Universität Hamburg

Convolutional Neural Networks vs Feedforward Neural Networks

Report for Module

"Seminar Computer Vision"

supervised by

Noha Sarhan

Simone Frintrop

Adrian Miska

5miska@informatik.uni-hamburg.de
Matriculation number: 6794673
Course of studies: B.Sc. Software-System-Entwicklung

Table of contents

1. Introduction

This report covers the fundamentals of Convolutional Neural Networks (CNN) by exploring their origin and the differences to traditional neural networks. The contents and equations are based on (Gonzales & Woods, 2018).

A neural network is generally used for *classification* which is a supervised learning task of associating an input pattern to one of many predetermined classes. To *train* a neural network is the process of applying the input pattern, computing the delta between actual and wanted activation and changing the weights accordingly. After training it gets tested with a separate test set to evaluate the performance. In Convolutional Neural Networks, convolutional filters are trained in this process.

Convolutional Neural Networks have been discussed at least since 1989 (LeCun, et al. 1989), but gained greater popularity when they showed remarkable results in the 2012 ImageNet Challenge (Krizhevsky, et al. 2012). As such they are a relatively recent movement with a lot of active research going on.

2. Feedforward Neural Networks

Before treating Convolutional Neural Networks, it is necessary to have an understanding of neural networks in general and their basic principles.

2.1 Artificial Neurons

A neural networks consists of layers of *artificial neurons*. The artificial neuron is a mathematical construct used to model computations similar to the propagation of activation signals in biological neurons. It is a refinement of the *Perceptron* (Rosenblatt 1958).

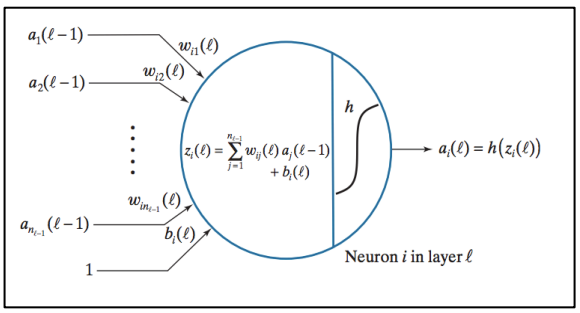

Figure 1 - Model of an artificial neuron (Gonzales & Woods 2018)

The neuron receives a set of input values (activations a_n of the previous layer's neurons) which are each multiplied by an associated weight (w) and then summed. The result of the sum is the net (i.e. total) input z. Regarding the i-th neuron, this gives the formula

$$z_i(l) = \sum_{j=1}^{n_{l-1}} w_{ij}(l)\, a_j(l-1) + b_i(l)$$

where l is the current layer, n_l is the number of neurons in that layer and b is a bias. The output of the neuron is given by

$$a_i(l) = h(z_i(l))$$

where h is the *activation function* which is usually a sigmoid or hyperbolic function. This output is then fed as the input to the next layer of neurons.

2.2 Forward Pass

A forward pass through the network means that an input is fed to the first layer of neurons which then propagate their activation through the network until it arrives at the last layer where the output can be read. This means that the number of neurons in the input layer should correspond to the size of the input vector. The output layer should have as many neurons as there are classes. That way the class can easily be derived by looking at the output neuron with the highest activation.

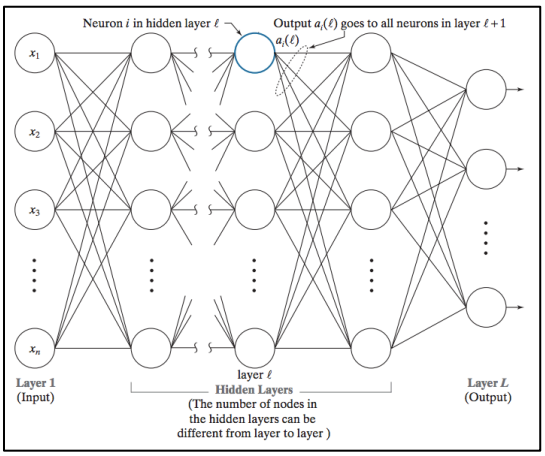

Figure 2 – A fully connected neural network (Gonzales & Woods 2018)

It can usually be assumed that every neuron of each layer is connected to every neuron in the following layer (as shown in figure 2). The layers between the first and last are referred to as *hidden layers*.
The input pattern gets applied to the first layer, where for each neuron the formulas given in section 2.1 are calculated. The activation a of each neuron is then passed as the input for each neuron in the following layer where the same equations apply.

