Chapter 8

Convolutional Neural Networks

"The soul never thinks without a picture."—Aristotle

8.1 Introduction

Convolutional neural networks are designed to work with grid-structured inputs, which have strong spatial dependencies in local regions of the grid. The most obvious example of grid-structured data is a 2-dimensional image. This type of data also exhibits spatial dependencies, because adjacent spatial locations in an image often have similar color values of the individual pixels. An additional dimension captures the different colors, which creates a 3-dimensional input *volume*. Therefore, the features in a convolutional neural network have dependencies among one another based on spatial distances. Other forms of sequential data like text, time-series, and sequences can also be considered special cases of grid-structured data with various types of relationships among adjacent items. The vast majority of applications of convolutional neural networks focus on image data, although one can also use these networks for all types of temporal, spatial, and spatiotemporal data.

An important property of image data is that it exhibits a certain level of *translation invariance*, which is not the case in many other types of grid-structured data. For example, a banana has the same interpretation, whether it is at the top or the bottom of an image. Convolutional neural networks tend to create similar feature values from local regions with similar patterns. One advantage of image data is that the effects of specific inputs on the feature representations can often be described in an intuitive way. Therefore, this chapter will primarily work with the image data setting. A brief discussion will also be devoted to the applications of convolutional neural networks to other settings.

C. C. Aggarwal, *Neural Networks and Deep Learning*,
https://doi.org/10.1007/978-3-319-94463-0_8

An important defining characteristic of convolutional neural networks is an operation, which is referred to as *convolution*. A convolution operation is a dot-product operation between a grid-structured set of weights and similar grid-structured inputs drawn from different spatial localities in the input volume. This type of operation is useful for data with a high level of spatial or other locality, such as image data. Therefore, convolutional neural networks are defined as networks that use the convolutional operation in at least one layer, although most convolutional neural networks use this operation in multiple layers.

8.1.1 Historical Perspective and Biological Inspiration

Convolutional neural networks were one of the first success stories of deep learning, well before recent advancements in training techniques led to improved performance in other types of architectures. In fact, the eye-catching successes of some convolutional neural network architectures in image-classification contests after 2011 led to broader attention to the field of deep learning. Long-standing benchmarks like *ImageNet* [581] with a top-5 classification error-rate of more than 25% were brought down to less than 4% in the years between 2011 and 2015. Convolutional neural networks are well suited to the process of hierarchical feature engineering with depth; this is reflected in the fact that the deepest neural networks in all domains are drawn from the field of convolutional networks. Furthermore, these networks also represent excellent examples of how biologically inspired neural networks can sometimes provide ground-breaking results. The best convolutional neural networks today reach or exceed human-level performance, a feat considered impossible by most experts in computer vision only a couple of decades back.

The early motivation for convolutional neural networks was derived from experiments by Hubel and Wiesel on a cat's visual cortex [212]. The visual cortex has small regions of cells that are sensitive to specific regions in the visual field. In other words, if specific areas of the visual field are excited, then those cells in the visual cortex will be activated as well. Furthermore, the excited cells also depend on the shape and orientation of the objects in the visual field. For example, vertical edges cause some neuronal cells to be excited, whereas horizontal edges cause other neuronal cells to be excited. The cells are connected using a layered architecture, and this discovery led to the conjecture that mammals use these different layers to construct portions of images at different levels of abstraction. From a machine learning point of view, this principle is similar to that of hierarchical feature extraction. As we will see later, convolutional neural networks achieve something similar by encoding primitive shapes in earlier layers, and more complex shapes in later layers.

Based on these biological inspirations, the earliest neural model was the *neocognitron* [127]. However, there were several differences between this model and the modern convolutional neural network. The most prominent of these differences was that the notion of weight sharing was not used. Based on this architecture, one of the first fully convolutional architectures, referred to as *LeNet-5* [279], was developed. This network was used by banks to identify hand-written numbers on checks. Since then, the convolutional neural network has not evolved much; the main difference is in terms of using more layers and stable activation functions like the ReLU. Furthermore, numerous training tricks and powerful hardware options are available to achieve better success in training when working with deep networks and large data sets.

