Convolutional Neural Networks

A Convolutional Neural Network (CNN) is a class of deep neural network which is typically employed to classify images. Whereas conventional neural networks utilise only fully-connected layers, CNNs make use of several types: convolutional, pooling and fully-connected layers. In the conventional neural network, the spatial structure of the input data is not considered, and image pixels (elements in an array) spatially far apart are treated identically to those that are close together. CNNs take advantage of the spatial structure of an image, applying filters to local regions of spatially-neighbouring pixels in the input image to enable the identification of spatial patterns such as lines, edges and shapes. In this fashion, CNNs are capable of accurately classifying images at a much lower computational cost than the conventional neural network.

1 Convolutional Layers

Convolution describes an operation in which a small array of numbers, known as a kernel, is applied across a larger input array of numbers. An element-wise product between the kernel and a region of the input array (of the same dimensionality) is computed and the results summed to produce the value of an element in another array called the feature map.

Considering Figure 1, we see that a 3×3 kernel is applied left-to-right, top-to-bottom to a 5×5 input array. The first element of the feature map is calculated as follows:

$$(1 \times 1) + (2 \times 0) + (1 \times 1) + (2 \times 0) + (0 \times 1) + (0 \times 0) + (1 \times 1) + (0 \times 0) + (2 \times 1) = 5$$

Ignoring the zeros, the second feature map element is

$$(2 \times 1) + (1 \times 1) = 3$$

and the final feature map element

$$(2 \times 1) + (2 \times 1) + (1 \times 1) + (2 \times 1) = 7$$

A feature map can be pictured as an array in which each element corresponds to the output of a neuron. The neuron output is given by

$$\sum_{i=0}^{a} \sum_{j=0}^{b} w_{ij} x_{ij} + b \tag{1}$$

where a and b represent the dimensionality of the kernel, w_{ij} is an element from an array of shared weights (the kernel) of dimensions $(a \times b)$, x_{ij} is an element from the relevant

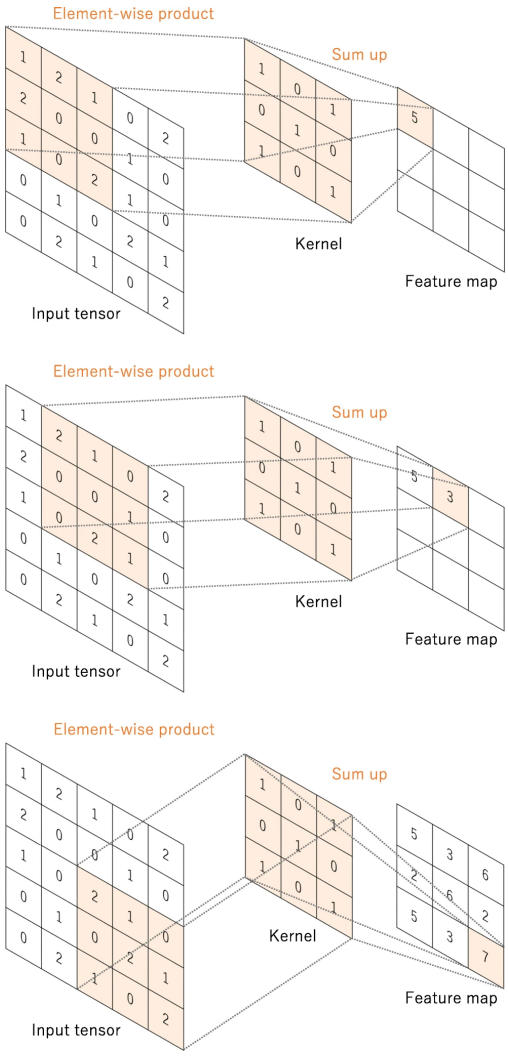

Figure 1: A visualisation of how a kernel is implemented to calculate the elements of a feature map. An element-wise product between the 3×3 kernel and a 3×3 region (local receptive field) of the input array is computed and the results summed. This value corresponds to an element of the feature map. The kernel is passed left-to-right, top-to-bottom over the input array, using a predefined stride length until all the elements of the feature map are computed. Taken from Yamashita et al, 2018.

$(a \times b)$ region of the input array and b is the shared bias. We say that the region of the input array involved in the element-wise product with the kernel is the local receptive field of the corresponding feature map neuron.

