Convolutional Neural Networks

Gain an understanding of the structure and background of CNNs

Gain an in-depth understanding of components and mechanism that makes up a CNN

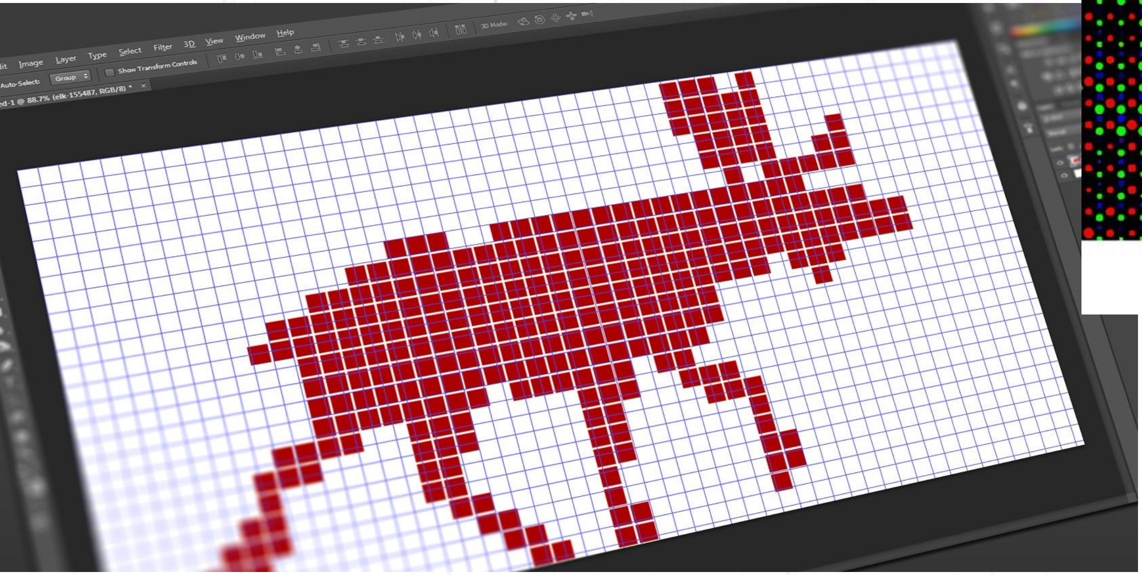
Implement simple Convolutions and Filters with numpy

SLIIT
FACULTY OF COMPUTING

Lecture Content

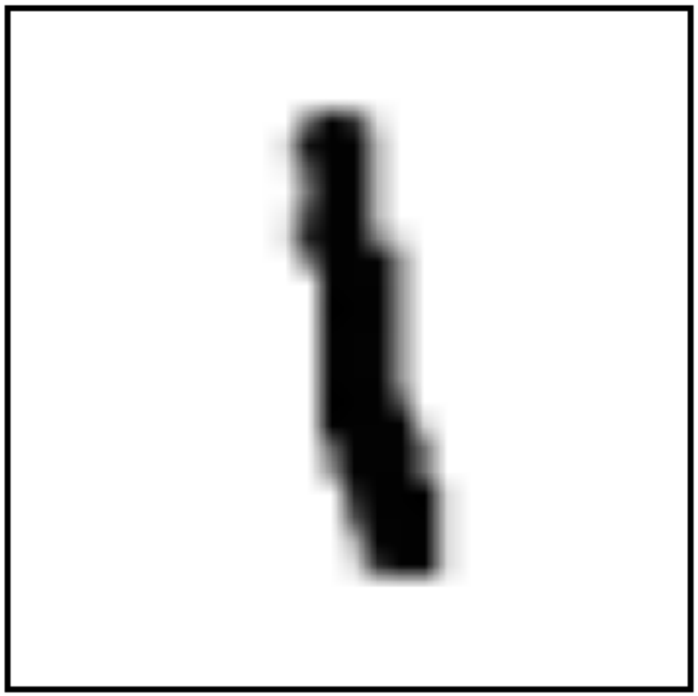
- What is a Convolutional Neural Network
- Structure and Components of a CNN
- Convolutions
- Filters
- Pooling
- Dropout
- Activation Functions
- Next Time: Image Classification

Word of Images



Representing Images with Numbers

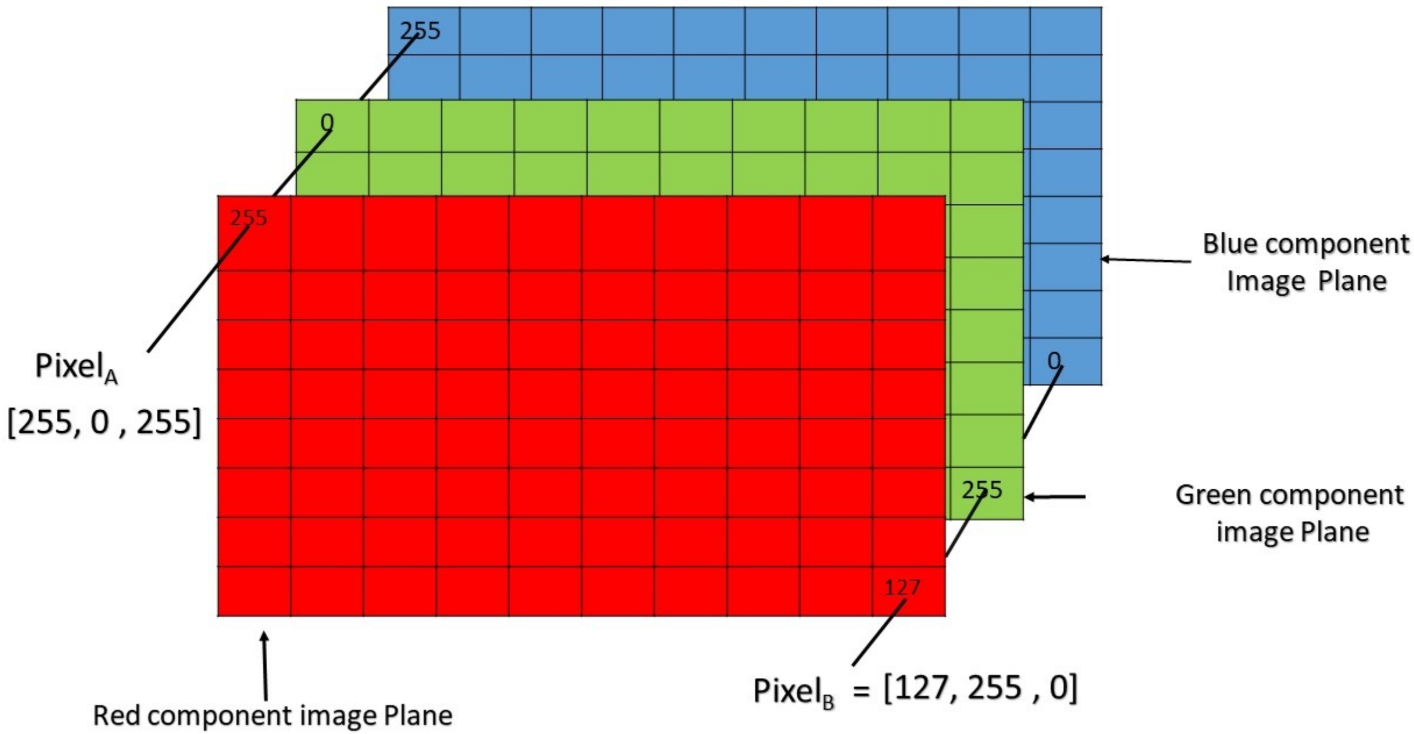
- Image can be represented as an array of pixel values.
- A black & white image is simply a 2D Matrix



$$\simeq \begin{bmatrix} 0&0&0&0&0&0&0&0&0&0&0&0&0&0 \\ 0&0&0&0&0&0&0&0&0&0&0&0&0&0 \\ 0&0&0&0&0&0&.6&.8&0&0&0&0&0&0 \\ 0&0&0&0&0&0&.7&1&0&0&0&0&0&0 \\ 0&0&0&0&0&0&.7&1&0&0&0&0&0&0 \\ 0&0&0&0&0&0&.5&1&.4&0&0&0&0&0 \\ 0&0&0&0&0&0&0&1&.4&0&0&0&0&0 \\ 0&0&0&0&0&0&0&1&.4&0&0&0&0&0 \\ 0&0&0&0&0&0&0&1&.7&0&0&0&0&0 \\ 0&0&0&0&0&0&0&1&1&0&0&0&0&0 \\ 0&0&0&0&0&0&0&.9&1&.1&0&0&0&0 \\ 0&0&0&0&0&0&0&.3&1&.1&0&0&0&0 \\ 0&0&0&0&0&0&0&0&0&0&0&0&0&0 \\ 0&0&0&0&0&0&0&0&0&0&0&0&0&0 \end{bmatrix}$$

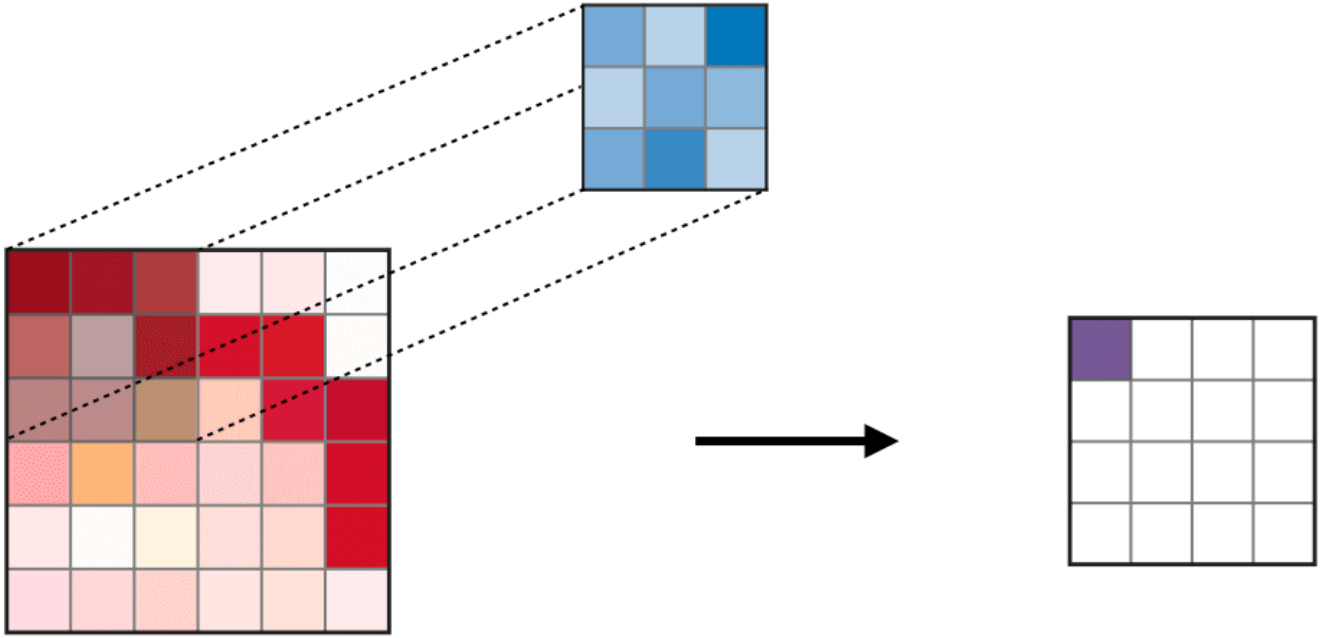
Color Images

- Will need a 3D matrix to represent a RGB image.