A factor that has played an important role in increasing the prominence of convolutional neural networks has been the annual *ImageNet* competition [582] (also referred to as *"ImageNet Large Scale Visual Recognition Challenge [ILSVRC]"*). The ILSVRC competition uses the *ImageNet* data set [581], which is discussed in Section 1.8.2 of Chapter 1. Convolutional

neural networks have been consistent winners of this contest since 2012. In fact, the dominance of convolutional neural networks for image classification is so well recognized today that almost all entries in recent editions of this contest have been convolutional neural networks. One of the earliest methods that achieved success in the 2012 *ImageNet* competition by a large margin was *AlexNet* [255]. Furthermore, the improvements in accuracy have been so extraordinarily large in the last few years that it has changed the landscape of research in the area. In spite of the fact that the vast majority of eye-catching performance gains have occurred from 2012 to 2015, the architectural differences between recent winners and some of the earliest convolutional neural networks are rather small at least at a conceptual level. Nevertheless, small details seem to matter a lot when working with almost all types of neural networks.

8.1.2 Broader Observations About Convolutional Neural Networks

The secret to the success of any neural architecture lies in tailoring the structure of the network with a semantic understanding of the domain at hand. Convolutional neural networks are heavily based on this principle, because they use sparse connections with a high-level of parameter-sharing in a domain-sensitive way. In other words, not all states in a particular layer are connected to those in the previous layer in an indiscriminate way. Rather, the value of a feature in a particular layer is connected only to a local spatial region in the previous layer with a consistent set of shared parameters across the full spatial footprint of the image. This type of architecture can be viewed as a domain-aware regularization, which was derived from the biological insights in Hubel and Wiesel's early work. In general, the success of the convolutional neural network has important lessons for other data domains. A carefully designed architecture, in which the relationships and dependencies among the data items are used in order to reduce the parameter footprint, provides the key to results of high accuracy.

A significant level of domain-aware regularization is also available in recurrent neural networks, which share the parameters from different temporal periods. This sharing is based on the assumption that temporal dependencies remain invariant with time. Recurrent neural networks are based on intuitive understanding of temporal relationships, whereas convolutional neural networks are based on an intuitive understanding of spatial relationships. The latter intuition was directly extracted from the organization of biological neurons in a cat's visual cortex. This outstanding success provides a motivation to explore how neuroscience may be leveraged to design neural networks in clever ways. Even though artificial neural networks are only caricatures of the true complexity of the biological brain, one should not underestimate the intuition that one can obtain by studying the basic principles of neuroscience [176].

Chapter Organization

This chapter is organized as follows. The next section will introduce the basics of a convolutional neural network, the various operations, and the way in which they are organized. The training process for convolutional networks is discussed in Section 8.3. Case studies with some typical convolutional neural networks that have won recent competitions are discussed in Section 8.4. The convolutional autoencoder is discussed in Section 8.5. A variety of applications of convolutional networks are discussed in Section 8.6. A summary is given in Section 8.7.

8.2 The Basic Structure of a Convolutional Network

In convolutional neural networks, the states in each layer are arranged according to a spatial grid structure. These spatial relationships are inherited from one layer to the next because each feature value is based on a small local spatial region in the previous layer. It is important to maintain these spatial relationships among the grid cells, because the convolution operation and the transformation to the next layer is critically dependent on these relationships. Each layer in the convolutional network is a 3-dimensional grid structure, which has a *height*, *width*, and *depth*. The depth of a layer in a convolutional neural network should not be confused with the depth of the network itself. The word "depth" (when used in the context of a single layer) refers to the number of *channels* in each layer, such as the number of primary color channels (e.g., blue, green, and red) in the input image or the number of feature maps in the hidden layers. The use of the word "depth" to refer to both the number of feature maps in each layer as well as the number of layers is an unfortunate overloading of terminology used in convolutional networks, but we will be careful while using this term, so that it is clear from its context.

The convolutional neural network functions much like a traditional feed-forward neural network, except that the operations in its layers are spatially organized with sparse (and carefully designed) connections between layers. The three types of layers that are commonly present in a convolutional neural network are *convolution*, *pooling*, and *ReLU*. The ReLU activation is no different from a traditional neural network. In addition, a final set of layers is often fully connected and maps in an application-specific way to a set of output nodes. In the following, we will describe each of the different types of operations and layers, and the typical way in which these layers are interleaved in a convolutional neural network.

