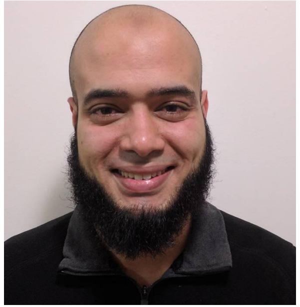
Machine Learning
Padding and Stride

Mostafa S. Ibrahim
Teaching, Training and Coaching for more than a decade!

Artificial Intelligence & Computer Vision Researcher
PhD from Simon Fraser University - Canada
Bachelor / MSc from Cairo University - Egypt
Ex-(Software Engineer / ICPC World Finalist)



Challenge at the Boundaries

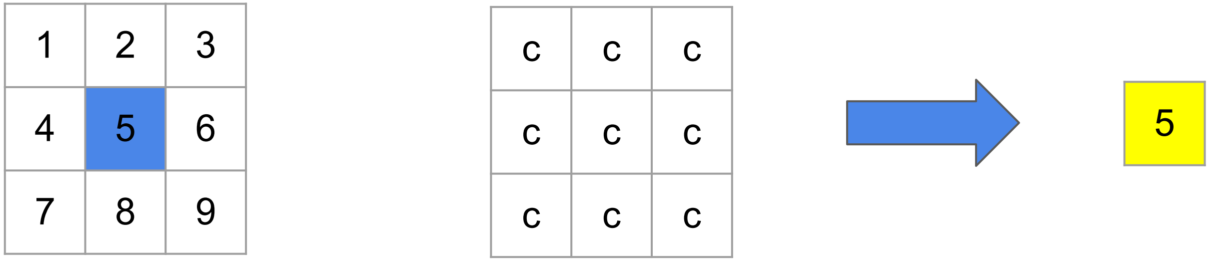
- Assume we have a 3x3 input matrix and a kernel 3x3 for average (c = 1/9)
- What is the output matrix? Its size?
- What is the output for location input[0][0]?

1	2	3
4	5	6
7	8	9

c	c	c
c	c	c
c	c	c

Challenge at the Boundaries

- One solution is to only compute output for **valid** input locations!
- A valid pixel is one where the kernel completely inside the matrix
- There is only one valid pixel here at (1, 1) with input value 5
  - (1+2+3+4+5+6+7+8+9) / 9 = 5
- Given input DxD and kernel KxK, what is the **reduced** output matrix size?!



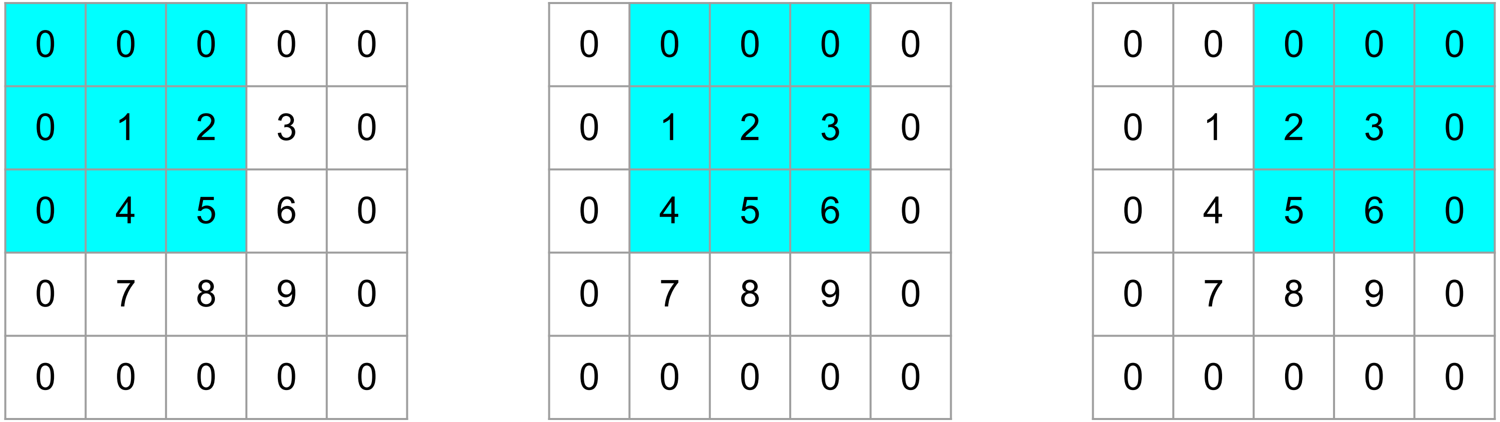
Challenge at the Boundaries

- Another approach is to pad the grid such that the output has the **same** dimensions as the input, then convolve on the valid pixels in padded grid

