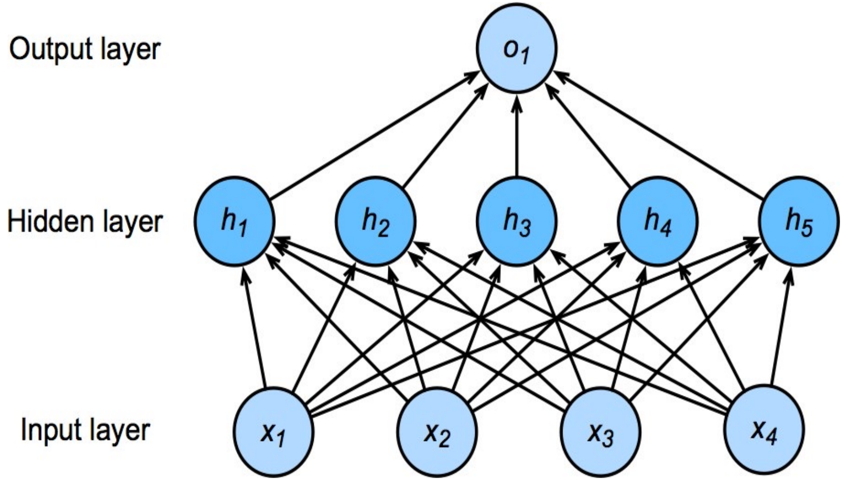
Convolution Neural Network

Flashback - Network with one hidden layer

64 x 64 x 3

1000 x 1000 x 3



If hidden layers has 1000 neurons. Parameters to estimate – 1000 *1000*3*1000=**3 Billion**

$$\mathbf{h} = \sigma(\mathbf{W}\mathbf{x} + \mathbf{b})$$

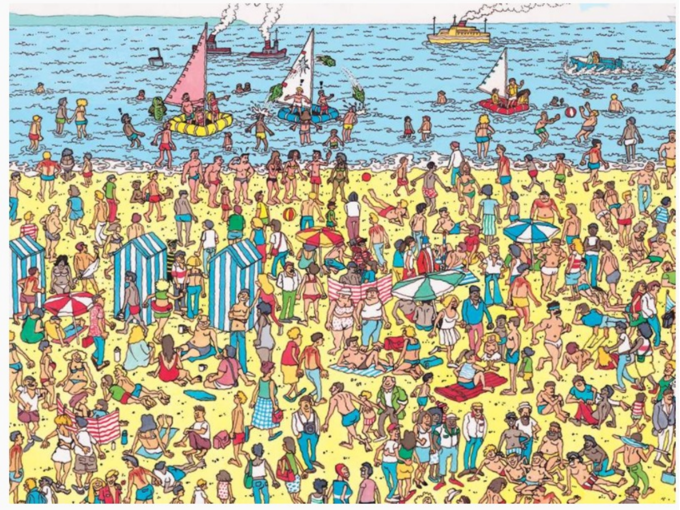
Solution - Convolutional Neural Network

- Images exhibit rich structure that can be exploited by humans and machine learning models alike.
- Convolutional neural networks (CNNs) are one creative way that machine learning has embraced for exploiting some of the known structure in natural images.