In the operations demonstrated in Figure 1, the feature map is reduced in size compared to the original input array. When we operate on each 3 x 3 local receptive field of the input array with a 3 x 3 kernel, we can imagine overlaying the kernel on the input array and summing the products of the overlapping elements. The centre of the kernel does not overlap with the elements that lie around the 'edges' of the input array, resulting in a feature map with reduced dimensions. One way to preserve the dimensions of the feature map is to use a technique known as zero padding (Figure 2). Rows and columns of zeros are added around the outside of the input array in order to enable the centre of the kernel to overlap with the elements that lie on the 'edges' of the original input array. The element-wise product computed in Figure 2 is

$$(0 \times 1) + (0 \times 0) + (0 \times 1) + (0 \times 0) + (1 \times 1) + (2 \times 0) + (0 \times 1) + (2 \times 0) + (0 \times 1) = 1$$

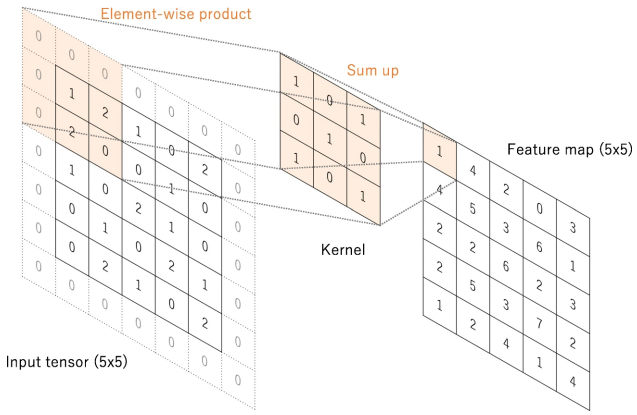

Figure 2: An illustration of the zero padding technique. Rows and columns of zeros are added around the 'edges' of the input array, so that the dimensions of the original input array are preserved. Taken from Yamashita et al, 2018.

The stride length specifies the number of elements the kernel is moved along the input array between successive computations of feature map elements (neuron outputs). In Figures 1 and 2, a stride length of one has been employed. It is obvious that as stride length is increased, the dimensions of the feature map are reduced.

Since each feature map utilises a single kernel to undertake its computations, the convolution procedure makes use of shared weights and biases (see equation (1)). Weight sharing is an important characteristic of CNNs. If a feature map identifies a particular feature in an area of the input array using a certain set of weights, then applying the kernel across the whole array means that it will look for that same feature everywhere in the array. In other words, weight sharing makes the convolution operation capable of dealing with translational invariance in the input array. Another key advantage of utilising shared weights instead of a fully connected input layer is that the number of trainable parameters in the network is greatly reduced.

The operations described up to now in the convolutional layers are linear. To enable the network to account for complex, non-linear behaviour we add non-linearity into the model. This is typically done by applying a non-linear activation function to the feature map (adding an activation layer). The result is the activation map, which will be an array with the same dimensionality as the feature map, since each element of the activation map is simply the result of the activation function applied to the corresponding element in the feature map. An activation map element can be thought of as a neuron with the output

$$\sigma\left(\sum_{i=0}^{a}\sum_{j=0}^{b} w_{ij}x_{i,j} + b\right) \tag{2}$$

where σ is a non-linear function (typically ReLU).

2 Pooling Layers

A pooling layer is normally used immediately after a convolutional layer and serves to reduce the dimensions of the activation maps, therefore simplifying the output from a convolutional layer. Each neuron in the pooling layer will summarise a region of neurons in the activation map. For example, each neuron in the pooling layer could summarise a region of 3 x 3 neurons in the activation map.

2.1 Max Pooling

One popular pooling technique is known as Max Pooling, in which a neuron in the pooling layer chooses the maximum output from amongst the region of neurons in the activation map it is responsible for summarising, and discards the other outputs. In Figure 3, each neuron in the max pooling layer is responsible for extracting the maximum output from a 2 x 2 region of neurons in the activation map. We can say that this max pooling filter operates with a stride length of two. Often, features of interest in the input array will take on the highest values (e.g. an array of pixel data from an image), which subsequently feedforward to the highest values in the feature maps (and then the activation maps). This explains why in Max Pooling we are only interested in extracting the highest output from a region.

2.2 Average Pooling

In Average Pooling, a neuron in the pooling layer takes the mean value of the outputs from the region of neurons it is responsible for summarising. Therefore, in contrast to Max Pooling, information about the 'less important' outputs in a region is retained for later use.

