Deep Convolutional Neural Nets

Carlo Tomasi

October 30, 2017

This note covers one particular way to build a particular type of deep feed-forward network. Such a network can be used for either classification or regression, and we will focus on classification. More variants and details can be found in many books or articles on neural nets [1], convolutional neural nets [6], and deep learning in general [2, 3].

A later note on training will describe how to determine the parameters (weights) of a deep feed-forward network of a given structure and for a given classification task.

1 Neurons

A *neuron* (in the computational sense) is a function $\mathbb{R}^D \to \mathbb{R}$ of the form

$$y = h(a(\mathbf{x})) \quad \text{where} \quad a = \mathbf{w}^T \tilde{\mathbf{x}} \quad , \quad \tilde{\mathbf{x}} = \begin{bmatrix} \mathbf{x} \\ 1 \end{bmatrix} .$$

The entries of the vector $\mathbf{w} \in \mathbb{R}^{D+1}$ are called the *weights*, and the *activation function* is a nonlinear and weakly monotonic function $\mathbb{R} \to \mathbb{R}$. The input $a(\mathbf{x})$ to h is called the *activation* of the neuron, and the particular type of activation function

$$h(a) = \max(0, a)$$

is called the *Rectified Linear Unit* (ReLU, Figure 1).

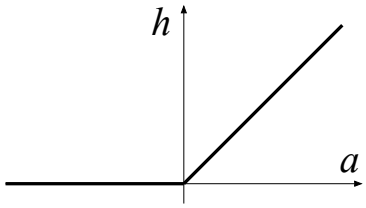

Figure 1: The Rectified Linear Unit (ReLU).

We view the tilde (as in $\tilde{\mathbf{x}}$) as an operator: Given any vector $\mathbf{x}$, this operator appends a 1 at the end of $\mathbf{x}$.

The activation can be rewritten as follows

$$a = \mathbf{v}^T \mathbf{x} + b \quad \text{where} \quad \mathbf{v}^T = [w_1, \ldots, w_D] \quad \text{and} \quad b = w_{D+1} ,$$

and is an inner product between a *gain*[1] vector $\mathbf{v}$ and the input $\mathbf{x}$ plus a *bias* b. Figure 2 shows a neuron in diagrammatic form.

[1] Gains are often called weights as well.

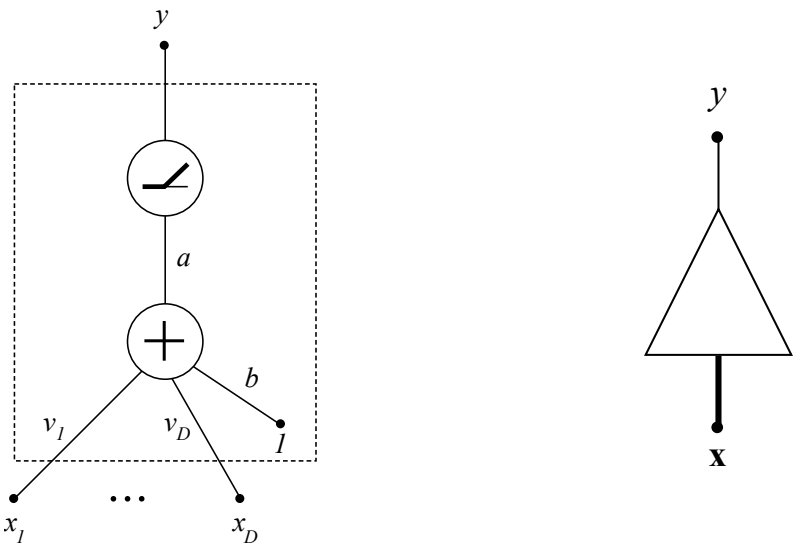

Figure 2: The internal structure of a neuron (left) and a neuron as a black box (right). The black box corresponds to the part inside the dashed rectangle on the left.

For different inputs $\mathbf{x}$ of the same magnitude[2], the activation is maximum when $\mathbf{x}$ is parallel to $\mathbf{v}$, and the latter can be viewed as a *pattern* or *template* to which $\mathbf{x}$ is compared. The bias b then raises or lowers the activation before it is passed through the activation function.