It can be shown that all computations for a forward pass can be done simultaneously for a large set of inputs by reducing the problem to a series of matrix multiplications (see pp. 950-953). This makes computation easier, even more so on specialized hardware.

2.3 Backpropagation

The next step is training the network to output meaningful results. The key to that are the weights and biases of the individual neurons.
For training, labeled data is used so for each input the desired output is known. The error (with respect to a single input vector) of the j-th neuron in the output layer is specified by

$$E_j = \frac{1}{2}(r_j - a_j(L))^2$$

where r_j is the expected output of the neuron and L denotes the last layer. The total output error is then calculated by summing the individual errors of the output neurons:

$$E = \sum_{j=1}^{n_L} E_j = \frac{1}{2}\sum_{j=1}^{n_L}(r_j - a_j(L))^2$$

It is now possible to adjust the weights of the output layer. However, there is no way to compute the error for neurons in the hidden layers yet. The error gradient for an individual neuron j is defined as

$$\partial_j(l) = \frac{\partial E}{\partial z_j(l)}$$

because this is equivalent to the change in the error in terms of the net input $z_j(l)$.
While it goes beyond the scope of this report, it is possible to express this in terms of the next layer. This means that the error can be read from the output layer and fed back through the network to compute the errors for every individual neuron. The weights and bias for every neuron can then be adjusted by applying

$$w_{ij}(l) = w_{ij}(l) - \alpha \frac{\partial E}{\partial w_{ij}(l)}$$

$$b_i(l) = b_i(l) - \alpha \frac{\partial E}{\partial b_i(l)}$$

where α is the learning rate. This process is known as *gradient descent*. The learning rate cannot easily be found as a too small one leads to a diminishing learning performance and a too large one stops us from reaching the minimum error as the process "overshoots".

Analogous to the forward pass, it is again possible to reduce the computations to a series of matrix multiplications. After multiple runs of forward pass and backpropagation one will end up with a network that has its weights and biases set in a way to do meaningful classification.

3. Convolutional Neural Networks

A basic neural network takes an input vector and assigns a label to it. This is not really suitable for image classification, since the extraction of a feature vector from an image has to be done manually. This section discusses how this can be integrated into the learning process.

3.1 Convolutions

To understand how a Convolutional Neural Network operates, the next step is to look at how convolutions work.

A convolution consists of moving a *kernel* over a source image. The kernel is a matrix filled with weights. To *convolve* an image with a kernel means to move the kernel over the image, computing the sum of the kernel weights multiplied with the underlying pixels and using this value as the corresponding pixel for the result. In analogy to biological vision the moving area in which pixels are read is called the *receptive field*.

0.1	0.1	0.2	0.3	1
0.4	0.2	0	0.9	0.9
0.5	0.2	0.4	0.7	0.8
0.3	0	0.4	0.5	0.6
0	0.2	0.3	0.5	0.6

Figure 3 - Example for a 5x5 kernel

3.2 Changes to the algorithms

The weights of a kernel can be treated in a similar matter to the weights of artificial neurons. This holds true because a convolution (denoted as $\star$) is a sum of multiplications described by

$$w \star a_{x,y} = \sum_{j=1}^{n*m} w_j a_j + b = z$$

where w is the kernel filled with $n \times m$ weights, a are the input values from the receptive field and an added bias b. Passing this through an activation function results in

$$a = h(z)$$

Now the same forward pass as discussed in section 2.2 can be applied.
The changes to the equations for backpropagation go beyond the scope of this report but again they involve working from the output towards the input and modifying the weights based on the error.