Two Principles

Translation Invariance

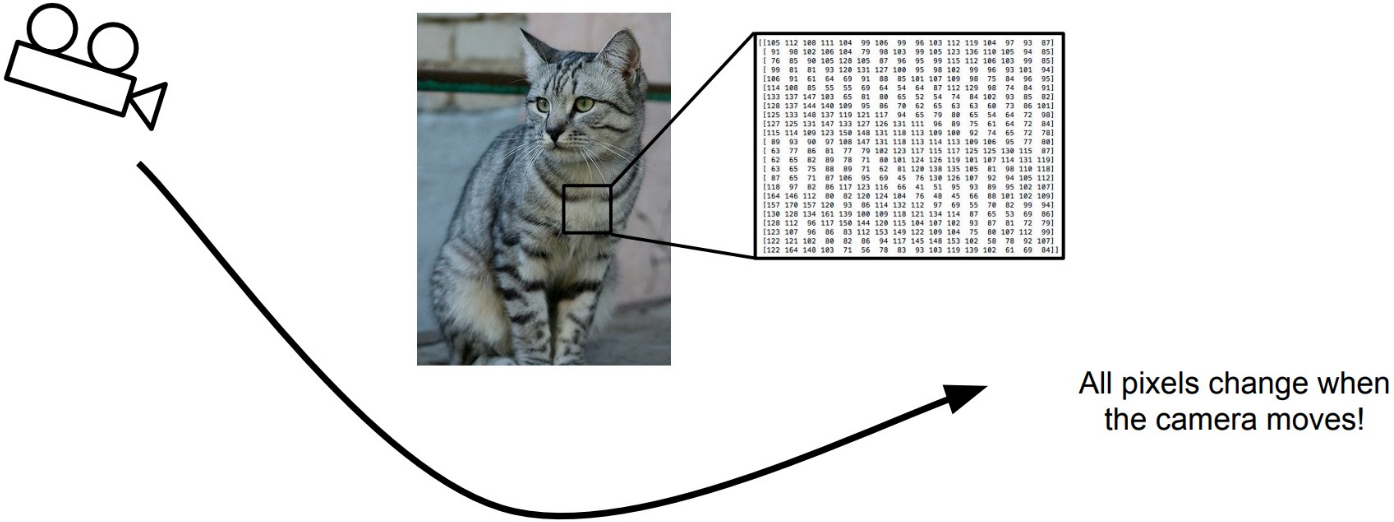
In the earliest layers, our network should respond similarly to the same patch, regardless of where it appears in the image.



Locality

- The earliest layers of the network should focus on local regions, without regard for the contents of the image in distant regions. This is the *locality* principle.
- Eventually, these local representations can be aggregated to make predictions at the whole image level

Translational Invariance - importance

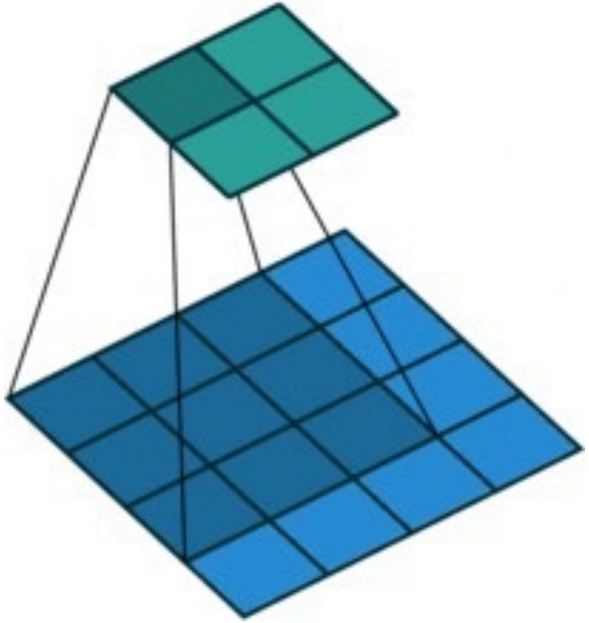


Source: http://cs231n.stanford.edu/slides/2018/cs231n_2018_lecture02.pdf

2-D Cross Correlation

$$\underset{\text{Input}}{\begin{bmatrix} 0 & 1 & 2 \\ 3 & 4 & 5 \\ 6 & 7 & 8 \end{bmatrix}} * \underset{\text{Kernel}}{\begin{bmatrix} 0 & 1 \\ 2 & 3 \end{bmatrix}} = \underset{\text{Output}}{\begin{bmatrix} 19 & 25 \\ 37 & 43 \end{bmatrix}}$$

$$\begin{aligned}
0\times 0+1\times 1+3\times 2+4\times 3 &= 19,\\
1\times 0+2\times 1+4\times 2+5\times 3 &= 25,\\
3\times 0+4\times 1+6\times 2+7\times 3 &= 37,\\
4\times 0+5\times 1+7\times 2+8\times 3 &= 43.
\end{aligned}$$



