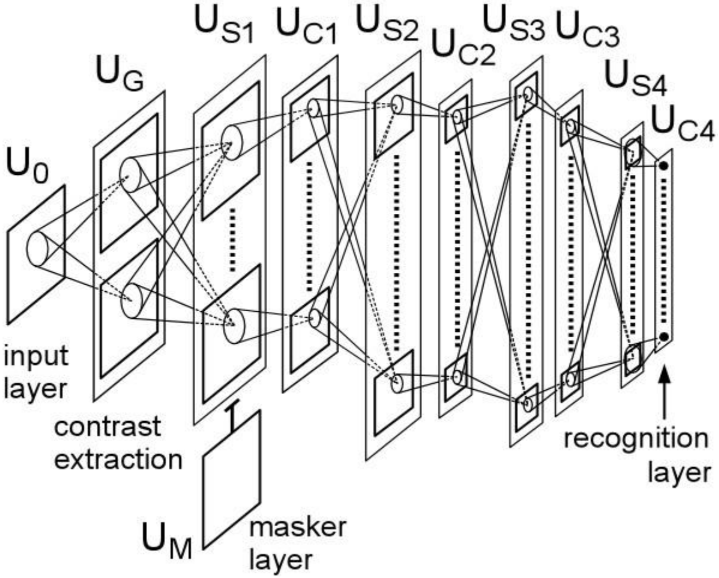
Convolutional Networks and Applications

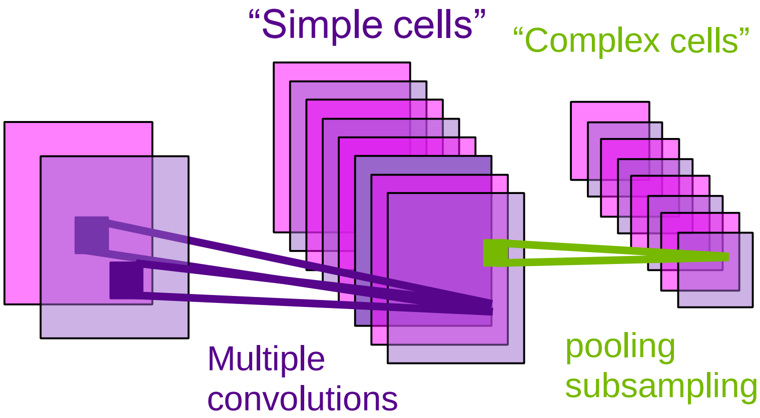
Early hierarchical feature models for vision

[Hubel & Wiesel 1962]:

– simple cells detect local features

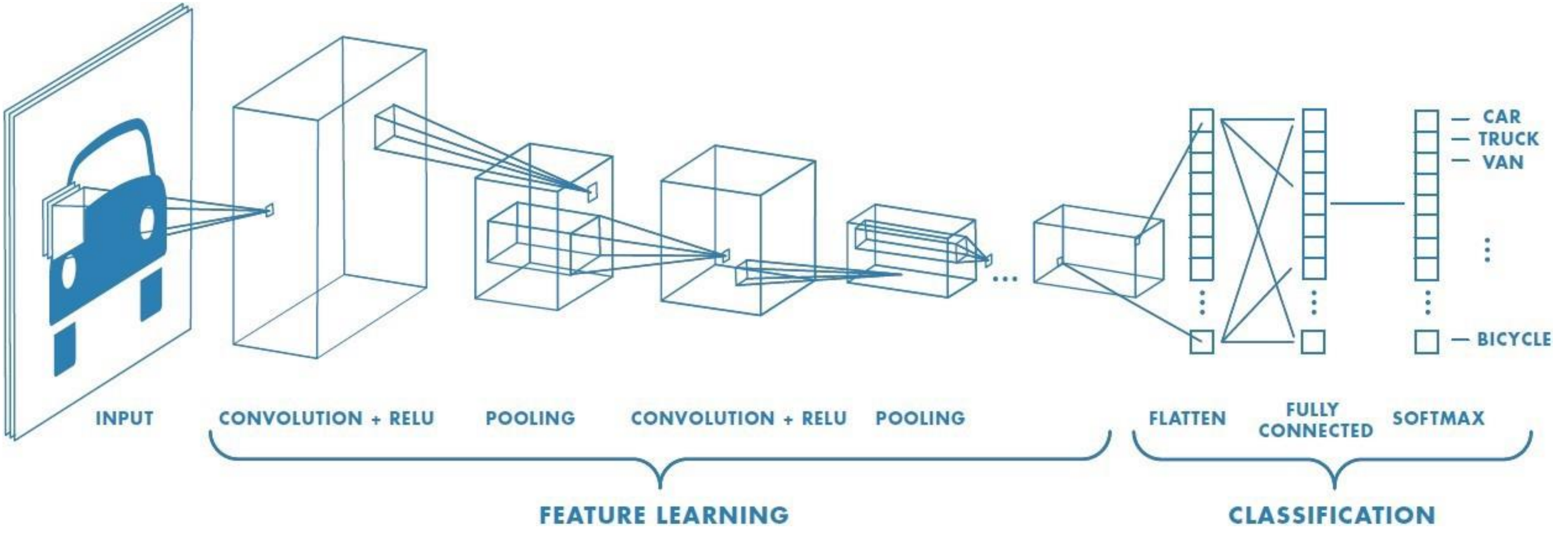
– complex cells “pool” the outputs of simple cells within a retinotopic neighborhood.





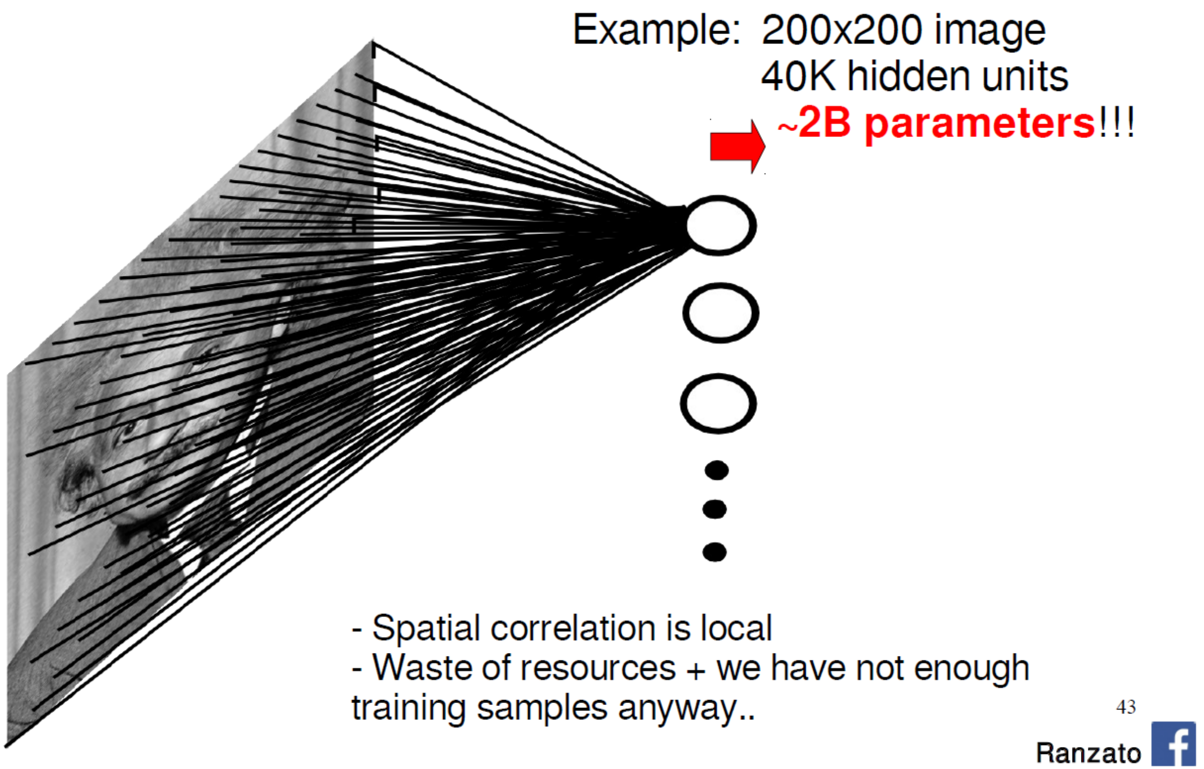
Cognitron & Neocognitron [Fukushima 1974-1982]

