Chapter 10

Convolutional Neural Networks

- Sparse Connectivity
- Shared Weights
- Max-pooling

The convolutional neural network (CNN) is a neural network technology that has profoundly impacted the area of computer vision (CV). Fukushima (1980) introduced the original concept of a convolutional neural network, and LeCun, Bottou, Bengio & Haffner (1998) greatly improved this work. From this research, Yan LeCun introduced the famous LeNet-5 neural network architecture. This chapter follows the LeNet-5 style of convolutional neural network.

Although computer vision primarily uses CNNs, this technology has some application outside of the field. You need to realize that if you want to utilize CNNs on non-visual data, you must find a way to encode your data so that it can mimic the properties of visual data.

CNNs are somewhat similar to the self-organizing map (SOM) architecture that we examined in Chapter 2, "Self-Organizing Maps." The order of the vector elements is crucial to the training. In contrast, most neural networks that are not CNNs or SOMs treat their input data as a long vector of values, and the order that you arrange the incoming features in this vector is irrelevant. For these types of neural networks, you cannot change the order after you

have trained the network. In other words, CNNs and SOMs do not follow the standard treatment of input vectors.

The SOM network arranged the inputs into a grid. This arrangement worked well with images because the pixels in closer proximity to each other are important to each other. Obviously, the order of pixels in an image is significant. The human body is a relevant example of this type of order. For the design of the face, we are accustomed to eyes being near to each other. In the same way, neural network types like SOMs adhere to an order of pixels. Consequently, they have many applications to computer vision.

Although SOMs and CNNs are similar in the way that they map their input into 2D grids or even higher-dimension objects such as 3D boxes, CNNs take image recognition to higher level of capability. This advance in CNNs is due to years of research on biological eyes. In other words, CNNs utilize overlapping fields of input to simulate features of biological eyes. Until this breakthrough, AI had been unable to reproduce the capabilities of biological vision.

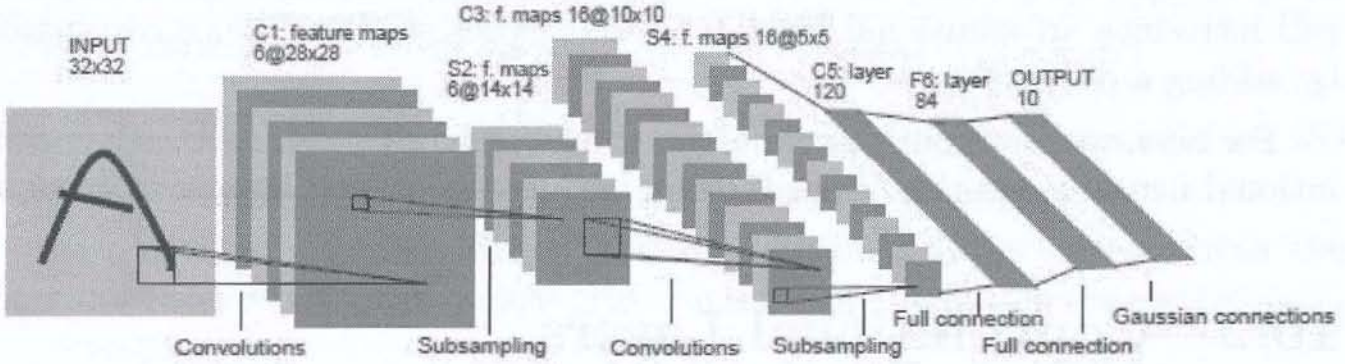
Scale, rotation, and noise have presented challenges in the past for AI computer vision research. You can observe the complexity of biological eyes in the example that follows. A friend raises a sheet of paper with a large number written on it. As your friend moves nearer to you, the number is still identifiable. In the same way, you can still identify the number when your friend rotates the paper. Lastly, your friend creates noise by drawing lines on top of the page, but you can still identify the number. As you can see, these examples demonstrate the high function of the biological eye and allow you to understand better the research breakthrough of CNNs. That is, this neural network has the ability to process scale, rotation, and noise in the field of computer vision.

10.1 LeNET-5

We can use the LeNET-5 architecture primarily for the classification of graphical images. This network type is similar to the feedforward network that we examined in previous chapters. Data flow from input to the output. However, the LeNET-5 network contains several different layer types, as Figure 10.1

illustrates:

Figure 10.1: A LeNET-5 Network (LeCun, 1998)



Several important differences exist between a feedforward neural network and a LeNET-5 network:

- Vectors pass through feedforward networks; 3D cubes pass through LeNET-5 networks.
- LeNET-5 networks contain a variety of layer types.
- Computer vision is the primary application of the LeNET-5.

