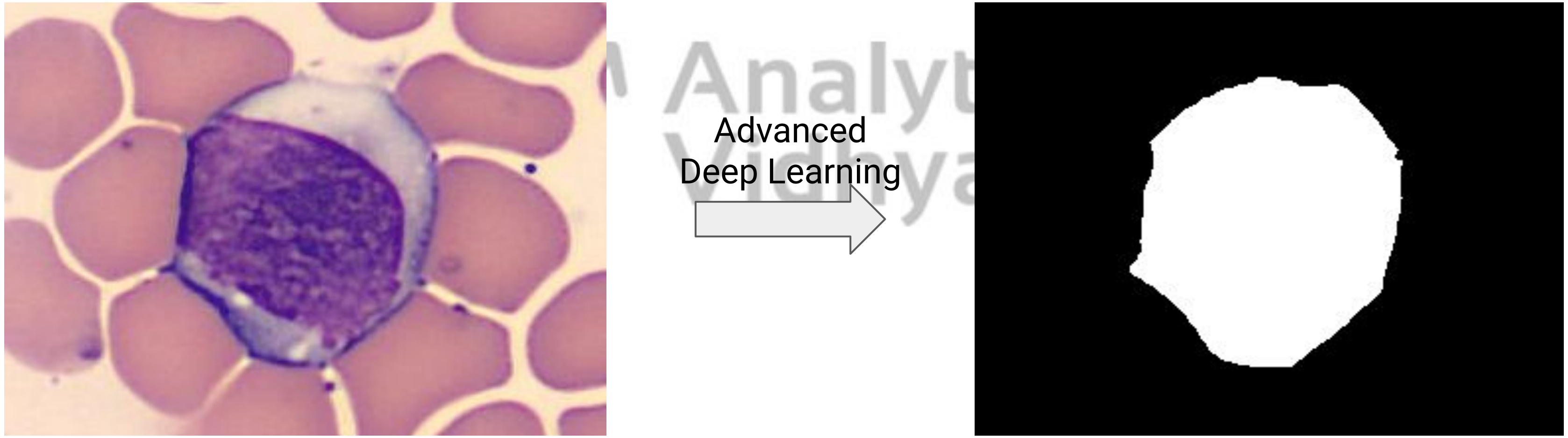
Approach 3 - End to End Deep Learning Approach

What we will be covering in this module?

- Introduction is Image Segmentation
- How to solve Image Segmentation problems?
- Approaches for Image Segmentation
  - Use Traditional Methods
  - Leverage Deep Learning
- Understanding Deep Learning Architectures for Image Segmentation
- Project on Lane Segmentation for Self Driving Cars
- What's Next?

Approach to solve Blood Cell Segmentation



Summary of Naive DL based Approach

Cons:

- Feasible but still computationally expensive
- Options for DL architecture is limited
- Simplistic DL model, doesn’t take ideas from complex networks

What do we need?

- Feasible but still computationally expensive (as no pooling layer is used)

What do we need?

- Feasible but still computationally expensive (as no pooling layer is used)

