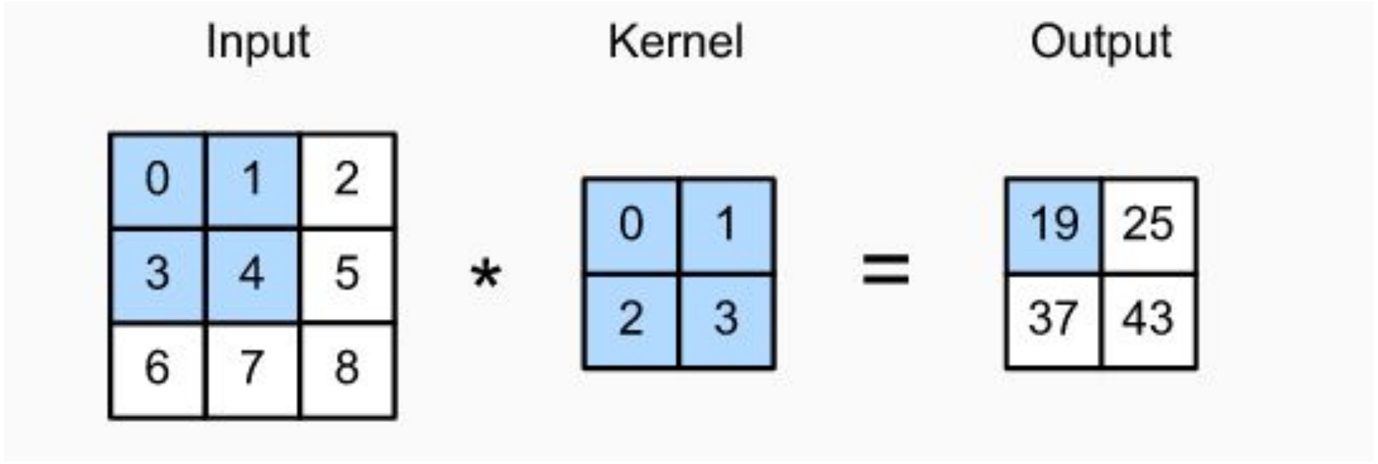
Convolutional Neural Networks

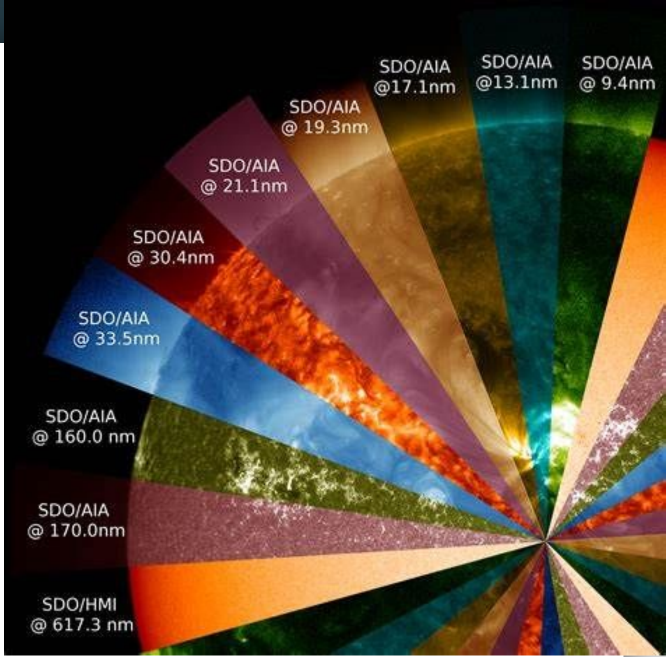
Sections 6.4-6.6

Review

- In the last lecture, convolutions with one input, one kernel, and one output were used similar to the figure below
- This was useful because it could allow us to identify certain aspects of the image, but it was also very limiting because of the use of only one input, kernel, and output
    - We are going to go into more depth on how this use can be extended today



Multiple Input Channels

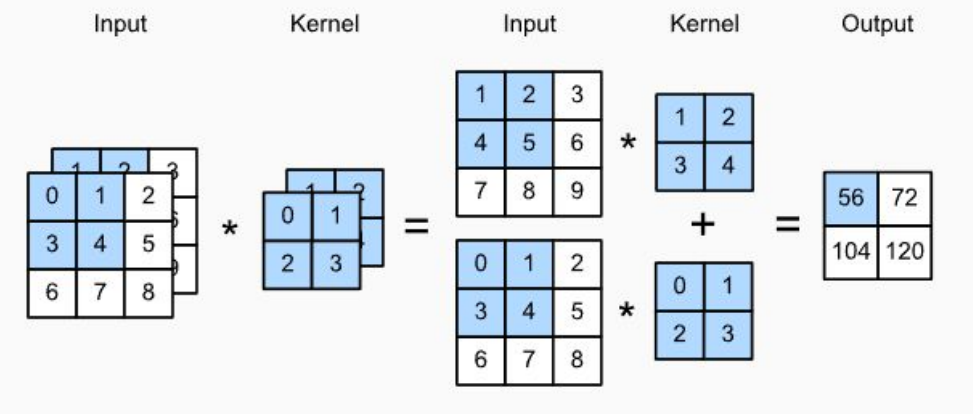


Why would we want this?

