Deep Learning Using TensorFlow

Dr. Ash Pahwa

Lesson 7:
Convolution Neural Network (CNN)
Lesson 7.3: Building Blocks of CNN
Convolution and Pooling

Outline

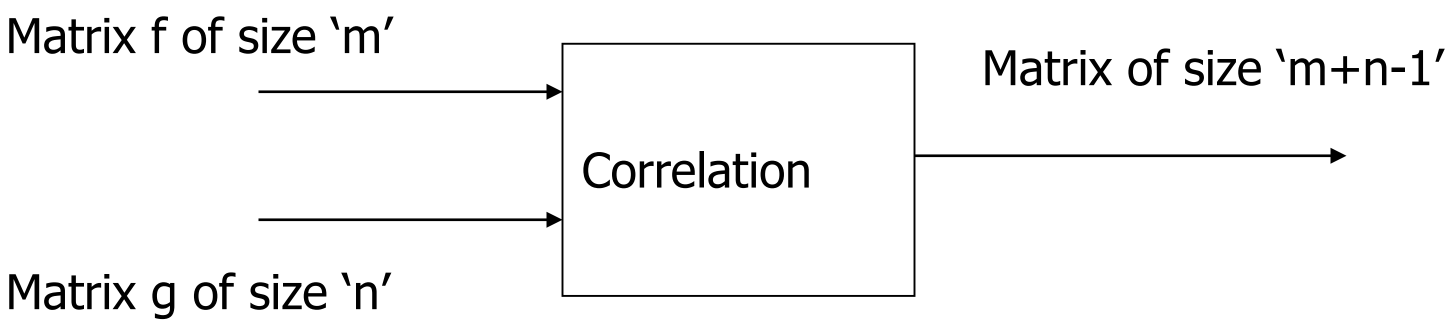
- Correlation Operator
- Convolution Operator
- Image Enhancement in Spatial Domain
- Types of Filters
  - Smoothing Filters: Gaussian Filter
  - Sharpening Filters: Laplacian Filter
- Pooling Layer

Operators

- These operate on
  - Image (Digital Image Processing)
  - Signals (Signal theory – Electrical Engineering / Physics)
  - Matrix / Functions (Math)
- Correlation
- Convolution
- Both can be defined over
  - 1 Dimensional matrix
  - 2 Dimensional matrix
- The only difference between correlation and convolution is that in convolution the kernel is inverted

Correlation Operator

- Slide one matrix over another
  - Over lapping numbers are multiplied
  - The response is the sum of all products



Example: 1 Dimensional Matrix

2 input matrices

f	2	9	16	24	2	95

Size= 6

g	2	8	2	1

Size = 4

Output matrix size = 6 + 4 – 1 = 9

Example Correlation

										Output
f				2	9	16	24	2	95	
g	2	8	2	1						
				2						2

f			2	9	16	24	2	95	
g	2	8	2	1					
			4	9					13

Example Correlation

Output

f		2	9	16	24	2	95
g	2	8	2	1			
		16	18	16			

50

f	2	9	16	24	2	95
g	2	8	2	1		
	4	72	32	24		

132

Example Correlation

Output

f

2	9	16	24	2	95

g

2	8	2	1

	18	128	48	2	

196

f

2	9	16	24	2	95

g

2	8	2	1

		32	192	4	95

323

Example Correlation

									Output
f	2	9	16	24	2	95			
g				2	8	2	1		
				48	16	190			254

f	2	9	16	24	2	95			
g					2	8	2	1	
					4	760			764

Example Correlation

Output

f

2	9	16	24	2	95

g

2	8	2	1

190

					190

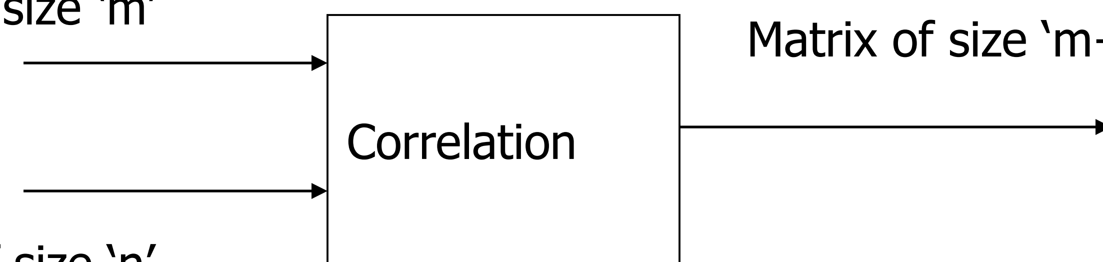
Correlation Operator

2	9	16	24	2	95

Matrix f of size ‘m’

2	13	50	132	196	323	254	764	190

Matrix of size ‘m+n-1’

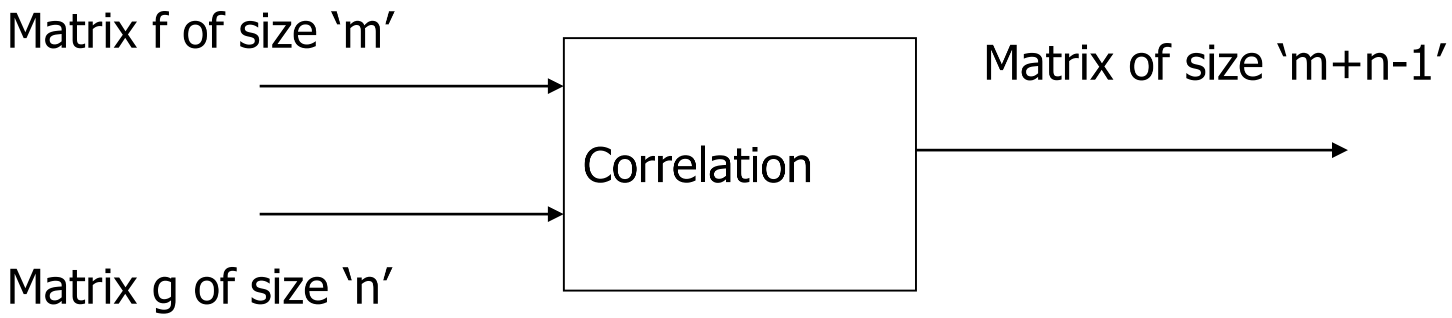


Matrix g of size ‘n’

2	8	2	1

Convolution Operator

- Flip the kernel
- Slide one matrix over another
  - Over lapping numbers are multiplied
  - The response is the sum of all products