➔ Inverse Operation for Pooling

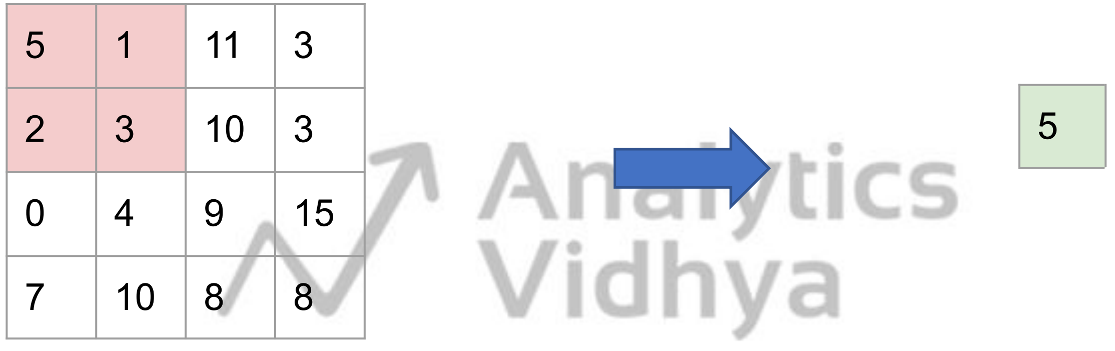
Recap - Max Pooling

5	1	11	3
2	3	10	3
0	4	9	15
7	10	8	8

4 x 4

Pool size = 2 x 2

Recap - Max Pooling



4 x 4

Pool size = 2 x 2

Recap - Max Pooling

5	1	11	3
2	3	10	3
0	4	9	15
7	10	8	8

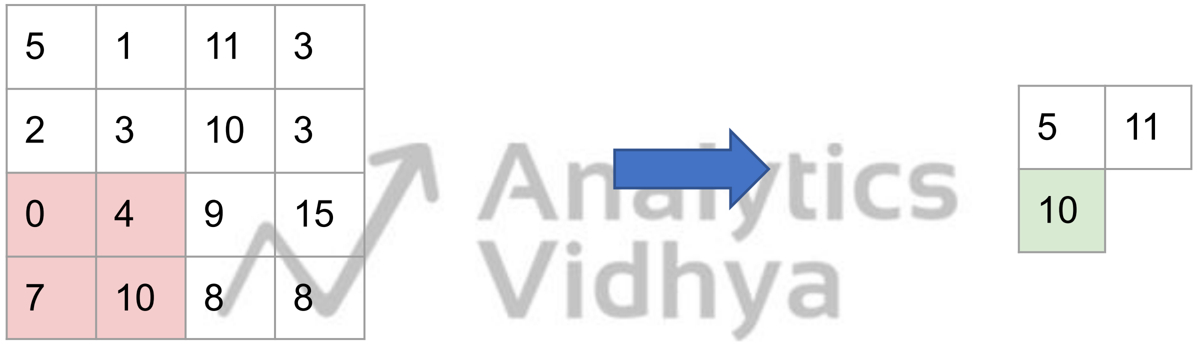
4 x 4

$\Rightarrow$

Pool size = 2 x 2

5	11

Recap - Max Pooling



5	1	11	3
2	3	10	3
0	4	9	15
7	10	8	8

4 x 4

Pool size = 2 x 2

5	11
10	

Recap - Max Pooling

5	1	11	3
2	3	10	3
0	4	9	15
7	10	8	8

4 x 4

Pool size = 2 x 2

5	11
10	15

2 x 2

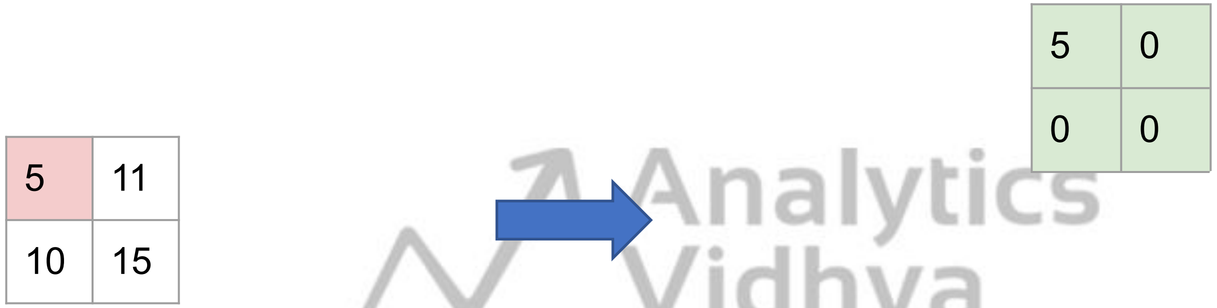
Method 1: Simple idea for UnPooling

5	11
10	15

2 x 2

Filter size = 2 x 2

Method 1: Simple idea for UnPooling



5	11
10	15

5	0
0	0

2 x 2

Filter size = 2 x 2

Method 1: Simple idea for UnPooling

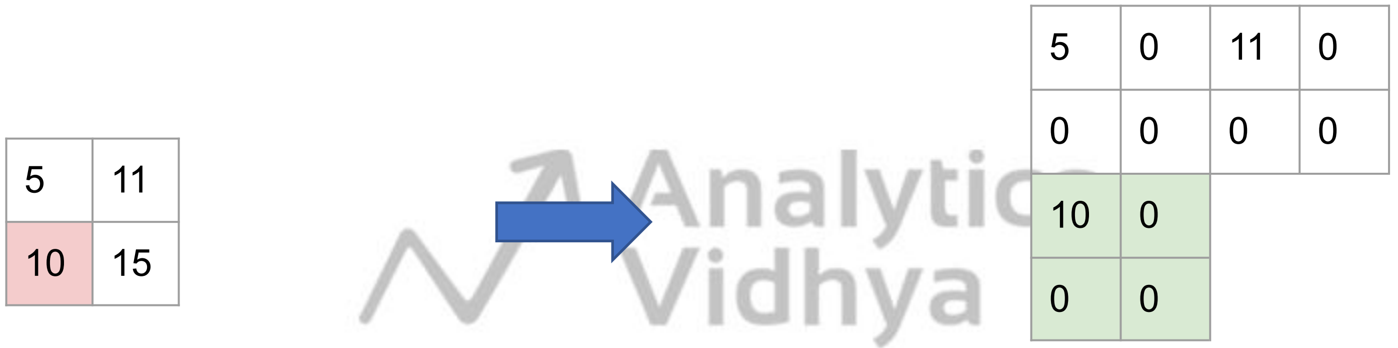
5	11
10	15

2 x 2

Filter size = 2 x 2

5	0	11	0
0	0	0	0

Method 1: Simple idea for UnPooling



5	11
10	15

2 x 2

Filter size = 2 x 2

5	0	11	0
0	0	0	0
10	0		
0	0		

Method 1: Simple idea for UnPooling

5	11
10	15

2 x 2

Filter size = 2 x 2

5	0	11	0
0	0	0	0
10	0	15	0
0	0	0	0

4 x 4

Method 2: Bilinear Interpolation Operation

5	11
10	15

2 x 2

Filter size = 2 x 2

Method 2: Bilinear Interpolation Operation

5	11
10	15

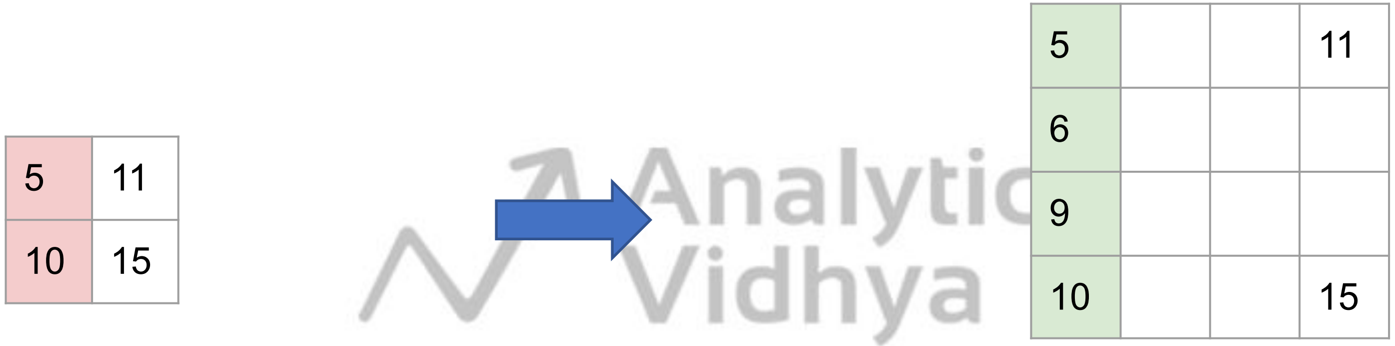
2 x 2

Filter size = 2 x 2

5			11
10			15

4 x 4

Method 2: Bilinear Interpolation Operation



2 x 2

Filter size = 2 x 2

4 x 4

Method 2: Bilinear Interpolation Operation

5	11
10	15

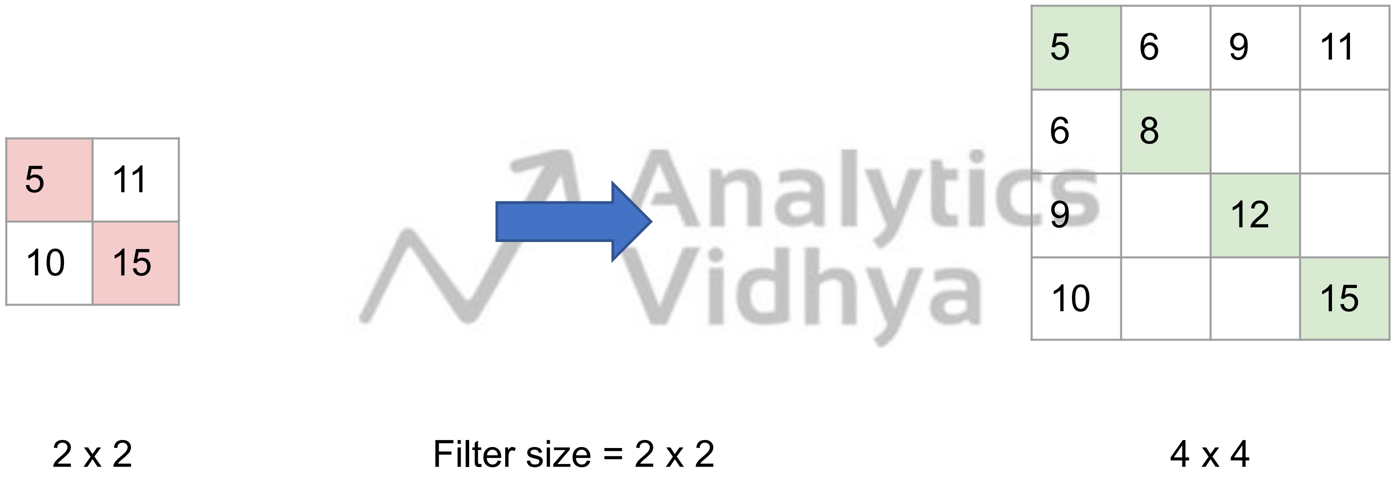
2 x 2

Filter size = 2 x 2

5	6	9	11
6			
9			
10			15

4 x 4

Method 2: Bilinear Interpolation Operation



5	11
10	15

2 x 2

Filter size = 2 x 2

5	6	9	11
6	8		
9		12	
10			15

4 x 4

Method 2: Bilinear Interpolation Operation

5	11
10	15

2 x 2

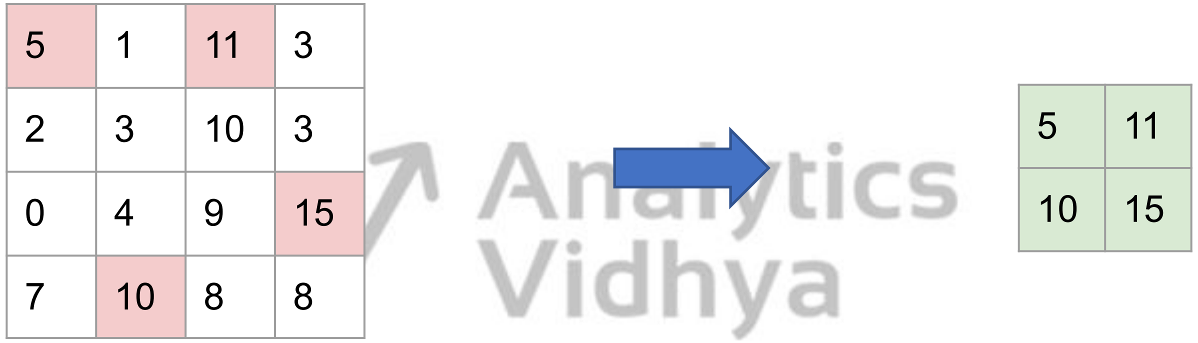
Filter size = 2 x 2

5	6	9	11
6	8	10	12
9	10	12	14
10	11	14	15

4 x 4

Method 3: Max Unpooling Operation

Method 3: Max Pooling Indices



5	1	11	3
2	3	10	3
0	4	9	15
7	10	8	8

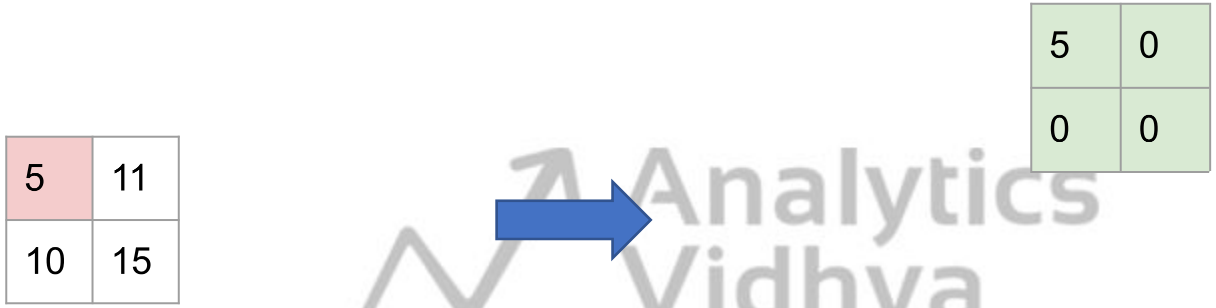
4 x 4

Pool size = 2 x 2

5	11
10	15

2 x 2

Method 3: Max Unpooling Operation



2 x 2

Filter size = 2 x 2

Method 3: Max Unpooling Operation

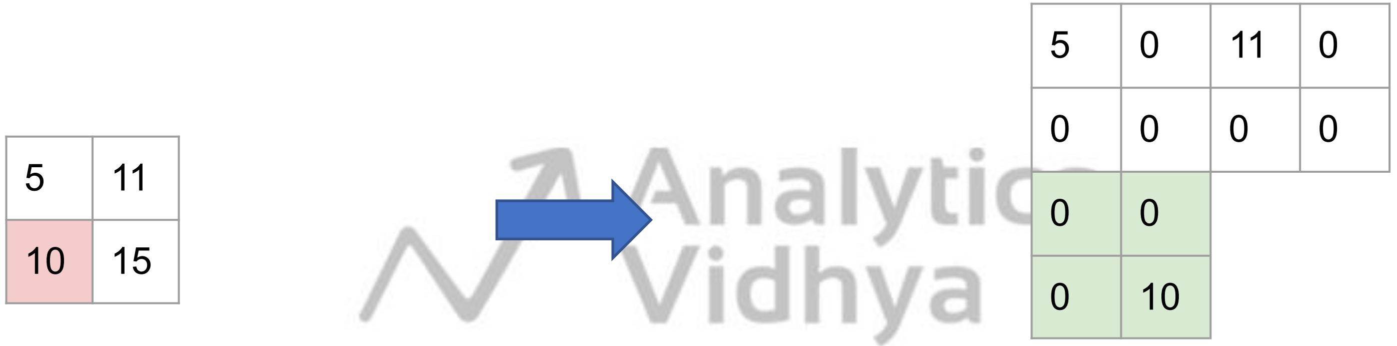
5	11
10	15

2 x 2

Filter size = 2 x 2

5	0	11	0
0	0	0	0

Method 3: Max Unpooling Operation



2 x 2

Filter size = 2 x 2

Method 3: Max Unpooling Operation

5	11
10	15

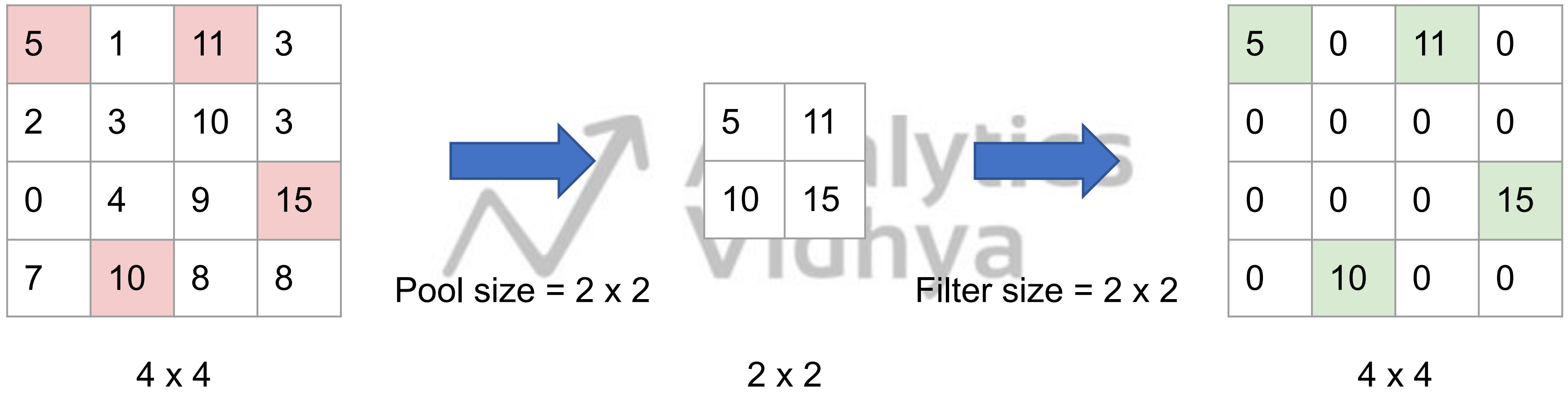
2 x 2

Filter size = 2 x 2

5	0	11	0
0	0	0	0
0	0	0	15
0	10	0	0

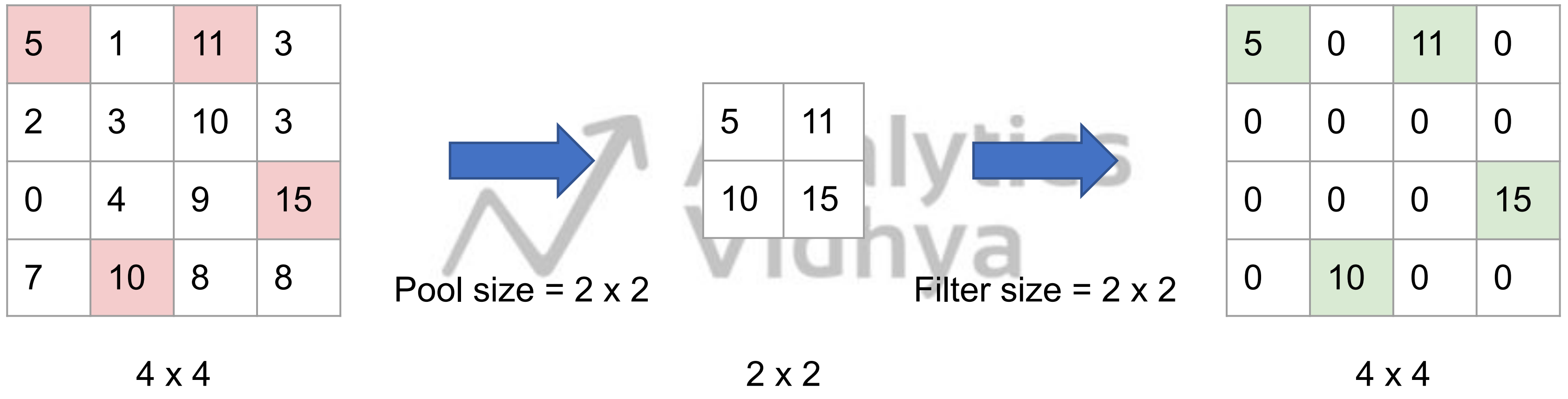
4 x 4

Method 3: Max Unpooling Operation



Method 3: Max Unpooling Operation

Max UnPooling is also called
In Network Upsampling



5	1	11	3
2	3	10	3
0	4	9	15
7	10	8	8

5	11
10	15

5	0	11	0
0	0	0	0
0	0	0	15
0	10	0	0

What do we need?