The ReLU will *respond* (that is, return a nonzero output) if the inner product $\mathbf{v}^T\mathbf{x}$ is greater than $-b$ (so that a is positive), and the response thereafter increases with the value of a. So the negative of the bias can be viewed as a *threshold* that the inner product between pattern and input must exceed before it is deemed to be significant, and the neuron can be viewed as a *score function* that measures the similarity of the suitably normalized input $\mathbf{x}$ to the pattern $\mathbf{v}$ when the similarity is significant (that is, greater than $-b$). When the similarity is not significant, the neuron does not respond.

A *pattern classifier* would add a stage that decides if the score is large enough to declare the input $\mathbf{x}$ to contain the pattern represented by $\mathbf{v}$. So another way to view a neuron is a pattern classifier without the decision stage.

2 Two-Layer Neural Nets

A neural-net *layer* is a vector of $D^{(1)}$ neurons, that is, a function $\mathbb{R}^D \to \mathbb{R}^{D^{(1)}}$

$$\mathbf{y} = h(\mathbf{a}(\mathbf{x})) \quad \text{where} \quad \mathbf{a}(\mathbf{x}) = W\tilde{\mathbf{x}}\,,$$

the weight matrix W is $D^{(1)} \times (D+1)$, and the activation function h is applied to each entry of the activation vector $\mathbf{a}(\mathbf{x}) \in \mathbb{R}^{D^{(1)}}$. So a neural-net layer can be viewed as a *bank* of pattern scoring devices, one pattern per neuron. Figure 3 illustrates.

To compute the output of a layer from its input one needs to perform $D^{(1)}(D+1)$ multiplications and almost as many additions to compute the activation vector, and then compute the activation function $D^{(1)}$

[2] As measured by its Euclidean norm $\|\mathbf{x}\|$.

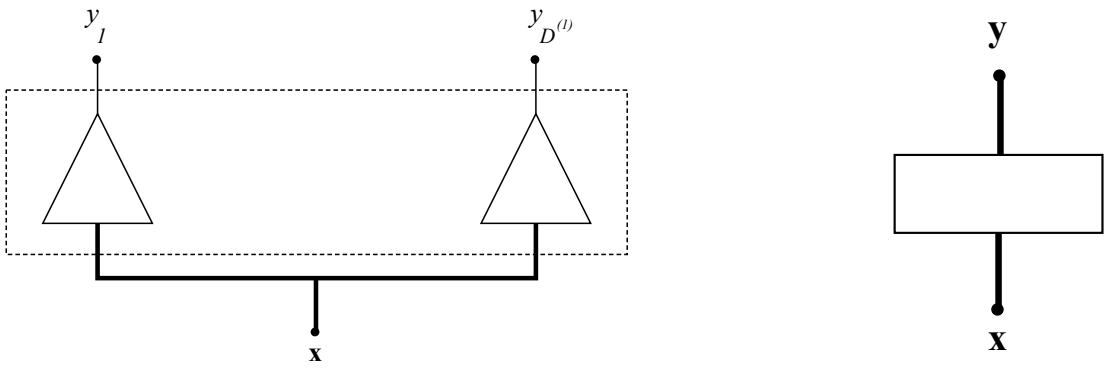

Figure 3: The internal structure of a layer (left) and a layer as a black box (right). The black box corresponds to the part inside the dashed rectangle on the left.

times. So if $D^{(1)}$ is of the same order of magnitude as D, the cost of this computation is quadratic in the size D of the input $\mathbf{x}$. Even more importantly, there are $O(D^2)$ parameters (the entries of W) that need to be determined when the layer is trained.

A *two-layer neural net* is a cascade of two layers,

$$\begin{aligned} \mathbf{x}^{(1)} &= h(W^{(1)}\tilde{\mathbf{x}}) \\ \mathbf{y} &= h_y(W^{(2)}\tilde{\mathbf{x}}^{(1)}) \end{aligned}$$

where the activation function h_y is in general different from h and $W^{(2)}$ is $D^{(2)} \times (D^{(1)} + 1)$.

It can be proven [7] that any mapping $\mathbb{R}^D \to \mathbb{R}^{D^{(2)}}$ that is Lebesgue-integrable and has Lebesgue-integrable Fourier transform[3] can be approximated to any finite degree of accuracy over a hypercube in $\mathbb{R}^D$ with a two-layer neural net where h_y is the identity function and h is the ReLU. This result, along with similar ones for other activation functions [4], shows that two-layer neural nets are *universal approximators*.

