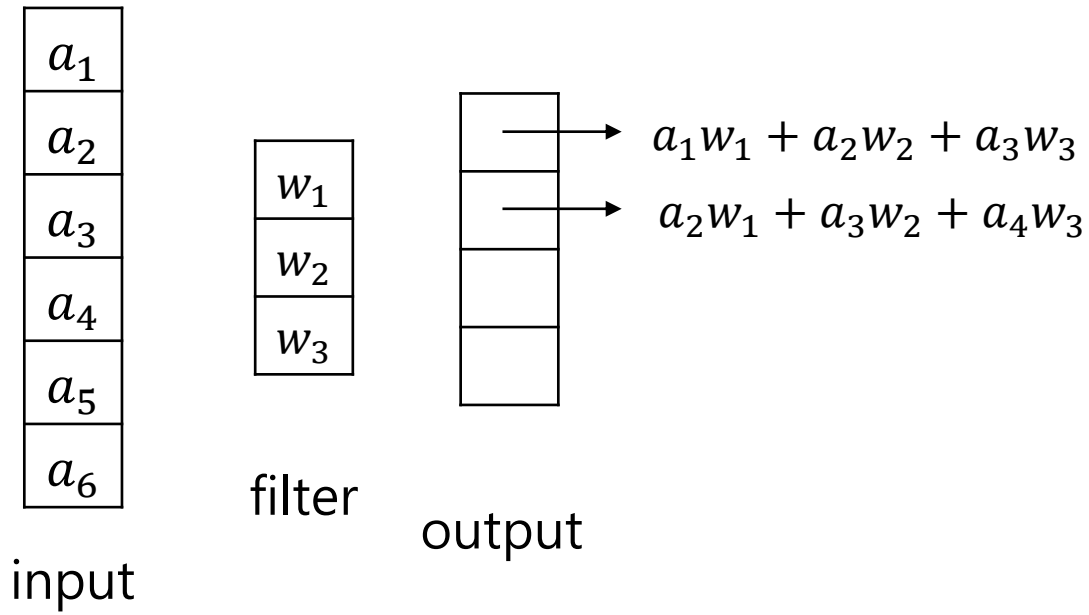
Chapter 2
Convolutional Neural Networks

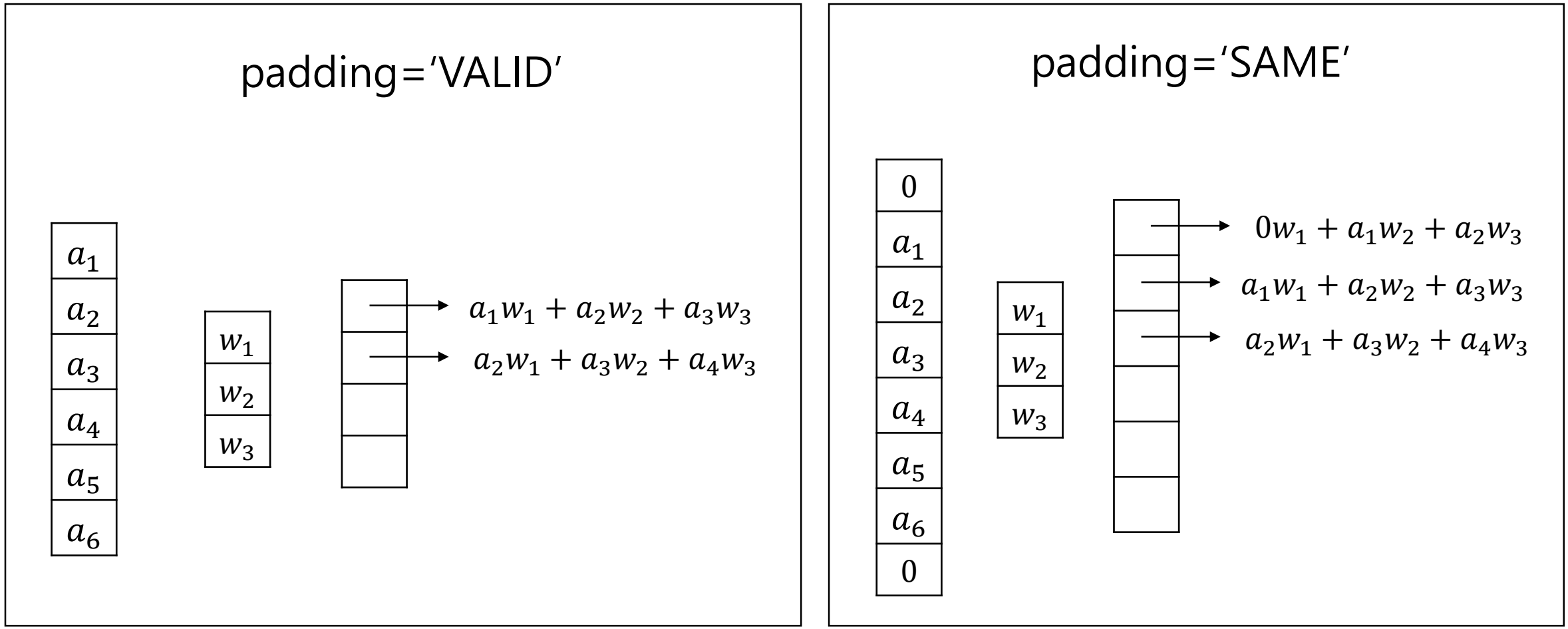
