Convolutional Networks

B AJAY RAM
ASSISTANT PROFESSOR
CSE DEPARTMENT

Introduction

- Convolutional networks, also known as convolution neural networks or CNNs, are a specialized kind of neural network for processing data that has a known, grid-like topology.

- Examples include time-series data, which can be thought of as a 1D grid taking samples at regular time intervals, and image data, which can be thought of as a 2D grid of pixels. 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.

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, i.e., we can get a different reading from the laser sensor at any instant in time.
- 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 our example, w needs to be a valid probability density function, or the output is not a weighted average. Also, w needs to be o for all negative arguments, or it will look into the future, which is presumably beyond our capabilities. These limitations are particular to our example though. In general, convolution is defined for any functions for which the above integral is defined, and may be used for other purposes besides taking weighted averages.
- In convolutional network terminology, the first argument (in this example, the function x) to the convolution is often referred to as the input and the second argument (in this example, the function w) as the kernel. The output is sometimes referred to as the feature map.
- In our example, the idea of a laser sensor that can provide measurements at every instant in time is not realistic. Usually, when we work with data on a computer, time will be discretized, and our sensor will provide data at regular intervals.
- In our example, it might be more realistic to assume that our laser provides a measurement once per second. The time index t can then take on only integer values. If we now assume that x and w are defined only on integer t, we can define the discrete convolution:

$$s(t) = (x * w)(t) = \sum_{a=-\infty}^{\infty} x(a)w(t-a)$$

- In machine learning applications, the input is usually a multidimensional array of data and the kernel is usually a multidimensional array of parameters that are adapted by the learning algorithm.
- We will refer to these multidimensional arrays as tensors. Because each element of the input and kernel must be explicitly stored separately, we usually assume that these functions are zero everywhere but the finite set of points for which we store the values.
- This means that in practice we can implement the infinite summation as a summation over a finite number of array elements. Finally, we often use convolutions over more than one axis at a time. For example, if we use a two-dimensional image I as our input, we probably also want to use a two-dimensional kernel K:

$$S(i,j) = (I * K)(i,j) = \sum_m \sum_n I(m,n)K(i-m, j-n).$$

Convolution is commutative, meaning we can equivalently write:

$$S(i,j) = (K * I)(i,j) = \sum_m \sum_n I(i-m, j-n)K(m,n).$$

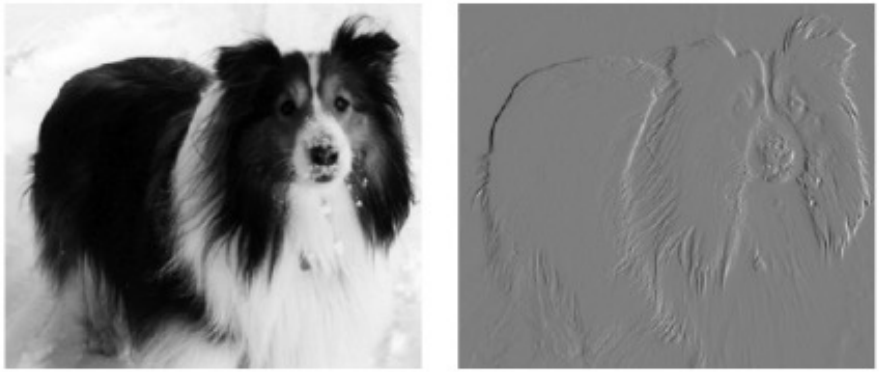
- The commutative property of convolution arises because we have flipped the kernel relative to the input, in the sense that as m increases, the index into the input increases, but the index into the kernel decreases.
- The only reason to flip the kernel is to obtain the commutative property. While the commutative property is useful for writing proofs, it is not usually an important property of a neural network implementation.
- Instead, many neural network libraries implement a related function called the cross-correlation, which is the same as convolution but without flipping the kernel:

$$S(i,j) = (I * K)(i,j) = \sum_m \sum_n I(i+m, j+n) K(m,n).$$

- Many machine learning libraries implement cross-correlation but call it convolution.
- Discrete convolution can be viewed as multiplication by a matrix. However, the matrix has several entries constrained to be equal to other entries. For example, for univariate discrete convolution, each row of the matrix is constrained to be equal to the row above shifted by one element. This is known as a Toeplitz matrix. In two dimensions, a doubly block circulant matrix corresponds to convolution.

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 activations. 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.
- 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 operation reports the maximum output within a rectangular neighborhood.
- Other popular pooling functions include the average of a rectangular neighborhood, the L2 norm of a rectangular neighborhood, or a weighted average based on the distance from the central pixel.



