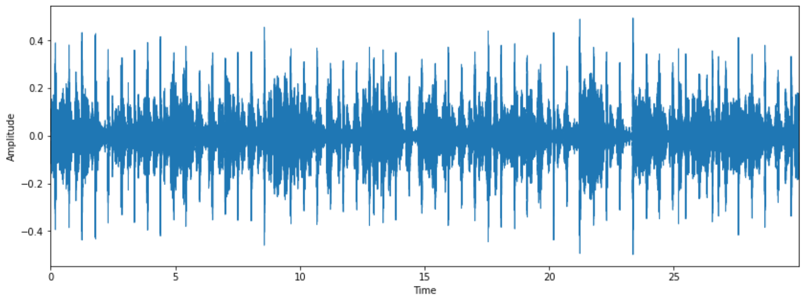
Convolutional Neural Network (CNN)

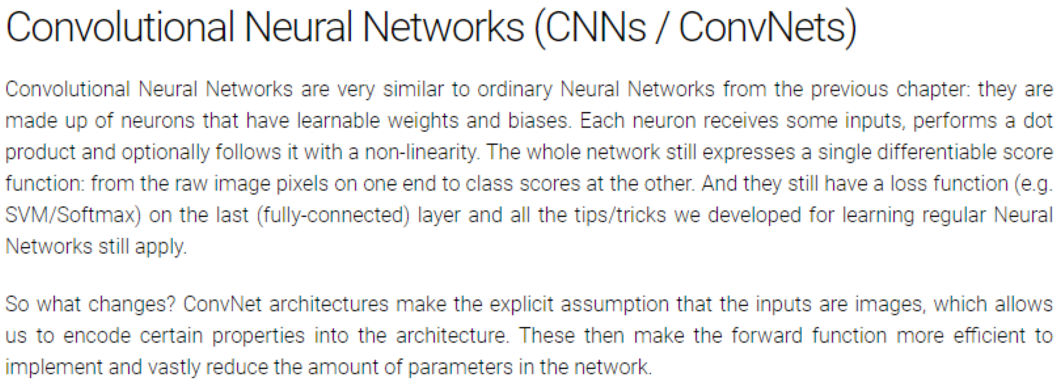
Minjong Lee

Pohang University of Science and Technology (Postech) CSE

Minjong.Lee@postech.ac.kr

Type of data





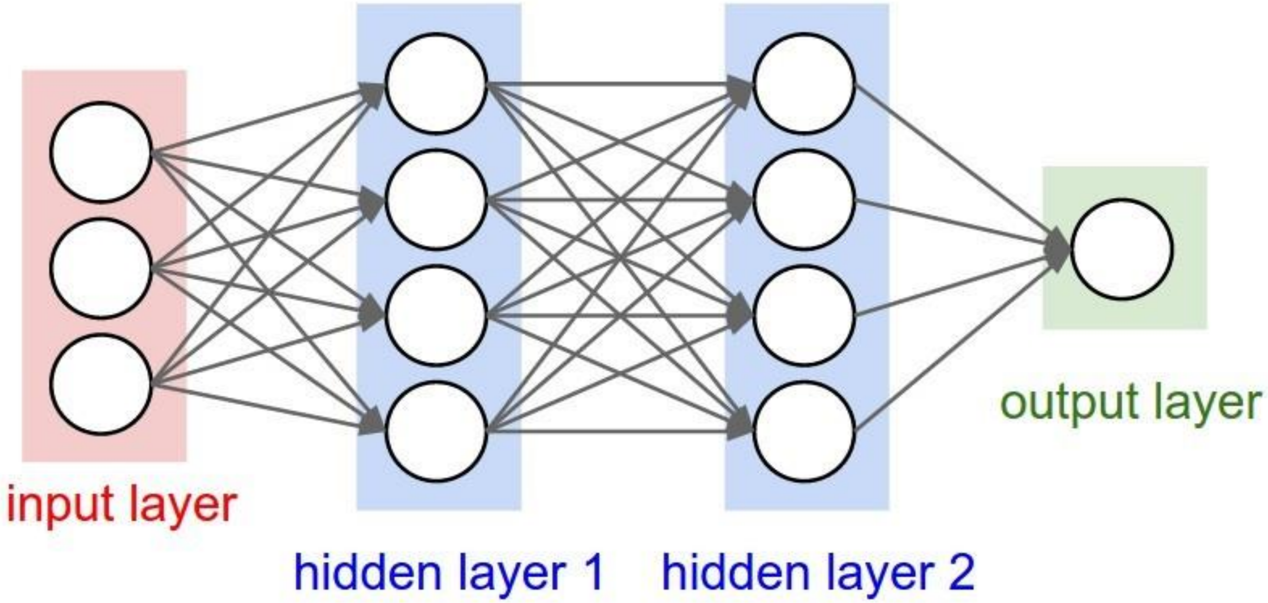
Convolutional Neural Networks (CNNs / ConvNets)

Convolutional Neural Networks are very similar to ordinary Neural Networks from the previous chapter: they are made up of neurons that have learnable weights and biases. Each neuron receives some inputs, performs a dot product and optionally follows it with a non-linearity. The whole network still expresses a single differentiable score function: from the raw image pixels on one end to class scores at the other. And they still have a loss function (e.g. SVM/Softmax) on the last (fully-connected) layer and all the tips/tricks we developed for learning regular Neural Networks still apply.

So what changes? ConvNet architectures make the explicit assumption that the inputs are images, which allows us to encode certain properties into the architecture. These then make the forward function more efficient to implement and vastly reduce the amount of parameters in the network.

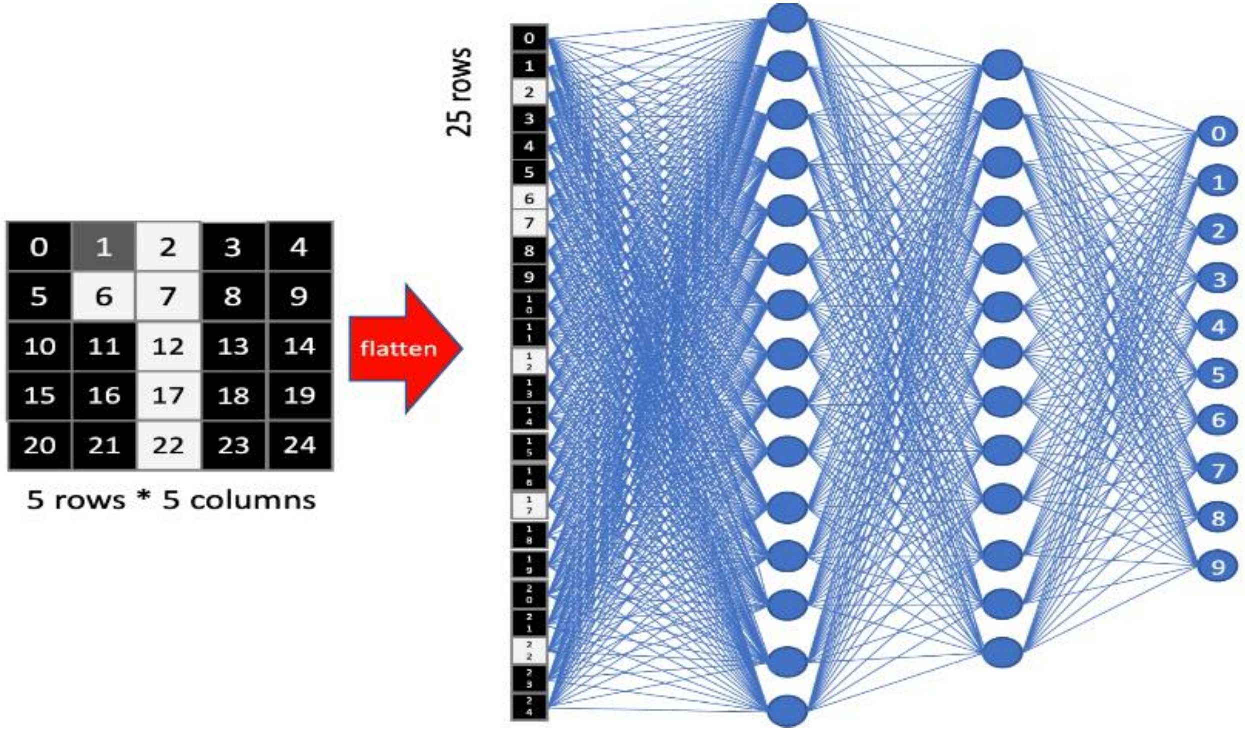
Deep Neural Network

- A three layers neural network with fully-connected (FC) layers:



http://cs231n.github.io/convolutional-networks/

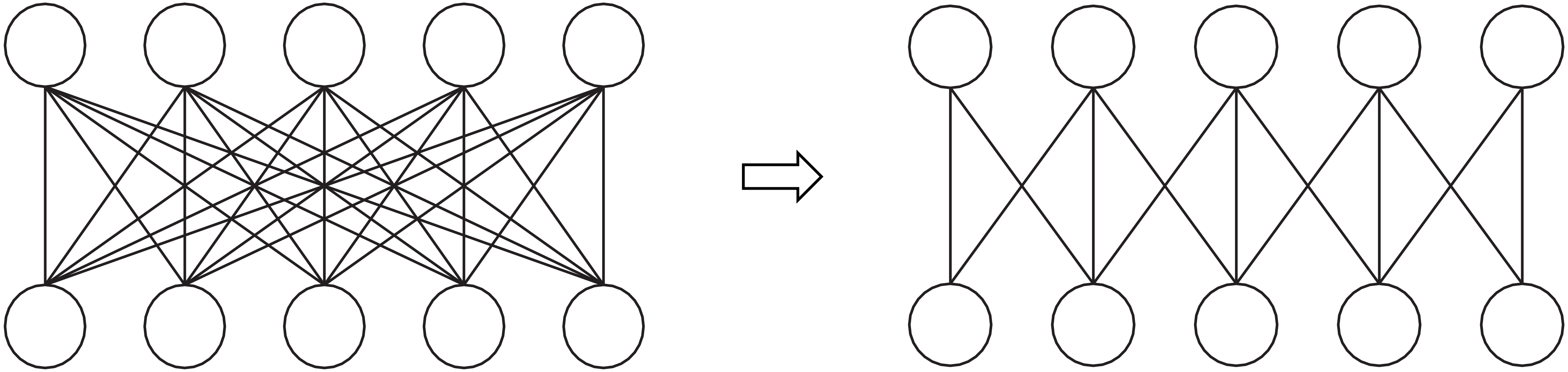
An image with FC layers



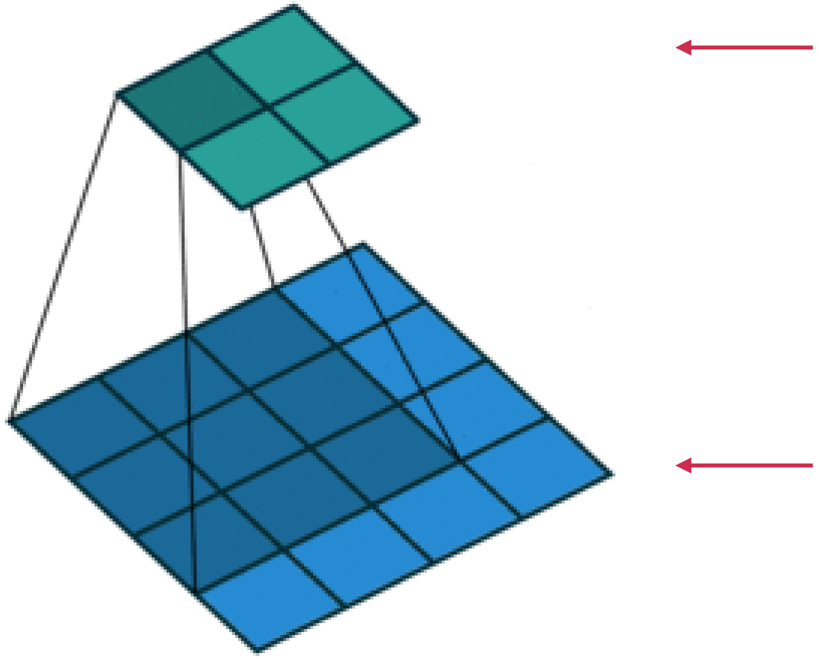
- 256 x 256 x 3 input image?

https://www.youtube.com/playlist?list=PLVNY1HnUlO241gILgQloWAs0xrrkqQfKe

Convolutional layer



Convolution



output

input

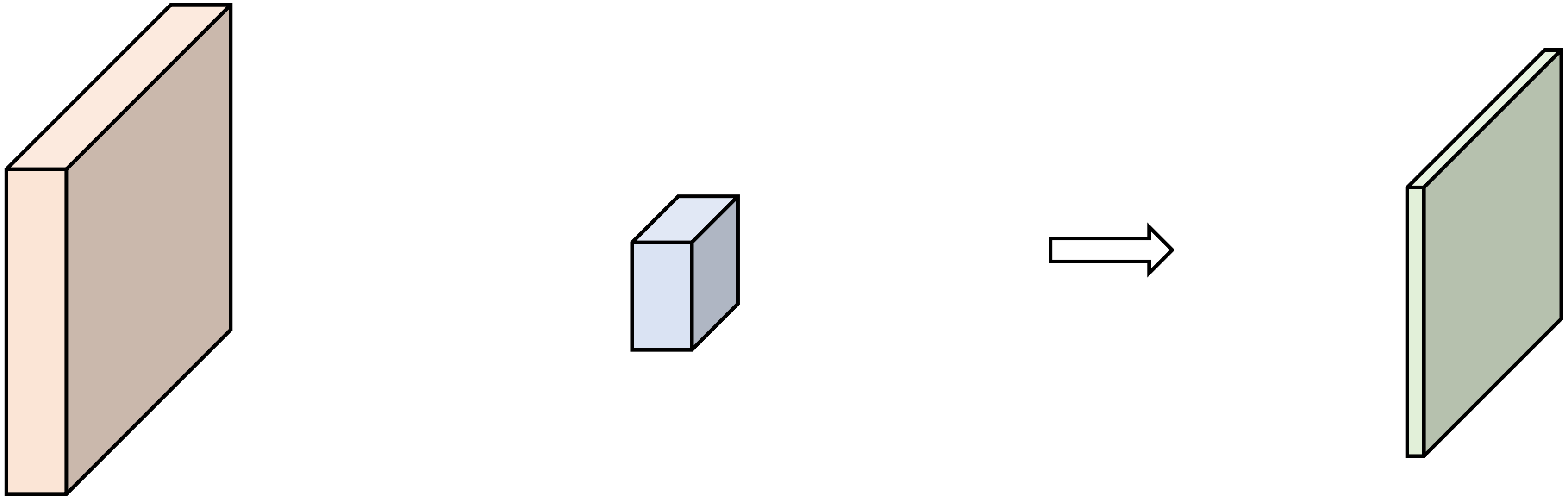
- **Local Connectivity**
  - We connect each neuron to only a local region of the input (**receptive field**).

Contents

- Convolutional layer
- Pooling layer
- CNN architecture

Convolutional layer

- Convolutional layer consists of a set of learnable filters (or kernels).
  - Convolution layer computes **dot product** between their weights and a small region they are connected to in the input volume.



32 x 32 x 3 Image

5 x 5 x 3 kernel

28 x 28 x 1 feature map

Convolution Layer

- Assume that there are 5 x 5 x 1 input image, 3 x 3 x 1 kernel.

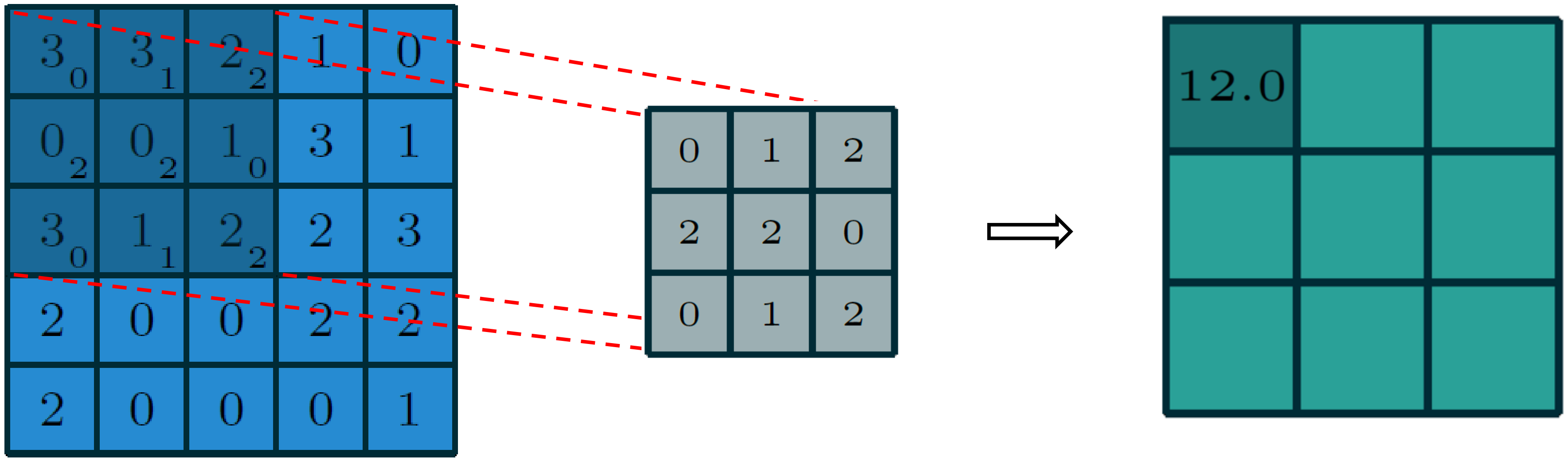
3	3	2	1	0
0	0	1	3	1
3	1	2	2	3
2	0	0	2	2
2	0	0	0	1

5 x 5 x 1 input image

0	1	2
2	2	0
0	1	2

3 x 3 x 1 kernel

Convolution Layer



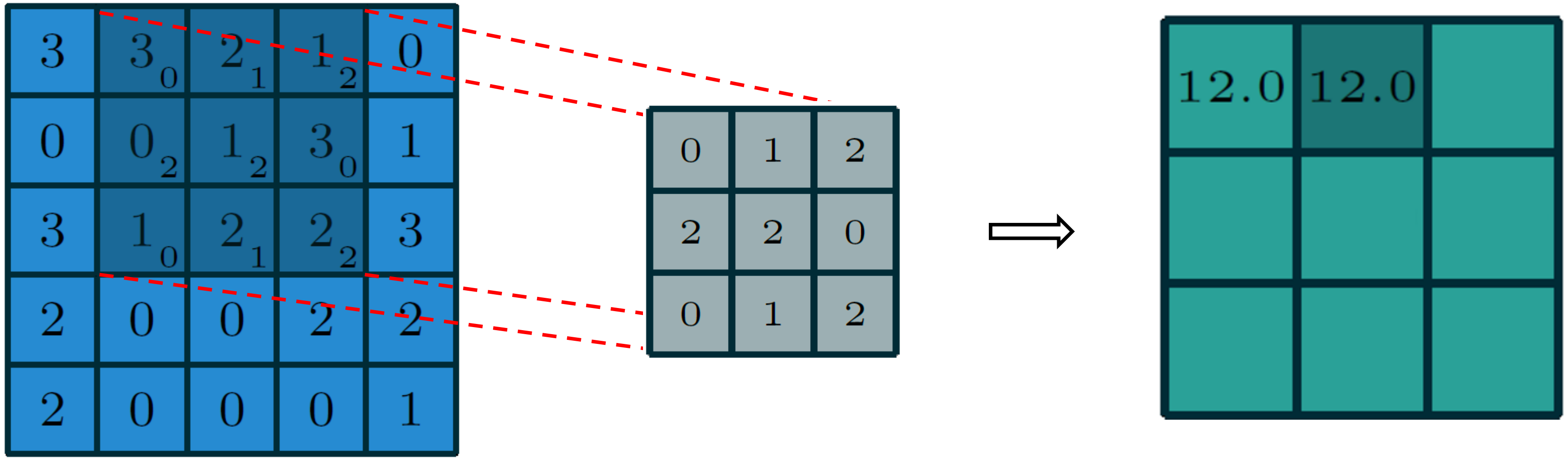
3_0	3_1	2_2	1	0
0_2	0_2	1_0	3	1
3_0	1_1	2_2	2	3
2	0	0	2	2
2	0	0	0	1

