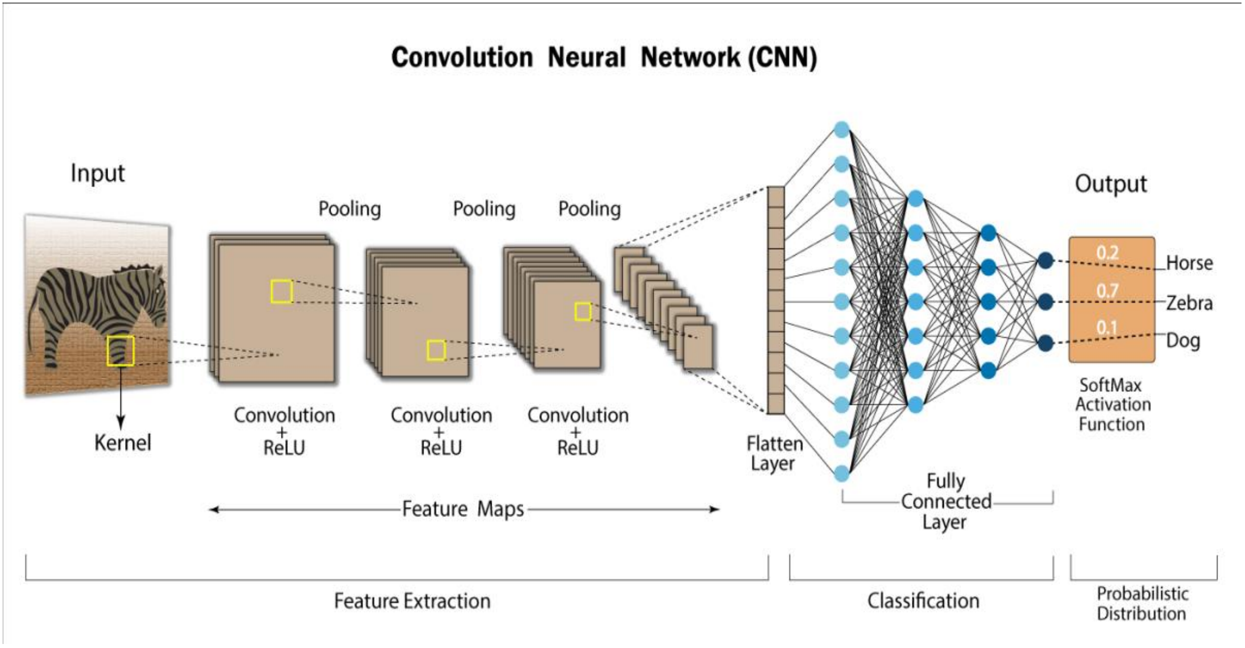
UNIT - III

Convolutional Networks : The Convolution Operation, Motivation, Pooling, Convolution and Pooling as an Infinitely Strong Prior, Variants of the Basic Convolution Function, Structured Outputs, Data Types, Efficient Convolution Algorithms, Random or Unsupervised Features.

Convolutional Networks

Convolutional networks, also known as convolutional neural networks or CNNs, are a specialized kind of neural network for processing data that has a known, grid-like topology.
Convolutional networks have been tremendously successful in practical applications. The name "convolutional neural network" indicates that the network employs a mathematical operation called convolution. Convolution is a specialized kind of linear operation.
Convolutional networks are simply neural networks that use convolution in place of general matrix multiplication in at least one of their layers.



Convolutional Neural Networks are a special type of feed-forward artificial neural network in which the connectivity pattern between its neuron is inspired by the visual cortex.

Layers used to build ConvNets:

A complete Convolution Neural Networks architecture is also known as covnets. A covnets is a sequence of layers, and every layer transforms one volume to another through a differentiable function.

- **Input Layers:** It's the layer in which we give input to our model. In CNN, Generally, the input will be an image or a sequence of images.
- **Convolutional Layers:** This is the layer, which is used to extract the feature from the input dataset. It applies a set of learnable filters known as the kernels to the input images. The filters/kernels are smaller matrices usually 2×2, 3×3, or 5×5 shape. it slides over the input image data and computes the dot product between kernel weight and the corresponding input image patch. The output of this layer is referred as feature maps.
- **Activation Layer:** activation layers add nonlinearity to the network. it will apply an element-wise activation function to the output of the convolution layer. Some common activation functions are RELU: max(0, x), Tanh, Leaky RELU, etc.
- **Pooling layer:** This layer is periodically inserted in the covnets and its main

function is to reduce the size of volume which makes the computation fast reduces memory and also prevents overfitting. Two common types of pooling layers are max pooling and average pooling.

- **Flattening:** The resulting feature maps are flattened into a one-dimensional vector after the convolution and pooling layers so they can be passed into a completely linked layer for categorization or regression. Fully Connected Layers: It takes the input from the previous layer and computes the final classification or regression task.
- **Output Layer:** The output from the fully connected layers is then fed into a logistic function for classification tasks like sigmoid or softmax which converts the output of each class into the probability score of each class.

Motivation

Convolution leverages three important ideas that can help improve a machine learning system: ***sparse interactions*, *parameter sharing* and *equivariant representations*.** Moreover, convolution provides a means for working with inputs of variable size. We now describe each of these ideas in turn.

1. Traditional neural network layers use matrix multiplication by a matrix of parameters with a separate parameter describing the interaction between each input unit and each output unit. This means every output unit interacts with every input unit. Convolutional networks, however, typically have ***sparse interactions*** (also referred to as *sparse connectivity* or *sparse weights*). This is accomplished by making the kernel smaller than the input.
2. ***Parameter sharing*** refers to using the same parameter for more than one function in a model. In a traditional neural net, each element of the weight matrix is used exactly once when computing the output of a layer.In a convolutional neural net, each member of the kernel is used at every position of the input. The parameter sharing used by the convolution operation means that rather than learning a separate set of parameters for every location, we learn only one set.
3. In the case of convolution, the particular form of parameter sharing causes the layer to have a property called ***equivariance*** to translation. To say a function is equivariant means that if the input changes, the output changes in the same way.

The Convolution Operation

convolution is an operation on two functions of a real valued argument.

Example : Suppose we are tracking the location of a spaceship with a laser sensor. Our laser sensor provides a single output x(t), the position of the spaceship at time t. Both x and t are real-valued.