- Feasible but still computationally expensive

➔ Inverse Operation for Pooling (Bilinear Interpolation, Max Unpooling)

What do we need?

- Feasible but still computationally expensive

→ Inverse Operation for Pooling (Bilinear Interpolation, Max Unpooling)

- Options for DL architecture is limited

What do we need?

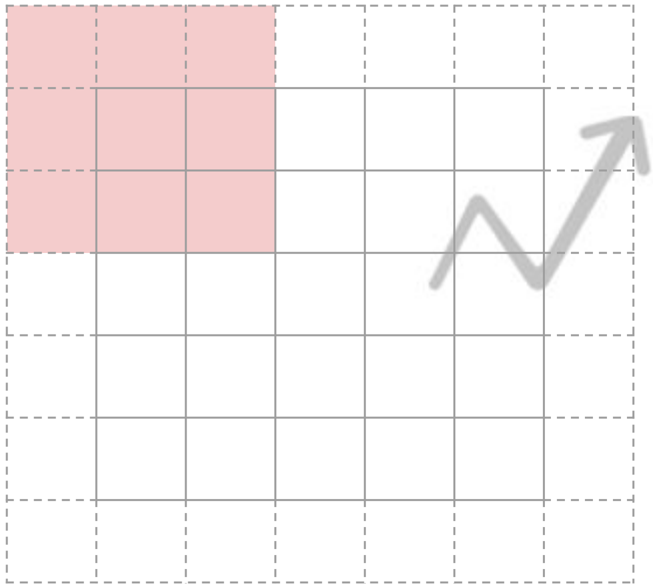
- Feasible but still computationally expensive

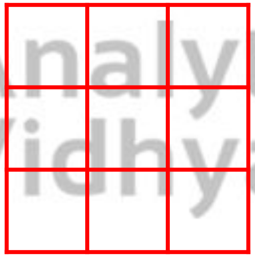
➔ Inverse Operation for Pooling (Bilinear Interpolation, Max Unpooling)

- Options for DL architecture is limited

➔ A Convolution Operation which increases size of output

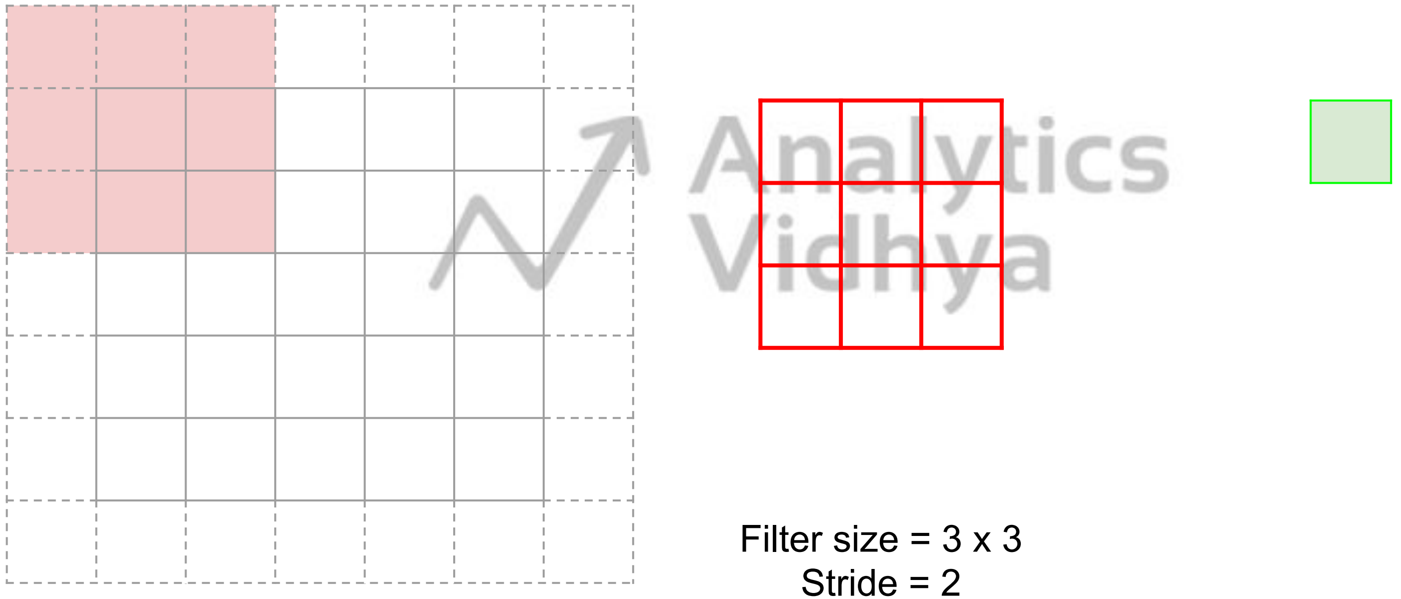
Recap - Convolution Operation


5 x 5



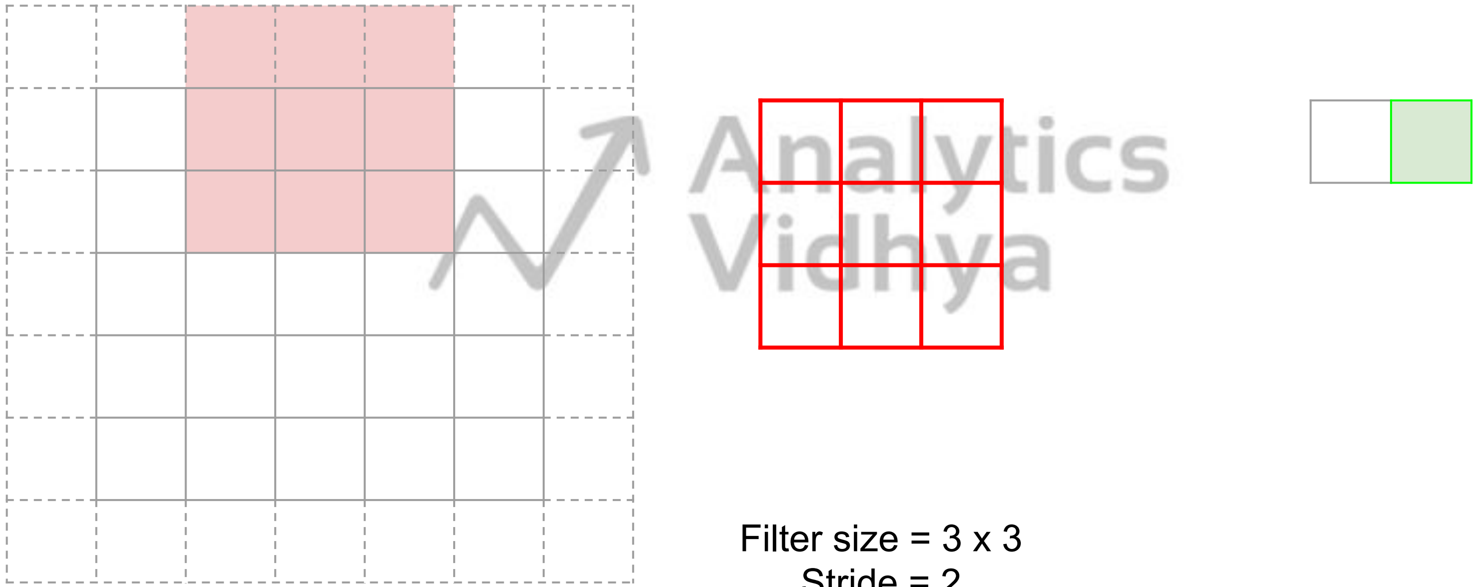
Filter size = 3 x 3
Stride = 2
Padding = 1

