CONVOLUTIONAL NEURAL NETWORK

한재근 과장 | jahan@nvidia.com
유현곤 부장 | hryu@nvidia.com / 양한별 과장 | hanbyuly@nvidia.com
NVIDIA

AGENDA

Background

Convolution Layer & Gradient descent

Pooling Layer

Normalization

CNN Models

What is a convolution?

A refresher of 2D convolutions

Image

I1	I2	I3	I4	I5	I6
I7	I8	I9	I10	I11	I12
I13	I14	I15	I16	I17	I18
I19	I20	I21	I22	I23	I24
I25	I26	I27	I28	I29	I30
I31	I32	I33	I34	I35	I36

Filter

F1	F2	F3
F4	F5	F6
F7	F8	F9

O1

= F1*I1 + F2*I2 + F3*I3 + F4*I7 + F5*I8 + F6*I9 + F7*I13 + F8*I14 + F9*I15

What is a convolution?

A refresher of 2D convolutions

Image

I1	I2	I3	I4	I5	I6
I7	I8	I9	I10	I11	I12
I13	I14	I15	I16	I17	I18
I19	I20	I21	I22	I23	I24
I25	I26	I27	I28	I29	I30
I31	I32	I33	I34	I35	I36

Filter

F1	F2	F3
F4	F5	F6
F7	F8	F9

O1	O2

What is a convolution?

A refresher of 2D convolutions

Image

I1	I2	I3	I4	I5	I6
I7	I8	I9	I10	I11	I12
I13	I14	I15	I16	I17	I18
I19	I20	I21	I22	I23	I24
I25	I26	I27	I28	I29	I30
I31	I32	I33	I34	I35	I36

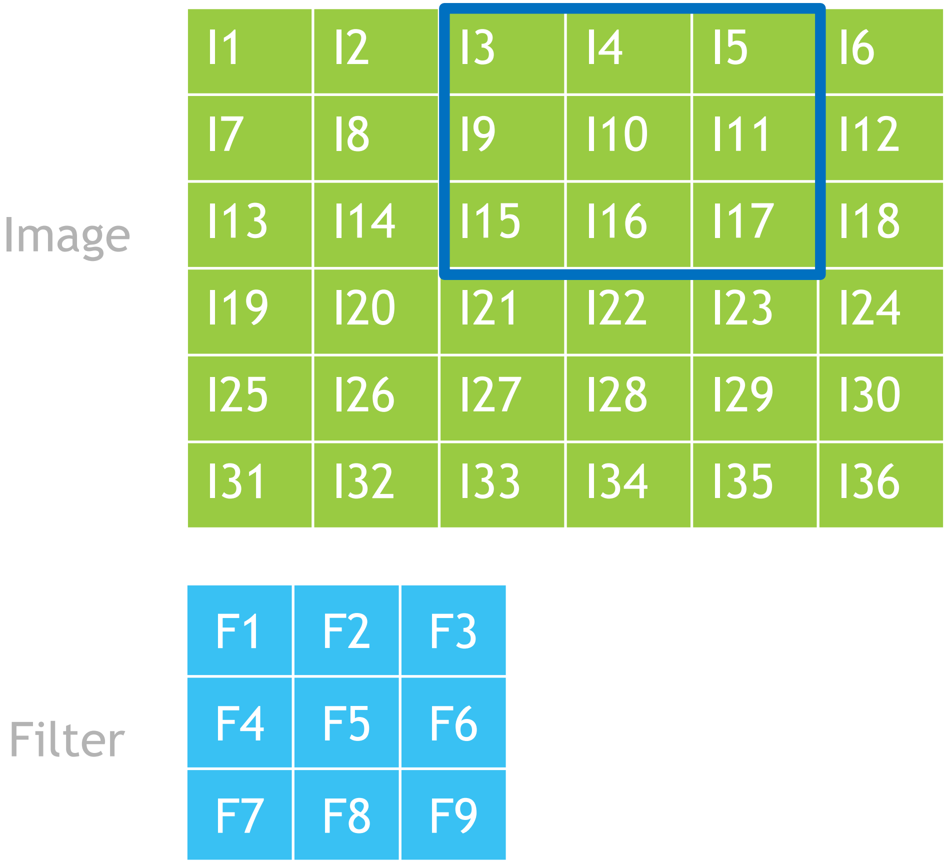
Filter

F1	F2	F3
F4	F5	F6
F7	F8	F9

O1	O2	O3

What is a convolution?

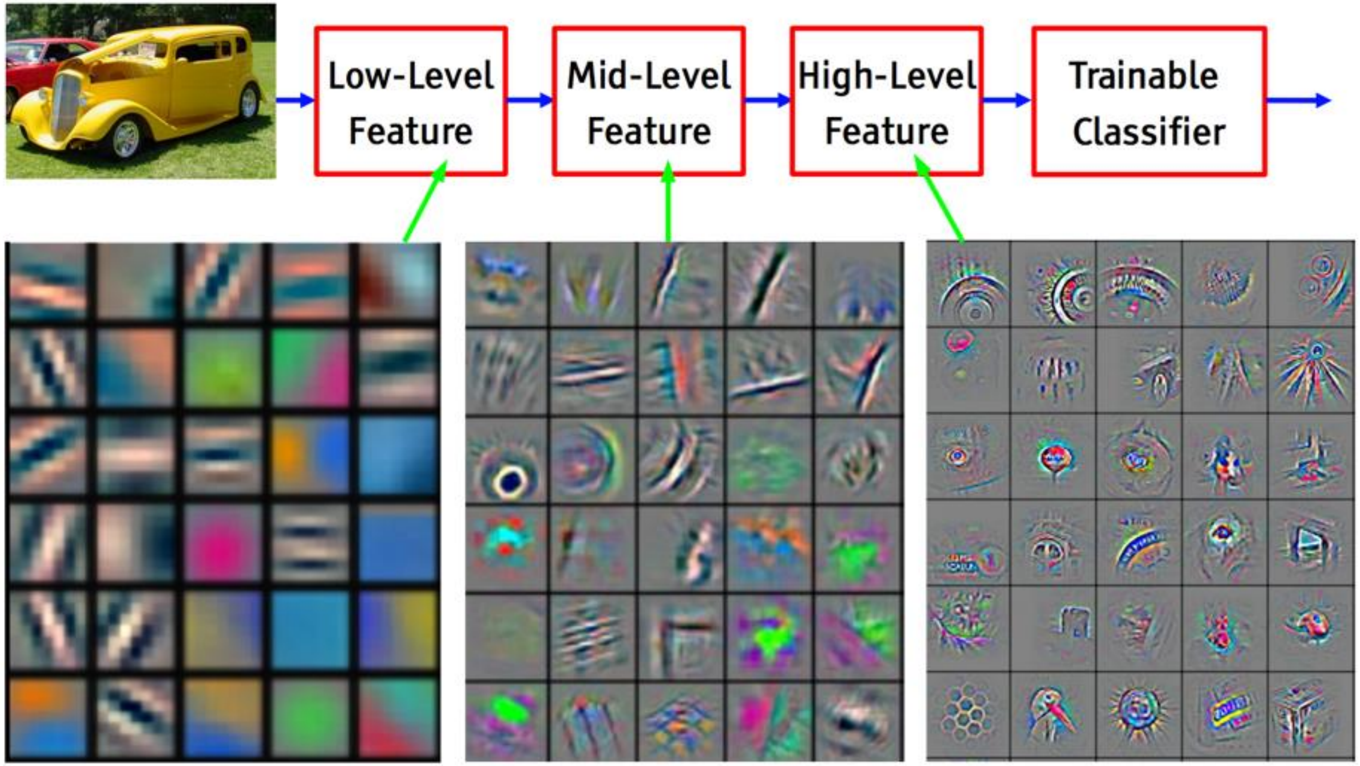
A refresher of 2D convolutions



… and so on until you've covered the entire image

Hierarchical Representations

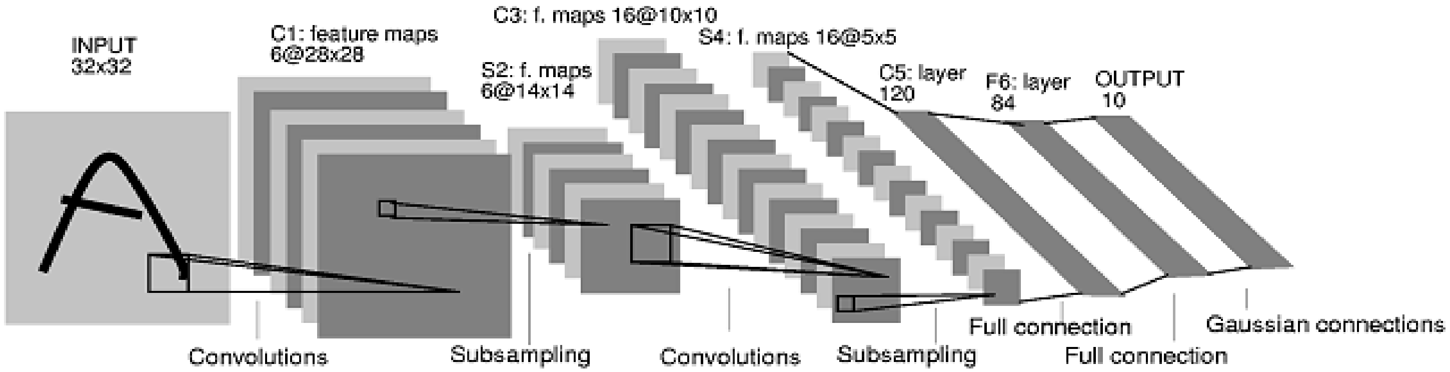
How CNNs work



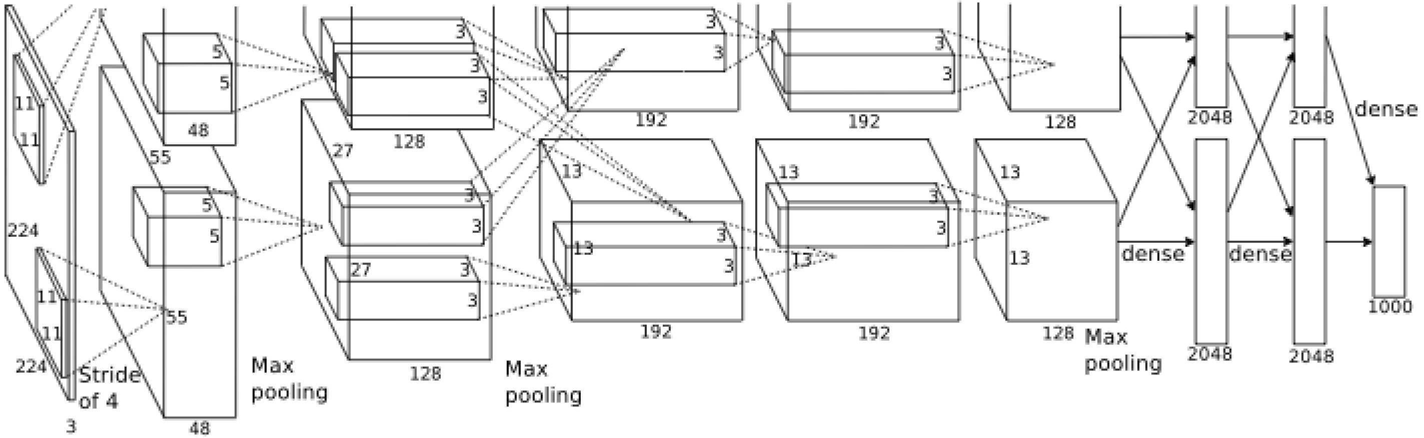
Feature visualization of convolutional net trained on ImageNet from [Zeiler & Fergus 2013]

Slide credit:
Yann LeCun

Some modern CNNs



Y. LeCun et al. 1989-1998 : Handwritten digit reading

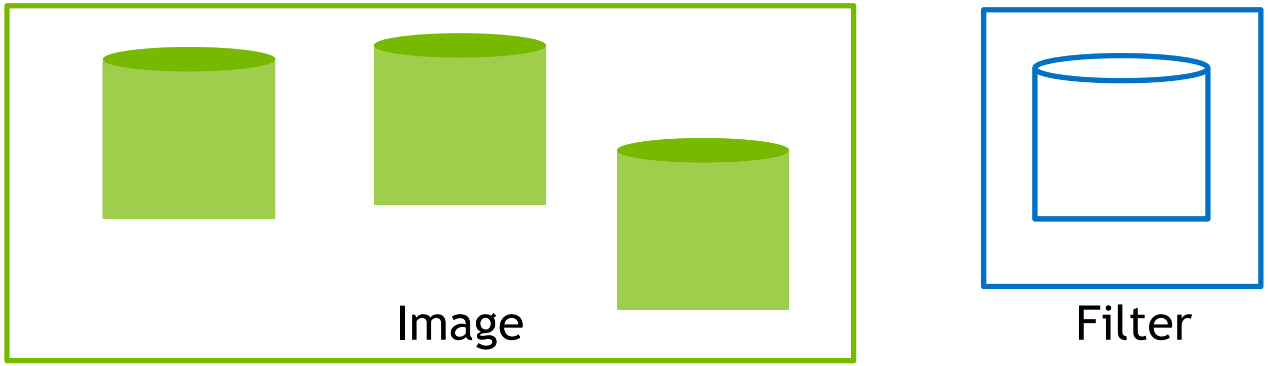


A. Krizhevsky, G. Hinton et al. 2012 : Imagenet classification winner

Convolutional Neural Network

What is it?