3.1 Filters, Strides, and Padding

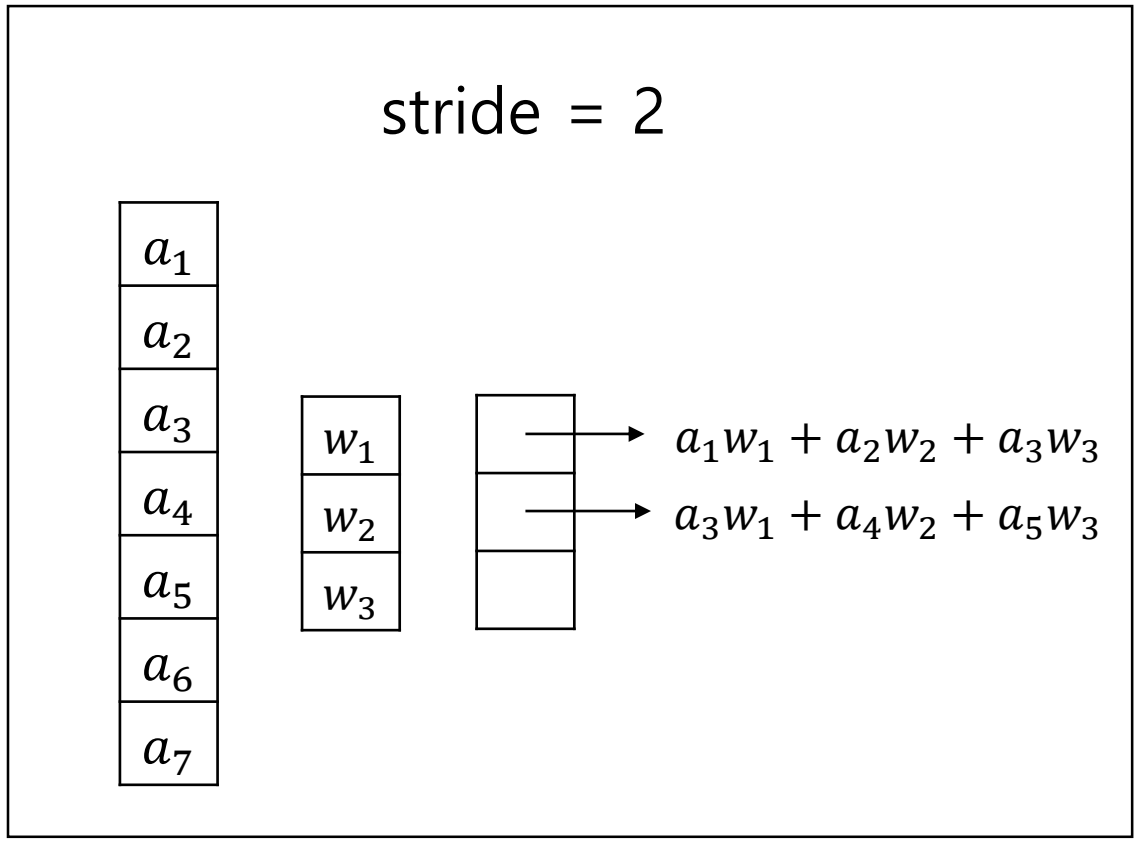
- Input I, kernel K