Recap - Convolution Operation



Filter size = 3 x 3
Stride = 2
Padding = 1

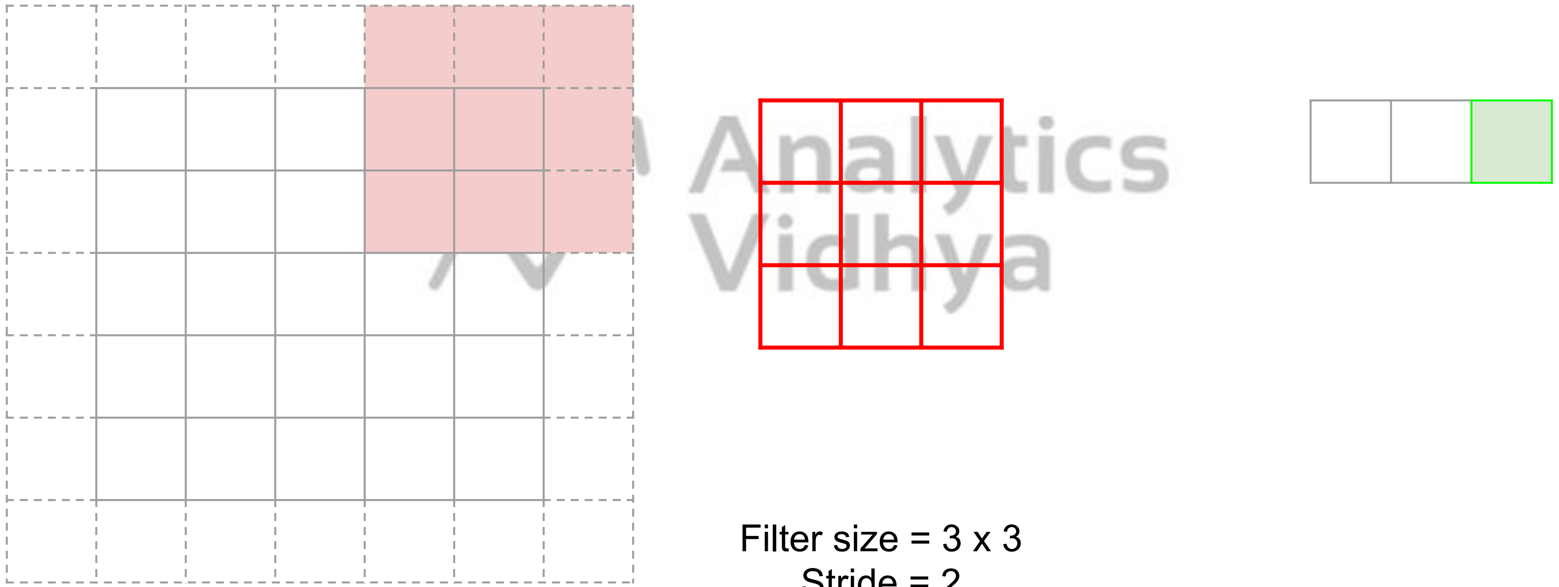
Recap - Convolution Operation



5 x 5

Filter size = 3 x 3
Stride = 2
Padding = 1

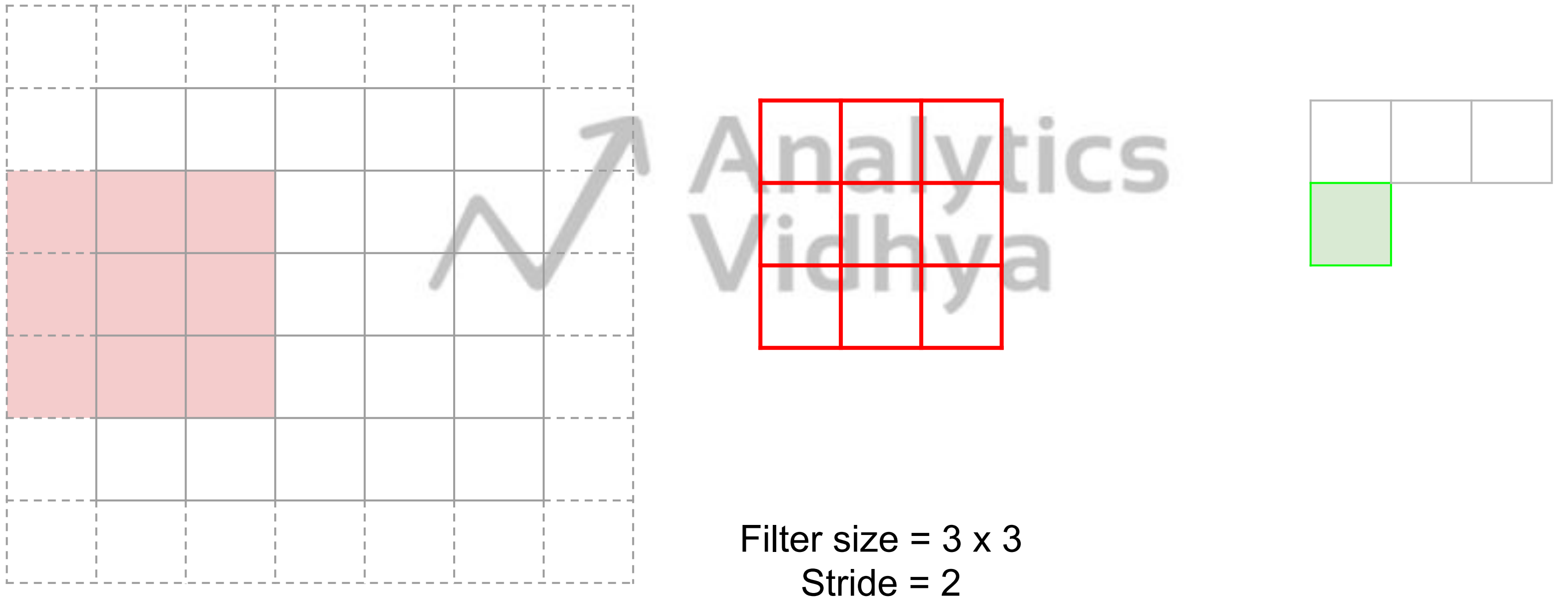
Recap - Convolution Operation



5 x 5

Filter size = 3 x 3
Stride = 2
Padding = 1

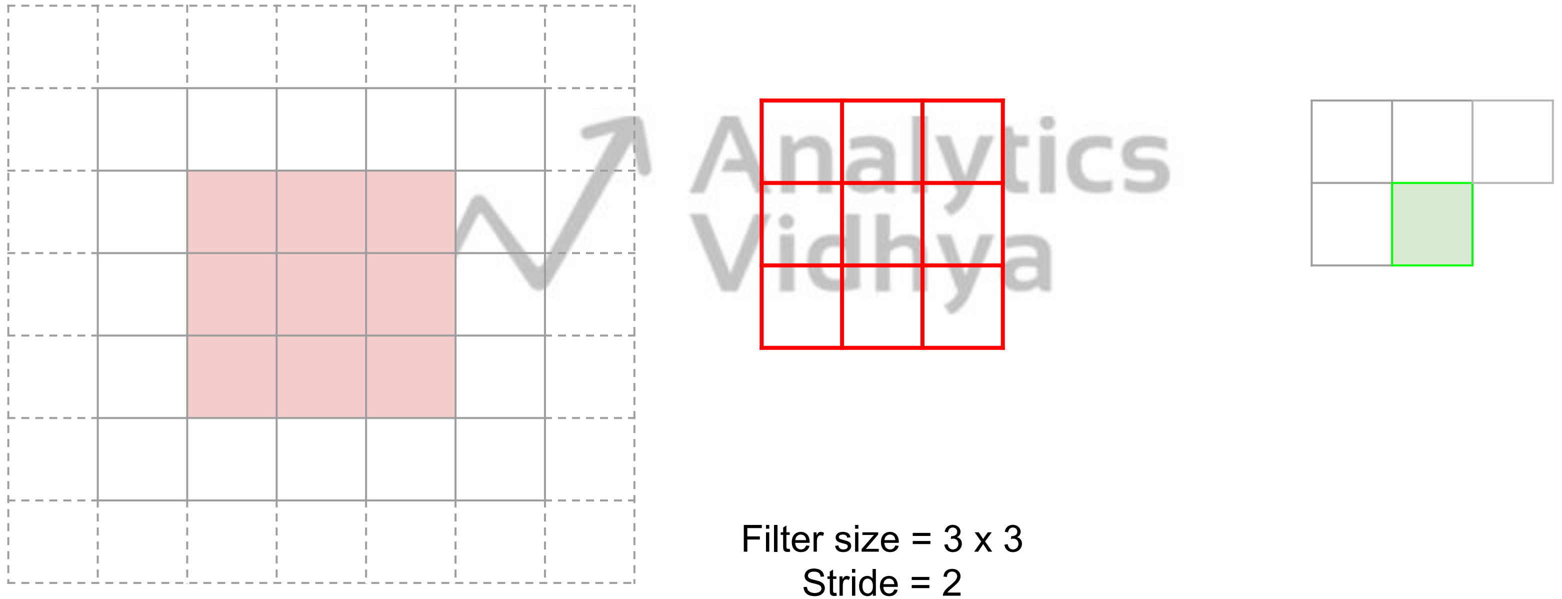
Recap - Convolution Operation



5 x 5

Filter size = 3 x 3
Stride = 2