The mathematics used in the proof of this statement for a ReLU-equipped net is complex. Here is a simpler but informal argument that might convince you of the universal approximator property, at least in one dimension ($D = D^{(2)} = 1$). This argument takes a function $f(x)$ defined on some interval of real numbers and an arbitrary accuracy requirement. It then builds a two-layer neural network whose input-output transformation $g(x)$ approximates $f(x)$ to the required accuracy.

First, the *triangle function*

$$t(x) = \begin{cases} 1 + x & \text{if } -1 \le x \le 0 \\ 1 - x & \text{if } 0 \le x \le 1 \\ 0 & \text{otherwise} \end{cases}$$

can be built as

$$t(x) = h(x + 1) - 2h(x) + h(x - 1) \tag{1}$$

where h is the ReLU. This construction is shown on the left in Figure 4. The function

$$\alpha\, t\Big(\frac{x}{T}\Big)$$

[3] Just think of these as mild requirements on the mapping. It is not important for our purposes to know what they mean.

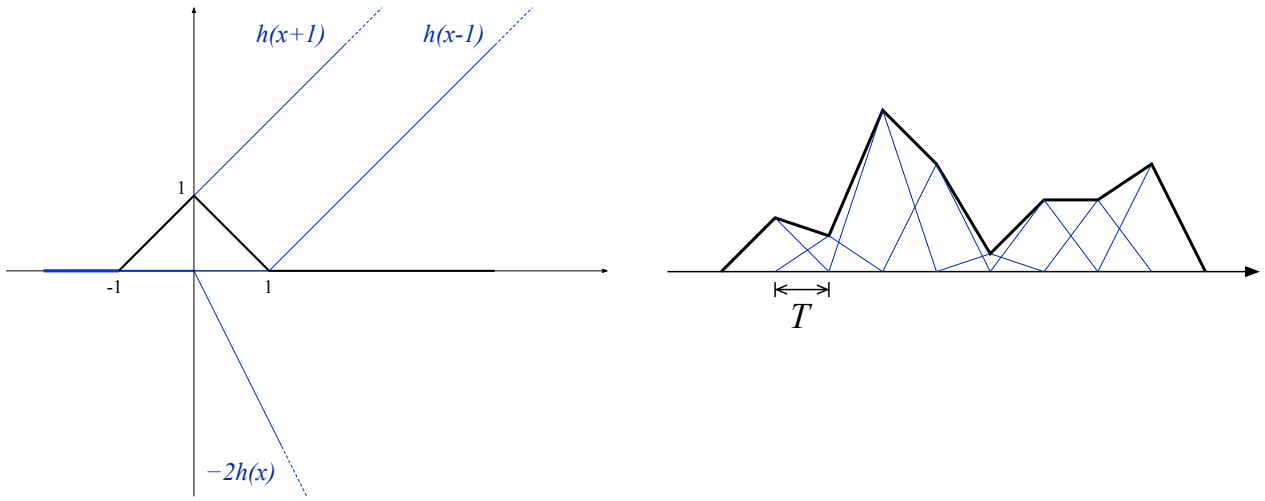

Figure 4: Left: Three ReLU functions with proper displacements (blue) can be linearly combined to form a unit triangle function (black). Right: Any piecewise-linear function (sampled at regular intervals as in the figure, or otherwise) can be written as a linear combination of triangle functions.

is a version of t that is scaled in both domain (by $1/T$) and range (by α). In a two-layer neural net, the domain shifts in equation (1) can be achieved by the biases in $W^{(1)}$, and domain scaling can be achieved by gains in $W^{(1)}$. Both range scaling and the linear-combination coefficients $(1, -2, 1)$ in equation (1) can be implemented through gains in $W^{(2)}$.

Given a function $f(x)$ defined over some interval X, approximate it with a piecewise linear function that interpolates linearly between samples of f taken with sampling period T:

$$g(x) = \sum_k f(kT)\, t\left(\frac{x - kT}{T}\right)$$

(see right panel in figure 4). By making T small enough, the quality of the approximation can be made as good as desired. Since g is in the form of a two-layer neural net, the argument is complete. The architecture of the approximation network is shown in figure 5.

However, a two-layer approximator to a given function f may be very expensive to implement. For instance, many triangle functions may be needed in the construction above, especially for functions whose domain is multidimensional (large D). Deep neural nets are introduced in the hope that they lead to more efficient approximations, as discussed in the next two sections.