Why do we need depth in each layer of a convolutional neural network? To understand this point, let us examine how the input to the convolutional neural network is organized. The input data to the convolutional neural network is organized into a 2-dimensional grid structure, and the values of the individual grid points are referred to as *pixels*. Each pixel, therefore, corresponds to a spatial location within the image. However, in order to encode the precise color of the pixel, we need a multidimensional array of values at each grid location. In the RGB color scheme, we have an intensity of the three primary colors, corresponding to red, green, and blue, respectively. Therefore, if the spatial dimensions of an image are 32×32 pixels and the depth is 3 (corresponding to the RGB color channels), then the overall number of pixels in the image is $32 \times 32 \times 3$. This particular image size is quite common, and also occurs in a popularly used data set for benchmarking, known as CIFAR-10 [583]. An example of this organization is shown in Figure 8.1(a). It is natural to represent the input layer in this 3-dimensional structure because two dimensions are devoted to spatial relationships and a third dimension is devoted to the independent properties along these channels. For example, the intensities of the primary colors are the independent properties in the first layer. In the hidden layers, these independent properties correspond to various types of shapes extracted from local regions of the image. For the purpose of discussion, assume that the input in the qth layer is of size $L_q \times B_q \times d_q$. Here, L_q refers to the *height* (or length), B_q refers to the width (or breadth), and d_q is the depth. In almost all image-centric applications, the values of L_q and B_q are the same. However, we will work with separate notations for height and width in order to retain generality in presentation.

For the first (input) layer, these values are decided by the nature of the input data and its preprocessing. In the above example, the values are $L_1 = 32$, $B_1 = 32$, and $d_1 = 3$. Later layers have exactly the same 3-dimensional organization, except that each of the d_q 2-dimensional grid of values for a particular input can no longer be considered a grid of

raw pixels. Furthermore, the value of d_q is much larger than three for the hidden layers because the number of independent properties of a given local region that are relevant to classification can be quite significant. For $q > 1$, these grids of values are referred to as *feature maps* or *activation maps*. These values are analogous to the values in the hidden layers in a feed-forward network.

In the convolutional neural network, the parameters are organized into sets of 3-dimensional structural units, known as *filters* or *kernels*. The filter is usually square in terms of its spatial dimensions, which are typically much smaller than those of the layer the filter is applied to. On the other hand, *the depth of a filter is always same is the same as that of the layer to which it is applied*. Assume that the dimensions of the filter in the qth layer are $F_q \times F_q \times d_q$. An example of a filter with $F_1 = 5$ and $d_1 = 3$ is shown in Figure 8.1(a). It is common for the value of F_q to be small and odd. Examples of commonly used values of F_q are 3 and 5, although there are some interesting cases in which it is possible to use $F_q = 1$.

The *convolution operation* places the filter at each possible position in the image (or hidden layer) so that the filter fully overlaps with the image, and performs a dot product between the $F_q \times F_q \times d_q$ parameters in the filter and the matching grid in the input volume (with same size $F_q \times F_q \times d_q$). The dot product is performed by treating the entries in the relevant 3-dimensional region of the input volume and the filter as vectors of size $F_q \times F_q \times d_q$, so that the elements in both vectors are ordered based on their corresponding positions in the grid-structured volume. How many possible positions are there for placing the filter? This question is important, because each such position therefore defines a spatial "pixel" (or, more accurately, a *feature*) in the next layer. In other words, the number of alignments between the filter and image defines the spatial height and width of the next hidden layer. The relative spatial positions of the features in the next layer are defined based on the relative positions of the upper left corners of the corresponding spatial grids in the previous layer. When performing convolutions in the qth layer, one can align the filter at $L_{q+1} = (L_q - F_q + 1)$ positions along the height and $B_{q+1} = (B_q - F_q + 1)$ along the width of the image (without having a portion of the filter "sticking out" from the borders of the image). This results in a total of $L_{q+1} \times B_{q+1}$ possible dot products, which defines the size of the next hidden layer. In the previous example, the values of L_2 and B_2 are therefore defined as follows:

$$L_2 = 32 - 5 + 1 = 28$$
$$B_2 = 32 - 5 + 1 = 28$$

The next hidden layer of size 28×28 is shown in Figure 8.1(a). However, this hidden layer also has a depth of size $d_2 = 5$. Where does this depth come from? This is achieved by using 5 different filters with their own independent sets of parameters. Each of these 5 sets of spatially arranged features obtained from the output of a single filter is referred to as a *feature map*. Clearly, an increased number of feature maps is a result of a larger number of filters (i.e., parameter footprint), which is $F_q^2 \cdot d_q \cdot d_{q+1}$ for the qth layer. *The number of filters used in each layer controls the capacity of the model because it directly controls the number of parameters*. Furthermore, increasing the number of filters in a particular layer increases the number of feature maps (i.e., depth) of the next layer. It is possible for different layers to have very different numbers of feature maps, depending on the number of filters we use for the convolution operation in the previous layer. For example, the input layer typically only has three color channels, but it is possible for the each of the later hidden layers to have depths (i.e., number of feature maps) of more than 500. The idea here is that each