Padding = 1

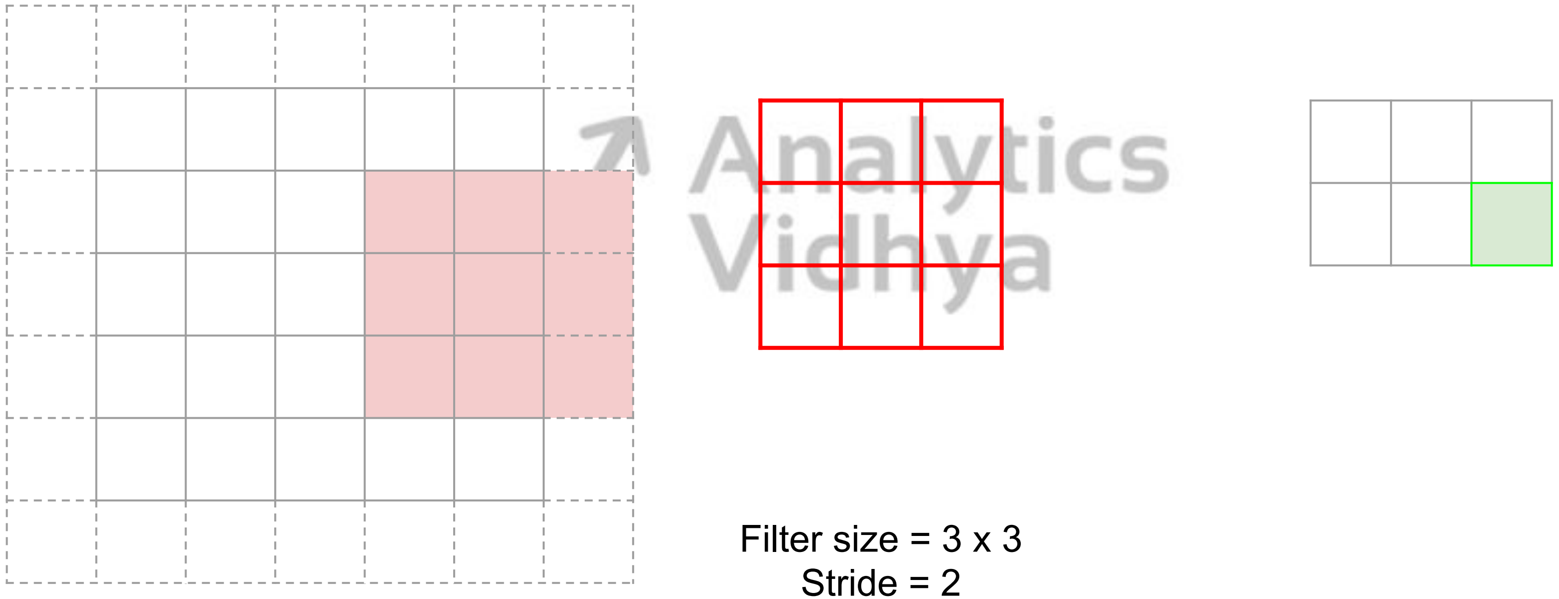
Recap - Convolution Operation



5 x 5

Filter size = 3 x 3
Stride = 2
Padding = 1

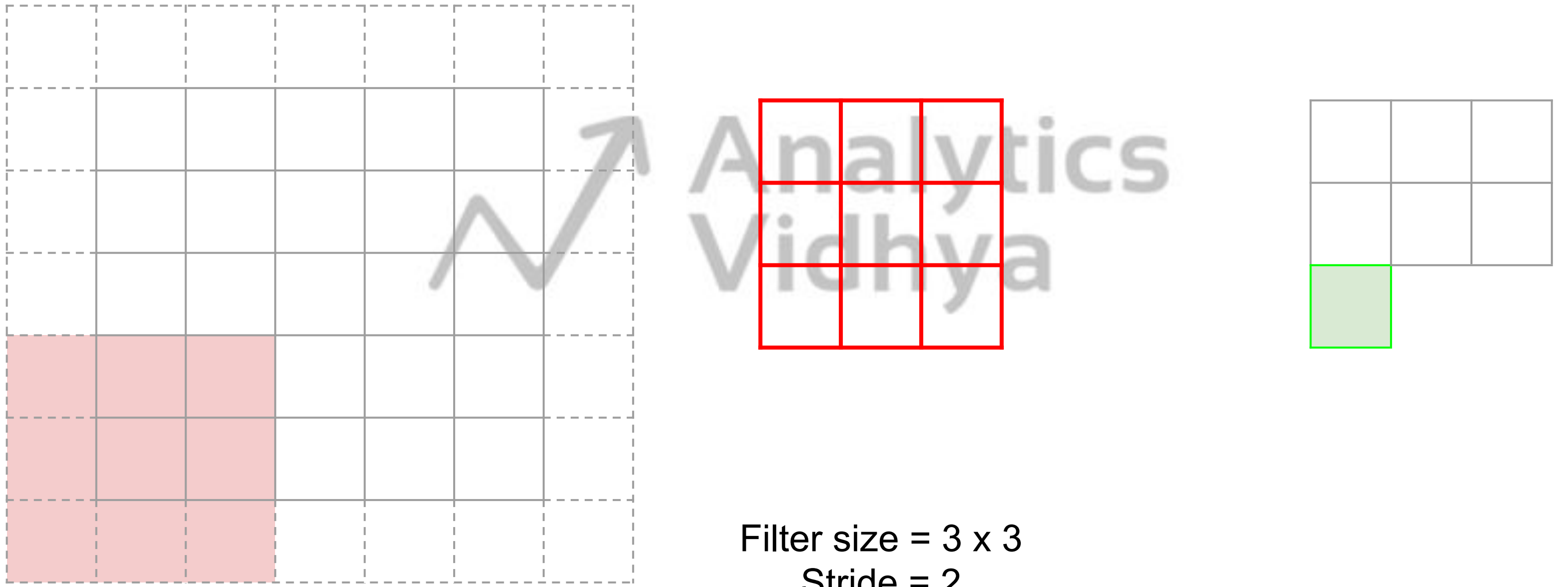
Recap - Convolution Operation



5 x 5

Filter size = 3 x 3
Stride = 2
Padding = 1

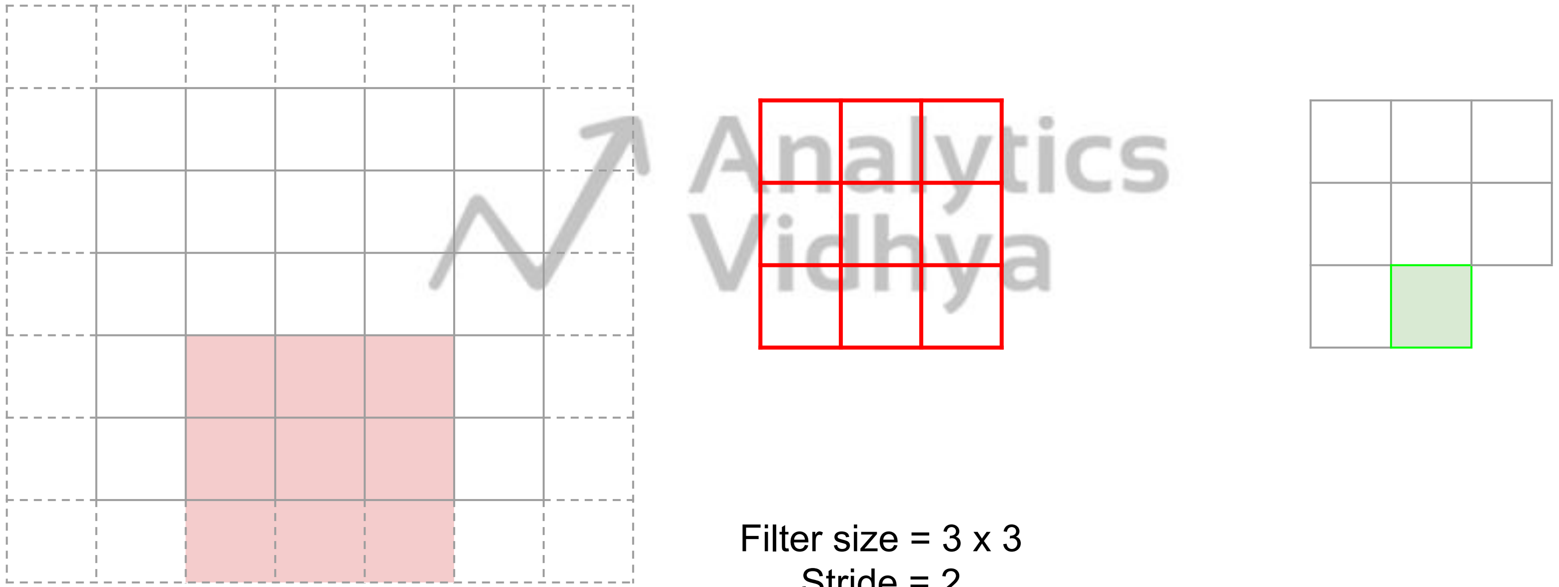
Recap - Convolution Operation



5 x 5

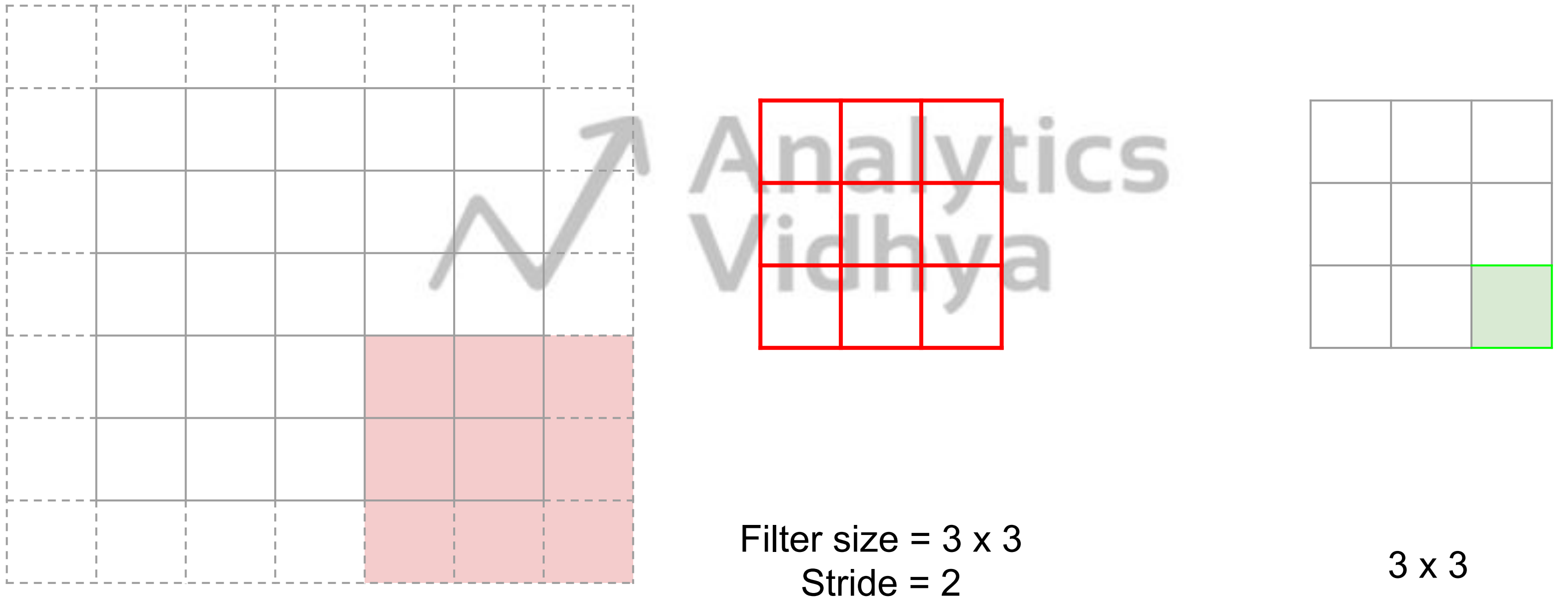
Filter size = 3 x 3
Stride = 2
Padding = 1