3 Convolutional Layers

A neuron matches the input $\mathbf{x}$ to a pattern $\mathbf{v}$. What should the patterns in an image recognition system be? One could make $\mathbf{x}$ be the entire image, with its pixels strung into a vector, and then $\mathbf{v}$ could be an image (in vector form as well) of the object to be recognized—say, your grandmother's face. This net would not work well, as your grandmother's face could show up in images that look very different from $\mathbf{v}$ because of viewpoint, lighting, facial expression, other objects or people in the image, and other causes of discrepancy.

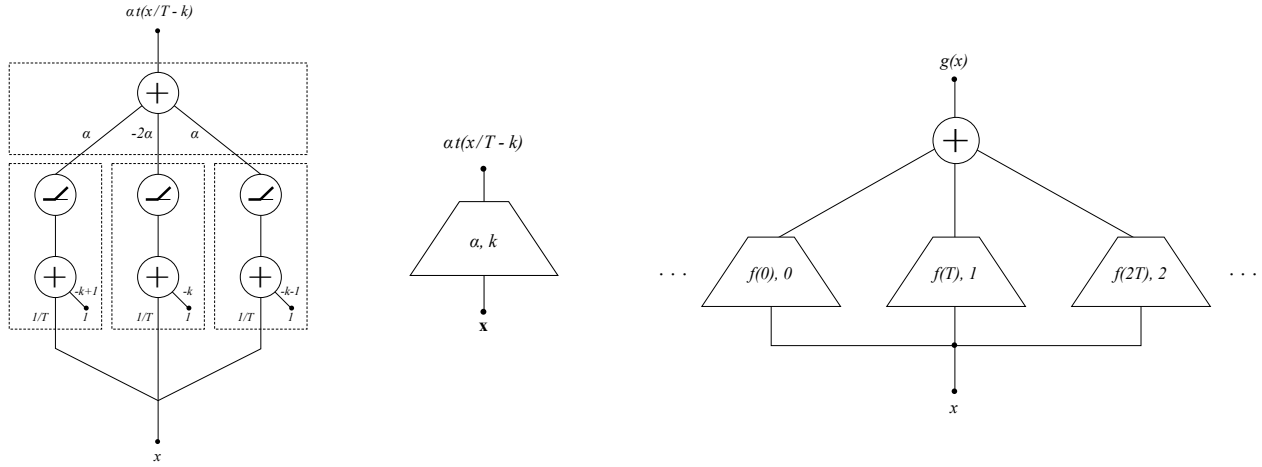

Figure 5: The function $\alpha t(x/T - k)$ can be implemented with a two-layer network module as shown on the left. This entire module is abstracted into a single block with parameters α and k (center). Enough of these blocks, with suitable parameters, can be arranged in parallel (right) to form a universal approximation network for functions from an interval of real numbers to the reals. The final summation in the diagram on the right can be "pushed inside" the blocks because addition is associative. As a result, the output layer in the approximation network has as many inputs as there are blocks.

Instead, observe that faces typically have eyes, noses, ears, hair, and wrinkles—especially for an older person. These features can be analyzed in turn in terms of image edges, corners, curved segments, small dark regions, and so forth. This suggest building a *hierarchy* of patterns, where higher-level ones are made of lower level ones, and only the lowest-level patterns are made directly out of pixels from the input image. At each level, each pattern should then take only a relatively small number of inputs in consideration, from a relatively small and compact part of the image: each neuron should have a *local receptive field*.

In addition, many of the lower-level features appear multiple times in images and across objects, and this suggests that the same neuron could score patterns of its own type no matter where they appear in an image: the same detector could be *reused* over its domain.

Finally, if higher-level patterns are somehow made somewhat insensitive to exactly where in the image the relevant lower-level patterns occur, then the overall system would be able to recognize your grandmother's face even in the presence of at least some amount of variation. A hierarchy with many levels may be able to achieve this even more easily, since a small amount of *resilience to spatial variation* in each layer might result in more significant resilience once it is compounded across layers.

These notions of locality, reuse, and resilience to spatial variations suggest imposing the following structure on a neural-net layer [6, 5].

- Think of the input $\mathbf{x}$ as a two-dimensional array, one entry per pixel, rather than a vector, so that the notion of locality is more readily expressed.
- Group the activations a_i (the entries of $\mathbf{a}$) into M *maps*: each map takes care of one type of pattern, each pattern has a small receptive field, there is a neuron for very many of the possible receptive field in the image, and all the neurons in the same map have the same weights and activation function. A pattern with a small receptive field is called a *feature*, so the M maps are called *feature maps*.