0	1	2
2	2	0
0	1	2

12.0		

(3 X 0) + (1 X 1) + (2 X 2) + (0 X 2) + (0 X 2) + (1 X 0) + (3 X 0) + (1 X 1) + (2 X 2) = 12

Convolution Layer



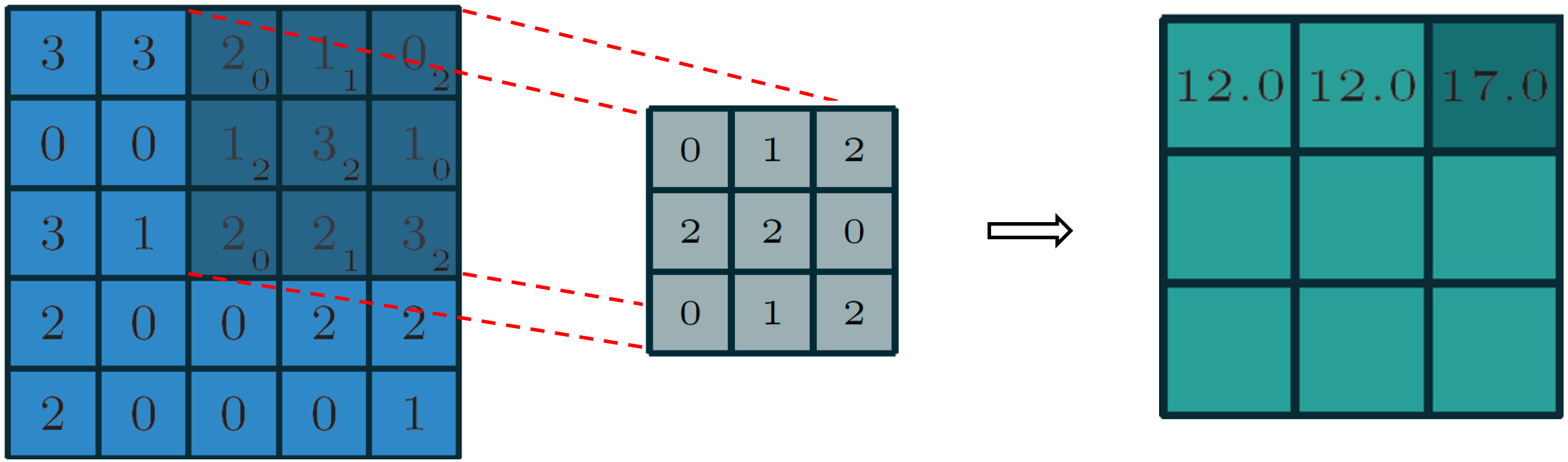
3	3_0	2_1	1_2	0
0	0_2	1_2	3_0	1
3	1_0	2_1	2_2	3
2	0	0	2	2
2	0	0	0	1

0	1	2
2	2	0
0	1	2

12.0	12.0	

(3 X 0) + (2 X 1) + (1 X 2) + (0 X 2) + (1 X 2) + (3 X 0) + (1 X 0) + (2 X 1) + (2 X 2) = 12

Convolution Layer



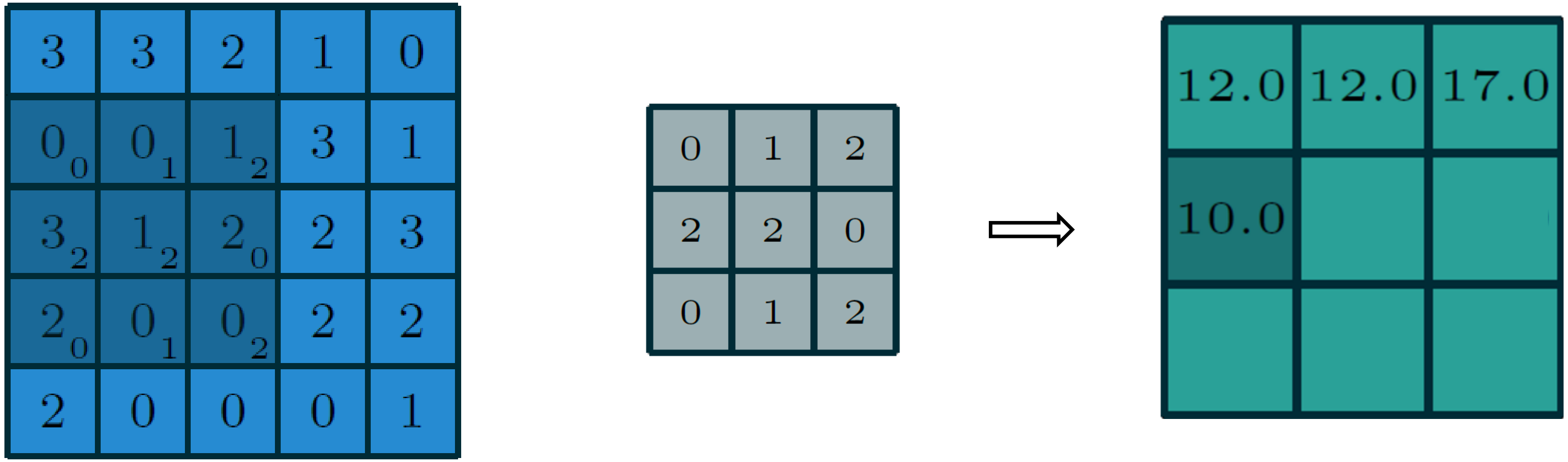
3	3	2_0	1_1	0_2
0	0	1_2	3_2	1_0
3	1	2_0	2_1	3_2
2	0	0	2	2
2	0	0	0	1

0	1	2
2	2	0
0	1	2

12.0	12.0	17.0

(2 X 0) + (1 X 1) + (0 X 2) + (1 X 2) + (3 X 2) + (1 X 0) + (2 X 0) + (2 X 1) + (3 X 2) = 17

Convolution Layer



3	3	2	1	0
0_0	0_1	1_2	3	1
3_2	1_2	2_0	2	3
2_0	0_1	0_2	2	2
2	0	0	0	1

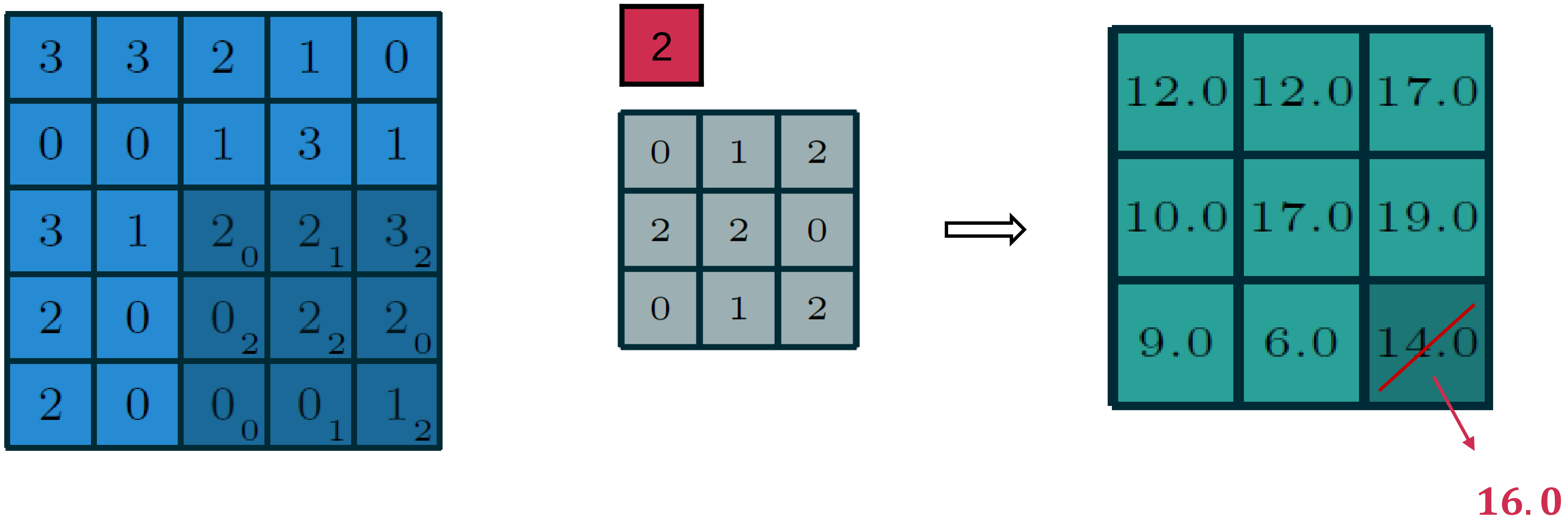
0	1	2
2	2	0
0	1	2

⟹

12.0	12.0	17.0
10.0		

(0 x 0) + (0 x 1) + (1 x 2) + (3 x 2) + (1 x 2) + (2 x 0) + (2 x 0) + (0 x 1) + (0 x 2) = 10

Convolution Layer with a bias term



3	3	2	1	0
0	0	1	3	1
3	1	2_0	2_1	3_2
2	0	0_2	2_2	2_0
2	0	0_0	0_1	1_2

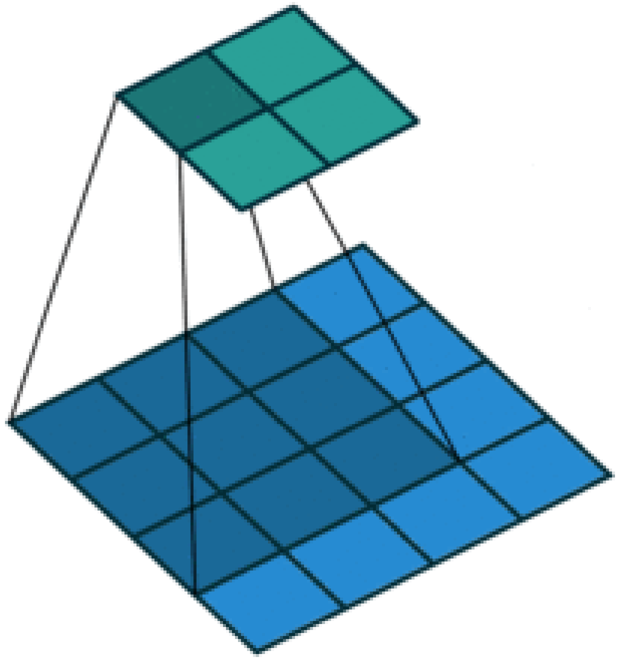
0	1	2
2	2	0
0	1	2

$\Longrightarrow$

12.0	12.0	17.0
10.0	17.0	19.0
9.0	6.0	~~14.0~~

(2 x 0) + (2 x 1) + (3 x 2) + (0 x 2) + (2 x 2) + (2 x 0) + (0 x 0) + (0 x 1) + (1 x 2) **+ 2** = **16**