Example:
1 Dimensional Matrix

2 input matrices

f

2	9	16	24	2	95

Size= 6

g

2	8	2	1

Size = 4

Output matrix size = 6 + 4 – 1 = 9

g

2	8	2	1

Example Convolution

Output

f

2	9	16	24	2	95

g

1	2	8	2

4					

4

f

2	9	16	24	2	95

g

1	2	8	2

16	18				

34

Example Convolution

								Output
f		2	9	16	24	2	95	
g	1	2	8	2				
		4	72	32				108

f	2	9	16	24	2	95	
g	1	2	8	2			
	2	18	128	48			196

Example Convolution

Output

f	2	9	16	24	2	95	
g		1	2	8	2		
		9	32	192	4		237

f	2	9	16	24	2	95	
g			1	2	8	2	
			16	48	16	190	270

Example Convolution

								Output
f	2	9	16	24	2	95		
g				1	2	8	2	
				24	4	760		788

f	2	9	16	24	2	95			
g					1	2	8	2	
					2	190			192

Example Convolution

Output

f	2	9	16	24	2	95			
g						1	2	8	2
						95		95	

Convolution Operator

2	9	16	24	2	95

Matrix f of size ‘m’

4	34	108	196	237	270	788	192	95

Matrix of size ‘m+n-1’

Convolution

Matrix g of size ‘n’

2	8	2	1

Python: numpy Convolution: 1D

```
################################################
# Convolution operation in Python
#
################################################
# 1.1 Load the libraries
#
import numpy as np
################################################
# Example#1
#
f = [2,9,16,24,2,95]
g = [2,8,2,1]
h1 = np.convolve(f,g)
h1
Out[14]: array([  4,  34, 108, 196, 237, 270, 788, 192,  95])

h2 = np.convolve(f,g,"full")
h2
Out[16]: array([  4,  34, 108, 196, 237, 270, 788, 192,  95])

h3 = np.convolve(f,g,"valid")
h3
Out[18]: array([196, 237, 270])
```

2	9	16	24	2	95

2	8	2	1

4	34	108	196	237	270	788	192	95

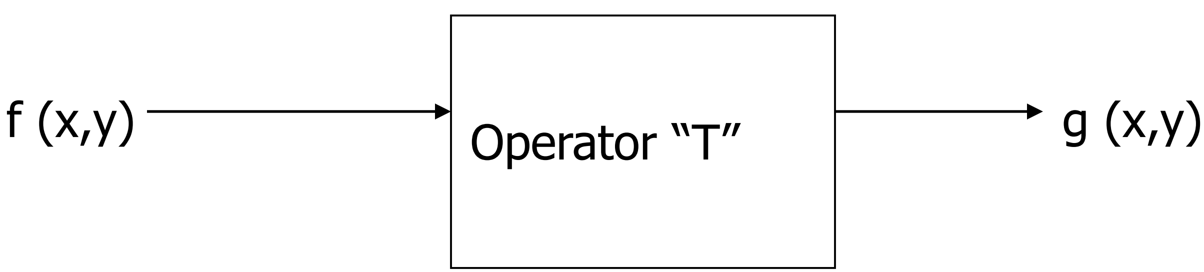
Image Enhancement in the Spatial Domain

Spatial Filtering : Convolution

Spatial Domain and Transformation

- Spatial Domain
  - Set of intensity values of an image
- Suppose “T” is an operator that is applied to all the pixel values of an image generating a new image.

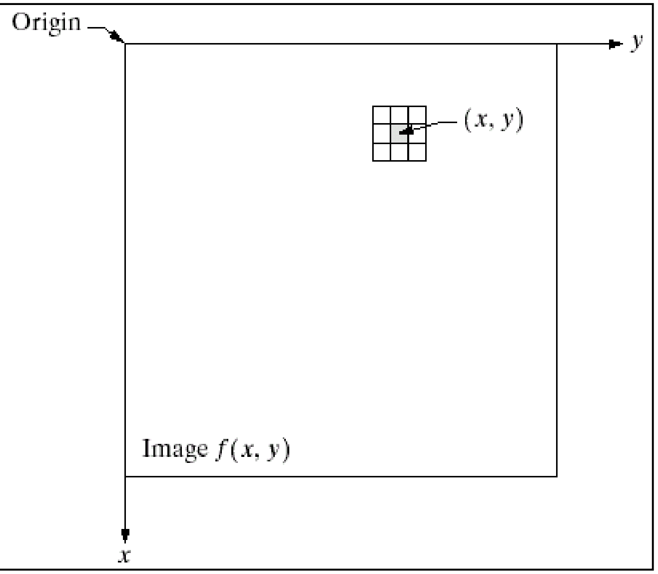
$$g(x,y) = T\,[f(x,y)]$$



Neighborhood Pixels

- Operator "T" is applied to a neighborhood pixels of
  - f(x,y) of size n x n

FIGURE 3.1 A 3×3 neighborhood about a point (x, y) in an image.



Neighborhood n x n

- Suppose the neighborhood size is n x n
  - Neighborhood is called
    - Mask
    - Kernel
    - Template
    - Window
- This operation is called “convolution”