Convolutional Neural Networks for Classificaiton



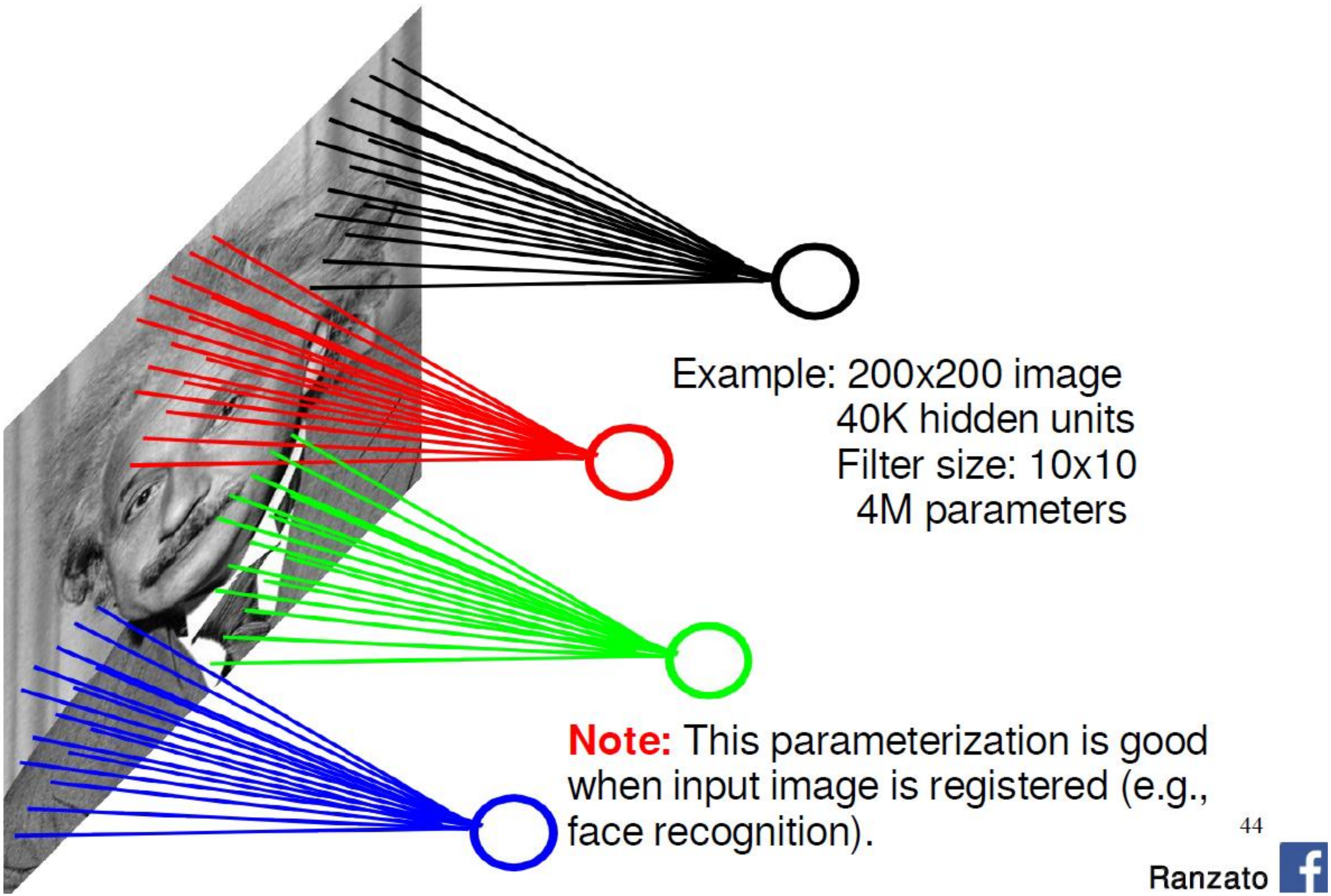
Fully Connected Layer

- Each neuron is connected to every neuron in the previous layer
- Each connection has it's own weight.



Source: Image Classification with Deep Learning, Marc'Aurelio Ranzato, Facebook A.I. Research
Stanford CS231A ww.cs.toronto.edu/~ranzato

Locally Connected Layer



Source: Image Classification with Deep Learning, Marc'Aurelio Ranzato, Facebook A.I. Research
Stanford CS231A ww.cs.toronto.edu/~ranzato

Locally Connected Layer

When Will this Work?

- This is good when the input is (roughly) registered

Source: https://www.cs.toronto.edu/~jlucas/teaching/csc411/lectures/lec11_handout.pdf

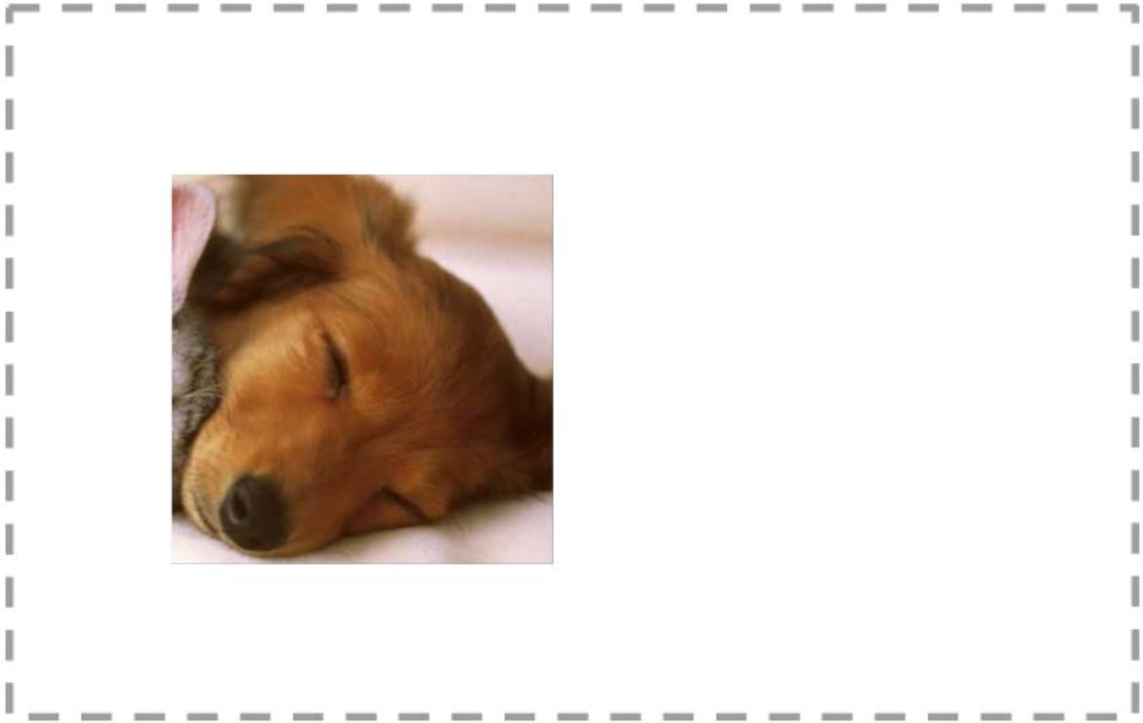
- The object can be anywhere

[Slide: Y. Zhu]

Source: https://www.cs.toronto.edu/~jlucas/teaching/csc411/lectures/lec11_handout.pdf

Locally Connected Layer

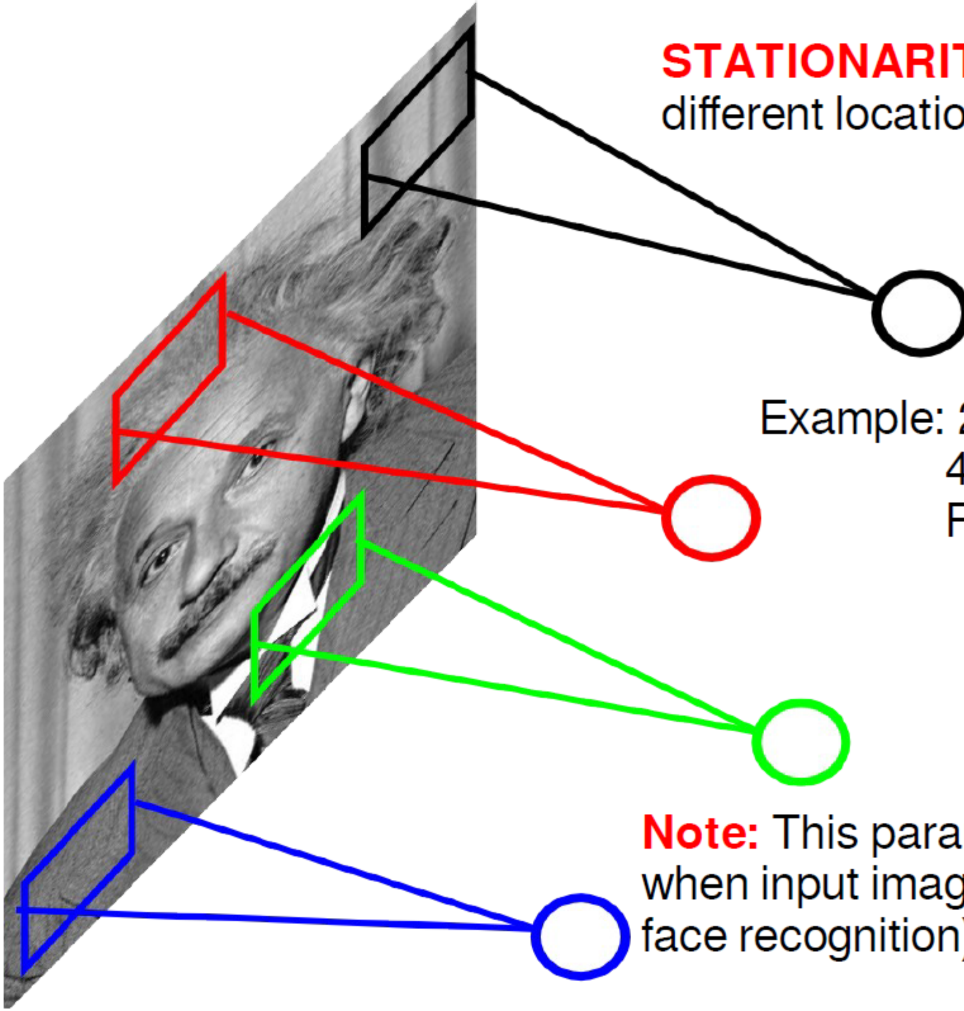
- The object can be anywhere



[Slide: Y. Zhu]

Source: https://www.cs.toronto.edu/~jlucas/teaching/csc411/lectures/lec11_handout.pdf

Locally Connected Layer



STATIONARITY? Statistics is similar at different locations

Example: 200x200 image
40K hidden units
Filter size: 10x10
4M parameters

Note: This parameterization is good when input image is registered (e.g., face recognition).