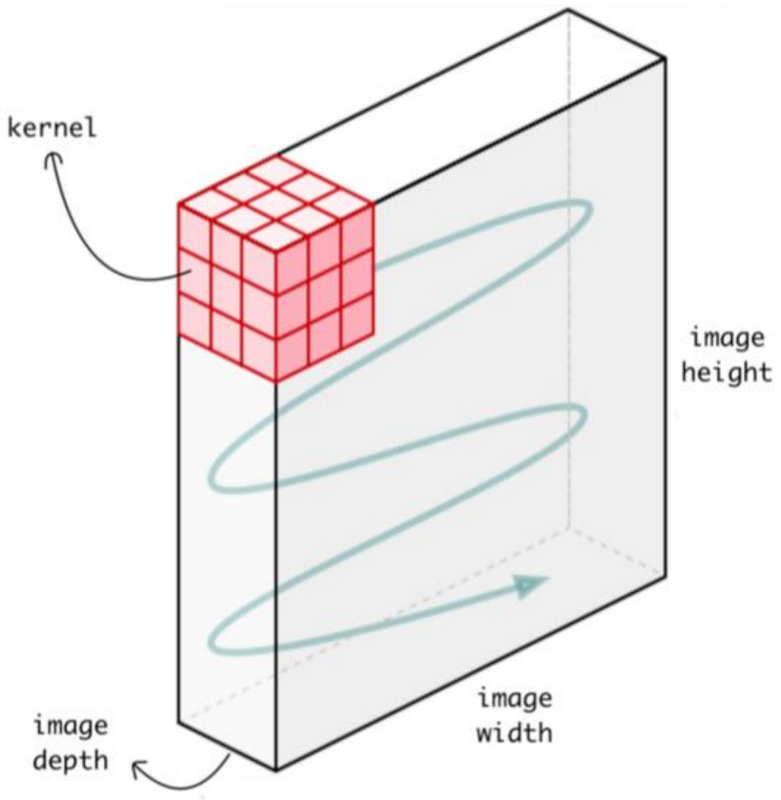
Convolution layer



output

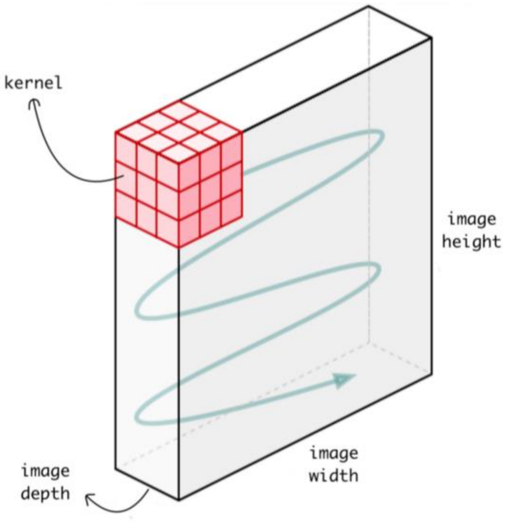
input

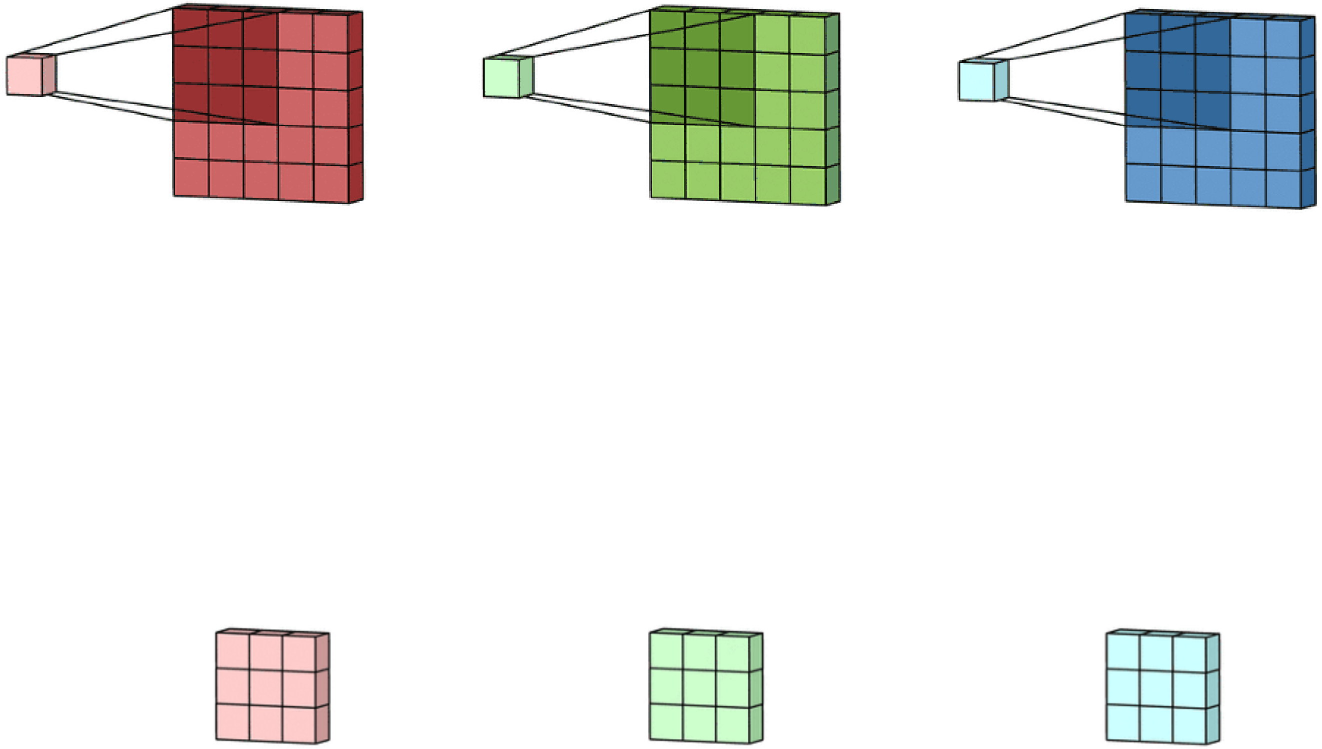
If channels = 3



If channels = 3

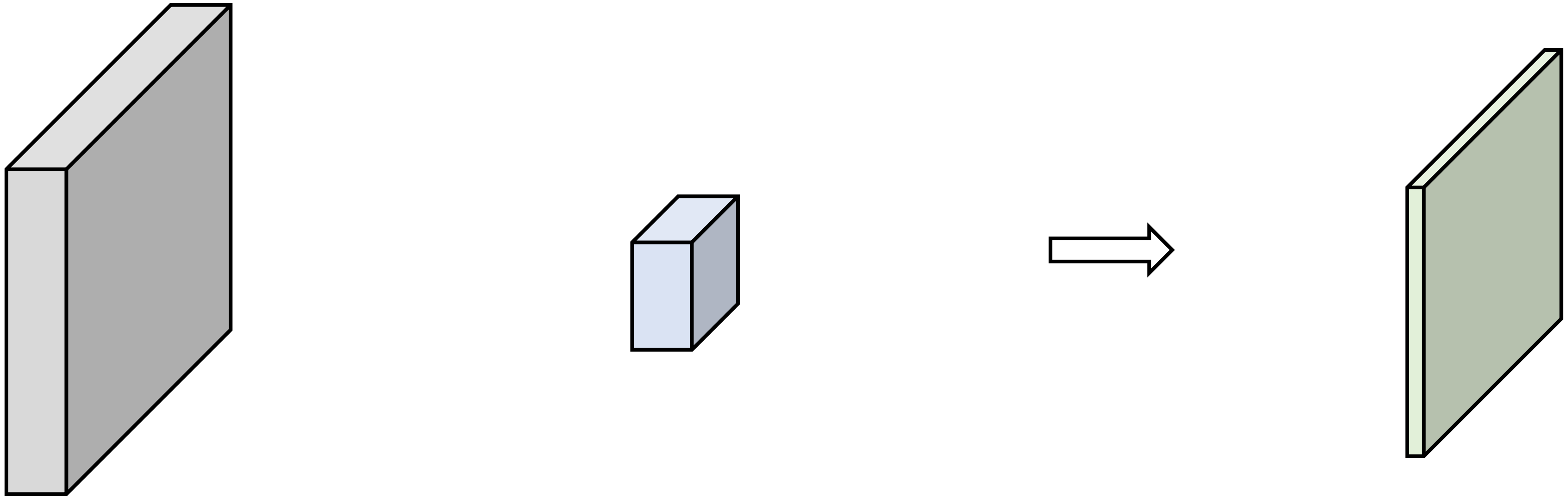
- Sum all the feature maps.





The number of parameters

- The number of parameters:
  - 5 x 5 x 3 = 75 (and +1 bias parameter)
- Parameter sharing!



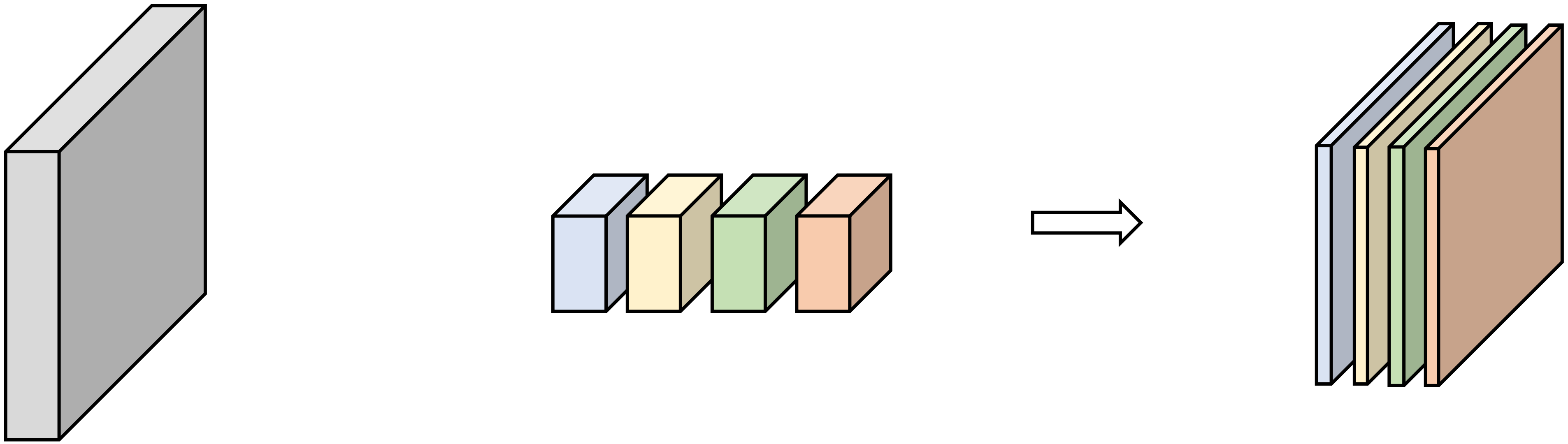
32 x 32 x 3 Image

5 x 5 x 3 kernel

28 x 28 x 1 feature map

Convolution with more kernels

- The number of parameters:
  - (5 x 5 x 3) x 4 = 300 (and +4 bias parameters)



