# Dive into DL

6.1 ~ 6.3

# Index

- 6. Convolutional Neural Networks
  - 6.1. From Fully-Connected Layers to Convolutions
  - 6.2. Convolutions for Images
  - 6.3. Padding and Stride

# 6.1. From Fully-Connected Layers to Convolutions

- fully-connected layer characterized: pixels x features
- ***translation invariance***:
  network should respond similarly to the same patch
- ***locality* principle**:
  focus on local regions, without regard for the contents of the image in distant regions.

# 6.1. From Fully-Connected Layers to Convolutions

## 6.1.2. Constraining the MLP

$$\begin{aligned}[\mathbf{H}]_{i,j} &= [\mathbf{U}]_{i,j} + \sum_k \sum_l [\mathsf{W}]_{i,j,k,l} [\mathbf{X}]_{k,l}\\ &= [\mathbf{U}]_{i,j} + \sum_a \sum_b [\mathsf{V}]_{i,j,a,b} [\mathbf{X}]_{i+a,j+b}.\end{aligned},$$

X : $Input$
H : hidden representations
W : fourth-order weight tensors
U : biases
re-index : k = i + a, l = j + b $[\mathsf{V}]_{i,j,a,b} = [\mathsf{W}]_{i,j,i+a,j+b}.$

### 6.1.2.1. Translation Invariance

only possible if $\mathsf{V}$ and $\mathbf{U}$ do not actually depend on $(i,j)$, i.e., we have $[\mathsf{V}]_{i,j,a,b} = [\mathbf{V}]_{a,b}$

$$[\mathbf{H}]_{i,j} = u + \sum_a \sum_b [\mathbf{V}]_{a,b} [\mathbf{X}]_{i+a,j+b}.$$

### 6.1.2.2. Locality

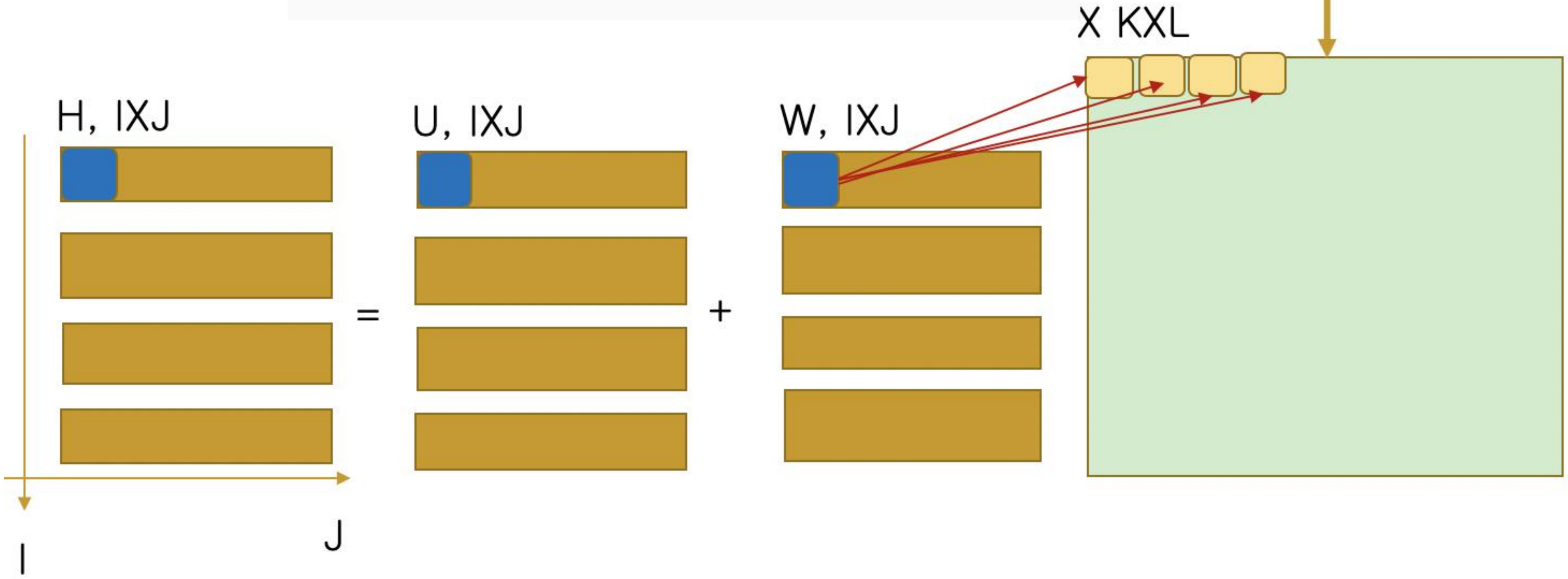
$|a| > \Delta$ or $|b| > \Delta$, set $[\mathbf{V}]_{a,b} = 0$.