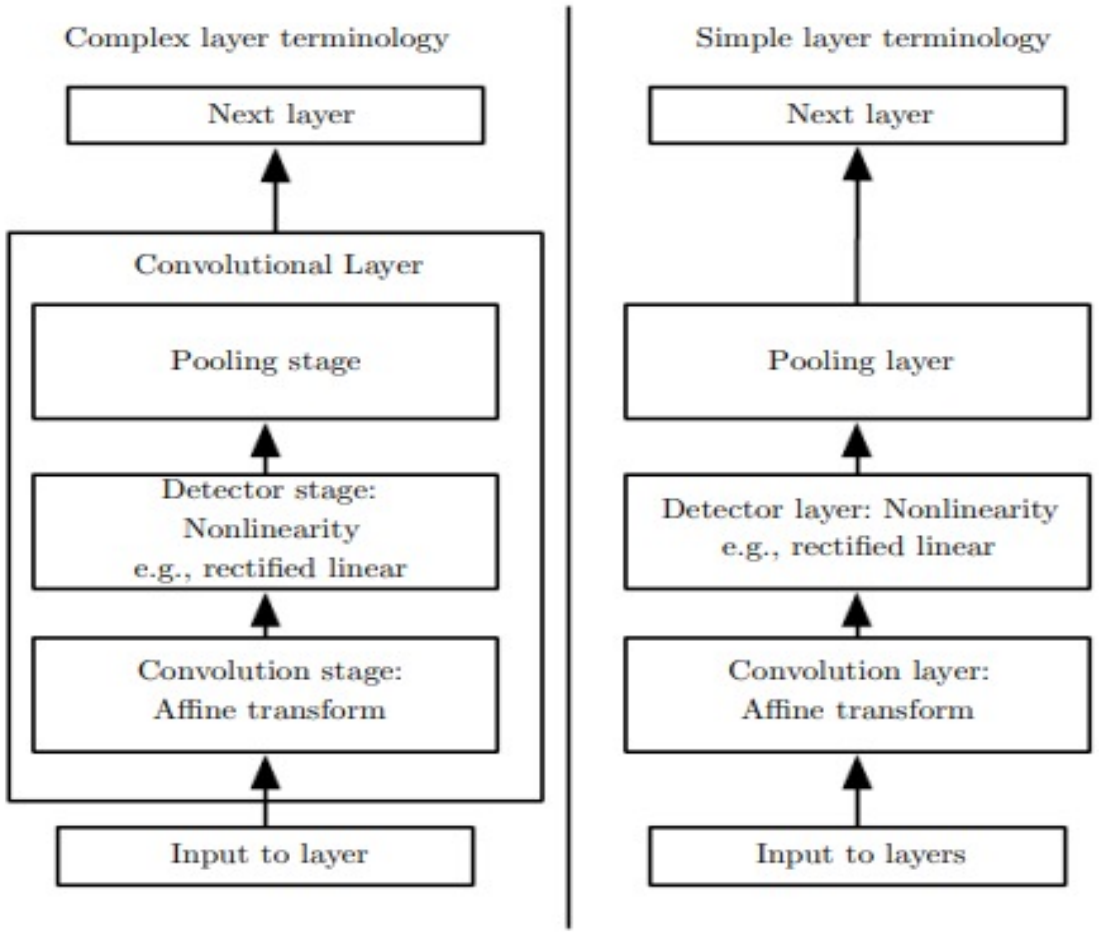


Figure 9.7: The components of a typical convolutional neural network layer. There are two commonly used sets of terminology for describing these layers. *(Left)*In this terminology, the convolutional net is viewed as a small number of relatively complex layers, with each layer having many "stages." In this terminology, there is a one-to-one mapping between kernel tensors and network layers. In this book we generally use this terminology. *(Right)*In this terminology, the convolutional net is viewed as a larger number of simple layers; every step of processing is regarded as a layer in its own right. This means that not every "layer" has parameters.

- 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. In other contexts, it is more important to preserve the location of a feature.
- For example, if we want to find a corner defined by two edges meeting at a specific orientation, we need to preserve the location of the edges well enough to test whether they meet.
- 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.
- Pooling over spatial regions produces invariance to translation, but if we pool over the outputs of separately parameterized convolutions, the features can learn which transformations to become invariant.
- Because pooling summarizes the responses over a whole neighborhood, it is possible to use fewer pooling units than detector units, by reporting summary statistics for pooling regions spaced k pixels apart rather than 1 pixel apart.

- For many tasks, pooling is essential for handling inputs of varying size. For example, if we want to classify images of variable size, the input to the classification layer must have a fixed size. This is usually accomplished by varying the size of an offset between pooling regions so that the classification layer always receives the same number of summary statistics regardless of the input size.
- For example, the final pooling layer of the network may be defined to output four sets of summary statistics, one for each quadrant of an image, regardless of the image size.