However, we have also explored the many similarities between the networks. The most important similarity is that we can train the LeNET-5 with the same backpropagation-based techniques. Any optimization algorithm can train the weights of either a feedforward or LeNET-5 network. Specifically, you can utilize simulated annealing, genetic algorithms, and particle swarm for training. However, LeNET-5 frequently uses backpropagation training.

The following three layer types comprise the original LeNET-5 neural networks:

- Convolutional Layers
- Max-pool Layers
- Dense Layers

Other neural network frameworks will add additional layer types related to computer vision. However, we will not explore these additions beyond the LeNET-5 standard. Adding new layer types is a common means of augmenting existing neural network research. Chapter 12, "Dropout and Regularization," will introduce an additional layer type that is designed to reduce overfitting by adding a dropout layer.

For now, we focus our discussion on the layer types associated with convolutional neural networks. We will begin with convolutional layers.

10.2 Convolutional Layers

The first layer that we will examine is the convolutional layer. We will begin by looking at the hyper-parameters that you must specify for a convolutional layer in most neural network frameworks that support the CNN:

- Number of filters
- Filter Size
- Stride
- Padding
- Activation Function/Non-Linearity

The primary purpose for a convolutional layer is to detect features such as edges, lines, blobs of color, and other visual elements. The filters can detect these features. The more filters that we give to a convolutional layer, the more features it can detect.

A filter is a square-shaped object that scans over the image. A grid can represent the individual pixels of a grid. You can think of the convolutional layer as a smaller grid that sweeps left to right over each row of the image. There is also a hyper-parameter that specifies both the width and height of the square-shaped filter. Figure 10.1 shows this configuration in which you see the six convolutional filters sweeping over the image grid:

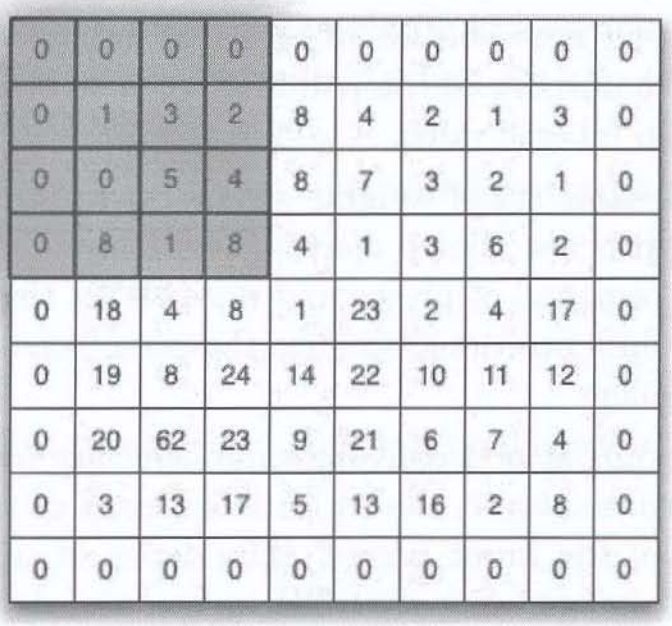
A convolutional layer has weights between it and the previous layer or image grid. Each pixel on each convolutional layer is a weight. Therefore, the number of weights between a convolutional layer and its predecessor layer or image field is the following:

```
[Filter Size] * [Filter Size] * [# of Filters]
```

For example, if the filter size were 5 (5x4) for 10 filters, there would be 250 weights.

You need to understand how the convolutional filters sweep across the previous layer's output or image grid. Figure 10.2 illustrates the sweep:

Figure 10.2: Convolutional Filter



The above figure shows a convolutional filter with a size of 4 and a padding size of 1. The padding size is responsible for the boarder of zeros in the area that the filter sweeps. Even though the image is actually 8x7, the extra padding provides a virtual image size of 9x8 for the filter to sweep across. The stride specifies the number of positions at which the convolutional filters will stop. The convolutional filters move to the right, advancing by the number of cells specified in the stride. Once the far right is reached, the convolutional

filter moves back to the far left, then it moves down by the stride amount and continues to the right again.