$$[\mathbf{H}]_{i,j} = u + \sum_{a=-\Delta}^{\Delta} \sum_{b=-\Delta}^{\Delta} [\mathbf{V}]_{a,b} [\mathbf{X}]_{i+a,j+b}.$$

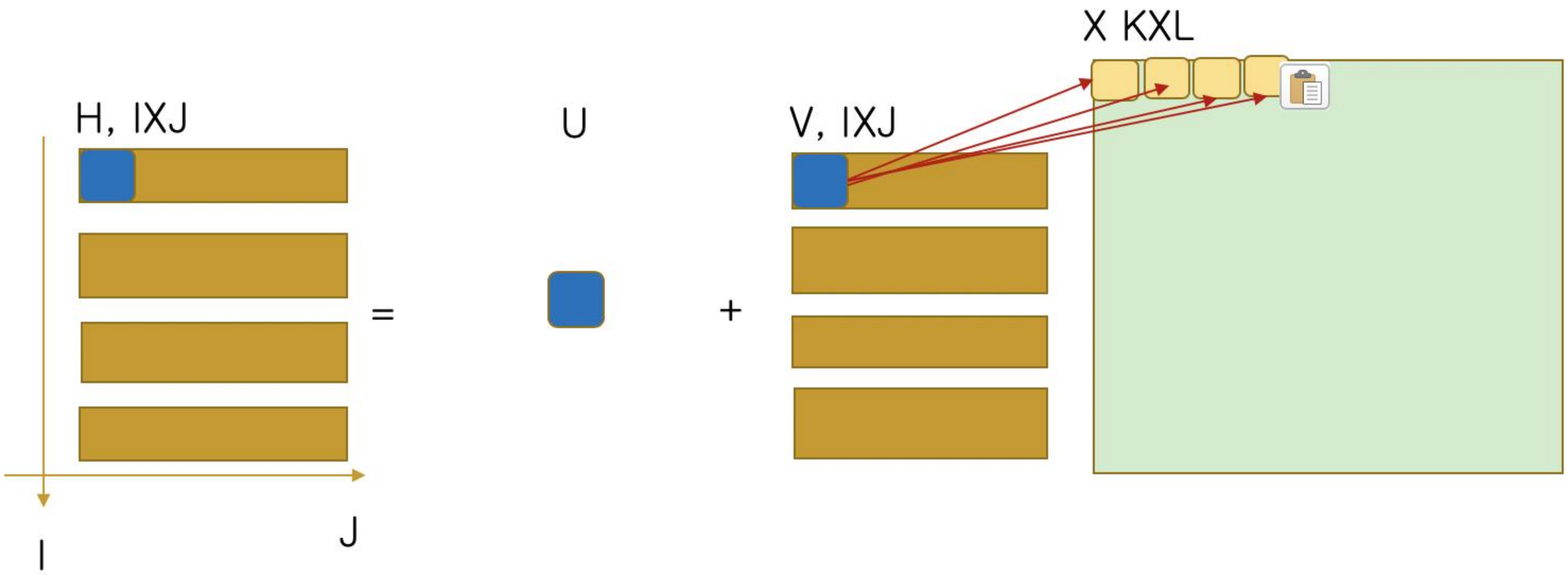
V is referred to as a *convolution kernel*, a *filter*, or simply the layer's *weights* that are often learnable parameters.