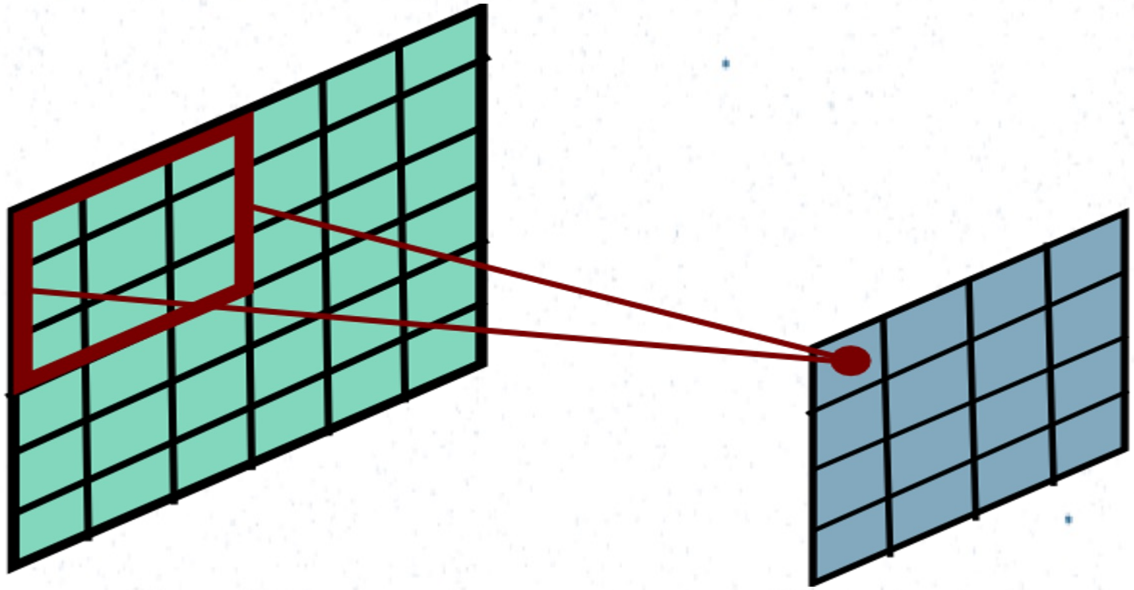


Pixel of an RGB image are formed from the corresponding pixel of the three component images

Convolution(convolution-layer-a.png (1150×500) (stanford.edu))

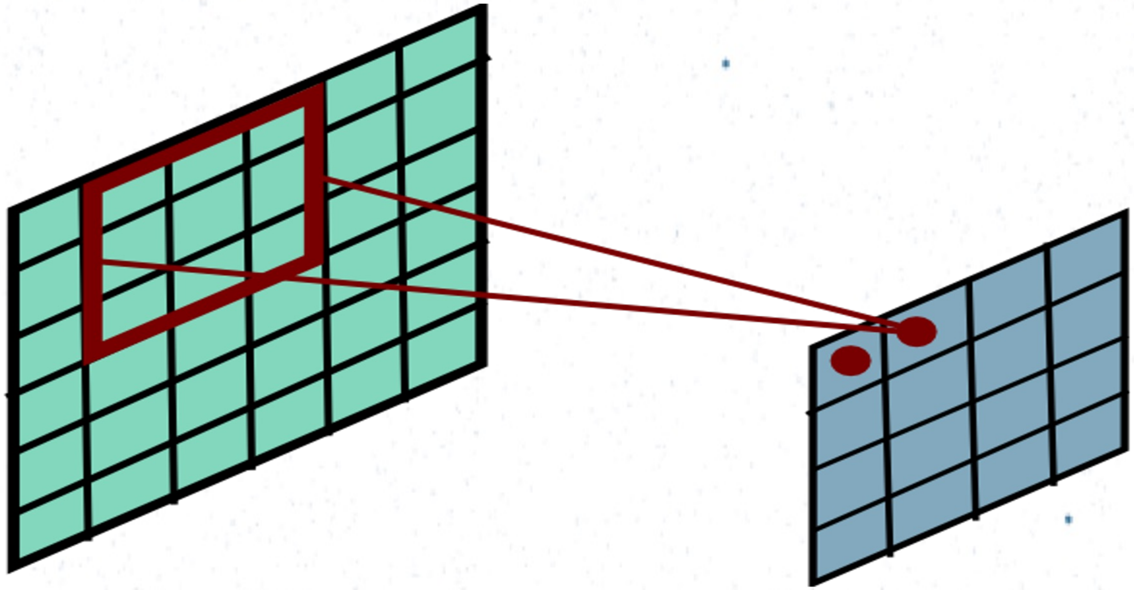


Convolutional Layer - Insight



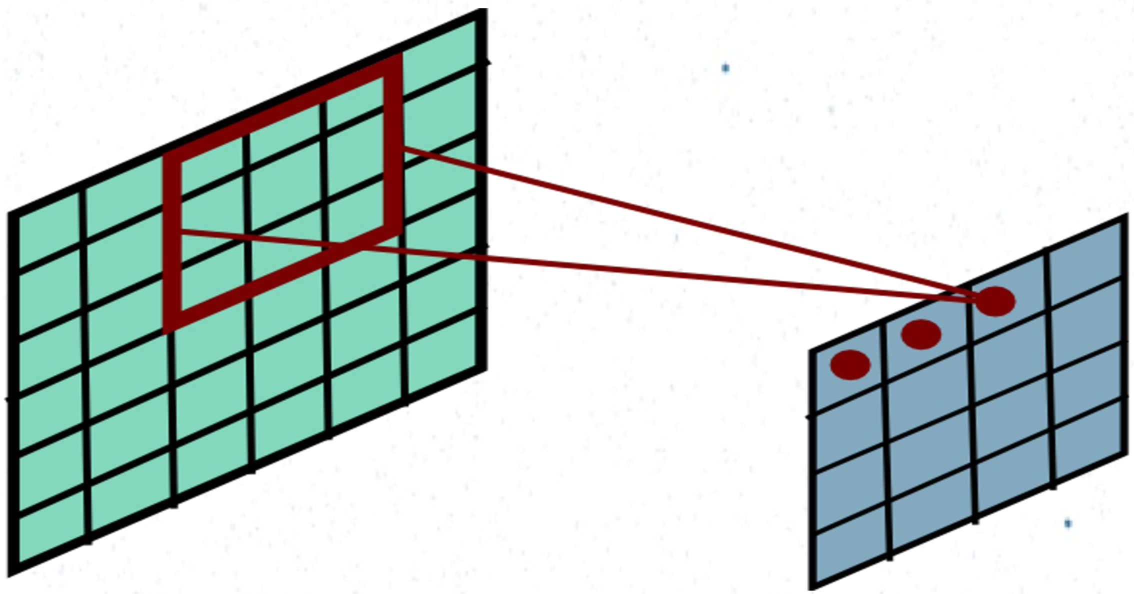
Ranzato

Convolutional Layer - Insight



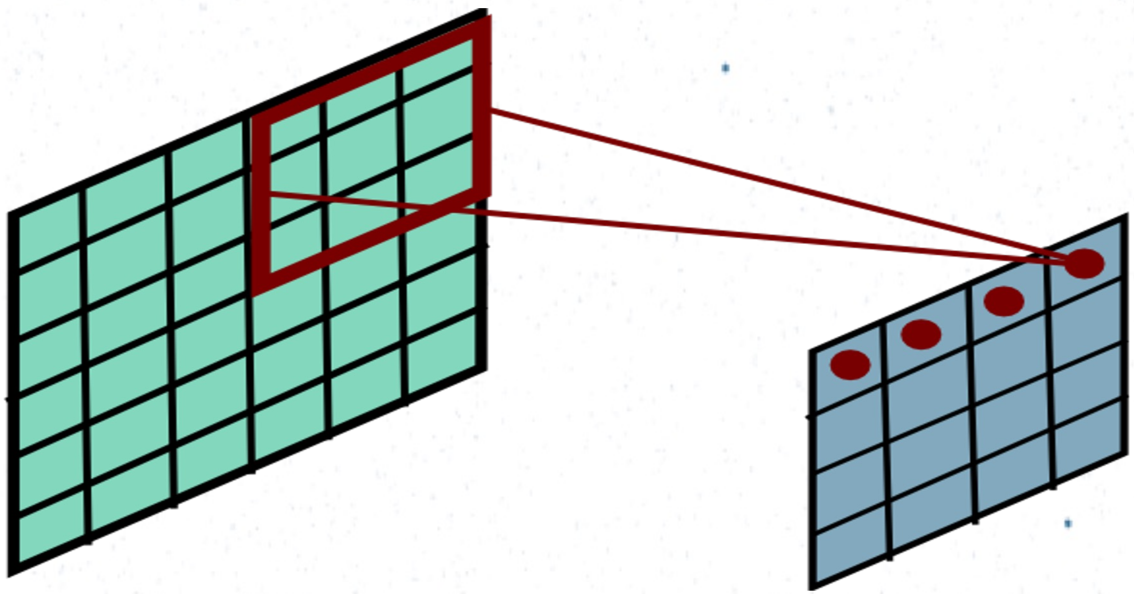
Ranzato

Convolutional Layer - Insight



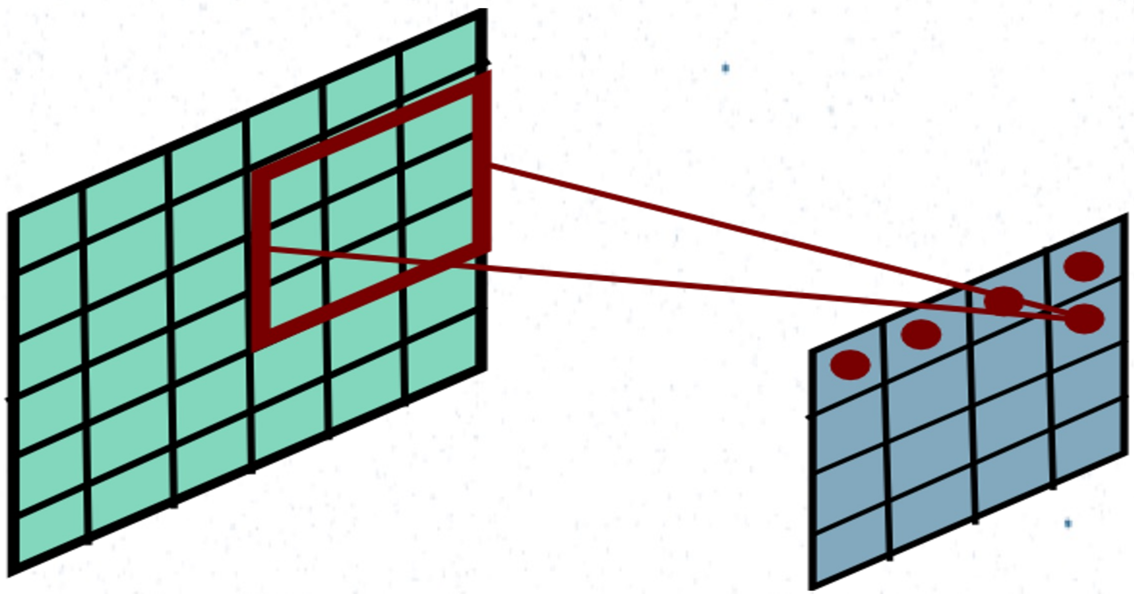
Ranzato

Convolutional Layer - Insight



Ranzato

Convolutional Layer - Insight



Ranzato

What Happens within the Convolution?