(vdumoulin@ Github)

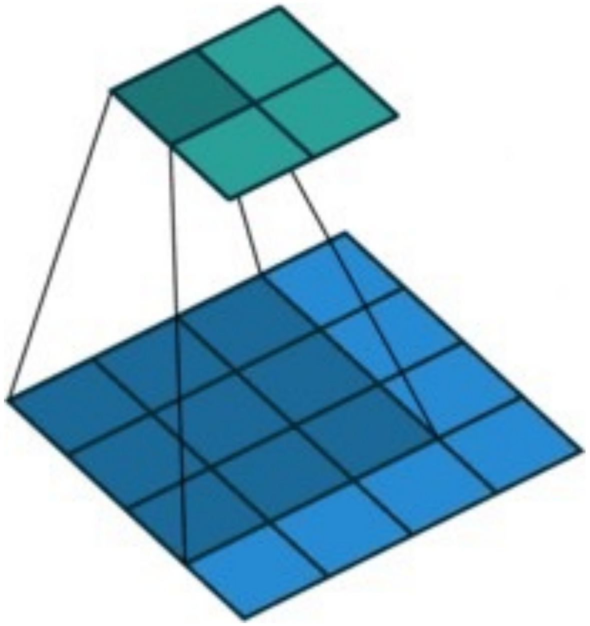
Slide credit: Alex Smola and Mu Li (Berkley)

2-D Cross Correlation

Input Kernel Output

$$\begin{bmatrix} 0 & 1 & 2 \\ 3 & 4 & 5 \\ 6 & 7 & 8 \end{bmatrix} * \begin{bmatrix} 0 & 1 \\ 2 & 3 \end{bmatrix} = \begin{bmatrix} 19 & 25 \\ 37 & 43 \end{bmatrix}$$

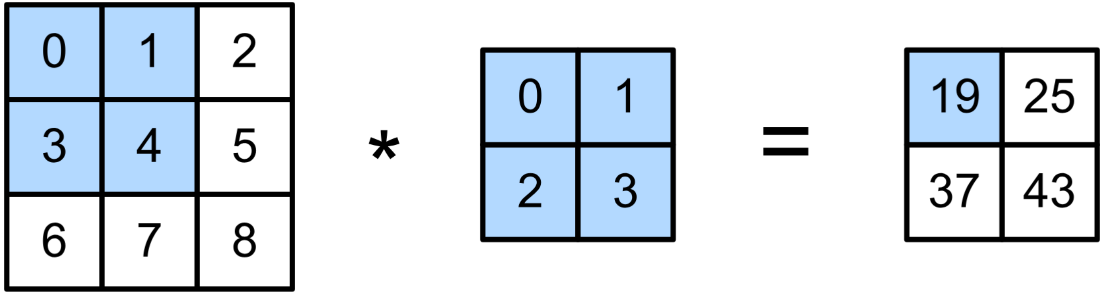
$$\begin{aligned} 0\times 0+1\times 1+3\times 2+4\times 3 &=19, \\ 1\times 0+2\times 1+4\times 2+5\times 3 &=25, \\ 3\times 0+4\times 1+6\times 2+7\times 3 &=37, \\ 4\times 0+5\times 1+7\times 2+8\times 3 &=43. \end{aligned}$$



(vdumoulin@ Github)

Slide credit: Alex Smola and Mu Li (Berkley)