- A Neural Network where the linear operator is a convolution
- Convolutions are nice because they’re invariant to translation
  - Filter doesn’t care where in the image the object of interest is located
- Force weight sharing across input pixels



The basic CNN layer

Three components

Translation-invariant linear transformation

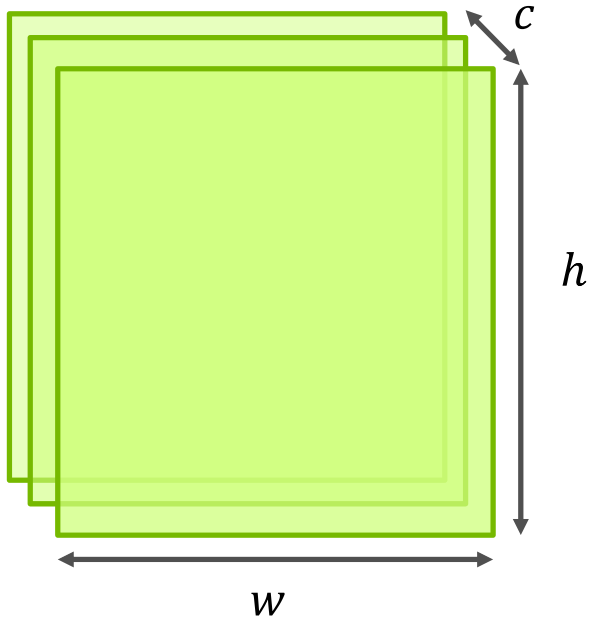
Downsampling

Non-linearity

Together, they comprise a non-linear 2-D filter that’s at the heart of the CNN

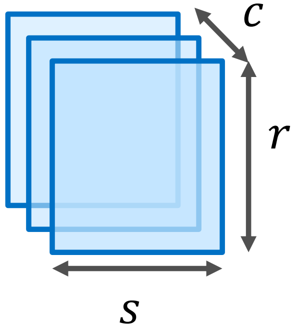
What does our data look like?

Image and Filter



Image

.... A batch of size n

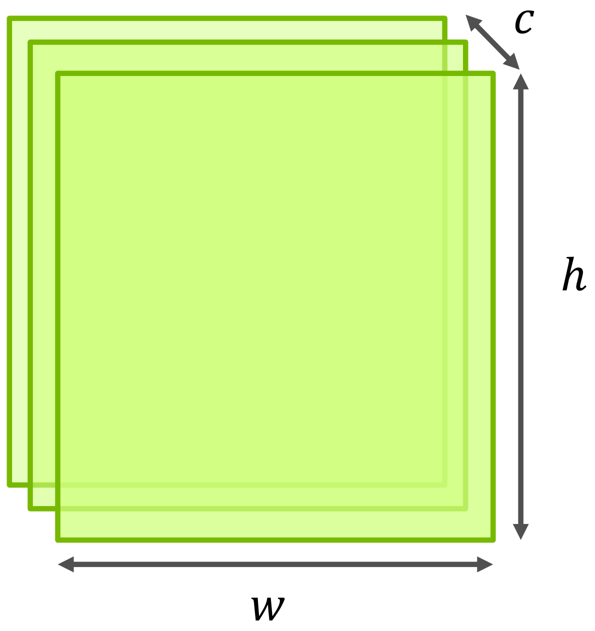


Filter

.... A set of k

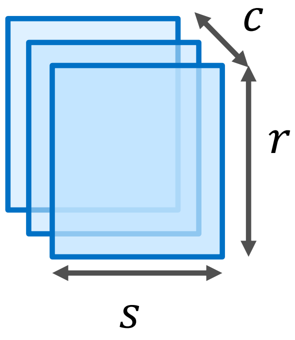
What does our data look like?

Image and Filter



Image

.... A batch of size n



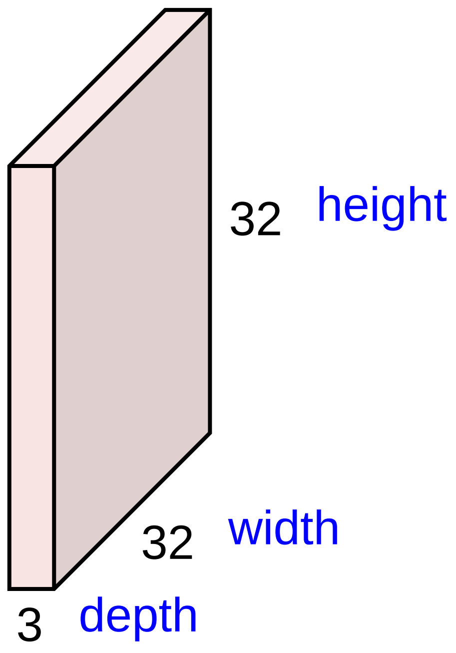
Filter

.... A set of k

Attribute	Symbol
Batch size	n
Input Channels (same for image and filter)	c
Image height x Image width	$h \ x \ w$
Output Channels (equal to number of filters)	k
Filter height x filter width	$r \ x \ s$