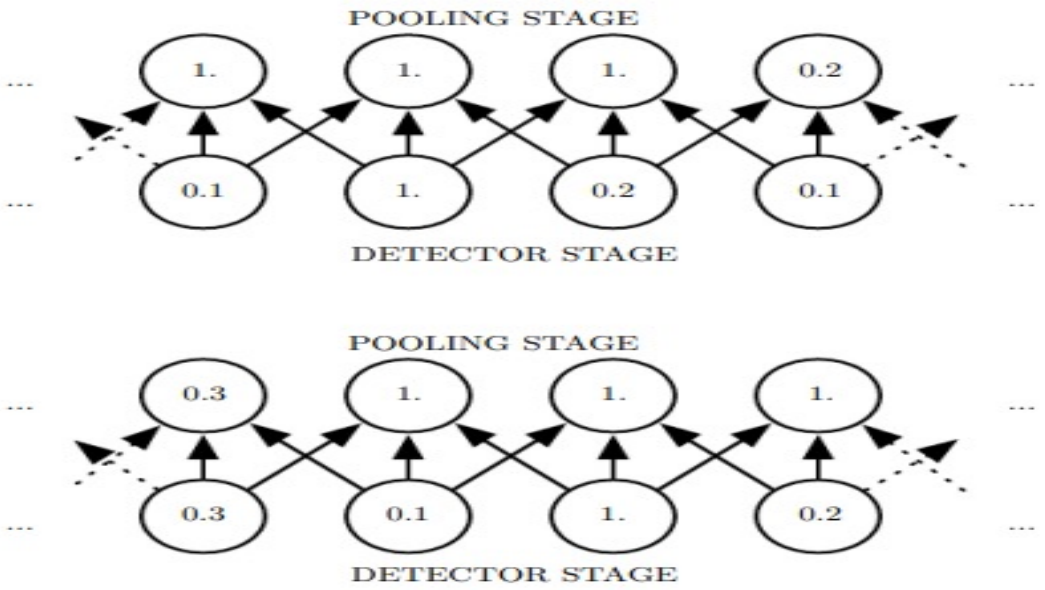


Figure 9.8: Max pooling introduces invariance. *(Top)*A view of the middle of the output of a convolutional layer. The bottom row shows outputs of the nonlinearity. The top row shows the outputs of max pooling, with a stride of one pixel between pooling regions and a pooling region width of three pixels. *(Bottom)*A view of the same network, after the input has been shifted to the right by one pixel. Every value in the bottom row has changed, but only half of the values in the top row have changed, because the max pooling units are only sensitive to the maximum value in the neighborhood, not its exact location.

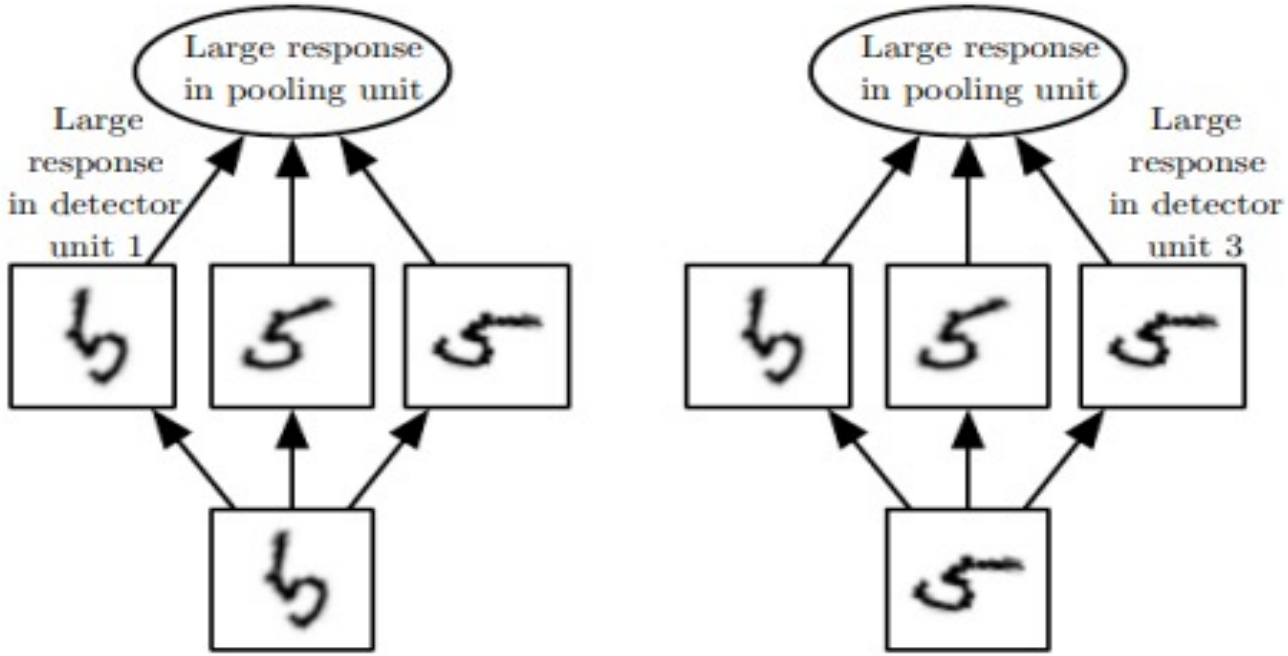


Figure 9.9: *Example of learned invariances:* A pooling unit that pools over multiple features that are learned with separate parameters can learn to be invariant to transformations of the input. Here we show how a set of three learned filters and a max pooling unit can learn to become invariant to rotation. All three filters are intended to detect a hand-written 5. Each filter attempts to match a slightly different orientation of the 5. When a 5 appears in the input, the corresponding filter will match it and cause a large activation in a detector unit. The max pooling unit then has a large activation regardless of which detector unit was activated. We show here how the network processes two different inputs, resulting in two different detector units being activated. The effect on the pooling unit is roughly the same either way. This principle is leveraged by maxout networks (Goodfellow *et al.*, 2013a) and other convolutional networks. Max pooling over spatial positions is naturally invariant to translation; this multi-channel approach is only necessary for learning other transformations.