Apply a Filter/Kernal

Pooling

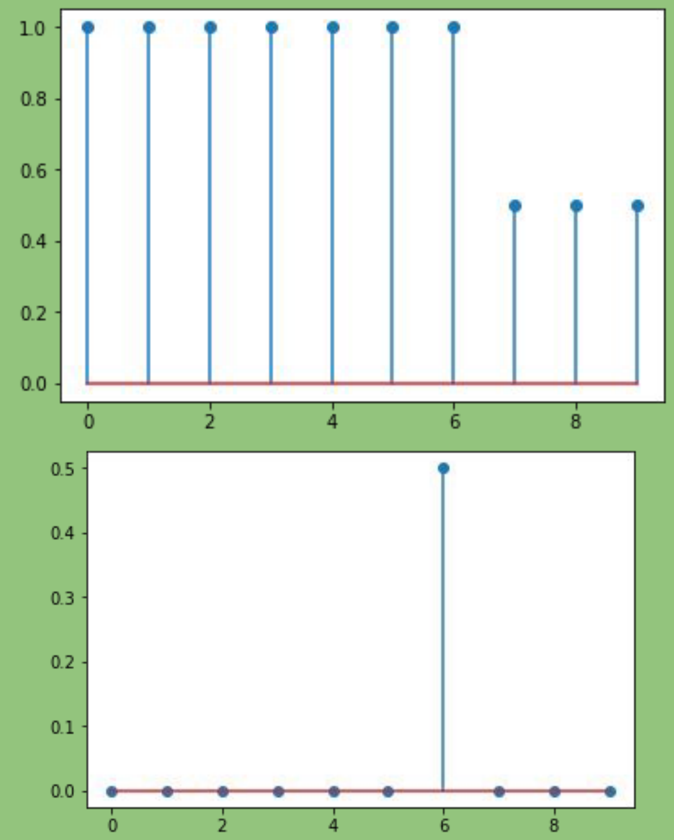
Padding

SLIIT FACULTY OF COMPUTING

Why Use Filters

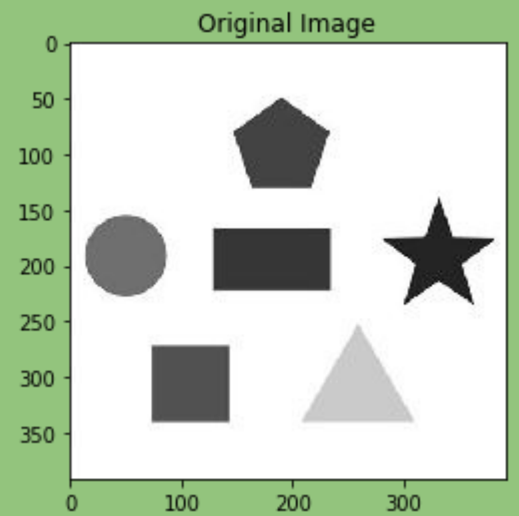
- Filters help extract Different Feature.
- Different Filters will give Different features.

A Simple 1 D Filter



Filter:
[-1, 1]

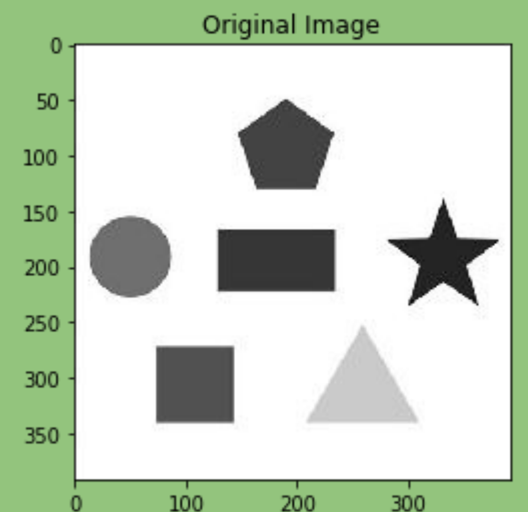
Filters in Action



Filter:

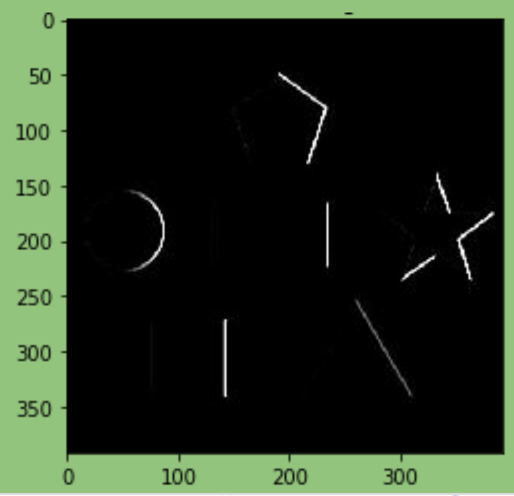
-1	0	1
-1	0	1
-1	0	1

Filters in Action

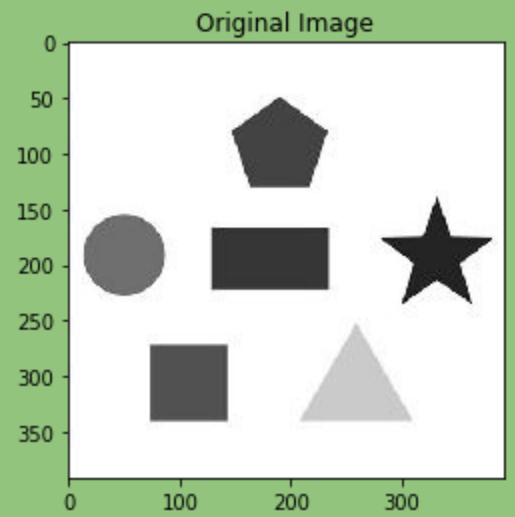


Filter:

-1	0	1
-1	0	1
-1	0	1



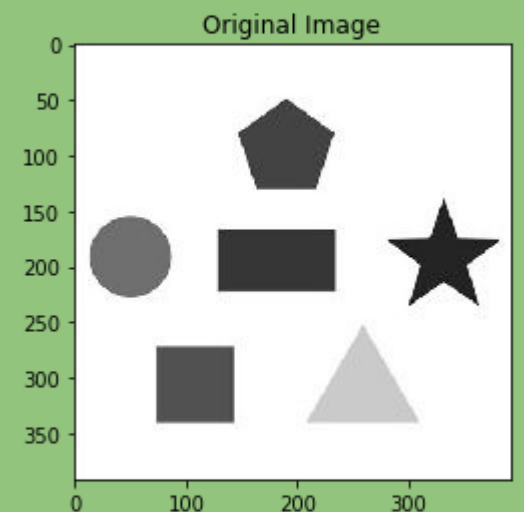
Filters in Action



Filter:

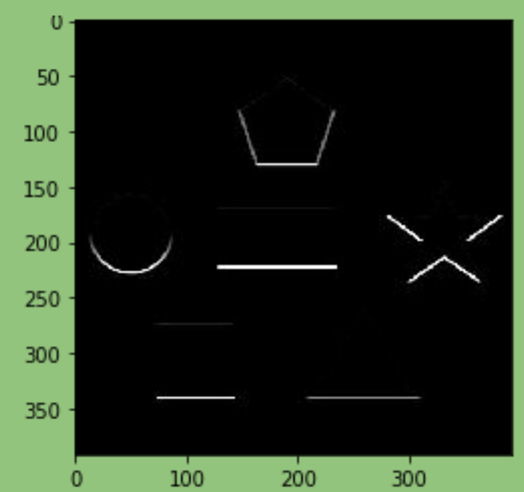
-1	-1	-1
0	0	0
1	1	1

Filters in Action



Filter:

-1	-1	-1
0	0	0
1	1	1



Convolutional Neural Networks - Key Idea

-1	0	1
-1	0	1
-1	0	1

Extracts vertical edges

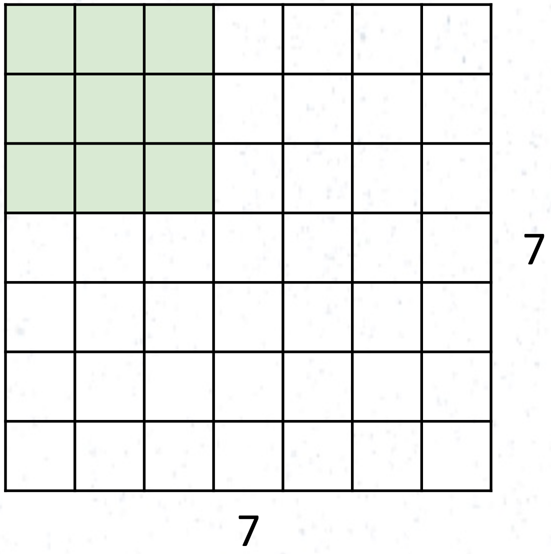
-1	-1	-1
0	0	0
1	1	1

Extracts horizontal edges

w0	w1	w2
w3	w4	w5
w6	w7	w8

Even more powerful & flexible features

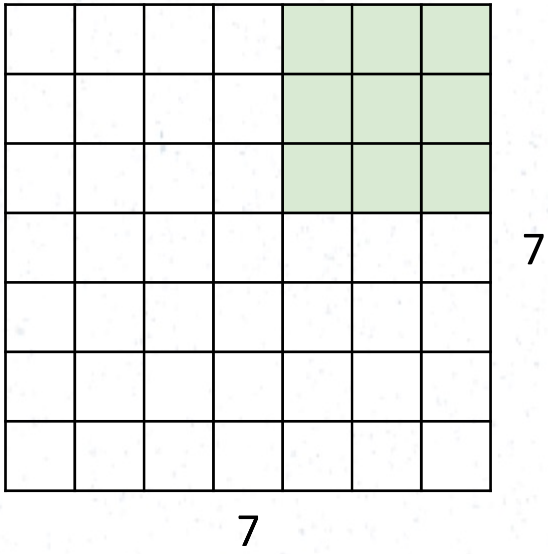
Convolution Exercise 1



Input: 7x7
Filter: 3x3

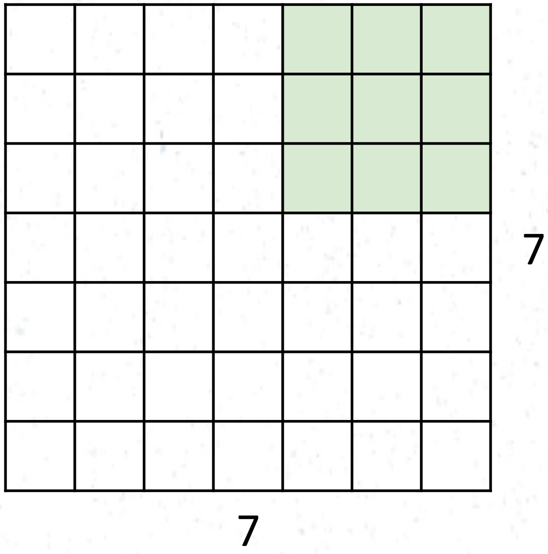
Q: How big is output?

Convolution Exercise 1 - Answer



Input: 7x7
Filter: 3x3
Output: 5x5

Convolution Spatial Dimensions



Input: 7x7
Filter: 3x3
Output: 5x5

In general:
Input: W
Filter: K
Output: W – K + 1

Convolution Calculation

5	3	4
1	0	0
3	6	1
9	0	2
7	4	3

0	1
-1	2

Time for you to Work

5	3	4
1	0	0
3	6	1
9	0	2
7	4	3

0	1
-1	2

2	4

What if we want to Preserve the output Dimensions?

- Padding to the rescue.
- Add zeros around the input set to make it look bigger.
- Increate the number of convolutions we can apply.