filter tries to identify a particular type of spatial pattern in a small rectangular region of the image, and therefore a large number of filters is required to capture a broad variety of the possible shapes that are combined to create the final image (unlike the case of the input layer, in which three RGB channels are sufficient). Typically, the later layers tend to have a smaller spatial footprint, but greater depth in terms of the number of feature maps. For example, the filter shown in Figure 8.1(b) represents a horizontal edge detector on a grayscale image with one channel. As shown in Figure 8.1(b), the resulting feature will have high activation at each position where a horizontal edge is seen. A perfectly vertical edge will give zero activation, whereas a slanted edge might give intermediate activation. Therefore, sliding the filter everywhere in the image will already detect several key outlines of the image in a single feature map of the output volume. Multiple filters are used to create an output volume with more than one feature map. For example, a different filter might create a spatial feature map of vertical edge activations.

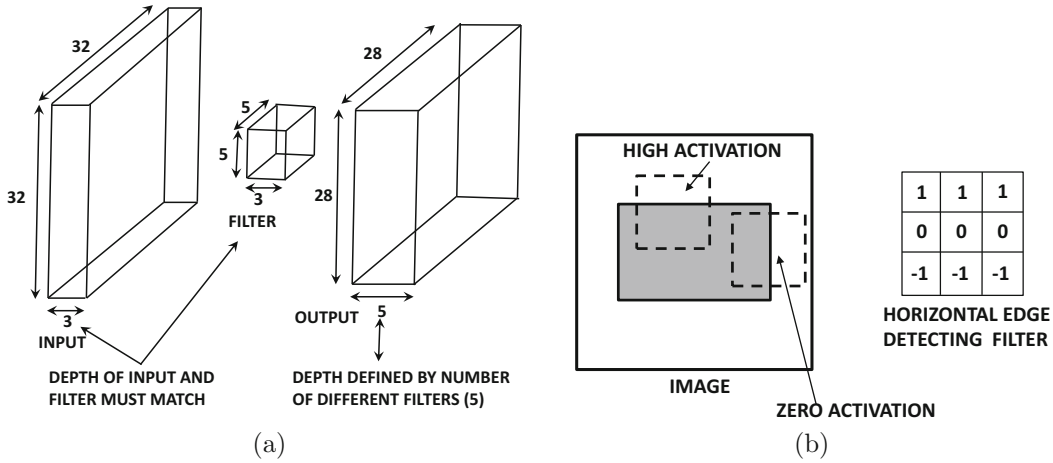

Figure 8.1: (a) The convolution between an input layer of size $32 \times 32 \times 3$ and a filter of size $5 \times 5 \times 3$ produces an output layer with spatial dimensions 28×28. The depth of the resulting output depends on the number of distinct filters and not on the dimensions of the input layer or filter. (b) Sliding a filter around the image tries to look for a particular feature in various windows of the image.

We are now ready to formally define the convolution operation. The pth filter in the qth layer has parameters denoted by the 3-dimensional tensor $W^{(p,q)} = [w_{ijk}^{(p,q)}]$. The indices i, j, k indicate the positions along the height, width, and depth of the filter. The feature maps in the qth layer are represented by the 3-dimensional tensor $H^{(q)} = [h_{ijk}^{(q)}]$. When the value of q is 1, the special case corresponding to the notation $H^{(1)}$ simply represents the input layer (which is not hidden). Then, the convolutional operations from the qth layer to the $(q+1)$th layer are defined as follows:

$$h_{ijp}^{(q+1)} = \sum_{r=1}^{F_q} \sum_{s=1}^{F_q} \sum_{k=1}^{d_q} w_{rsk}^{(p,q)} h_{i+r-1,j+s-1,k}^{(q)} \qquad \forall i \in \{1 \ldots, L_q - F_q + 1\}$$

$$\forall j \in \{1 \ldots B_q - F_q + 1\}$$

$$\forall p \in \{1 \ldots d_{q+1}\}$$

The expression above seems notationally complex, although the underlying convolutional operation is really a simple dot product over the entire volume of the filter, which is repeated over all valid spatial positions (i, j) and filters (indexed by p). It is intuitively helpful to understand a convolution operation by placing the filter at each of the 28×28 possible spatial positions in the first layer of Figure 8.1(a) and performing a dot product between the vector of $5 \times 5 \times 3 = 75$ values in the filter and the corresponding 75 values in $H^{(1)}$. Even though the size of the input layer in Figure 8.1(a) is 32×32, there are only $(32-5+1) \times (32-5+1)$ possible spatial alignments between an input volume of size 32×32 and a filter of size 5×5.