45

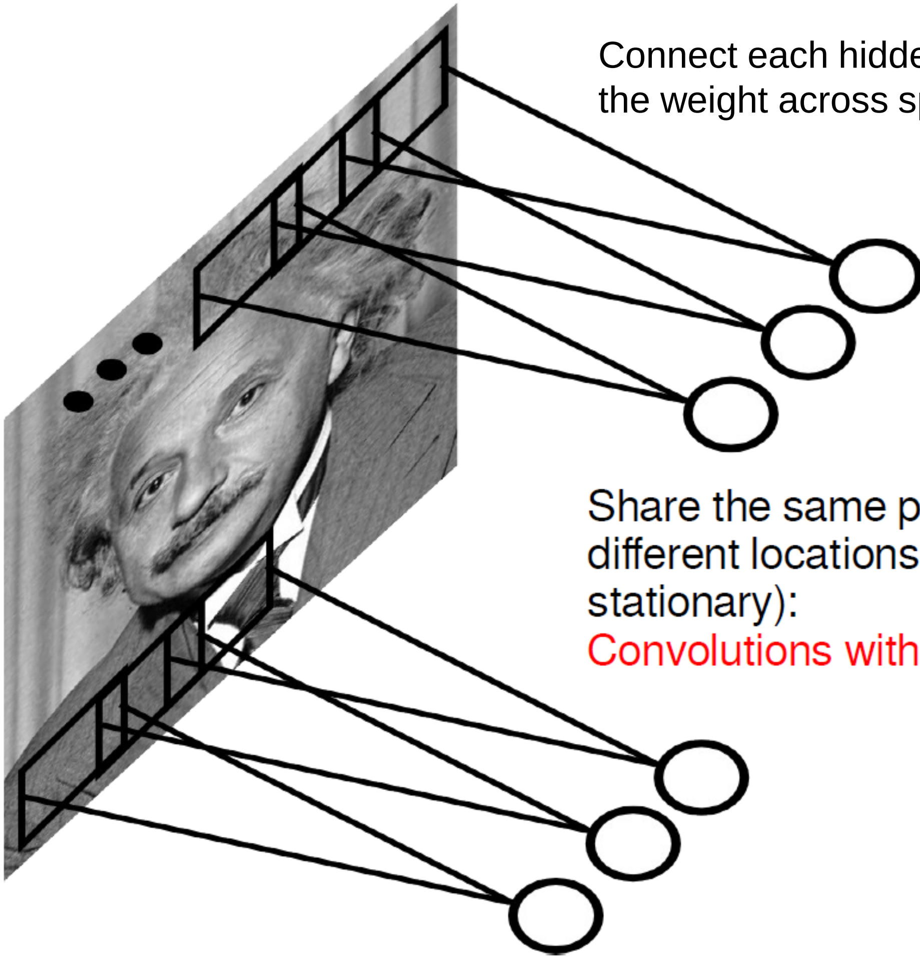
Ranzato

Source: Image Classification with Deep Learning, Marc'Aurelio Ranzato, Facebook A.I. Research
Stanford CS231A www.cs.toronto.edu/~ranzato

Convolutional Layer

Idea: statistics are similar at different locations - stationarity (Lecun 1998)

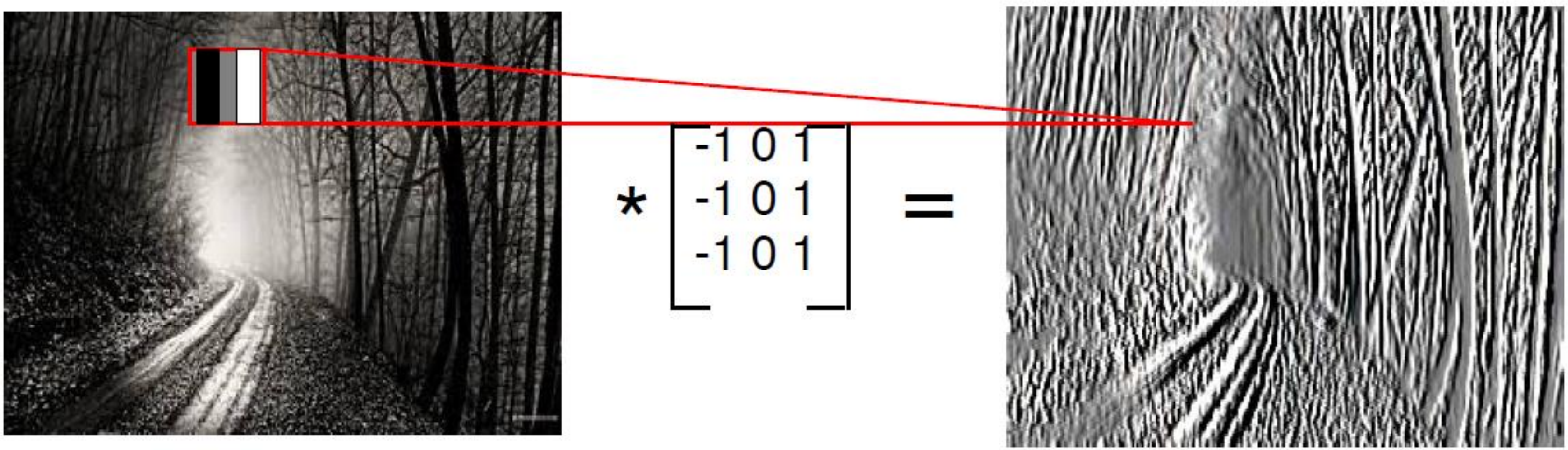
Connect each hidden unit to a small input patch and share the weight across space: convolution layer.



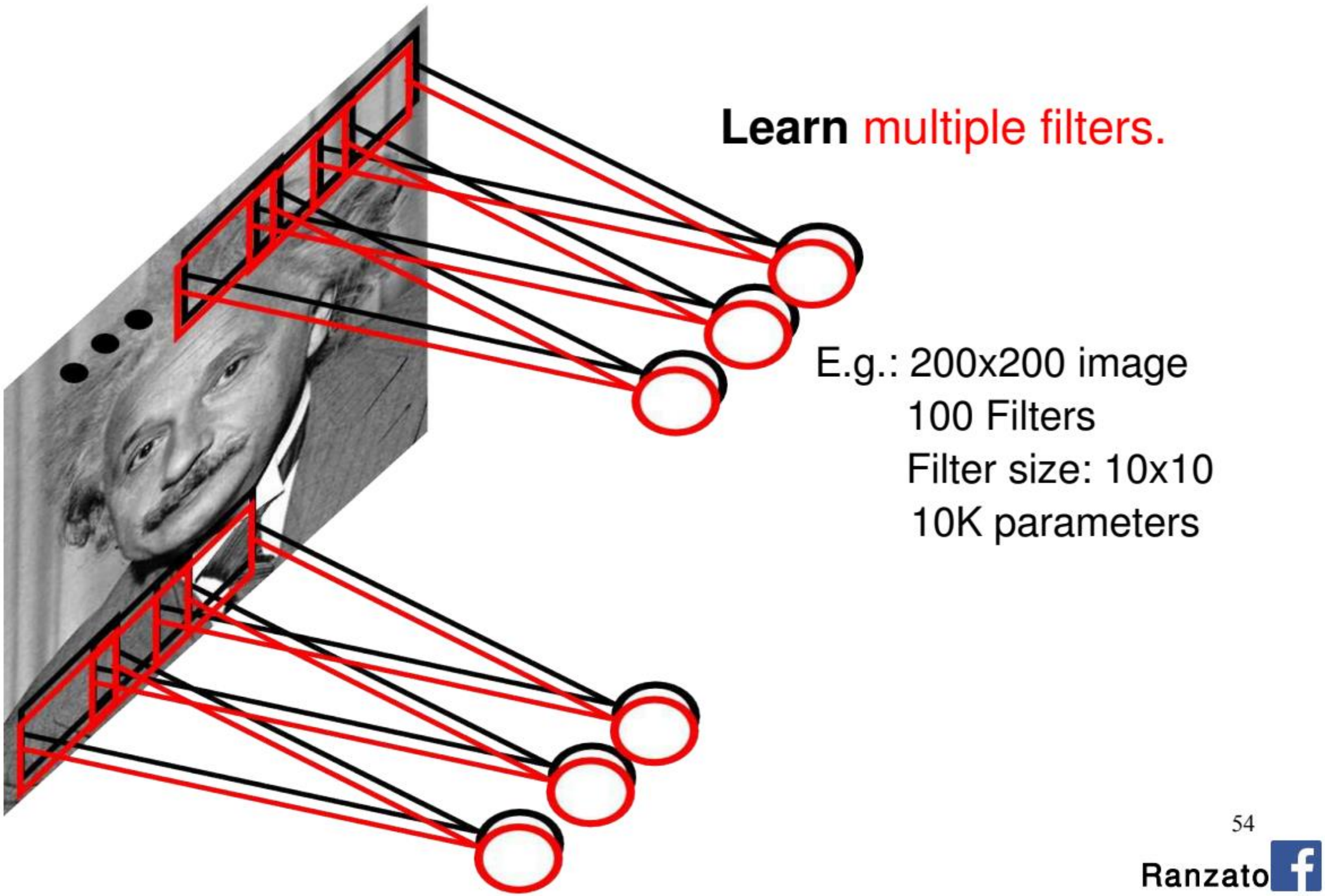
Share the same parameters across different locations (assuming input is stationary):
Convolutions with learned kernels