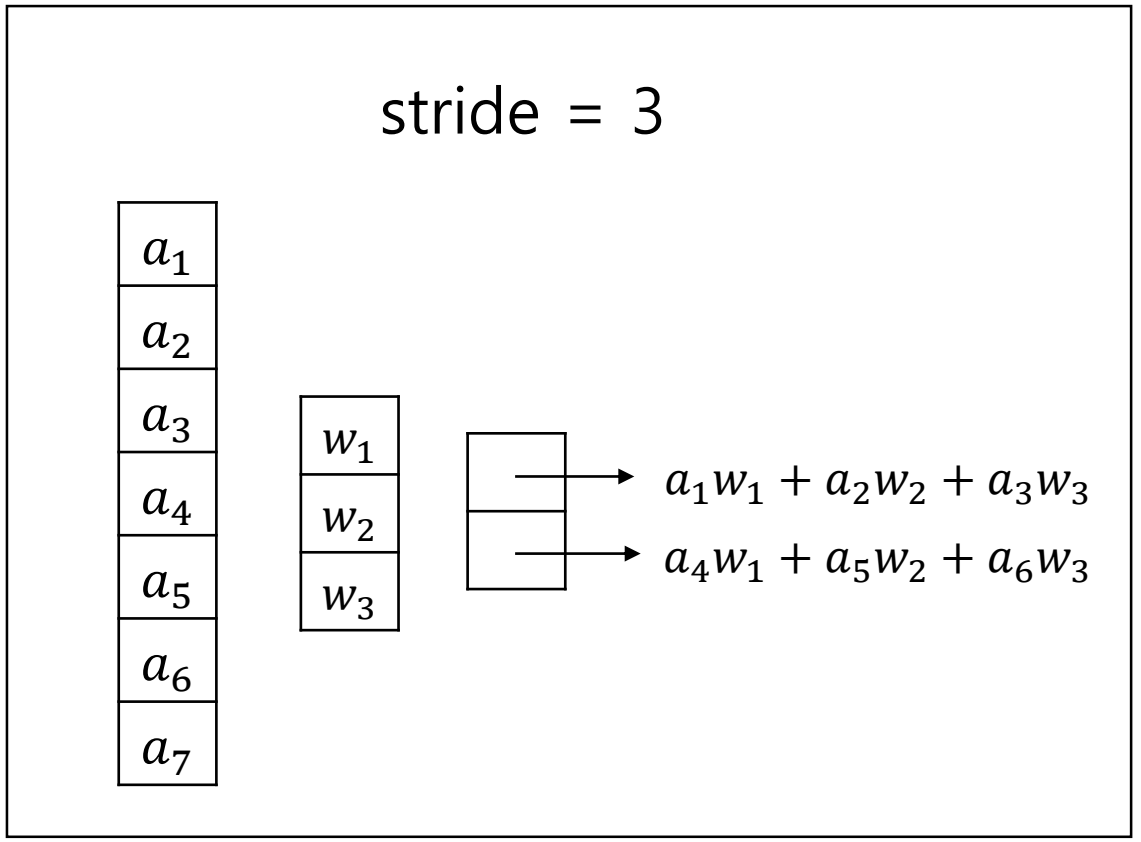
$$V(x, y) = (I \cdot K)(x, y) = \sum_m \sum_n I(x + m, y + n) K(m, n)$$