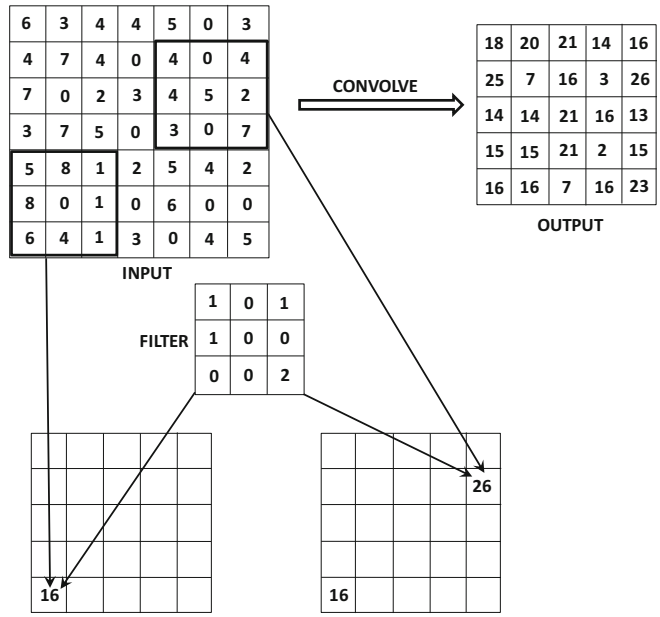

Figure 8.2: An example of a convolution between a $7 \times 7 \times 1$ input and a $3 \times 3 \times 1$ filter with stride of 1. A depth of 1 has been chosen for the filter/input for simplicity. For depths larger than 1, the contributions of each input feature map will be added to create a single value in the feature map. A single filter will always create a single feature map irrespective of its depth.

The convolution operation brings to mind Hubel and Wiesel's experiments that use the activations in small regions of the visual field to activate particular neurons. In the case of convolutional neural networks, this visual field is defined by the filter, which is applied to all locations of the image in order to detect the presence of a shape at each spatial location. Furthermore, the filters in earlier layers tend to detect more primitive shapes, whereas the filters in later layers create more complex compositions of these primitive shapes. This is not particularly surprising because most deep neural networks are good at hierarchical feature engineering.

One property of convolution is that it shows *equivariance to translation*. In other words, if we shifted the pixel values in the input in any direction by one unit and then applied convolution, the corresponding feature values will shift with the input values. This is because of the shared parameters of the filter across the entire convolution. The reason for sharing

Figure 8.18: Example of image classification/localization in which the class "*fish*" is identified together with its bounding box. The image is illustrative only.

be performed by deriving the weights from a trained *deep-belief convolutional network* [285]. This is analogous to the approach in traditional neural networks, where stacked Boltzmann machines were among the earliest models used for pretraining.

8.6 Applications of Convolutional Networks

Convolutional neural networks have several applications in object detection, localization, video, and text processing. Many of these applications work on the basic principle of using convolutional neural networks to provide engineered features, on top of which multidimensional applications can be constructed. The success of convolutional neural networks remains unmatched by almost any class of neural networks. In recent years, competitive methods have even been proposed for sequence-to-sequence learning, which has traditionally been the domain of recurrent networks.

8.6.1 Content-Based Image Retrieval

In content-based image retrieval, each image is first engineered into a set of multidimensional features by using a pretrained classifier like *AlexNet*. The pretraining is typically done up front using a large data set like *ImageNet*. A huge number of choices of such pretrained classifiers is available at [586]. The features from the fully connected layers of the classifier can be used to create a multidimensional representation of the images. The multidimensional representations of the images can be used in conjunction with any multidimensional retrieval system to provide results of high quality. The use of neural codes for image retrieval is discussed in [16]. The reason that this approach works is because the features extracted from *AlexNet* have semantic significance to the different types of shapes present in the data. As a result, the quality of the retrieval is generally quite high when working with these features.