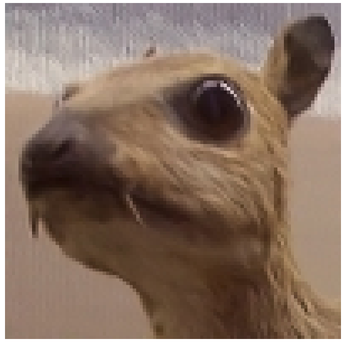
32 x 32 x 3 Image

(5 x 5 x 3) x 4 kernel

28 x 28 x 4 feature map

Learning kernels

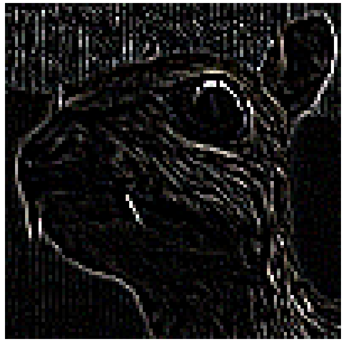
Input image



Convolution Kernel

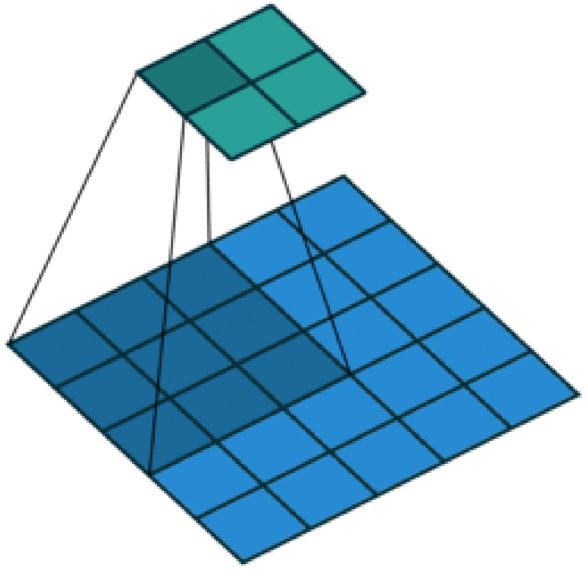
$$\begin{bmatrix} -1 & -1 & -1 \\ -1 & 8 & -1 \\ -1 & -1 & -1 \end{bmatrix}$$

Feature map



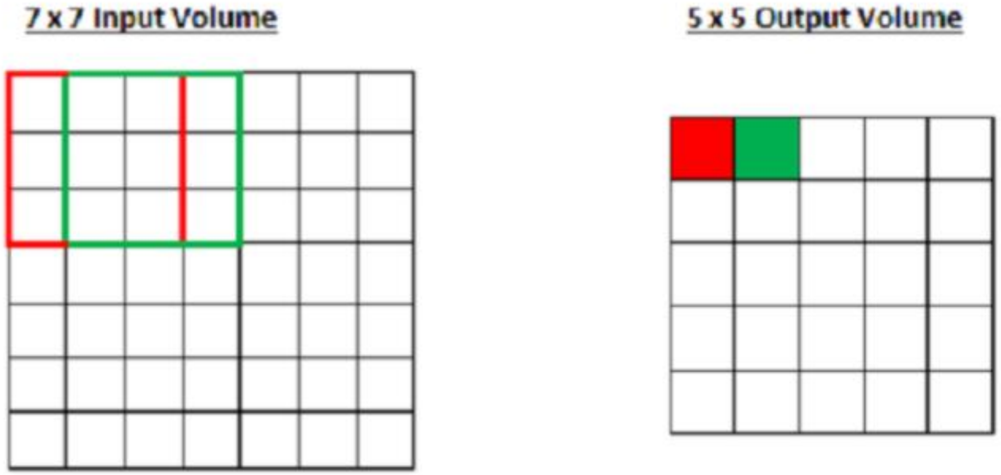
Convolution layer: stride

- Step size for the convolution operation.
- Reduce the size of output feature maps.

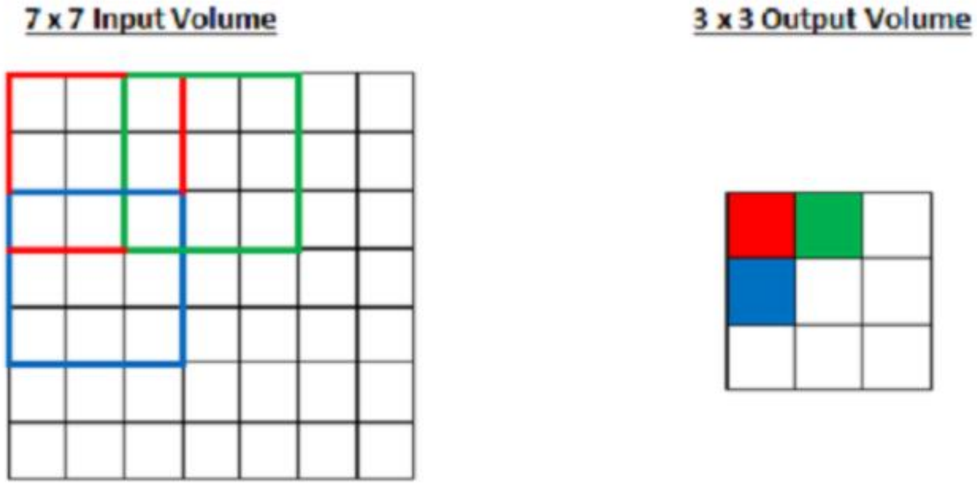


Example with kernel size 3x3 and a stride of 2

Convolution layer: stride



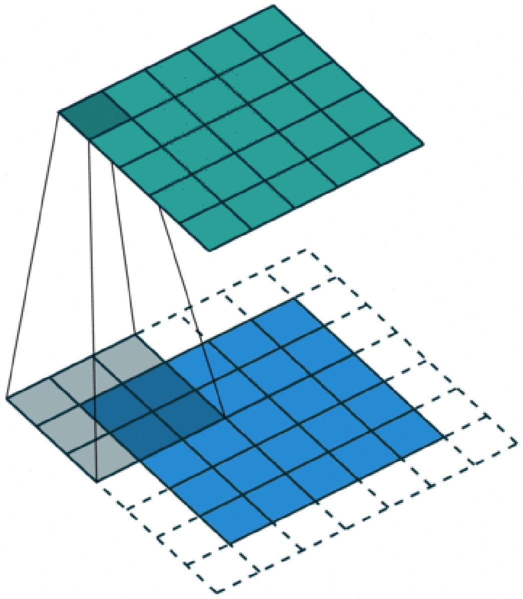
Stride with value 1 *(default)*



Stride with value 2

Convolution layer: padding

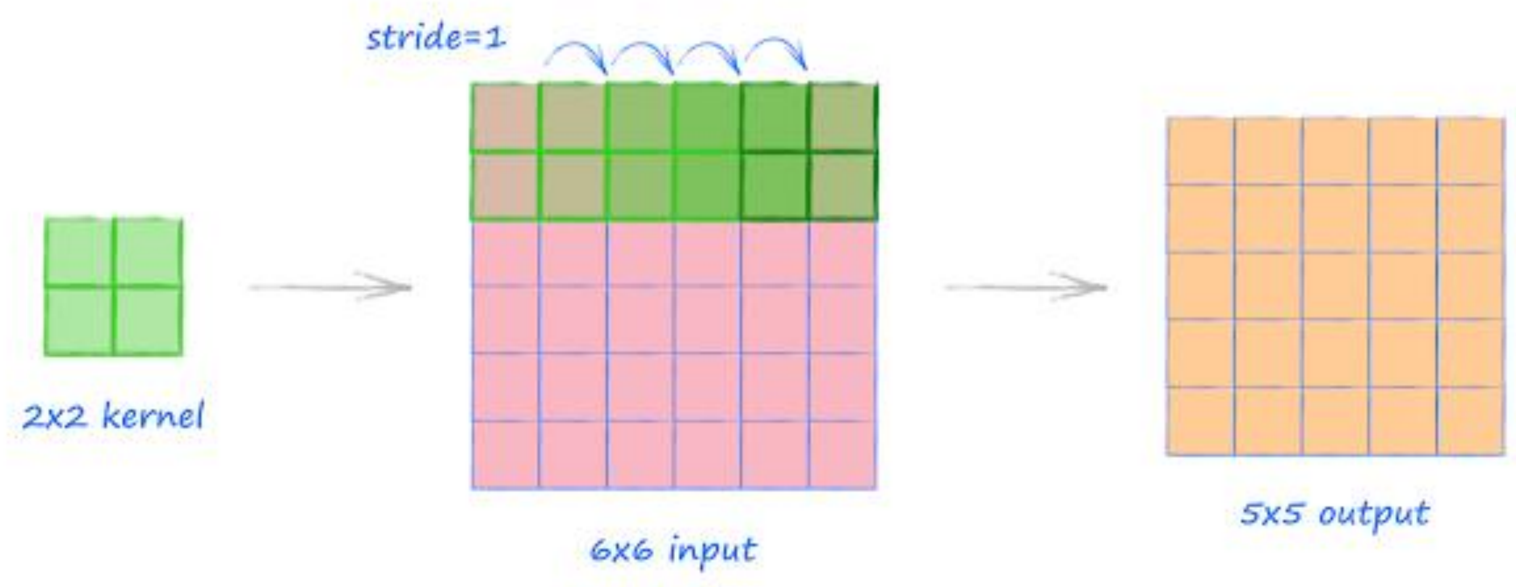
- The padding operation pad the input with some values around the border.
- Padding operation will allow us to control the spatial size of the output.
  - People usually fill the border with zeros (zero padding).