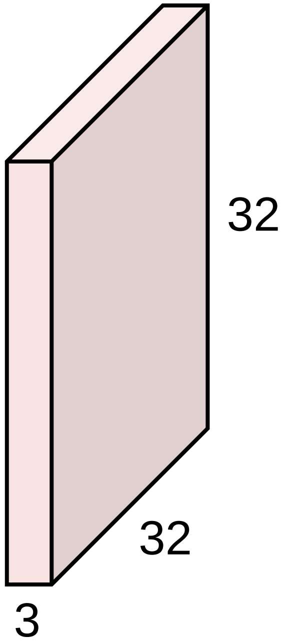
Convolution Layer

32x32x3 image

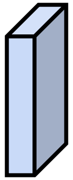


Convolution Layer

32x32x3 image

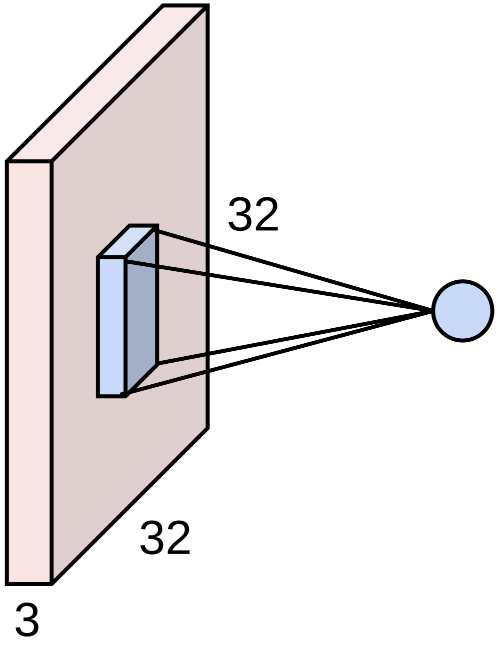


5x5x3 filter



Convolution Layer

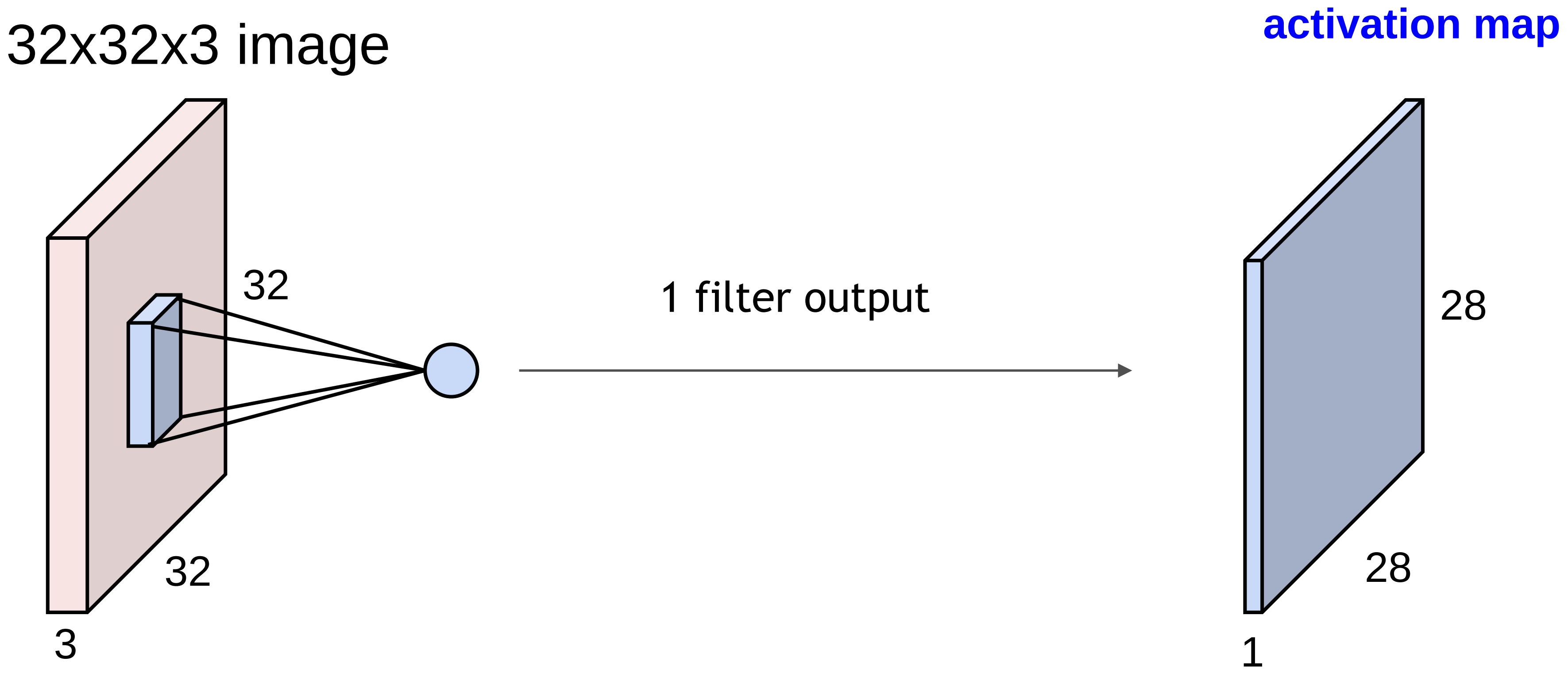
32x32x3 image



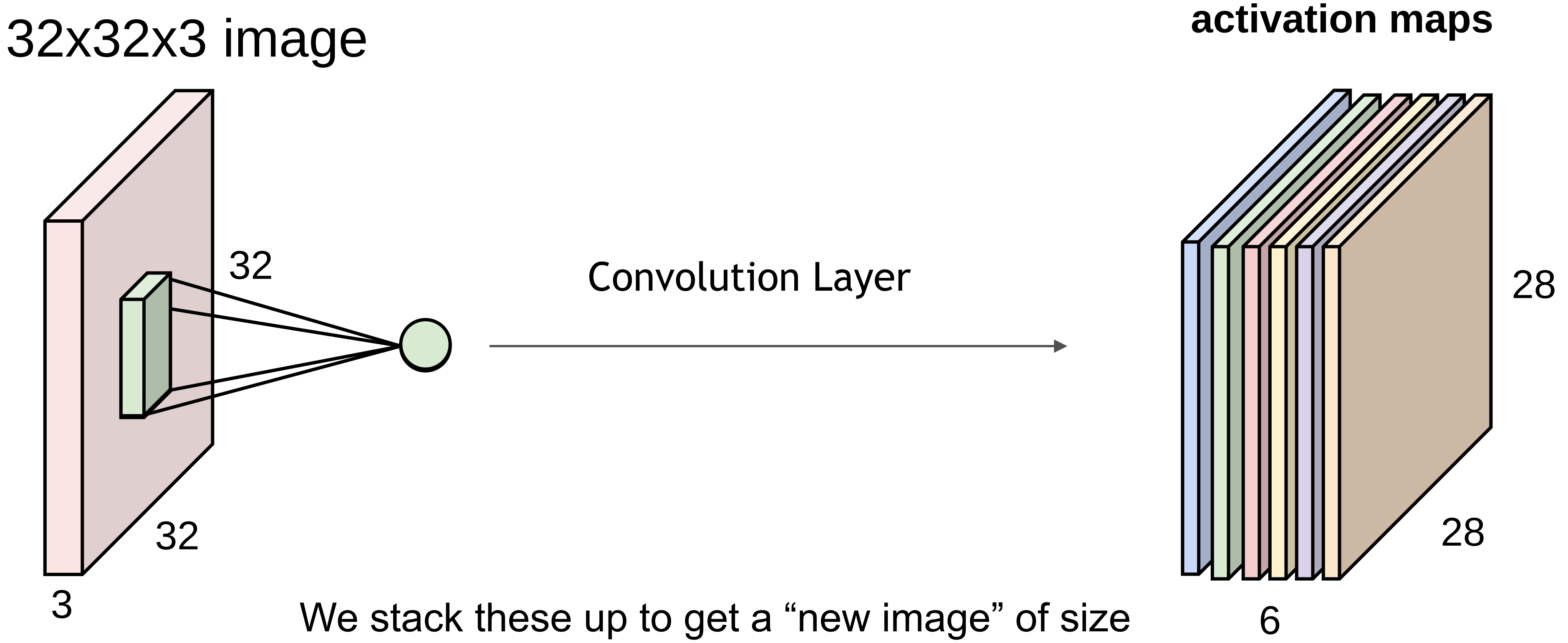
1 number

$$w^T x + b$$

Convolution Layer

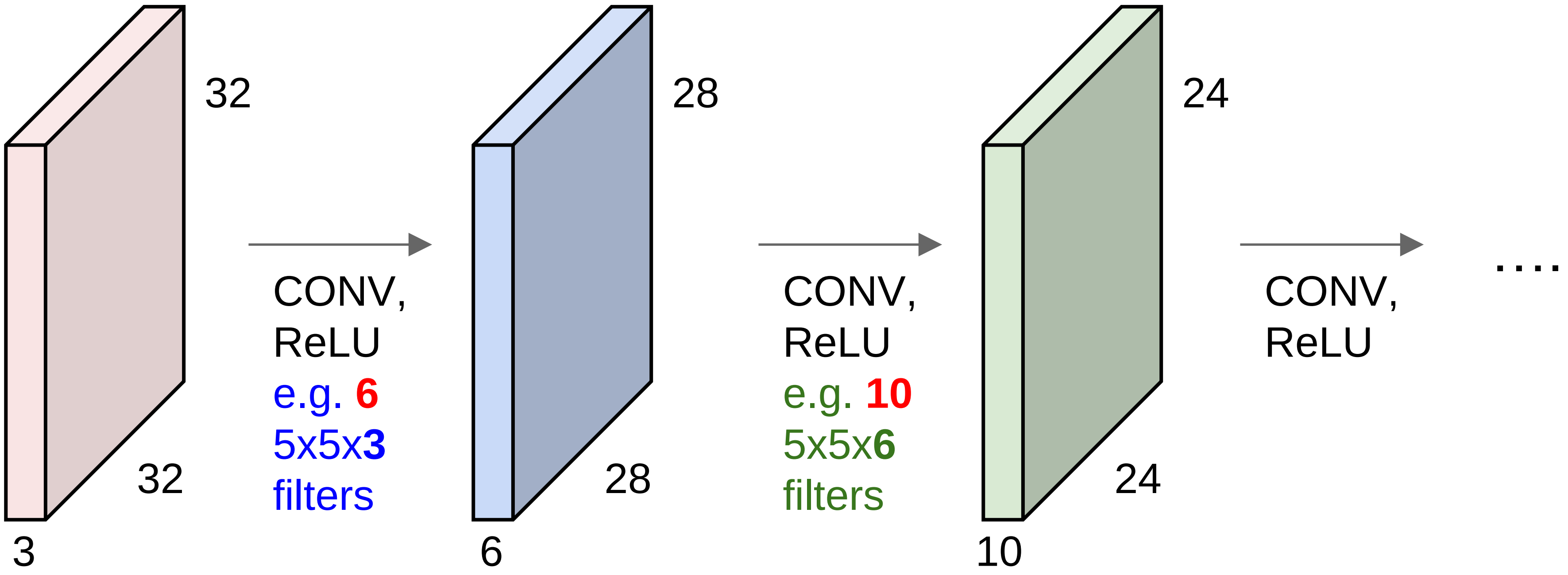


Convolution Layer

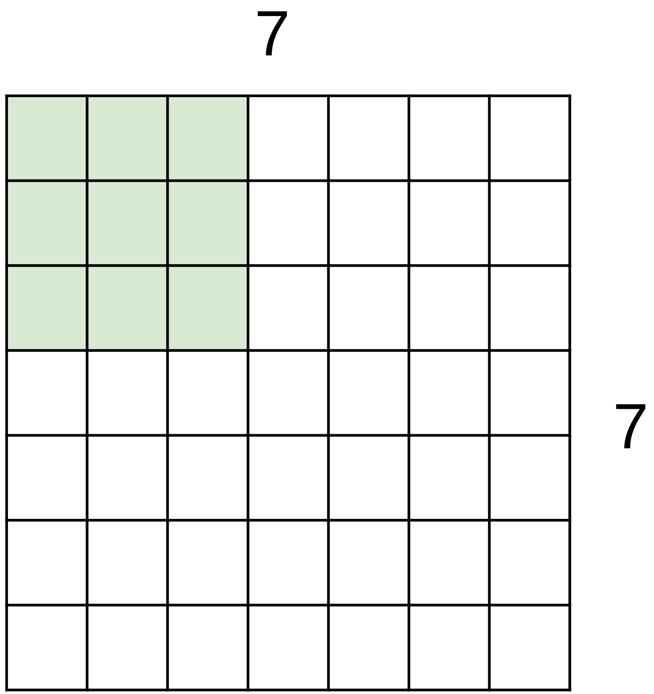


We stack these up to get a “new image” of size 28x28x6!

Convolution Layer

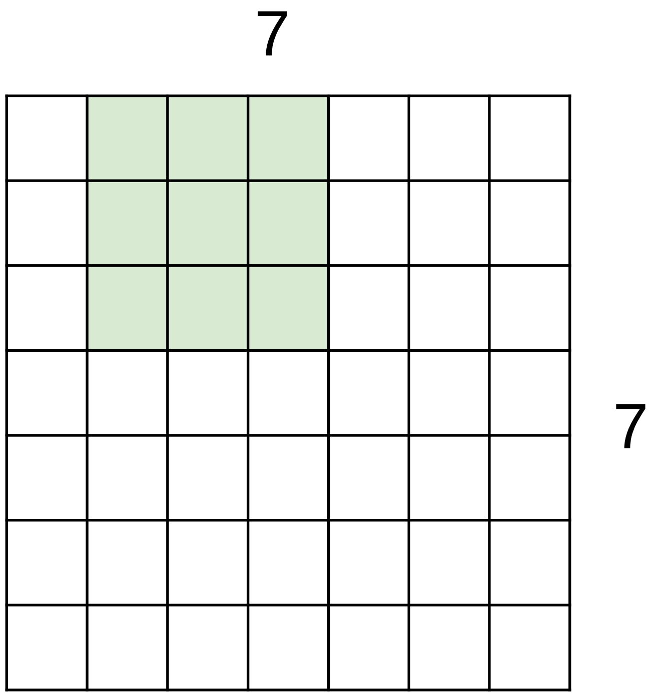


A Closer look at spatial dimension



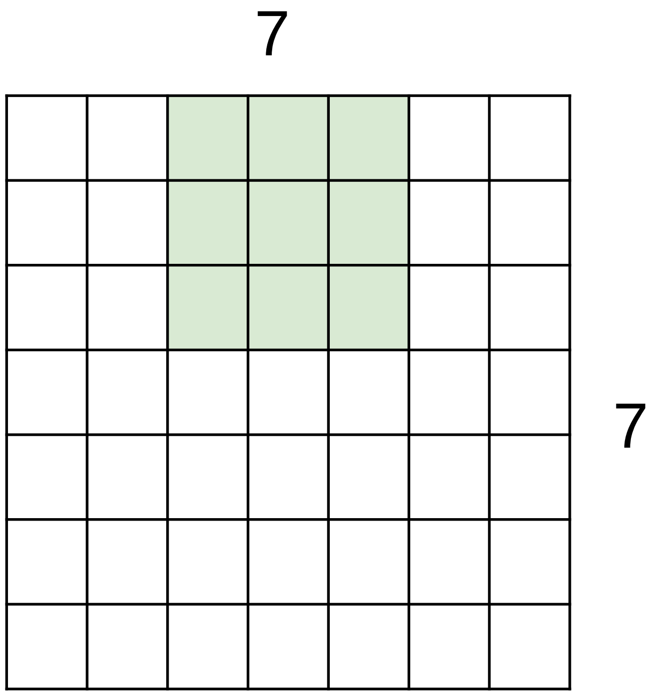
7x7 input (spatially)
assume 3x3 filter

A Closer look at spatial dimension



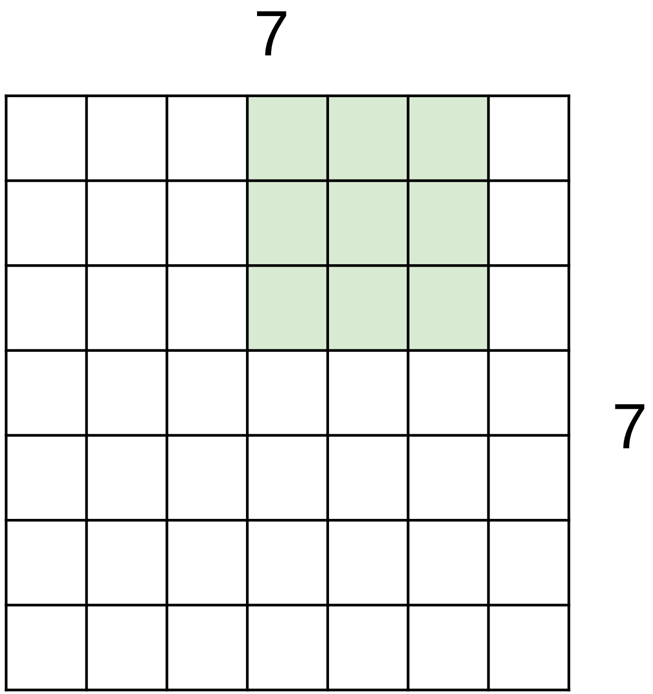
7x7 input (spatially)
assume 3x3 filter

A Closer look at spatial dimension



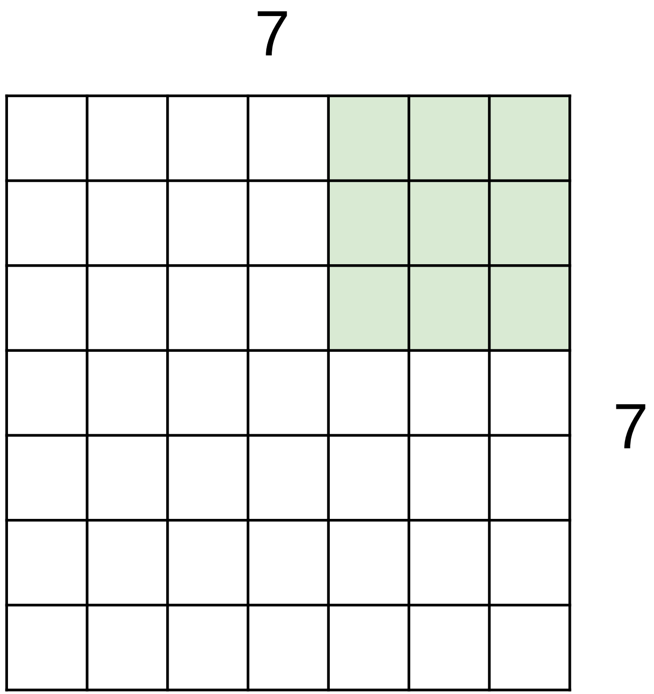
7x7 input (spatially)
assume 3x3 filter

A Closer look at spatial dimension



7x7 input (spatially)
assume 3x3 filter

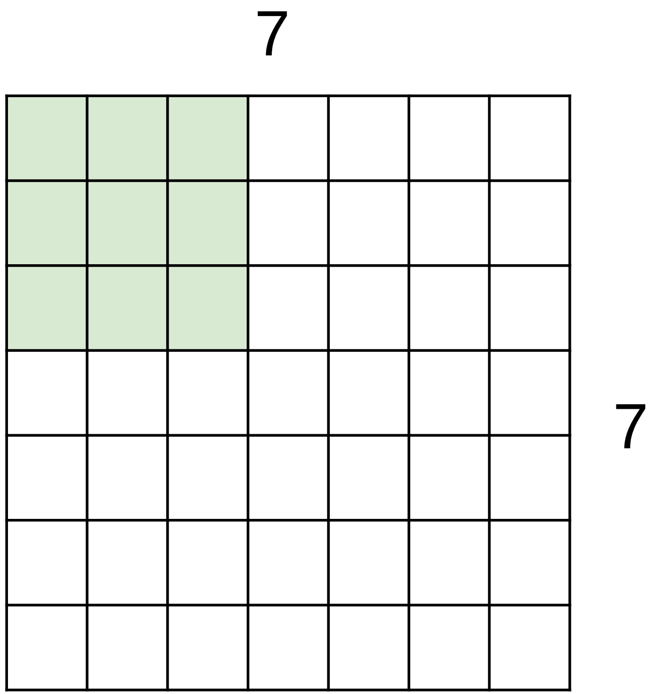
A Closer look at spatial dimension



7x7 input (spatially)
assume 3x3 filter

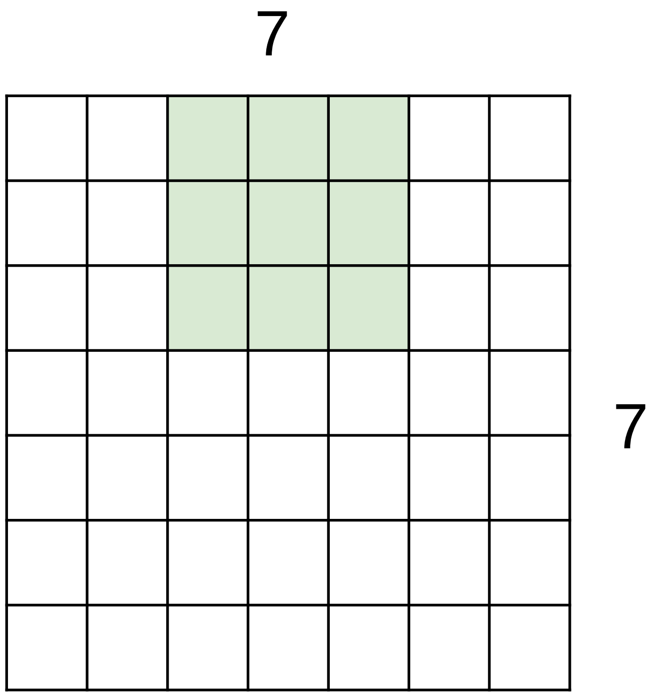
➔ **5x5 output**

A Closer look at spatial dimension



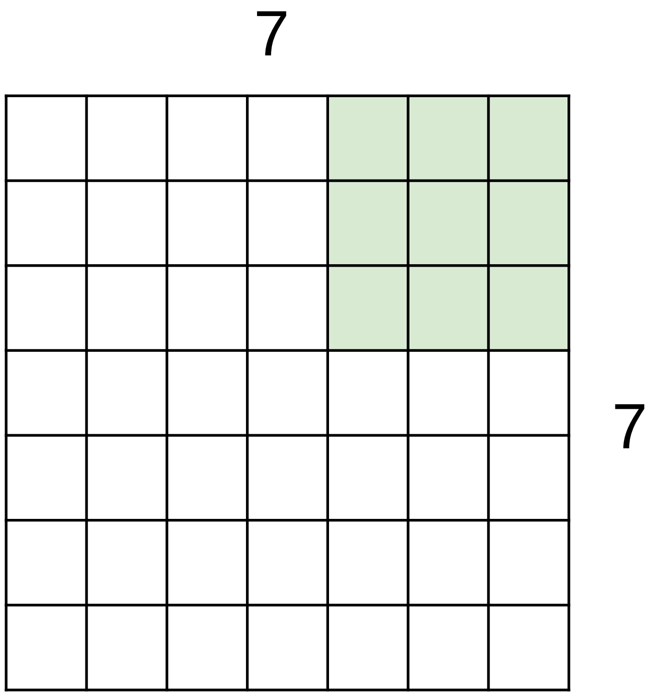
7x7 input (spatially)
assume 3x3 filter
applied with **stride 2**