Recap - Convolution Operation



Filter size = 3 x 3
Stride = 2
Padding = 1

Recap - Convolution Operation



5 x 5

Filter size = 3 x 3
Stride = 2
Padding = 1

3 x 3

Transpose Convolution Operation

1	2	3
4	5	6
7	8	9

Input
3 x 3

0	0	0
1	1	1
0	0	0

Filter size = 3 x 3
Stride = 2
Padding = 1

Transpose Convolution Operation

1	2	3
4	5	6
7	8	9

Input
3 x 3

1	1	1
1	1	1
1	1	1

Broadcast
3 x 3

0	0	0
1	1	1
0	0	0

Filter size = 3 x 3
Stride = 2
Padding = 1

Transpose Convolution Operation

1	2	3
4	5	6
7	8	9

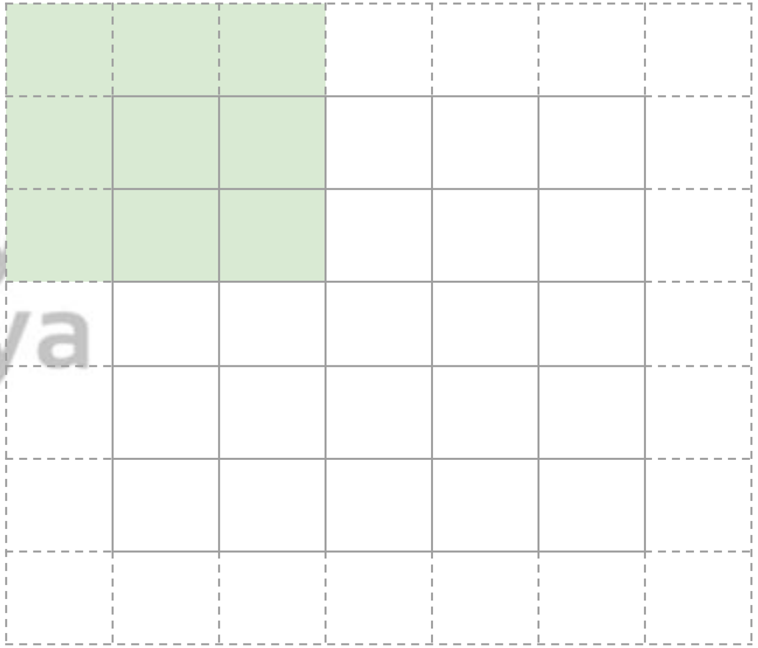
Input
3 x 3

1	1	1
1	1	1
1	1	1

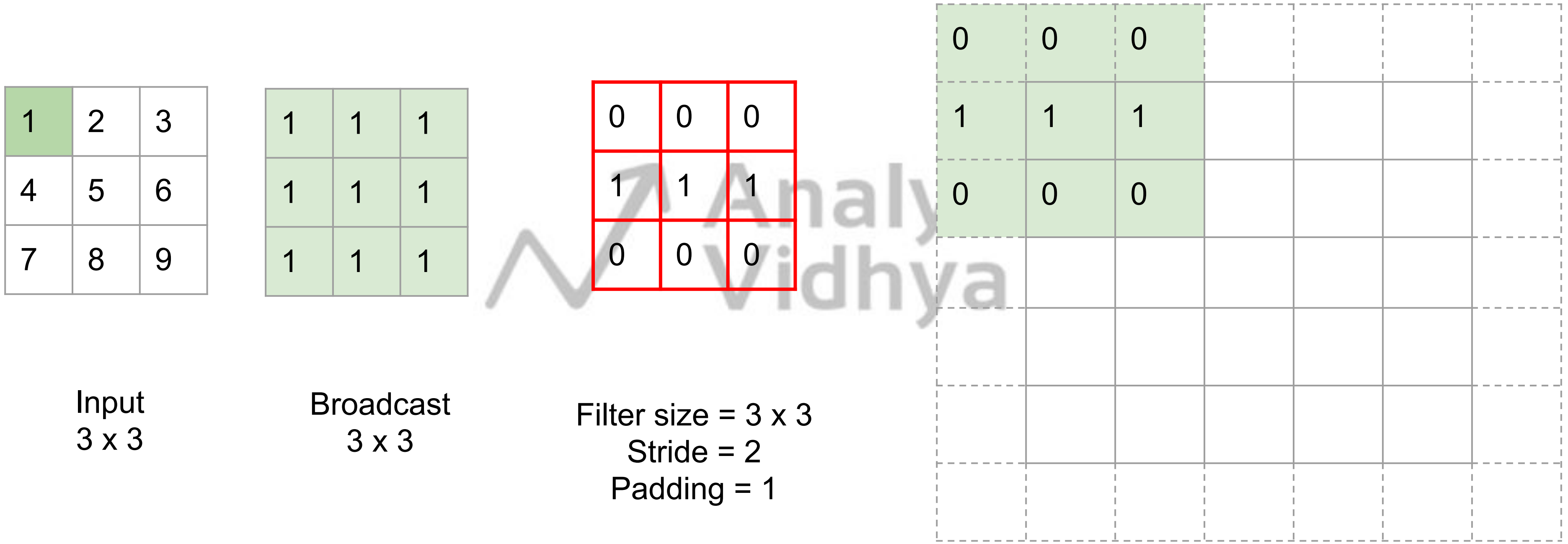
Broadcast
3 x 3

0	0	0
1	1	1
0	0	0

Filter size = 3 x 3
Stride = 2
Padding = 1



Transpose Convolution Operation



Input
3 x 3

Broadcast
3 x 3

Filter size = 3 x 3
Stride = 2
Padding = 1

Transpose Convolution Operation

1	2	3
4	5	6
7	8	9

Input
3 x 3

2	2	2
2	2	2
2	2	2

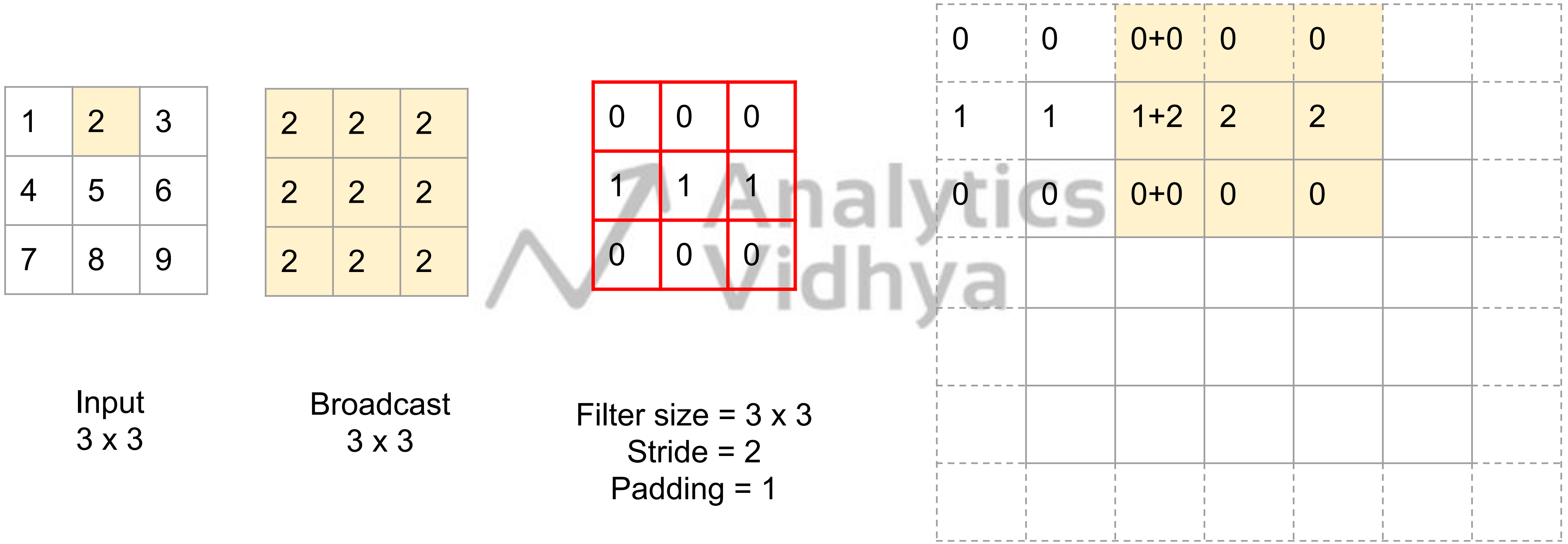
Broadcast
3 x 3

0	0	0
1	1	1
0	0	0

Filter size = 3 x 3
Stride = 2
Padding = 1

0	0	0				
1	1	1				
0	0	0				

Transpose Convolution Operation



Transpose Convolution Operation

1	2	3
4	5	6
7	8	9

Input
3 x 3

3	3	3
3	3	3
3	3	3

Broadcast
3 x 3

0	0	0
1	1	1
0	0	0

Filter size = 3 x 3
Stride = 2
Padding = 1

0	0	0	0	0		
1	1	1+2	2	2		
0	0	0	0	0		

Transpose Convolution Operation

1	2	3
4	5	6
7	8	9

Input
3 x 3

3	3	3
3	3	3
3	3	3

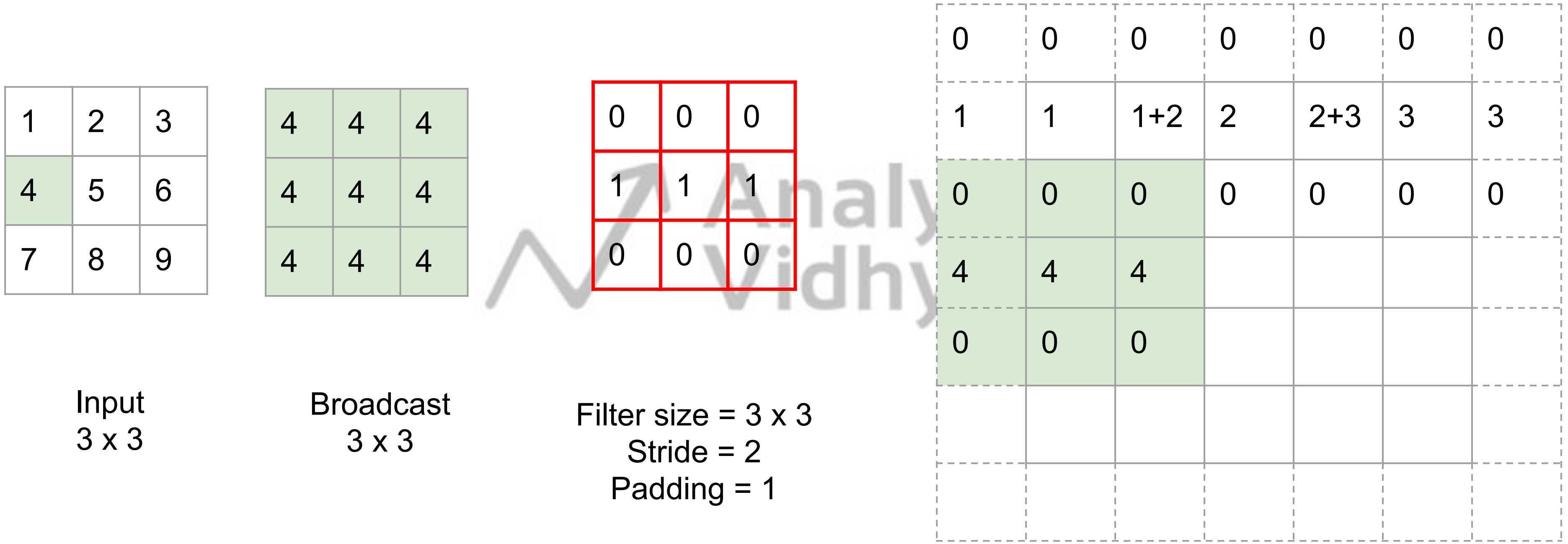
Broadcast
3 x 3

0	0	0
1	1	1
0	0	0

Filter size = 3 x 3
Stride = 2
Padding = 1

0	0	0	0	0	0	0
1	1	1+2	2	2+3	3	3
0	0	0	0	0	0	0

Transpose Convolution Operation



1	2	3
4	5	6
7	8	9

Input
3 x 3

4	4	4
4	4	4
4	4	4

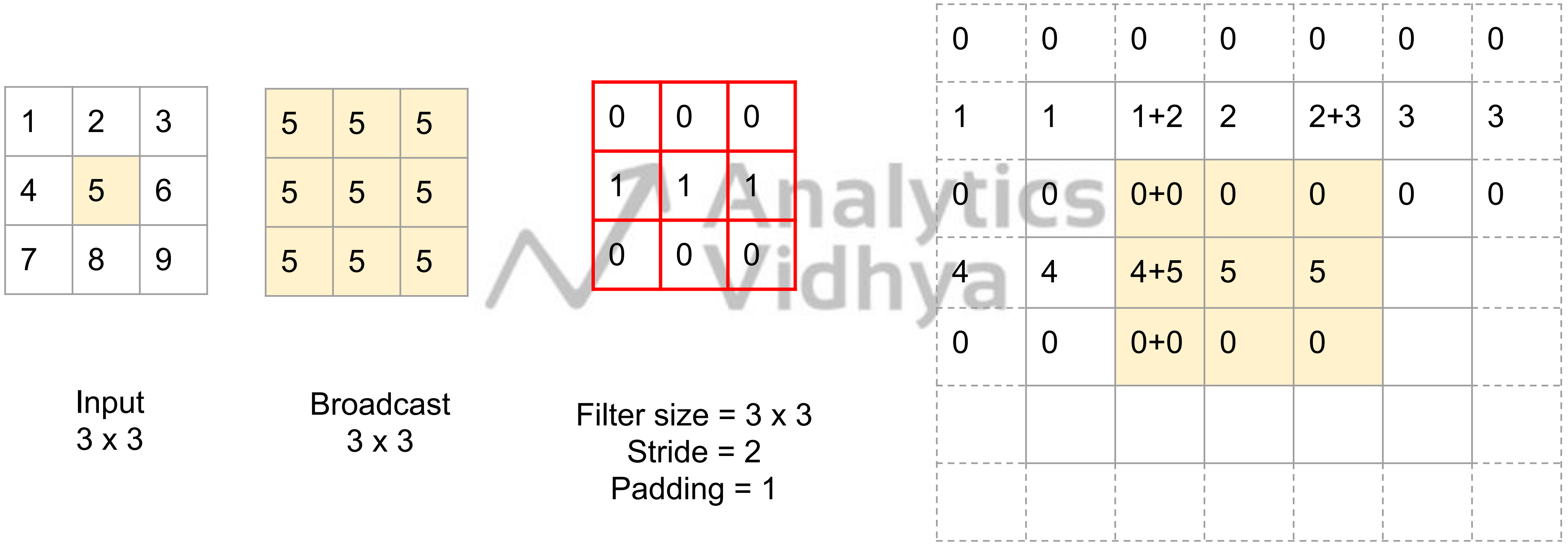
Broadcast
3 x 3

0	0	0
1	1	1
0	0	0

Filter size = 3 x 3
Stride = 2
Padding = 1

0	0	0	0	0	0	0
1	1	1+2	2	2+3	3	3
0	0	0	0	0	0	0
4	4	4				
0	0	0				

Transpose Convolution Operation



1	2	3
4	5	6
7	8	9

Input
3 x 3

5	5	5
5	5	5
5	5	5

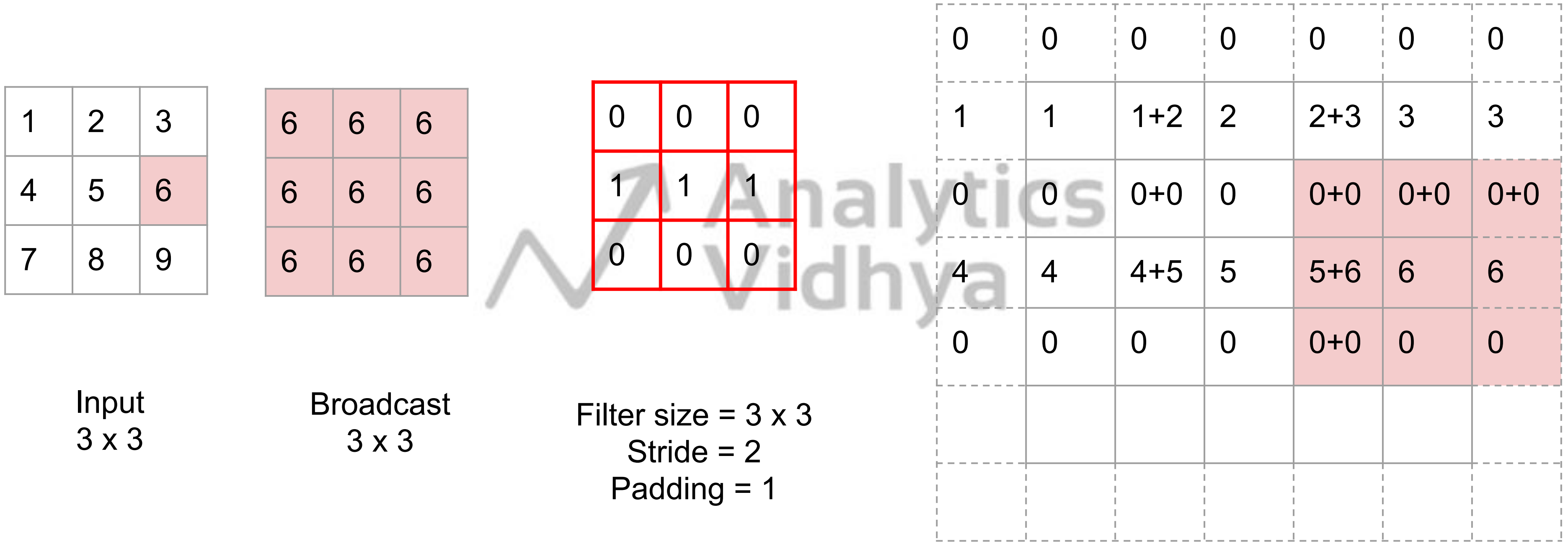
Broadcast
3 x 3

0	0	0
1	1	1
0	0	0

Filter size = 3 x 3
Stride = 2
Padding = 1

0	0	0	0	0	0	0
1	1	1+2	2	2+3	3	3
0	0	0+0	0	0	0	0
4	4	4+5	5	5		
0	0	0+0	0	0		

Transpose Convolution Operation



1	2	3
4	5	6
7	8	9

Input
3 x 3

6	6	6
6	6	6
6	6	6

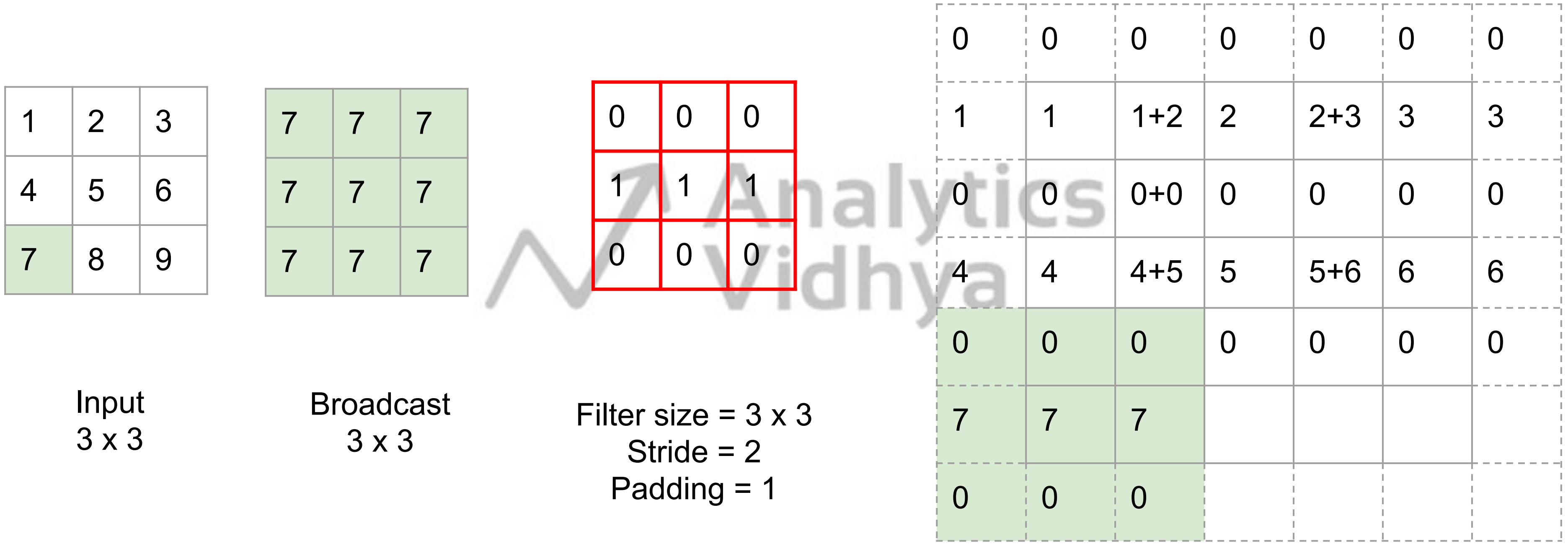