- It allows us to cross correlate the input data across the multiple channels/inputs
  - For example, for a person to recognize emotion we take visual, audio, text input to put together how someone is feeling, similarly the CNN can use the multiple inputs to build a better recognition system for the inputs.

Multiple Input Channels continued

- Mathematical Definitions:
  - c_i=number of channels for input data
  - k_h,k_w=height, width of kernel
  - k_h*k_w=kernels window shape
  - c_i*k_h*k_w= convolutional kernel of this shape



$$\begin{bmatrix}0&1&2\\3&4&5\\6&7&8\end{bmatrix}, \begin{bmatrix}1&2&3\\4&5&6\\7&8&9\end{bmatrix} * \begin{bmatrix}0&1\\2&3\end{bmatrix}, \begin{bmatrix}1&2\\3&4\end{bmatrix} = \begin{bmatrix}1&2&3\\4&5&6\\7&8&9\end{bmatrix} * \begin{bmatrix}1&2\\3&4\end{bmatrix} + \begin{bmatrix}0&1&2\\3&4&5\\6&7&8\end{bmatrix} * \begin{bmatrix}0&1\\2&3\end{bmatrix} = \begin{bmatrix}56&72\\104&120\end{bmatrix}$$

Fig. 6.4.1 Cross-correlation computation with 2 input channels.

Multiple Input Channels Continued

We can perform a cross correlation operation on the 2-D tensor of the input and 2-D tensor of the convolutional kernel for each channel, then sum the channels to get a 2-D tensor. This is similar to when there was one channel, we are simply summing them together now.

We can apply different filters to different inputs before cross correlation this way.

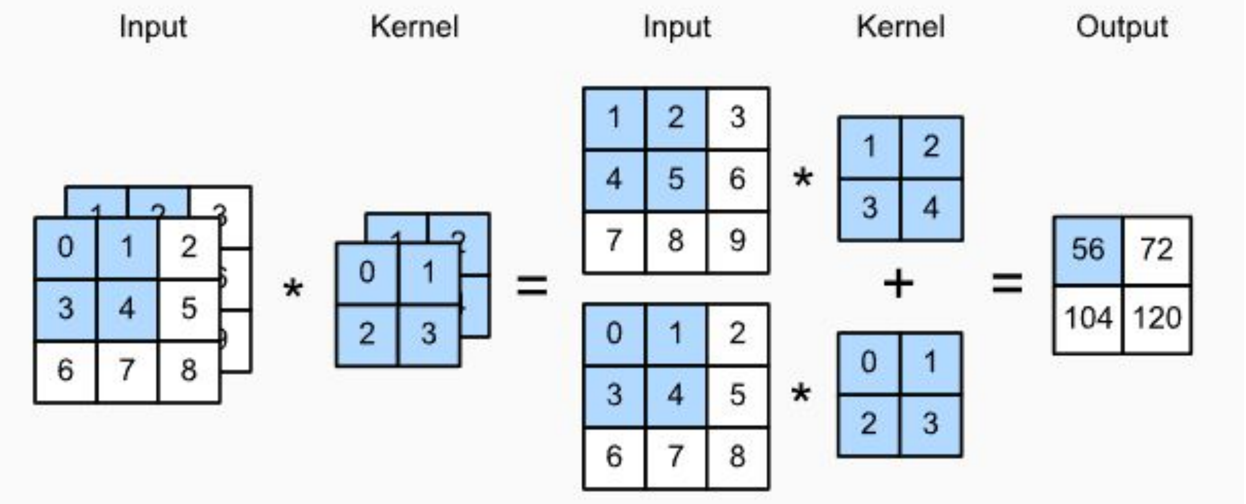
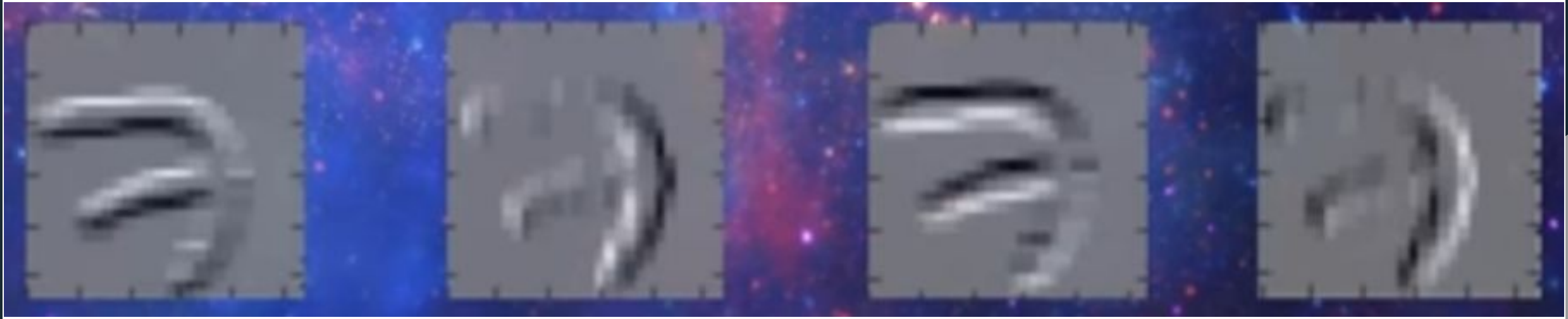