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.
- We can imagine a convolutional net as being similar to a fully connected net, but with an infinitely strong prior over its weights. 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. This prior says that the function the layer should learn contains only local interactions and is equivariant to translation. Likewise, the use of pooling is an infinitely strong prior that each unit should be invariant to small translations.
- 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.
- One key insight is that convolution and pooling can cause underfitting. Like any prior, convolution and pooling are only useful when the assumptions made by the prior are reasonably accurate. If a task relies on preserving precise spatial information, then using pooling on all features can increase the training error.
- Other key insight from this view is that we should only compare convolutional models to other convolutional models in benchmarks of statistical learning performance. Models that do not use convolution would be able to learn even if we permuted all of the pixels in the image. For many image datasets, there are separate benchmarks for models that are permutation invariant and must discover the concept of topology via learning, and models that have the knowledge of spatial relationships hard-coded into them by their designer.

Basic Convolution Function

- Convolution in the context of neural networks does not refer exactly to the standard convolution operation in mathematics
- The functions used differ slightly
- Here we describe the differences in detail and highlight their useful propertied.

Convolution Operation in Neural Networks

1. It refers to an operation that consists of many applications of convolution in parallel
   - This is because convolution with a single kernel can only extract one kind of feature, albeit at many locations
   - Usually we want to extract many kinds of features at many locations
2. Input is usually not a grid of real values

   Rather it is a vector of observations
   - E.g., a color image has R ,G, B values at each pixel

   Input to the next layer is the output of the first layer which has many different convolutions at each position
   - When working with images, input and output are 3-D tensors

Four indices with image software

1. One index for Channel
2. Two indices for spatial coordinates of each channel
3. Fourth index for different samples in a batch
    - We omit the batch axis for simplicity of discussion

Multichannel Convolution

- Because we are dealing with multichannel convolution, linear operations are not usually commutative, even of kernel flipping is used
- These multi-channel operations are only commutative if each operation has the same number of output channels as input channels.

Definition of 4-D kernel tensor

- Assume we have a 4-D kernel tensor K with element Ki,j,k,l giving the connection strength between
    - a unit in channel i of the output and
    - a unit in channel j of the input,
    - with an offset of k rows and l columns between output and input units •

- Assume our input consists of observed data V with element Vi,j,k giving the value of the input unit
  - within channel i at row j and column k.
  - Assume our output consists of Z with the same format as V.
  - If Z is produced by convolving K across V without flipping K, then

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

Convolution with a stride

We may want to skip over some positions in the kernel to reduce computational cost

- At the cost of not extracting fine features

We can think of this as down-sampling the output of the full convolution function

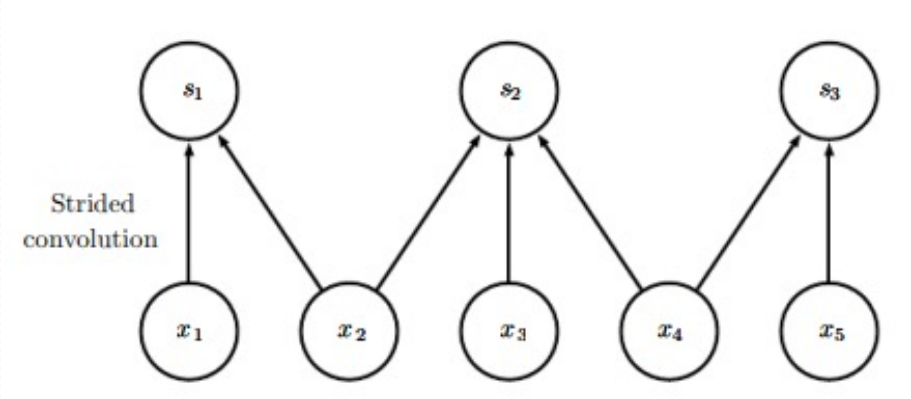
If we want to sample only every s pixels in each direction of output, then we can define a down-sampled convolution function c such that