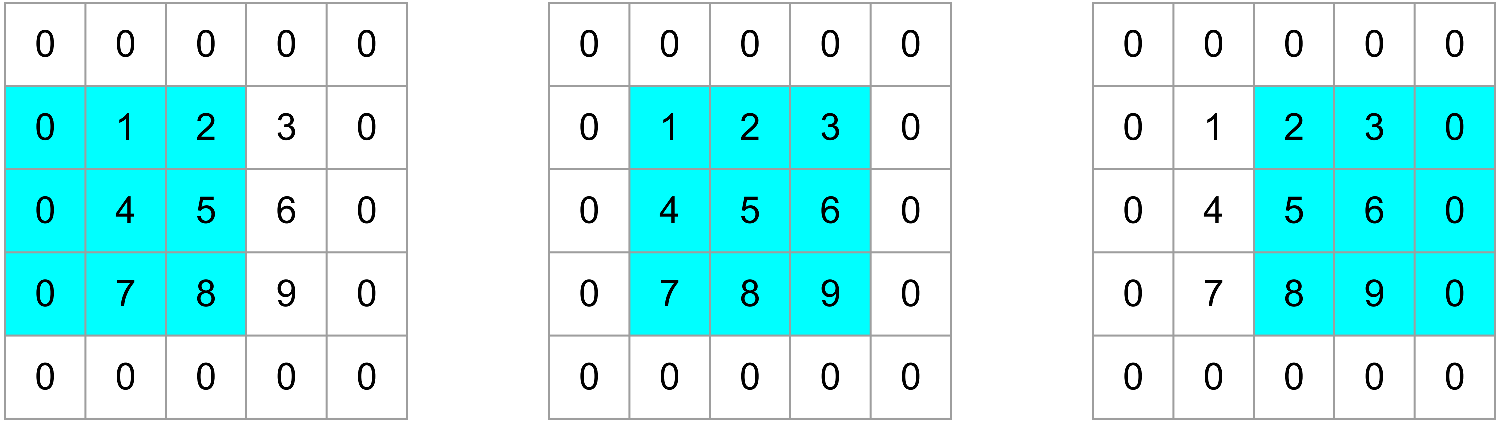
0	0	0	0	0
0	1	2	3	0
0	4	5	6	0
0	7	8	9	0
0	0	0	0	0

c	c	c
c	c	c
c	c	c

1.3	2.3	1.8
3	5	**3.7**
2.7	4.3	3.1



1st operation: (0 + 0 + 0 + 0 + 1 + 2 + 0 + 4 + 5) / 9 = 1.3



6th operation: (2+3+0+5+6+8+9+0) / 9 = 3.7

Pytorch

- Pytorch provides us the flexibility of the 3 approaches for padding
  - Padding controls the amount of padding applied to the input.
  - String: **valid** ⇒ only valid pixels ⇒ grid size is reduced
  - String: **same** ⇒ pad with zeros such that output has the same input's resolution
    - Most common in practice
    - Specifically, pad with (k-1)/2 from each side
  - An int / a tuple of ints giving the amount of implicit padding applied on both sides.

PyTorch Kernel

- We can create a custom kernel. Here is a simple one for averaging
- It creates KxK matrix for averaging

```python
class AverageConvolution(nn.Module):
    def __init__(self, k = 3, padding = 'same'):
        super(AverageConvolution, self).__init__()
        self.padding = padding
        self.filter = torch.ones(1, 1, k, k) / (k * k)

    def forward(self, x):
        return torch.conv2d(x, self.filter, padding=self.padding)
```

```python
inp_d = 10
input = torch.arange(1, inp_d*inp_d+1).view(1, 1, inp_d, inp_d).float()

kernel = AverageConvolution(k=3, padding='same')
print(kernel(input).shape)    # torch.Size([1, 1, 10, 10])

# With valid, output is (D - K + 1) x (D - K + 1)
kernel = AverageConvolution(k=3, padding='valid')
print(kernel(input).shape)    # torch.Size([1, 1, 8, 8])

kernel = AverageConvolution(k=5, padding='valid')
print(kernel(input).shape)    # torch.Size([1, 1, 6, 6])

kernel = AverageConvolution(k=7, padding='valid')
print(kernel(input).shape)    # torch.Size([1, 1, 4, 4])

kernel = AverageConvolution(k=9, padding='valid')
print(kernel(input).shape)    # torch.Size([1, 1, 2, 2])

kernel = AverageConvolution(k=10, padding='valid')
print(kernel(input).shape)    # torch.Size([1, 1, 1, 1])
```