Now suppose that our laser sensor is somewhat noisy. To obtain a less noisy estimate of the spaceship's position, we would like to average together several measurements.

Of course, more recent measurements are more relevant, so we will want this to be a weighted average that gives more weight to recent measurements. We can do this with a weighting function w(a), where a is the age of a measurement.

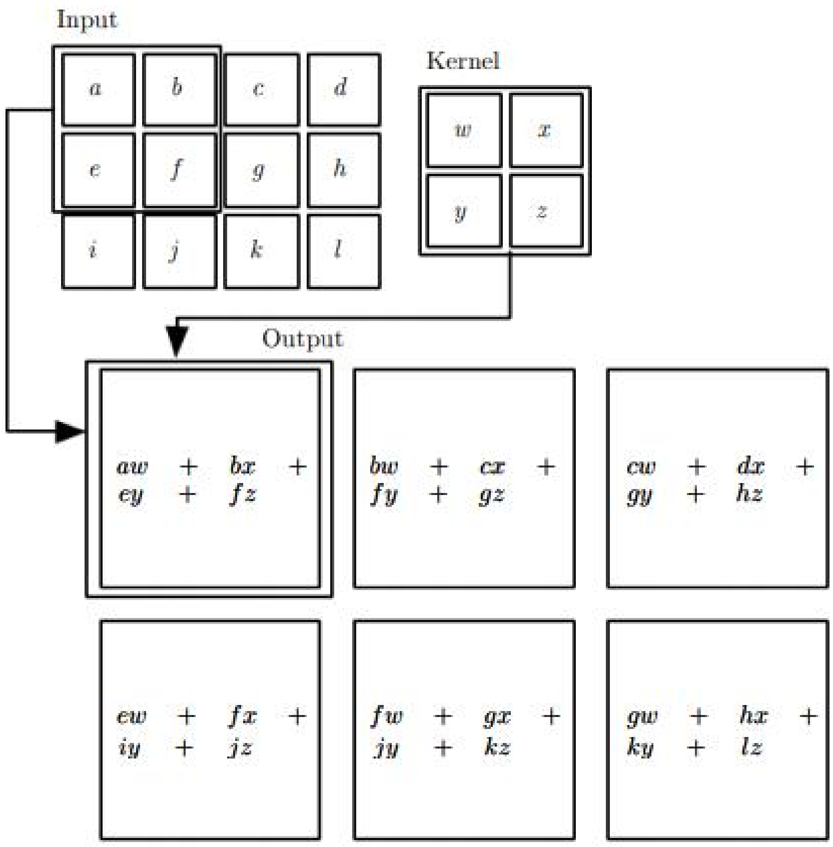
If we apply such a weighted average operation at every moment, we obtain a new function s providing a smoothed estimate of the position of the spaceship:

$$s(t) = \int x(a)w(t-a)da$$

This operation is called convolution. The convolution operation is typically denoted with an asterisk:

$$s(t) = (x * w)(t)$$

In convolutional network terminology, the first argument to the convolution is often referred to as the input and the second argument as the kernel or filter. The output is sometimes referred to as the feature map.



1. Padding

padding is a technique used to preserve the spatial dimensions of the input image after convolution operations on a feature map. Padding involves adding extra pixels around the border of the input feature map before convolution.

This can be done in two ways:

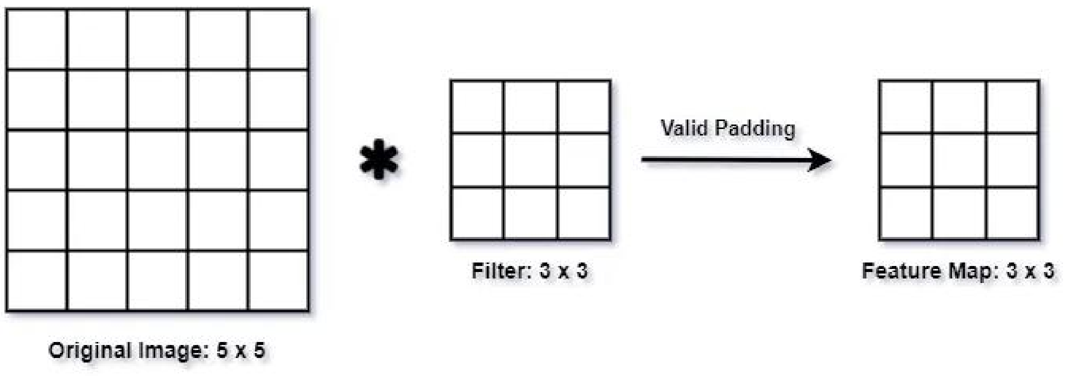
Valid Padding: In the valid padding, no padding is added to the input feature map, and the output feature map is smaller than the input feature map. This is useful when we want to reduce the spatial dimensions of the feature maps.

[(nxn)*(fxf)]—>(n-f+1 * n-f+1) image]

where,

- nxn is the dimension of input image

- fxf is kernel size
- n-f+1 is output image size
- * represents a convolution operation.

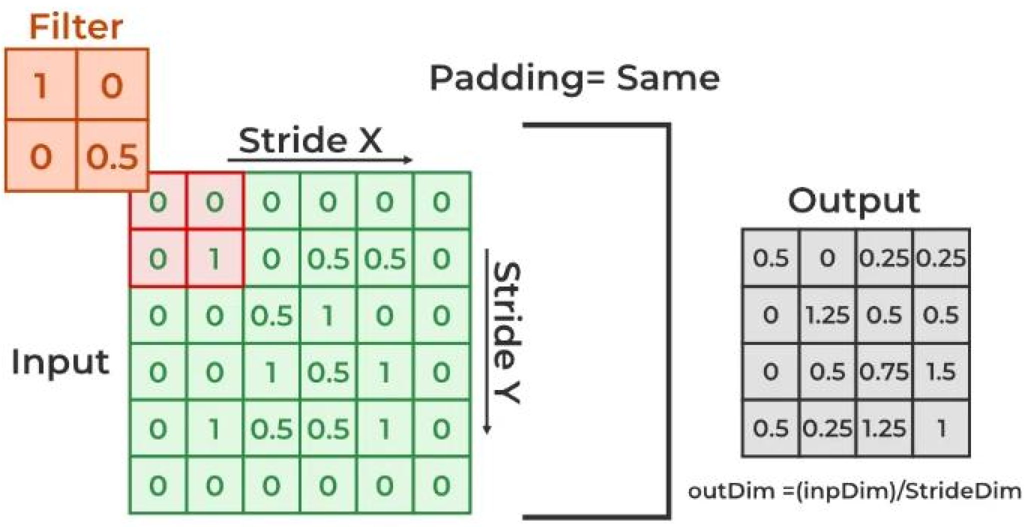


Valid Padding Output

<u>Same Padding:</u> In the same padding, padding is added to the input feature map such that the size of the output feature map is the same as the input feature map. This is useful when we want to preserve the spatial dimensions of the feature maps