## Constraining the MLP

$$\begin{aligned}[\mathbf{H}]_{i,j} &= [\mathbf{U}]_{i,j} + \boxed{\sum_k \sum_l [\mathsf{W}]_{i,j,k,l} [\mathbf{X}]_{k,l}} \\ &= [\mathbf{U}]_{i,j} + \sum_a \sum_b [\mathsf{V}]_{i,j,a,b} [\mathbf{X}]_{i+a,j+b}.\end{aligned}$$



## Translation Invariance



# 6.1. From Fully-Connected Layers to Convolutions

## 6.1.2. Constraining the MLP

$$[\mathbf{H}]_{i,j} = [\mathbf{U}]_{i,j} + \sum_k \sum_l [\mathsf{W}]_{i,j,k,l} [\mathbf{X}]_{k,l}$$
$$= [\mathbf{U}]_{i,j} + \sum_a \sum_b [\mathsf{V}]_{i,j,a,b} [\mathbf{X}]_{i+a,j+b}.$$

X : $Input$
H : hidden representations
W : fourth-order weight tensors
U : biases
re-index : k = i + a, l = j + b $[\mathsf{V}]_{i,j,a,b} = [\mathsf{W}]_{i,j,i+a,j+b}.$

### 6.1.2.1. Translation Invariance

only possible if $\mathsf{V}$ and $\mathbf{U}$ do not actually depend on $(i,j)$, i.e., we have $[\mathsf{V}]_{i,j,a,b} = [\mathbf{V}]_{a,b}$

$$[\mathbf{H}]_{i,j} = u + \sum_a \sum_b [\mathbf{V}]_{a,b} [\mathbf{X}]_{i+a,j+b}.$$

### 6.1.2.2. Locality

$|a| > \Delta$ or $|b| > \Delta$, set $[\mathbf{V}]_{a,b} = 0$.

$$[\mathbf{H}]_{i,j} = u + \sum_{a=-\Delta}^{\Delta} \sum_{b=-\Delta}^{\Delta} [\mathbf{V}]_{a,b} [\mathbf{X}]_{i+a,j+b}.$$

V is referred to as a *convolution kernel*, a *filter*, or simply the layer's *weights* that are often learnable parameters.

# 6.1. From Fully-Connected Layers to Convolutions

## 6.1.3. Convolutions

$f, g : \mathbb{R}^d \to \mathbb{R}$

$$(f * g)(\mathbf{x}) = \int f(\mathbf{z}) g(\mathbf{x} - \mathbf{z}) d\mathbf{z}.$$

running over Z

$$(f * g)(i) = \sum_a f(a) g(i - a).$$

sum with indices $(a, b)$ for $f$ and $(i - a, j - b)$ for $g$,

$$(f * g)(i, j) = \sum_a \sum_b f(a, b) g(i - a, j - b).$$

# 6.1. From Fully-Connected Layers to Convolutions

- 6.1.4. "Where's Waldo" Revisited

  The convolutional layer picks windows of a given size and weighs intensities according to the filter $V$

# 6.1. From Fully-Connected Layers to Convolutions

## 6.1.5. Channel

In reality, third-order tensors, characterized by a height, width, and channel
3 channels: red, green, and blue

support multiple channels in both inputs X and hidden representations H

$$[\mathbf{H}]_{i,j} = u + \sum_{a=-\Delta}^{\Delta} \sum_{b=-\Delta}^{\Delta} [\mathbf{V}]_{a,b} [\mathbf{X}]_{i+a,j+b}.$$

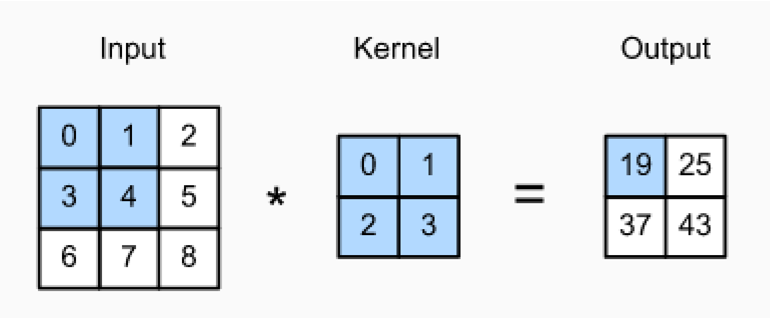
$$[\mathsf{H}]_{i,j,d} = \sum_{a=-\Delta}^{\Delta} \sum_{b=-\Delta}^{\Delta} \sum_{c} [\mathsf{V}]_{a,b,c,d} [\mathsf{X}]_{i+a,j+b,c},$$

d : indexes the output channels in the hidden representations

# 6.2. Convolutions for Images

## 6.2.1. The Cross-Correlation Operation

an input tensor and a kernel tensor are combined to produce an output tensor.