A Closer look at spatial dimension



7x7 input (spatially)
assume 3x3 filter
applied with **stride 2**

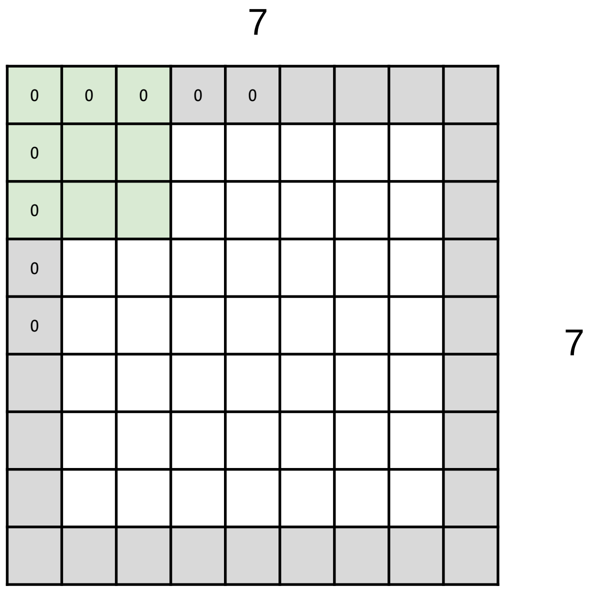
A Closer look at spatial dimension



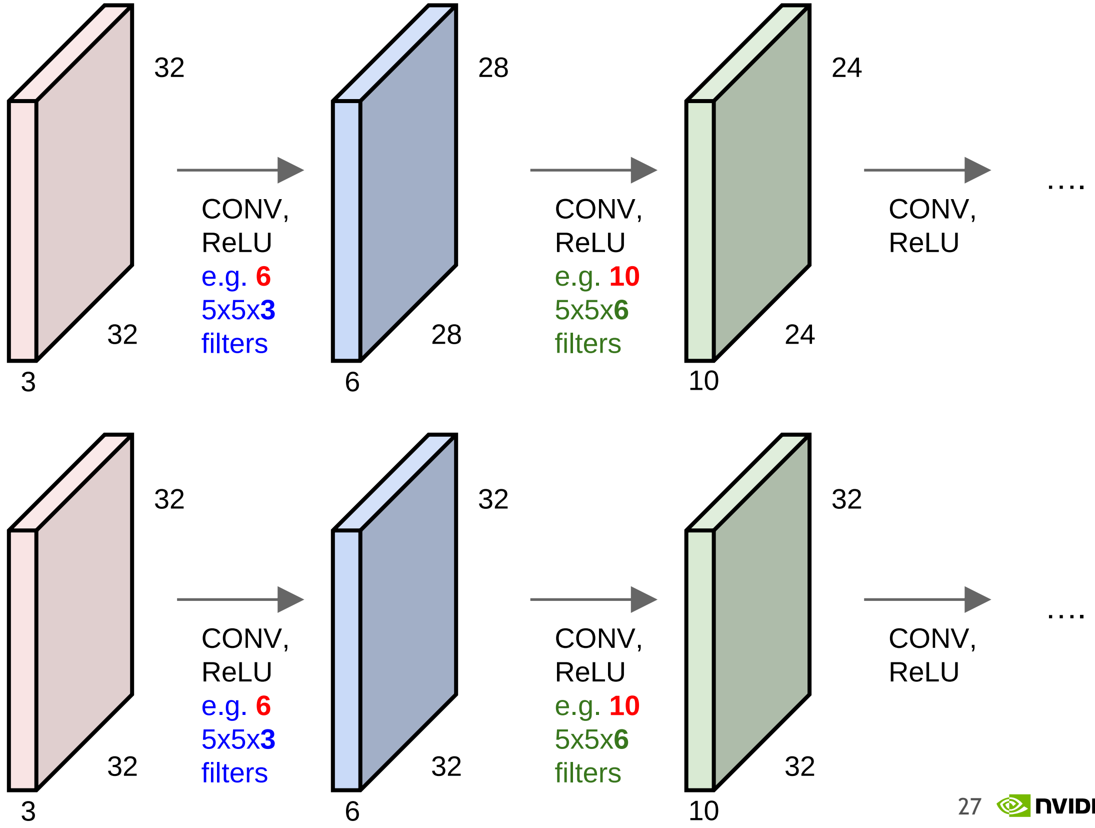
7x7 input (spatially)
assume 3x3 filter
applied with **stride 2**
➔ **3x3 output**

Zero Padding

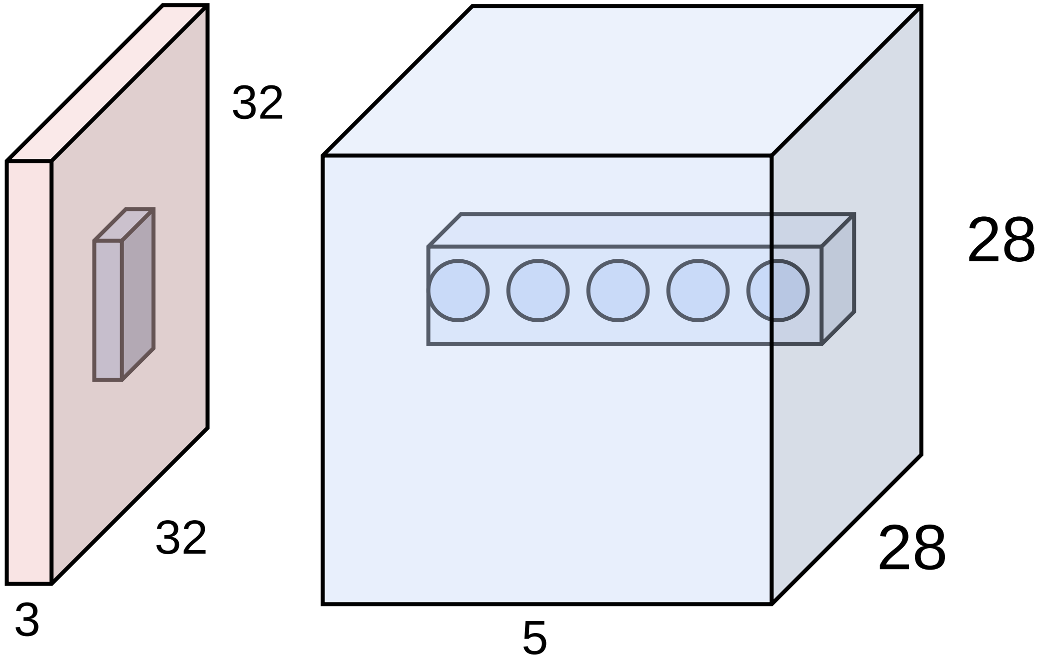
To avoid spatial shrinking by convolution

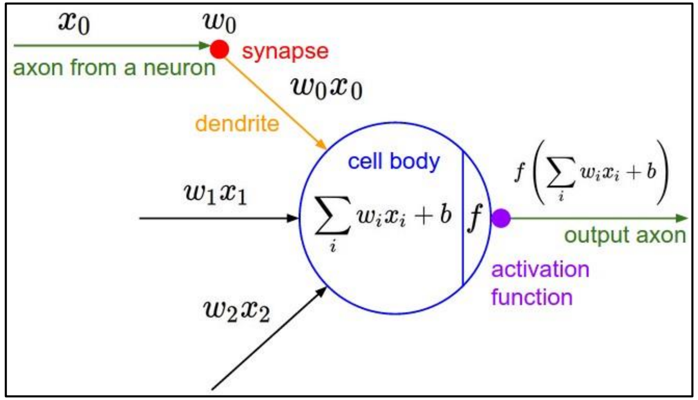


$$Output\ Size = \frac{input\ size + 2 \times pad\ size - filter\ size}{stride\ size} + 1$$



Convolution Layer as a neuron





Another example

This time, a tabular representation

Overfeat, 2014

Layer	1	2	3	4	5	6	7	8	Output 9
Stage	conv + max	conv + max	conv	conv	conv	conv + max	full	full	full
# channels	96	256	512	512	1024	1024	4096	4096	1000
Filter size	7x7	7x7	3x3	3x3	3x3	3x3	-	-	-
Conv. stride	2x2	1x1	1x1	1x1	1x1	1x1	-	-	-
Pooling size	3x3	2x2	-	-	-	3x3	-	-	-
Pooling stride	3x3	2x2	-	-	-	3x3	-	-	-
Zero-Padding size	-	-	1x1x1x1	1x1x1x1	1x1x1x1	1x1x1x1	-	-	-
Spatial input size	221x221	36x36	15x15	15x15	15x15	15x15	5x5	1x1	1x1

Another example

This time, a tabular representation

Overfeat, 2014

Layer	1	2	3	4	5	6	7	8	Output 9
Stage	conv + max	conv + max	conv	conv	conv	conv + max	full	full	full
# channels	96	256	512	512	1024	1024	4096	4096	1000
Filter size	7x7	7x7	3x3	3x3	3x3	3x3	-	-	-
Conv. stride	2x2	1x1	1x1	1x1	1x1	1x1	-	-	-
Pooling size	3x3	2x2	-	-	-	3x3	-	-	-
Pooling stride	3x3	2x2	-	-	-	3x3	-	-	-
Zero-Padding size	-	-	1x1x1x1	1x1x1x1	1x1x1x1	1x1x1x1	-	-	-
Spatial input size	221x221	36x36	15x15	15x15	15x15	15x15	5x5	1x1	1x1

Convolutional Layers (Feature extractor)

Another example

This time, a tabular representation

Overfeat, 2014

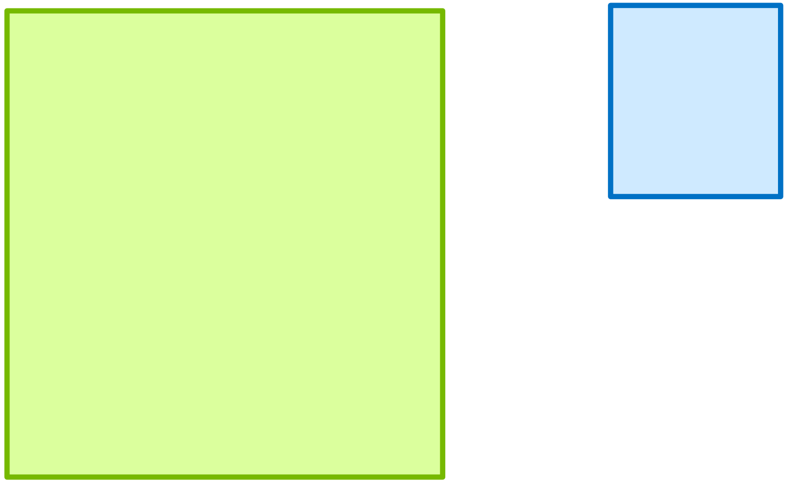
Layer	1	2	3	4	5	6	7	8	Output 9
Stage	conv + max	conv + max	conv	conv	conv	conv + max	full	full	full
# channels	96	256	512	512	1024	1024	4096	4096	1000
Filter size	7x7	7x7	3x3	3x3	3x3	3x3	-	-	-
Conv. stride	2x2	1x1	1x1	1x1	1x1	1x1	-	-	-
Pooling size	3x3	2x2	-	-	-	3x3	-	-	-
Pooling stride	3x3	2x2	-	-	-	3x3	-	-	-
Zero-Padding size	-	-	1x1x1x1	1x1x1x1	1x1x1x1	1x1x1x1	-	-	-
Spatial input size	221x221	36x36	15x15	15x15	15x15	15x15	5x5	1x1	1x1

Convolutional Layers (Feature extractor)

Fully connected Layers (Classifier)