Ranzato

Convolution Layer

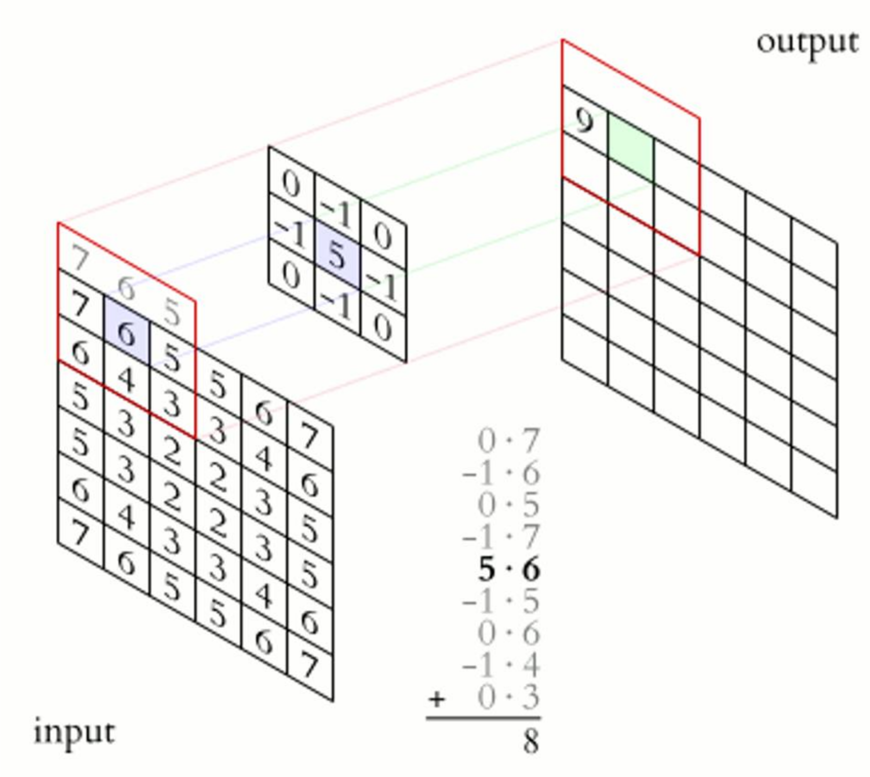


Convolution Layer



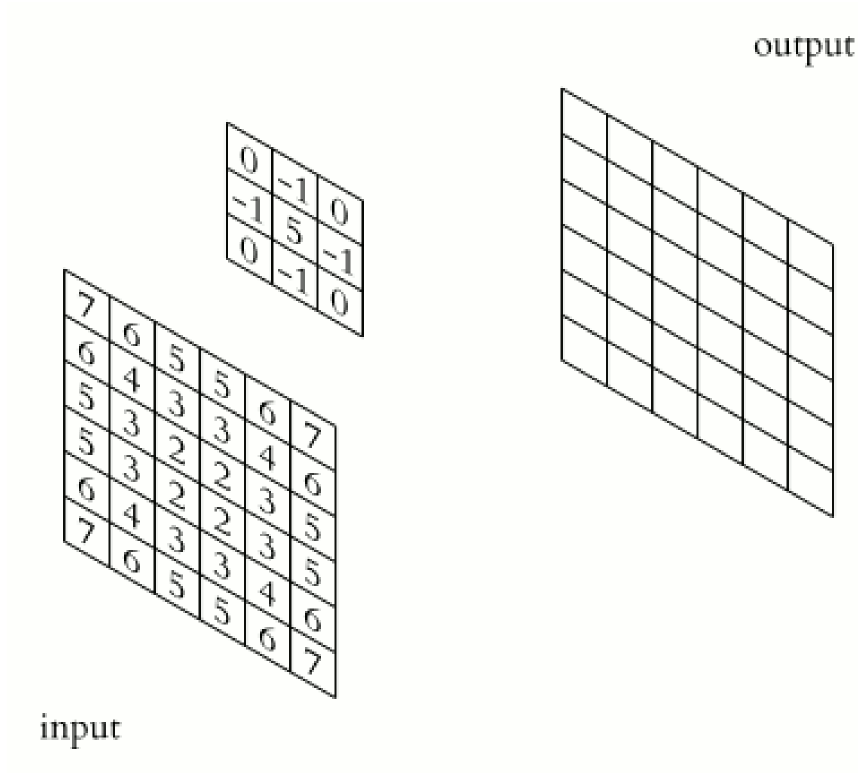
Source: Image Classification with Deep Learning, Marc'Aurelio Ranzato, Facebook A.I. Research
Stanford CS231A www.cs.toronto.edu/~ranzato

Illustration of Convolution Operation



Source: https://medium.com/data-science-group-iitr/building-a-convolutional-neural-network-in-python-with-tensorflow-d251c3ca8117

Illustration of Convolution Operation



Source: https://medium.com/data-science-group-iitr/building-a-convolutional-neural-network-in-python-with-tensorflow-d251c3ca8117

Convolution Operation

Question:
Kernel size: KxK
Input size: DxD
Stride: 1

What is the size of the output?
What's the computational cost?

Answer:
- the output has size (D-K+1)x(D-K+1)

- What is the total computational cost?

- the kernel has KxK coefficients
- cost: K*K*(D-K+1)*(D-K+1)

Convolution Operation

Question:

Kernel size: KxK

Stride: 1

Input size: DxD - > output size (D-K+1)x(D-K+1)

What could we do to keep the output size the same?

Answer:

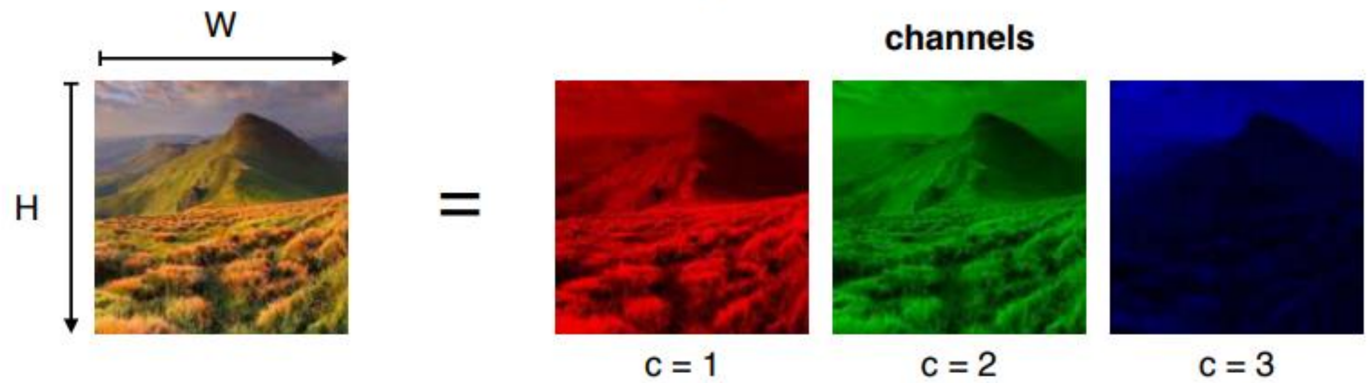
- Padding

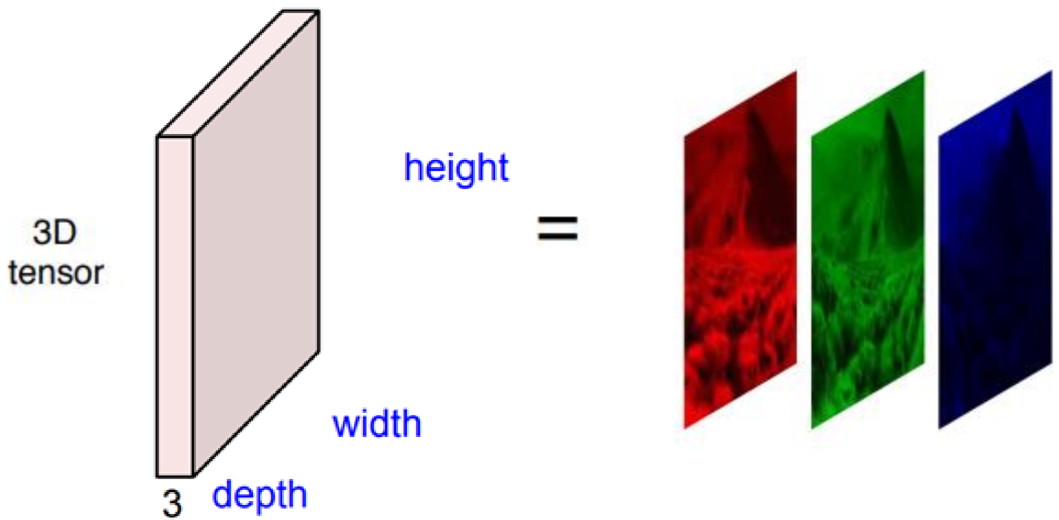
4	5	8
6	3	5
7	5	6

Zero-padding →

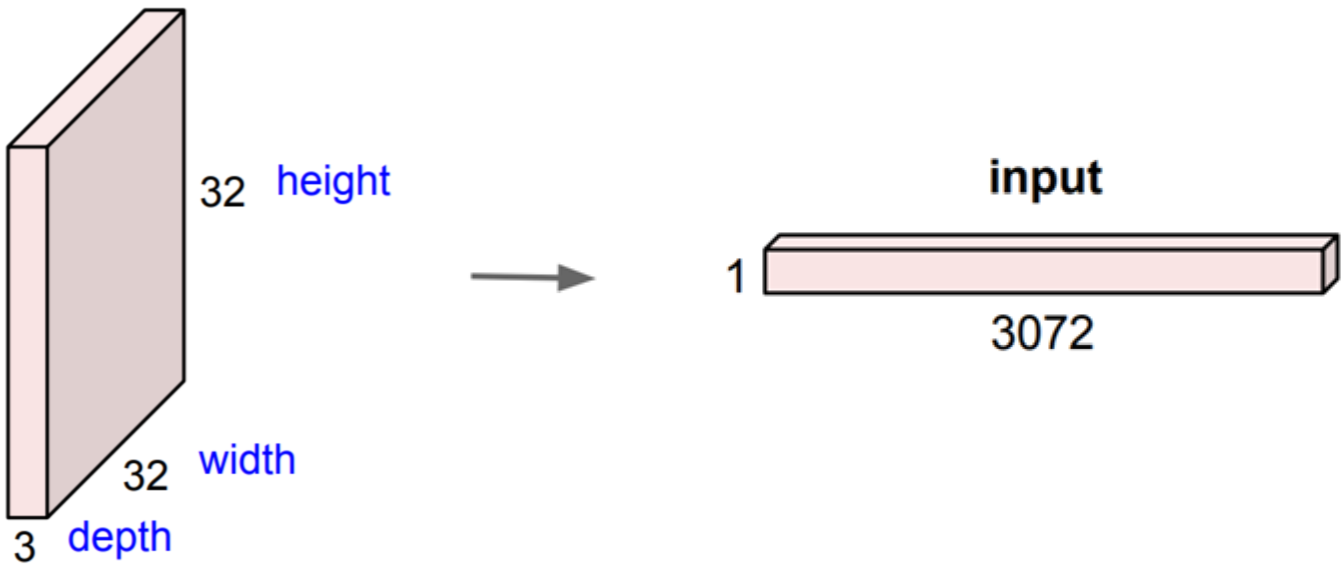
0	0	0	0	0
0	4	5	8	0
0	6	3	5	0
0	7	5	6	0
0	0	0	0	0