$$
\begin{aligned}
0\times0+1\times1+3\times2+4\times3&=19,\\
1\times0+2\times1+4\times2+5\times3&=25,\\
3\times0+4\times1+6\times2+7\times3&=37,\\
4\times0+5\times1+7\times2+8\times3&=43.
\end{aligned}
$$

```python
from d2l import tensorflow as d2l
import tensorflow as tf

def corr2d(X, K):  #@save
    """Compute 2D cross-correlation."""
    h, w = K.shape
    Y = tf.Variable(tf.zeros((X.shape[0] - h + 1, X.shape[1] - w + 1)))
    for i in range(Y.shape[0]):
        for j in range(Y.shape[1]):
            Y[i, j].assign(tf.reduce_sum(
                X[i: i + h, j: j + w] * K))
    return Y
```

```python
X = tf.constant([[0.0, 1.0, 2.0], [3.0, 4.0, 5.0], [6.0, 7.0, 8.0]])
K = tf.constant([[0.0, 1.0], [2.0, 3.0]])
corr2d(X, K)
```

```
<tf.Variable 'Variable:0' shape=(2, 2) dtype=float32, numpy=
array([[19., 25.],
       [37., 43.]], dtype=float32)>
```

## 6.2.2. Convolutional Layers

A convolutional layer cross-correlates the input and kernel and adds a scalar bias to produce an output

# 6.2. Convolutions for Images

## 6.2.3. Object Edge Detection in Images

construct an “image” of 6×8 pixels

```
X = tf.Variable(tf.ones((6, 8)))
X[:, 2:6].assign(tf.zeros(X[:, 2:6].shape))
X
```

```
<tf.Variable 'Variable:0' shape=(6, 8) dtype=float32, numpy=
array([[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.]], dtype=float32)>
```

construct a kernel K with a height of 1 and a width of 2.

```
K = tf.constant([[1.0, -1.0]])
```

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

```
<tf.Variable 'Variable:0' shape=(6, 7) dtype=float32, numpy=
array([[ 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.]], dtype=float32)>
```

The kernel K only detects vertical edges.

```
corr2d(tf.transpose(X), K)
```

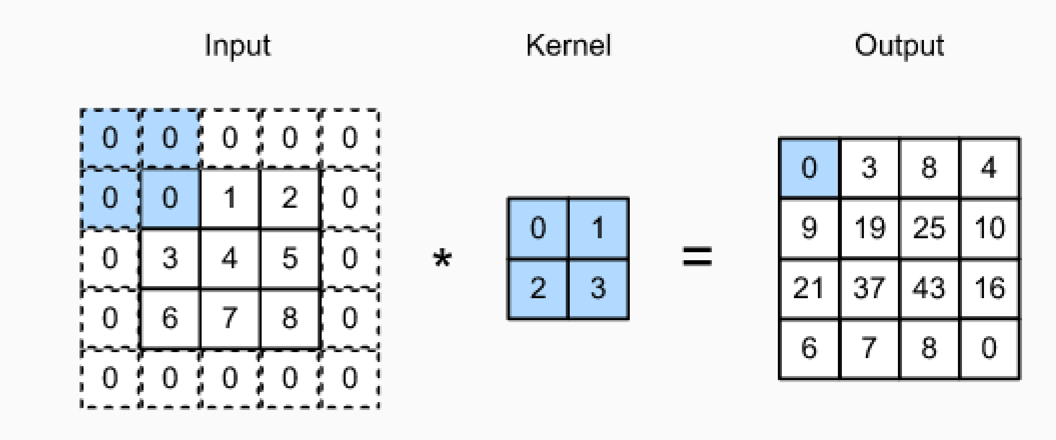
```
<tf.Variable 'Variable:0' shape=(8, 5) dtype=float32, numpy=
array([[0., 0., 0., 0., 0.],
       [0., 0., 0., 0., 0.],
       [0., 0., 0., 0., 0.],
       [0., 0., 0., 0., 0.],
       [0., 0., 0., 0., 0.],
       [0., 0., 0., 0., 0.],
       [0., 0., 0., 0., 0.],
       [0., 0., 0., 0., 0.]], dtype=float32)>
```