Convolution Operation

Convolution Operation

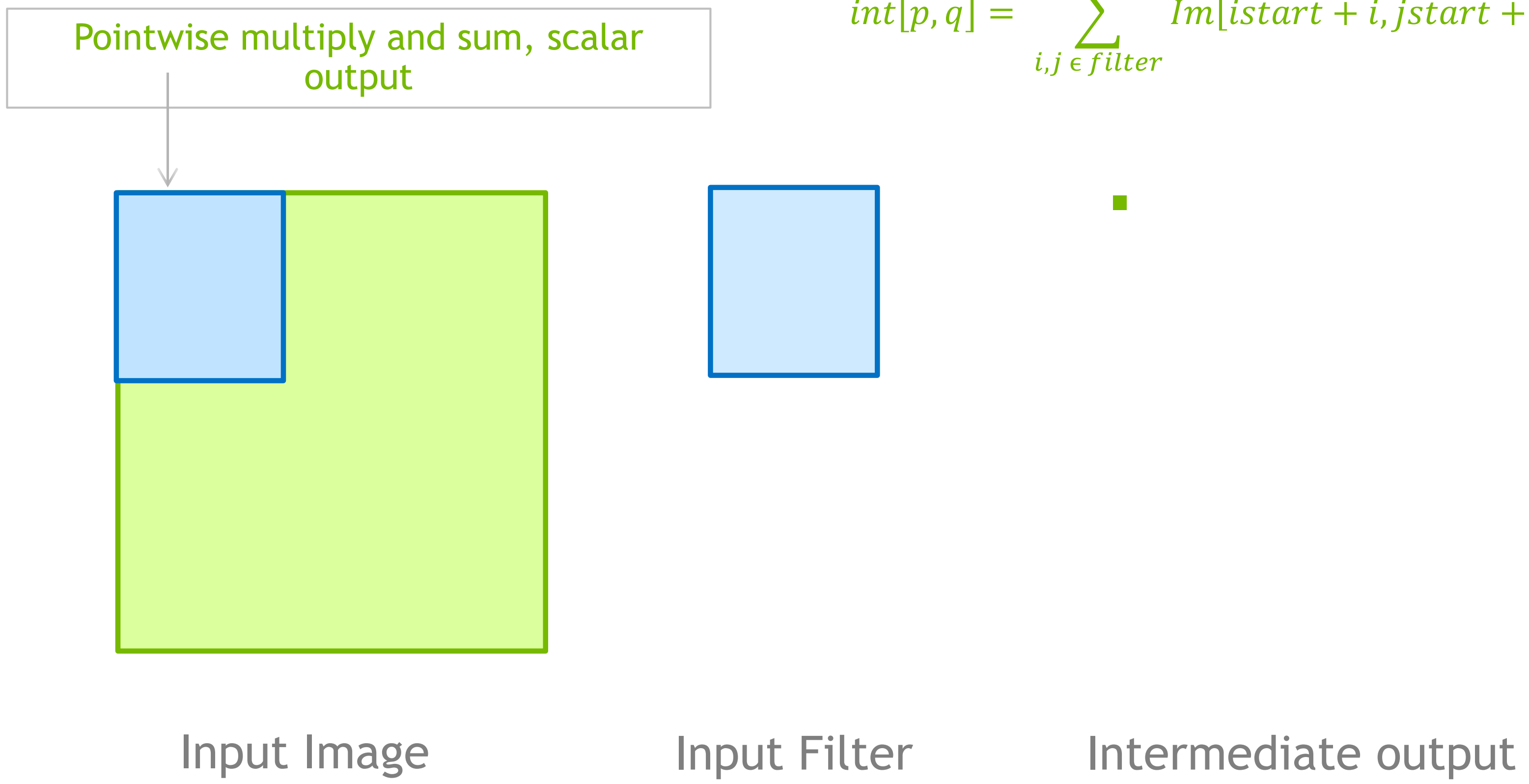


Input Image

Input Filter

Convolution Operation

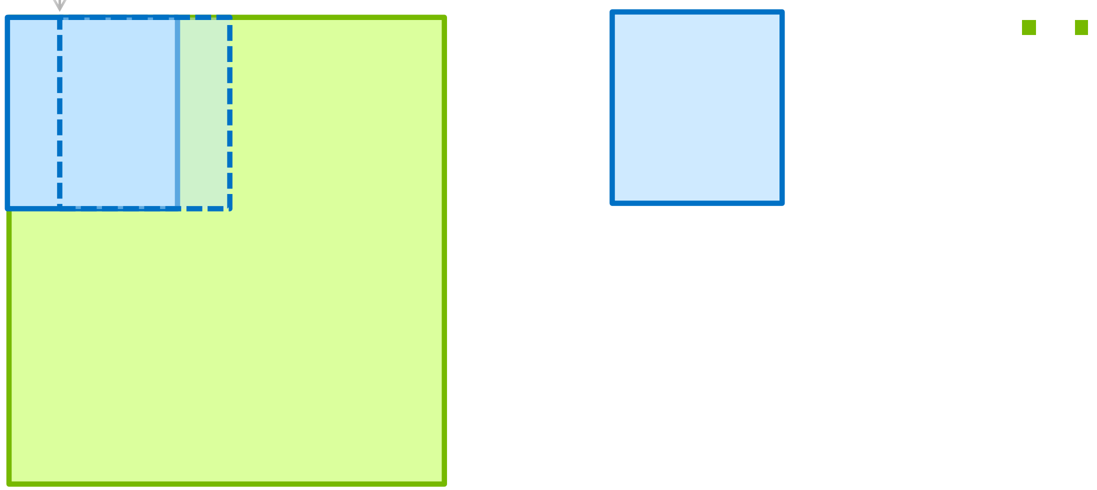
$$int[p,q] = \sum_{i,j\,\epsilon\, filter} Im[istart+i, jstart+j]\,.\,Filt[i,j]$$



Convolution Operation

$$int[p,q] = \sum_{i,j\,\epsilon\,filter} Im[istart + i, jstart + j]\,.\,Filt[i,j]$$

Pointwise multiply and sum, scalar output



Input Image

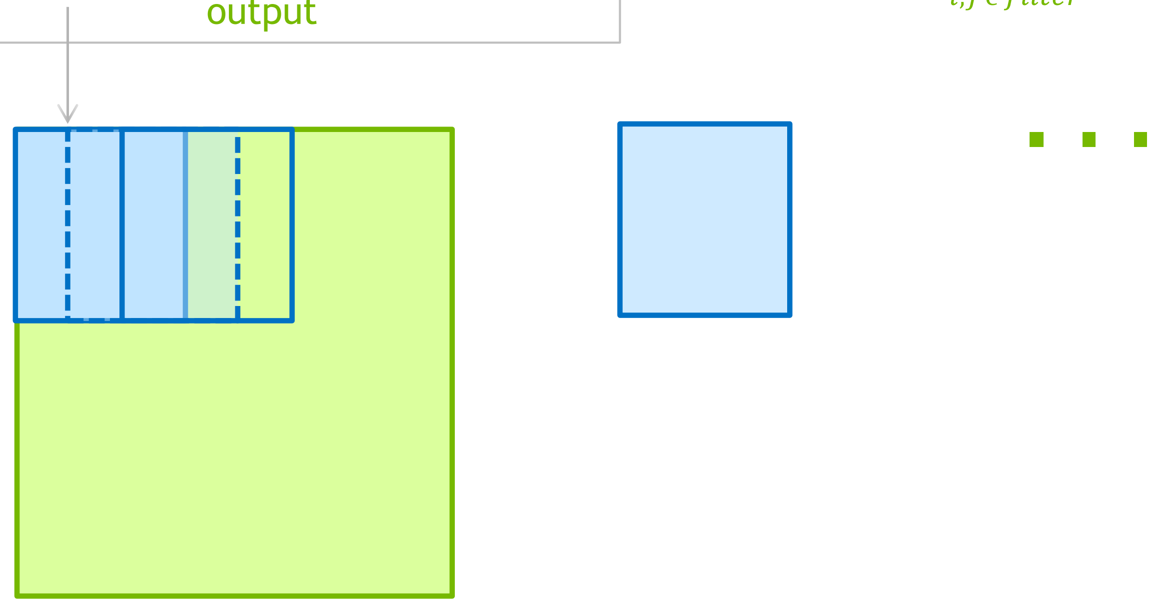
Input Filter

Intermediate output

Convolution Operation

$$int[p,q] = \sum_{i,j \in filter} Im[istart + i, jstart + j] \,.\, Filt[i,j]$$

Pointwise multiply and sum, scalar output



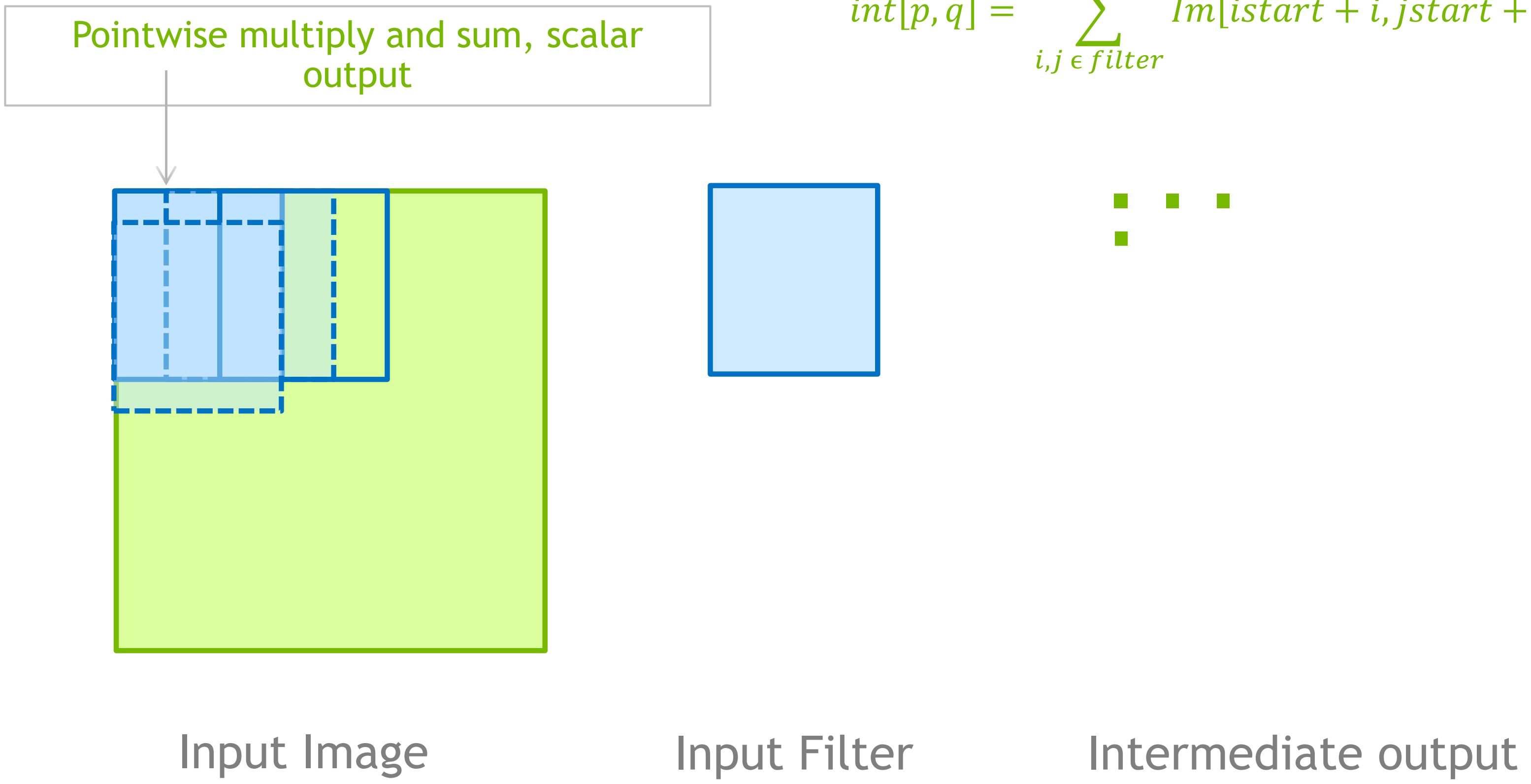
Input Image

Input Filter

Intermediate output

Convolution Operation

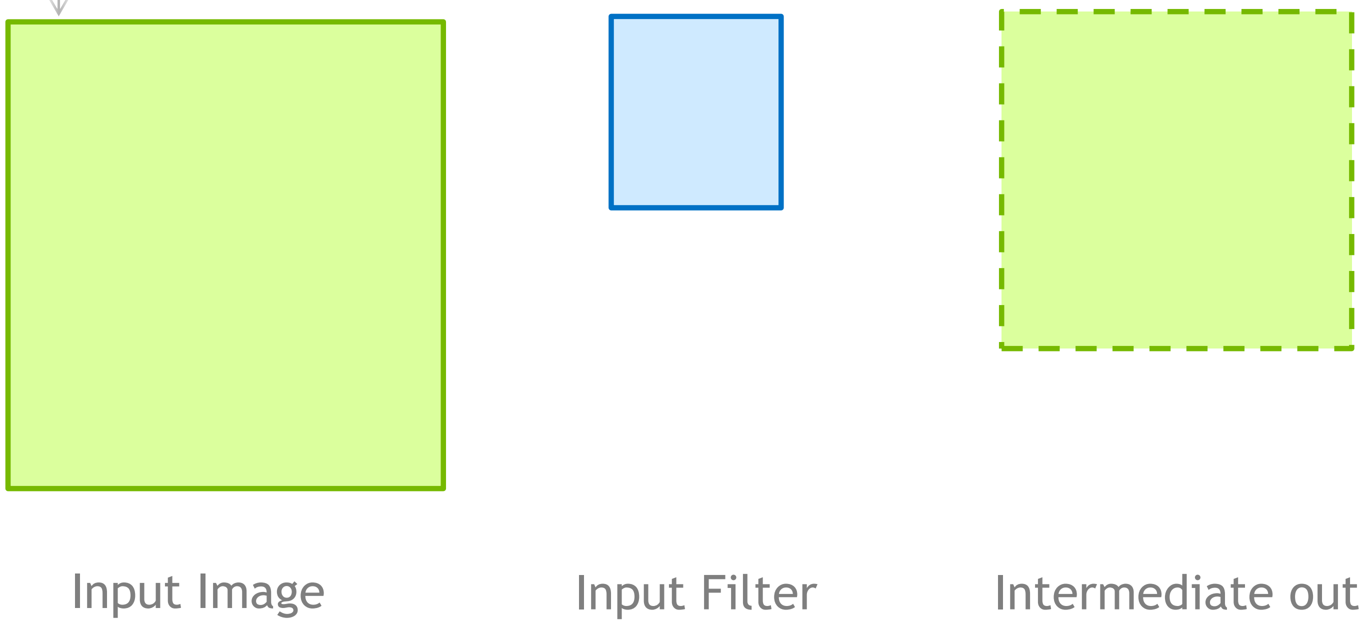
$$int[p,q] = \sum_{i,j \,\epsilon\, filter} Im[istart + i, jstart + j] \,.\, Filt[i,j]$$