Some constraints exist in relation to the size of the stride. Obviously, the stride cannot be 0. The convolutional filter would never move if the stride were set to 0. Furthermore, neither the stride, nor the convolutional filter size can be larger than the previous grid. There are additional constraints on the stride (s), padding (p) and the filter width (f) for an image of width (w). Specifically, the convolutional filter must be able to start at the far left or top boarder, move a certain number of strides, and land on the far right or bottom boarder. Equation 10.1 shows the number of steps a convolutional operator must take to cross the image:

$$steps = \frac{w - f + 2p}{s + 1} \tag{10.1}$$

The number of steps must be an integer. In other words, it cannot have decimal places. The purpose of the padding (p) is to be adjusted to make this equation become an integer value.

We can use the same set of weights as the convolutional filter sweeps over the image. This process allows convolutional layers to share weights and greatly reduce the amount of processing needed. In this way, you can recognize the image in shift positions because the same convolutional filter sweeps across the entire image.

The input and output of a convolutional layer are both 3D boxes. For the input to a convolutional layer, the width and height of the box is equal to the width and height of the input image. The depth of the box is equal to the color depth of the image. For an RGB image, the depth is 3, equal to the components of red, green, and blue. If the input to the convolutional layer is another layer, then it will also be a 3D box; however, the dimensions of that 3D box will be dictated by the hyper-parameters of that layer.

Like any other layer in the neural network, the size of the 3D box output by a convolutional layer is dictated by the hyper-parameters of the layer. The width and height of this box are both equal to the filter size. However, the depth is equal to the number of filters.

10.3 Max-Pool Layers

Max-pool layers downsample a 3D box to a new one with smaller dimensions. Typically, you can always place a max-pool layer immediately following a convolutional layer. Figure 10.1 shows the max-pool layer immediately after layers C1 and C3. These max-pool layers progressively decrease the size of the dimensions of the 3D boxes passing through them. This technique can avoid overfitting (Krizhevsky, Sutskever & Hinton, 2012).

A pooling layer has the following hyper-parameters:

- Spatial Extent (f)
- Stride (s)

Unlike convolutional layers, max-pool layers do not use padding. Additionally, max-pool layers have no weights, so training does not affect them. These layers simply downsample their 3D box input.

The 3D box output by a max-pool layer will have a width equal to Equation 10.2:

$$w_2 = \frac{w_1 - f}{s + 1} \tag{10.2}$$

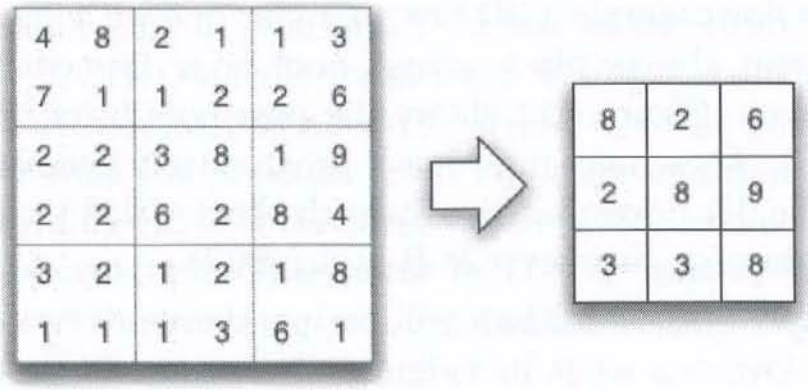
The height of the 3D box produced by the max-pool layer is calculated similarly with Equation 10.3:

$$h_2 = \frac{h_1 - f}{s + 1} \tag{10.3}$$

The depth of the 3D box produced by the max-pool layer is equal to the depth the 3D box received as input.

The most common setting for the hyper-parameters of a max-pool layer are f=2 and s=2. The spatial extent (f) specifies that boxes of 2x2 will be scaled down to single pixels. Of these four pixels, the pixel with the maximum value will represent the 2x2 pixel in the new grid. Because squares of size 4 are replaced with size 1, 75% of the pixel information is lost. Figure 10.3 shows this transformation as a 6x6 grid becomes a 3x3:

Figure 10.3: Max-pooling (f=2,s=2)



Of course, the above diagram shows each pixel as a single number. A grayscale image would have this characteristic. For an RGB image, we usually take the average of the three numbers to determine which pixel has the maximum value.