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

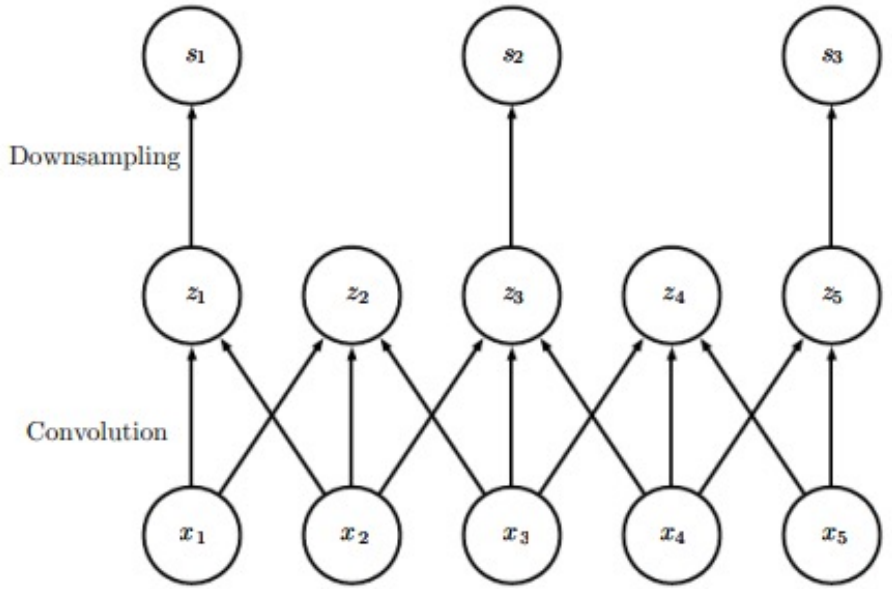
We refer to s as the stride. It is possible to define a different stride for each direction

Convolution with a stride: Implementation

- Convolution with a stride of length two implemented in a single operation.

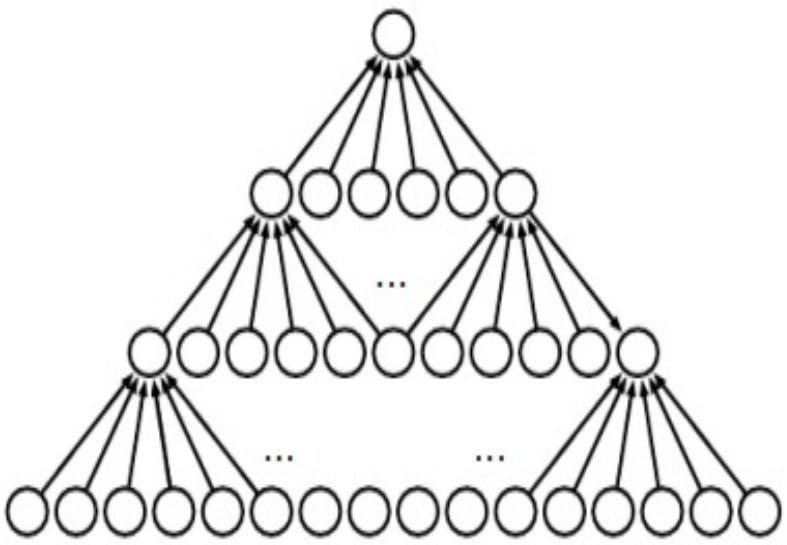


- Convolution with a stride greater than one pixel is mathematically equivalent to convolution with a unit stride followed by down-sampling.
- Two-step approach is computationally wasteful, because it discard many values that are discarded.



Effect of Zero-padding on network size

- Convolutional net with a kernel of width 6 at every layer .No pooling, so only convolution shrinks network size
- We do not use any implicit zero padding Causes representation to shrink by five pixels at each layer Starting from an input of 16 pixels we are only able to have 3 convolutional layers and the last layer does not ever move the kernel, so only two layers are convolutional
- By adding 5 implicit zeroes to Each layer, we prevent the Representation from shrinking with depth This allows us to make an arbitrarily deep convolutional network.