Stride

- What if we don’t want a response for every pixel and would like to jump some in the middle?
  - For example, to reduce output size as in DNN
- The stride controls the **step size** at which the kernel is moved across the input data
  - A stride of 1 means the kernel moves on every pixel (as we did)
  - Most common stride

Stride

- Assume input is 6x6 and kernel = 3x3 with padding = valid (no padding)
- If the stride is S=3, this means jump 3 locations after each calculation
- Below 2 computations only per first 3 rows

1	2	3	4	5	6
7	8	9	10	11	12
13	14	15	16	17	18
19	20	21	22	23	24
25	26	27	28	29	30
31	32	33	34	35	36

(1+2+3+
7+8+9+
13+14+15)
/ 9 = 8

1	2	3	4	5	6
7	8	9	10	11	12
13	14	15	16	17	18
19	20	21	22	23	24
25	26	27	28	29	30
31	32	33	34	35	36

(4+5+6+
10+11+12+
16+17+18)
/ 9 = 11

Stride

- Similarly, we can evaluate the below 2 corners

1	2	3	4	5	6
7	8	9	10	11	12
13	14	15	16	17	18
19	20	21	22	23	24
25	26	27	28	29	30
31	32	33	34	35	36

(19+20+21+
25+26+27+
31+32+33)
/ 9 = 26

1	2	3	4	5	6
7	8	9	10	11	12
13	14	15	16	17	18
19	20	21	22	23	24
25	26	27	28	29	30
31	32	33	34	35	36

(22+23+24+
28+29+30+
34+35+36)
/ 9 = 29

Stride

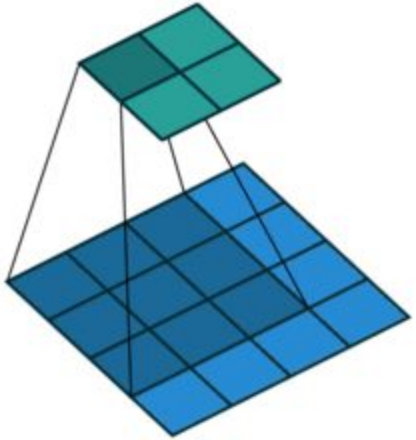
- So overall, we get 2x2 matrix from stride=3
- Given input DxD, kernel KxK, No padding and stride S, the reduced matrix has a dimension ((D - K) / S) + 1

1	2	3	4	5	6
7	8	9	10	11	12
13	14	15	16	17	18
19	20	21	22	23	24
25	26	27	28	29	30
31	32	33	34	35	36