Broadcast
3 x 3

0	0	0
1	1	1
0	0	0

Filter size = 3 x 3
Stride = 2
Padding = 1

0	0	0	0	0	0	0
1	1	1+2	2	2+3	3	3
0	0	0+0	0	0+0	0+0	0+0
4	4	4+5	5	5+6	6	6
0	0	0	0	0+0	0	0

Transpose Convolution Operation



1	2	3
4	5	6
7	8	9

Input
3 x 3

7	7	7
7	7	7
7	7	7

Broadcast
3 x 3

0	0	0
1	1	1
0	0	0

Filter size = 3 x 3
Stride = 2
Padding = 1

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

Transpose Convolution Operation

1	2	3
4	5	6
7	8	9

Input
3 x 3

8	8	8
8	8	8
8	8	8

Broadcast
3 x 3

0	0	0
1	1	1
0	0	0

Filter size = 3 x 3
Stride = 2
Padding = 1

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

Transpose Convolution Operation

1	2	3
4	5	6
7	8	9

Input
3 x 3

9	9	9
9	9	9
9	9	9

Broadcast
3 x 3

0	0	0
1	1	1
0	0	0

Filter size = 3 x 3
Stride = 2
Padding = 1

0	0	0	0	0	0	0
1	1	1+2	2	2+3	3	3
0	0	0+0	0	0	0	0
4	4	4+5	5	5+6	6	6
0	0	0	0	0+0	0+0	0+0
7	7	7+8	8	8+9	9	9
0	0	0	0	0	0	0

Transpose Convolution Operation

1	2	3
4	5	6
7	8	9

Input
3 x 3

9	9	9
9	9	9
9	9	9

Broadcast
3 x 3

0	0	0
1	1	1
0	0	0

Filter size = 3 x 3
Stride = 2
Padding = 1

1	3	2	5	3
0	0	0	0	0
4	9	5	11	6
0	0	0	0	0
7	15	8	17	9

Output - 5 x 5

Transpose Convolution Operation

Transpose Convolution is also called
Fractional strided Convolution

1	2	3
4	5	6
7	8	9

Input
3 x 3

0	0	0
1	1	1
0	0	0

Filter size = 3 x 3
Stride = 2
Padding = 1

1	3	2	5	3
0	0	0	0	0
4	9	5	11	6
0	0	0	0	0
7	15	8	17	9

Output - 5 x 5

Transpose Convolution Operation

Transpose Convolution is also called
Learnable Upsampling

1	2	3
4	5	6
7	8	9

Input
3 x 3

0	0	0
1	1	1
0	0	0

Filter size = 3 x 3
Stride = 2
Padding = 1

1	3	2	5	3
0	0	0	0	0
4	9	5	11	6
0	0	0	0	0
7	15	8	17	9

Output - 5 x 5

What do we need?

- Feasible but still computationally expensive

➔ Inverse Operation for Pooling (Bilinear Interpolation, Max Unpooling)

- Options for DL architecture is limited

➔ A Convolution Operation which increases size of output (Transpose Convolution)

What do we need?

- Feasible but still computationally expensive

➔ Inverse Operation for Pooling (Bilinear Interpolation, Max Unpooling)

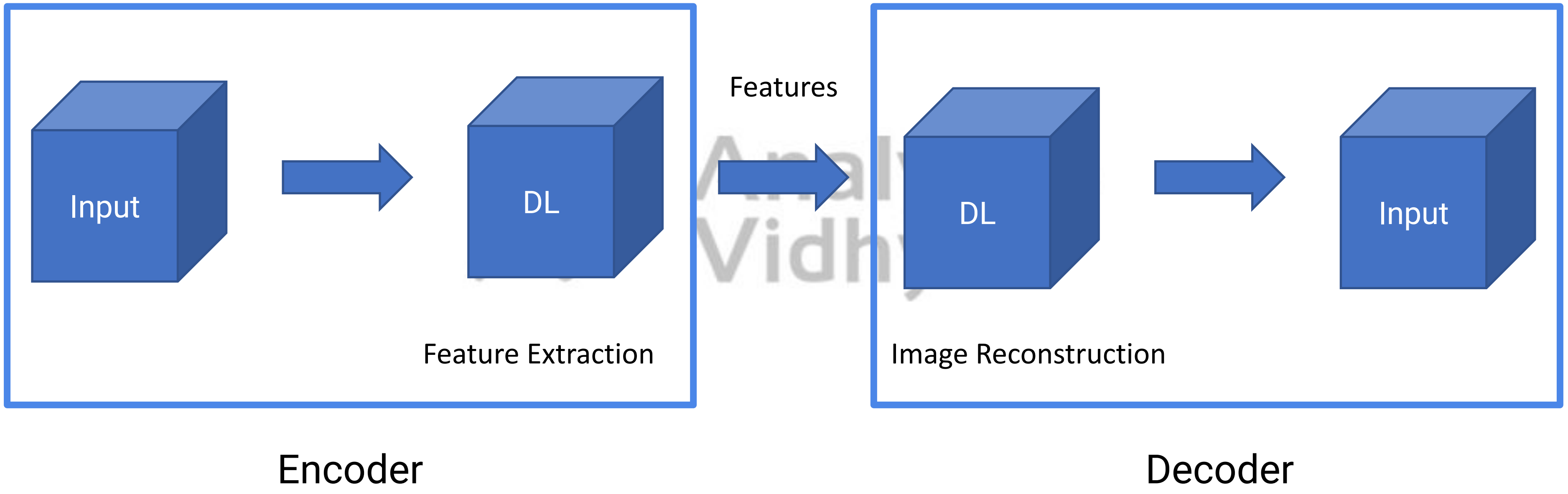
- Options for DL architecture is limited

➔ A Convolution Operation which increases size of output (Transpose Convolution)