Convolution Spatial Dimensions

Input: 7x7
Filter: 3x3
Output: 5x5

0	0	0	0	0	0	0	0	0
0								0
0								0
0								0
0								0
0								0
0								0
0								0
0	0	0	0	0	0	0	0	0

In general:
Input: W
Filter: K
Padding: P
Output: W – K + 1 + 2P

Very common: “same padding”
Set P = (K – 1) / 2
Then output size = input size

Padding

0	0	0	0
0	5	3	4
0	1	0	0
0	3	6	1
0	9	0	2
0	7	4	3

0	1
-1	2

Padding Exercise

0	0	0	0
0	5	3	4
0	1	0	0
0	3	6	1
0	9	0	2
0	7	4	3

0	1
-1	2

10		

Padding

0	0	0	0
0	5	3	4
0	1	0	0
0	3	6	1
0	9	0	2
0	7	4	3

0	1
-1	2

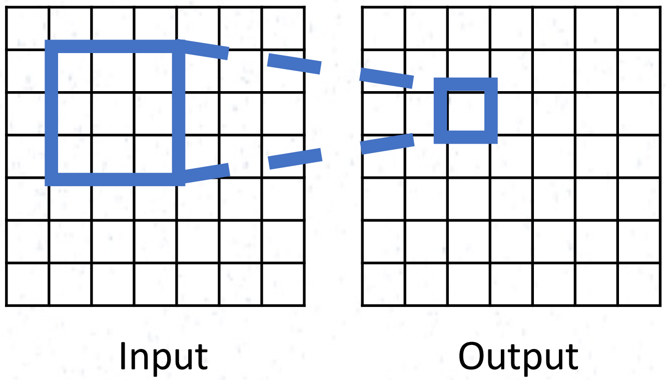
10	1	5
7	2	4
7	9	-4
21	-3	5
23	1	4

Pooling

Downsampling

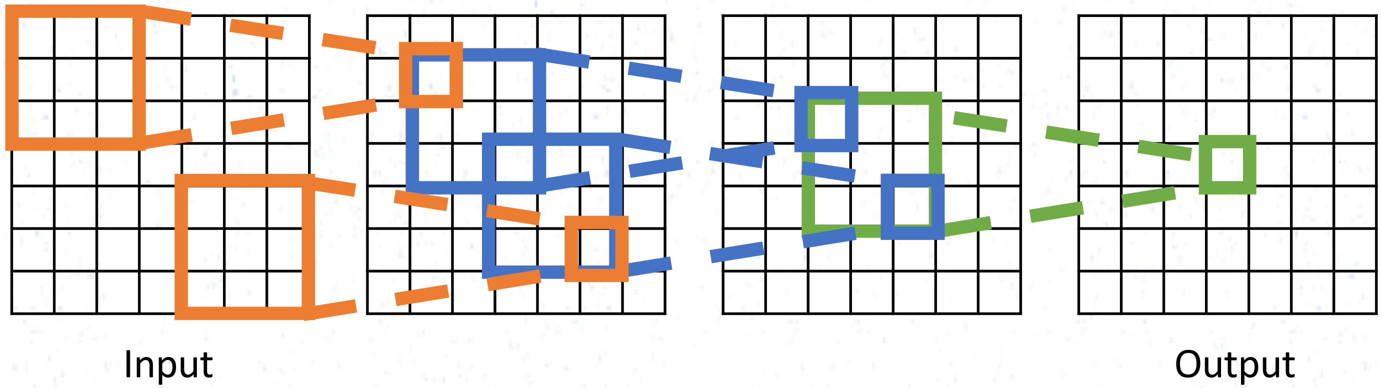
Receptive Fields

For convolution with kernel size K, each element in the output depends on a K x K **receptive field** in the input



Receptive Fields

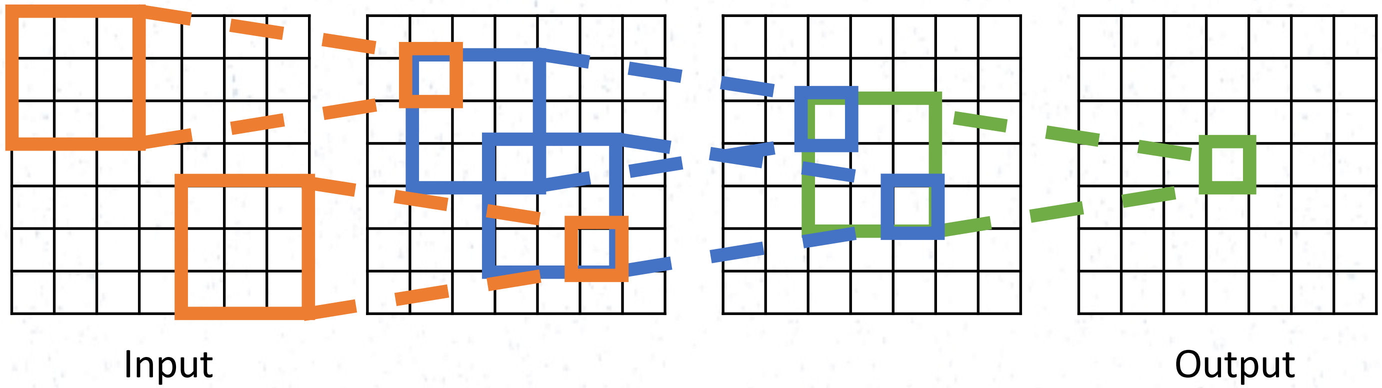
Each successive convolution adds K – 1 to the receptive field size
With L layers the receptive field size is 1 + L * (K – 1)



Careful – ”receptive field wrt to the input”
vs “receptive field wrt the previous layer”

Receptive Fields

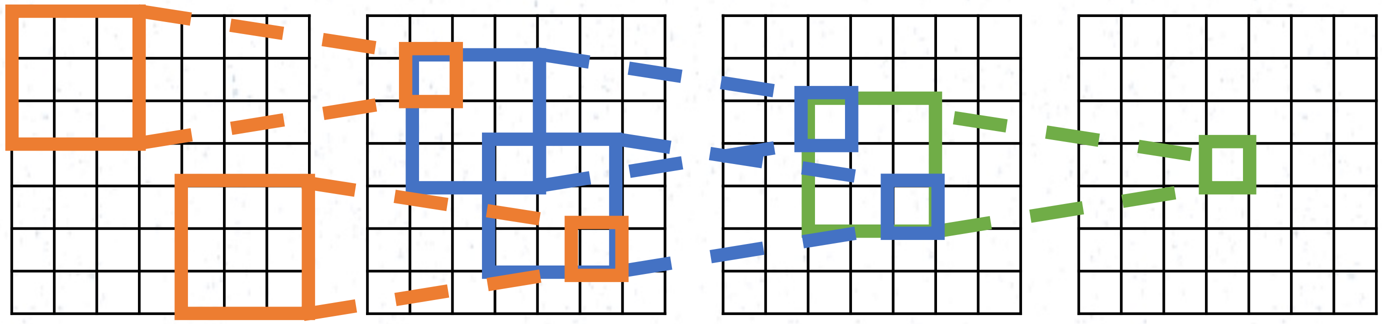
Each successive convolution adds K – 1 to the receptive field size
With L layers the receptive field size is 1 + L * (K – 1)



Problem: For large images we need many layers for each output to “see” the whole image image

Receptive Fields

Each successive convolution adds K – 1 to the receptive field size
With L layers the receptive field size is 1 + L * (K – 1)

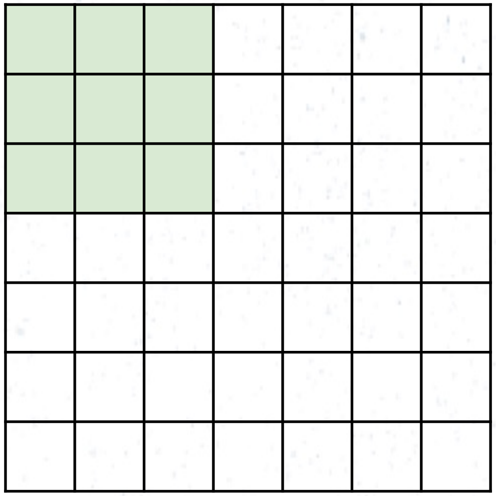


Problem: For large images we need many layers for each output to “see” the whole image image

Solution: Downsample inside the network

Side Note on Strides

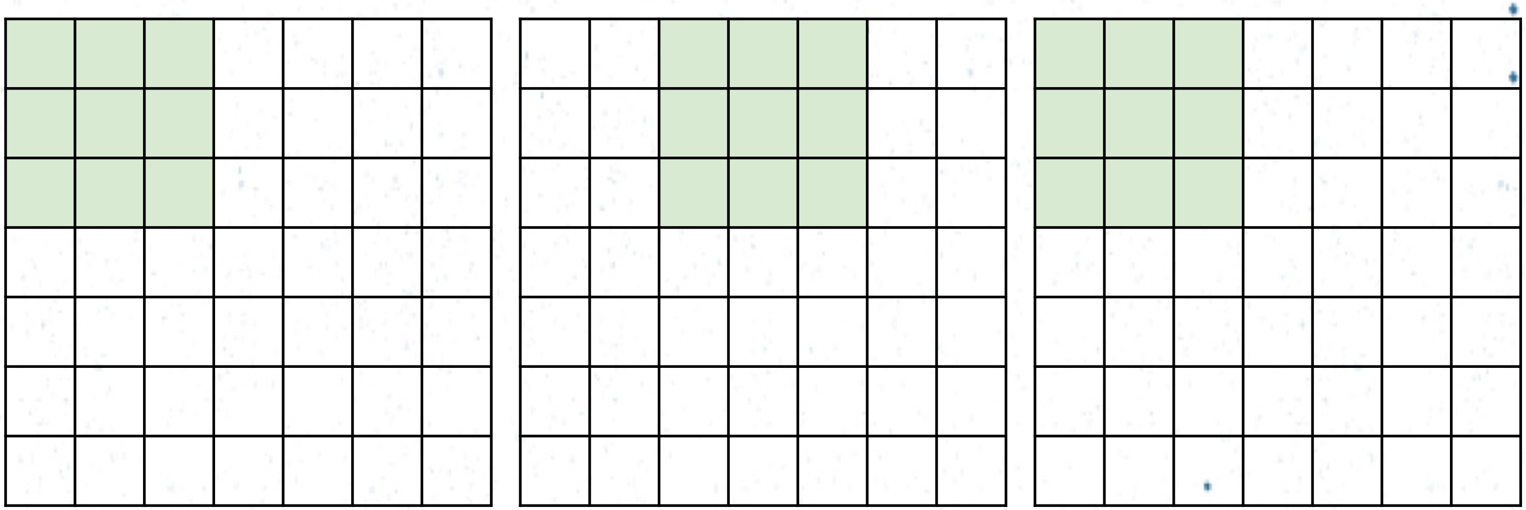
Strided Convolution



Input: 7x7
Filter: 3x3
Stride: 2

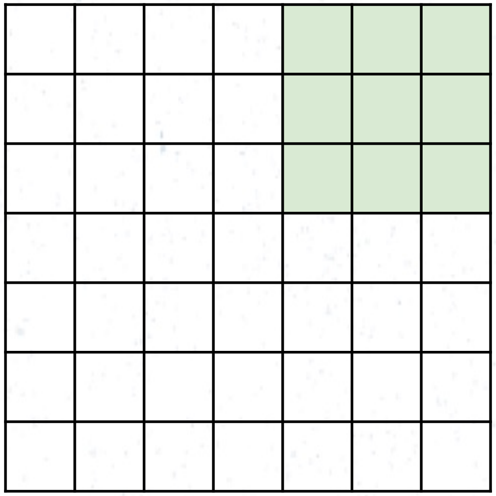
36

Strided Convolution



Input: 7x7
Filter: 3x3
Stride: 2

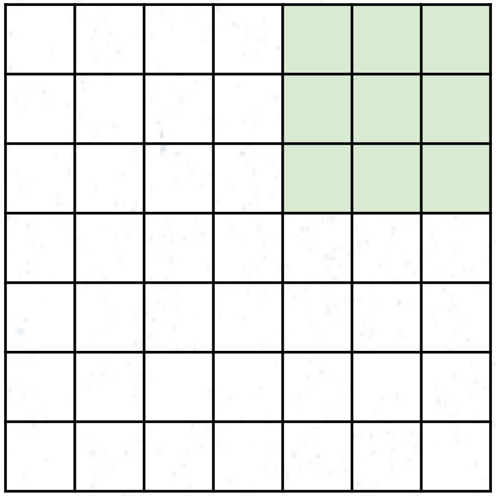
Strided Convolution



Input: 7x7
Filter: 3x3
Stride: 2

Output: 3x3

Strided Convolution



Input: 7x7

Filter: 3x3

Stride: 2

Output: 3x3

In general:

Input: W

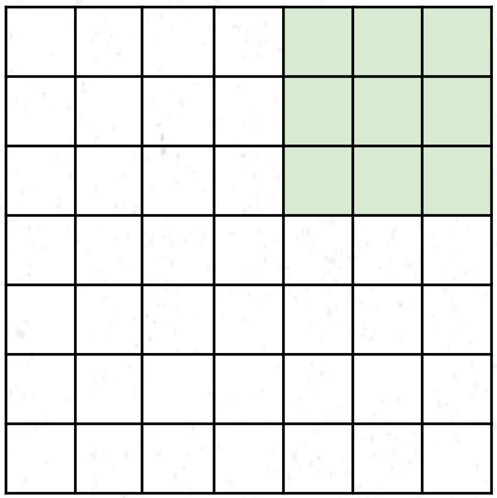
Filter: K

Padding: P

Stride: S

Output: (W – K + 2P) / S + 1

Can we do stride 3?



