\
\text{Bias} &: n_c^{[l]}
\end{aligned}$$

8 Simple Deep CNN Example

Let's take the example in Figure 7. In the layer 0 or the input layer ($a^{[0]}$ or X), it has a $39 \times 39 \times 3$ image. So, according to our notational convention, $n_H^{[0]} = n_W^{[0]} = 39, n_c^{[0]} = 3$. To get the first output at layer 1, we have set a filter size of 3, stride size of 1, paading size of 0, and 10 number of filters[1]. So, according to the notational convention, $f^{[1]} = 3, s^{[1]} = 1, p^{[1]} = 0, n_c^{[1]} = 10$.

Using the equation (1), we will get the output at layer 1, $\lfloor \frac{39+2\times0-3}{1} + 1 \rfloor \times \lfloor \frac{39+2\times0-3}{1} + 1 \rfloor \Rightarrow 37 \times 37 \times 10$.

So, in the layer 1, $n_H^{[0]} = n_W^{[0]} = 37, n_c^{[0]} = 10$. To get the output at layer 2, we have set the hyperparameters, $f^{[2]} = 5, s^{[2]} = 2, p^{[2]} = 0, n_c^{[2]} = 20$. And, we will get the output using equation (1), $a^{[2]} \Rightarrow 17 \times 17 \times 20$. As we have set a stride greater than 1, the dimension shrunk much faster.

Similarly, we have got the layer 3 output, $a^{[3]} \Rightarrow 7 \times 7 \times 40$. It's the final output volume for the example. Finally, to complete the ConvNet, feed the layer 3 output into a logistic regression unit, or a Softmax unit after flattening the output into a $7 \times 7 \times 40 = 1960$ units long vector (Figure 8).

We have seen convolution type of convolutional layer in this example. There can be 3 types of layer in a convolutional network:

- Convolution layer ($CONV$)
- Pooling layer ($POOL$)
- Fully connected layer (FC)

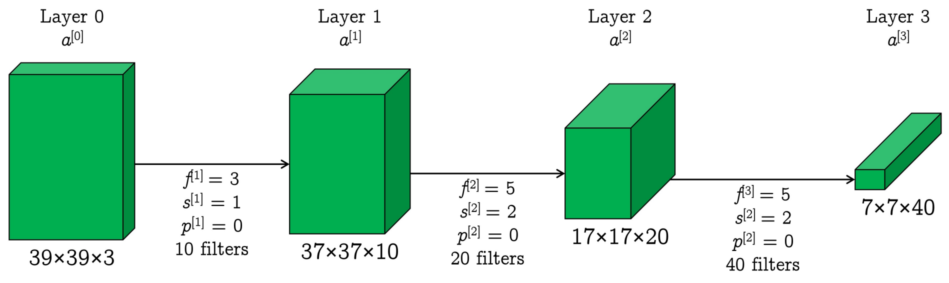

Figure 7: Simple deep CNN example

[1]These are hyperparameters for a ConvNet. By definition, hyperparameters are parameters whose values are set before starting the model training process.

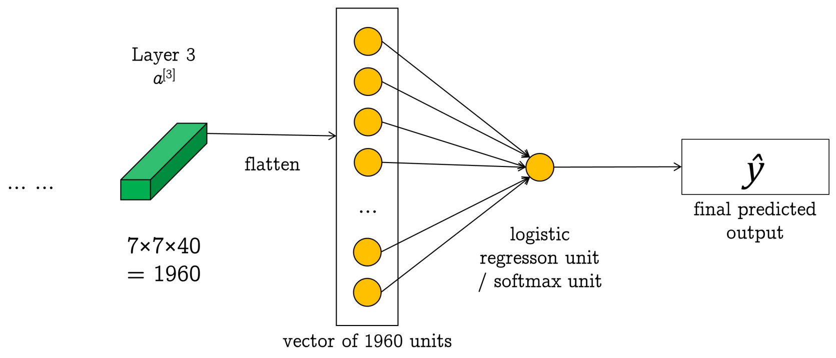

Figure 8: Simple deep CNN example: feeding into neural network

9 Pooling Layers

The pooling (POOL) layer reduces the height and width of the input. It helps reduce computation, as well as helps make feature detectors more invariant to its position in the input.