Convolution Operation

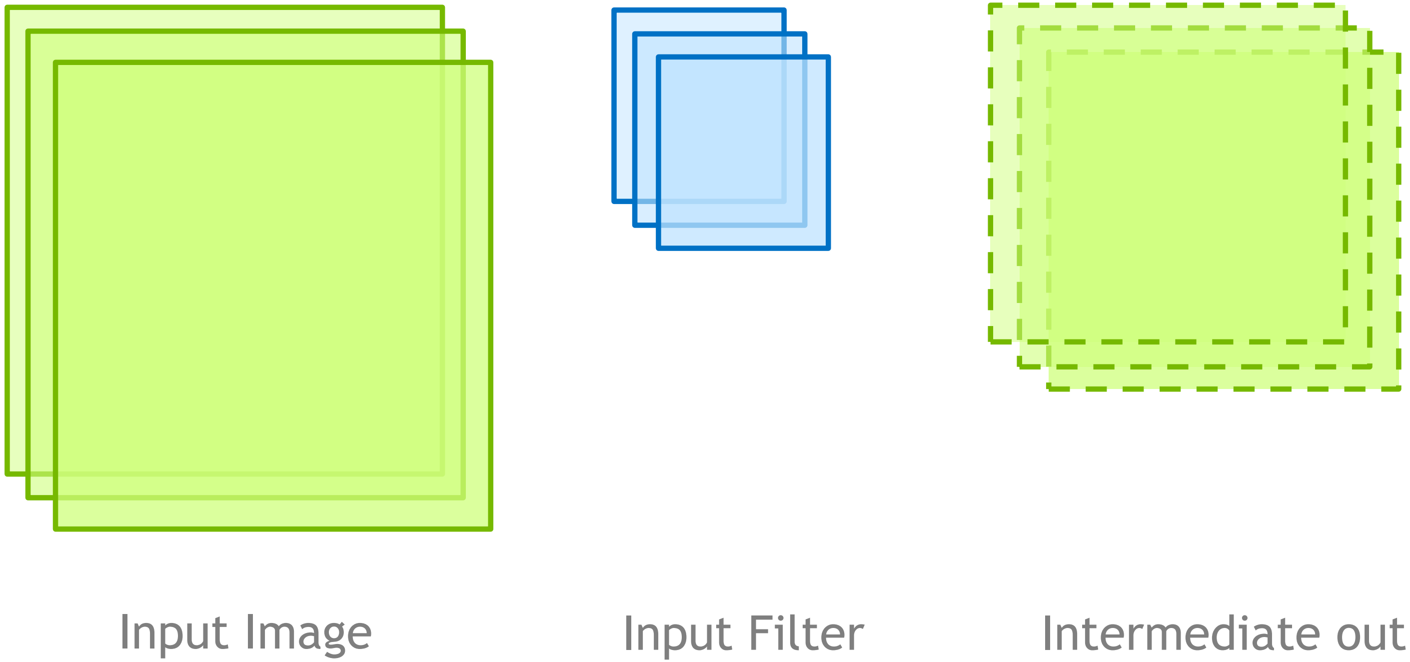
$$int[p,q] = \sum_{i,j\,\epsilon\,filter} Im[istart + i, jstart + j]\,.\,Filt[i,j]$$

Pointwise multiply and sum, scalar output



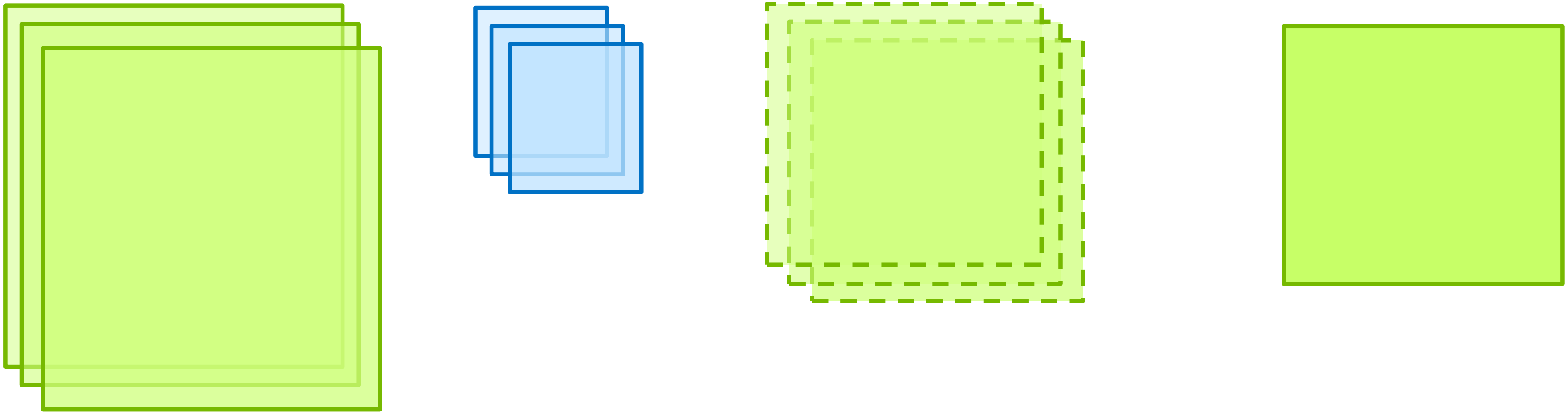
Convolution Operation

$$int[c,p,q] = \sum_{i,j \,\epsilon\, filter} Im[c][istart+i, jstart+j] \,.\, Filt[c][i,j]$$



Convolution Operation

$$int[c,p,q] = \sum_{i,j \in filter} Im[c][istart + i, jstart + j] \,.\, Filt[c][i,j]$$



Input Image

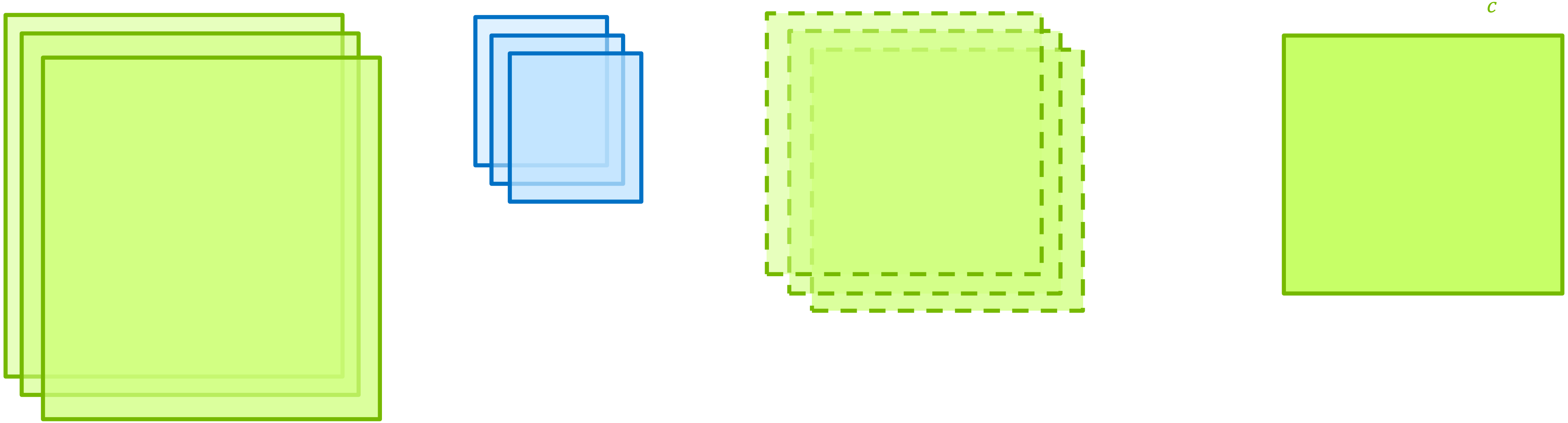
Input Filter

Intermediate output

Convolution Operation

$$int[c,p,q] = \sum_{i,j\,\epsilon\,filter} Im[c][istart+i, jstart+j]\,.\,Filt[c][i,j]$$

$$output[p,q] = \sum_{c} int[c,p,q]$$



Input Image

Input Filter

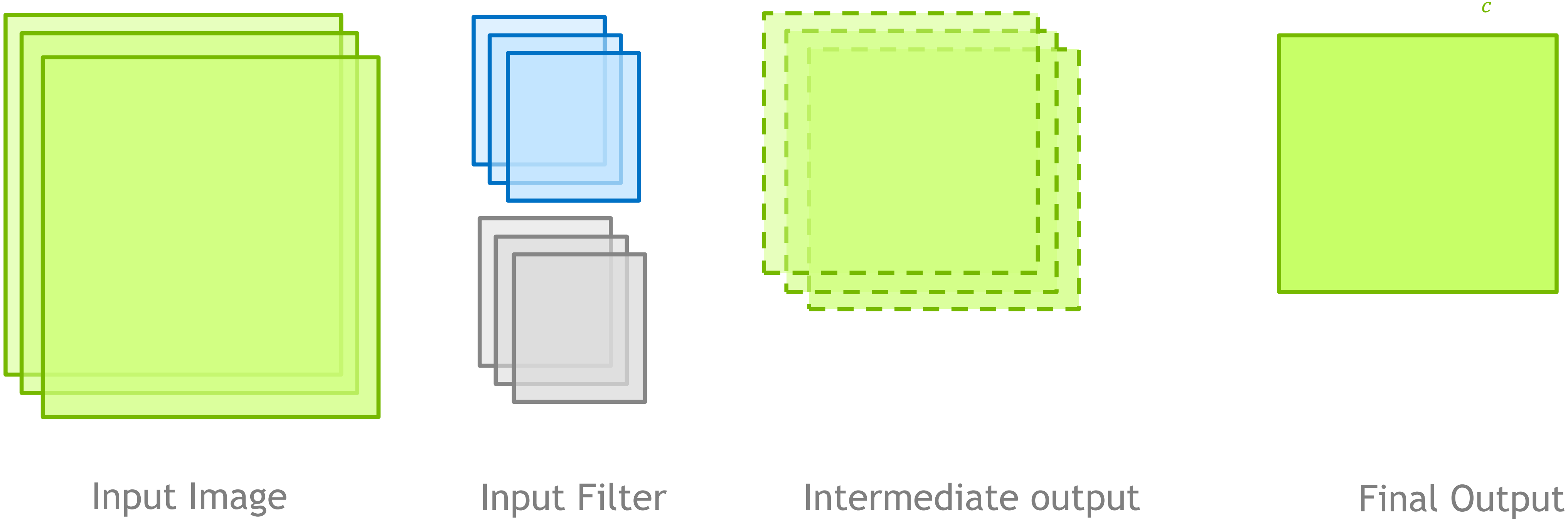
Intermediate output

Final Output

Convolution Operation

$$int[c,p,q] = \sum_{i,j \, \epsilon \, filter} Im[c][istart+i, jstart+j] \, . \, Filt[c][i,j]$$

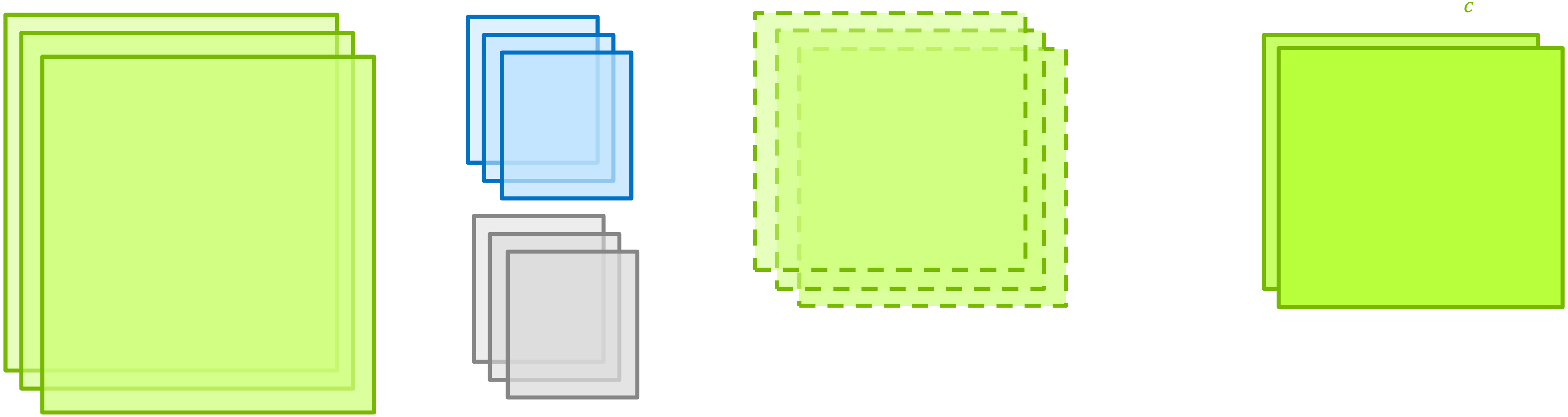
$$output[p,q] = \sum_{c} int[c,p,q]$$



Convolution Operation

$$int[c,p,q] = \sum_{i,j \,\epsilon\, filter} Im[c][istart+i, jstart+j]\,.\,Filt[c][i,j]$$

