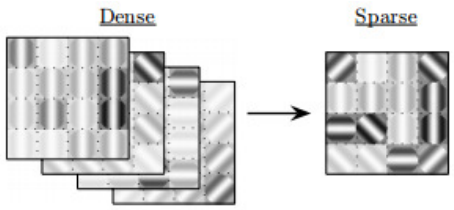

Figure 3.11: Cross-Channel Pooling Illustration. (left) Dense simple cell responses resulting from filtering operations with Gabor filters tuned to various orientations (4 orientations are shown here for illustration purposes) (right) Sparsified simple cell responses resulting from cross-channel pooling using the max operator (*i.e.* for each pixel location, the maximum response across feature maps is kept). Figure reproduced from [110].

Discussion

Overall, based on the description of complex cells, it seems that from a biological perspective both average and max pooling are plausible, although there is more work arguing in favor of average pooling. Independently from the choice of the pooling operator, the fact is that there is general agreement on the existence and significance of pooling. A probably more important question lies in the choice of the receptive field or the units over which pooling is performed. This aspect of the pooling operation is further explored in more theory driven work, as will be described in the next section.

3.4.2 Theoretical perspective

Pooling has been a component of the computer vision representational pipelines for some time, *e.g.* [30, 49, 89, 91, 99], with the goal of introducing some level of invariance to image transformations and better robustness to noise and clutter. From a theoretical perspective, probably one of the most influential works discussing the importance and role of pooling was Koendrink's concept of locally orderless images [87]. This work argued in favor of pooling whereby the exact position of pixels within a Region Of Interest (ROI), *i.e.* a pooling region, can be neglected even while conserving the global image structure. Currently, virtually all convolutional architectures include a pooling block as part of its processing stages. As with biologically motivated models, more theory driven approaches typically employ either average or max pooling.

Recent work approaching their network's design from a purely theory based perspective, *e.g.* ScatNet [15] and SOE-Net [60], rely on a form of average pooling. In particular, their networks rely on a weighted sum pooling operation. These approaches tackle the pooling problem from a frequency domain point of view; therefore, their choice of average pooling is motivated by a desire to keep track of the frequency content of the signal. Average pooling allows these networks to act on different frequencies at each layer while downsampling the images to increase invariance and reduce redundancies. At the same time their controlled approach to specifying pooling parameters allows them to avoid aliasing during the pooling operation. Notably, in SOE-Net's investigation, the superiority of weighted average pooling was empirically demonstrated over both simple box car pooling and max pooling.

Interestingly, most of the early convolutional architectures relied on average pooling as well, *e.g.* [49, 91], but it has slowly fallen out of favor in many learning based convolutional architectures and been replaced by max pooling. This trend has been mainly driven by small differences in performance. However, the role of pooling in a network is significant and needs more careful consideration. In fact, early work exploring the role of pooling [77] demonstrated that the type of pooling plays such a significant role in a ConvNet architecture that even an otherwise randomly initialized network yielded competitive results on the task of object recognition provided the appropriate type of pooling is used. In particular, this work compared average and max pooling and demonstrated that with a randomly initialized network average pooling yields superior performance.

Other work more systematically compared average and max pooling empirically [128] and suggested that there exists a complementarity between the two types of pooling depending on the input type and the transformations it undergoes. Therefore, this work implied that ConvNets can benefit from using more than one pooling option throughout the architecture. Yet other work considered the question from a purely theoretical perspective [12]. Specifically, this work examined the effect of average versus max pooling on the separability of extracted features. The main conclusions of this paper can be summarized in two points. First, the authors argue

that max pooling is more suitable when the pooled features are very sparse (*e.g.* when pooling is preceded by a ReLU). Second, the authors suggest that the pooling cardinality should increase with the input size and that the pooling cardinality affects the pooling function. More generally, it was shown that beyond the pooling type, the pooling size plays an important role as well.

The importance of the pooling cardinality was also explored in various other studies, albeit empirically [27, 81]. Indeed, the role of pooling cardinality was first discussed in the context of the earlier hand-crafted feature extraction pipeline [81]. In particular, this work builds on the spatial pyramid pooling [89] encoding method while highlighting the shortcoming of using predetermined fixed-size pooling grids. The authors suggest learning the pooling windows' sizes as part of the classifier training. More specifically, the authors suggest randomly picking various pooling regions of different cardinalities and training the classifier to pick the pooling region that yields the highest accuracy. The main motivation behind this learning based strategy is to make pooling adaptive to the dataset. For example, the optimal pooling regions for an outdoor scene may lie along the horizon, which does not necessarily apply to indoor scenes. Similarly, for video action recognition it proved more perspicuous to adapt the pooling region to the most salient parts of a video [39]. The role of the pooling window size or cardinality was also directly explored in a neural network context [27]. Here, the authors suggest that features that are most similar should be pooled together. The authors propose finetuning the pooling support (*i.e.* pooling regions) of their network in an unsupervised manner. In particular, pooling windows are chosen to group together similar features according to a pairwise similarity matrix, where the similarity measure is squared correlation. Beyond average and max pooling operations, the common thread across these investigations is the importance of the pooling region independently from the pooling function.