Spatial Filtering

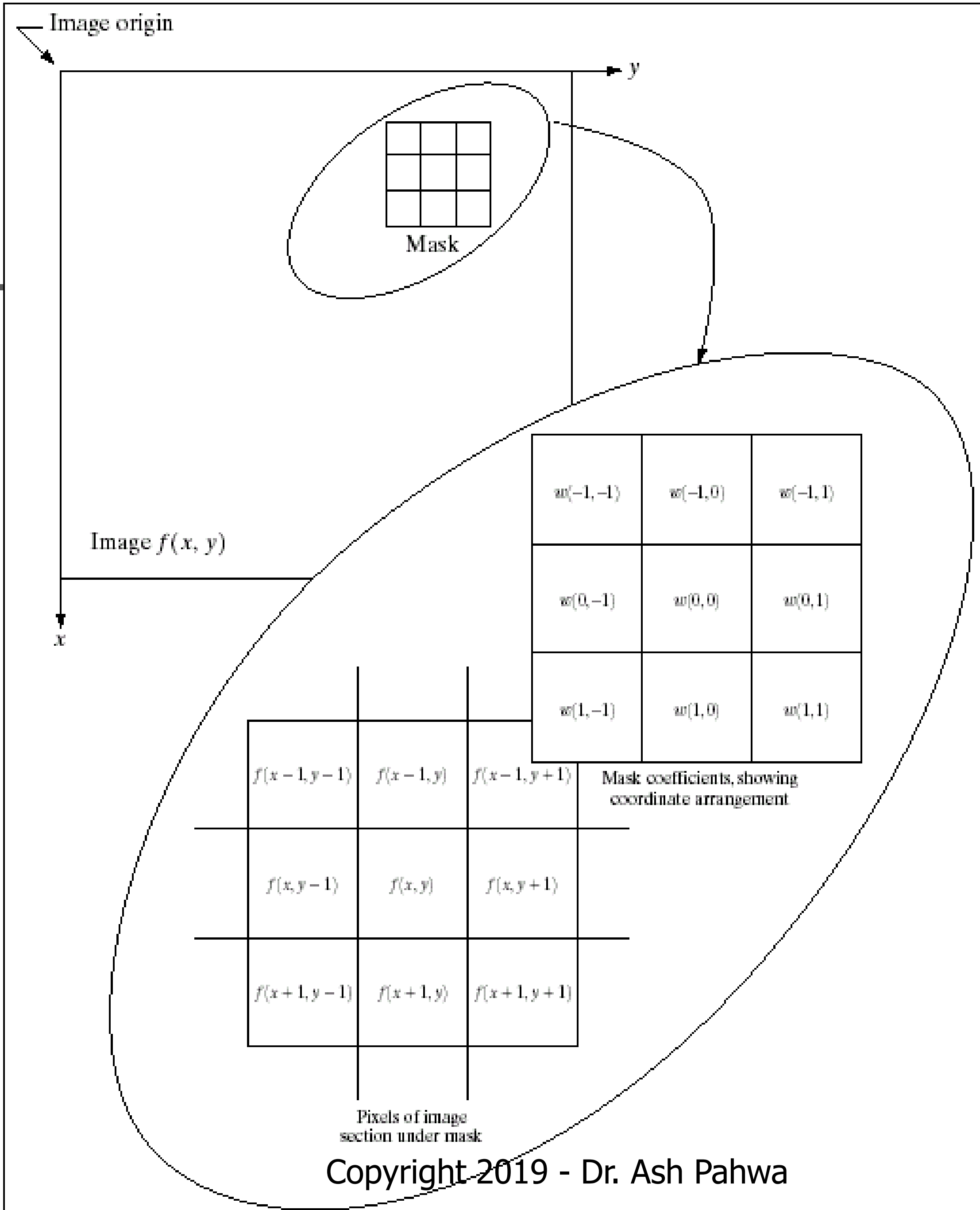


FIGURE 3.32 The mechanics of spatial filtering. The magnified drawing shows a 3×3 mask and the image section directly under it; the image section is shown displaced out from under the mask for ease of readability.

Input Image / Filter Output Image

a_{00}	a_{01}	a_{02}	a_{03}	a_{04}	a_{05}
a_{10}	a_{11}	a_{12}	a_{13}	a_{14}	a_{15}
a_{20}	a_{21}	a_{22}	a_{23}	a_{24}	a_{25}
a_{30}	a_{31}	a_{32}	a_{33}	a_{34}	a_{35}
a_{40}	a_{41}	a_{42}	a_{43}	a_{44}	a_{45}
a_{50}	a_{51}	a_{52}	a_{53}	a_{54}	a_{55}

f_{00}	f_{01}	f_{02}
f_{10}	f_{11}	f_{12}
f_{20}	f_{21}	f_{22}

$m \times n = 3 \times 3$

c_{00}	c_{01}	c_{02}	c_{03}	c_{04}	c_{05}
c_{10}	c_{11}	c_{12}	c_{13}	c_{14}	c_{15}
c_{20}	c_{21}	c_{22}	c_{23}	c_{24}	c_{25}
c_{30}	c_{31}	c_{32}	c_{33}	c_{34}	c_{35}
c_{40}	c_{41}	c_{42}	c_{43}	c_{44}	c_{45}
c_{50}	c_{51}	c_{52}	c_{53}	c_{54}	c_{55}

Spatial Filtering

a_{00} f_{00}	a_{01} f_{01}	a_{02} f_{02}	a_{03}	a_{04}	a_{05}
a_{10} f_{10}	a_{11} f_{11}	a_{12} f_{12}	a_{13}	a_{14}	a_{15}
a_{20} f_{20}	a_{21} f_{21}	a_{22} f_{22}	a_{23}	a_{24}	a_{25}
a_{30}	a_{31}	a_{32}	a_{33}	a_{34}	a_{35}
a_{40}	a_{41}	a_{42}	a_{43}	a_{44}	a_{45}
a_{50}	a_{51}	a_{52}	a_{53}	a_{54}	a_{55}

c11 =
(f00 * a00) +
(f01 * a01) +
(f02 * a02) +
(f10 * a10) +
(f11 * a11) +
(f12 * a12) +
(f20 * a20) +
(f21 * a21) +
(f22 * a22)

Spatial Filtering

a_{00}	a_{01} f_{00}	a_{02} f_{01}	a_{03} f_{02}	a_{04}	a_{05}
a_{10}	a_{11} f_{10}	a_{12} f_{11}	a_{13} f_{12}	a_{14}	a_{15}
a_{20}	a_{21} f_{20}	a_{22} f_{21}	a_{23} f_{22}	a_{24}	a_{25}
a_{30}	a_{31}	a_{32}	a_{33}	a_{34}	a_{35}
a_{40}	a_{41}	a_{42}	a_{43}	a_{44}	a_{45}
a_{50}	a_{51}	a_{52}	a_{53}	a_{54}	a_{55}

c12 =
(f00 * a01) +
(f01 * a02) +
(f02 * a03) +
(f10 * a11) +
(f11 * a12) +
(f12 * a13) +
(f20 * a21) +
(f21 * a22) +
(f22 * a23)

Spatial Filtering

a_{00}	a_{01}	f_{00} a_{02}	f_{01} a_{03}	f_{02} a_{04}	a_{05}
a_{10}	a_{11}	f_{10} a_{12}	f_{11} a_{13}	f_{12} a_{14}	a_{15}
a_{20}	a_{21}	f_{20} a_{22}	f_{21} a_{23}	f_{22} a_{24}	a_{25}
a_{30}	a_{31}	a_{32}	a_{33}	a_{34}	a_{35}
a_{40}	a_{41}	a_{42}	a_{43}	a_{44}	a_{45}
a_{50}	a_{51}	a_{52}	a_{53}	a_{54}	a_{55}

c13 =
(f00 * a02) +
(f01 * a03) +
(f02 * a04) +
(f10 * a12) +
(f11 * a13) +
(f12 * a14) +
(f20 * a22) +
(f21 * a23) +
(f22 * a24)