[(n + 2p) x (n + 2p) image] * [(f x f) filter] —> [(n x n) image]

which gives **p = (f – 1) / 2 (because n + 2p – f + 1 = n).**



Problem With Convolution Layers Without Padding

For a grayscale (n x n) image and (f x f) filter/kernel, the dimensions of the image resulting from a convolution operation is **(n – f + 1) x (n – f + 1)**.

For example, for an (8 x 8) image and (3 x 3) filter, the output resulting after the convolution operation would be of size (6 x 6). Thus, the image shrinks every time a convolution operation is performed. This places an upper limit to the number of times such an operation could be performed before the image reduces to nothing thereby precluding us from building deeper networks.

Also, the pixels on the corners and the edges are used much less than those in the middle.

Effect Of Padding On Input Images

Padding is simply a process of adding layers of zeros to our input images so as to avoid the problems mentioned above through the following changes to the input image.

Padding prevents the shrinking of the input image.

***p** = number of layers of zeros added to the border of the image, then*

***(n* n)** image —> **(n + 2p) * (n + 2p)** image after padding.*

***(n + 2p) * (n + 2p) * (f x f) ——> outputs (n + 2p – f + 1) * (n + 2p – f + 1)** images*

For example, by adding one layer of padding to an (8 x 8) image and using a (3 x 3) filter we would get an (8 x 8) output after performing a convolution operation.

2. <u>Striding :</u>

Stride is the distance, or number of pixels, that the kernel moves over the input matrix. While stride values of two or greater is rare, a larger stride yields a smaller output.

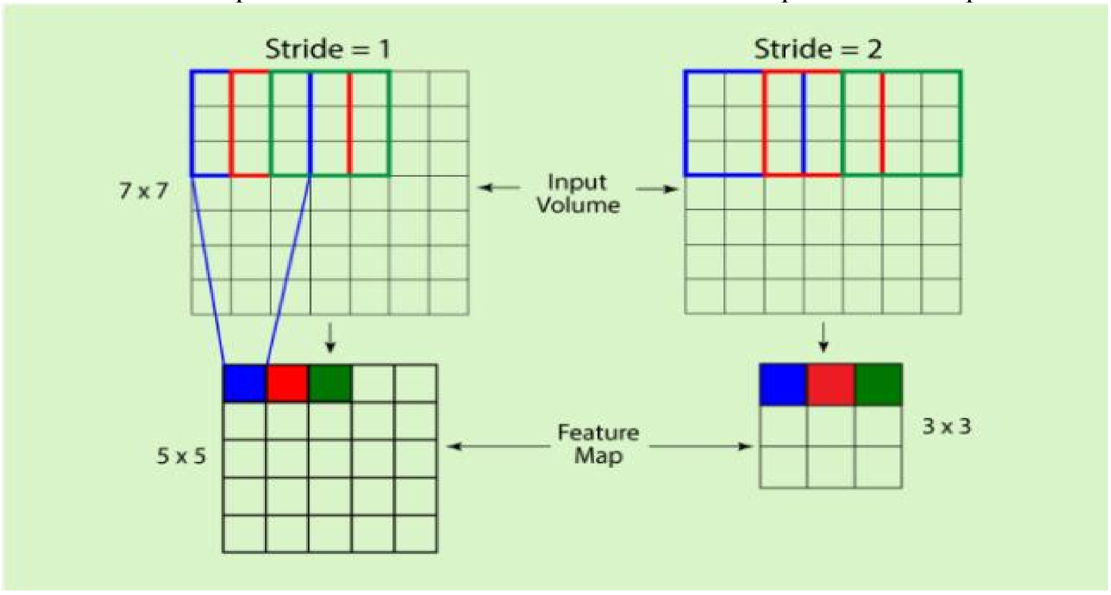
A stride of 1 means the filter moves one pixel at a time, while a stride of 2 means it moves two pixels at a time, and so on.

The output size of a convolutional layer in a CNN can be calculated using the following formula:

$$\text{Output size} = \frac{\text{Input size} - \text{Filter size} + 2 \times \text{Padding}}{\text{Stride}} + 1$$

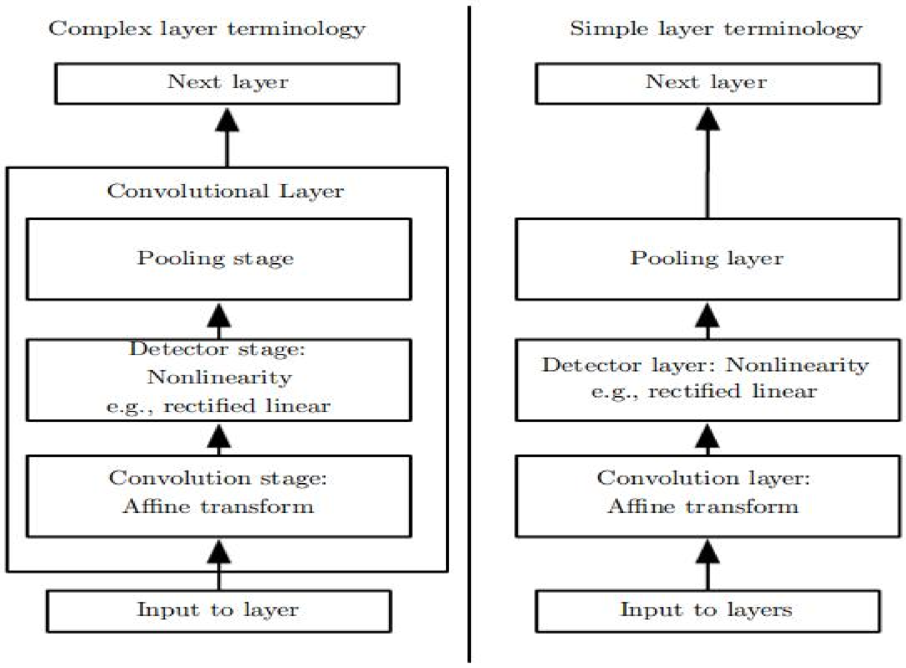
Where

- Input size: Size of the input feature map (in terms of width or height).
- Filter size: Size of the convolutional filter (in terms of width or height).
- Padding: The number of pixels added around the input feature map. Padding is optional and is typically used to preserve spatial dimensions.
- Stride: The step size of the filter as it moves across the input feature map