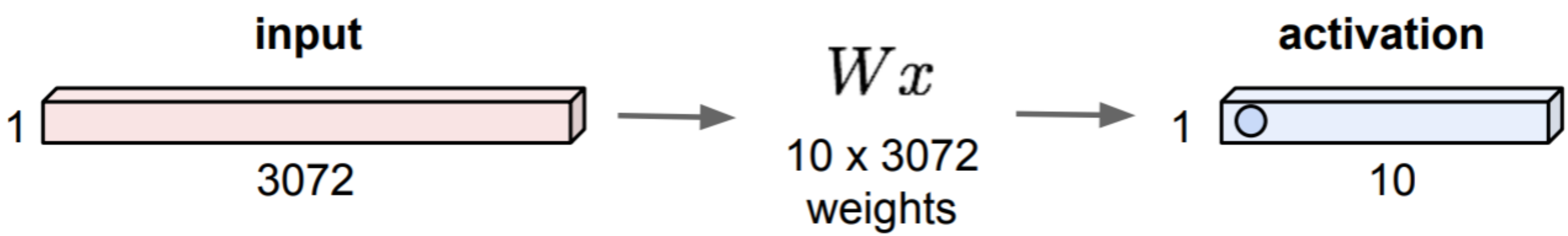
RGB Images as 3-D Tensors





Flatten the tensor and fully connect

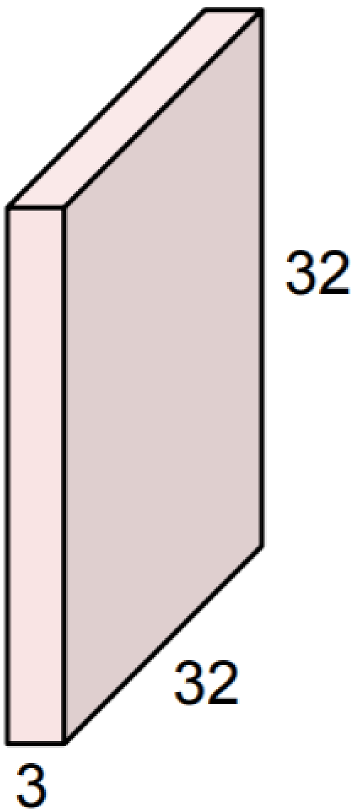




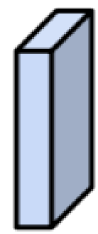
Source: http://cs231n.github.io/convolutional-networks/

Convolution Operation

32x32x3 image



5x5x3 filter



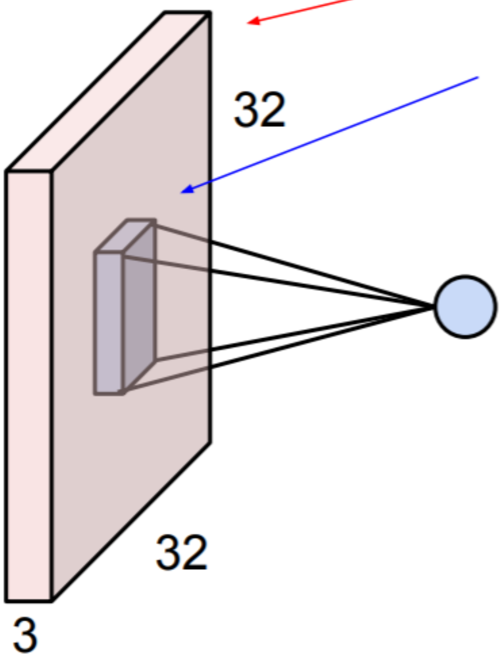
Filters always extend the full depth of the input volume

Source: http://cs231n.github.io/convolutional-networks/

Convolution Operation

32x32x3 image

5x5x3 filter w



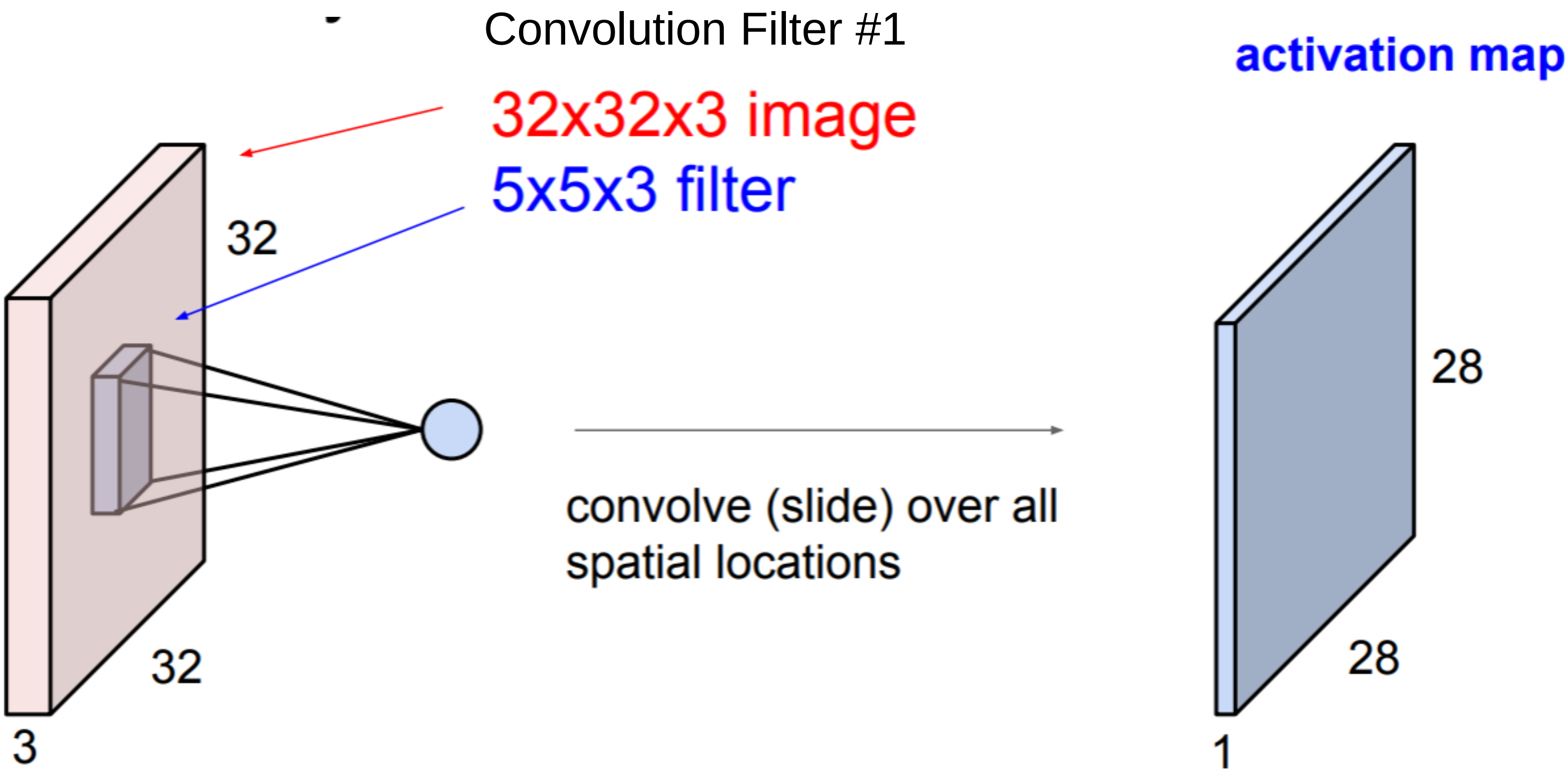
1 number:

the result of taking a dot product between the filter and a small 5x5x3 chunk of the image (i.e. 5*5*3 = 75-dimensional dot product + bias)