Output Image

- $c_{11} = (f_{00} * a_{00}) + (f_{01} * a_{01}) + (f_{02} * a_{02}) +$
  $(f_{10} * a_{10}) + (f_{11} * a_{11}) + (f_{12} * a_{12}) +$
  $(f_{20} * a_{20}) + (f_{21} * a_{21}) + (f_{22} * a_{22})$

- $c_{12} = (f_{00} * a_{01}) + (f_{01} * a_{02}) + (f_{02} * a_{03}) +$
  $(f_{10} * a_{11}) + (f_{11} * a_{12}) + (f_{12} * a_{13}) +$
  $(f_{20} * a_{21}) + (f_{21} * a_{22}) + (f_{22} * a_{23})$

- $c_{13} = \ldots$

Example - Input Image / Filter

1	4	6	10	14	12
18	20	26	25	13	10
6	5	4	3	1	2
2	4	5	10	12	26
38	25	49	24	26	30
2	40	36	44	25	13

1	2	1
2	4	2
1	2	1

$m \times n = 3 \times 3$

Example - Output Image

1	4	6	10	14	12
18	20	26	25	13	10
6	5	4	3	1	2
2	4	5	10	12	26
38	25	49	24	26	30
2	40	36	44	25	13

Input Image

1	2	1
2	4	2
1	2	1

Filter

?	?	?	?	?	?
?	*203*	*236*	*229*	*179*	?
?	*139*	*153*	*148*	*135*	?
?	*187*	*211*	*208*	*233*	?
?	*407*	*474*	*432*	*379*	?
?	?	?	?	?	?

Output Image

Computing Boundary Values
Strategy#1: FULL Convolution

1	2	1
2	4	2
1	2	1

1	4	6	10	14	12
18	20	26	25	13	10
6	5	4	3	1	2
2	4	5	10	12	26
38	25	49	24	26	30
2	40	36	44	25	13

Assuming the image values are 0 where filter does not overlap the image

Image size = 6 x 6
Final convoluted image = (6+2) x (6+2) = 8 x 8

The final image can be cropped
Final convoluted image = 6 x 6

Computing Boundary Values
Strategy#2: VALID Convolution

1 1	4 2	6 1	10	14	12
18 2	20 4	26 2	25	13	10
6 1	5 2	4 1	3	1	2
2	4	5	10	12	26
38	25	49	24	26	30
2	40	36	44	25	13

Move the filter over the image only

Image size = 6 x 6
Final convoluted image = (6-2) x (6-2) = 4 x 4

?	?	?	?	?	?
?	*203*	*236*	*229*	*179*	?
?	*139*	*153*	*148*	*135*	?
?	*187*	*211*	*208*	*233*	?
?	*407*	*474*	*432*	*379*	?
?	?	?	?	?	?

Python: scipy Convolution: 2D

?	?	?	?	?	?
?	*203*	*236*	*229*	*179*	?
?	*139*	*153*	*148*	*135*	?
?	*187*	*211*	*208*	*233*	?
?	*407*	*474*	*432*	*379*	?
?	?	?	?	?	?

```
#################################
# Convolution operation in Python
#################################
# 1.1 Load the libraries
#
from scipy import signal as sg
#################################
# Example#1
# 2 dimensional Convolution
#
I = [ [ 1,  4,  6, 10, 14, 12],
      [18, 20, 26, 25, 13, 10],
      [ 6,  5,  4,  3,  1,  2],
      [ 2,  4,  5, 10, 12, 26],
      [38, 25, 49, 24, 26, 30],
      [ 2, 40, 36, 44, 25, 13] ]

g = [ [1,2,1],
      [2,4,2],
      [1,2,1] ]
```

```
sg.convolve(I,g)
Out[13]:
array([[  1,   6,  15,  26,  40,  50,  38,  12],
       [ 20,  68, 114, 149, 169, 161, 109,  34],
       [ 43, 135, 203, 236, 229, 179, 109,  34],
       [ 32,  98, 139, 153, 148, 135, 107,  40],
       [ 48, 134, 187, 211, 208, 233, 219,  84],
       [ 80, 254, 407, 474, 432, 379, 287,  99],
       [ 42, 189, 373, 459, 421, 320, 188,  56],
       [  2,  44, 118, 156, 149, 107,  51,  13]])

sg.convolve(I,g,"valid")
Out[14]:
array([[203, 236, 229, 179],
       [139, 153, 148, 135],
       [187, 211, 208, 233],
       [407, 474, 432, 379]])
```

Python: TensorFlow Convolution: 2D

Create Image

```
#############################################
# Convolution operation in Python and TensorFlow
##############################################
# 1.1 Load the libraries
#
import tensorflow as tf
##############################################
input_list = tf.constant( [
      [ 1.,  4.,  6, 10, 14, 12],
      [18,  20,  26, 25, 13, 10],
      [ 6,   5,   4,  3,  1,  2],
      [ 2,   4,   5, 10, 12, 26],
      [38,  25,  49, 24, 26, 30],
      [ 2,  40,  36, 44, 25, 13] ])

input_list.shape
Out[11]: TensorShape([Dimension(6), Dimension(6)])

input = tf.reshape(input_list,[1,6,6,1])

input.shape
Out[13]: TensorShape([Dimension(1), Dimension(6), Dimension(6), Dimension(1)])
```

Image shape:
- Batch Size = 1
- Width
- Height
- Number of channels = 1

Change the shape of image
From: 6 x 6
To: 1 x 6 x 6 x 1

Python: TensorFlow Convolution: 2D

Create Filter

Filter shape:

- Width
- Height
- Number of channels = 1
- Numbers of Filters

Change the shape of filter
From: 3 x 3
To: 3 x 3 x 1 x 1

```
filter_list = tf.constant ([
      [1.,2.,1.],
      [2,4,2],
      [1,2,1] ])

filter_list.shape
Out[15]: TensorShape([Dimension(3), Dimension(3)])

filter = tf.reshape(filter_list,[3,3,1,1])

filter.shape
Out[17]: TensorShape([Dimension(3), Dimension(3), Dimension(1), Dimension(1)])
```