3.Pooling

A typical layer of a convolutional network consists of three stages. In the first stage, the layer performs several convolutions in parallel to produce a set of linear activation's. In the second stage, each linear activation is run through a
nonlinear activation function, such as the rectified linear activation function. This stage is sometimes called the *detector* stage. In the third stage, we use a *pooling function* to modify the output of the layer further.


Complex layer terminology
Next layer
Convolutional Layer
Pooling stage
Detector stage: Nonlinearity e.g., rectified linear
Convolution stage: Affine transform
Input to layer
Simple layer terminology
Next layer
Pooling layer
Detector layer: Nonlinearity e.g., rectified linear
Convolution layer: Affine transform
Input to layers

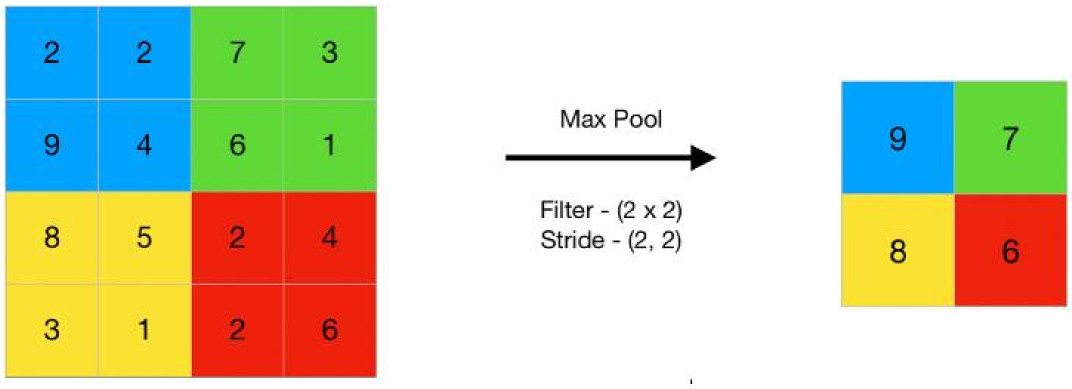
A pooling function replaces the output of the net at a certain location with a summary statistic of the nearby outputs. For example, the ***max pooling*** (Zhou and Chellappa, 1988) operation reports the maximum output within a rectangular neighborhood. Other popular pooling functions include the average of a rectangular neighborhood, the *L*2 norm of a rectangular neighborhood, or a weighted average based on the distance from the central pixel.
In all cases, pooling helps to make the representation become approximately ***invariant*** to small translations of the input. Invariance to translation means that if we translate the input by a small amount, the values of most of the pooled outputs do not change.
Invariance to local translation can be a very useful property if we care more about whether some feature is present than exactly where it is.
For example, when determining whether an image contains a face, we need not know the location of the eyes with pixel-perfect accuracy, we just need to know that there is an eye on the left side of the face and an eye on the right side of the face.
The use of pooling can be viewed as adding an infinitely strong prior that the function the layer learns must be invariant to small translations. When this assumption is correct, it can greatly improve the statistical efficiency of the network. **Example :**

Efficiency of edge detection. The image on the right was formed by taking each pixel in the original image and subtracting the value of its neighboring pixel on the left. This shows the strength of all of the vertically oriented edges in the input image, which can be a useful operation for object detection.

There are several types of pooling layers commonly used in CNNs:

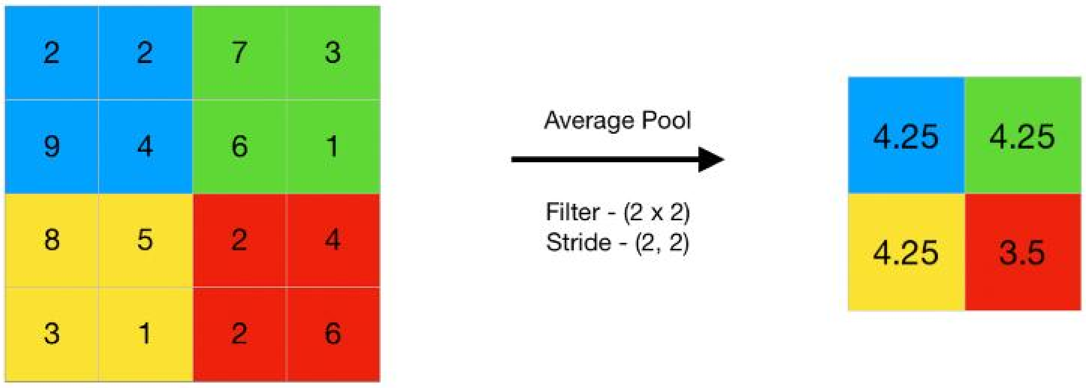
- **<u>Max Pooling</u>**

Max pooling is a pooling operation that selects the maximum element from the region of the feature map covered by the filter. Thus, the output after max-pooling layer would be a feature map containing the most prominent features of the previous feature map.



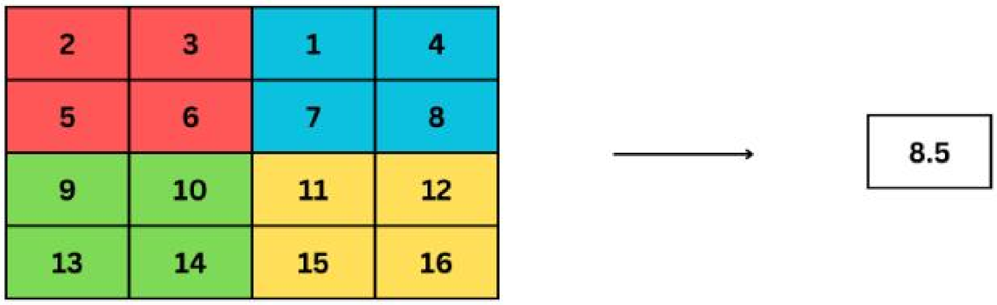
- **<u>Average Pooling</u>**

Average pooling computes the average of the elements present in the region of feature map covered by the filter. Thus, while max pooling gives the most prominent feature in a particular patch of the feature map, average pooling gives the average of features present in a patch.