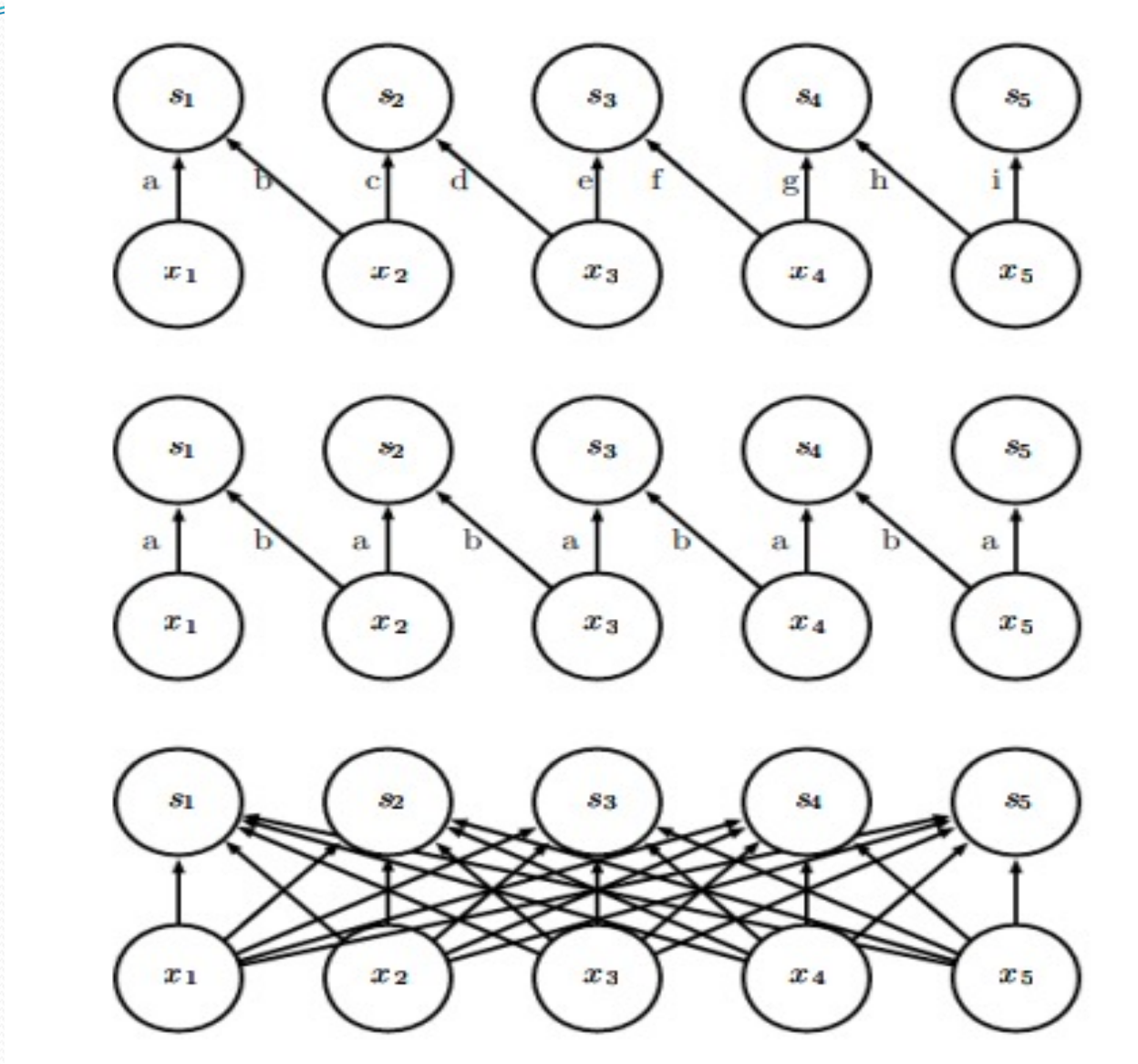
Locally connected layer

- In some cases, we do not actually want to use convolution, but rather locally connected layers
    - adjacency matrix in the graph of our MLP is the same, but every connection has its own weight, specified by a 6-D tensor W.
    - The indices into W are respectively:
    - i, the output channel,
    - j, the output row,
    - k, the output column,
    - l, the input channel,
    - m, the row offset within the input, and
    - n, the column offset within the input.
- The linear part of a locally connected layer is then given by

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

- This is sometimes also called unshared convolution, because it is a similar operation to discrete convolution with a small kernel, but without sharing parameters across locations.

Local connections, convolution, full connections



Use of locally connected layers

- Locally connected layers are useful when
  - we know that each feature should be a function of a small part of space, but there is no reason to think that the same feature should occur across all of space
  - Ex: if we want to tell if an image is a picture of a face, we only need to look for the mouth in the bottom half of the image.