Other work approaches the choice of pooling and its corresponding parameters from a pure machine learning point of view [58, 96, 153]. From this perspective, pooling is advocated as a regularization technique that allows for varying the network's structure during training. In particular, pooling allows for the creation of sub-models within the big architecture thanks to the variation of pathways that a

back propagated signal may take during training. These variations are achieved with methods such as stochastic pooling [153] or cross-channel pooling used in the maxout network [58] and Network in Network (NiN) [96]. NiN was first introduced as a way to deal with overfitting and correct for the over-complete representation of ConvNets [96] . In particular, due to the large number of kernels used at each layer, it was noticed that many networks often end up learning redundant filters after training. Therefore, NiN is introduced to reduce redundancies at each layer by training the network to learn which feature maps to combine using a weighted linear combination. Similar to NiN, the Maxout network [58] introduces cross channel pooling wherein the output is set as the maximum across k feature maps on a channelwise basis. Notably, a recent proposal also relied on cross channel pooling to minimize redundancies [60] even while being completely learning free. In this work, the network is based on a fixed vocabulary of filters and cross channel pooling is designed to group together feature maps resulting from filtering operations with the same kernel. Beyond minimizing redundancies, this approach was adopted to allow the network size to remain manageable, while maintaining interpretability.

Stochastic Pooling (SP) [153] was also introduced as a regularization technique. However, different from maxout and NiN, which perform cross channel pooling, SP acts within a feature map. In particular, stochastic pooling is inspired from the dropout technique that is widely used in fully connected layers, but SP is applied to convolutional layers instead. It relies on introducing stochasticity to the pooling operation that forces the back propagated signal to randomly take different pathways at each iteration during training. The method starts by normalizing feature map responses, a_i, within each region to be pooled, R_j, as

$$p_i = \frac{a_i}{\sum_{k \in R_j} a_k}. \tag{3.11}$$

The normalized values, p_i, are then used as the probabilities of a multinomial distribution, which is in turn used to sample a location l within the region to be pooled. The corresponding activation a_l is the pooled value. Importantly, although stochastic pooling relies on selecting one value from any region R_j (*i.e.* similar to max pooling), the pooled value is not necessarily the largest in R_j. Here, it is important

to note that a different pooling strategy is adopted during testing. At test time, the probabilities are no longer used to sample a location during pooling; instead, they are used as the weights of a weighted sum pooling operation. Hence stochastic pooling is closer in spirit to max pooling during training and closer to average pooling during testing. The authors argue that the adopted pooling strategy during training allows for creating different models thanks to varying pathways, while the pooling used during testing allows for creating a rough average approximation over all possible models seen during training. In summary, stochastic pooling can be seen as an attempt to take the best of both average and max pooling.

Another approach that attempts to achieve a balance between average and max pooling suggests letting the network learn the optimal pooling method [95]. This idea of multiple pooling strategies is motivated by experiments demonstrating that the choice of optimal pooling strategy is affected by the input [128]. In particular, the authors propose three different methods of combining the benefits of average and max pooling, namely; mixed, gated and tree pooling. Mixed pooling combines average and max pooling independently from the region to be pooled, where the network is trained to learn the mixing proportion according to

$$f_{mix}(\mathbf{x}) = a_l f_{max}(\mathbf{x}) + (1 - a_l) f_{avg}(\mathbf{x}), \tag{3.12}$$

subject to the constraint $a_l \in [0, 1]$. In gated max-average pooling the mixing proportion is adaptive to the region to be pooled. In particular, the network is trained to learn a gating mask, $\mathbf{w}$, that is applied to the input data via pointwise multiplication. Using this gating mask, the mixing function is now defined as

$$f_{mix}(\mathbf{x}) = \sigma(\mathbf{w}^T\mathbf{x}) f_{max}(\mathbf{x}) + (1 - \sigma(\mathbf{w}^T\mathbf{x})) f_{avg}(\mathbf{x}), \tag{3.13}$$

with $\sigma(\mathbf{w}^T\mathbf{x}) = \frac{1}{(1+exp(-\mathbf{w}^T\mathbf{x}))}$.

The third pooling strategy proposed in this work is tree pooling, which can be viewed as an extreme version of gated pooling. In tree pooling, not only the mixing proportions are learned but the pooling functions to be combined are learned as well. Specifically, a tree structure is adopted to learn the parameters of the

individual functions and their mixing strategy as well. The difference between the three pooling methods is illustrated in Figure 3.12. In sum, the main idea behind these proposals is letting the pooling strategy adapt to the region being pooled. Following this strategy, the authors were able to demonstrate the value of not only combining average and max pooling but also that of adapting the pooling function to the region to be pooled.