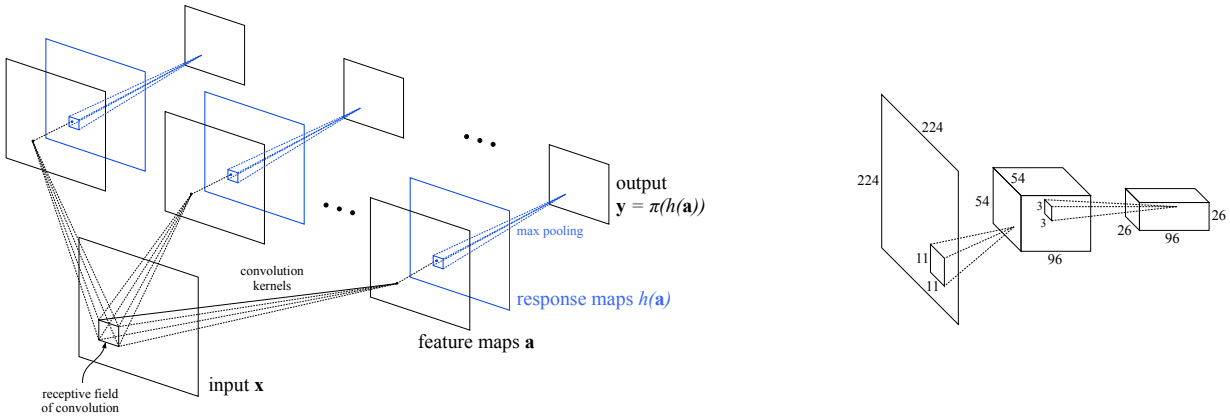

Figure 6: (Left) The structure of a convolutional neural-net layer. (Right) In the literature, neural nets with many layers are drawn with each layer shown in a more compact way than on the left, although there is no standard format. Typically, the maps are stacked in a block, as shown here, rather than drawn side-to-side. Sometimes, max-pooling is only mentioned and not shown explicitly.

This *convolutional organization* is illustrated in figure 6, and is now described with terminology that relates to the first layer in a net. Let the input image be square and with $N \times N$ pixels, so that $D = N^2$, and let the receptive field of a neuron be an $R \times R$ square of pixels where $R = 2H + 1$ is a small odd integer, maybe 5. If the receptive field is to be fully contained in the image, its upper-left corner can be anywhere at row 1 through $N - R + 1$ and column 1 through $N - R + 1$. Consider all these positions sampled with a *stride* s, so that only rows $1, 1 + s, 1 + 2s, \ldots$ are considered, and similarly for the columns.[4] Then, each feature map has one activation per receptive field in the input image, and is therefore itself an array (or an "image") of size $N^{(1)} \times N^{(1)}$, where

$$N^{(1)} = \left\lfloor \frac{N - R}{s} + 1 \right\rfloor . \tag{2}$$

Each activation in every feature map is the inner product of the pixels in its receptive field with a vector that depends on the map but not on the receptive field:

$$a_{m,i_1,i_2} = \mathbf{w}_m^T \tilde{\mathbf{x}}^{(i_1,i_2)} \quad \text{where} \quad \mathbf{x}^{(i_1,i_2)} = \begin{bmatrix} x_{i_1-H,i_2-H} & \cdots & x_{i_1-H,i_2+H} \\ \vdots & & \vdots \\ x_{i_1+H,i_2-H} & \cdots & x_{i_1+H,i_2+H} \end{bmatrix}$$

and, as usual,

$$\tilde{\mathbf{x}}^{(i_1,i_2)} = \begin{bmatrix} \mathbf{x}^{(i_1,i_2)}(:) \\ 1 \end{bmatrix}$$

where the colon notation, borrowed from MATLAB, strings the entries of $\mathbf{x}^{(i_1,i_2)}$ into a column vector. Of course, the vector of *all* activations is still of the form

$$\mathbf{a} = W\tilde{\mathbf{x}}$$

[4] A stride $s = 1$ is not uncommon.

where $\mathbf{x}$ is a vector with $D = N^2$ entries, and $\tilde{x}$ has one more entry than that, as usual, to handle bias. However, now W is a column of M blocks—one per feature map. Each block has size $(N^{(1)})^2 \times (N^2 + 1)$ and is sparse, with at most $R^2 + 1$ nonzero weights in each row. Furthermore, the nonzero entries have the same values in all the rows of the same block. Of course, the matrix W is not built explicitly.