2-D Convolution Layer



Dimensionality of input matrix $\boldsymbol{X}: n_h, n_w$
Dimensionality of kernel matrix $\boldsymbol{W}: k_h, k_w$
b: Scaler bias
Dimensionality of output matrix $\boldsymbol{Y}: n_h - k_h + 1, n_w - k_w + 1$

$$\boldsymbol{Y} = \boldsymbol{X} \star \boldsymbol{W} + b$$

Edge Detection

```
X = torch.ones((6, 8))
X[:, 2:6] = 0
```

```
K = torch.tensor([[1.0, -1.0]])
```

```
([[1., 1., 0., 0., 0., 0., 1., 1.],
  [1., 1., 0., 0., 0., 0., 1., 1.],
  [1., 1., 0., 0., 0., 0., 1., 1.],
  [1., 1., 0., 0., 0., 0., 1., 1.],
  [1., 1., 0., 0., 0., 0., 1., 1.],
  [1., 1., 0., 0., 0., 0., 1., 1.]])
```

```
Y = corr2d(X, K)
```

```
tensor([[ 0.,  1.,  0.,  0.,  0., -1.,  0.],
        [ 0.,  1.,  0.,  0.,  0., -1.,  0.],
        [ 0.,  1.,  0.,  0.,  0., -1.,  0.],
        [ 0.,  1.,  0.,  0.,  0., -1.,  0.],
        [ 0.,  1.,  0.,  0.,  0., -1.,  0.],
        [ 0.,  1.,  0.,  0.,  0., -1.,  0.]])
```

Learning a Kernel

```
conv2d = nn.Conv2d(1,1, kernel_size=(1, 2), bias=False)
```

Input channels, output channels, kernel size

```
X = X.reshape((1, 1, 6, 8))
```

(Batch_size, number of channels, height, width)

```
Y = Y.reshape((1, 1, 6, 7))
```

```
for i in range(10):
    Y_hat = conv2d(X)
    l = (Y_hat - Y) ** 2
    conv2d.zero_grad()
    l.sum().backward()
    # Update the kernel
    conv2d.weight.data[:] -= 3e-2 * conv2d.weight.grad
    if (i + 1) % 2 == 0:
        print(f'batch {i + 1}, loss {l.sum():.3f}')
```

More Edge Detection examples

Vertical edge detection

10	10	10	0	0	0
10	10	10	0	0	0
10	10	10	0	0	0
10	10	10	0	0	0
10	10	10	0	0	0
10	10	10	0	0	0

6 x 6

*

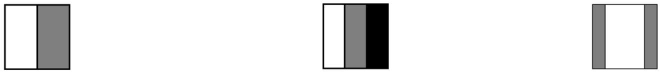
1	0	-1
1	0	-1
1	0	-1

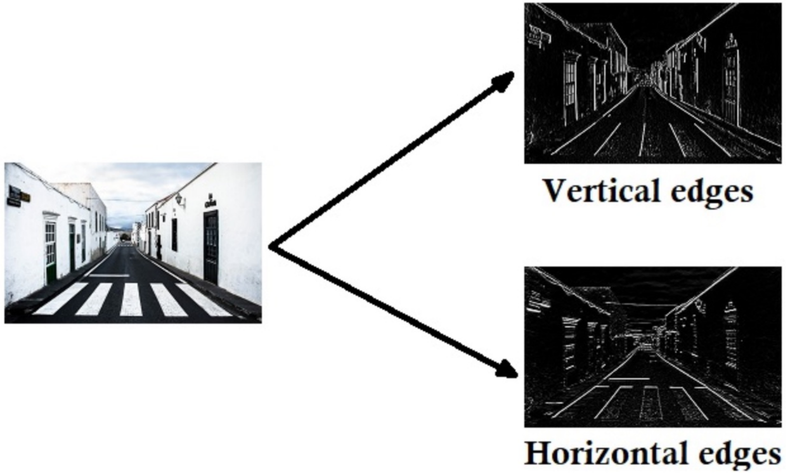
3 x 3

=

-0	30	30	0
0	30	30	0
0	30	30	0
0	30	30	0

4 x 4





Source Figures : http://datahacker.rs/edge-detection/

Examples

(wikipedia)