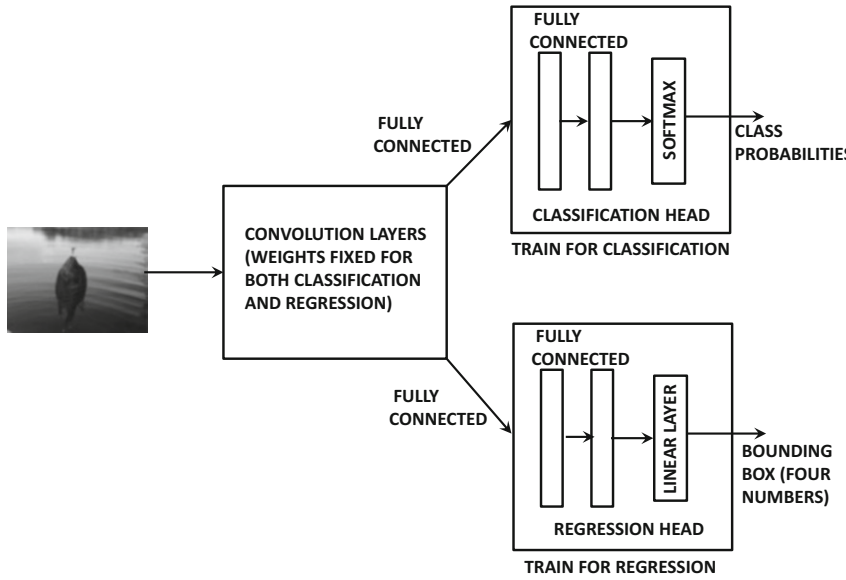

Figure 8.19: The broad framework of classification and localization

8.6.2 Object Localization

In object localization, we have a fixed set of objects in an image, and we would like to identify the rectangular regions in the image in which the object occurs. The basic idea is to take an image with a *fixed* number of objects and encase each of them in a bounding box. In the following, we will consider the simple case in which a single object exists in the image. Image localization is usually integrated with the classification problem, in which we first wish to classify the object in the image and draw a bounding box around it. For simplicity, we consider the case in which there is a single object in the image. We have shown an example of image classification and localization in Figure 8.18, in which the class "*fish*" is identified, and a bounding box is drawn around the portion of the image that delineates that class.

The bounding box of an image can be uniquely identified with four numbers. A common choice is to identify the top-left corner of the bounding box, and the two dimensions of the box. Therefore, one can identify a box with four unique numbers. This is a regression problem with multiple targets. Here, the key is to understand that one can train almost the same model for both classification and regression, which vary only in terms of the final two fully connected layers. This is because the semantic nature of the features extracted from the convolution network are often highly generalizable across a wide variety of tasks. Therefore, one can use the following approach:

1. First, we train a neural network classifier like *AlexNet* or use a pretrained version of this classifier. In the first phase, it suffices to train the classifier only with image-class pairs. One can even use an off-the-shelf pretrained version of the classifier, which was trained on *ImageNet*.

2. The last two fully connected layers and softmax layers are removed. This removed set of layers is referred to as the *classification head*. A new set of two fully connected

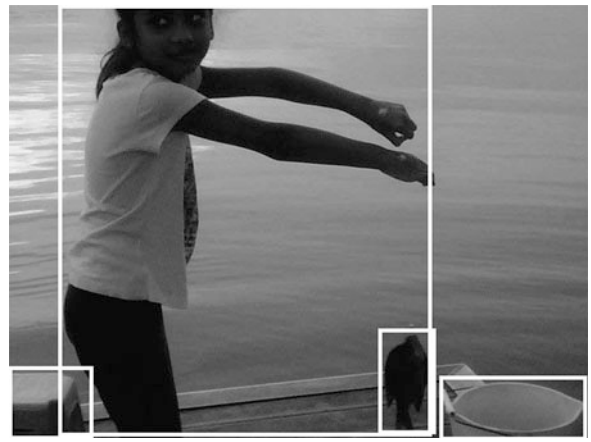

Figure 8.20: Example of object detection. Here, four objects are identified together with their bounding boxes. The four objects are "*fish*," "*girl*," "*bucket*," and "*seat*." The image is illustrative only.

layers and a linear regression layer is attached. Only these layers are then trained with training data containing images and their bounding boxes. This new set of layers is referred to as the *regression head*. Note that the weights of the convolution layers are fixed, and are not changed. Both the classification and regression heads are shown in Figure 8.19. Since the classification and regression heads are not connected to one another in any way, these two layers can be trained independently. The convolution layers play the role of creating visual features for both classification and regression.