# 6.3. Padding and Stride

## 6.3.1. Padding

when applying convolutional layers is that we tend to lose pixels on the perimeter of our image.

set the values of the extra pixels to zero.




$$0 \times 0 + 0 \times 1 + 0 \times 2 + 0 \times 3 = 0.$$

```
import tensorflow as tf

# We define a convenience function to calculate the convolutional layer. This
# function initializes the convolutional layer weights and performs
# corresponding dimensionality elevations and reductions on the input and
# output
def comp_conv2d(conv2d, X):
    # Here (1, 1) indicates that the batch size and the number of channels
    # are both 1
    X = tf.reshape(X, (1, ) + X.shape + (1, ))
    Y = conv2d(X)
    # Exclude the first two dimensions that do not interest us: examples and
    # channels
    return tf.reshape(Y, Y.shape[1:3])
# Note that here 1 row or column is padded on either side, so a total of 2
# rows or columns are added
conv2d = tf.keras.layers.Conv2D(1, kernel_size=3, padding='same')
X = tf.random.uniform(shape=(8, 8))
comp_conv2d(conv2d, X).shape
```

```
TensorShape([8, 8])
```

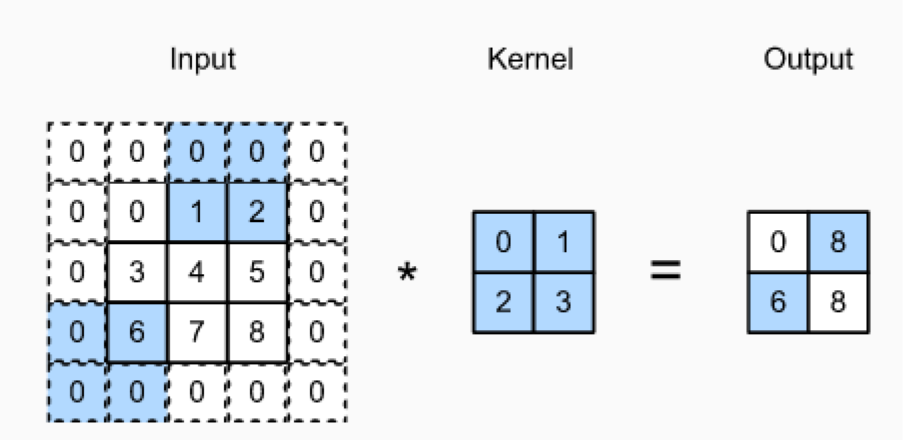
```
# Here, we use a convolution kernel with a height of 5 and a width of 3. The
# padding numbers on either side of the height and width are 2 and 1,
# respectively
conv2d = tf.keras.layers.Conv2D(1, kernel_size=(5, 3), padding='valid')
comp_conv2d(conv2d, X).shape
```

```
TensorShape([4, 6])
```

# 6.3. Padding and Stride

## 6.3.2. Stride

controls how depth columns around the spatial dimensions (width and height) are allocated




stride of 3 vertically and 2 horizontally

$$0\times0+0\times1+1\times2+2\times3=8, 0\times0+6\times1+0\times2+0\times3=6.$$

```
conv2d = tf.keras.layers.Conv2D(1, kernel_size=3, padding='same', strides=2)
comp_conv2d(conv2d, X).shape
```

```
TensorShape([4, 4])
```

```
conv2d = tf.keras.layers.Conv2D(1, kernel_size=3, padding='same', strides=2)
comp_conv2d(conv2d, X).shape
```

```
TensorShape([4, 4])
```