$$\begin{bmatrix} -1 & -1 & -1 \\ -1 & 8 & -1 \\ -1 & -1 & -1 \end{bmatrix}$$

Edge Detection

$$\begin{bmatrix} 0 & -1 & 0 \\ -1 & 5 & -1 \\ 0 & -1 & 0 \end{bmatrix}$$

Sharpen

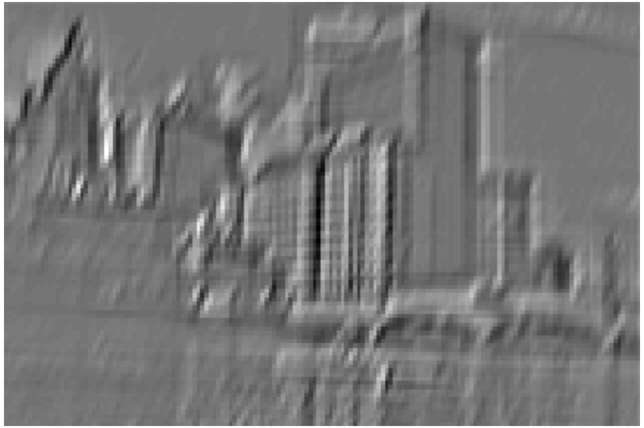
$$\frac{1}{16}\begin{bmatrix} 1 & 2 & 1 \\ 2 & 4 & 2 \\ 1 & 2 & 1 \end{bmatrix}$$

Gaussian Blur

Slide credit: Alex Smola and Mu Li (Berkley)

Examples

(Rob Fergus)



Slide credit: Alex Smola and Mu Li (Berkley)

Examples

Example of an edge (left), feature (center) and face detection (right)

Source Figures : http://datahacker.rs/edge-detection/

1-D and 3-D Cross Correlations

- 1-D

$$y_i = \sum_{a=1}^{h} w_a x_{i+a}$$

  - Text
  - Voice
  - Time series

- 3-D

$$y_{i,j,k} = \sum_{a=1}^{h} \sum_{b=1}^{w} \sum_{c=1}^{d} w_{a,b,c} x_{i+a,j+b,k+c}$$

  - Video
  - Medical images

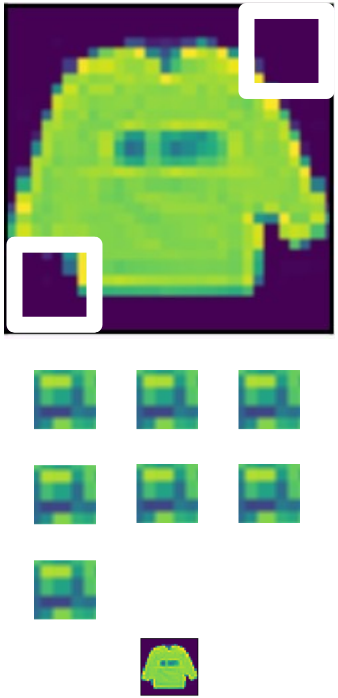
Padding and Stride

Slide credit: Alex Smola and Mu Li (Berkley)

Padding

- Given a 32 x 32 input image
- Apply convolutional layer with 5 x 5 kernel
  - 28 x 28 output with 1 layer
  - 4 x 4 output with 7 layers
- Shape decreases faster with larger kernels
  - Shape reduces from $n_h \times n_w$ to