Fig. 6.4.1 Cross-correlation computation with 2 input channels.

$$(1\times1+2\times2+4\times3+5\times4)+(0\times0+1\times1+3\times2+4\times3)=56.$$

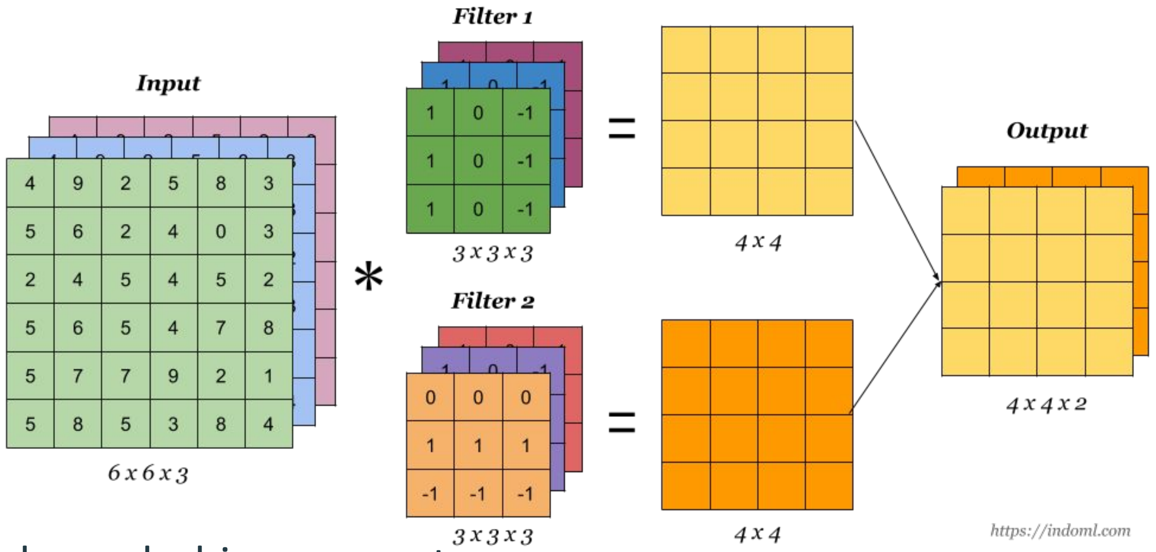
Multiple Output Channels



Why would we want this?

- It allows us to use multiple kernels on one (or more) inputs to generate outputs based on each kernel
  - Kernels are essentially filters that are recognizing one specific aspect of the image
- This allows us to capture multiple aspects of an image at each spatial location