It should be noted though, that while in a fully connected neural network, a single neuron receives the activations from every single neuron in the previous layer as input, a kernel only receives $n \times m$ inputs. This gives CNNs a sense of locality.

3.3 Architecture

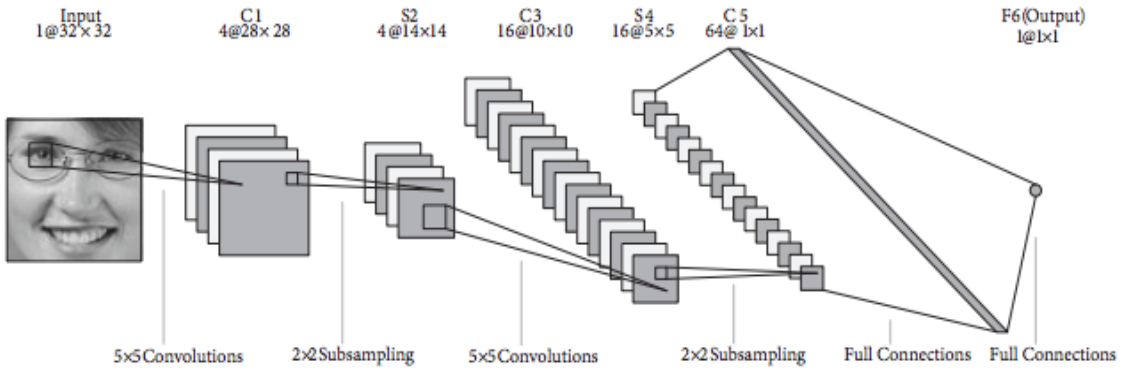

Figure 4 – Example for a CNN architecture as described by (Liew, et al. 2016)

Figure 4 shows an example for a typical CNN architecture. In contrast to a traditional feedforward neural network, the input to a CNN is a matrix of values representing an image rather than an input vector. The image is then subjected to convolution with multiple kernels resulting in multiple *feature maps* (C1 in Figure 4). The feature maps are then subsampled, or pooled, which reduces their resolution (S2 in Figure 4). In the simplest form, pooling can average over a region of pixels, while *max pooling* takes the maximum. The pooled feature maps are then convolved and subsampled again in further convolutional and subsampling layers (C3 and S4 in Figure 4). After the last convolutional layer, the feature maps get

converted into a single vector in a process called vectorization (C5 in Figure 4). This can be done by taking the columns of the two dimensional feature maps and concatenating them. The resulting vector is then fed into a series of fully connected layers which acts as the actual classifier.

When using color images, the architecture fundamentally stays the same but using a three dimensional array as input and convolving it with three dimensional kernels.

4. Conclusion

In conclusion, it can be said that it is only a small step from fully connected neural nets toward CNNs. However, the latter provide the advantage of automatically obtaining meaningful features out of images. When using a standard neural net for image classification, one has to manually find means to extract features out of the images and then train the neural net with these features. With a CNN the images themselves can be fed through the network drastically reducing the effort needed. There is also a good probability that when using a sensible architecture, the trained feature extraction provides better results compared to manually selecting features.

Bibliography

Gonzales, Rafael C., and Richard E. Woods. 2018. *Digital Image Processing, 4th Edition.* Pearson.

Krizhevsky, Alex, Ilya Sutskever, and Geoffrey E Hinton. 2012. "Imagenet classification with deep convolutional neural networks." *Advances in neural information processing systems.* 1097-1105.

LeCun, Yann, Bernhard Boser, John S Denker, Donnie Henderson, Richard E Howard, Wayne Hubbard, and Lawrence D Jackel. 1989. "Backpropagation applied to handwritten zip code recognition." *Neural computation*, 12: 541-551.

Liew, Shan Sung, Mohamed Khalil-Hani, Syafeeza Ahmad Radzi, and Rabia Bakhteri. 2016. "Gender Classification: A Convolutional Neural Network Approach." *Turkish Journal of Electrical Engineering and Computer Sciences*, 03: 1248-1264.

Rosenblatt, Frank. 1958. "The perceptron: A probabilistic model for information storage and organization in the brain." *Psychological Review*, 386-408.