Output size of convolution

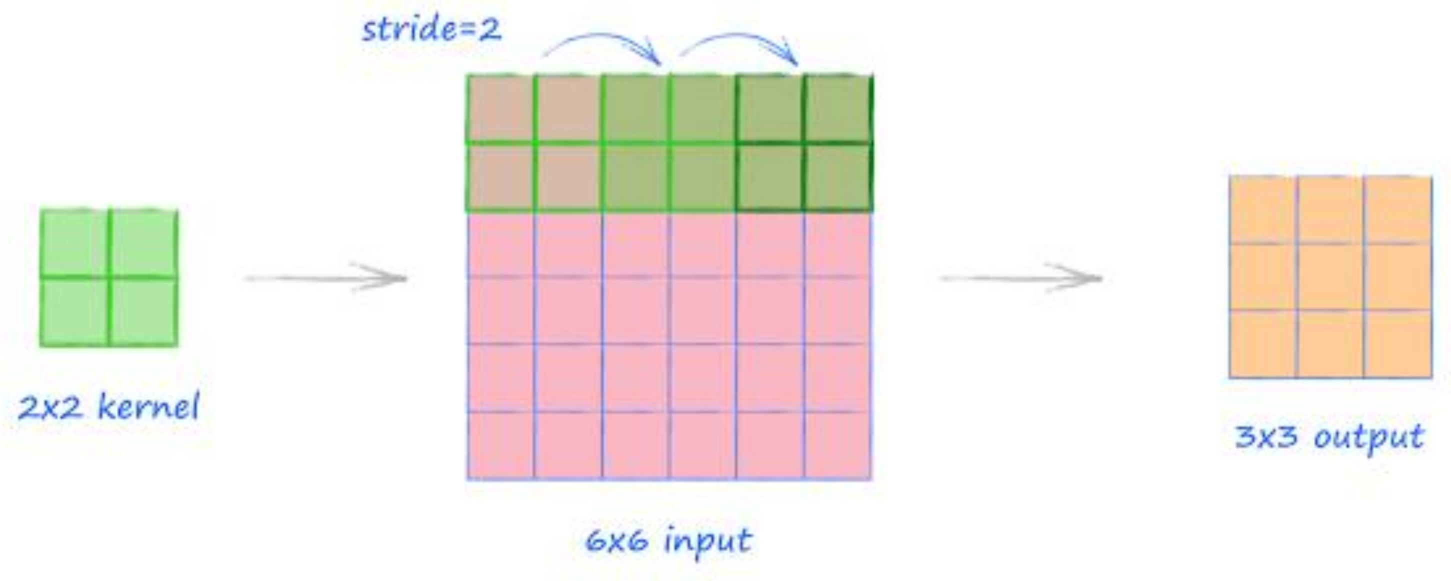
- Input size: $W_1 \ \times H_1 \times D_1$
- Hyper-parameters:
  - Number of filters K
  - Filter size F
  - Stride S
  - Zero padding P
- Output size:
  - $W_2 = (W_1 - F + 2P)/S + 1,$ $\mathrm{H}_2 = (\mathrm{H}_1 - \mathrm{F} + 2P)/S + 1$
  - $D_2 = K$

Output size of convolution



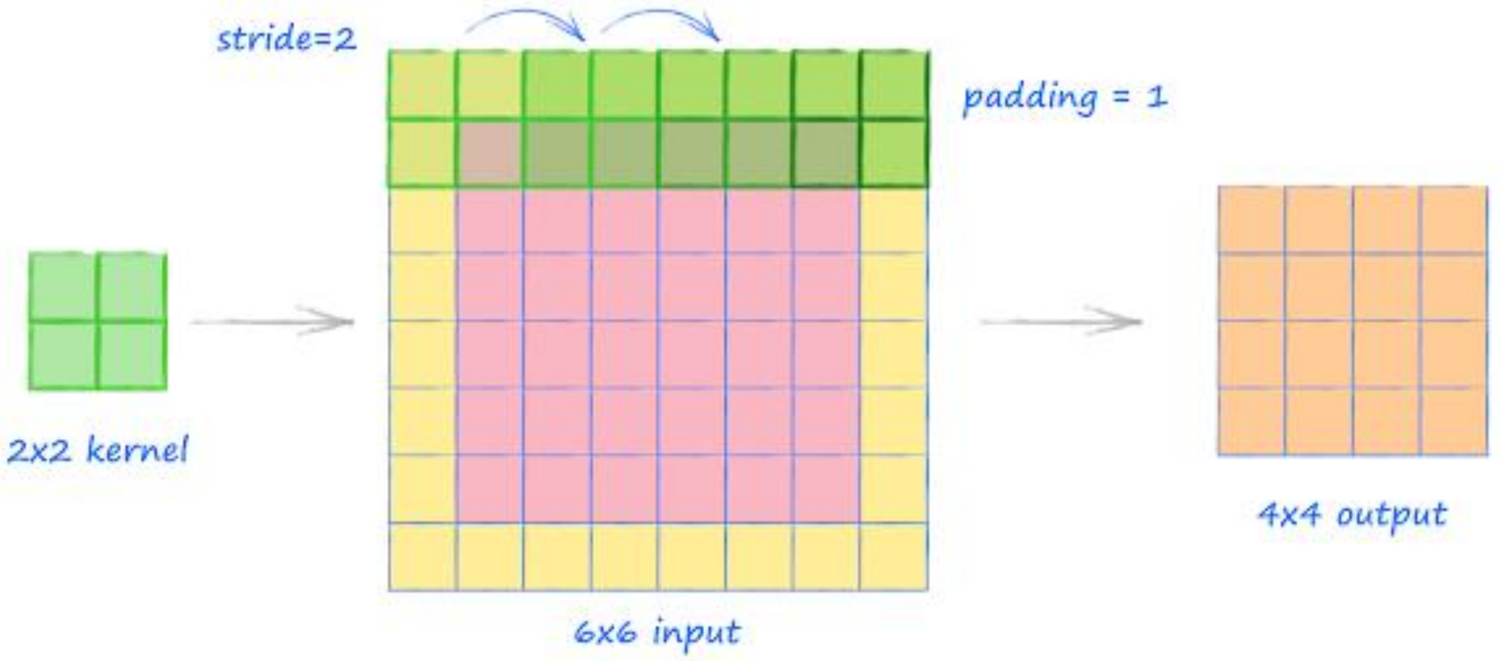
- Output of width = $(6-2+2\times 0)/1+1=5$
  - $W_2=(W_1-F+2P)/S+1$

Output size of convolution



- Output of width = $(6 - 2 + 2 \times 0)/2 + 1 = 3$
  - $W_2 = (W_1 - F + 2P)/S + 1$

Output size of convolution



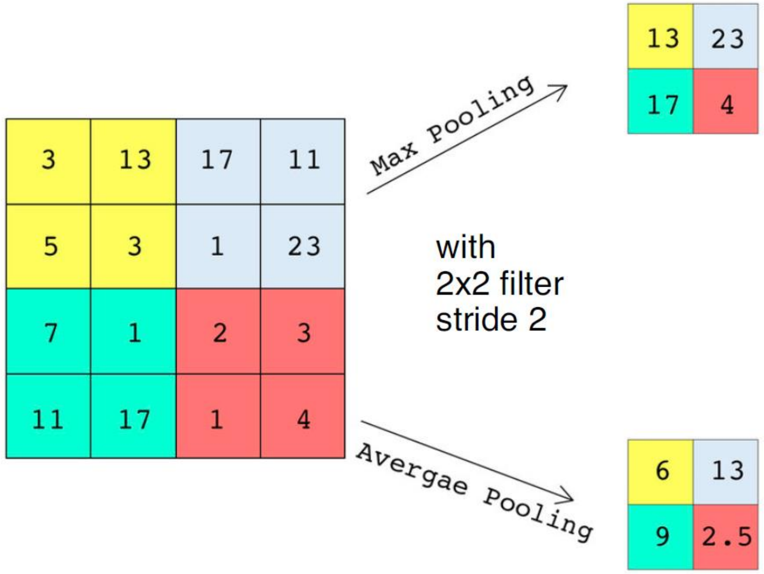
- Output of width = $(6 - 2 + 2 \times 1)/2 + 1 = 4$
  - $W_2 = (W_1 - F + 2P)/S + 1$

Contents

- Convolutional layer
- Pooling layer
- CNN architecture

Pooling layer

- Pooling layer computes a value in a sliding window
  - Max pooling
  - Average pooling
- Pooling layer reduce spatial resolution for faster computation