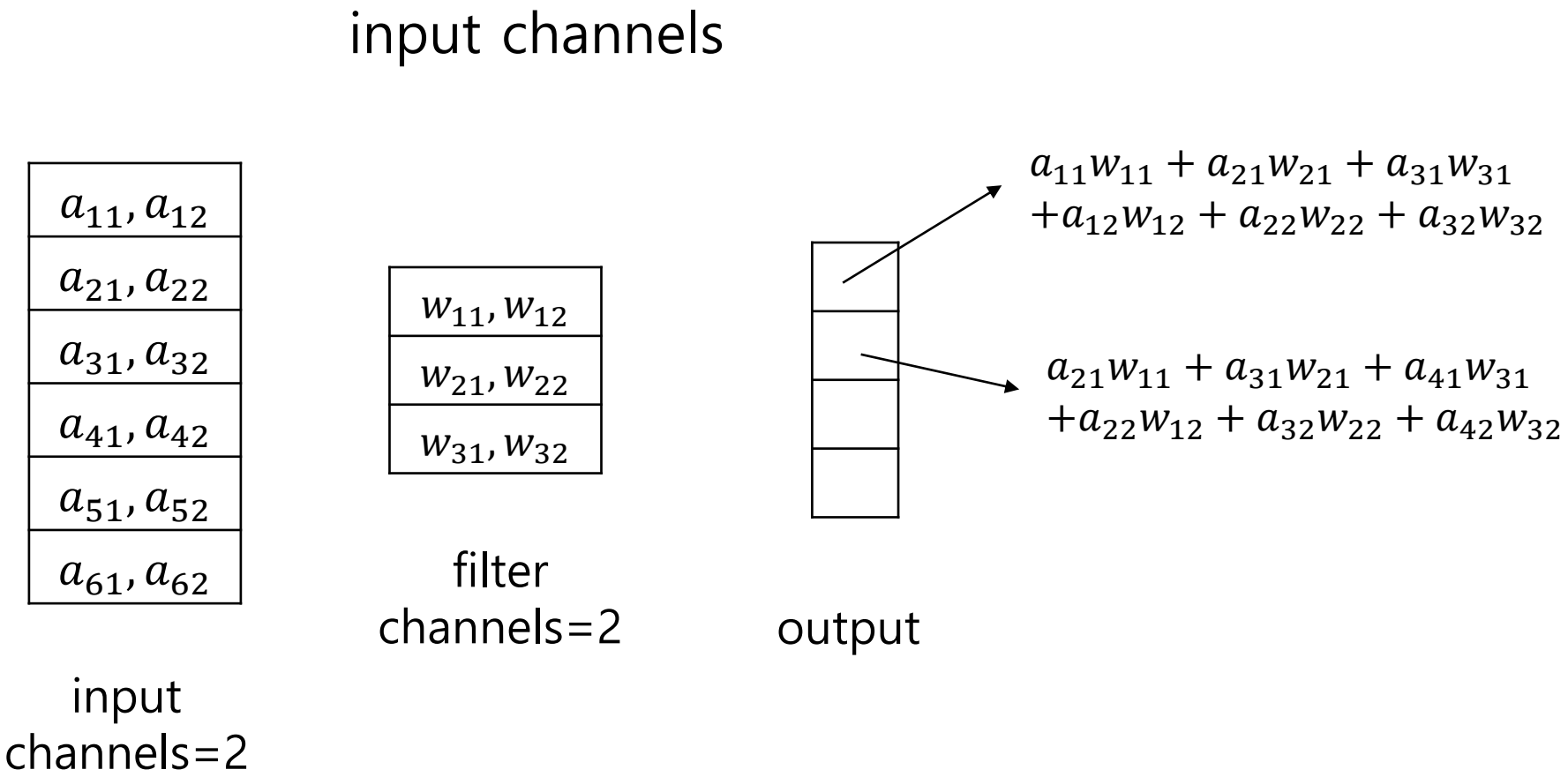
1-dimensional CNN



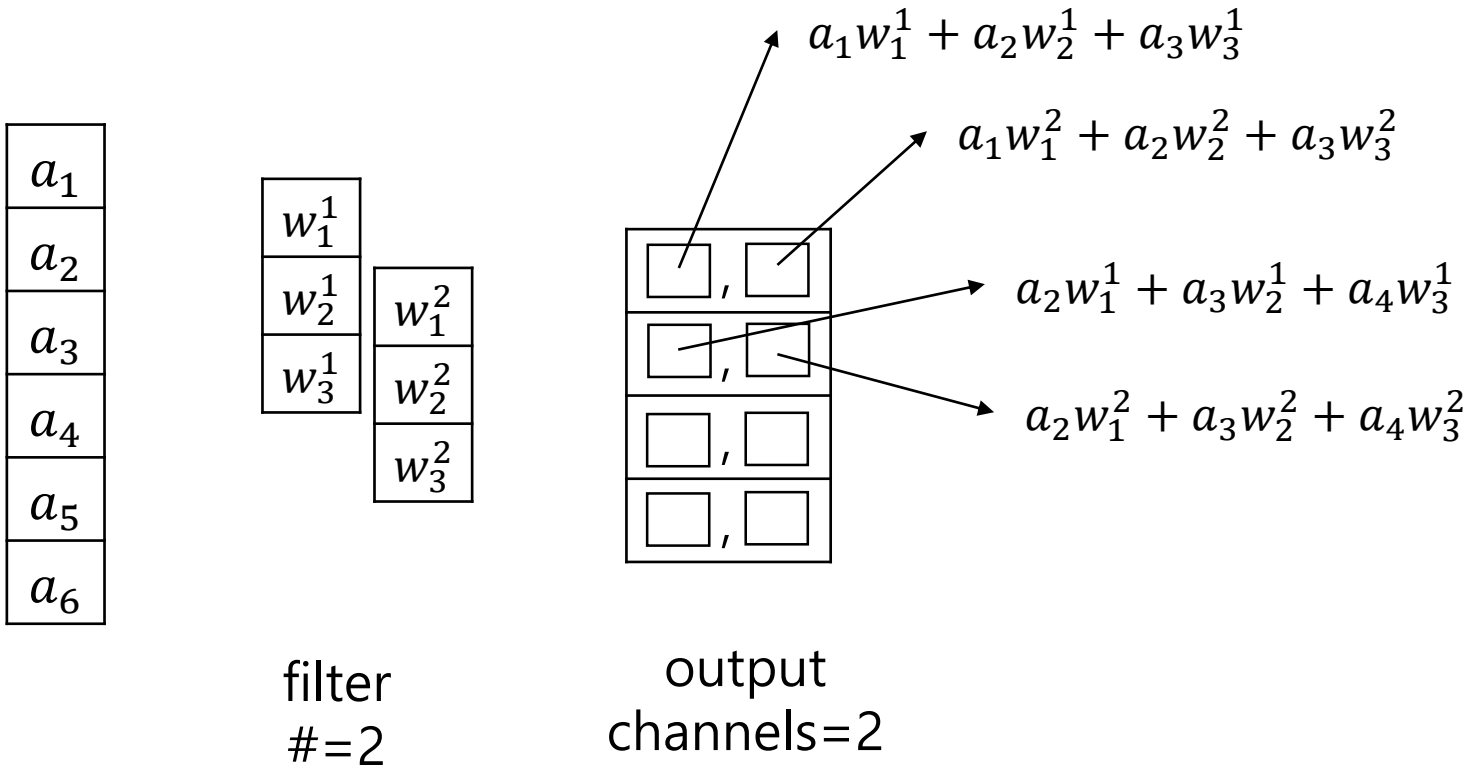

padding='VALID'
a_1
a_2
a_3
a_4
a_5
a_6
w_1
w_2
w_3
$a_1w_1 + a_2w_2 + a_3w_3$
$a_2w_1 + a_3w_2 + a_4w_3$
padding='SAME'
0
a_1
a_2
a_3
a_4
a_5
a_6
0
w_1
w_2
w_3
$0w_1 + a_1w_2 + a_2w_3$
$a_1w_1 + a_2w_2 + a_3w_3$
$a_2w_1 + a_3w_2 + a_4w_3$


stride = 2
a_1
a_2
a_3
a_4
a_5
a_6
a_7
w_1
w_2
w_3
$a_1w_1 + a_2w_2 + a_3w_3$
$a_3w_1 + a_4w_2 + a_5w_3$


stride = 3
a_1
a_2
a_3
a_4
a_5
a_6
a_7
w_1
w_2
w_3
$a_1w_1 + a_2w_2 + a_3w_3$
$a_4w_1 + a_5w_2 + a_6w_3$


input channels
a_{11}, a_{12}
a_{21}, a_{22}
a_{31}, a_{32}
a_{41}, a_{42}
a_{51}, a_{52}
a_{61}, a_{62}
input channels=2
w_{11}, w_{12}
w_{21}, w_{22}
w_{31}, w_{32}
filter channels=2
$a_{11}w_{11} + a_{21}w_{21} + a_{31}w_{31}$
$+a_{12}w_{12} + a_{22}w_{22} + a_{32}w_{32}$
$a_{21}w_{11} + a_{31}w_{21} + a_{41}w_{31}$
$+a_{22}w_{12} + a_{32}w_{22} + a_{42}w_{32}$
output