$$w^T x + b$$

Source: http://cs231n.github.io/convolutional-networks/

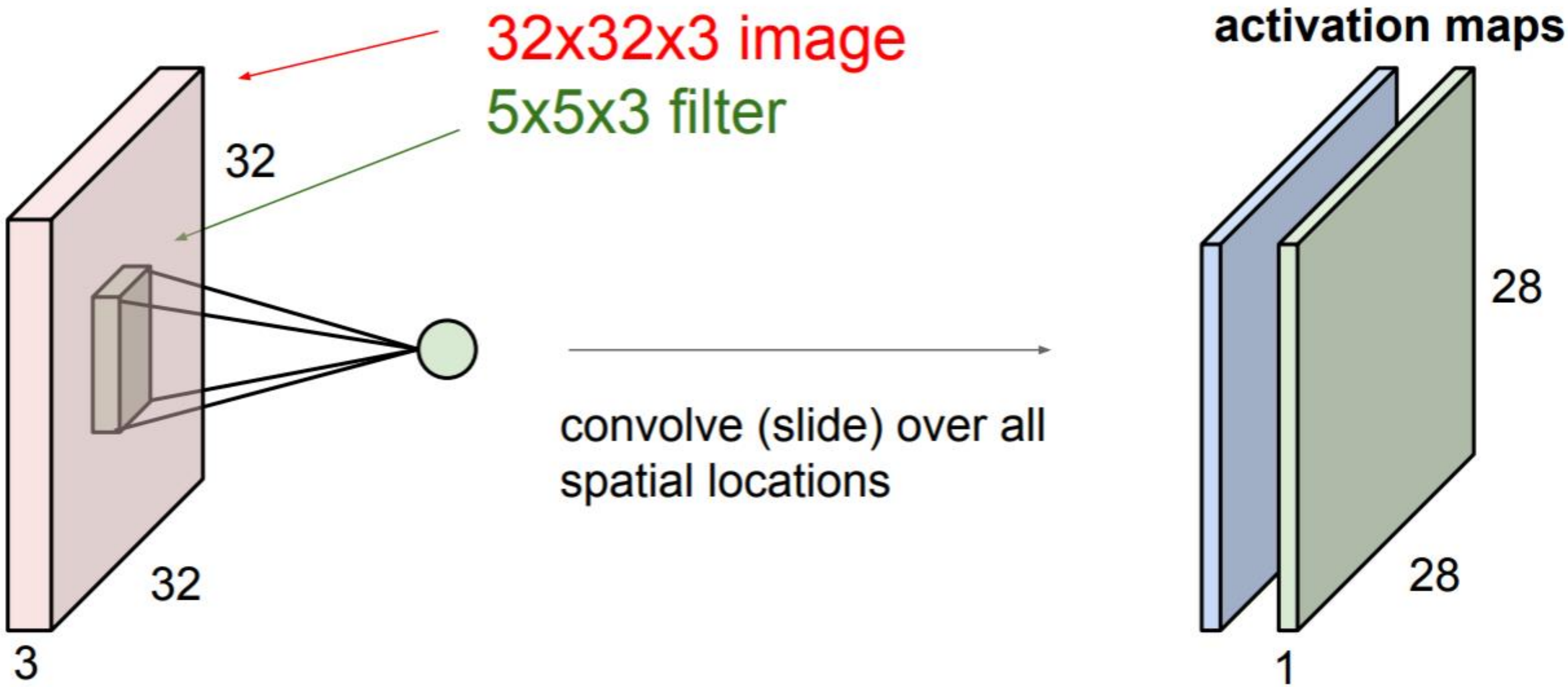
Convolution Operation



Source: http://cs231n.github.io/convolutional-networks/

Convolution Operation

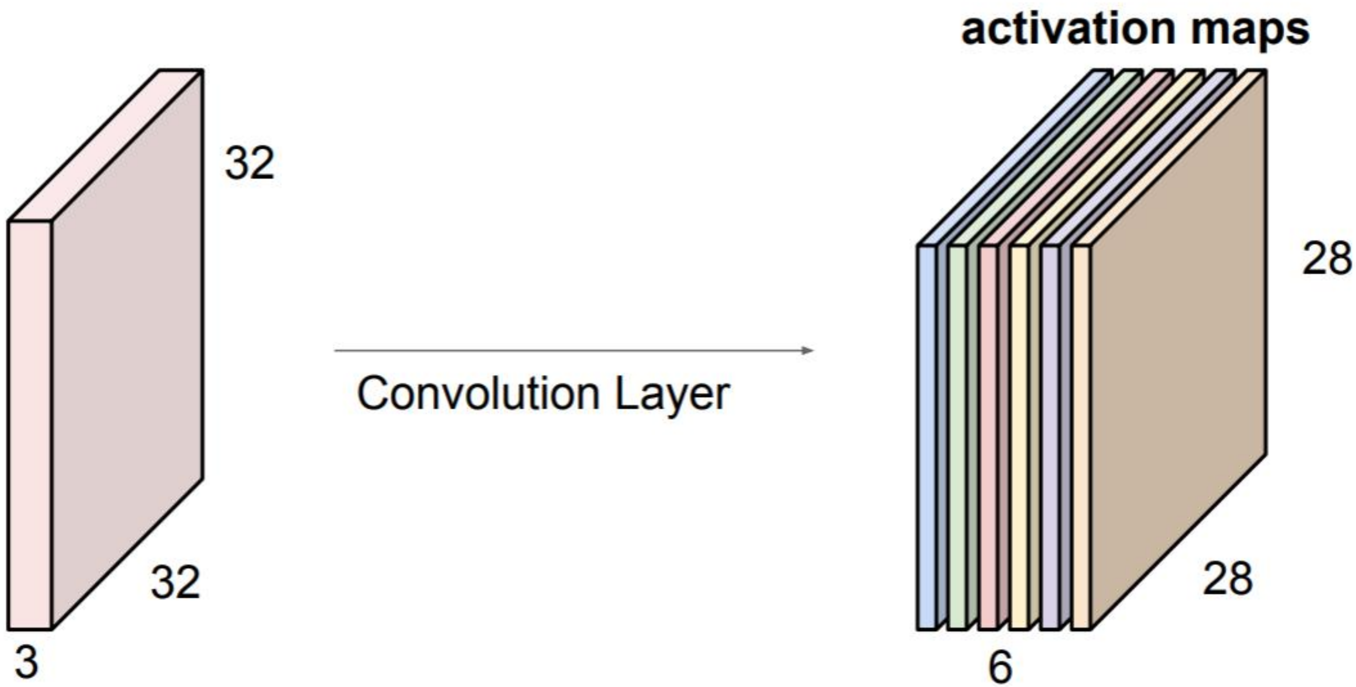
Convolution Filter #2



Source: http://cs231n.github.io/convolutional-networks/

Convolution Operation

For example, if we had 6 5x5 filters, we'll get 6 separate activation maps:



We stack these up to get a "new image" of size 28x28x6!

Source: http://cs231n.github.io/convolutional-networks/

Convolution Operation

Question:
Kernels: N@KxK
Input size: DxD, M channels
Stride: 1

What is the size of the output?
What's the computational cost?

Answer:

- We will have N output feature maps and the output has size: N@(D-K+1)x(D-K+1)
- the kernels have N x M x K x K coefficients
- cost: M*K*K*N*(D-K+1)*(D-K+1)

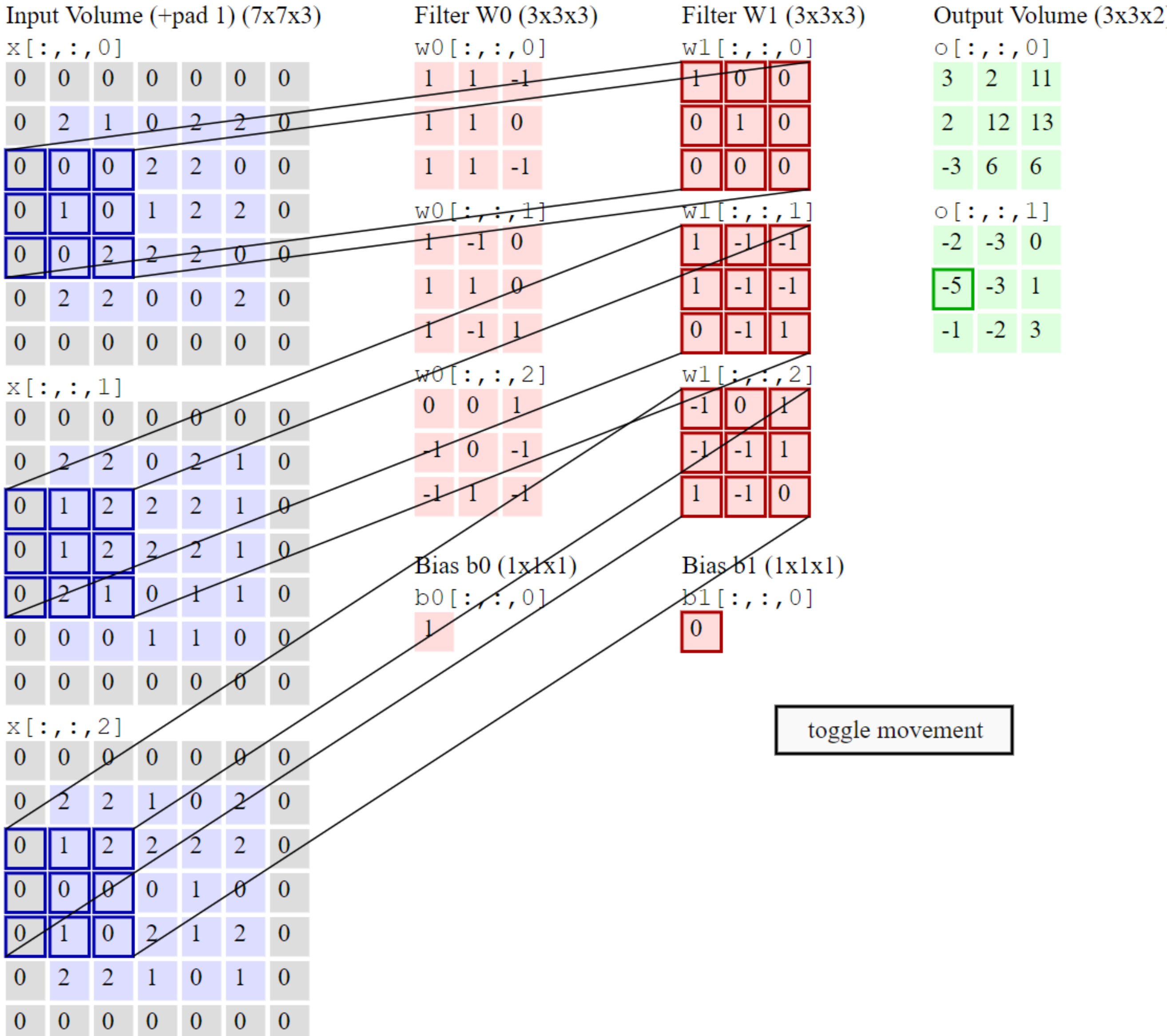
Illustration of Convolution Operation

- The input is RGB image (3D volume), the filters are also 3D
- Since 3D volumes are hard to visualize, all the volumes are visualized with each depth slice stacked in rows
  - the input volume: blue
  - the weight volumes: red
  - the output volume: in green

- The input volume is of size 5x5x3

- CONV layer parameters are N=2,K=3,S=2,P=1
  - That is, we have two filters of size 3×3 and they are applied with a stride of 2.
  - Notice that a padding of P=1 is applied to the input volume, making the outer border of the input volume zero.
  - Therefore, the output volume size has spatial size (5 - 3 + 2)/2 + 1 = 3.