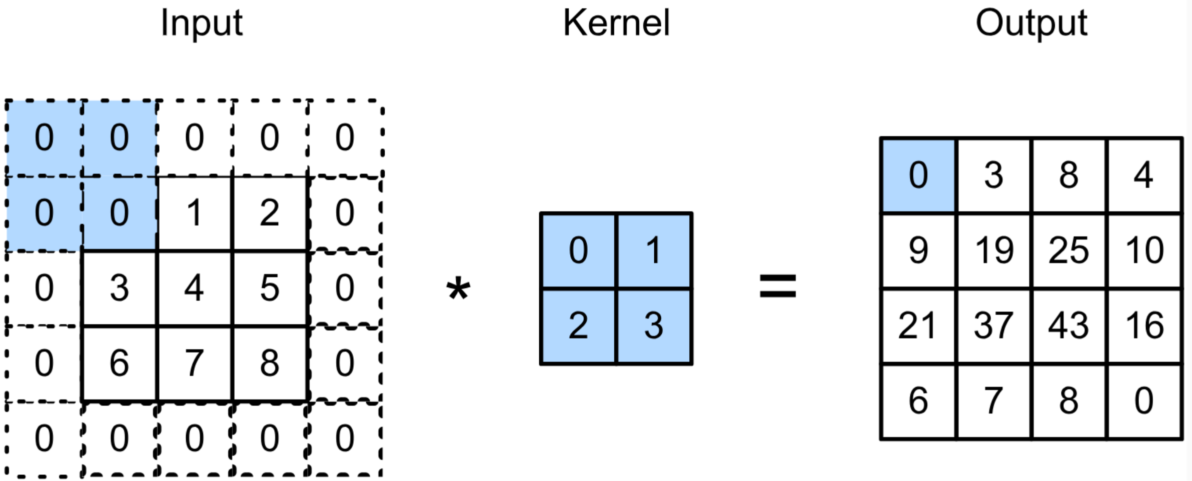
$$(n_h - k_h + 1) \times (n_w - k_w + 1)$$

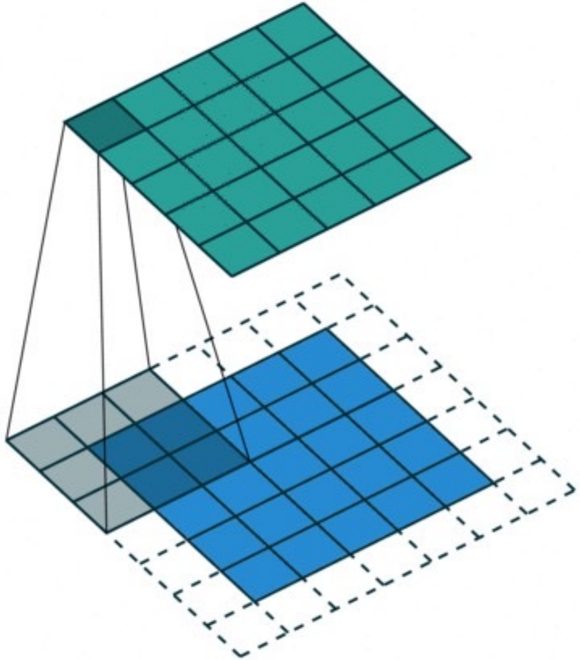


Slide credit: Alex Smola and Mu Li (Berkley)

Padding

Padding adds rows/columns around input





$$0\times0+0\times1+0\times2+0\times3=0$$

Slide credit: Alex Smola and Mu Li (Berkley)