Input: 7x7

Filter: 3x3

Stride: 3?

Output: 3x3

Output: (N-F) / S + 1

e.g. N = 7, F = 3:

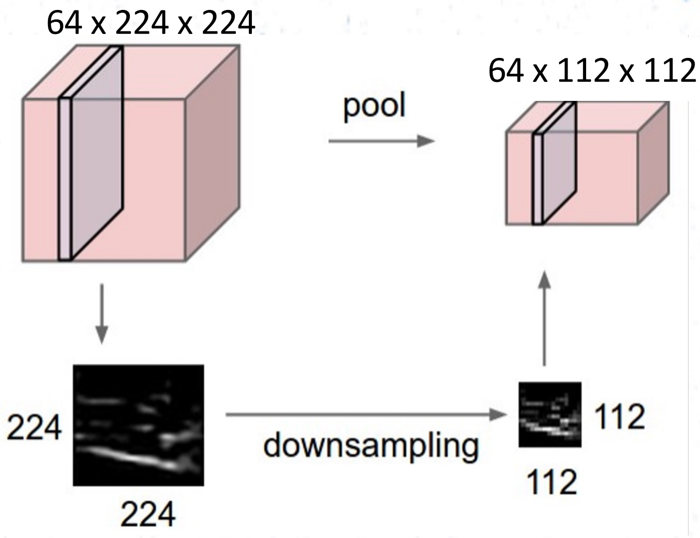
stride 1 => (7 - 3)/1 + 1 = 5

stride 2 => (7 - 3)/2 + 1 = 3

stride 3 => (7 - 3)/3 + 1 = 2.33 ☹

Back to Max Pooling

Pooling Layers: Downsampling



Hyperparameters:
Kernel Size
Stride
Pooling function

Max Pooling

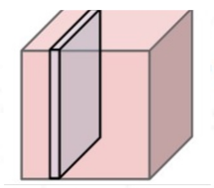
Single depth slice

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

x

y

64 x 224 x 224



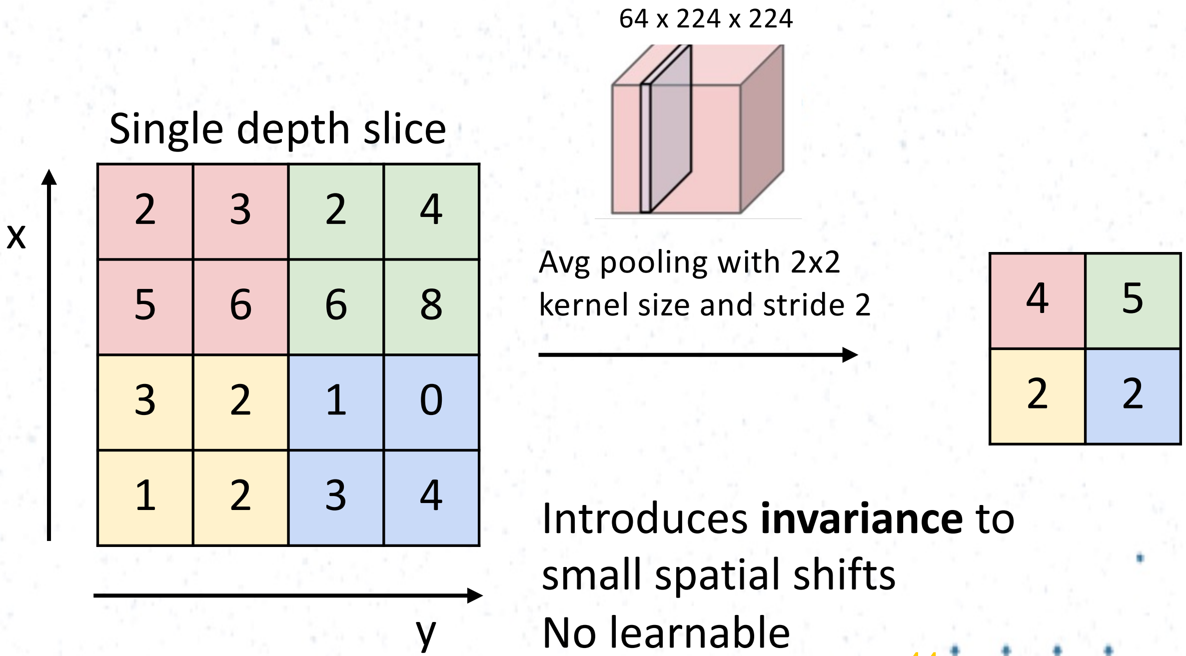
Max pooling with 2x2 kernel size and stride 2

6	8
3	4

Introduces **invariance** to small spatial shifts

No learnable parameters!

Average Pooling



Introduces **invariance** to small spatial shifts

No learnable parameters!

Pooling Summary

Input: C x H x W

Hyperparameters:
- Kernel size: K
- Stride: S
- Pooling function (max, avg)

Output: C x H’ x W’ where
- H’ = (H – K) / S + 1
- W’ = (W – K) / S + 1

Learnable parameters: None!

Common settings:
max, K = 2, S = 2
max, K = 3, S = 2 (AlexNet)

Tips : Normalization

Normalization

Why do we need to normalize our data?

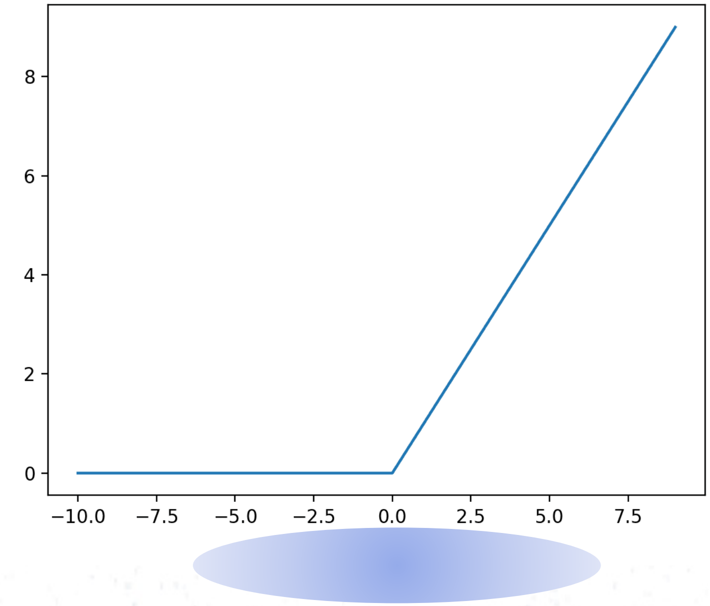
Extreme example:

- sometimes pixel range is [0, 255], sometimes it's [0, 1]

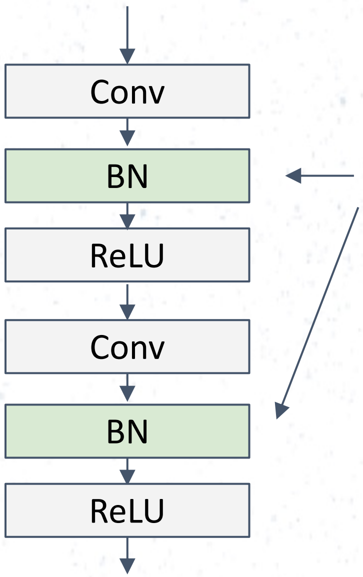
What is the problem?

Network activations will be completely different!!

You want the inputs to be in a similar range → low variance



Normalization



Usually inserted after Fully Connected or Convolutional layers, and before nonlinearity.

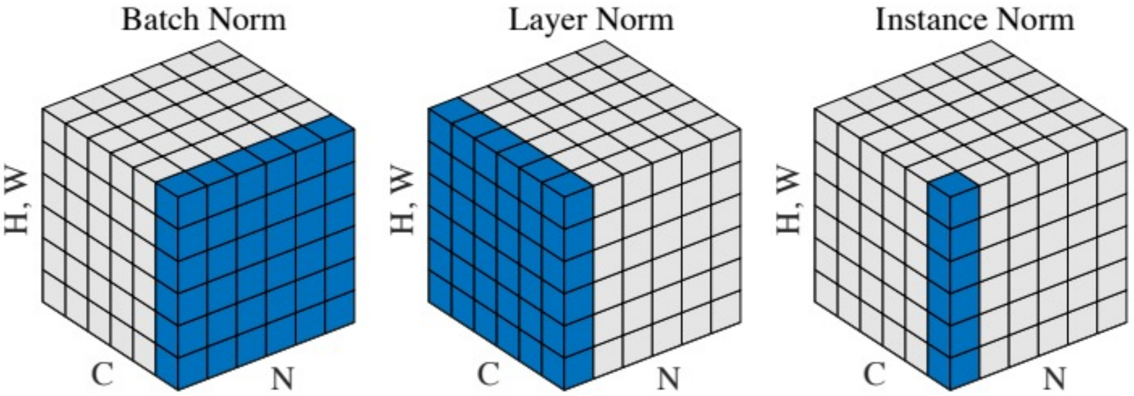
$$\hat{x} = \frac{x - E[x]}{\sqrt{Var[x]}}$$

How to normalize

Next Q: from **what** do you compute the mean & variance?

- BatchNorm computes mean and var from each batch
- Depends on the batch, so need to keep a running average and store these.

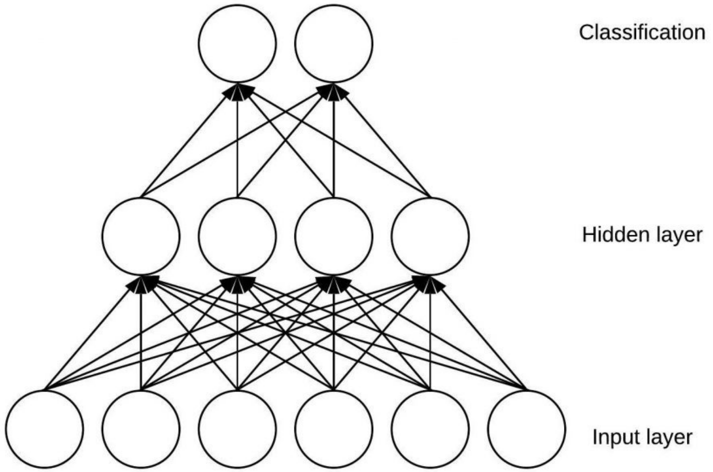
- LayerNorm/InstanceNorm do not need this.