Multiple Output Channels Continued

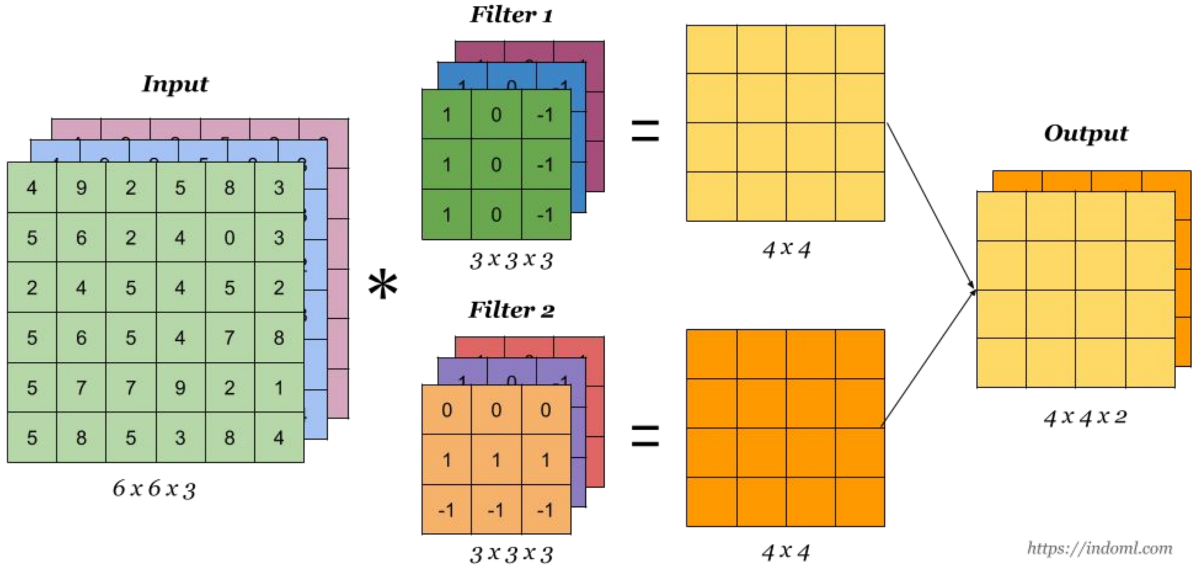


Mathematical Definitions:

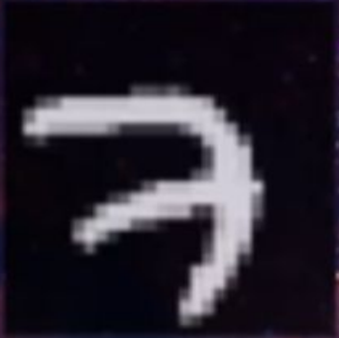
- c_o=number of output channels, bias parameters
- To create output of multiple channels we start with $c_i*k_h*k_w$ for every output channel.
- $c_o*c_i*k_h*k_w$= convolutional kernel, number of weight parameters

Multiple Output Channels Continued

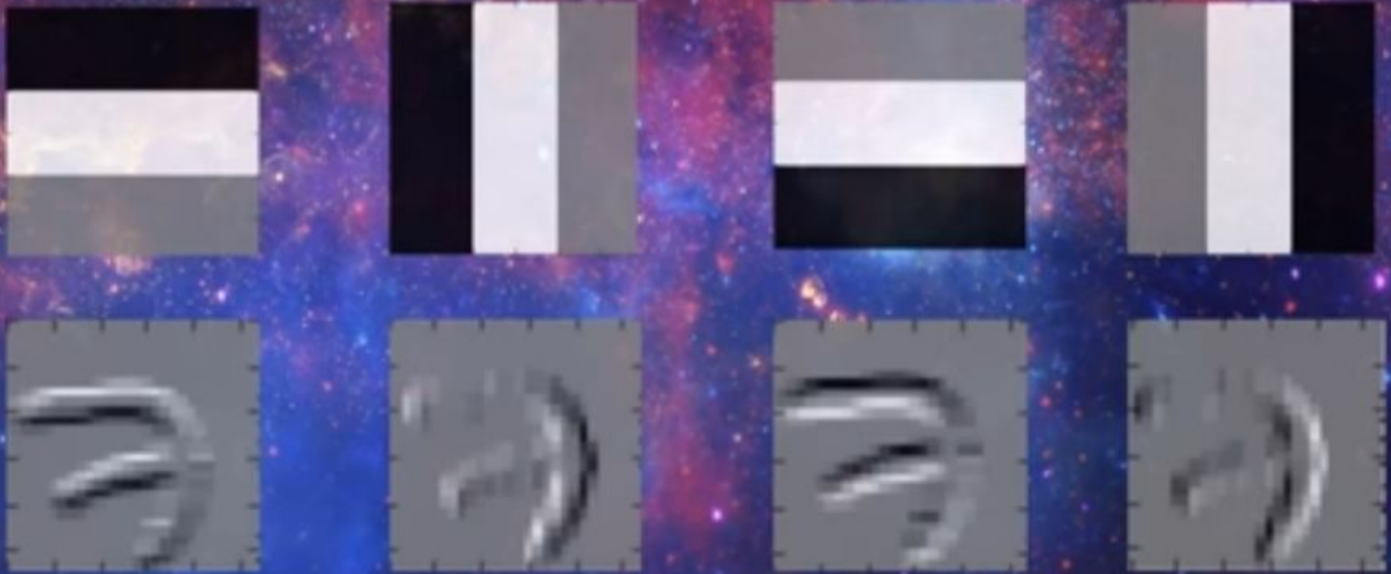
- For cross correlation operations, the result of each output channel is calculated from the convolutional kernel corresponding to that output channel, that takes input from all channels in input tensor.