3. One can optionally fine-tune the convolution layers to be sensitive to both classification and regression (since they were originally trained only for classification). In such a case, both classification and regression heads are attached, and the training data for images, their classes, and bounding boxes are shown to the network. Backpropagation is used to fine-tune all layers. This full architecture is shown in Figure 8.19.

4. The entire network (with both classification and regression heads attached) is then used on the test images. The outputs of the classification head provide the class probabilities, whereas the outputs of the regression head provide the bounding boxes.

One can obtain results of superior quality by using a sliding-window approach. The basic idea in the sliding-window approach is to perform the localization at multiple locations in the image with the use of a sliding window, and then integrate the results of the different runs. An example of this approach is the *Overfeat* method [441]. Refer to the bibliographic notes for pointers to other localization methods.

8.6.3 Object Detection

Object detection is very similar to object localization, except that there is a *variable* number of objects of different classes in the image. In this case, one wishes to identify all the objects in the image together with their classes. We have shown an example of object detection in Figure 8.20, in which there are four objects corresponding to the classes "*fish*," "*girl*," "*bucket*," and "*seat*." The bounding boxes of these classes are also shown in the figure.

Object detection is generally a more difficult problem than that of localization because of the variable number of outputs. In fact, one does not even know a priori how many objects there are in the image. For example, one cannot use the architecture of the previous section, where it is not clear how many classification or regression heads one might attach to the convolutional layers.

The simplest approach to this problem is to use a sliding window approach. In the sliding window approach, one tries all possible bounding boxes in the image, on which the object localization approach is applied to detect a single object. As a result, one might detect different objects in different bounding boxes, or the same object in overlapping bounding boxes. The detections from the different bounding boxes can then be integrated in order to provide the final result. Unfortunately, the approach can be rather expensive. For an image of size $L \times L$, the number of possible bounding boxes is L^4. Note that one would have to perform the classification/regression for each of these L^4 possibilities for each image at test time. This is a problem, because one generally expects the testing times to be modest enough to provide real-time responses.

In order to address this issue *region proposal methods* were advanced. The basic idea of a region proposal method is that it can serve as a general-purpose object detector that merges regions with similar pixels together to create larger regions. Therefore, the region proposal methods are used to first create a set of candidate bounding boxes, and then the object classification/localization method is run in each of them. Note that some candidate regions might not have valid objects, and others might have overlapping objects. These are then used to integrate and identify all the objects in the image. This broader approach has been used in various techniques like *MCG* [172], *EdgeBoxes* [568], and *SelectiveSearch* [501].

8.6.4 Natural Language and Sequence Learning

While the preferred way of machine learning with text sequences is that of recurrent neural networks, the use of convolutional neural networks has become increasingly popular in recent years. At first sight, convolutional neural networks do not seem like a natural fit for text-mining tasks. First, image shapes are interpreted in the same way, irrespective of where they are in the image. This is not quite the case for text, where the position of a word in a sentence seems to matter quite a bit. Second, issues such as position translation and shift cannot be treated in the same way in text data. Neighboring pixels in an image are usually very similar, whereas neighboring words in text are almost never the same. In spite of these differences, the systems based on convolutional networks have shown improved performance in recent years.

Just as an image is represented as a 2-dimensional object with an additional depth dimension defined by the number of color channels, a text sequence is represented as 1-dimensional object with depth defined by its dimensionality of representation. The dimensionality of representation of a text sentence is equal to the lexicon size for the case of one-hot encoding. Therefore, instead of 3-dimensional boxes with a spatial extent and a depth (color channels/feature maps), the filters for text data are 2-dimensional boxes with a window (sequence) length for sliding along the sentence and a depth defined by the lexicon. In later layers of the convolutional network, the depth is defined by the number of feature maps rather than the lexicon size. Furthermore, the number of filters in a given layer defines the number of feature maps in the next layer (as in image data). In image data, one performs convolutions at all 2-dimensional locations, whereas in text data one performs convolutions at all 1-dimensional points in the sentence with the same filter. One challenge

with this approach is that the use of one-hot encoding increases the number of channels, and therefore blows up the number of parameters in the filters in the first layer. The lexicon size of a typical corpus may often be of the order of 10^6. Therefore, various types of pretrained embeddings of words, such as *word2vec* or *GLoVe* [371] are used (cf. Chapter 2) in lieu of the one-hot encodings of the individual words. Such word encodings are semantically rich, and the dimensionality of the representation can be reduced to a few thousand (from a hundred-thousand). This approach can provide an order of magnitude reduction in the number of parameters in the first layer, in addition to providing a semantically rich representation. All other operations (like max-pooling or convolutions) in the case of text data are similar to those of image data.