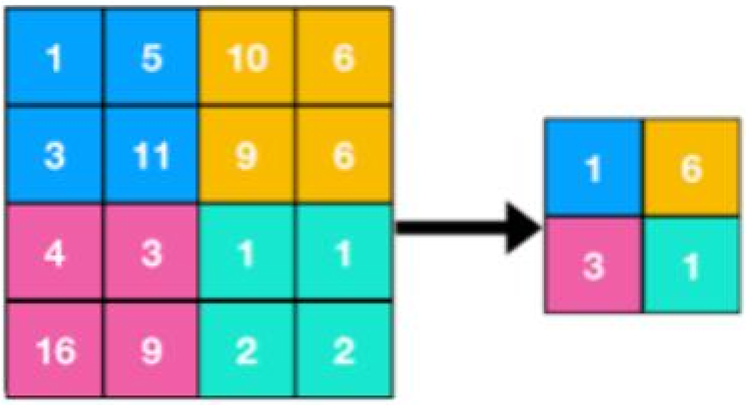
- **<u>Global Pooling</u>**

Global average pooling is a variation of average pooling where instead of applying pooling to each spatial location independently, it computes the average of the entire feature map. This results in a single value for each channel of the feature map, effectively reducing the spatial dimensions to 1x1. Global average pooling is often used in the final layers of CNNs for classification tasks, where the spatial information is aggregated to produce a prediction.



- **Min Pooling:**

Min pooling is less common compared to max and average pooling. In min pooling, for each region of the input feature map, the minimum value is retained and the rest are discarded. Min pooling may be useful in certain scenarios, but it is not as widely used as max or average pooling.



Convolution and Pooling as an Infinitely Strong Prior

Priors can be considered weak or strong depending on how concentrated the probability density in the prior is. A weak prior is a prior distribution with high entropy, such as a Gaussian distribution with high variance. Such a prior allows the data to move the parameters more or less freely. A strong prior has very low entropy, such as a Gaussian distribution with low variance. Such a prior plays a more active role in determining where the parameters end up.
An infinitely strong prior places zero probability on some parameters and says that these parameter values are completely forbidden, regardless of how much support the data gives to those values.
This infinitely strong prior says that the weights for one hidden unit must be identical to the weights of its neighbor, but shifted in space. The prior also says that the weights must be zero, except for in the small, spatially contiguous receptive field assigned to that hidden unit. Overall, we can think of the use of convolution as introducing an infinitely strong prior probability distribution over the parameters of a layer.
Of course, implementing a convolutional net as a fully connected net with an infinitely strong prior would be extremely computationally wasteful. But thinking of a convolutional net as a fully connected net with an infinitely strong prior can give us some insights into how convolutional nets work.

Variants of the Basic Convolution Function

- Convolution in the context of NN means an operation that consists of many applications of convolution in parallel.
- **Kernel K** with element $K_{i,j,k,l}$ giving the connection strength between a unit in channel i of output and a unit in channel j of the input, with an offset of k rows and l columns between the output unit and the input unit.

- Input: $V_{i,j,k}$ with channel i, row j and column k
- Output Z same format as V
- Use 1 as first entry

- **Full Convolution**

  0 Padding 1 stride

$$Z_{i,j,k}=\sum_{l,m,n} V_{l,j+m-1,k+n-1} K_{i,l,m,n}$$

  0 Padding s stride

$$Z_{i,j,k}=c(K,V,s)_{i,j,k}=\sum_{l,m,n}[V_{l,s*(j-1)+m,s*(k-1)+n} K_{i,l,m,n}]$$

Convolution with a stride greater than 1 pixel is equivalent to conv with 1 stride followed by downsampling:

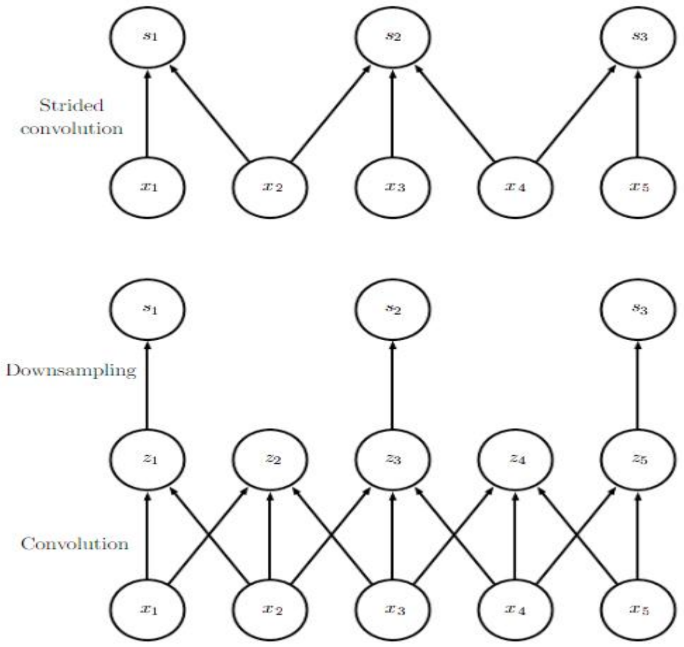


Figure 9.12: Convolution with a stride. In this example, we use a stride of two. *(Top)*Convolution with a stride length of two implemented in a single operation. *(Bottom)*Convolution with a stride greater than one pixel is mathematically equivalent to convolution with unit stride followed by downsampling. Obviously, the two-step approach involving downsampling is computationally wasteful, because it computes many values that are then discarded.

Some 0 Paddings and 1 stride

Without 0 paddings, the width of representation shrinks by one pixel less than the kernel width at each layer. We are forced to choose between shrinking the spatial extent of the network rapidly and using small kernel. 0 padding allows us to control the kernel width and the size of the output independently.