Constraining Outputs

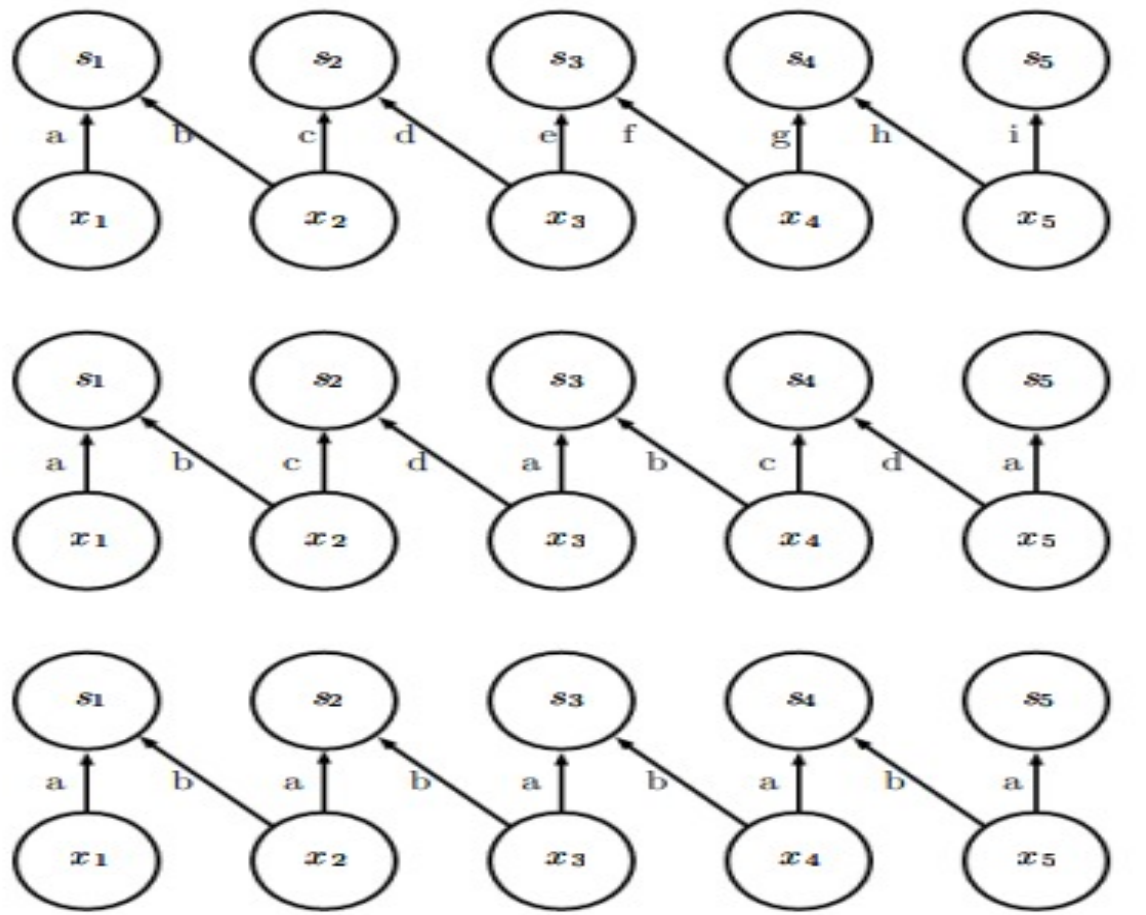
- Constrain each output channel i to be a function of only a subset of the input channels l
  - Make the first m output channels connect to only the first n input channels,
  - The second m output channels connect to only the second n input channels, etc
- Modeling interactions between few channels allows fewer parameters to:
  - Reduce memory, increase statistical efficiency, reduce computation for forward/back-propagation.
  - It accomplishes these goals without reducing no. of hidden units.

Tiled Convolution

- Compromise between a convolutional layer and a locally connected layer.
  - Rather than learning a separate set of weights at every spatial location, we learn a set of kernels that we rotate through as we move through space.
- This means that immediately neighboring locations will have different filters, like in a locally connected layer,
  - but the memory requirements for storing the parameters will increase only by a factor of the size of this set of kernels
  - rather than the size of the entire output feature map.

Comparison of locally connected layers, tiled convolution and standard convolution

- A locally connected layer Has no sharing at all Each connection has its own weight.
- Tiled convolution Has a set of different kernels With t=2
- Traditional convolution Equivalent to tiled convolution with t=1 There is only one kernel and it is applied everywhere



Both locally connected layers and tiled convolutional layers have an interesting interaction with max-pooling: the detector units of these layers are driven by different filters.

Other operations besides convolution are usually necessary to implement a convolutional network. To perform learning, one must be able to compute the gradient with respect to the kernel, given the gradient with respect to the outputs.

Defining Tiled Convolution Algebraically

- Let k be a 6-D tensor, where two of the dimensions correspond to different locations in the output map.
- Rather than having a separate index for each location in the output map, output locations cycle through a set of t different choices of kernel stack in each direction.
- If t is equal to the output width, this is the same as a locally connected layer.

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

- • where % is the modulo operation, with t%t = 0, (t + 1)%t = 1, etc.

Operations to implement convolutional nets

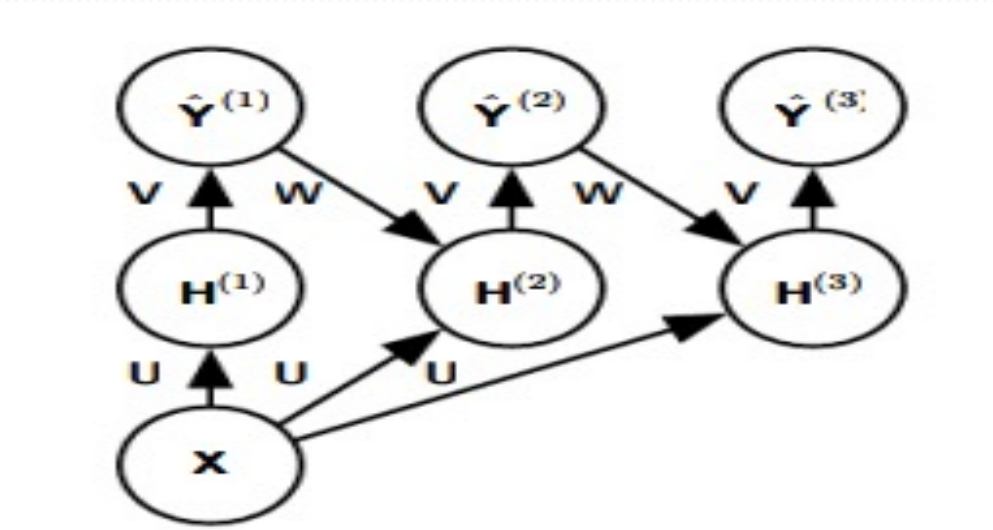
- Besides convolution, other operations are necessary to implement a convolution network.
- To perform learning, need to compute gradient wrt the kernel, given the gradient with respect to the outputs.
- In some simple cases, this operation can be performed using the convolution operation, but when stride greater than 1, do not have this property.