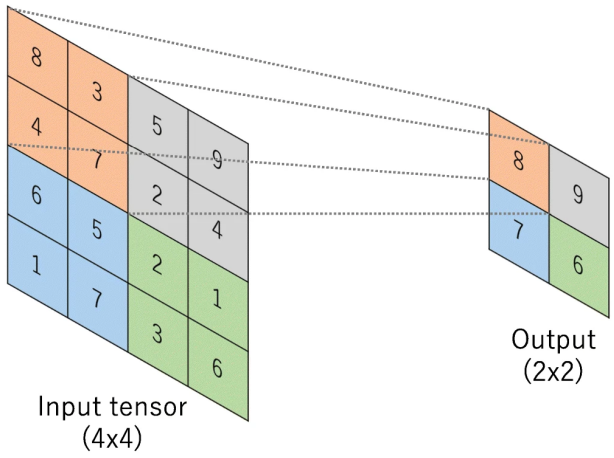

Figure 3: A visualisation of Max Pooling. Each neuron in the pooling layer summarises a region of four neurons in the activation map by selecting the maximum neuron output. Taken from Yamashita et al, 2018.

3 Fully Connected Layers

The final layer (or layers) of the CNN is typically a fully-connected layer (dense layer). That is, every neuron in the pooling layers is connected to every neuron in the output layer by a trainable weight. The final, fully-connected layer will usually contain the same number of neurons as there are number of classes, i.e., for an n class classification task there will be n output neurons in the final layer. If there is more than one fully-connected layer, the non-linear ReLU activation function is often chosen for any that are hidden (see Neural Networks). The activation function selected for neurons in the final fully-connected layer (the output layer) depends on the task. Similarly, the cost function employed to train the network is task dependent.

4 CNN Architecture

Figure 4 provides a visualisation of a CNN. An image is input into the network, typically as an array of values representing pixel intensity. The convolution layers come first: multiple and unique kernels are simultaneously applied to the input array; element-wise products between regions of the input and each kernel are computed and summed, and correspond to elements (neurons) in the feature maps (each kernel outputs to its own feature map). The feature maps are followed by activation maps. Next come the pooling layers, which downsample the activation maps whilst extracting the most important information. For each kernel in the convolutional layers, these operations run side by side before being all brought back together with at least one fully-connected layer. The output of the network is then compared to the desired (actual) output with an appropriately selected cost function. In this fashion, the input information is forward-propagated through the CNN. Finally, the backpropagation algorithm is employed to update the weights in the fully-connected layers and the shared weights in the kernels, before forward propagation commences again.

More advanced CNNs will often use several, alternating convolutional and pooling layers (Figure 5). The first convolutional layer will extract simple features, such as lines or edges. The subsequent convolutional layer that acts on the output from the first pooling layer will extract more abstract, complex features involving combinations of features identified in the first layer. Eventually, very deep layers will be extracting highly complex features from the input image. In a CNN the convolutional layers identify and extract spatial features from the

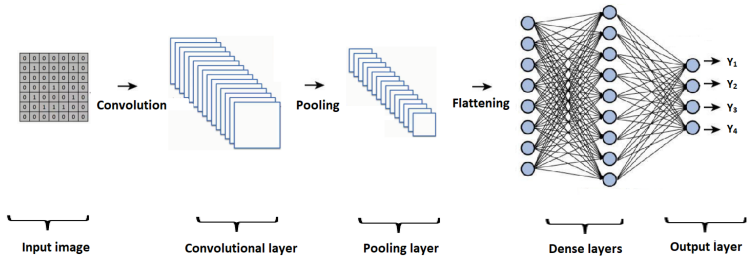

Figure 4: The architecture of a CNN. Taken from Maeda-Gutiérrez et al, 2020.

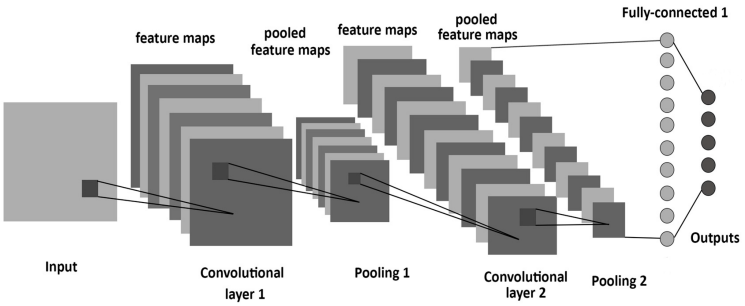

Figure 5: A CNN with two convolutional and pooling layers each. Taken from Albelwi & Mahmood, 2017.

input data. The fully connected layers take these features and use them to make predictions, as in a standard neural network.

References

Albelwi, S., Mahmood, A. (2017). *A Framework for Designing the Architectures of Deep Convolutional Neural Networks*. Entropy, 19(6).

Maeda-Gutiérrez, V., et al. (2020). *Comparison of Convolutional Neural Network Architectures for Classification of Tomato Plant Diseases*. Applied Sciences, 10(4).

Nielsen, M. (2015). *Neural Networks and Deep Learning*. Determination Press.

Yamashita, R., et al. (2018). *Convolutional neural networks: an overview and application in radiology*. Insights into Imaging, (9).