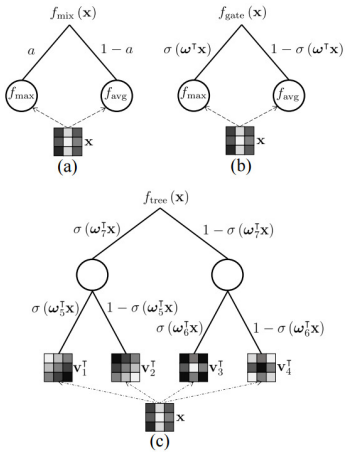

Figure 3.12: Mixed, Gated and Tree Pooling. Illustration of the described (a) mixed max-average pooling, (b) gated max-average pooling and (c) tree pooling. Figure reproduced from [95].

Finally, it is worth mentioning under this section one last type of pooling, referred to as global pooling. Global pooling has been used in some prominent ConvNet models in an effort to deal with more practical issues relevant to ConvNet architecture design [62, 96]. For example, it is known that standard ConvNets rely on convolutional layers for feature learning/extraction and fully connected layers followed by a softmax for classification. However, fully connected layers entail the use of a large number of parameters and are thereby prone to overfitting. Many methods were introduced to deal with overfitting induced by fully connected layers, perhaps the most widely used of which is dropout [88]. However, a more elegant way that fits naturally in a convolutional framework was introduced in NiN [96] and it is called global average pooling. It simply relies on aggregating the last layer features across the entire feature map support. Another example of reliance on global pooling is also found in the so called SPP-Net [62]. In this work, Spatial Pyramid Pooling (SPP) [89], is used to enable convolutional networks to accept input images of any

size. In fact, ConvNets require fixed size input due to the use of fully connected layers. SPP-Net introduces spatial pyramid pooling after the last convolutional layer to correct for this difficulty. In particular, spatial pyramid pooling is used to generate a fixed size representation independently from the size of the input image as illustrated in Figure 3.13. Notably, global average pooling used in NiN, is akin to performing spatial pyramid pooling at the last layer of the ConvNet where the pyramid consists of only the coarsest level.

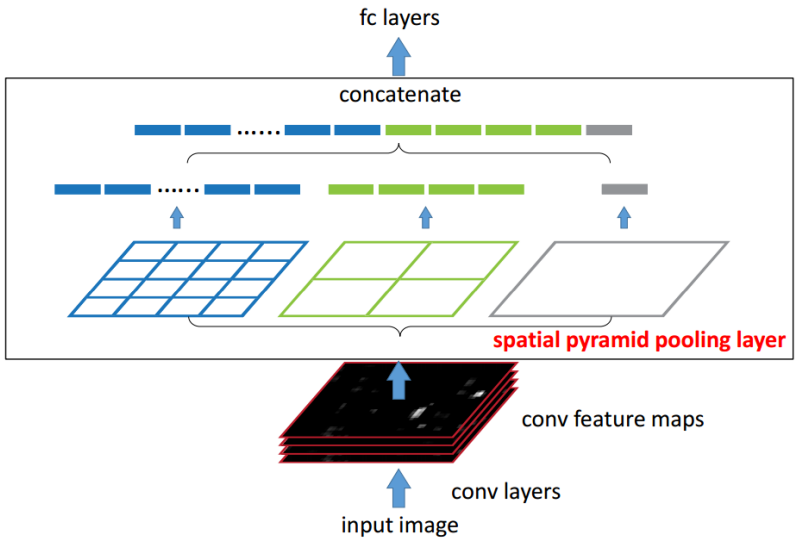

Figure 3.13: Spatial Pyramid Pooling Network. SPP is applied to the feature maps of the last convolutional layer of a network. Because the spatial bins are proportional to the image size, SPP generates feature vectors of the same size independently of the input image size. Hence SPP-Net does not require input images to be pre-processed such that they are of the same size. Figure reproduced from [62].

Discussion

Traditionally, the default functions used in pooling have relied on either the average or max operators. However, several investigations revealed a certain complementary between the two showing that more parameters should be taken into account when choosing the pooling operation. Due to such observations, recent research has been pushing to extend the idea of training to include learning the pooling functions and their parameters. However, this direction entails an increase in the number of parameters to be learned and thereby more chances of overfitting. Importantly, this approach is to be taken with caution as it would likely further obscure our knowl-

edge and understanding of the learned representations. In complement, pooling parameters can be specified on a theoretical basis for cases where previous stages of processing have adequate analytic detail. Overall, pooling should be viewed as a way to summarize information from multiple features into a compact form that preserves the important aspects of the signal while discarding details. Beyond deciding how to summarize the features, it is clear that the harder problem is to determine what constitutes data that should be pooled and where that data is present.