This computation is called *convolutional* because when the stride is 1 each feature map is the convolution (or more precisely the correlation) of the input array with the array of weights associated with the map's receptive field.[5] The weights for a feature map are sometimes called the map's *kernel*. The (common) activation function h is then applied to each feature map, entry by entry, to obtain the neural responses for the layer.

As a result of this structure, the number of distinct parameters in W drops dramatically, from about N^4 for a fully-connected layer with equal input and output sizes to $M(R^2 + 1)$ if the image is black-and-white. For color images, the count drops from about $9N^4$ to $3M(R^2+1)$, where each color band gets a separate set of feature maps. For instance [5], for a $224 \times 224 \times 3$-pixel color image and 96 maps each with an 11×11 receptive field, the drop is from about 22.7 billion to a mere $96 \times 11^2 = 11{,}616$ parameters!

- Reduce the size of each feature map $h(\mathbf{a}_m(\mathbf{x}))$ by *max-pooling*. Specifically, square receptive fields are defined in each feature map in turn, with a size and stride that is common to all the feature maps. A new, smaller feature map is then computed whose values are each the maximum value in its receptive field. If max-pooling is denoted by the function $\pi(\cdot)$, the output from the whole layer is[6]

$$\mathbf{y} = [h_y(\mathbf{a}_1^T(\mathbf{x})), \ldots, h_y(\mathbf{a}_M^T(\mathbf{x}))]^T = [\pi(h(\mathbf{a}_1^T(\mathbf{x}))), \ldots, \pi(h(\mathbf{a}_M^T(\mathbf{x})))]^T;.$$

  In addition to reducing the size of the feature maps in $\mathbf{y}$, max-pooling makes the output of the layer somewhat less sensitive to the exact location of the features in the image. For instance, with a 3×3 receptive field for max-pooling and a stride of 3 (no overlap between pools), the output of the maximum is oblivious to which of those 3^2 activations produced the final output from the layer. In other words, max-pooling achieves some degree of *translation-invariance*, similarly to what we saw in the computation of HOG features. If the same is done in every layer of a deep network, the amounts of invariance add up.

In one example [5] illustrated in Figure 6, the first layer of a convolutional net has a 224×224 color image as input, so that the input dimensionality is $D = 224^2 \times 3 = 150{,}528$. There are 96 kernels, feature maps, and response maps, 32 per color. The convolution kernels have receptive fields of size 11×11 pixels, so each output pixel in each color channel is computed from $11^2 = 121$ input pixels which are combined through 122 weights, including a bias value. The stride for computing activations is 4 pixels in each direction, so the activation maps and feature maps are 54×54 pixels each[7]. Max-pooling uses 3×3-pixel receptive fields and a stride of 2 pixels, and produces output maps of size 26×26 pixels[8]. Both map sizes are computed from equation (2).

The set of activation maps computed by the 96 kernels is a $54 \times 54 \times 96$ block, rather than a single image, and max pooling is applied to each of the 96 slices in this block. If a subsequent layer applies

[5] With a stride greater than 1, the feature map can be seen as the result of a convolution followed by sampling, although this implementation would be inefficient: There is no need to compute convolution values that are then omitted by sampling.

[6] The function π is no longer $\mathbb{R} \to \mathbb{R}$.

[7] The paper itself [5] states that the activation maps have size 55×55. I am not sure about the reason for this discrepancy.

[8] Consistently with 55×55 feature maps before max-pooling, the paper [5] gives 27×27 output maps.

convolution to the resulting $26 \times 26 \times 96$ output block from this layer, that convolution kernel is in general three-dimensional, $m \times n \times p$, although p can be equal to 1 for an effectively two-dimensional kernel. In each of the three dimensions, convolution can be set up with one of the `'same'`, `'valid'`, or `'full'` options we saw when we studied convolution, thereby producing further blocks of activation maps of varying sizes.

For the layer in the Figure, the output dimensionality is $D^{(1)} = 26^2 \times 96 = 64{,}896$, a bit less than half of the input dimensionality $D = 224^2 \times 3 = 150{,}528$. On the other hand, the map *resolution* decreases more than eightfold, from 224 to 26 pixels on each side. The representation of the image has become more abstract, changing from a pure pixel-by-pixel list of its colors to a coarser map of how much each of 32 features is present at each location in the image and in each color channel.