- Basically rather than having multiple input tensors, we now have multiple kernels to create multiple output channels from one (or more) input channel, allowing us to look at different aspects of the input



filter 1	filter 2	filter 3	filter 4
$\begin{matrix} -1 & -1 & -1 \\ 1 & 1 & 1 \\ 0 & 0 & 0 \end{matrix}$	$\begin{matrix} -1 & 1 & 0 \\ -1 & 1 & 0 \\ -1 & 1 & 0 \end{matrix}$	$\begin{matrix} 0 & 0 & 0 \\ 1 & 1 & 1 \\ -1 & -1 & -1 \end{matrix}$	$\begin{matrix} 0 & 1 & -1 \\ 0 & 1 & -1 \\ 0 & 1 & -1 \end{matrix}$



1x1 Convolutional Layer

- 1x1 convolution (k_h=k_w=1) are sometimes included in complex deep networks
- This method can no longer recognize patterns among adjacent elements, only on the channel dimension will it recognize these
- This is equivalent to the fully-connected layer for per pixel bias
  - Summarizes the info across inputs for that location
- Requires c_o*c_i weights (plus the bias)

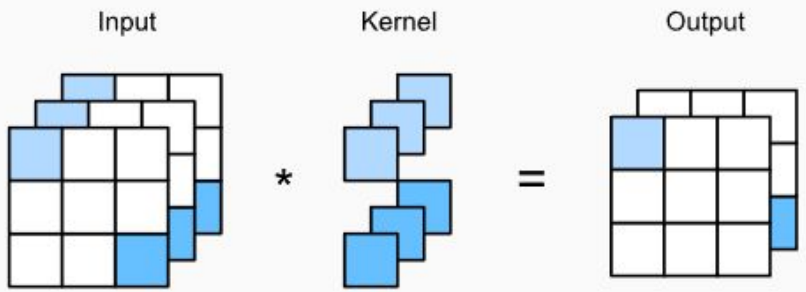


Fig. 6.4.2 The cross-correlation computation uses the 1×1 convolution kernel with 3 input channels and 2 output channels. The input and output have the same height and width.

1x1 Convolutional Layer

- Often used to adjust number of channels between network layers and control model complexity
- This does not change spatial dimensionality, only the number of channels, like pooling but for channels
- Each output comes from a linear combination of elements at the same position in the input image

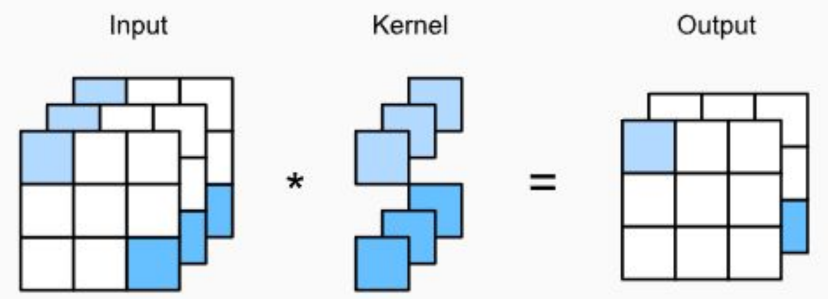


Fig. 6.4.2 The cross-correlation computation uses the 1×1 convolution kernel with 3 input channels and 2 output channels. The input and output have the same height and width.

Pooling

- Like convolutional layers except there is no kernel (parameters), but rather it is determining the maximum or average value within each group it slides over.
- Often labeled p*q pooling where p*q is the pooling window shape
- Method can alleviate the excessive sensitivity of the convolutional layer to location

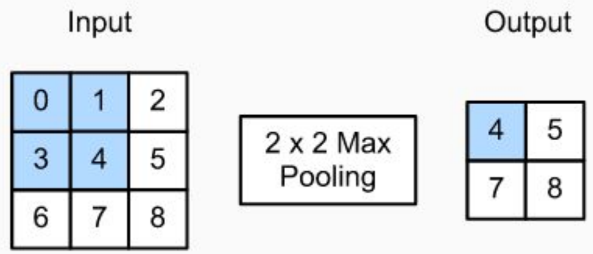


Fig. 6.5.1 Maximum pooling with a pooling window shape of 2×2. The shaded portions are the first output element as well as the input tensor elements used for the output computation: $\max(0, 1, 3, 4) = 4$.