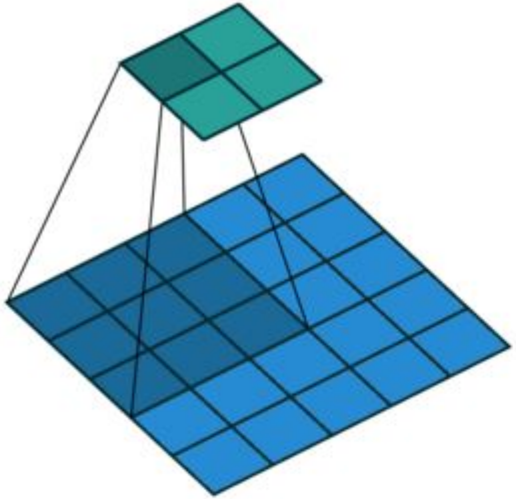
c	c	c
c	c	c
c	c	c

8	11
26	29

Stride + Padding



no_padding_no_strides

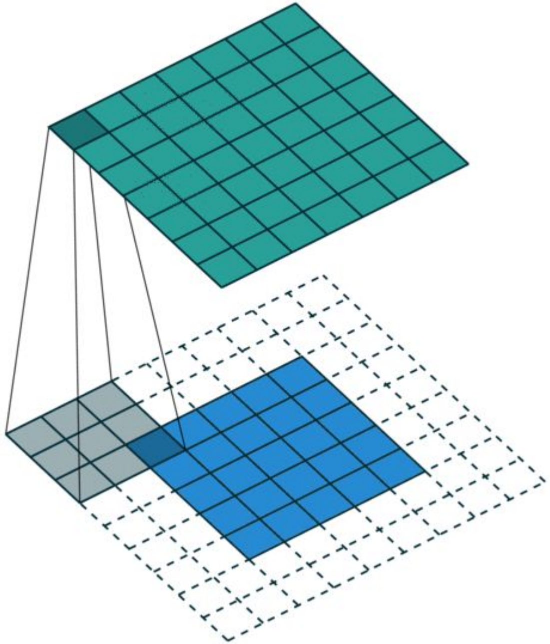


no_padding_strides

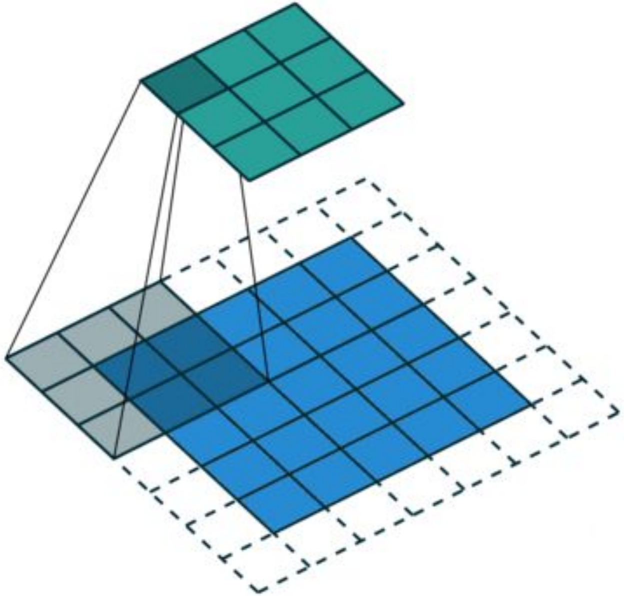
Img src

Stride + Padding

Given input DxD, kernel KxK, padding P and stride S, the output matrix has a dimension of: **[(D - K + 2P) / S] + 1**



padding_no_strides



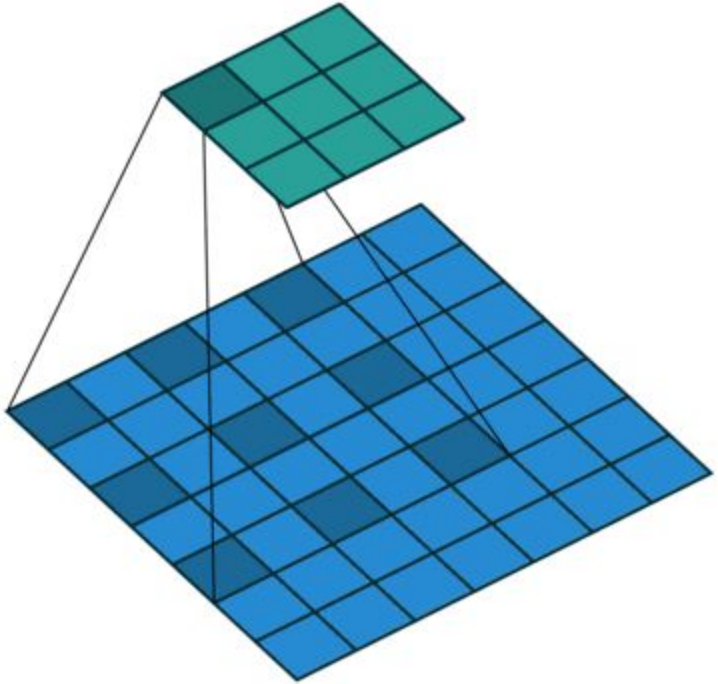
padding_strides

Dilated (Atrous) Convolution

- Each region where the kernel is extracting information is called a receptive field
- When we use a small 3x3 kernel, we see a very local region, so we lack more of the surrounding region
  - Later, we will have 3x3 weights for kernel. Bigger kernel are more computationally expensive.
- A meet in middle solution is to have a bigger view that works on a small subset of pixels by inserting gaps(dilations) between the kernel elements
- **Dilation Rate**: controls the spacing between the kernel points.
  - A dilation rate of 1 corresponds to the standard (non-dilated) convolution.
  - Larger dilation rates result in larger receptive fields.

Dilated (Atrous) Convolution

- This is a convolution operator with no stride or padding but dilation rate = 1
- It consists of 3x3 values computations, but extracting them from 5x5 region



torch.nn.Conv2d

- We now know many of pytorch conv2d parameters
- In future, explore **groups** parameter
  - controls the connections between inputs and outputs.
  - in_channels and out_channels must both be divisible by groups

CLASS torch.nn.Conv2d(*in_channels*, *out_channels*, *kernel_size*, *stride=1*, *padding=0*, *dilation=1*, *groups=1*, *bias=True*, *padding_mode='zeros'*, *device=None*, *dtype=None*) [SOURCE]

Applies a 2D convolution over an input signal composed of several input planes.

Relevant Materials

- A guide to convolution arithmetic for deep learning

"Acquire knowledge and impart it to the people."

"Seek knowledge from the Cradle to the Grave."