Padding

- Padding p_h rows and p_w columns, output shape will be

$$(n_h - k_h + p_h + 1) \times (n_w - k_w + p_w + 1)$$

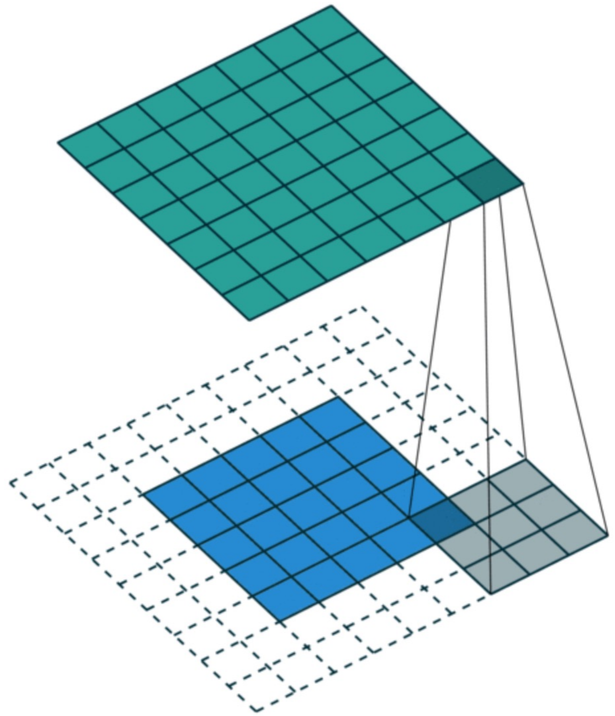
- A common choice is $p_h = k_h - 1$ and $p_w = k_w - 1$

- Assuming that k_h is odd here, we will pad $\frac{p_h}{2}$ rows on both sides of the height .
- Here p_w, p_h are total rows and columsn added (combined – considering both sides)

- If size of output is same as input – "same" Convolution
- If size of output is smaller than output – "valid" Convolution

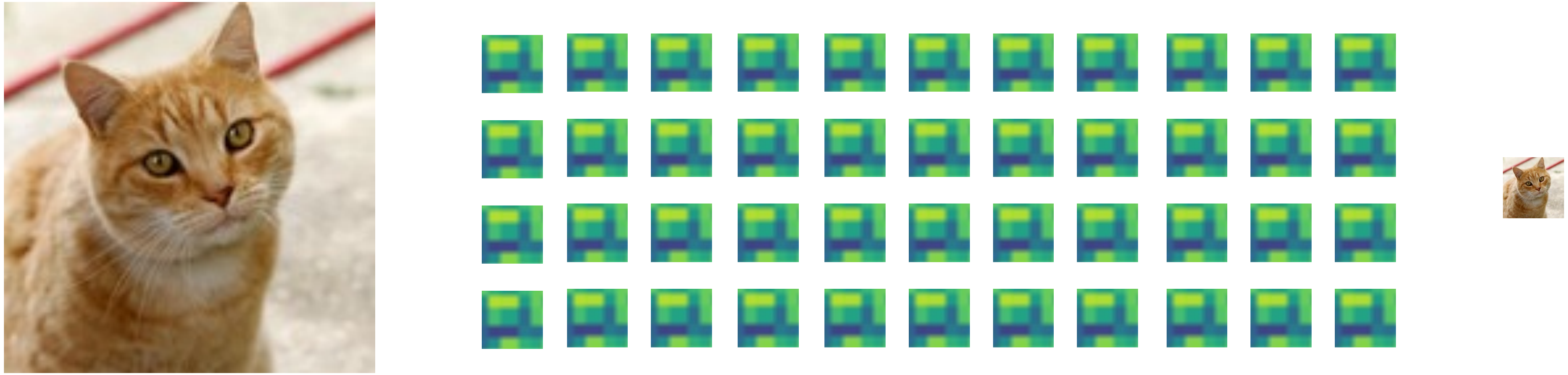
Full Padding

- In this setting every possible partial or complete superimposition of the kernel on the input feature map is taken into account.
- Size of output will be greater than input.
- Rarely used in practice.