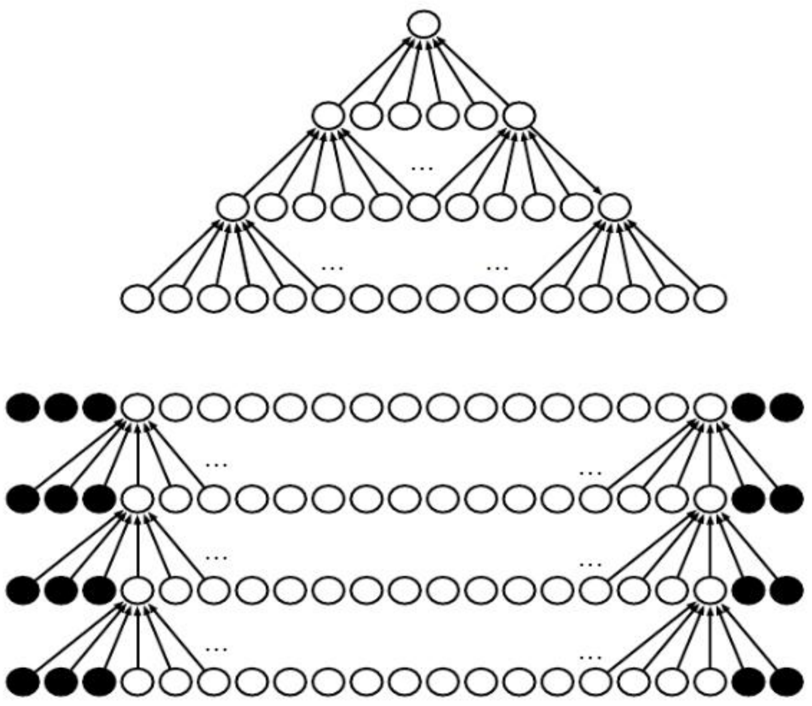


Figure 9.13: The effect of zero padding on network size. Consider a convolutional network with a kernel of width six at every layer. In this example, we do not use any pooling, so only the convolution operation itself shrinks the network size. *(Top)*In this convolutional network, we do not use any implicit zero padding. This causes the representation to shrink by five pixels at each layer. Starting from an input of sixteen pixels, we are only able to have three convolutional layers, and the last layer does not ever move the kernel, so arguably only two of the layers are truly convolutional. The rate of shrinking can be mitigated by using smaller kernels, but smaller kernels are less expressive, and some shrinking is inevitable in this kind of architecture. *(Bottom)*By adding five implicit zeros to each layer, we prevent the representation from shrinking with depth. This allows us to make an arbitrarily deep convolutional network.

Special case of 0 padding:

- **Valid:** no 0 padding is used. Limited number of layers.
- **Same:** keep the size of the output to the size of input. Unlimited number of layers. Pixels near the border influence fewer output pixels than the input pixels near the center.
- **Full:** Enough zeros are added for every pixels to be visited k (kernel width) times in each direction, resulting width **m + k - 1.** Difficult to learn a single kernel that performs well at all positions in the convolutional feature map.

Usually the optimal amount of 0 padding lies somewhere between **'Valid' or 'Same'**

- **Unshared Convolution**

In some case when we do not want to use convolution but want to use locally connected layer. We use **Unshared convolution**. Indices into weight W

- i: the output channel
- j: the output row;
- k: the output column
- l: the input channel
- m: row offset within input
- n: column offset within input

$$Z_{i,j,k}=\sum_{l,m,n}[V_{l,i+m-1,j+n-1}W_{i,j,k,l,m,n}]$$

Comparison on local connections, convolution and full connection

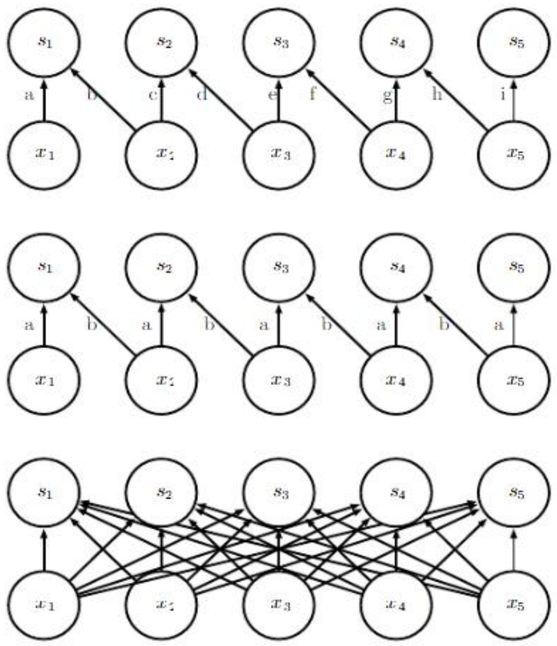


Figure 9.14: Comparison of local connections, convolution, and full connections. *(Top)*A locally connected layer with a patch size of two pixels. Each edge is labeled with a unique letter to show that each edge is associated with its own weight parameter. *(Center)*A convolutional layer with a kernel width of two pixels. This model has exactly the same connectivity as the locally connected layer. The difference lies not in which units interact with each other, but in how the parameters are shared. The locally connected layer has no parameter sharing. The convolutional layer uses the same two weights repeatedly across the entire input, as indicated by the repetition of the letters labeling each edge. *(Bottom)*A fully connected layer resembles a locally connected layer in the sense that each edge has its own parameter (there are too many to label explicitly with letters in this diagram). It does not, however, have the restricted connectivity of the locally connected layer.

Useful when we know that each feature should be a function of a small part of space, but no reason to think that the same feature should occur across all the space. eg: look for mouth only in the bottom half of the image.

It can be also useful to make versions of convolution or local connected layers in which the connectivity is further restricted, eg: constrain each output channel i to be a function of only a subset of the input channel.

Adv: * reduce memory consumption * increase statistical efficiency * reduce computation for both forward and backward prop.