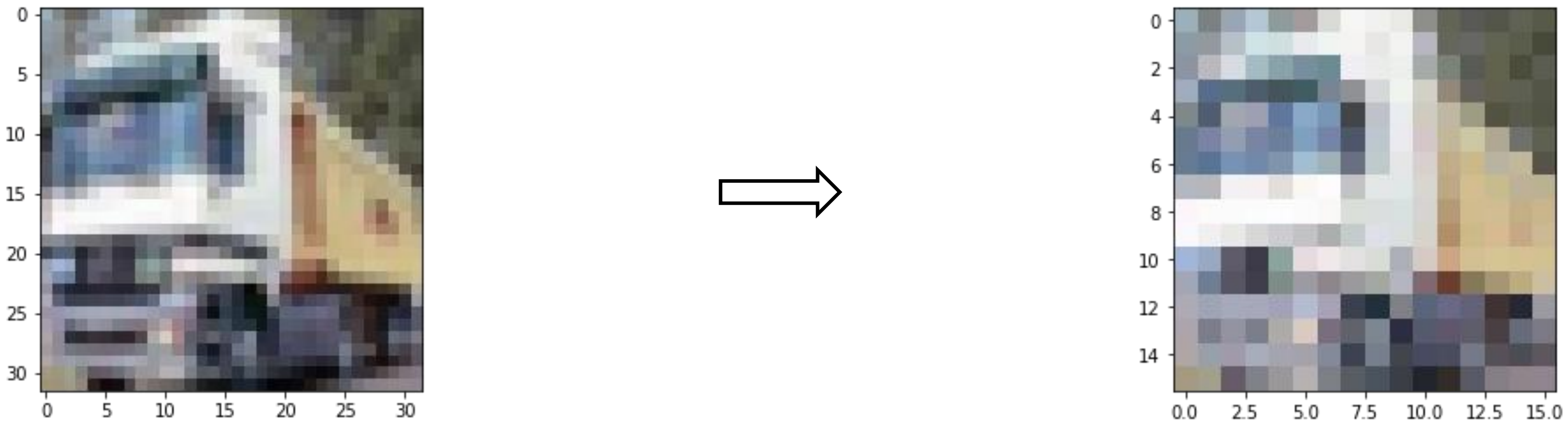


3	13	17	11
5	3	1	23
7	1	2	3
11	17	1	4

13	23
17	4

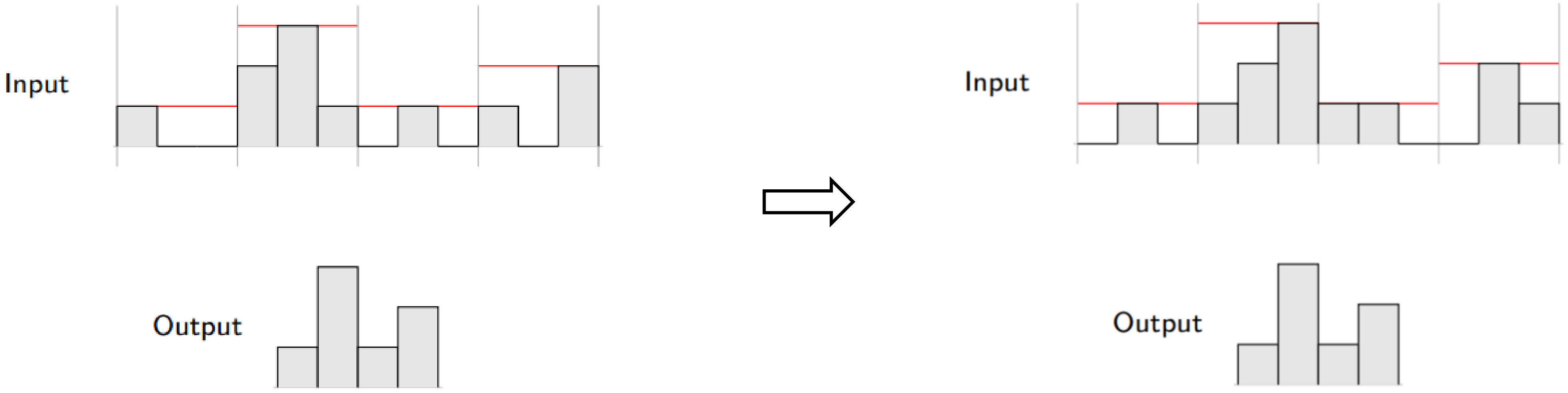
6	13
9	2.5

Pooling layer



Pooling layer: invariance

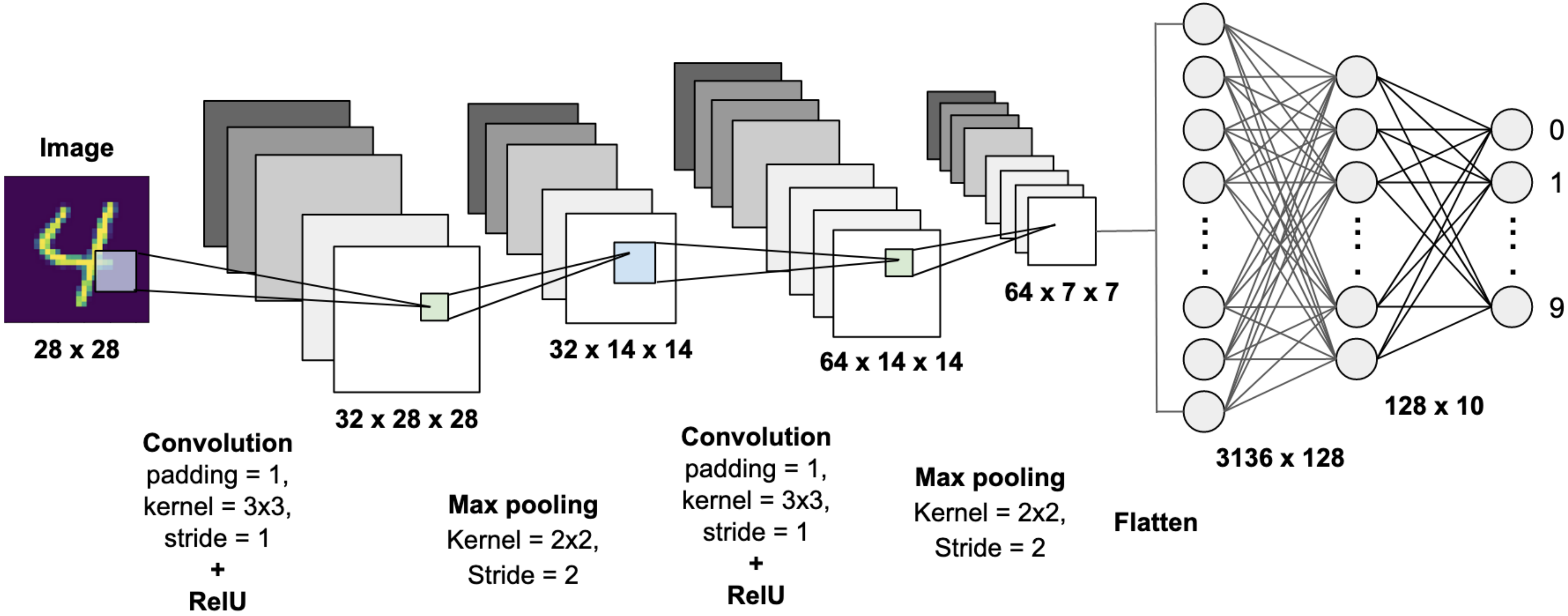
- Max pooling provides **invariance**.
  - This is helpful if we care more about presence of a pattern.
  - For detecting a face, we just need to know that an eye is present in a region not its exact location.



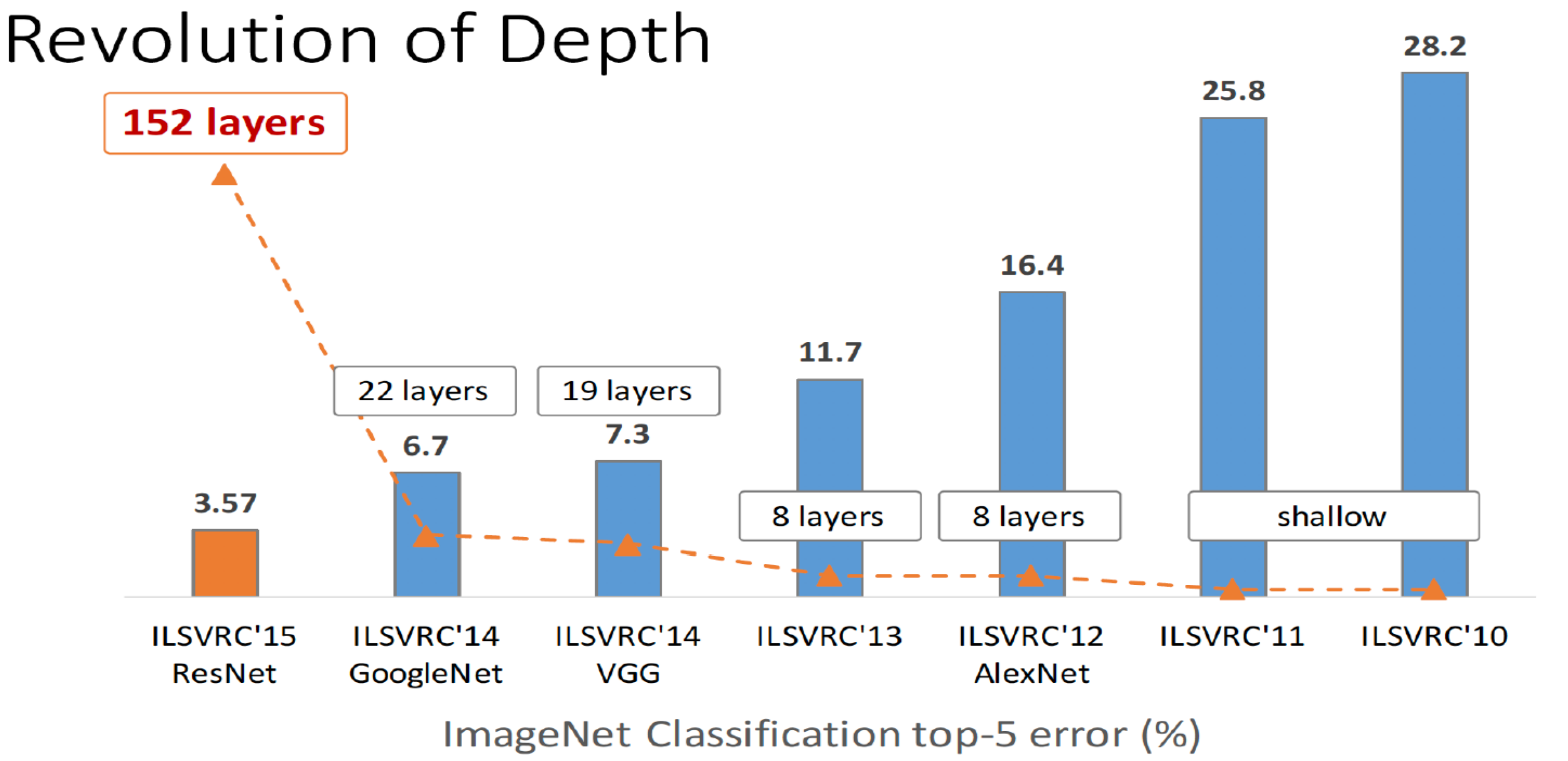
Contents

- Convolutional layer
- Pooling layer
- Fully-Connected layer
- CNN architecture

CNN architecture



Improvement in CNN



(Slides from Kaiming He's recent presentation)

Improvements in CNN

- Use smaller kernels (e.g. 5x5, 3x3 and 1x1)

- Use deeper neural network (e.g. 20, 50, 101 layers)

- Use other techniques (e.g. ReLU, Batch Normalization, Dropout)

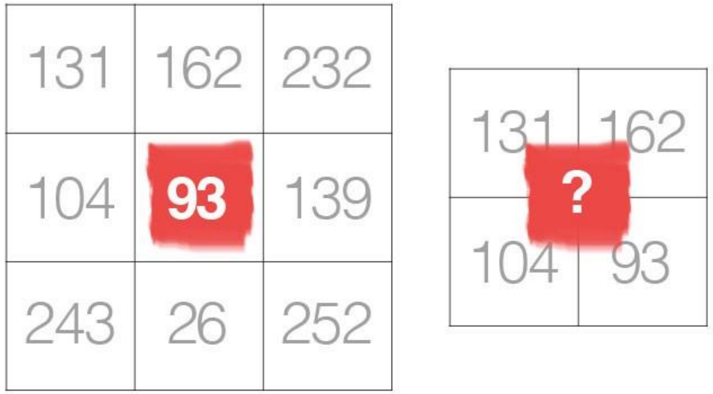
Tip

- Use filter size of **odd numbers** (ex) 1, 3, 5, 7).
  - Smaller filters is preferred than bigger one.

- Use stride = 1 or 2.