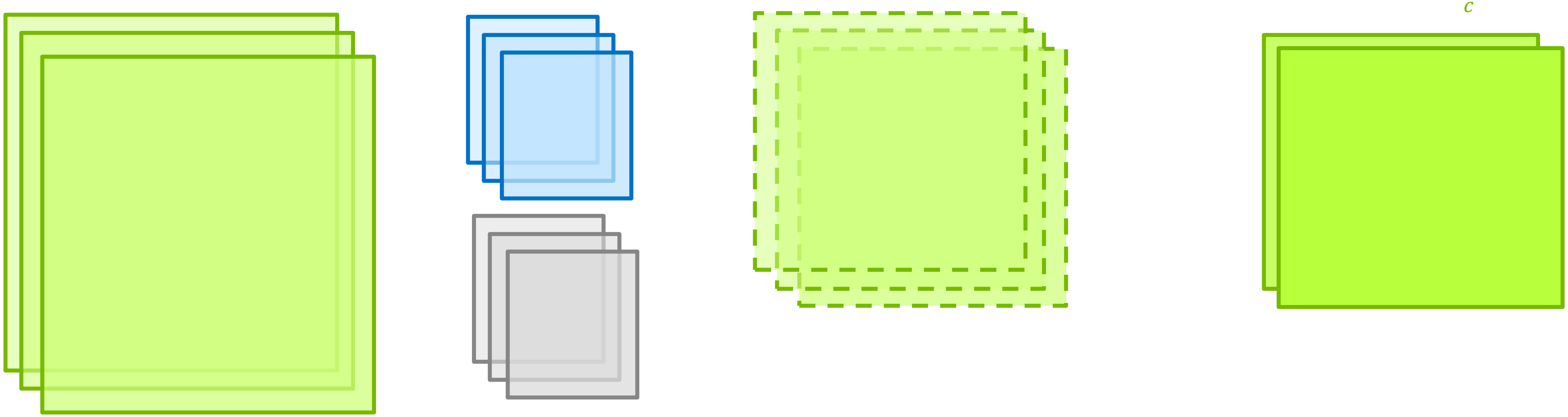
$$output[p,q] = \sum_{c} int[c,p,q]$$



Convolution Operation

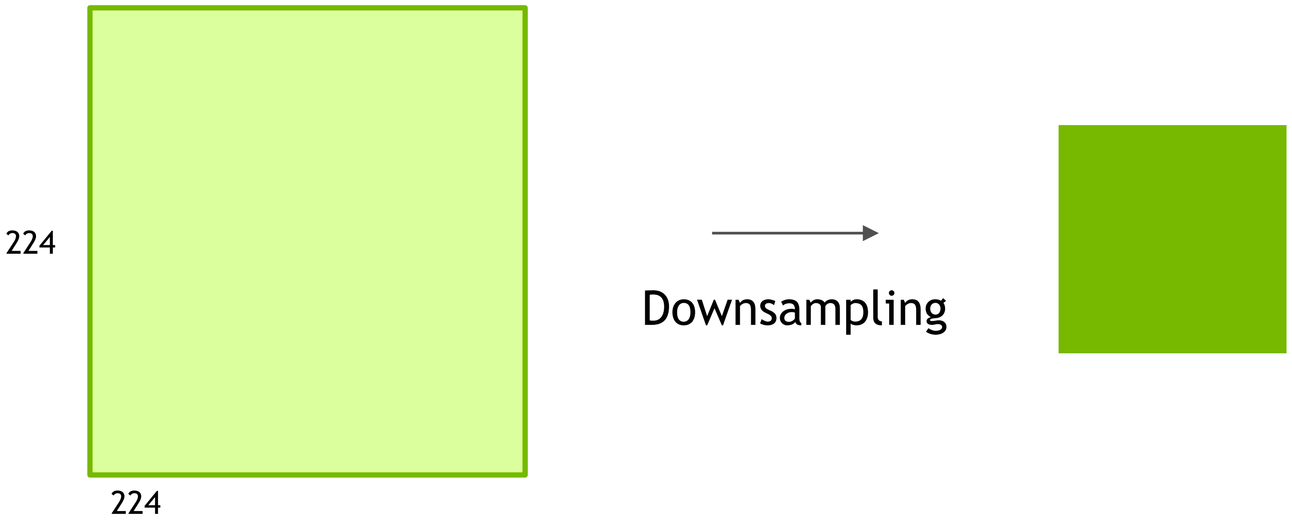
$$int[c,p,q] = \sum_{i,j\,\epsilon\,filter} Im[c][istart + i, jstart + j]\,.\,Filt[c][i,j]$$

$$output[p,q] = \sum_{c} int[c,p,q]$$

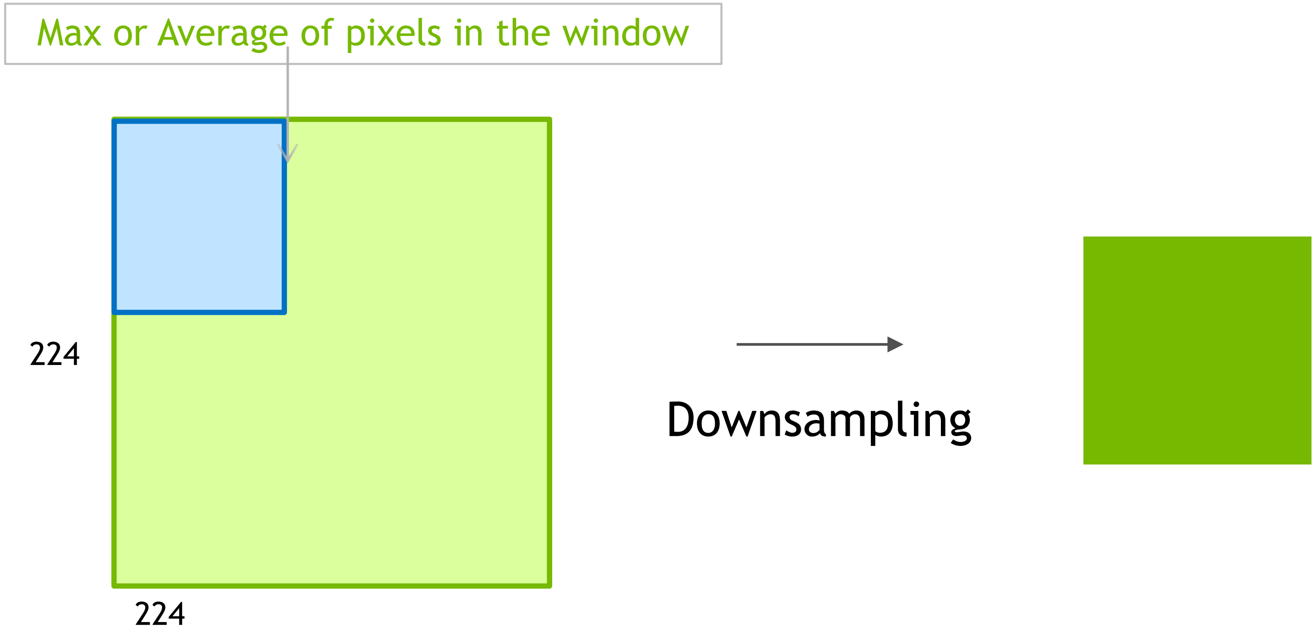


Why do it once if you can do it n times ? Batch the whole thing.

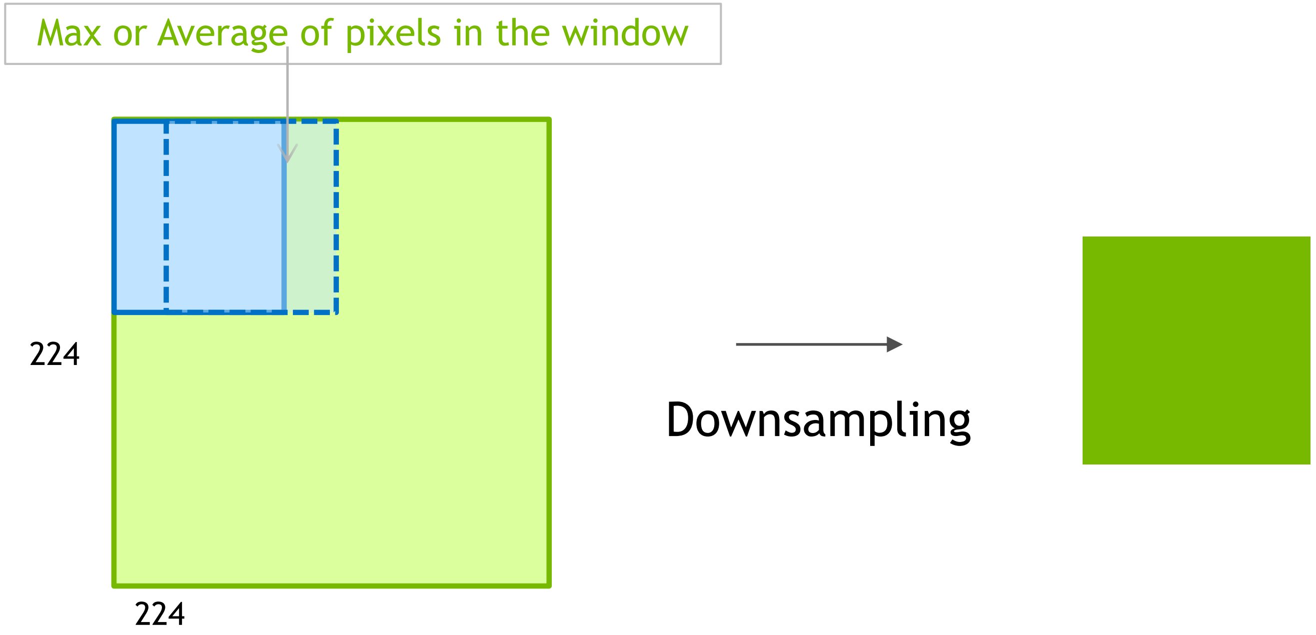
Pooling



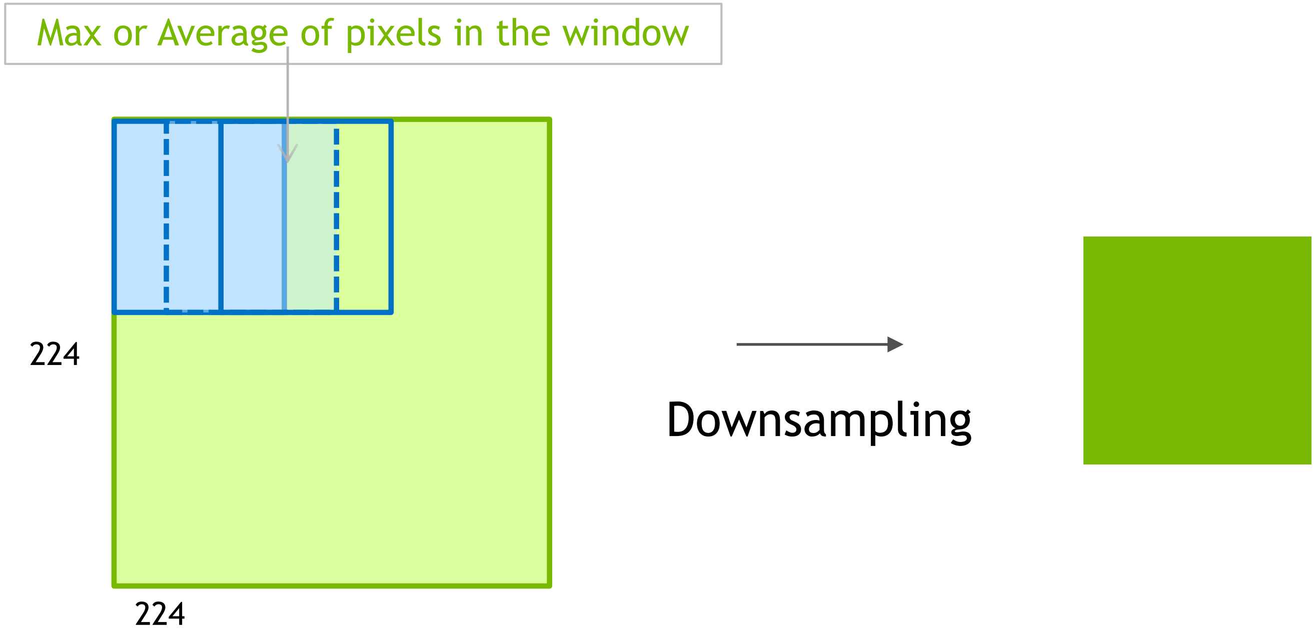
Pooling



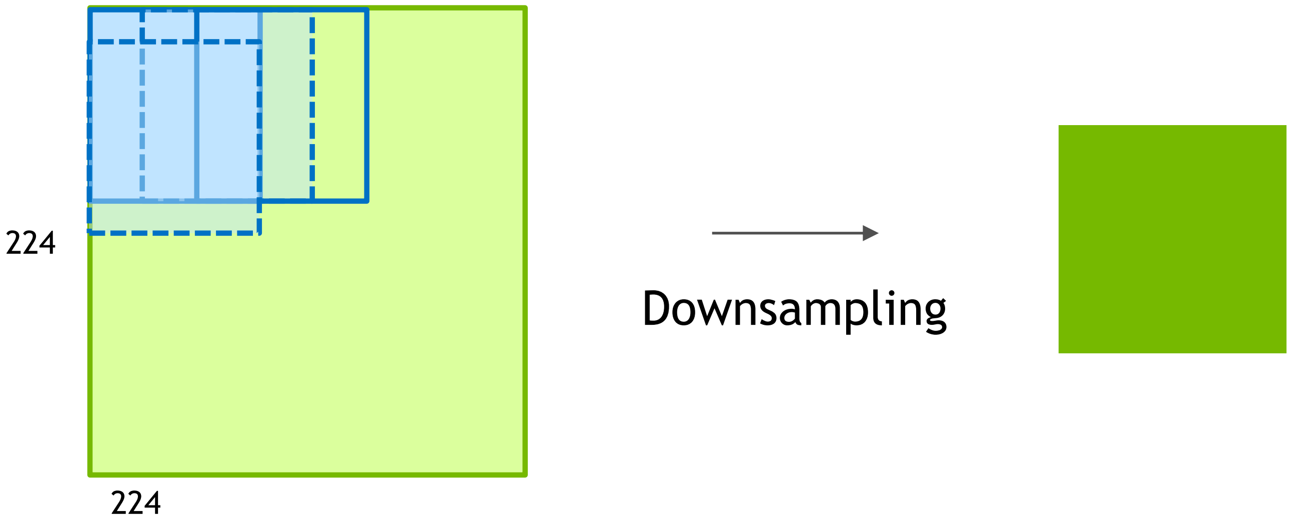
Pooling



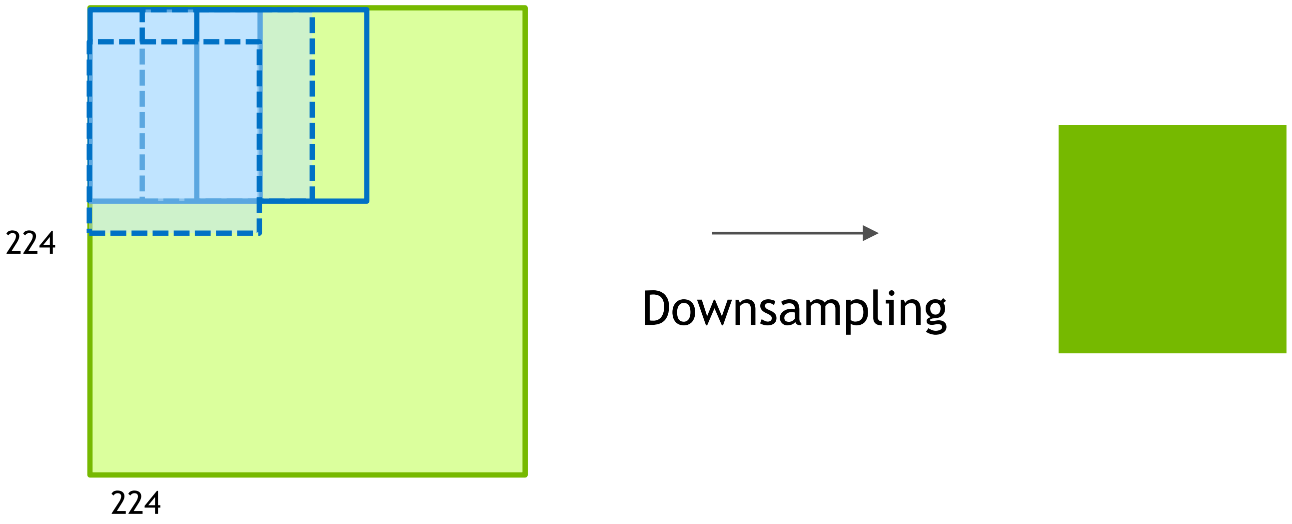
Pooling



Pooling



Pooling



Pooling accomplishes 2 things:

1. Acts as a smoother, reducing the number of small, high-frequency variations
2. Increases effective receptive field by down sampling the image at each step

- May be counterproductive when pixel accurate locations are required

Examples of Pooling

For single depth slice:

pool with
2x2 filters and stride 2

Common Filter Size:
2x2/3x3
Common Stride Size:
2

Max pooling

1	1	2	4
5	6	7	8
3	2	1	0
1	2	3	4

→

6	8
3	4

Mean pooling

1	1	2	4
5	6	7	8
3	2	1	0
1	2	3	4

→

3	5
2	2

Stochastic pooling

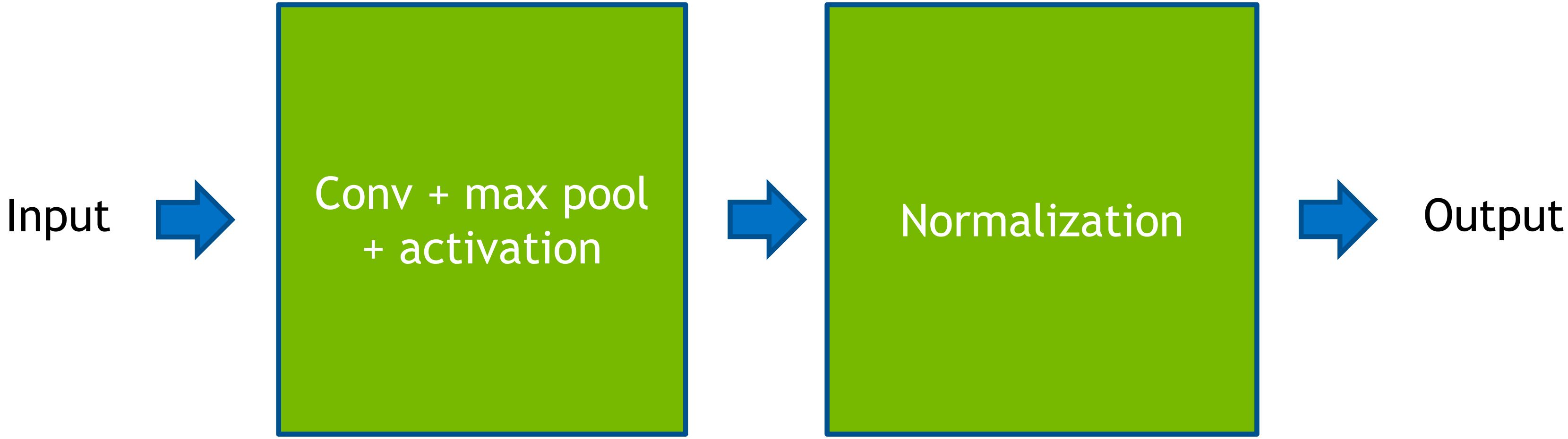
1	1	2	4
5	6	7	8
3	2	1	0
1	2	3	4

→

5	4
2	0

Optional layers

Local Response Normalization



Local Response Normalization

Alexnet 2012

$$b^i = a^i / \sqrt{\sum (a^i)^2}$$

is the channel index of the input image

- $ange$ is some hard-coded subset of channels to normalize over
- , β and k are also predetermined parameters
- $= 2,\ range = 5, \alpha = 10^{-4}$ and $\beta = 0.75$

Local Response Normalization

Alexnet 2012

$$b^{i}{}_{(x,y)} = a^{i}{}_{(x,y)} / (k + \alpha \sum_{\substack{j\,\epsilon \\ range}} a^{j}{}_{(x,y)}{}^{2})^{\beta} \qquad b^{i} = a^{i} / \sqrt{\sum (a^{i})^{2}}$$

- $57.0\beta\beta = \ dna\, 10\ \ - 4\ \ 4\, 10\ \ - 4 - 0\ 10\ \ - 4\ 1\alpha\alpha = , 5rraanngg ee = \ , 2i$ is the channel index of the input image
- $range$ is some hard-coded subset of channels to normalize over
- $\alpha,\ \beta$ and k are also predetermined parameters
- Motivation is to tone down the effect of “rogue” extreme hot-spots

Other normalization routines

Less frequent, but have been used effectively

Local Contrast Normalization :

- Operates within a feature map
- Look at a neighborhood of pixels centered at point-of-interest in (x,y) space
- Compute Gaussian-weighted mean, and variance of these pixels
- Normalize at point-of-interest $a_{norm} = (a_{in} - \mu)/\sigma$

Batch Normalization:

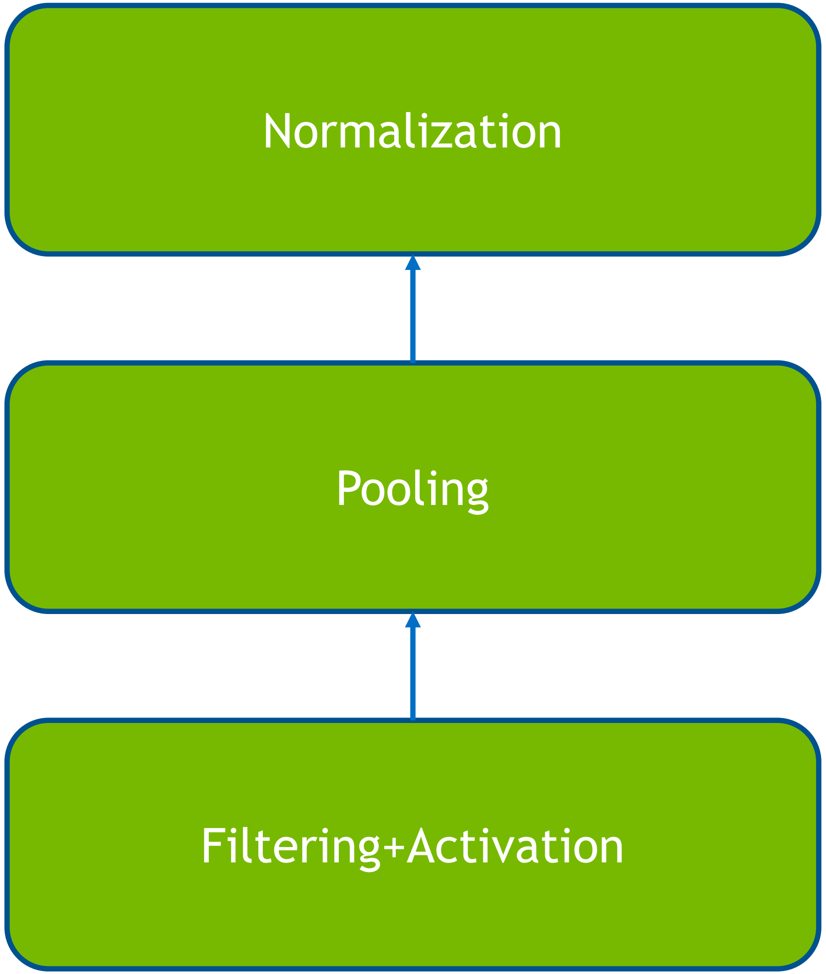
- Show remarkable improvement in convergence
- Normalize across whole batch of images, rather than on a per-image basis

Batch Normalization

Motivation

- From the perspective of a layer i, the distribution of inputs in each iteration of the training process vary wildly
  - Think of a histogram of raw pixel values present in each batch of inputs to the layer
- This is because, at each iteration, the layers *before* it are being modified, possibly significantly
  - Even with the simplifying assumption that the inputs to the network as a whole exhibit a somewhat constant distribution across all minibatches
- B.N. tries to normalize the distribution of inputs to a layer, to make the training process easier

CNN parameters you have to choose



- Normalization kernel – typical: gaussian with mean=0, std=1, over 7x7 pixels
- Number of kernels to normalize over – typical: 5

- Pooling ratio – typical value: (2,2)
- Pooling function – max, sum, avg – max is most common

- Number of filters – typical values: 32, 64, 128
- Filter size – typical values: 3x3, 4x4, 5x5 pixels
- Step size
- Activation function – use ReLU at the moment

Loss functions

Quantify “wrongness”

- Measures model quality : Lower the loss, higher the accuracy
- The term “loss function” comes from the field of Optimization
    - Conventional problem statement : “Minimize loss subject to constraints”
- Corrections in model during training originate with this value
- A conscious design choice, have significant impact on the learned model
- Euclidean loss is a familiar option : $L(y, \hat{y}) = \frac{1}{k} \sum (y_n - \widehat{y_n})^2$
- More exotic variants exist, and are the subject of another session

Choosing an architecture

AKA black magic

- Start with one of the “standard” architectures with similar input data properties
- Modify the fully-connected layers according to output objective
- Modify layer sizes until you overfit – then control overfitting (more later)

Describing the network

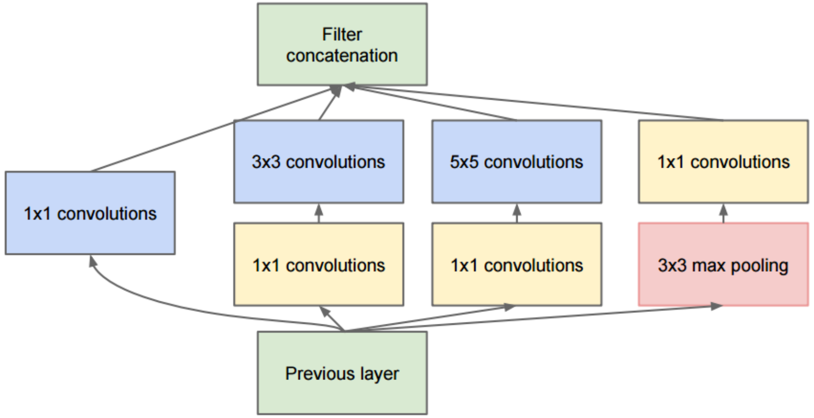
Caffe protobuf files

```
layer {
  name: "conv1"
  type: "Convolution"
  bottom: "data"
  top: "conv1"
  # learning rate and decay multipliers for the filters
  param { lr_mult: 1 decay_mult: 1 }
  # learning rate and decay multipliers for the biases
  param { lr_mult: 2 decay_mult: 0 }
  convolution_param {
    num_output: 96     # learn 96 filters
    kernel_size: 11    # each filter is 11x11
    stride: 4          # step 4 pixels between each filter application
    weight_filler {
      type: "gaussian" # initialize the filters from a Gaussian
      std: 0.01        # distribution with stdev 0.01 (default mean: 0)
    }
    bias_filler {
      type: "constant" # initialize the biases to zero (0)
      value: 0
    }
  }
}
```

CNN architectures: GoogleNet

Introduced multi-scale convolutional layers called “Inception” layers

type	patch size/ stride	output size	depth	#1×1	#3×3 reduce	#3×3	#5×5 reduce	#5×5	pool proj	params	ops
convolution	7×7/2	112×112×64	1							2.7K	34M
max pool	3×3/2	56×56×64	0								
convolution	3×3/1	56×56×192	2		64	192				112K	360M
max pool	3×3/2	28×28×192	0								
inception (3a)		28×28×256	2	64	96	128	16	32	32	159K	128M
inception (3b)		28×28×480	2	128	128	192	32	96	64	380K	304M
max pool	3×3/2	14×14×480	0								
inception (4a)		14×14×512	2	192	96	208	16	48	64	364K	73M
inception (4b)		14×14×512	2	160	112	224	24	64	64	437K	88M
inception (4c)		14×14×512	2	128	128	256	24	64	64	463K	100M
inception (4d)		14×14×528	2	112	144	288	32	64	64	580K	119M
inception (4e)		14×14×832	2	256	160	320	32	128	128	840K	170M
max pool	3×3/2	7×7×832	0								
inception (5a)		7×7×832	2	256	160	320	32	128	128	1072K	54M
inception (5b)		7×7×1024	2	384	192	384	48	128	128	1388K	71M
avg pool	7×7/1	1×1×1024	0								
dropout (40%)		1×1×1024	0								
linear		1×1×1000	1							1000K	1M
softmax		1×1×1000	0								



Practice

Image classification for CIFAR-10

NVIDIA