Pooling Continued

- The pooling layer pools each input channel separately, rather than summing them, meaning the number of output channels is the same as the number of input channels
- Reduces the spatial dimensions just like convolutions

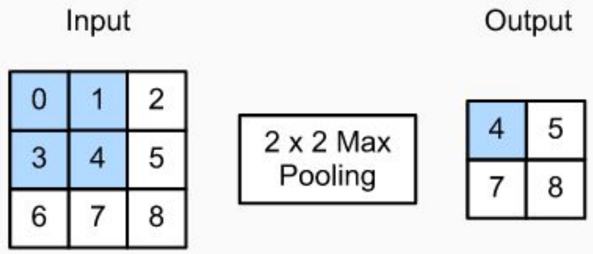
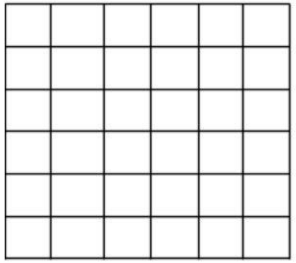
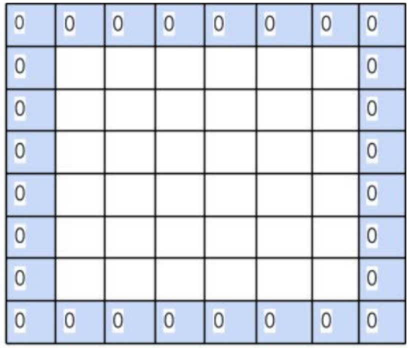


Fig. 6.5.1 Maximum pooling with a pooling window shape of 2×2. The shaded portions are the first output element as well as the input tensor elements used for the output computation: $\max(0, 1, 3, 4) = 4$.

Padding For Pooling

- Padding preserves the size of the original image, fixing the shrinking problem in CNN’s as well as making sure it hits the edges as often as the middle of the input
- It does this by essentially adding a border around the input similar to the figure below
- If an n*n input matrix convolves with an f*f filter matrix with padding p, output image will be (n+2p-f+1)*(n+2p-f+1) where p=1 in the below case.

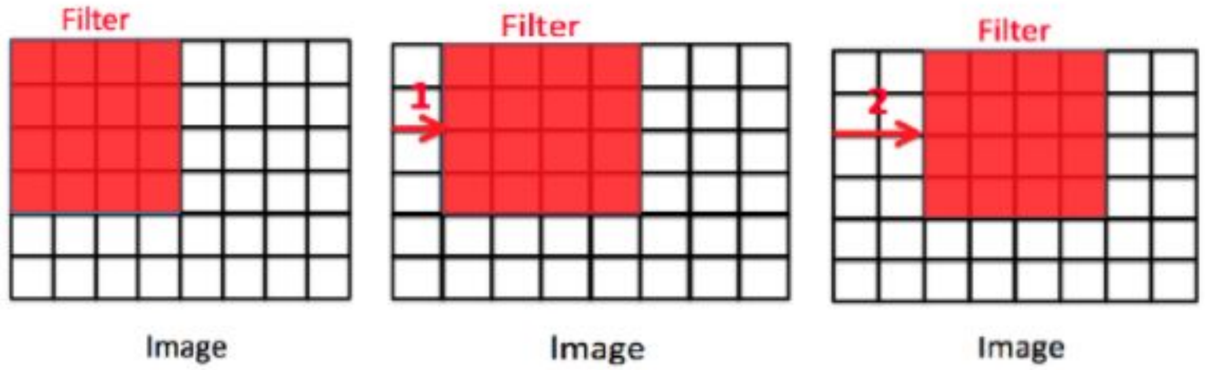

6x6 image



6x6 image with 1 layer of zero padding

Stride for Pooling

- Stride is the number of pixels the filter or pooling shifts over
- Padding p, input n*n, filter f*f, and stride s, the output image
- Will be [{(n+2p-f+1)/s}+1]*[{(n+2p-f+1)/s}+1]
- This is essentially the same as it was for convolutional layers



left image: stride =0, middle image: stride = 1, right image: stride =2

LeNet (Convolutional Neural Networks)

First successful implementation of a CNN

Used to identify handwritten digits

Basic units in each convolutional block:

- Convolutional layer (5x5 kernel)
- Sigmoid activation function
- Subsequent average pooling operation

Though ReLUs and max-pooling work better now

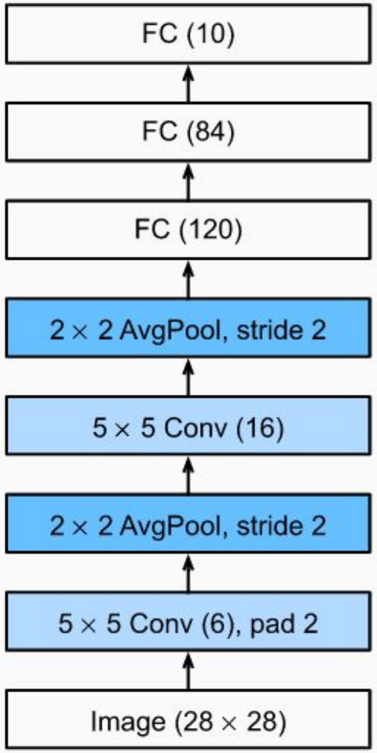


Fig. 6.6.2 Compressed notation for LeNet-5.