Python: TensorFlow Convolution: 2D

Convolution Operation

```
########################################################

op1 = tf.nn.conv2d(input,filter,strides=[1,1,1,1], padding='VALID')

op2 = tf.nn.conv2d(input,filter,strides=[1,1,1,1], padding='SAME')
```

Python: TensorFlow Convolution: 2D Run the DAG

1	4	6	10	14	12
18	20	26	25	13	10
6	5	4	3	1	2
2	4	5	10	12	26
38	25	49	24	26	30
2	40	36	44	25	13

1	2	1
2	4	2
1	2	1

```
with tf.Session() as sess:
    print("image")
    print(sess.run(input))

    print("filter")
    print(sess.run(filter))

image
[[[[  1.]
   [  4.]
   [  6.]
   [ 10.]
   [ 14.]
   [ 12.]]

  [[ 18.]
   [ 20.]
   [ 26.]
   [ 25.]
   [ 13.]
   [ 10.]]

  [[  6.]
   [  5.]
   [  4.]
   [  3.]
   [  1.]
   [  2.]]
```

```
  [[  2.]
   [  4.]
   [  5.]
   [ 10.]
   [ 12.]
   [ 26.]]

  [[ 38.]
   [ 25.]
   [ 49.]
   [ 24.]
   [ 26.]
   [ 30.]]

  [[  2.]
   [ 40.]
   [ 36.]
   [ 44.]
   [ 25.]
   [ 13.]]]]
```

```
filter
[[[[ 1.]]

   [[ 2.]]

   [[ 1.]]]


  [[[ 2.]]

   [[ 4.]]

   [[ 2.]]]


  [[[ 1.]]

   [[ 2.]]

   [[ 1.]]]]
```

Python: TensorFlow Convolution: 2D Run the DAG

?	?	?	?	?	?
?	203	236	229	179	?
?	139	153	148	135	?
?	187	211	208	233	?
?	407	474	432	379	?
?	?	?	?	?	?

```
with tf.Session() as sess:
    print("Result VALID\n")
    print(sess.run(op1))
    print("\n")

    print("Result SAME\n")
    print(sess.run(op2))
    print("\n")


Result VALID

[[[[ 203.]
   [ 236.]
   [ 229.]
   [ 179.]]

  [[ 139.]
   [ 153.]
   [ 148.]
   [ 135.]]

  [[ 187.]
   [ 211.]
   [ 208.]
   [ 233.]]

  [[ 407.]
   [ 474.]
   [ 432.]
   [ 379.]]]]
```

```
Result SAME

[[[[  68.]
   [ 114.]
   [ 149.]
   [ 169.]
   [ 161.]
   [ 109.]]

  [[ 135.]
   [ 203.]
   [ 236.]
   [ 229.]
   [ 179.]
   [ 109.]]

  [[  98.]
   [ 139.]
   [ 153.]
   [ 148.]
   [ 135.]
   [ 107.]]
```

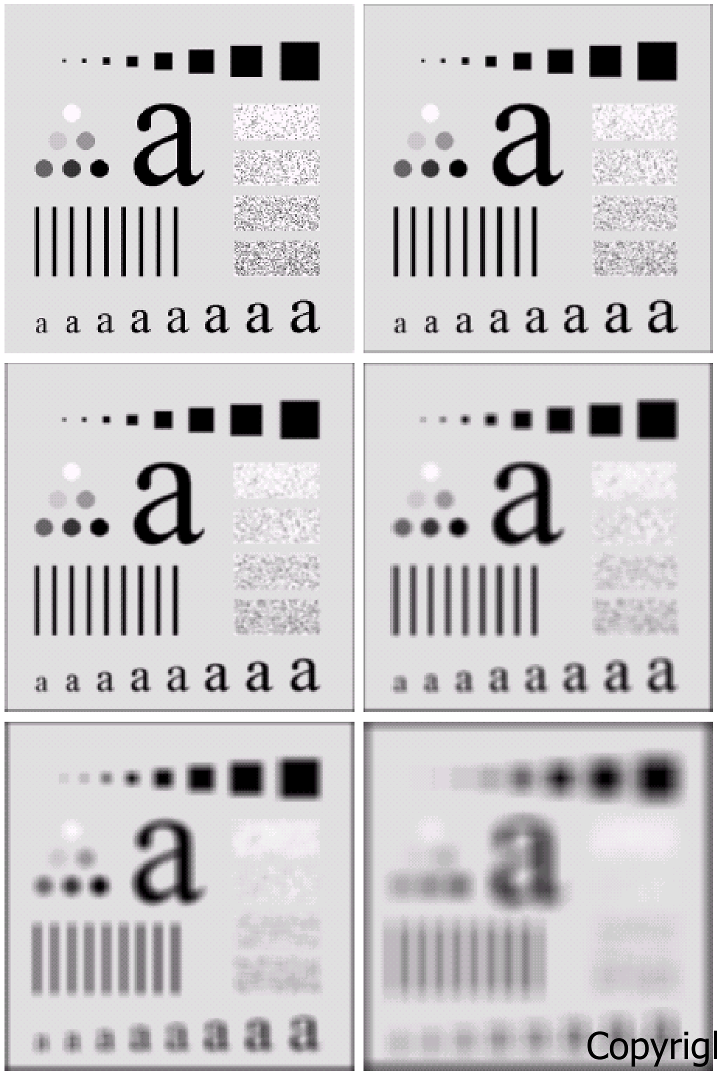
```
Result SAME

  [[ 134.]
   [ 187.]
   [ 211.]
   [ 208.]
   [ 233.]
   [ 219.]]

  [[ 254.]
   [ 407.]
   [ 474.]
   [ 432.]
   [ 379.]
   [ 287.]]

  [[ 189.]
   [ 373.]
   [ 459.]
   [ 421.]
   [ 320.]
   [ 188.]]]]
```

Types of Filters

Types of Filters

- Smoothing Filters
  - Blurring
  - Noise reduction
  - Linear Filters
    - Average
    - Gaussian
  - Non-linear Filter
    - Order-Statistics Filters
- Sharpening Filters
  - High light the fine details