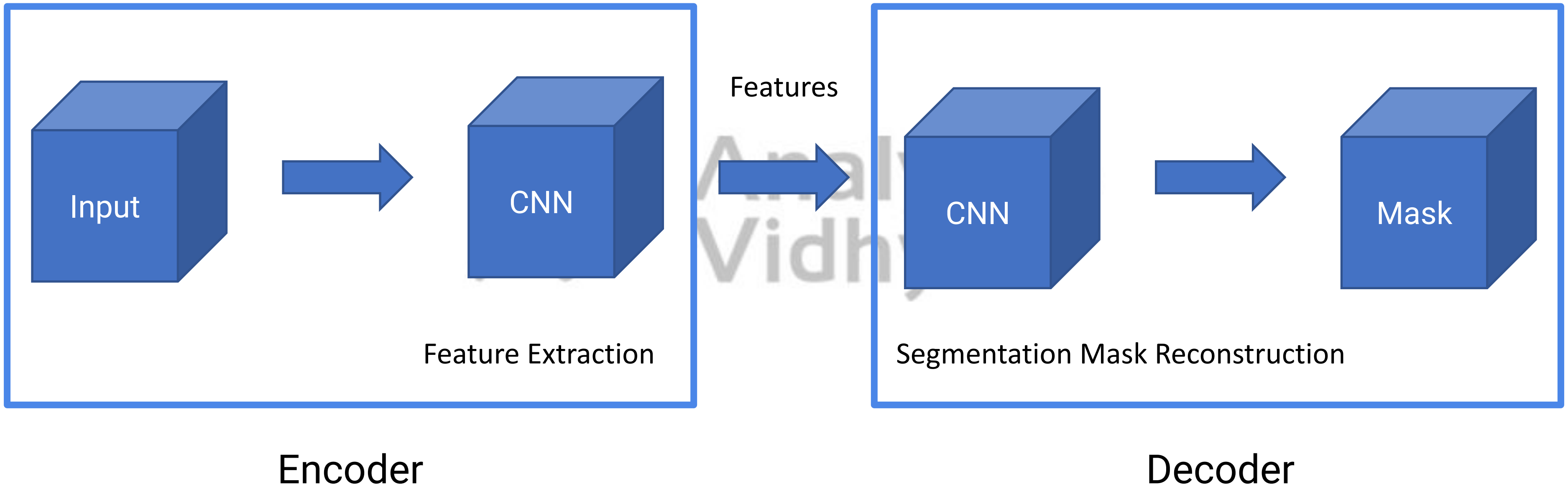
- Simplistic DL model, doesn’t take ideas from complex networks

➔ Better Deep Learning architecture

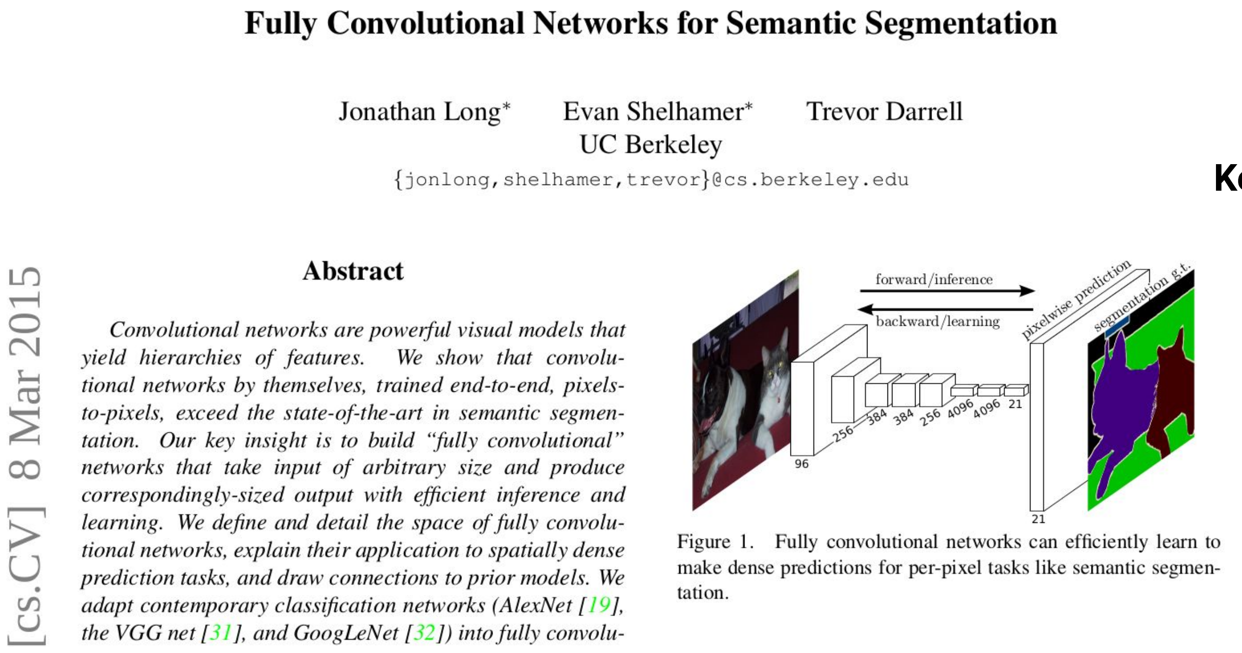
Recap - Encoder Decoder Architecture for AutoEncoders



Encoder Decoder Architecture for Image Segmentation



Fully Convolutional Network (FCN) Architecture



Fully Convolutional Networks for Semantic Segmentation

Jonathan Long* Evan Shelhamer* Trevor Darrell
UC Berkeley
{jonlong,shelhamer,trevor}@cs.berkeley.edu

Abstract

Convolutional networks are powerful visual models that yield hierarchies of features. We show that convolutional networks by themselves, trained end-to-end, pixels-to-pixels, exceed the state-of-the-art in semantic segmentation. Our key insight is to build "fully convolutional" networks that take input of arbitrary size and produce correspondingly-sized output with efficient inference and learning. We define and detail the space of fully convolutional networks, explain their application to spatially dense prediction tasks, and draw connections to prior models. We adapt contemporary classification networks (AlexNet [19], the VGG net [31], and GoogLeNet [32]) into fully convolu-

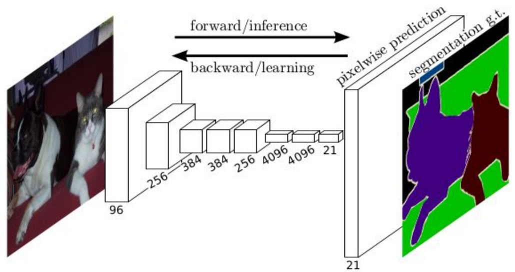
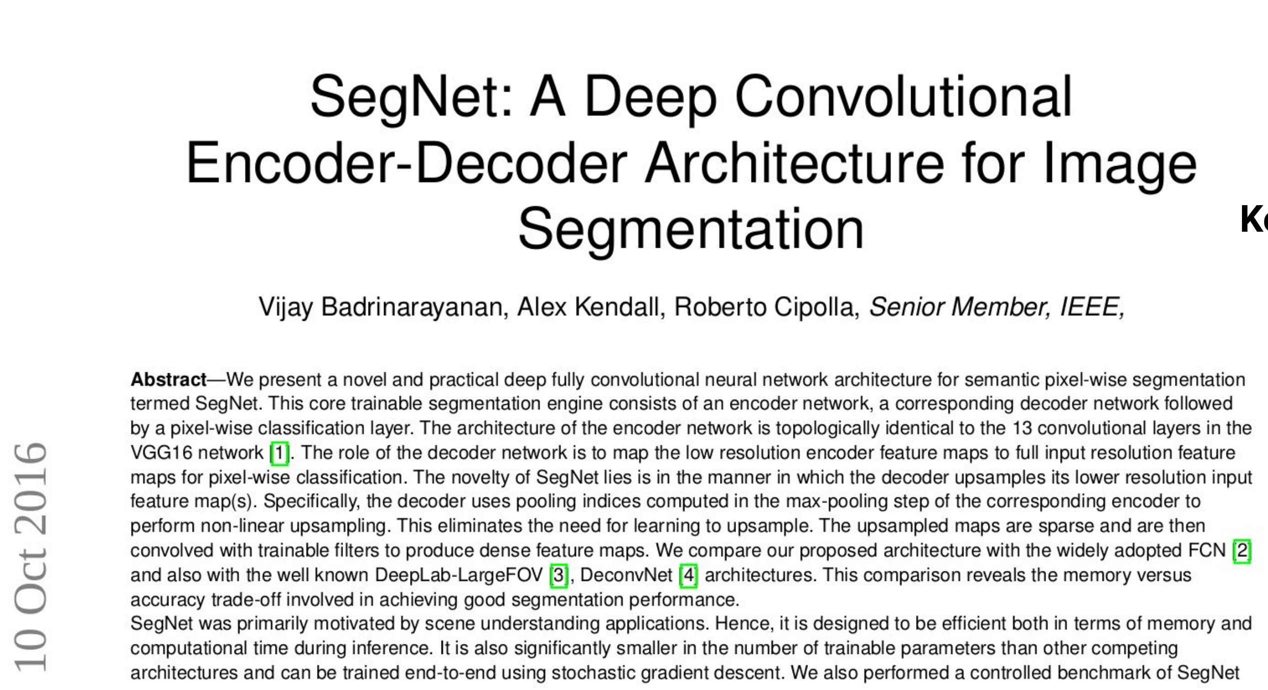


Figure 1. Fully convolutional networks can efficiently learn to make dense predictions for per-pixel tasks like semantic segmentation.

Key Takeaways:

- Upsampling using Interpolation or Transpose Convolution
- Transfer Learning

Segmentation Network (SegNet) Architecture



1

SegNet: A Deep Convolutional Encoder-Decoder Architecture for Image Segmentation

Vijay Badrinarayanan, Alex Kendall, Roberto Cipolla, *Senior Member, IEEE,*

Abstract—We present a novel and practical deep fully convolutional neural network architecture for semantic pixel-wise segmentation termed SegNet. This core trainable segmentation engine consists of an encoder network, a corresponding decoder network followed by a pixel-wise classification layer. The architecture of the encoder network is topologically identical to the 13 convolutional layers in the VGG16 network [1]. The role of the decoder network is to map the low resolution encoder feature maps to full input resolution feature maps for pixel-wise classification. The novelty of SegNet lies is in the manner in which the decoder upsamples its lower resolution input feature map(s). Specifically, the decoder uses pooling indices computed in the max-pooling step of the corresponding encoder to perform non-linear upsampling. This eliminates the need for learning to upsample. The upsampled maps are sparse and are then convolved with trainable filters to produce dense feature maps. We compare our proposed architecture with the widely adopted FCN [2] and also with the well known DeepLab-LargeFOV [3], DeconvNet [4] architectures. This comparison reveals the memory versus accuracy trade-off involved in achieving good segmentation performance.
SegNet was primarily motivated by scene understanding applications. Hence, it is designed to be efficient both in terms of memory and computational time during inference. It is also significantly smaller in the number of trainable parameters than other competing architectures and can be trained end-to-end using stochastic gradient descent. We also performed a controlled benchmark of SegNet

10 Oct 2016

Key Takeaways:

- Encoder Decoder Network
- Upsampling using Max Unpooling
- Transfer Learning

Steps for Image Segmentation using End-to-End model

1. Data Loading and Preprocessing

1.1 Load the Data

1.2 Define custom dataset and dataloader

1.3 Data Exploration

2. Image Segmentation through End to End Deep Learning model

2.1 Define model architecture

2.2 Train the model

2.3 Calculate IoU score

Thank you

Code Walkthrough of
End-to-End Deep Learning Approach