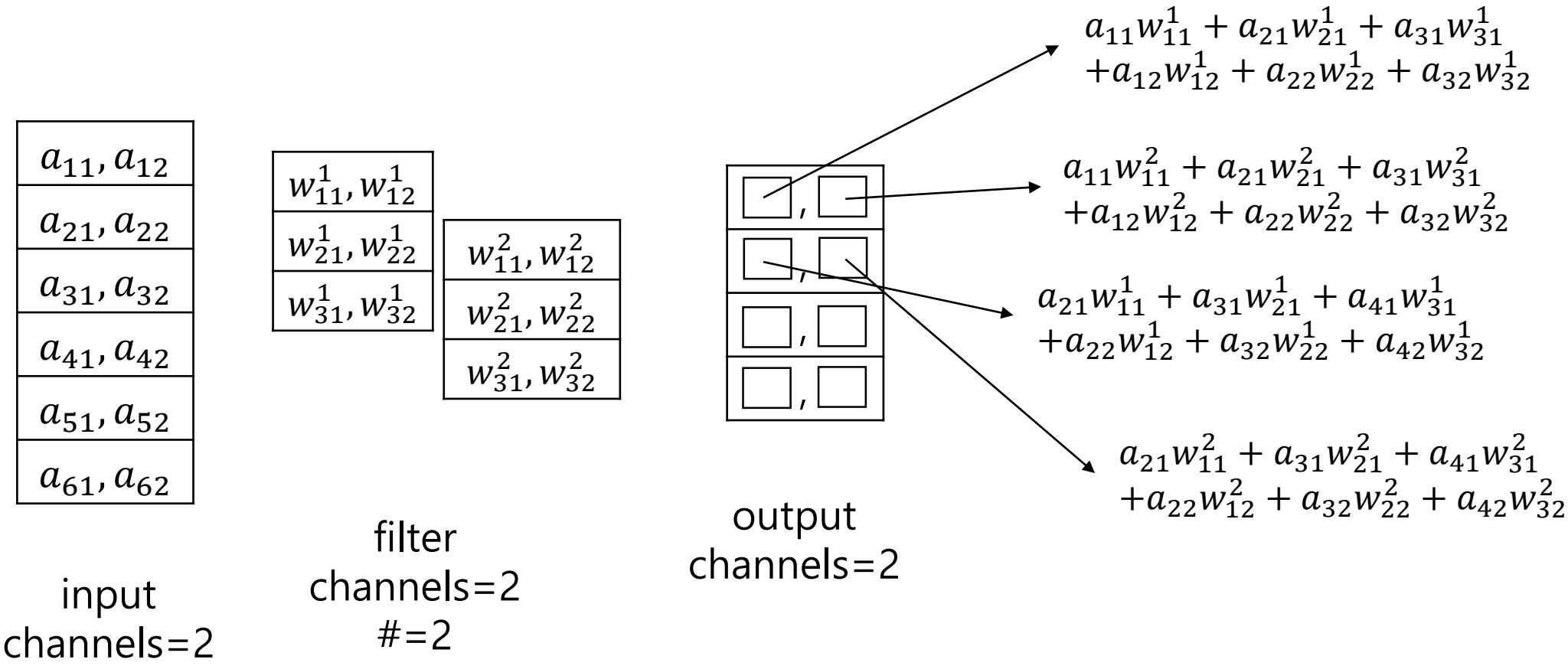
output channels
=# of filters



filter
#=2

output
channels=2

input channels and output channels

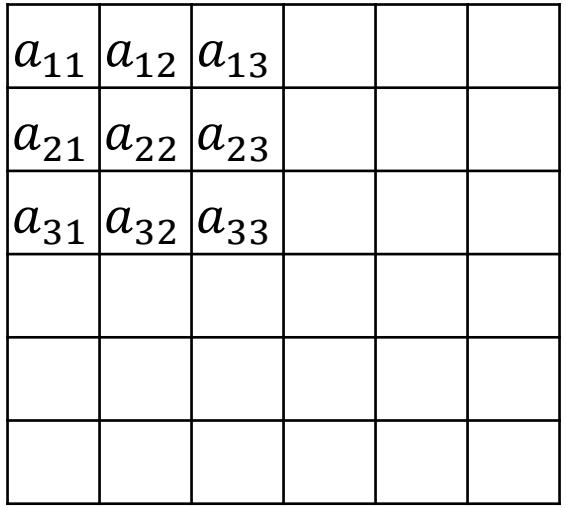


input channels=2

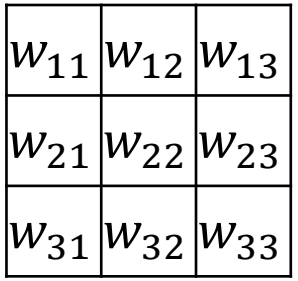
filter channels=2 #=2

output channels=2

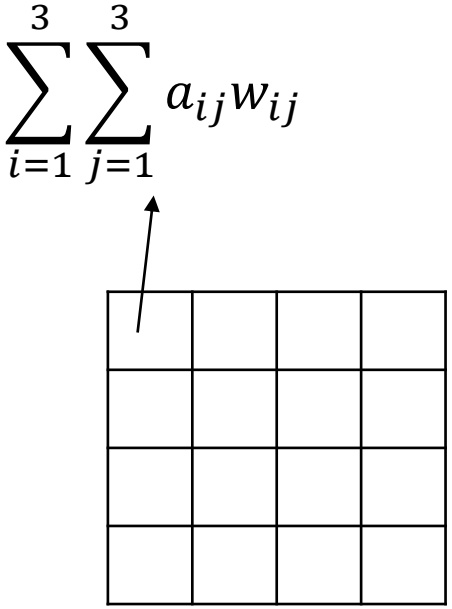
2-dimensional CNN