3.5 Overall discussion

This chapter discussed the role and importance of the most widely used building blocks in a typical ConvNet architecture in an effort to understand the working principles of ConvNets. In particular, the details of each block were addressed from both biological and theoretical perspectives. Overall, various common threads emerge from the exploration of the discussed building blocks. In particular, it appears that all blocks find relatively strong motivations from the operations taking place in the visual cortex. Further, although all blocks play a significant role in ConvNets, it appears that the selection of the convolutional kernels is the most important aspect, as evidenced by the larger body of literature tackling this block. More importantly, it seems that more recent ConvNet architectures discussed throughout this chapter (*e.g.* [15, 28, 60, 75, 148]) are aiming at minimizing the need for heavy training based solutions by incorporating more controlled building blocks at various stages of their networks. These recent approaches are in turn motivated by various efforts that revealed the sub-optimality of the learning based ConvNets (*e.g.* predominant redundancies in some of the widely used learned ConvNets) via layerwise visualization and ablation studies, as will be discussed in the next chapter.

Chapter 4

Current Status

The review of the role of the various components of ConvNet architectures emphasized the importance of the convolutional block, which is largely responsible for most abstractions captured by the network. In contrast, this component remains the least understood block of processing given that it entails the heaviest learning. This chapter reviews the current trends in attempting to understand what is being learned at various ConvNet layers. In light of these trends, various critical open problems that remain will be highlighted.

4.1 Current trends

While various ConvNet models continue to push the state-of-the-art further in several computer vision applications, understanding of how and why these systems work so well is limited. This question has sparked the interest of various researchers and in response several approaches are emerging as ways of understanding ConvNets. In general, these approaches can be divided into three tacks: those that rely on visualizations of the learned filters and the extracted feature maps, those that rely on ablation studies as inspired from biological approaches to understanding the visual cortex and those that rely on minimizing learning by introducing analytic principles into their network's design. Each of these approaches will be briefly reviewed in this section.

4.1.1 Understanding ConvNets via visualization

Although several methods have been proposed in the literature for visualizing the feature maps extracted by ConvNets, in this section we will focus on the two most prominent approaches and discuss their different variations.

The first approach to ConvNet visualization is known as a dataset-centric approach [151] because it relies on probing the network with input from a dataset to find maximally responding units in the network. One of the earliest approaches falling under this category is known as DeConvNet [154], where visualization is achieved in two steps. First, a ConvNet is presented with several images from a dataset and the feature maps responding maximally to this input are recorded. Second, these feature maps are projected back to the image space using the DeConvNet architecture, which consists of blocks that invert the operations of the ConvNet used. In particular, DeConvNet inverts the convolution operations (*i.e.* performs "de-convolution") via use of the transpose of the learned filters in the ConvNet under consideration. Here, it is worth noting that taking the transpose is not guaranteed to invert a convolution operation. For "un-pooling", DeConvNet relies on recording the locations corresponding to max-pooled responses in the ConvNet and uses those locations for "un-pooling" or upsampling the feature maps. These operations are summarized in Figure 4.1.

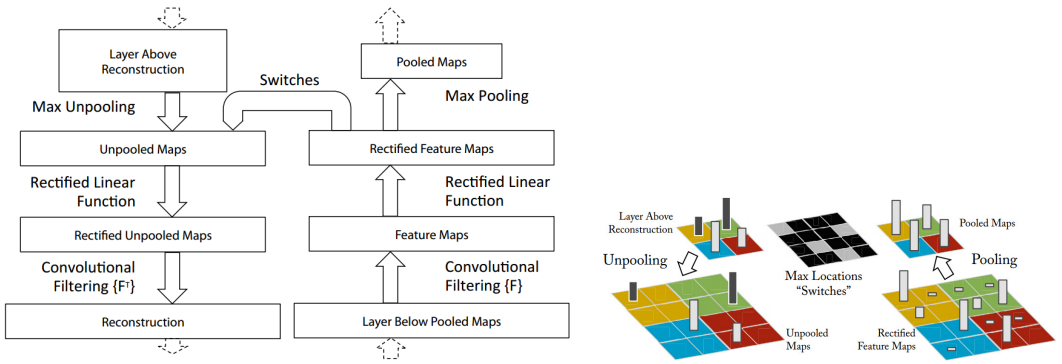

Figure 4.1: DeConvNet building blocks. (a) Illustrates a DeConvNet operation that can be applied to project the feature maps, extracted from any layer of a ConvNet, back to image space. (b) Illustrates the "un-pooling" operation via use of "switches", which correspond to the locations responding to the max pooling operation. Figure reproduced from [154].