LeNet (Convolutional Neural Networks)

Consists of 2 parts:

- Convolutional encoder consisting of 2 convolutional layers
- A dense block consisting of 3 fully-connected layers

First layer has 2 pixels of padding to compensate for reduction in size, the number of channels increases throughout, while the pooling layer halves the height and width, leaving an output with dimensions equal to the number of classes

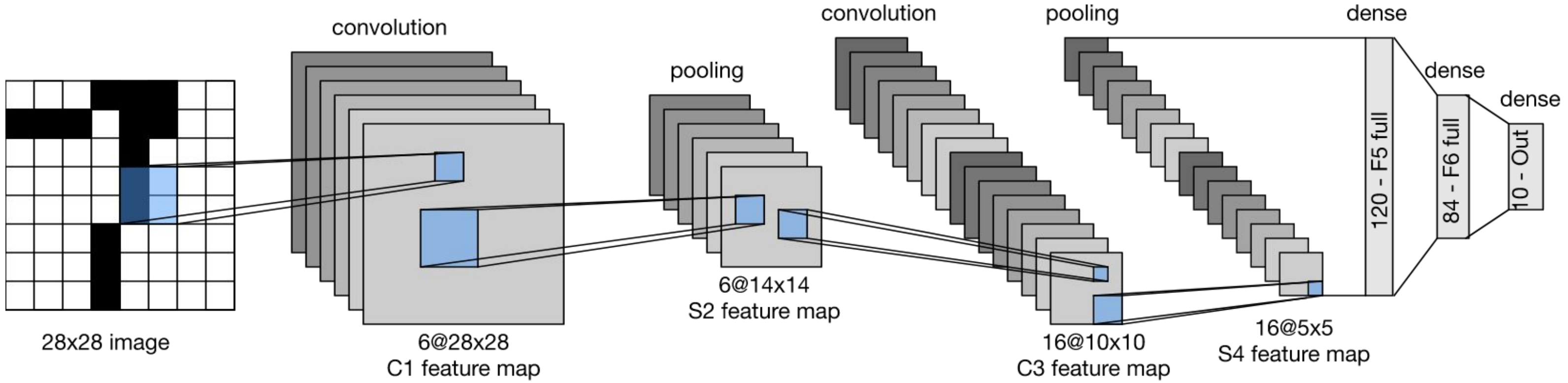
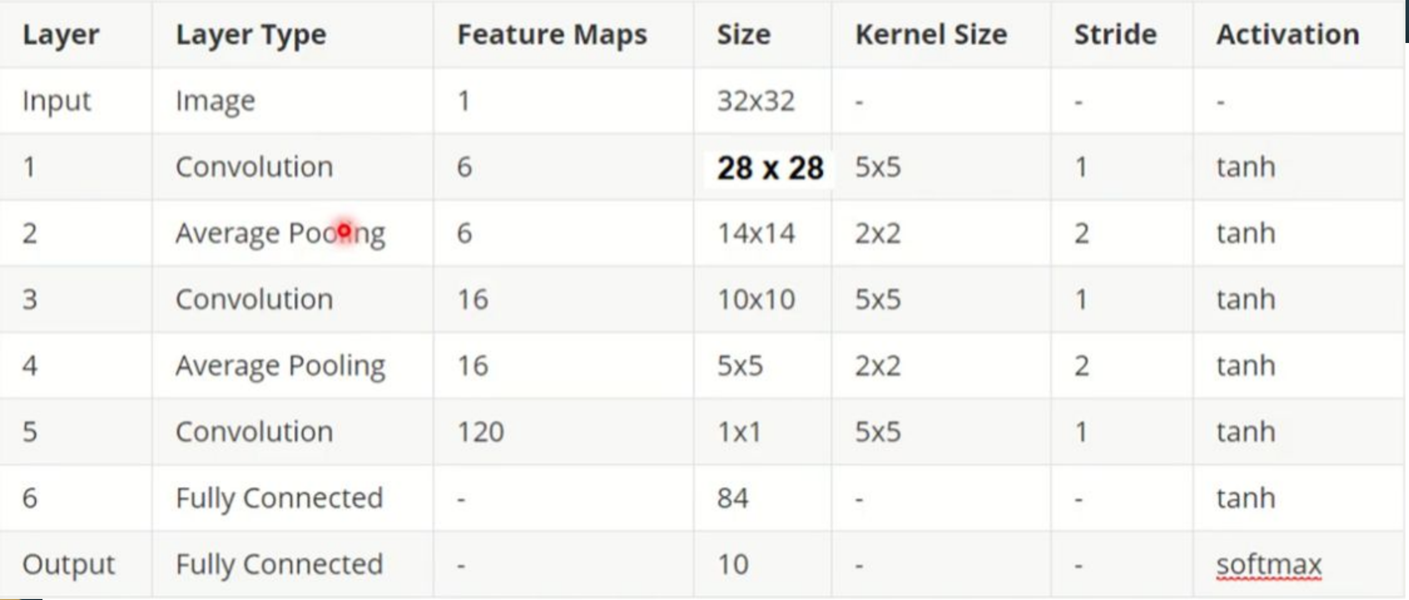
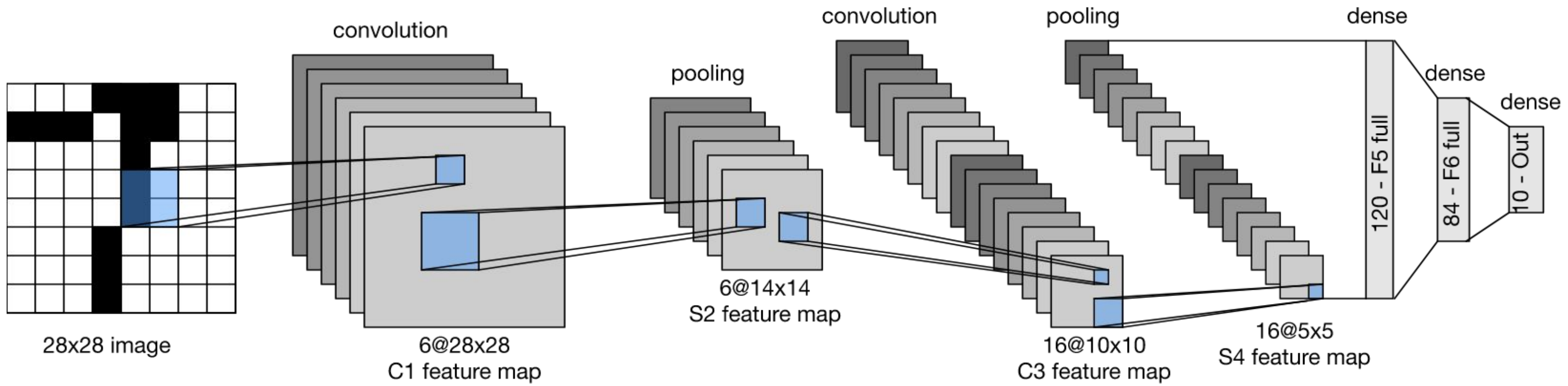