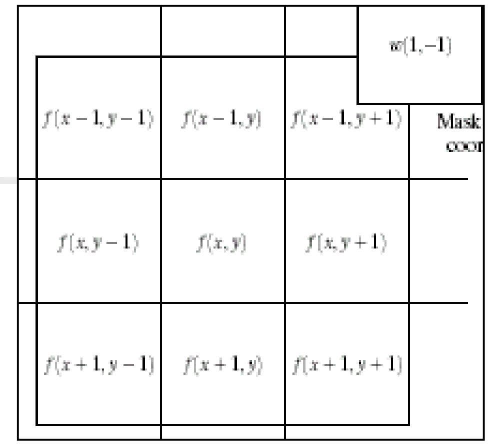
Effect of Smoothing/Linear Filter



a b
c d
e f

FIGURE 3.35 (a) Original image, of size 500 × 500 pixels. (b)–(f) Results of smoothing with square averaging filter masks of sizes $n = 3, 5, 9, 15$, and 35, respectively. The black squares at the top are of sizes 3, 5, 9, 15, 25, 35, 45, and 55 pixels, respectively; their borders are 25 pixels apart. The letters at the bottom range in size from 10 to 24 points, in increments of 2 points; the large letter at the top is 60 points. The vertical bars are 5 pixels wide and 100 pixels high; their separation is 20 pixels. The diameter of the circles is 25 pixels, and their borders are 15 pixels apart; their gray levels range from 0% to 100% black in increments of 20%. The background of the image is 10% black. The noisy rectangles are of size 50 × 120 pixels.

Laplacian Filter



$$\nabla^2 f(x,y) = \frac{\partial^2 f}{\partial x^2} + \frac{\partial^2 f}{\partial y^2}$$

$$\frac{\partial^2 f}{\partial x^2} = f(x+1,y) + f(x-1,y) - 2f(x,y)$$

$$\frac{\partial^2 f}{\partial y^2} = f(x,y+1) + f(x,y-1) - 2f(x,y)$$

$$\nabla^2 f(x,y) = f(x+1,y) + f(x-1,y) + f(x,y+1) + f(x,y-1) - 4f(x,y)$$

Convolution filter

0	1	0
1	-4	1
0	1	0

0	-1	0
-1	4	-1
0	-1	0

Laplacian Filter

Diagonal neighbors can also be included in the filter kernel

a b
c d

(a)

0	1	0
1	−4	1
0	1	0

(b)

1	1	1
1	−8	1
1	1	1

(c)

0	−1	0
−1	4	−1
0	−1	0

(d)

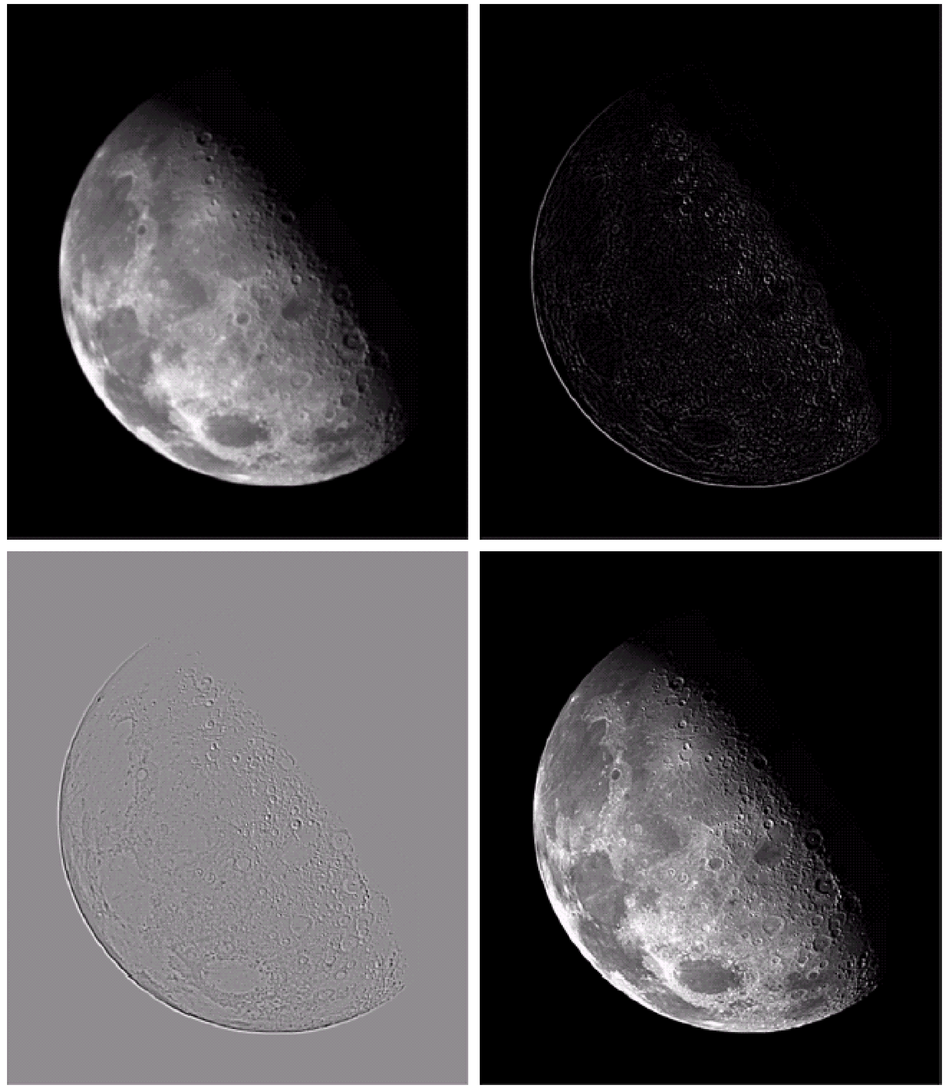
−1	−1	−1
−1	8	−1
−1	−1	−1

FIGURE 3.39 (a) Filter mask used to implement the digital Laplacian, as defined in Eq. (3.7-4). (b) Mask used to implement an extension of this equation that includes the diagonal neighbors. (c) and (d) Two other implementations of the Laplacian.

Laplacian Filter

a b
c d

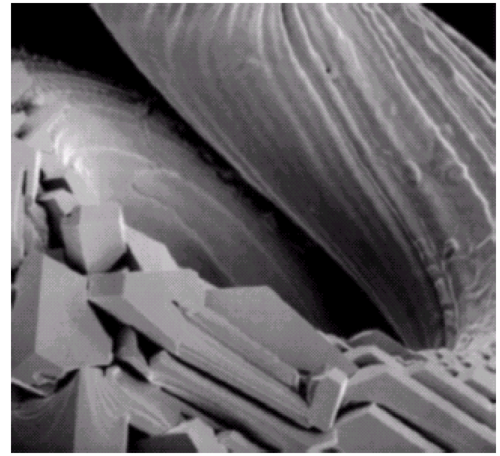
FIGURE 3.40
(a) Image of the North Pole of the moon.
(b) Laplacian-filtered image.
(c) Laplacian image scaled for display purposes.
(d) Image enhanced by using Eq. (3.7-5).
(Original image courtesy of NASA.)