Implementation of Convolution

- Convolution is a linear operation and can thus be described as a matrix multiplication
    - if we first reshape the input tensor into a flat vector
- Matrix involved is a function of the convolution kernel
    - Matrix is sparse and each element of the kernel is copied to several elements of the matrix.
- This view helps us to derive some of the other operations needed to implement a convolution network

Structured Outputs

- Convolutional networks can be used to output a high-dimensional, structured object, rather than just predicting a class label for a classification task or a real value for a regression task.
- Typically this object is just a tensor, emitted by a standard convolutional layer.
- **For example**, the model might emit a tensor S, where Si,j,k is the probability that pixel (j, k) of the input to the network belongs to class i. This allows the model to label every pixel in an image and draw precise masks that follow the outlines of individual objects.



- One issue that often comes up is that the output plane can be smaller than the input plane, as shown. In the kinds of architectures typically used for classification of a single object in an image, the greatest reduction in the spatial dimensions of the network comes from using pooling layers with large stride.
- In order to produce an output map of similar size as the input, one can avoid pooling altogether. Another strategy is to simply emit a lower-resolution grid of labels.
- One strategy for pixel-wise labeling of images is to produce an initial guess of the image labels, then refine this initial guess using the interactions between neighboring pixels.
- Repeating this refinement step several times corresponds to using the same convolutions at each stage, sharing weights between the last layers of the deep net. This makes the sequence of computations performed by the successive convolutional layers with weights shared across layers a particular kind of recurrent network.
- The general idea is to assume that large groups of contiguous pixels tend to be associated with the same label. Graphical models can describe the probabilistic relationships between neighboring pixels.
- Alternatively, the convolutional network can be trained to maximize an approximation of the graphical model training objective

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.
- One advantage to convolutional networks is that they can also process inputs with varying spatial extents. These kinds of input simply cannot be represented by traditional, matrix multiplication-based neural networks.
- This provides a compelling reason to use convolutional networks even when computational cost and overfitting are not significant issues.
- For example, consider a collection of images, where each image has a different width and height. It is unclear how to model such inputs with a weight matrix of fixed size.
- Convolution is straightforward to apply; the kernel is simply applied a different number of times depending on the size of the input, and the output of the convolution operation scales accordingly.
- Convolution may be viewed as matrix multiplication; the same convolution kernel induces a different size of doubly block circulant matrix for each size of input.

- In other cases, the network must produce some fixed-size output, for example if we want to assign a single class label to the entire image.
- In this case we must make some additional design steps, like inserting a pooling layer whose pooling regions scale in size proportional to the size of the input, in order to maintain a fixed number of pooled outputs.
- 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.

	Single channel	Multi-channel
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 2D 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 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.

Efficient Convolution Algorithms

- Modern convolutional network applications often involve networks containing more than one million units. Powerful implementations exploiting parallel computation resources are essential. 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 the kernel is separable, naive convolution is inWhen a d-dimensional kernel can be expressed as the outer product of d vectors, one vector per dimension, the kernel is called separable. efficient. It is equivalent to compose d one-dimensional convolutions with each of these vectors.
- Devising faster ways of performing convolution or approximate convolution without harming the accuracy of the model is an active area of research. Even techniques that improve the efficiency of only forward propagation are useful because in the commercial setting, it is typical to devote more resources to deployment of a network than to its training.

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. One is to simply initialize them randomly. Another is to design them by hand, for example by setting each kernel to detect edges at a certain orientation or scale. Finally, one can learn the kernels with an unsupervised criterion.
- 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.
- They argue that this provides an inexpensive way to choose the architecture of a convolutional network: first evaluate the performance of several convolutional network architectures by training only the last layer, then take the best of these architectures and train the entire architecture using a more expensive approach.
- An intermediate approach is to learn the features, but using methods that do not require full forward and back-propagation at every gradient step. As with multilayer perceptrons, we use greedy layer-wise pretraining, to train the first layer in isolation, then extract all features from the first layer only once, then train the second layer in isolation given those features, and so on.
- Convolutional networks offer us the opportunity to take the pretraining strategy one step further than is possible with multilayer perceptrons. Instead of training an entire convolutional layer at a time, we can train a model of a small patch as that of with k-means.
- This means that it is possible to use unsupervised learning to train a convolutional network without ever using convolution during the training process. Using this approach, we can train very large models and incur a high computational cost only at inference time.
- Unsupervised pretraining may offer some regularization relative to supervised training, or it may simply allow us to train much larger architectures due to the reduced computational cost of the learning rule