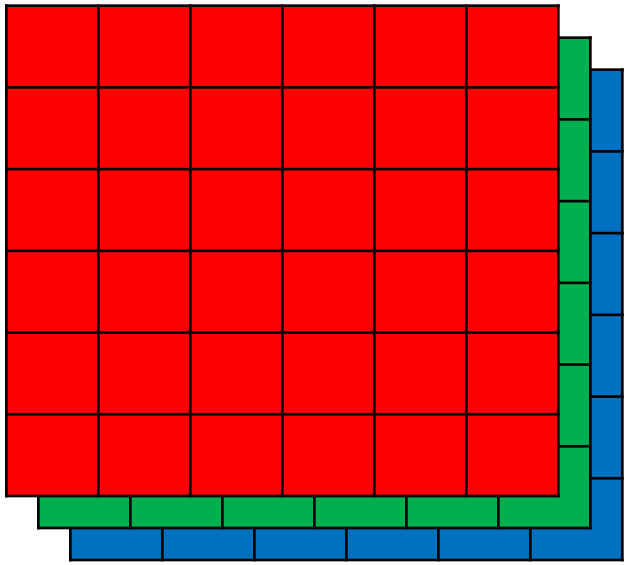
input



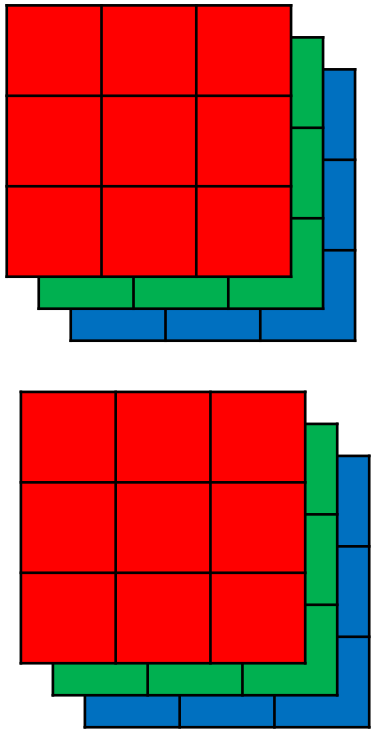
filter



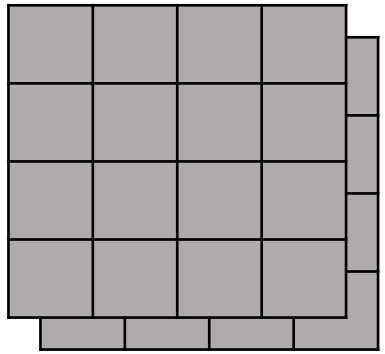
output



input
channels=3



filter
channels=3
#=2



output
channels=2

- Stride s_h, s_v
- Padding ‘SAME’, ‘VALID’
- Tensorflow code

```
tf.nn.conv2d(input, filters, strides, padding)
```