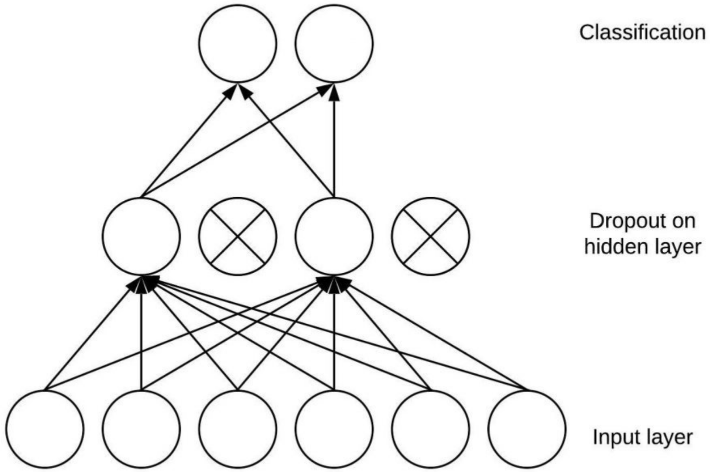
Use of Dropout Layers

Dropout

- Literally used to drop out some neurons
- It acts as a mask that nullifies the contribution of random neuron.
- This prevents them from passing in values to the next layer.
- Good for dealing with overfitting



Without Dropout



With Dropout

What Activation Functions can we use

Non-negative Activations

Softmax (output layer)

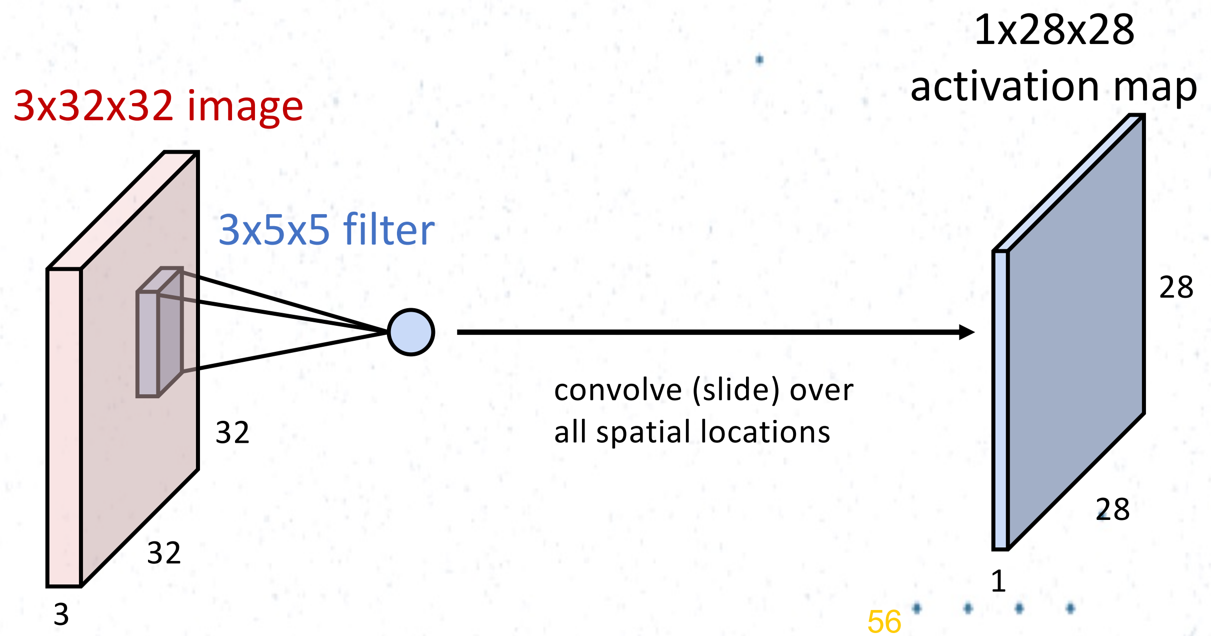
Other Components

- Flattening Layer
- Fully Connected Layer

Overall Architecture

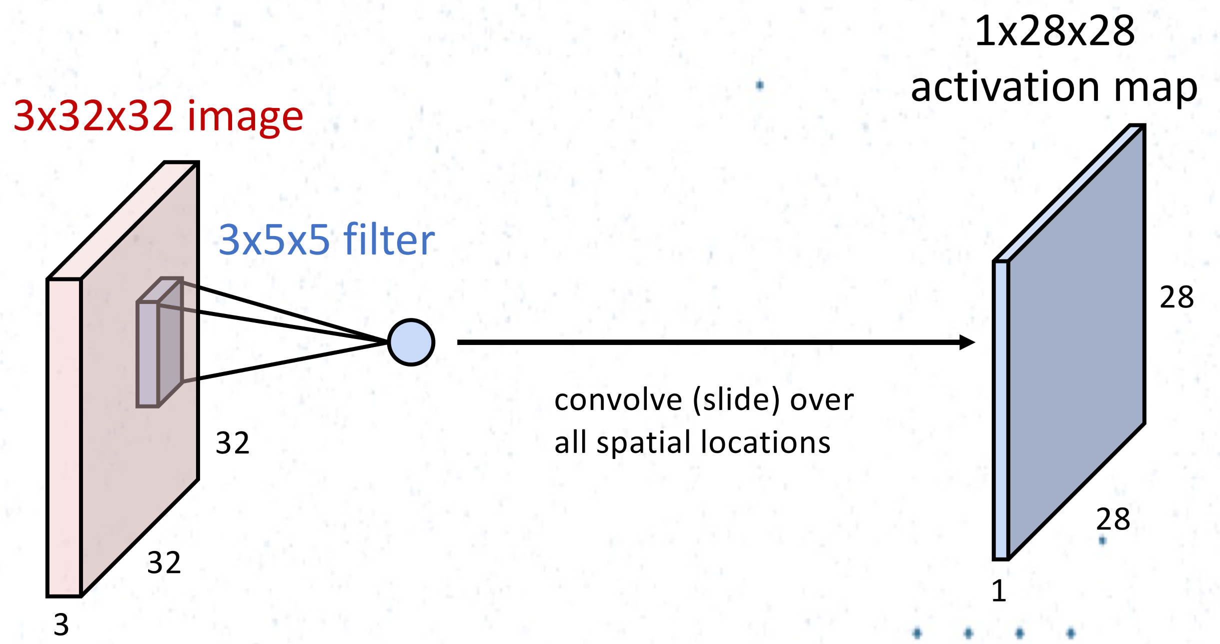
Convolution Layer

One neuron, that looks at 5x5 region and outputs a sheet of activation map

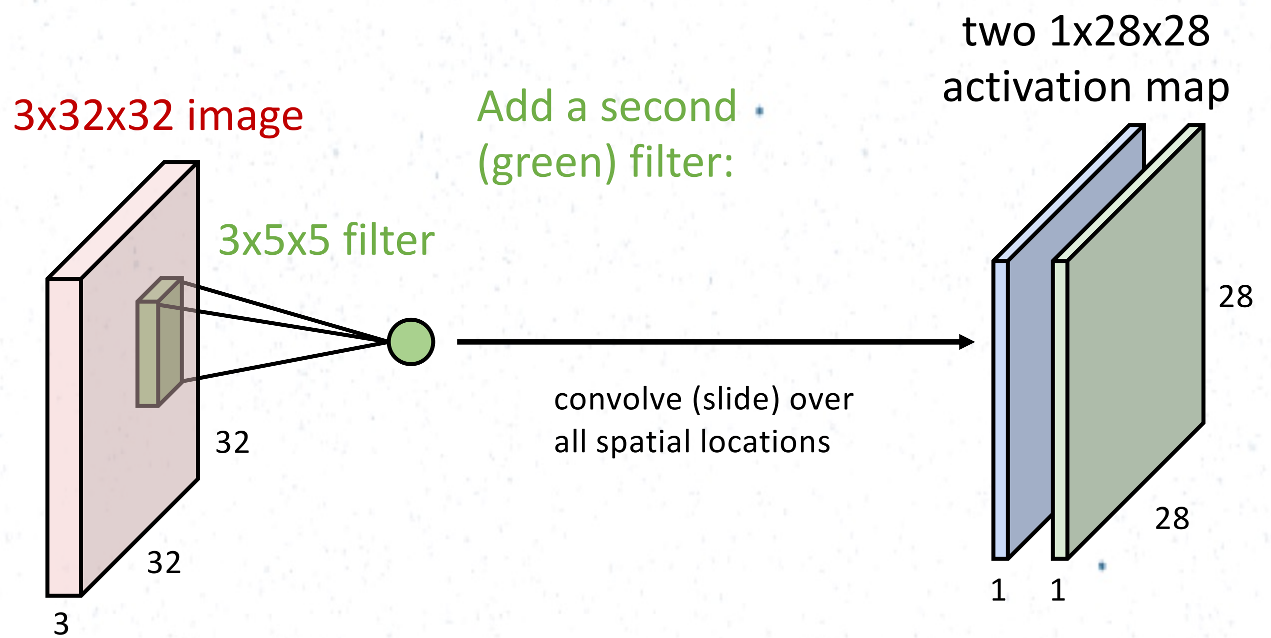


Convolution Layer

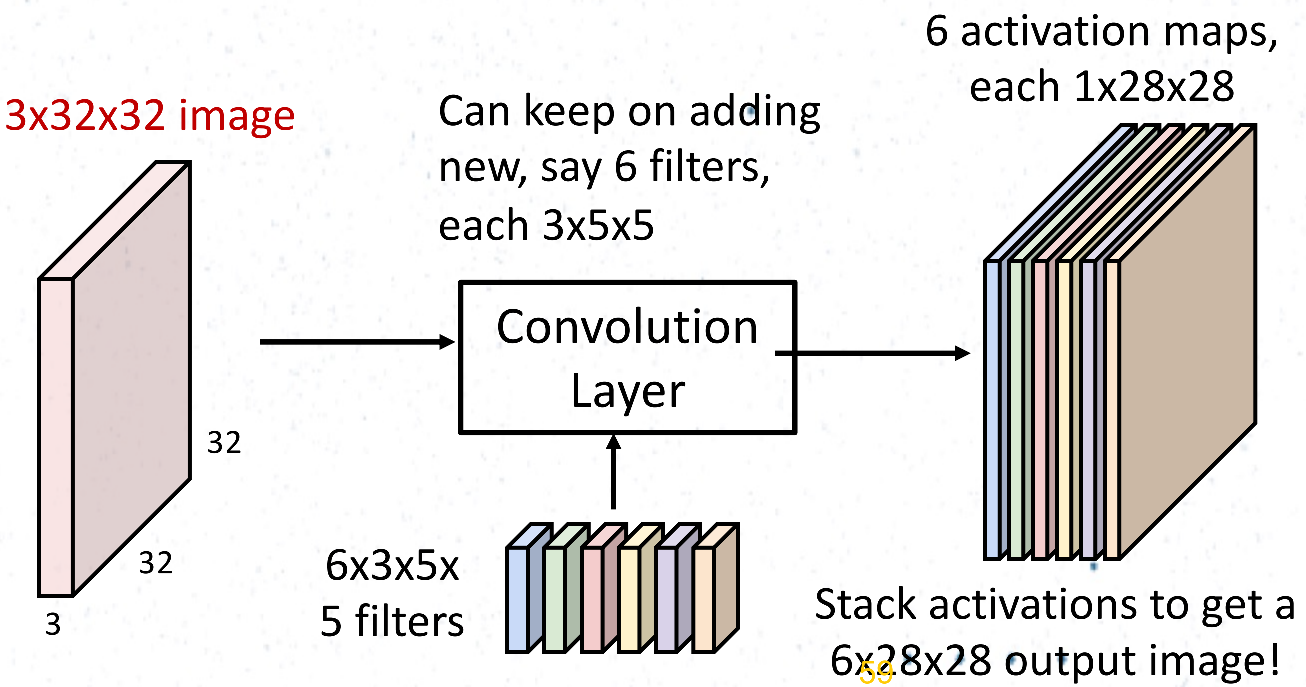
One neuron, that looks at 5x5 region and outputs a sheet of activation map