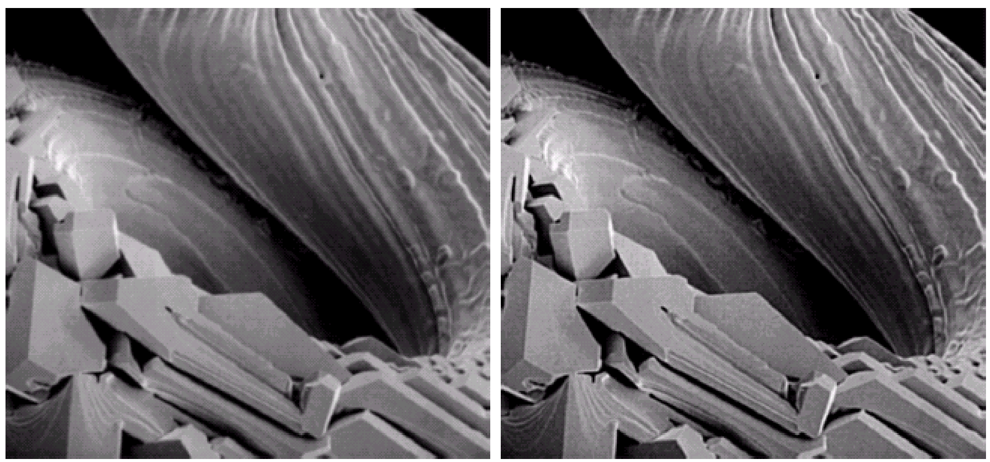


Laplacian Filter

0	−1	0
−1	5	−1
0	−1	0

−1	−1	−1
−1	9	−1
−1	−1	−1





a b c
d e

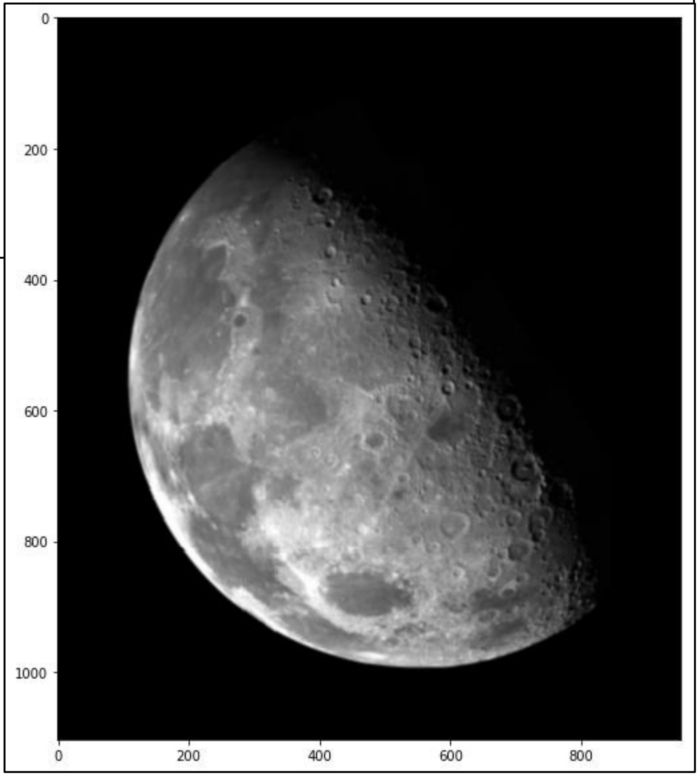
FIGURE 3.41 (a) Composite Laplacian mask. (b) A second composite mask. (c) Scanning electron microscope image. (d) and (e) Results of filtering with the masks in (a) and (b), respectively. Note how much sharper (e) is than (d). (Original image courtesy of Mr. Michael Shaffer, Department of Geological Sciences, University of Oregon, Eugene.)

Convolution of an Image in Python Step#1: Load Libraries

```
##################################################
# Read an image
# Apply Filter on that image
#################################################
#
import numpy as np
from scipy import signal
import matplotlib.pyplot as plt
from PIL import Image
```


Convolution of an Image in Python Step#2: Read an Image File

```
##############################################################
# Read an image file
#

im = Image.open('Fig0317(a).png')

fig, aux = plt.subplots(figsize=(10,10))
aux.imshow(im, cmap='gray')
```



Convolution of an Image in Python Step#3: Create Laplacian Filter

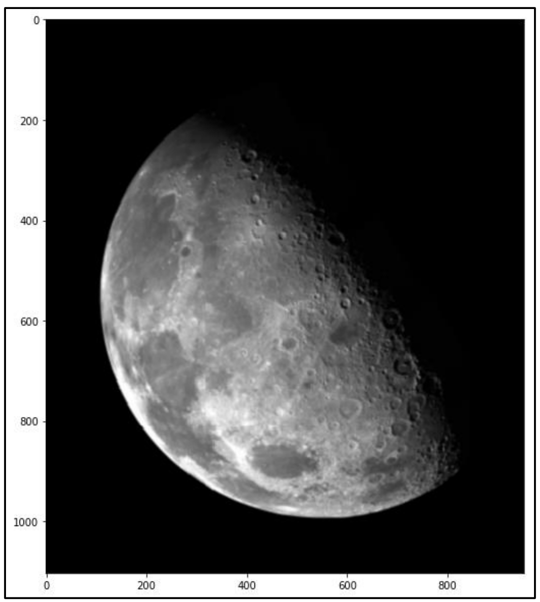
```
########################################################
image_gr = im.convert("L")
arr = np.asarray(image_gr)

#########################################################
# Create Laplacian Filter
#
kernel = np.array ([
        [0,  1,  0],
        [1, -4,  1],
        [0,  1,  0]
        ])
```