3.2 A Simple TF Convolution Example

```
ii = [[[[0], [0], [2], [2]],
      [[0], [0], [2], [2]],
      [[0], [0], [2], [2]],
      [[0], [0], [2], [2]]]]
I = tf.constant(ii, tf.float32)
print(I.shape) # (1, 4, 4, 1) – batch size, width, height, input channels
ww = [[[[-1]], [[-1]], [[1]]],
      [[[-1]], [[-1]], [[1]]],
      [[[-1]], [[-1]], [[1]]]]
W = tf.constant(ww, tf.float32)
print(W.shape) # (3, 3, 1, 1) – width, height, input channels, # of filters
C = tf.nn.conv2d(I, W, strides=[1, 1, 1, 1], padding='VALID')
sess = tf.Session()
print(res.shape)
print(res) # (1, 2, 2, 1) – batch size, width, height, output channels
```

0	0	2	2
0	0	2	2
0	0	2	2
0	0	2	2

−1	−1	1
−1	−1	1
−1	−1	1

$$0 * -1 + 0 * -1 + 2 * 1 + 0 * -1 + 0 * -1 + 2 * 1 + 0 * -1 + 0 * -1 + 2 * 1$$

6	0
6	0

$$0 * -1 + 2 * -1 + 2 * 1 + 0 * -1 + 2 * -1 + 2 * 1 + 0 * -1 + 2 * -1 + 2 * 1$$

Modified Figure 2.2

```
# ================================================================
image = tf.reshape(img, [batchSz, 28, 28, 1])
flts = tf.Variable(tf.truncated_normal([4, 4, 1, 4], stddev=0.1))
convOut = tf.nn.conv2d(image, flts, [1, 2, 2, 1], 'SAME’)
convOut = tf.nn.relu(convOut)
print(convOut.shape, 'batch size, width, height, numFlters=output channels')
convOut = tf.reshape(convOut, [100, 784])
prbs = tf.nn.softmax(tf.matmul(convOut, W) + b)
# prbs = tf.nn.softmax(tf.matmul(img, W) + b)
# ================================================================
```

Output

(100, 14, 14, 4) batch size, width, height, numFlters=output channels
Test Accuracy: 0.9625000046491623

3.3 Multilevel Convolution

```
# ====================================================
image = tf.reshape(img, [batchSz, 28, 28, 1])
flts = tf.Variable(tf.random_normal([4, 4, 1, 16], stddev=0.1))
convOut = tf.nn.conv2d(image, flts, [1, 2, 2, 1], "SAME")
convOut = tf.nn.relu(convOut)
flts2 = tf.Variable(tf.random_normal([2, 2, 16, 32], stddev=0.1))
convOut2 = tf.nn.conv2d(convOut, flts2, [1, 2, 2, 1], "SAME")
convOut2 = tf.reshape(convOut2, [100, 1568])
prbs = tf.nn.softmax(tf.matmul(convOut2, W) + b)
# prbs = tf.nn.softmax(tf.matmul(img, W) + b)
# ====================================================
```

Test Accuracy: 0.9663000059127808

3.4 Convolution Details

- 3.4.1 Biases

```
bias = tf.Variable(tf.zeros [16])
convOut += bias
```

  - Number of output filters = 16
- 3.4.2 Layers with Convolution

```
tf.contrib.layers.conv2d(inpt, numFlts, fltDim, strides, pad)
convOut = layers.conv2d(image, 16, [4,4], 2, "Same")
```

- 3.4.3 Pooling

```
convOut = tf.nn.conv2d(image, flts, [1,1,1,1], "SAME")
convOut = tf.nn.max pool(convOut, [1,2,2,1], [1,2,2,1], "SAME")
```