Stride

- Padding reduces shape linearly with #layers
  - Given a 224 x 224 input with a 5 x 5 kernel, needs 44 layers to reduce the shape to 4 x 4
  - Requires a large amount of computation

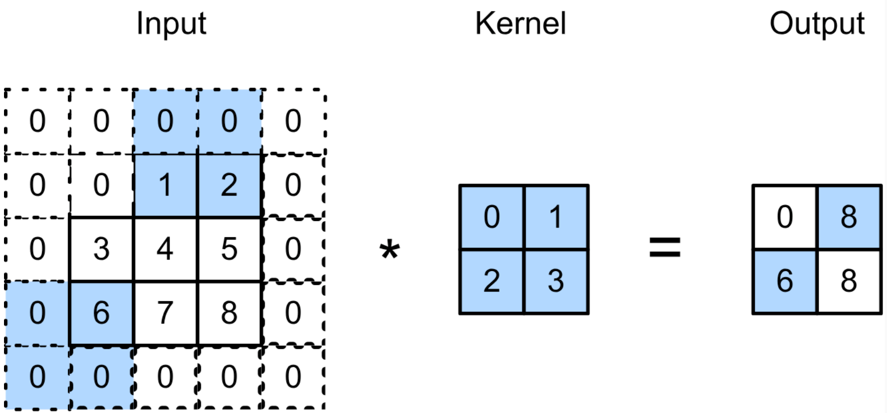


Slide credit: Alex Smola and Mu Li (Berkley)

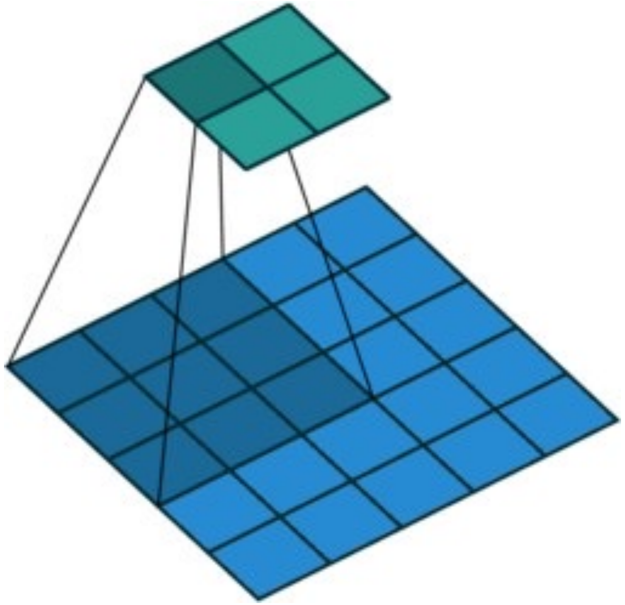
Stride

- Stride is the #rows/#columns per slide

Strides of 3 and 2 for height and width



$$0 \times 0 + 0 \times 1 + 1 \times 2 + 2 \times 3 = 8$$
$$0 \times 0 + 6 \times 1 + 0 \times 2 + 0 \times 3 = 6$$



Slide credit: Alex Smola and Mu Li (Berkley)

Stride

- Given stride s_h for the height and stride s_w for the width, the output shape is

$$\lfloor (n_h - k_h + p_h + s_h)/s_h \rfloor \times \lfloor (n_w - k_w + p_w + s_w)/s_w \rfloor$$

- With $p_h = k_h - 1$ and $p_w = k_w - 1$

$$\lfloor (n_h + s_h - 1)/s_h \rfloor \times \lfloor (n_w + s_w - 1)/s_w \rfloor$$

- If input height/width are divisible by strides

$$(n_h/s_h) \times (n_w/s_w)$$

Slide credit: Alex Smola and Mu Li (Berkley)

Padding Code PyTorch

```
conv2d = nn.Conv2d(1, 1, kernel_size=3, padding=1)
```

Here $k_h = k_w = 3, p_h = p_w = 2$

Note that here 1 row and column is padded on either side, so a total of 2

```
conv2d = nn.Conv2d(1, 1, kernel_size=(5, 3), padding=(2, 1))
```

Here $k_h = 5, k_w = 3, p_h = 4, \ p_w = 2$

```
conv2d = nn.Conv2d(1, 1, kernel_size=3, padding=1, stride=2)
```

```
conv2d = nn.Conv2d(1, 1, kernel_size=(3, 5), padding=(0, 1), stride=(3, 4))
```