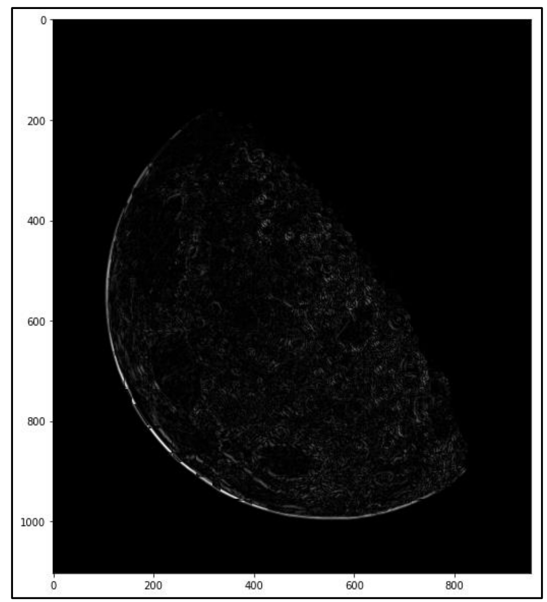
Convolution of an Image in Python

Step#4: Apply Filter on Image

Convolution Operator

```
#########################################################
# Apply the convolution filter
#

grad = signal.convolve2d(arr,kernel, mode='same', boundary='symm')

fig, aux = plt.subplots(figsize=(10,10))
aux.imshow(np.absolute(grad), cmap='gray')
```

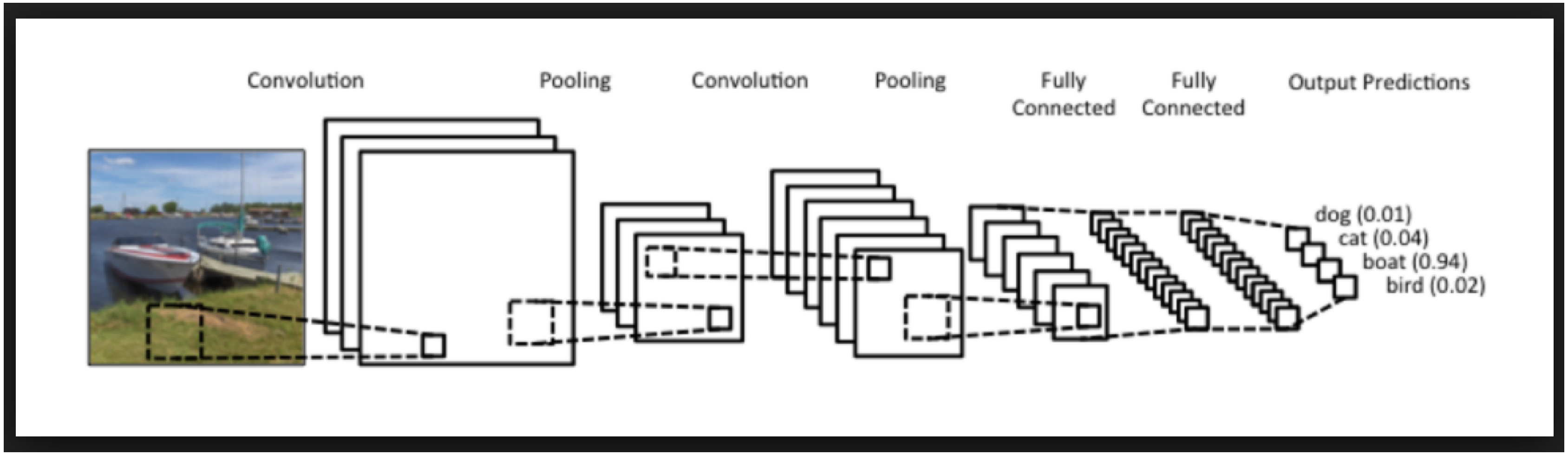




Pooling Layer

Convolution Neural Network

- Convolution Layer
- Pooling Layer



Why Pooling Layer

- Why Pooling Layer?
  - Reduce the size of the image representation to speed up computation
  - Feature detection becomes more robust

Pooling Layer: Max Pooling

1	3	2	1
2	9	1	1
1	3	2	3
5	6	1	2

Max Pooling

- Image Size = 4x4
- Filter size = 2x2
- Stride = 2

1	3	2	1
2	9	1	1
1	3	2	3
5	6	1	2

9	2
6	3

Max Pooling

- Image Size = 5x5
- Filter size = 3x3
- Stride = 1

1	3	2	1	3
2	9	1	1	5
1	3	2	3	2
8	3	5	1	0
5	6	1	2	9

9		

Max Pooling

- Image Size = 5x5
- Filter size = 3x3
- Stride = 1

1	3	2	1	3
2	9	1	1	5
1	3	2	3	2
8	3	5	1	0
5	6	1	2	9

9	9	

Max Pooling

- Image Size = 5x5
- Filter size = 3x3
- Stride = 1

1	3	2	1	3
2	9	1	1	5
1	3	2	3	2
8	3	5	1	0
5	6	1	2	9

9	9	5

Max Pooling

- Image Size = 5x5
- Filter size = 3x3
- Stride = 1

1	3	2	1	3
2	9	1	1	5
1	3	2	3	2
8	3	5	1	0
5	6	1	2	9

9	9	5
9		

Max Pooling

- Image Size = 5x5
- Filter size = 3x3
- Stride = 1

1	3	2	1	3
2	9	1	1	5
1	3	2	3	2
8	3	5	1	0
5	6	1	2	9

9	9	5
9	9	5
8	6	9

2-Channel Image

- Image Size = 5x5
- Filter size = 3x3
- Stride = 1

1	3	2	1	3
2	9	1	1	5
1	3	2	3	2
8	3	5	1	0
5	6	1	2	9

9	9	5
9	9	5
8	6	9

Average Pooling

- Image Size = 4x4
- Filter size = 2x2
- Stride = 2

1	3	2	1
2	9	1	1
1	4	2	3
5	6	1	2

3.75	1.25
4	2

Image Size After Pooling

- Image size = Height x Width x Channels = $n_H \times n_w \times n_c$
- Hyperparameters
- f: filter size = f
- s: stride = s
- Max or average pooling

- Image size after pooling =$\left\lfloor\frac{n_H - f}{s} + 1\right\rfloor \times \left\lfloor\frac{n_W - f}{s} + 1\right\rfloor \times n_C$

Summary

- Correlation Operator
- Convolution Operator
- Image Enhancement in Spatial Domain
- Types of Filters
  - Smoothing Filters: Gaussian Filter
  - Sharpening Filters: Laplacian Filter
- Pooling Layer