4 Deep Convolutional Neural Nets

The architecture of a neural-net layer embodies the principles of feature reuse, locality, and translation-invariance. *Deep* Convolutional Neural Nets (CNNs) are CNNs with many layers, and reflect the principle of hierarchy. After several convolutional layers, deep CNNs typically add one or a few fully-connected layers, that is, layers where the weight matrix W is dense. The reasons for doing so are somewhat mixed and not entirely compelling, but are nonetheless plausible: Far away from the input, spatial location is both partially lost and relatively irrelevant to, say, recognition, so the local receptive fields of CNNs are no longer useful. In addition, signals in late stages of a deep net have relatively low dimensionality, and one can then better afford the greater representational flexibility that a fully-connected layer carries.

The output from a deep CNN is fed to a computation that depends on the purpose of the net. For regression, for instance, the outputs may be used as they are. For classification, one could use the outputs as inputs to a support vector machine or random forest or, if the classifier requires input quantities that behave like probabilities, the output stage could be a *softmax* function,

$$\mathbf{z} = \sigma(\mathbf{y}) = \frac{\exp(\mathbf{y})}{\mathbf{1}^T \exp(\mathbf{y})}$$

where $\mathbf{1}$ is a column vector of ones. Pragmatically speaking, the exponential makes all quantities positive, and normalization makes sure the entries of $\mathbf{z}$ add up to 1. More formally[9], the softmax function can be viewed as a multidimensional generalization of the sigmoid function used in logistic regression [1]. The reason why this function is called "softmax" is that if one of the y_i entries, say y_{i_0}, is much bigger than all of the others, then $\mathbf{1}^T \exp(\mathbf{y}) \approx \exp(y_{i_0})$, so that

$$\mathbf{z}_{i_0} \approx 1 \quad \text{and} \quad \mathbf{z}_i \approx 0 \quad \text{for} \quad i \neq i_0 \ ,$$

and the function effectively acts as an indicator of the largest entry in $\mathbf{y}$. Somewhat more precisely, and quite obviously,

$$\lim_{\alpha \to \infty} \mathbf{y}^T \sigma(\alpha \mathbf{y}) = \max(\mathbf{y}) \ .$$

In summary, a deep CNN performs the following computation:

$$\begin{aligned}
\mathbf{x}^{(0)} &= \mathbf{x} \\
\mathbf{x}^{(\ell)} &= \pi(h(W^{(\ell)} \tilde{\mathbf{x}}^{(\ell-1)})) \quad \text{for} \quad \ell = 1, \ldots, L_c \\
\mathbf{x}^{(\ell)} &= h(W^{(\ell)} \tilde{\mathbf{x}}^{(\ell-1)}) \quad \text{for} \quad \ell = L_c + 1, \ldots, L \\
\mathbf{y} &= h_y(\mathbf{x}^{(L)})
\end{aligned}$$

[9] But not necessarily more appropriately.

where the output activation function h_y may be the identity, softmax, or other function. The matrices $W^{(\ell)}$ have $D^{(\ell)}$ rows and $D^{(\ell-1)}+1$ columns with $D^{(0)}=D$ and $D^{(\ell)}$ for $\ell>0$ being equal to the number of outputs (or activations) in layer number ℓ. The first L_c layers are convolutional and the remaining ones are fully-connected.

References

[1] C. M. Bishop. *Pattern Recognition and Machine Learning*. Springer, 2006.

[2] L. Deng and D. Yu. *Deep Learning: Methods and Applications*, volume 7(3-4) of *Foundations and Trends in Signal Processing*. Now Publishers, 2014.

[3] I. Goodfellow, Y. Bengio, and A. Courville. *Deep Learning*. The MIT Press, Cambridge, MA, 2016.

[4] V. Y. Kreinovic. Arbitrary nonlinearity is sufficient to represent all functions by neural networks: a theorem. *Neural Networks*, 4:381–383, 1991.

[5] A. Krizhevsky, I. Sutskever, and G. Hinton. ImageNet classification with deep convolutional neural networks. In *Advances in Neural Information Processing Systems*, volume 25, pages 1106–1114, 2012.

[6] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, November 1998.

[7] S. Sonoda and N. Murata. Neural network with unbounded activations is universal approximator. Technical Report 1505.3654 [cs.NE], arXiv, 2015.