Fig. 6.6.1 Data flow in LeNet. The input is a handwritten digit, the output a probability over 10 possible outcomes.



Layer	Layer Type	Feature Maps	Size	Kernel Size	Stride	Activation
Input	Image	1	32x32	-	-	-
1	Convolution	6	**28 x 28**	5x5	1	tanh
2	Average Pooling	6	14x14	2x2	2	tanh
3	Convolution	16	10x10	5x5	1	tanh
4	Average Pooling	16	5x5	2x2	2	tanh
5	Convolution	120	1x1	5x5	1	tanh
6	Fully Connected	-	84	-	-	tanh
Output	Fully Connected	-	10	-	-	softmax



Training a CNN

- Training a CNN typically consists of a forward and backward phase
  - Forward phase caches data
  - Backward phase receives a gradient of loss with respect to output, and returns one with respect to the input
- CNN’s have fewer parameters than other machine learning programs, however they can be more expensive to run if you aren’t careful, because each parameter participates in many more multiplications

Summary

- Multiple channels
  - Can extend model parameters of the convolutional layer
  - 1x1 convolutional layers is the same as a fully-connected layer on a per pixel basis
  - 1x1 convolutional layer adjusts the number of channels between network layers and to control the model complexity
- Pooling
  - Maximum pooling assigns maximum value as the output, average pooling assign average
  - Pooling alleviates sensitivity of the convolutional layer to location
  - Padding and stride also apply to pooling layer
    - Can reduce spatial dimensions
  - Pooling layers number of output and input channels are the same
- Convolutional Neural Network
  - Employs convolutional layers
  - Interleave convolutions, nonlinearities, and often pooling operations
  - Usually decrease the spatial resolution while increasing the number of channels