8.6.5 Video Classification

Videos can be considered generalizations of image data in which a temporal component is inherent to a sequence of images. This type of data can be considered *spatio-temporal data*, which requires us to generalize the 2-dimensional spatial convolutions to 3-dimensional spatio-temporal convolutions. Each frame in a video can be considered an image, and one therefore receives a sequence of images in time. Consider a situation in which each image is of size $224 \times 224 \times 3$, and a total of 10 frames are received. Therefore, the size of the video segment is $224 \times 224 \times 10 \times 3$. Instead of performing spatial convolutions with a 2-dimensional spatial filter (with an additional depth dimension capturing 3 color channels), we perform spatiotemporal convolutions with a 3-dimensional spatiotemporal filter (and a depth dimension capturing the color channels). Here, it is interesting to note that the nature of the filter depends on the data set at hand. A purely sequential data set (e.g., text) requires 1-dimensional convolutions with windows, an image data set requires 2-dimensional convolutions, and a video data set requires 3-dimensional convolutions. We refer to the bibliographic notes for pointers to several papers that use 3-dimensional convolutions for video classification.

An interesting observation is that 3-dimensional convolutions add only a limited amount to what one can achieve by averaging the classifications of individual frames by image classifiers. A part of the problem is that motion adds only a limited amount to the information that is available in the individual frames for classification purposes. Furthermore, sufficiently large video data sets are hard to come by. For example, even a data set containing a million videos is often not sufficient because the amount of data required for 3-dimensional convolutions is much larger than that required for 2-dimensional convolutions. Finally, 3-dimensional convolutional neural networks are good for relatively short segments of video (e.g., half a second), but they might not be so good for longer videos.

For the case of longer videos, it makes sense to combine recurrent neural networks (or LSTMs) with convolutional neural networks. For example, we can use 2-dimensional convolutions over individual frames, but a recurrent network is used to carry over states from one frame to the next. One can also use 3-dimensional convolutional neural networks over short segments of video, and then hook them up with recurrent units. Such an approach helps in identifying actions over longer time horizons. Refer to the bibliographic notes for pointers to methods that combine convolutional and recurrent neural networks.

in the data set has size $32 \times 32 \times 3$. It is noteworthy that the *CIFAR-10* data set is a small subset of the *tiny images data set* [642], which originally contains 80 million images. The *CIFAR-10* data set is often used for smaller scale testing, before a more large-scale training is done with *ImageNet*. The *CIFAR-100* data set is just like the *CIFAR-10* data set, except that it has 100 classes, and each class contains 600 instances. The 100 classes are grouped into 10 super-classes.

8.9 Exercises

1. Consider a 1-dimensional time-series with values 2, 1, 3, 4, 7. Perform a convolution with a 1-dimensional filter 1, 0, 1 and zero padding.

2. For a one-dimensional time series of length L and a filter of size F, what is the length of the output? How much padding would you need to keep the output size to a constant value?

3. Consider an activation volume of size $13 \times 13 \times 64$ and a filter of size $3 \times 3 \times 64$. Discuss whether it is possible to perform convolutions with strides 2, 3, 4, and 5. Justify your answer in each case.

4. Work out the sizes of the spatial convolution layers for each of the columns of Table 8.2. In each case, we start with an input image volume of $224 \times 224 \times 3$.

5. Work out the number of parameters in each spatial layer for column D of Table 8.2.

6. Download an implementation of the *AlexNet* architecture from a neural network library of your choice. Train the network on subsets of varying size from the *ImageNet* data, and plot the top-5 error with data size.

7. Compute the convolution of the input volume in the upper-left corner of Figure 8.2 with the horizontal edge detection filter of Figure 8.1(b). Use a stride of 1 without padding.

8. Perform a 4×4 pooling at stride 1 of the input volume in the upper-left corner of Figure 8.4.

9. Discuss the various type of pretraining that one can use in the image captioning application discussed in Section 7.7.1 of Chapter 7.

10. You have a lot of data containing ratings of users for different images. Show how you can combine a convolutional neural network with the collaborative filtering ideas discussed in Chapter 2 to create a hybrid between a collaborative and content-centric recommender system.