Source: http://cs231n.github.io/convolutional-networks/

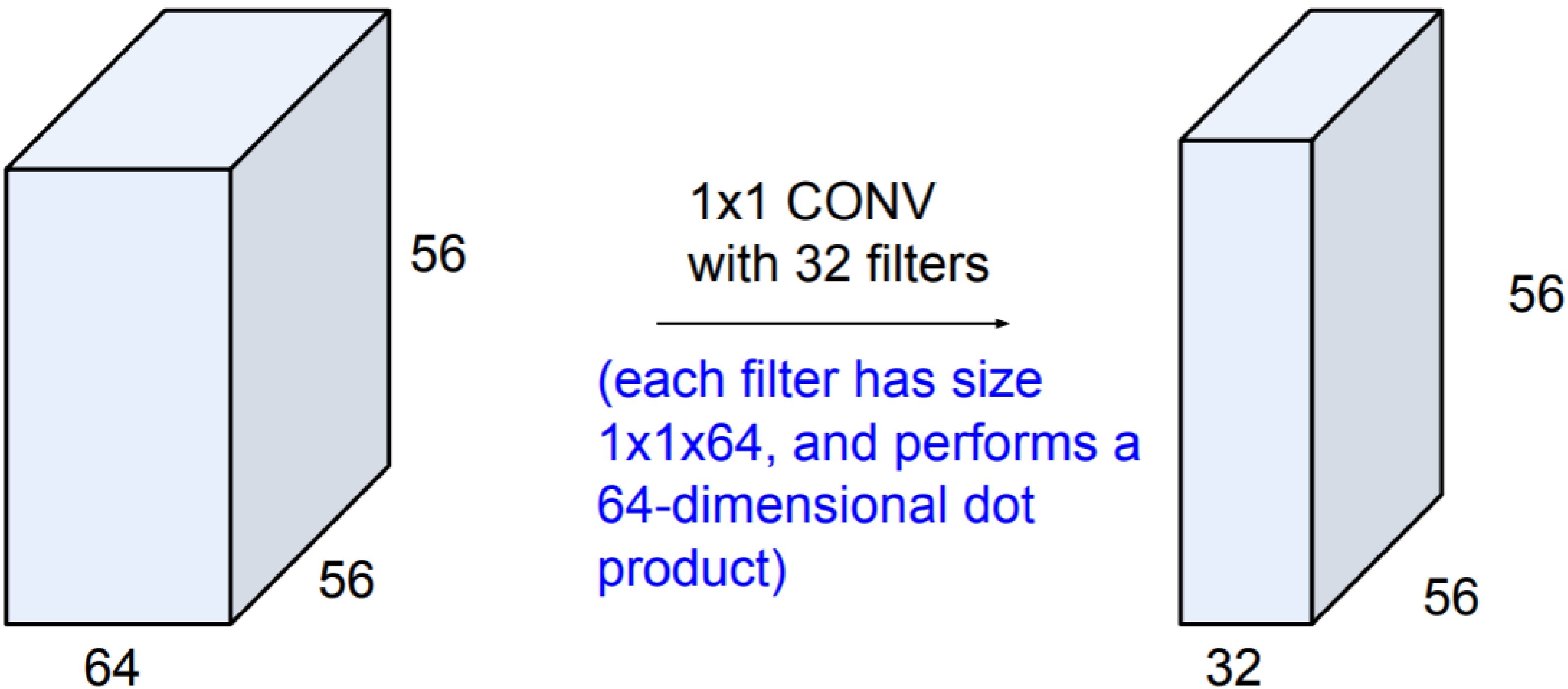


Source: http://cs231n.github.io/convolutional-networks/

1x1 convolutions

- It might be confusing to see 1x1 convolutions in several papers, especially if you are from signal processing background.
- For 2-dimensional signals 1x1 convolutions do not make sense (it’s just pointwise scaling).
- However, in ConvNets this is not the case because one must remember that we operate over multi-dimensional volumes, and that the filters always extend through the full depth of the input volume.
- For example, if the input is [32x32x3] then doing 1x1 convolutions would effectively be doing 3-dimensional dot products (since the input depth is 3 channels).

1x1 convolutions



Accelerating Convolution Calculations

- When you use a Fourier transform on both the kernel and the feature map, then the convolute operation is simplified significantly (integration becomes mere multiplication).

- Convolution in the frequency domain can be faster than in the time domain by using the Fast Fourier Transform (FFT) algorithm.

- Some of the fastest GPU implementations of convolutions (for example some implementations in the NVIDIA cuDNN library) currently make use of Fourier transforms.

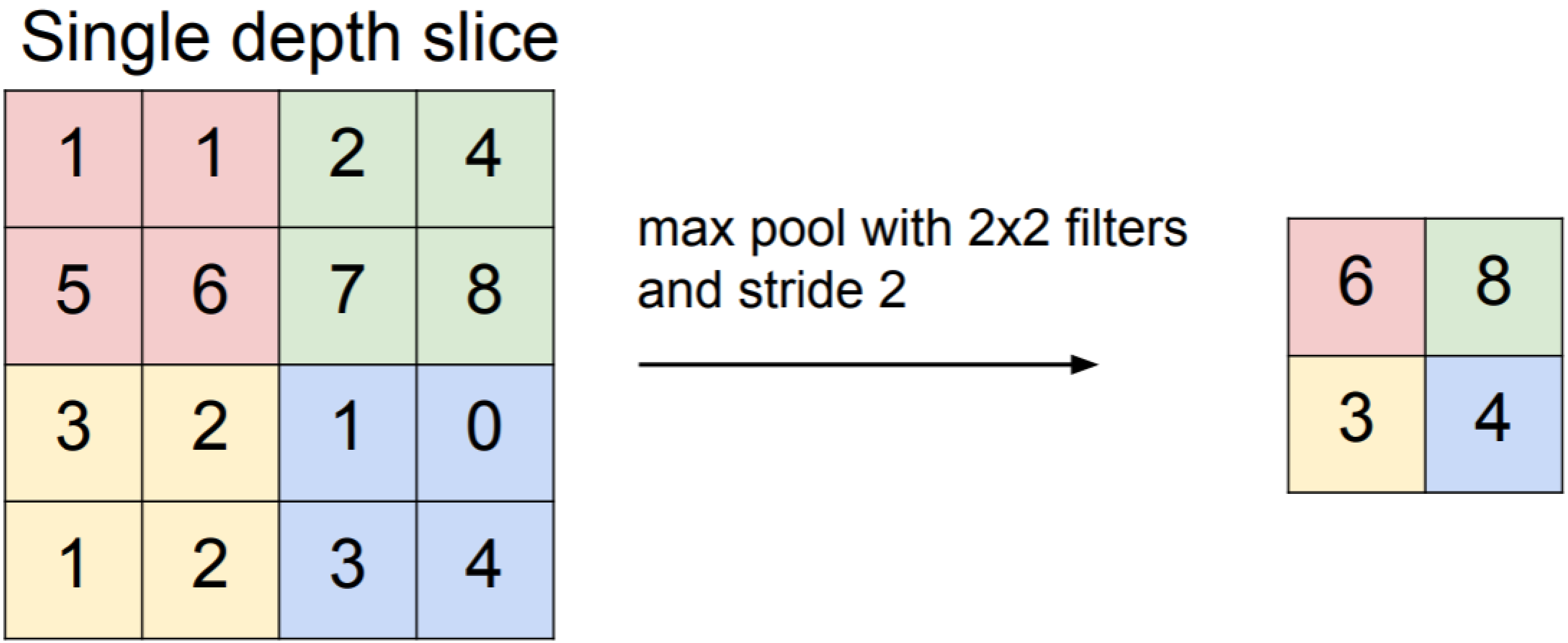
- Tensor cores in Volta and Turing Architecture GPUs accelerate convolution operation in hardware.

- Tensor cores are accessible and exposed as Warp-Level Matrix Operations in the CUDA 9 C++ API. They are also used by cuDNN.

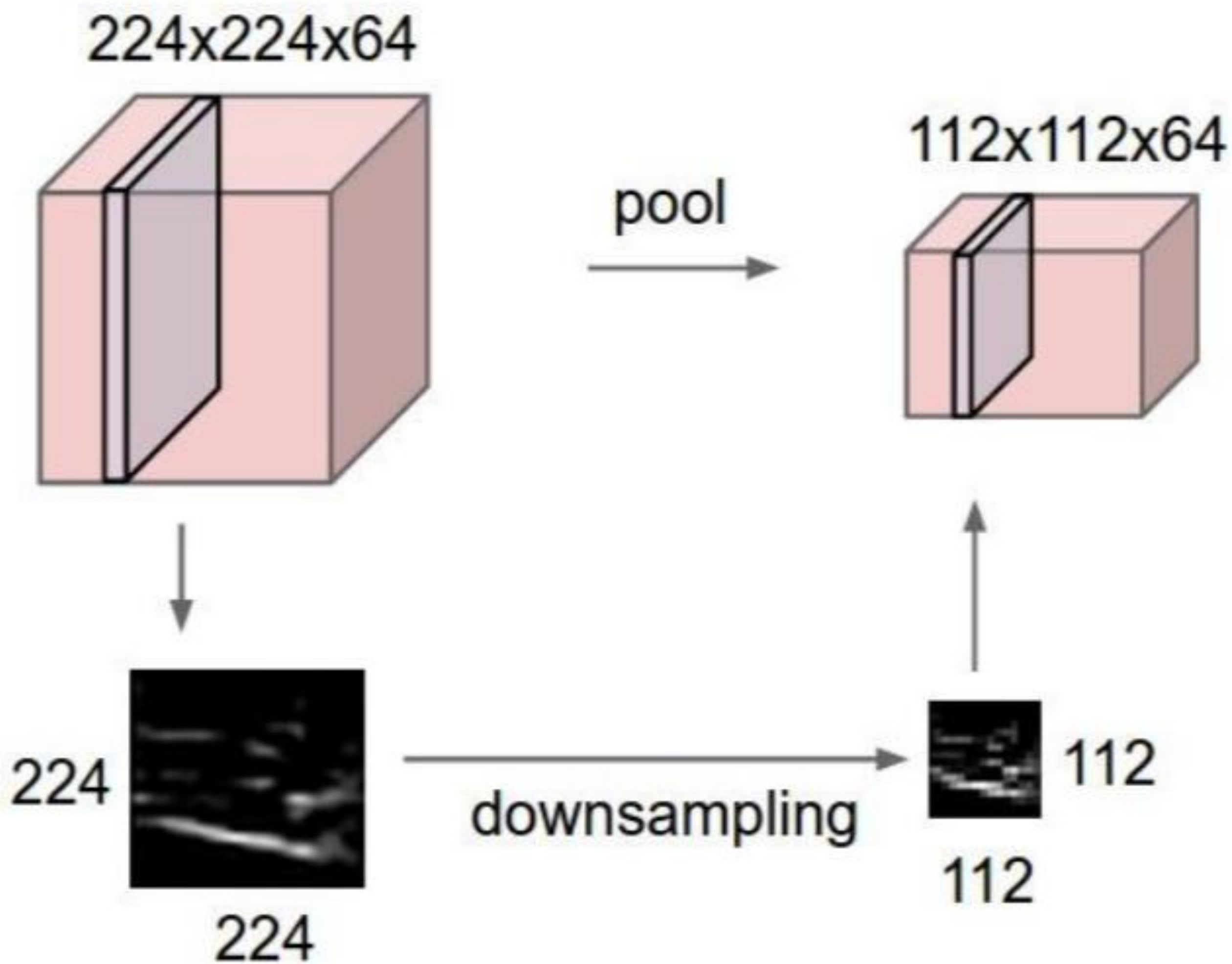
Pooling

- Pooling is a down-sampling technique
  - Reduces the spatial size of the representation
  - Reduces number of parameters and number of computations (in upcoming layer)
  - Limits overfitting
- No parameters (weights) in the pooling layer
- Typically involves using MAX operation with a 2 X 2 or 3x3 filter with a stride of 2

Max Pooling


Single depth slice
1 1 2 4
5 6 7 8
3 2 1 0
1 2 3 4
max pool with 2x2 filters
and stride 2
6 8
3 4

Pooling layer



Pooling Operation

Question:
Filter size: FxF
Input size: M@DxD
Stride: 1

What is the size of the output?