- **Tiled Convolution**

Learn a set of kernels that we rotate through as we move through space. Immediately neighboring locations will have different filters, but the memory requirement for storing the parameters will increase by a factor of the size of this set of kernels. Comparison on locally connected layers, tiled convolution and standard convolution:

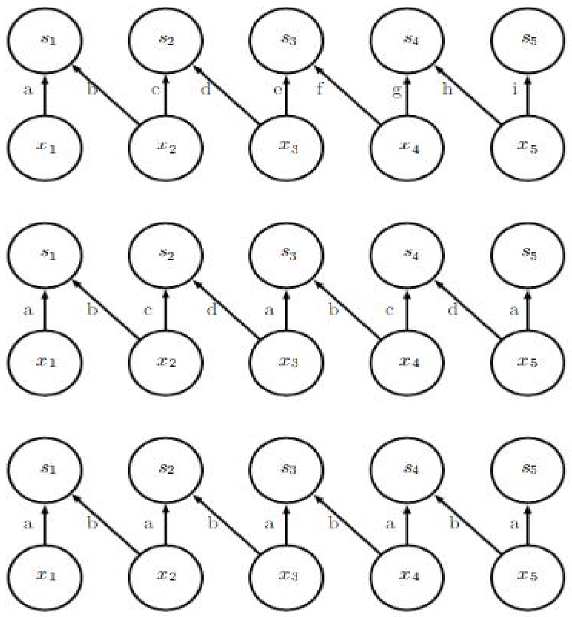


Figure 9.16: A comparison of locally connected layers, tiled convolution, and standard convolution. All three have the same sets of connections between units, when the same size of kernel is used. This diagram illustrates the use of a kernel that is two pixels wide. The differences between the methods lies in how they share parameters. *(Top)* A locally connected layer has no sharing at all. We indicate that each connection has its own weight by labeling each connection with a unique letter. *(Center)* Tiled convolution has a set of t different kernels. Here we illustrate the case of $t = 2$. One of these kernels has edges labeled "a" and "b," while the other has edges labeled "c" and "d." Each time we move one pixel to the right in the output, we move on to using a different kernel. This means that, like the locally connected layer, neighboring units in the output have different parameters. Unlike the locally connected layer, after we have gone through all t available kernels, we cycle back to the first kernel. If two output units are separated by a multiple of t steps, then they share parameters. *(Bottom)* Traditional convolution is equivalent to tiled convolution with $t = 1$. There is only one kernel, and it is applied everywhere, as indicated in the diagram by using the kernel with weights labeled "a" and "b" everywhere.

Structured Outputs :

Convolution networks can be used to output a high-D structured object, rather than just redicting a class label for a classification task or a real value for regression tasks. **Eg:** The model might emit a tensor S where $S_{i,j,k}$ is the probability that pixel (j, k) of the input belongs to class i.

Issue, the output plane can be smaller than input plane. Review:

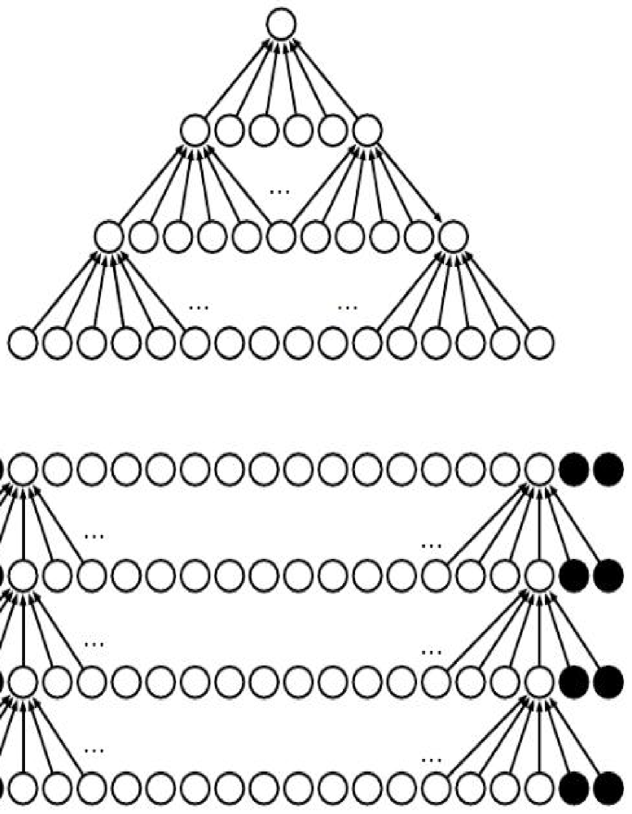


Figure 9.13: The effect of zero padding on network size. Consider a convolutional network with a kernel of width six at every layer. In this example, we do not use any pooling, so only the convolution operation itself shrinks the network size. *(Top)*In this convolutional network, we do not use any implicit zero padding. This causes the representation to shrink by five pixels at each layer. Starting from an input of sixteen pixels, we are only able to have three convolutional layers, and the last layer does not ever move the kernel, so arguably only two of the layers are truly convolutional. The rate of shrinking can be mitigated by using smaller kernels, but smaller kernels are less expressive, and some shrinking is inevitable in this kind of architecture. *(Bottom)*By adding five implicit zeros to each layer, we prevent the representation from shrinking with depth. This allows us to make an arbitrarily deep convolutional network.