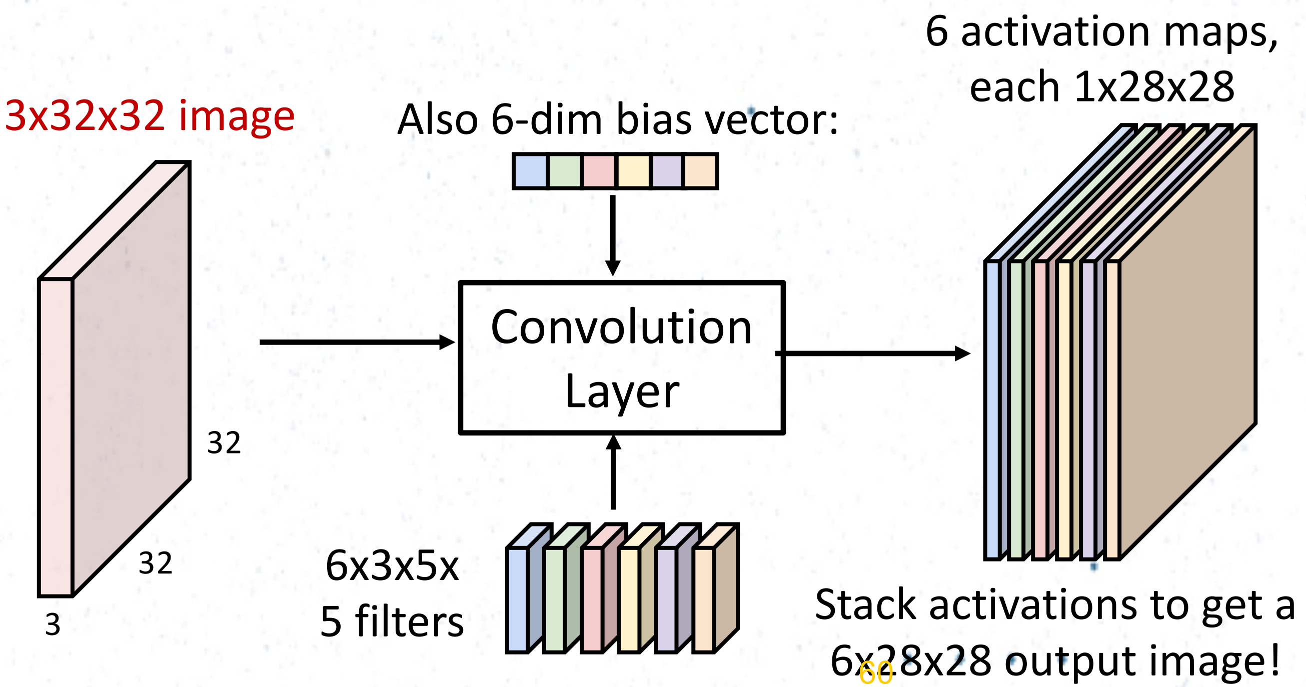
Convolution Layer:



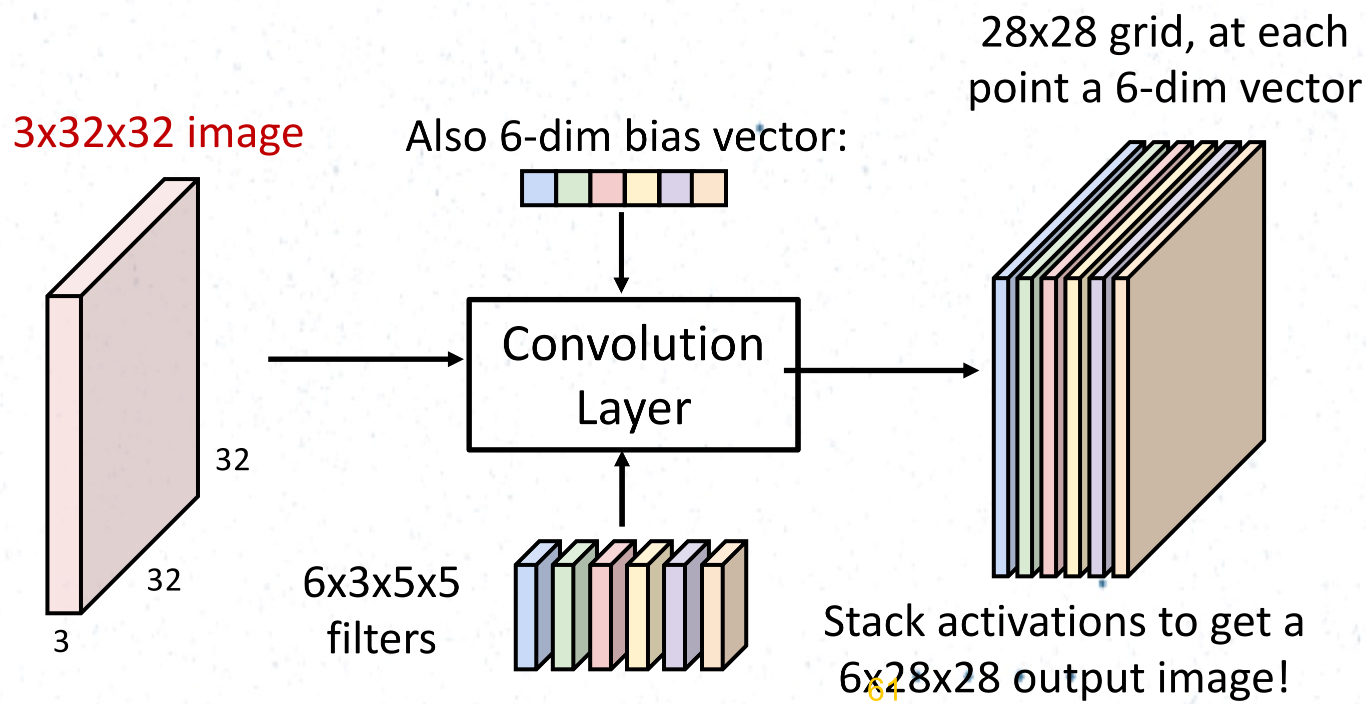
Convolution Layer



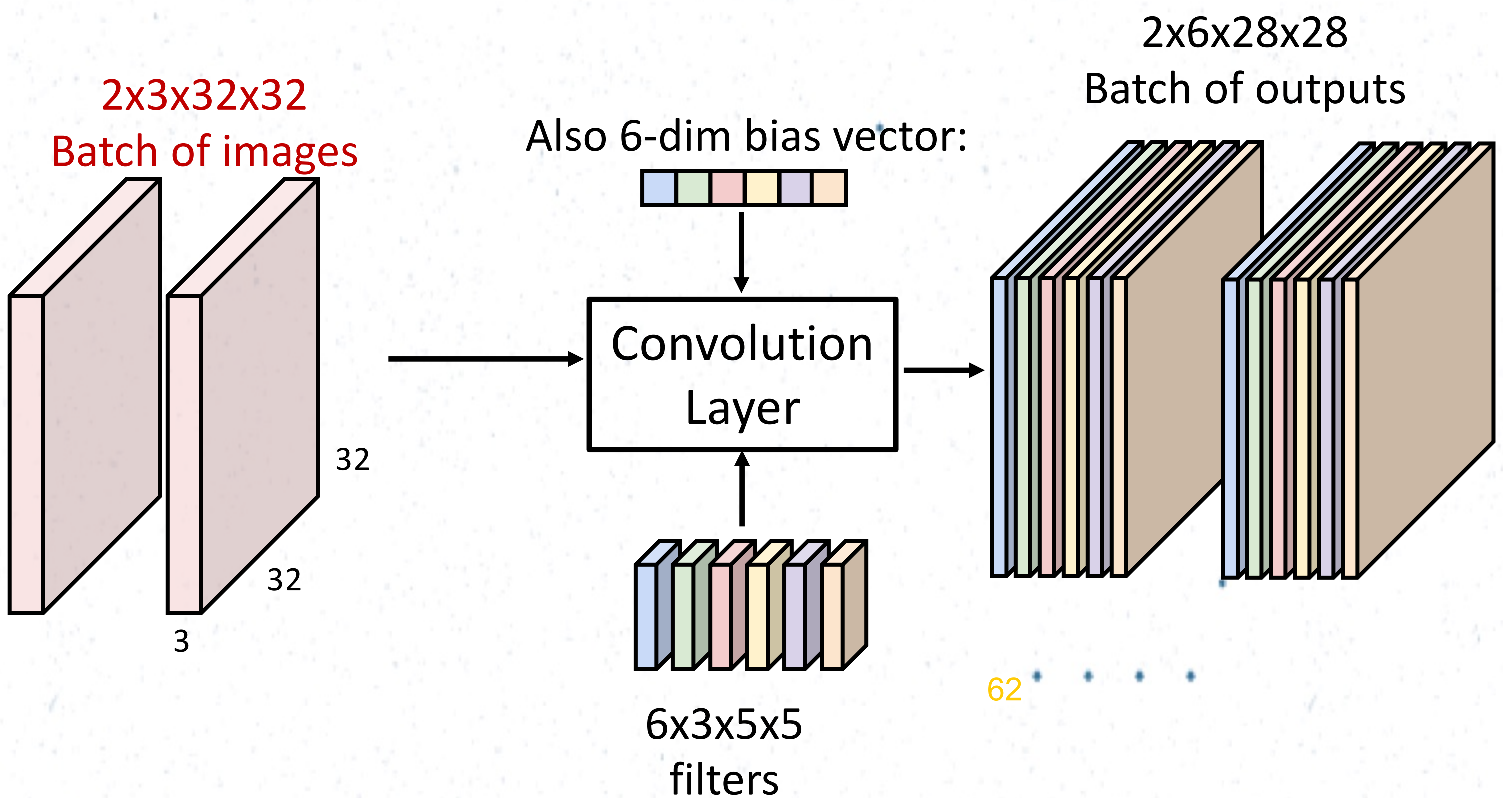
Convolution Layer



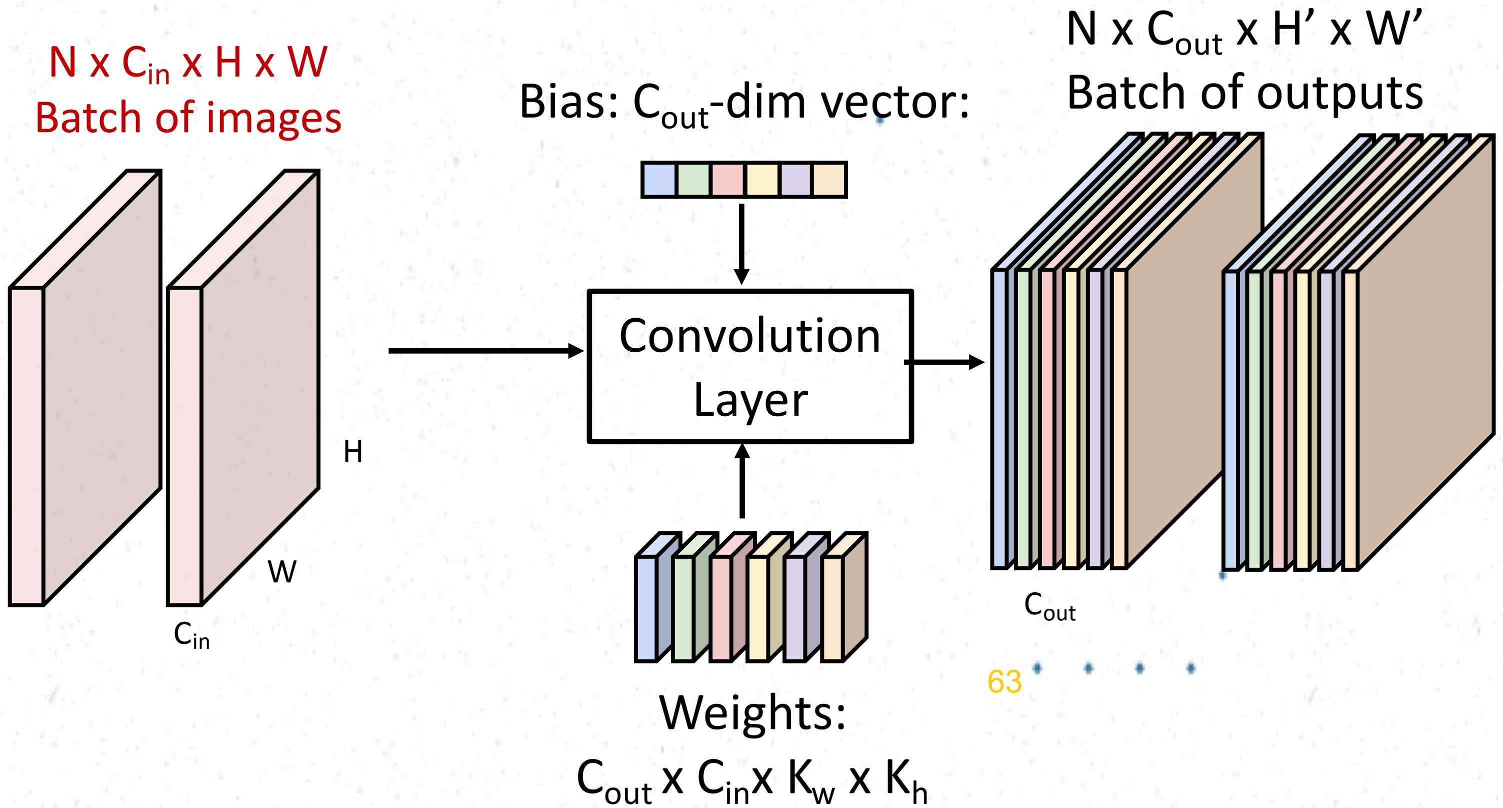
Convolution Layer



Convolution Layer: With many images

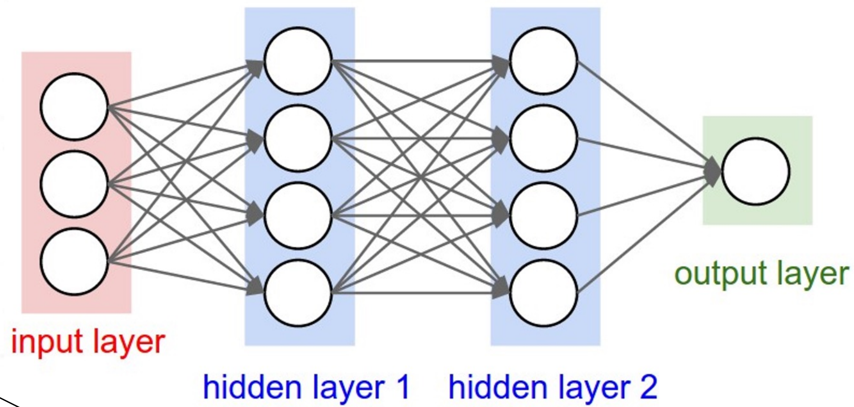


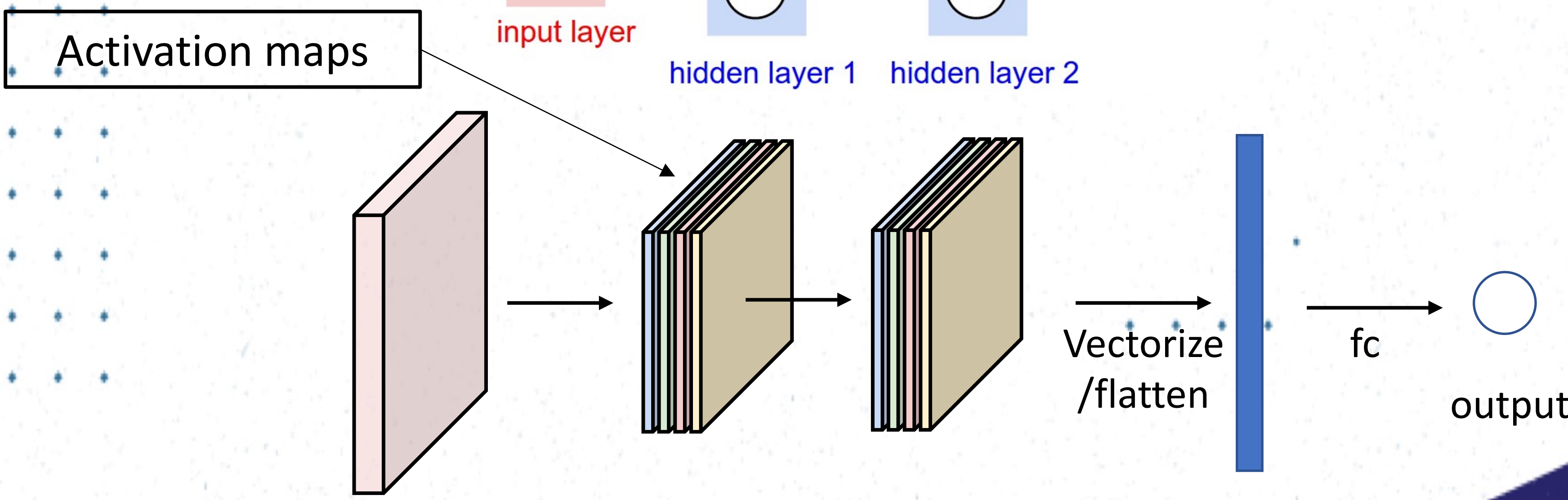
Convolution Layer



SLIIT FACULTY OF COMPUTING

Compared with MLPs





Detecting the Parameter counts

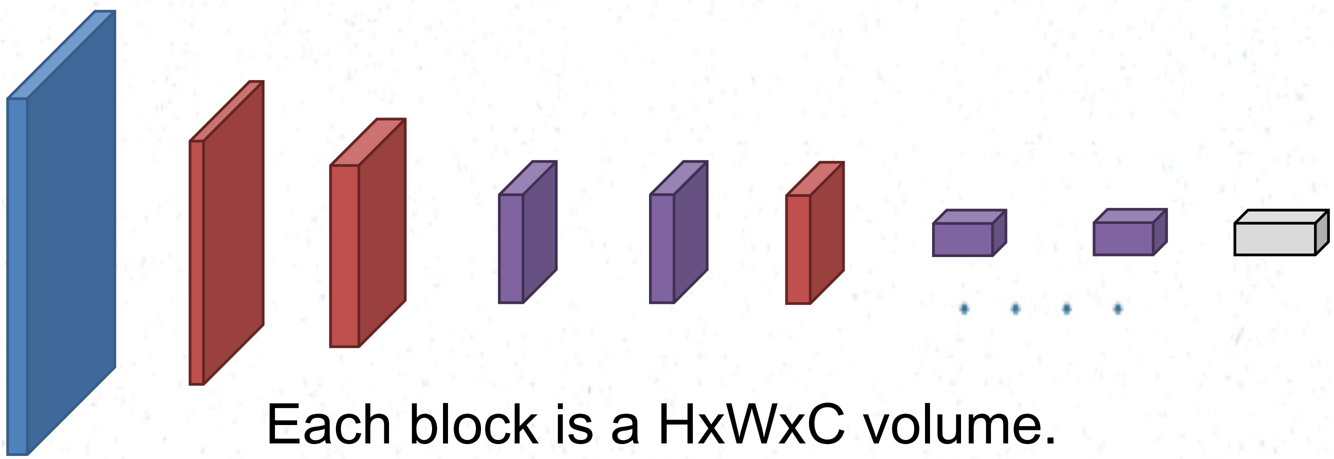
SLIIT FACULTY OF COMPUTING

Case Study: AlexNet

[Krizhevsky, Sutskever, Hinton, NeurIPS 2012]

Task: ImageNet 1000-class classification

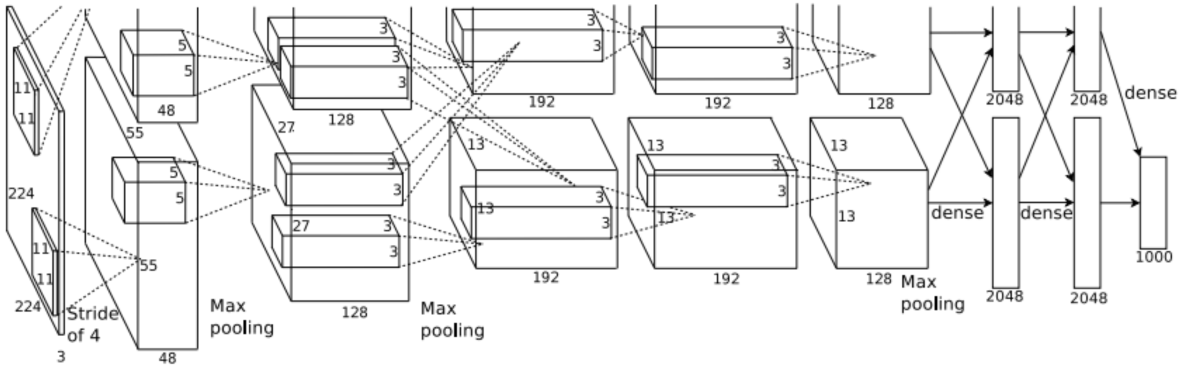
	Input	Conv 1	Conv 2	Conv 3	Conv 4	Conv 5	FC 6	FC 7	Output
HxW	227x 227	55x55	27x27	13x13	13x13	13x13	1x1	1x1	1x1
C	3	96	256	384	384	256	4096	4096	1000



Each block is a HxWxC volume.
You transform one volume to another with convolution

Case Study: AlexNet

[Krizhevsky, Sutskever, Hinton, NeurIPS 2012]



Input: 227x227x3 images
First layer (CONV1): 96 11x11 filters applied at stride 4

Hint:
(227-11)/4 + 1 = 55

Q: What is the output volume size after Conv1?

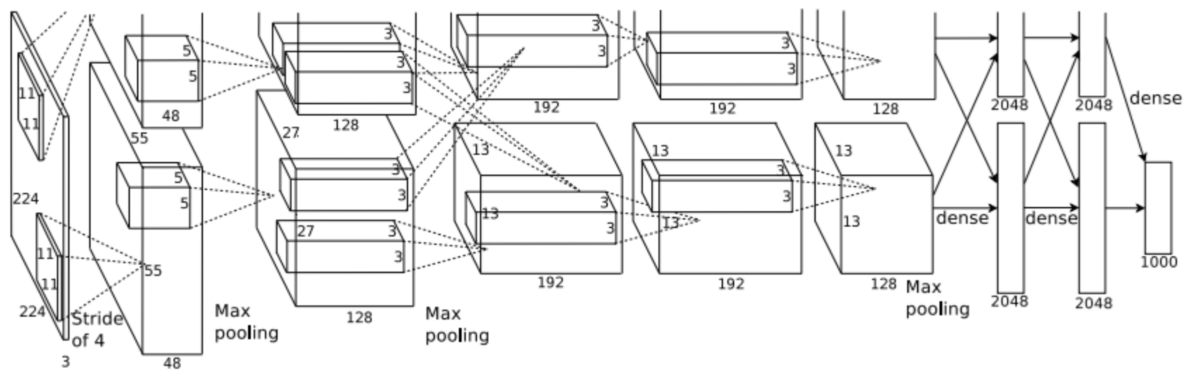
A: 55x55x96

Q: how many parameters in this layer?

A: (3*11*11) * 96

Case Study: AlexNet

[Krizhevsky, Sutskever, Hinton, NeurIPS 2012]



Input: 227x227x3 images
After Conv1: 55x55x96
Second layer: Pool1: 3x3 at stride 2

Q: What is the output volume size after Pool1?

A: (55-3)/2 + 1 = 27
27x27x96

Q: how many parameters in this layer?

A: 0!!!

How to Prevent Overfitting?

- Early stopping - don't overdo training!
- Weight decay / regularization with L1/L2 norm

$$\boxed{E(W) = \sum_{i=0}^{N} \mathrm{loss}(f_W(x_i), y_i) + \gamma \|W\|^2}$$

- Drop out
- Use loads of training data with good variability
- Use a carefully designed network architecture with batch normalization etc.