10.4 Dense Layers

The final layer type in a LeNET-5 network is a dense layer. This layer type is exactly the same type of layer as we've seen before in feedforward neural networks. A dense layer connects every element (neuron) in the previous layer's output 3D box to each neuron in the dense layer. The resulting vector is passed through an activation function. LeNET-5 networks will typically use a ReLU activation. However, we can use a sigmoid activation function; this technique is less common. A dense layer will typically contain the following hyper-parameters:

- Neuron Count
- Activation Function

The neuron count specifies the number of neurons that make up this layer. The activation function indicates the type of activation function to use. Dense layers can employ many different kinds of activation functions, such as ReLU, sigmoid or hyperbolic tangent.

LeNET-5 networks will typically contain several dense layers as their final layers. The final dense layer in a LeNET-5 actually performs the classification. There should be one output neuron for each class, or type of image, to classify. For example, if the network distinguishes between dogs, cats, and birds, there will be three output neurons. You can apply a final softmax function to the final layer to treat the output neurons as probabilities. Softmax allows each neuron to provide the probability of the image representing each class. Because the output neurons are now probabilities, softmax ensures that they sum to 1.0 (100%). To review softmax, you can reread Chapter 4, "Feedforward Neural Networks."

10.5 ConvNets for the MNIST Data Set

In Chapter 6, "Backpropagation Training," we used the MNIST handwritten digits as an example of using backpropagation. In Chapter 10, we present an example about improving our recognition of the MNIST digits, as a deep convolutional neural network. The convolutional network, being a deep neural network, will have more layers than the feedforward neural network seen in Chapter 6. The hyper-parameters for this network are as follows:

- Input: Accepts box of [1,96,96]
- Convolutional Layer: filters=32, filter_size=[3,3]
- Max-pool Layer: [2,2]
- Convolutional Layer: filters=64, filter_size=[2,2]
- Max-pool Layer: [2,2]
- Convolutional Layer: filters=128, filter_size=[2,2]
- Max-pool Layer: [2,2]

- Dense Layer: 500 neurons
- Output Layer: 30 neurons

This network uses the very common pattern to follow each convolutional layer with a max-pool layer. Additionally, the number of filters decreases from the input to the output layer, thereby allowing a smaller number of basic features, such as edges, lines, and small shapes to be detected near the input field. Successive convolutional layers roll up these basic features into larger and more complex features. Ultimately, the dense layer can map these higher-level features into each x-coordinate and y-coordinate of the actual 15 digit features.

Training the convolutional neural network takes considerable time, especially if you are not using GPU processing. As of July 2015, not all frameworks have equal support of GPU processing. At this time, using Python with a Theano-based neural network framework, such as Lasange, provides the best results. Many of the same researchers who are improving deep convolutional networks are also working with Theano. Thus, they promote it before other frameworks on other languages.

For this example, we used Theano with Lasange. The book's example download may have other languages available for this example as well, depending on the frameworks available for those languages. Training a convolutional neural network for digit feature recognition on Theano took less time with a GPU than a CPU, as a GPU helps considerably for convolutional neural networks. The exact amount of performance will vary according to hardware and platform. The accuracy comparison between the convolutional neural network and the regular ReLU network is shown here:

```
Relu:
Best valid loss was 0.068229 at epoch 17.
Incorrect 170/10000 (1.7000000000000002%)
ReLU+Conv:
Best valid loss was 0.065753 at epoch 3.
Incorrect 150/10000 (1.5%)
```

If you compare the results from the convolutional neural network to the standard feedforward neural network from Chapter 6, you will see the convolutional neural network performed better. The convolutional neural network is capable

of recognizing sub-features in the digits to boost its performance over the standard feedforward neural network. Of course, these results will vary, depending on the platform used.

10.6 Chapter Summary

Convolutional neural networks are a very active area in the field of computer vision. They allow the neural network to detect hierarchies of features, such as lines and small shapes. These simple features can form hierarchies to teach the neural network to recognize complex patterns composed of the more simple features. Deep convolutional networks can take considerable processing power. Some frameworks allow the use of GPU processing to enhance performance.

Yann LeCun introduced the LeNET-5, the most common type of convolutional network. This neural network type is comprised of dense layers, convolutional layers and max-pool layers. The dense layers work exactly the same way as traditional feedforward networks. Max-pool layers can downsample the image and remove detail. Convolutional layers detect features in any part of the image field.

There are many different approaches to determine the best architecture for a neural network. Chapter 8, "NEAT, CPPN and HyperNEAT," introduced a neural network algorithm that could automatically determine the best architecture. If you are using a feedforward neural network you will most likely arrive at a structure through pruning and model selection, which we discuss in the next chapter.