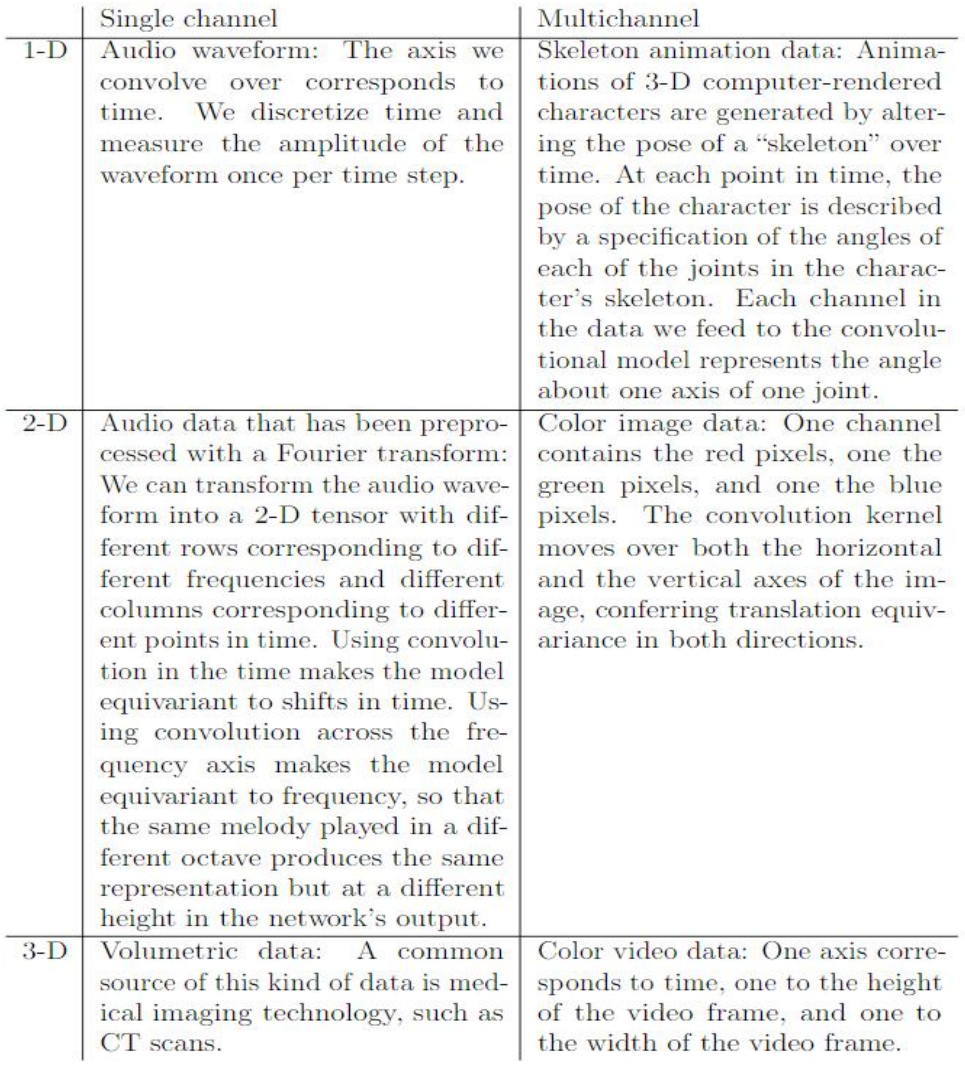
- **Strategy for size reduction issue:**
  - avoid pooling altogether
  - emit a lower-resolution grid of labels
  - pooling operator with unit stride

One strategy for pixel-wise labeling of images is to produce an initial guess of the image label.

1. produce an initial guess of the image labels.
2. refine this initial guess using the interactions between neighboring pixels.

Data Types

The data used with a convolutional network usually consists of several channels, each channel being the observation of a different quantity at some point in space or time.



	Single channel	Multichannel
1-D	Audio waveform: The axis we convolve over corresponds to time. We discretize time and measure the amplitude of the waveform once per time step.	Skeleton animation data: Animations of 3-D computer-rendered characters are generated by altering the pose of a "skeleton" over time. At each point in time, the pose of the character is described by a specification of the angles of each of the joints in the character's skeleton. Each channel in the data we feed to the convolutional model represents the angle about one axis of one joint.
2-D	Audio data that has been preprocessed with a Fourier transform: We can transform the audio waveform into a 2-D tensor with different rows corresponding to different frequencies and different columns corresponding to different points in time. Using convolution in the time makes the model equivariant to shifts in time. Using convolution across the frequency axis makes the model equivariant to frequency, so that the same melody played in a different octave produces the same representation but at a different height in the network's output.	Color image data: One channel contains the red pixels, one the green pixels, and one the blue pixels. The convolution kernel moves over both the horizontal and the vertical axes of the image, conferring translation equivariance in both directions.
3-D	Volumetric data: A common source of this kind of data is medical imaging technology, such as CT scans.	Color video data: One axis corresponds to time, one to the height of the video frame, and one to the width of the video frame.

Table 9.1: Examples of different formats of data that can be used with convolutional networks.

Convolution does not make sense if the input has variable size because it can optionally include different kinds of observations. **For example,** if we are processing college applications, and our features consist of both grades and standardized test scores, but not every applicant took the standardized test, then it does not make sense to convolve the same weights over both the features corresponding to the grades and the features corresponding to the test scores.

Efficient Convolution Algorithms

Modern convolutional network applications often involve networks containing more than one million units. Powerful implementations exploiting parallel computation resources.

However, in many cases it is also possible to speed up convolution by selecting an appropriate convolution algorithm.

Convolution is equivalent to converting both the input and the kernel to the frequency domain using a Fourier transform, performing point-wise multiplication of the two signals, and converting back to the time domain using an inverse Fourier transform. For some problem sizes, this can be faster than the naive implementation of discrete convolution.

When a d-dimensional kernel can be expressed as the outer product of d vectors, one vector per dimension, the kernel is called *separable*.

When the kernel is separable, naive convolution is inefficient. It is equivalent to compose *d one-dimensional* convolutions with each of these vectors.

The composed approach is significantly faster than performing one d-dimensional convolution with their outer product. The kernel also takes fewer parameters to represent as vectors.

If the kernel is w elements wide in each dimension, then naive multidimensional convolution requires $\boldsymbol{O(w^d)}$ runtime and parameter storage space, while separable convolution requires $\boldsymbol{O(w \times d)}$ runtime and parameter storage space. Of course, not every convolution can be represented in this way.

Random or Unsupervised Features

Typically, the most expensive part of convolutional network training is learning the features. The output layer is usually relatively inexpensive due to the small number of features provided as input to this layer after passing through several layers of pooling. When performing supervised training with gradient descent, every gradient step requires a complete run of forward propagation and backward propagation through the entire network. One way to reduce the cost of convolutional network training is to use features that are not trained in a supervised fashion.

There are three basic strategies for obtaining convolution kernels without supervised training.

1. One is to simply initialize them randomly.
2. Another is to design them by hand, for example by setting each kernel to detect edges at a certain orientation or scale.
3. Finally, one can learn the kernels with an unsupervised criterion.

For example: apply k-means clustering to small image patches, then use each learned centroid as a convolution kernel. Part 3 describes many more unsupervised learning approaches. Learning the features with an unsupervised criterion allows them to be determined separately from the classifier layer at the top of the architecture. One can then extract the features for the entire training set just once, essentially constructing a new training set for the last layer. Learning the last layer is then typically a convex optimization problem, assuming the last layer is something like logistic regression or an SVM.

Random filters often work surprisingly well in convolutional networks showed that layers consisting of convolution following by pooling naturally become frequency selective and translation invariant when assigned random weights.