Answer:

- The output will have M slices and the output has size M@(D-F+1)x(D-F+1)

Pooling Operation – with stride

Question:
Filter size: FxF
Input size: M@DxD
Stride: S

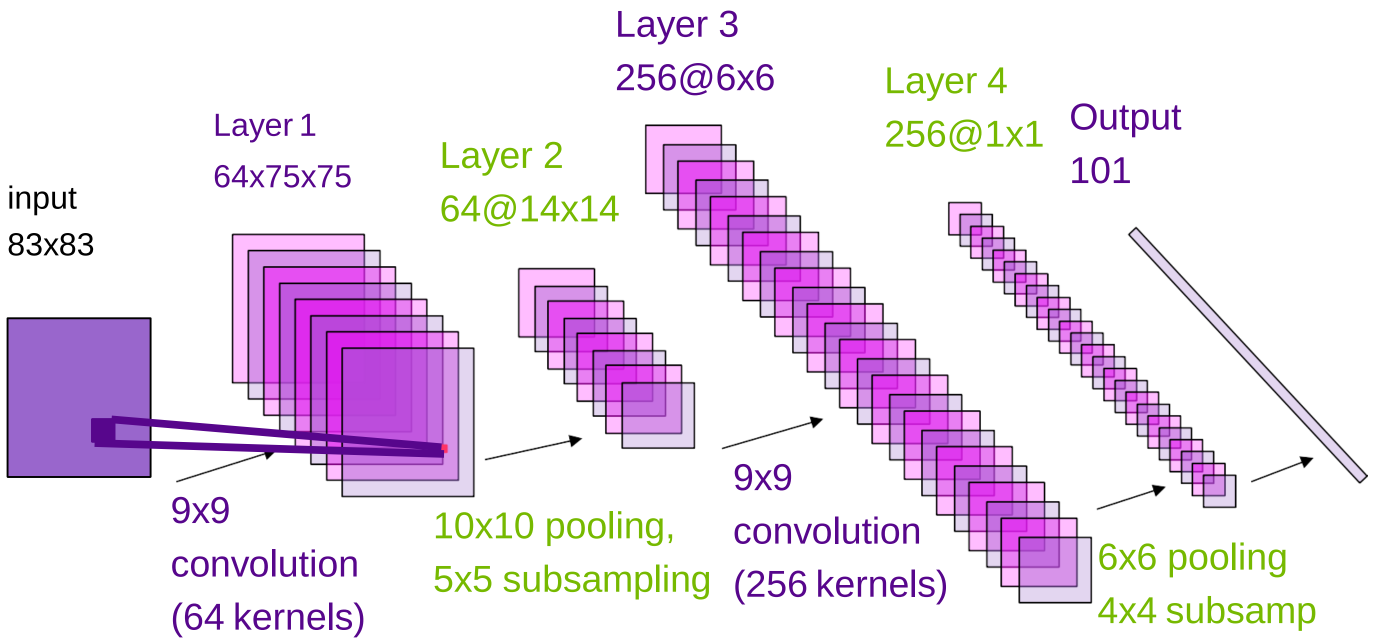
What is the size of the output?

Answer:
- the output has size M@ ((D-F)/S + 1) x ((D-F)/S + 1)

Pooling

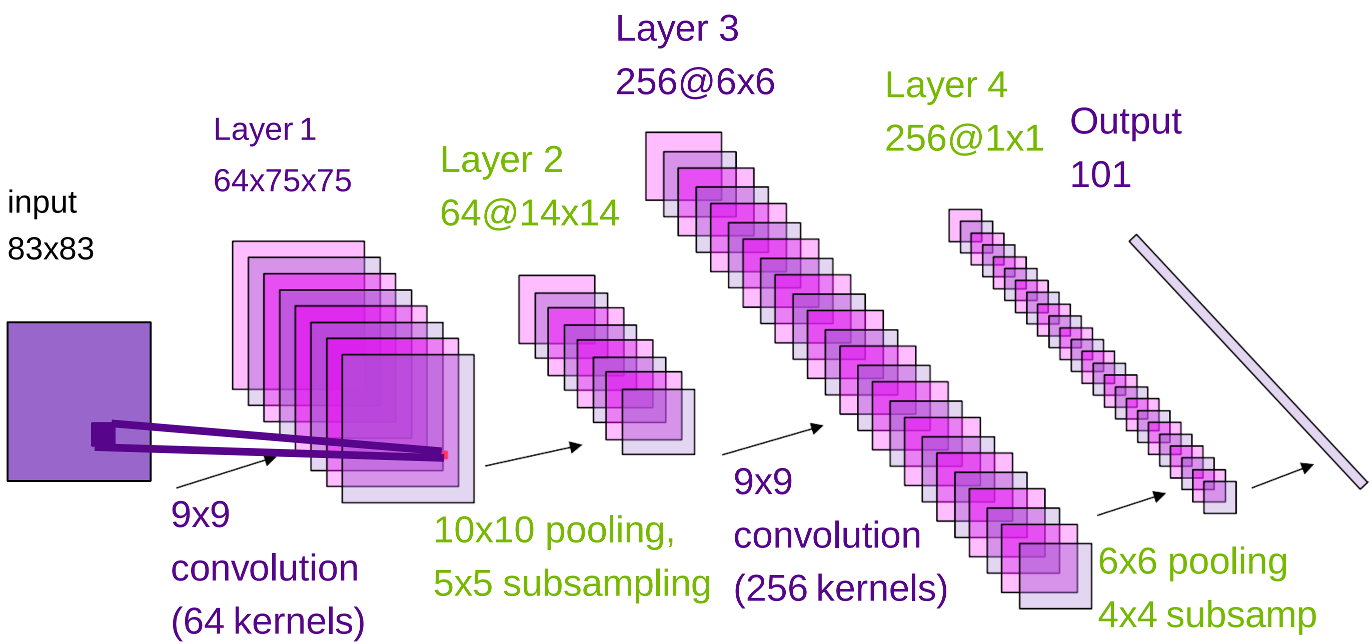
- Notice that pooling does not introduce any new parameters since it computes a fixed function of the input.
- It is not common to use zero-padding for pooling
- Usually there is a stride
- Average pooling (taking the average of the values falling in the area of the filter) could also be used instead of max pooling

Convolutional Network (ConvNet)



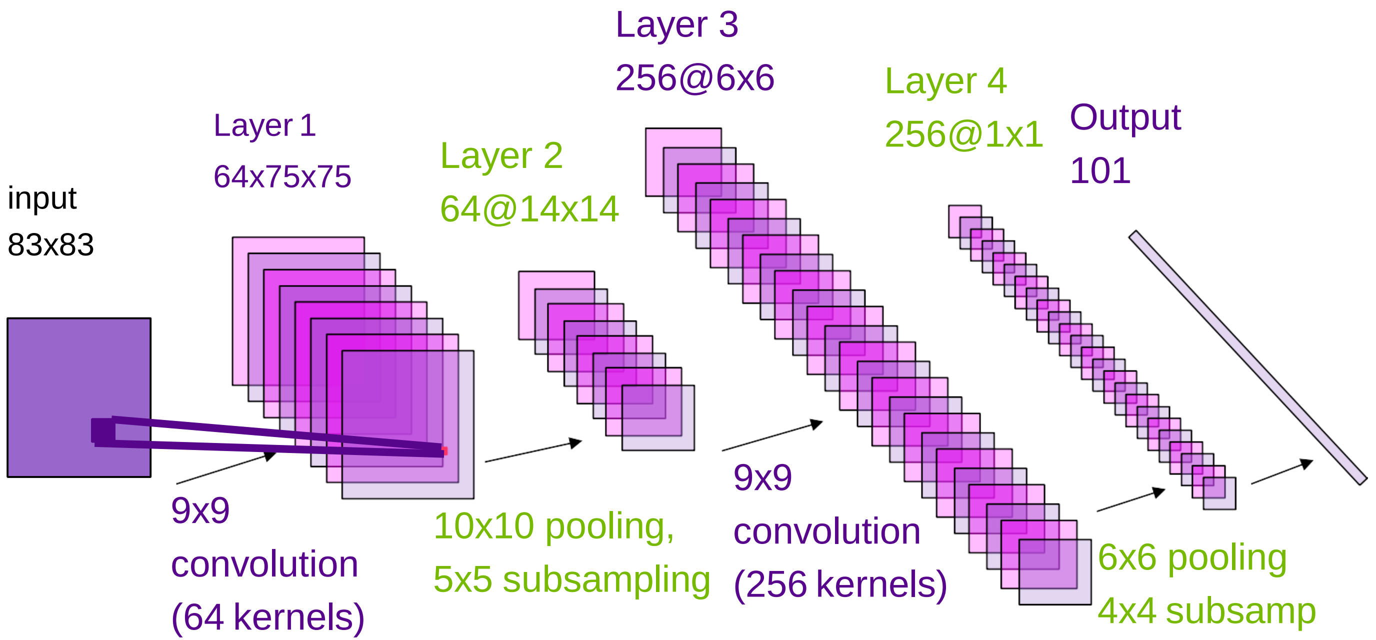
First convolution: (83-9)+1 = 75x75, 64 kernels -> 64@75x75

Convolutional Network (ConvNet)



First pooling: (75-10)/5 +1 = 14x14, 64 input slices -> 64@14x14

Convolutional Network (ConvNet)



Second convolution: (14-9) +1 = 6, 256 kernels-> 256@6x6