ConvNets often use pooling layers to reduce the size of the representation, to speed the computation, as well as make some of the features that detects a bit more robust.

9.1 Max Pooling

Max pooling is a pooling operation that selects the maximum element from the region of the feature map covered by the filter. Thus, the output after max-pooling layer would be a feature map containing the most prominent features of the previous feature map.

It slides an $f \times f$ window over the input and stores the max value of the window in the output. From the following 4×4 image, we performed max pooling considering a filter size, $f = 2$, and stride size, $s = 2$. The operation takes the maximum value from a 2×2 region. Using the equation (1), we get the output size would be, 2×2.

$$\begin{bmatrix} 1 & 3 & 2 & 1 \\ 2 & 9 & 1 & 1 \\ 1 & 3 & 2 & 3 \\ 5 & 6 & 1 & 2 \end{bmatrix} \xrightarrow[f=2, s=2]{\text{max pooling}} \begin{bmatrix} 9 & 2 \\ 6 & 3 \end{bmatrix}$$

In the following example, we have a 5×5 image. If we perform max pooling considering filter, $f = 3$, stride, $s = 1$ (zero padding), we will get a

$\frac{5+0-3}{1} \times \frac{5+0-3}{1}$ or 3×3 output. We will perform max pooling in the 3×3 window of the image and take one step left and one step below.

$$\begin{bmatrix} 1 & 3 & 2 & 1 & 3 \\ 2 & 9 & 1 & 1 & 5 \\ 1 & 3 & 2 & 3 & 2 \\ 8 & 3 & 5 & 1 & 0 \\ 5 & 6 & 1 & 2 & 9 \end{bmatrix} \xrightarrow[f=3, s=1]{\text{max pooling}} \begin{bmatrix} 9 & 9 & 5 \\ 9 & 9 & 5 \\ 8 & 6 & 9 \end{bmatrix}$$

9.2 Average Pooling

Average pooling is not very commonly used pooling. It performs the same operation stated in the earlier section. It takes the average of that 2×2 window instead of taking the maximum value.

$$\begin{bmatrix} 1 & 3 & 2 & 1 \\ 2 & 9 & 1 & 1 \\ 1 & 3 & 2 & 3 \\ 5 & 6 & 1 & 2 \end{bmatrix} \xrightarrow[f=2, s=2]{\text{average pooling}} \begin{bmatrix} 3.75 & 1.25 \\ 4 & 2 \end{bmatrix}$$

10 Deep CNN Example

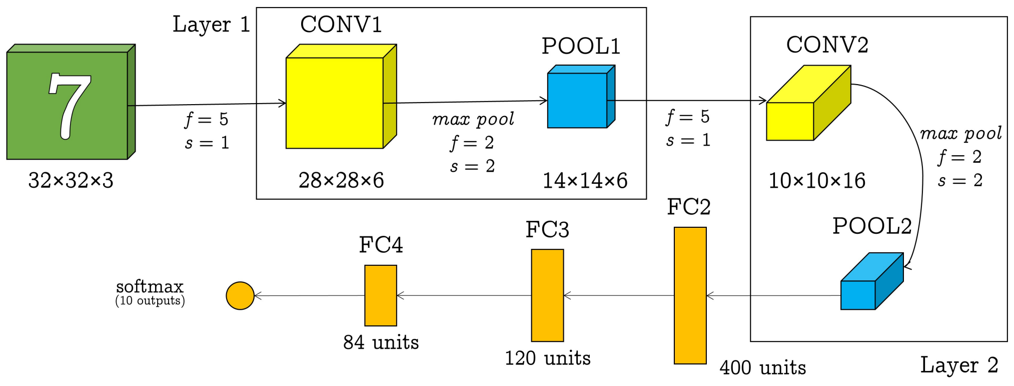

Figure 9: Deep CNN example