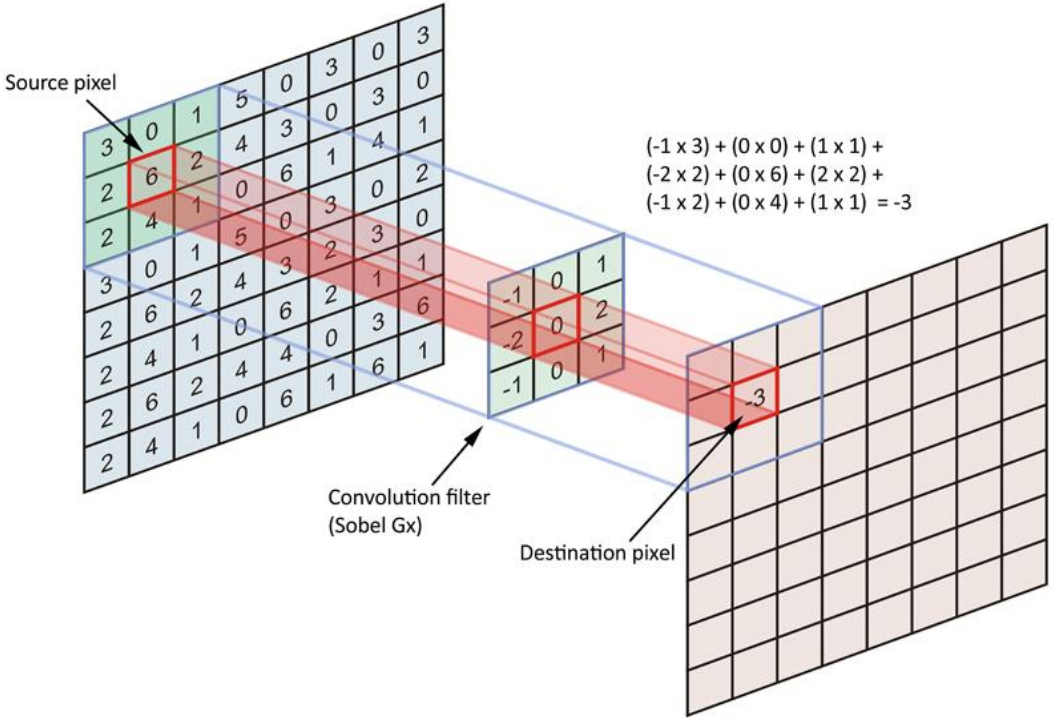
Tip

- Use filter size of **odd numbers** (ex) 1, 3, 5, 7).
  - Smaller filters are preferred than bigger ones.



131	162	232
104	93	139
243	26	252

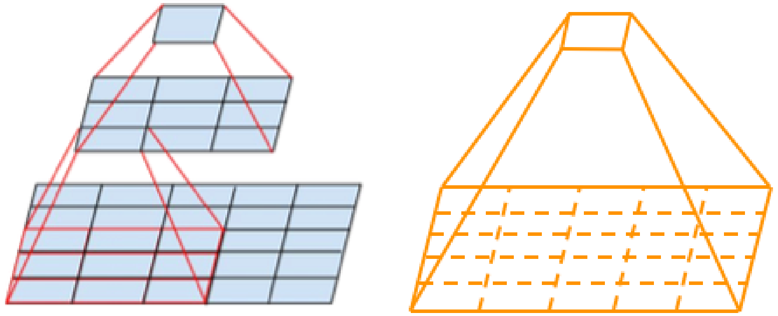
131	162
104	93



(-1 x 3) + (0 x 0) + (1 x 1) +
(-2 x 2) + (0 x 6) + (2 x 2) +
(-1 x 2) + (0 x 4) + (1 x 1) = -3

Tip

- The number of parameters of 5 x 5 filter:
  - 5 x 5 = 25
- The number of parameters of 2 3 x 3 filters:
  - 3 x 3 + 3 x 3 = 18
- As the number of parameters decreases, the depth increases.


two successive
3x3 convolutions

5x5 convolution

Tip

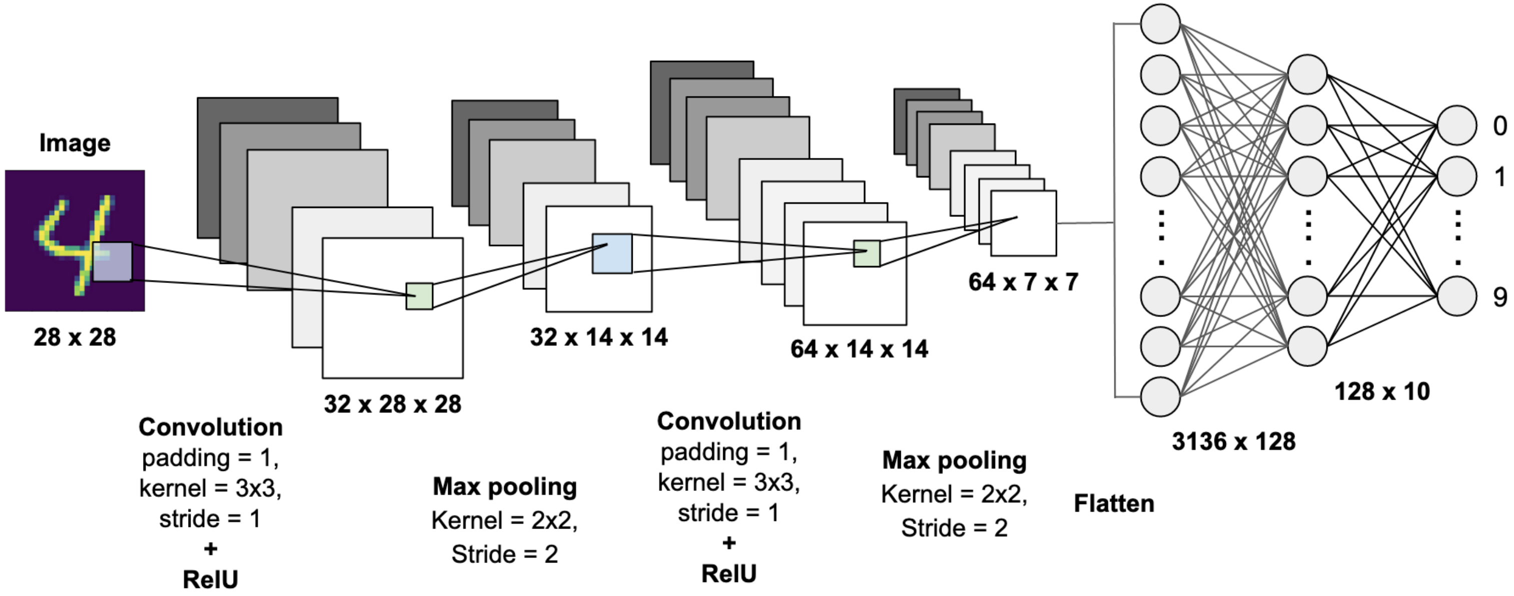
- Deeper Neural Networks are better than shallow ones.

- There is a non-linear function for each convolutional layer.
  - In deeper networks, more complex non-linear functions and features can be learned.

- Therefore, smaller filter sizes and deeper networks are used for better performance.

Tip

- Use filter size (3, 3) or (1, 1).
- Use deeper networks like 50, 100, 200, …

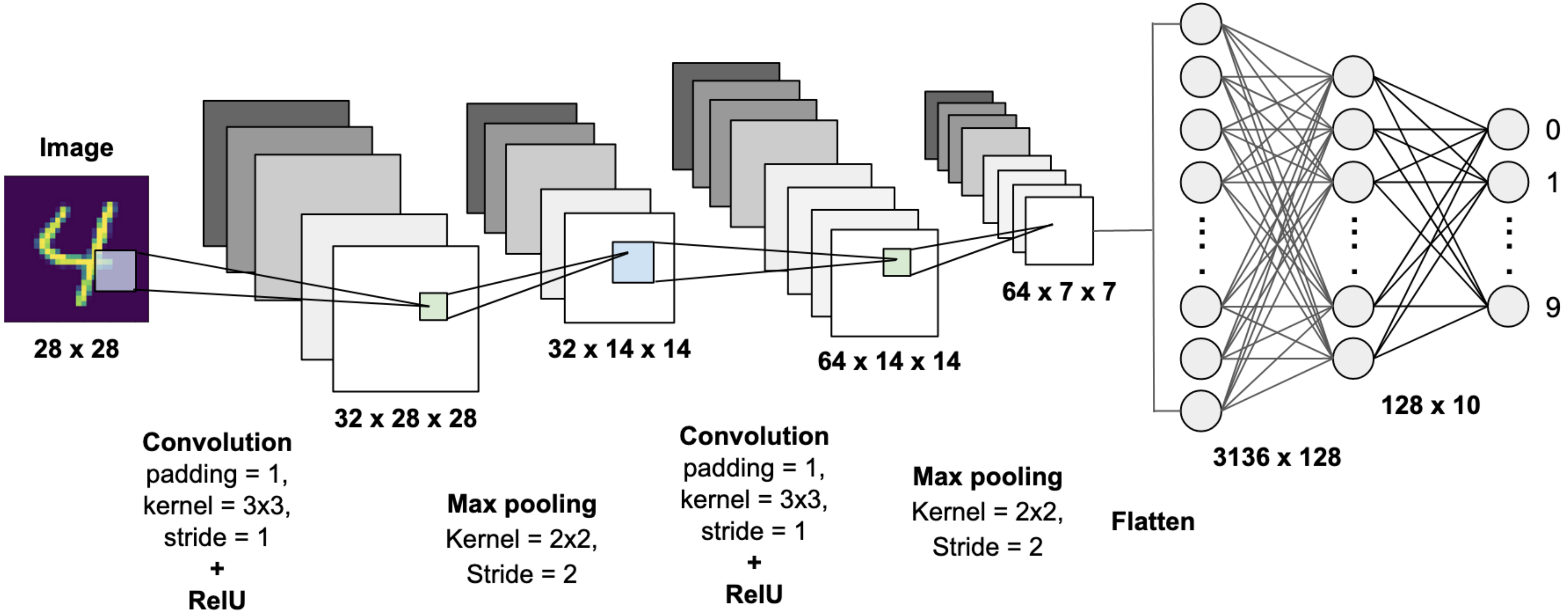


Tip

- Use convolution layers with stride = 1 and zero padding for using the same size of image in the next convolutional layer.
- Use max pooling to reduce the dimension of the feature maps.

- When we use 100 convolutional layers…

Practice



Convolution
padding = 1,
kernel = 3x3,
stride = 1
+
RelU

Max pooling
Kernel = 2x2,
Stride = 2

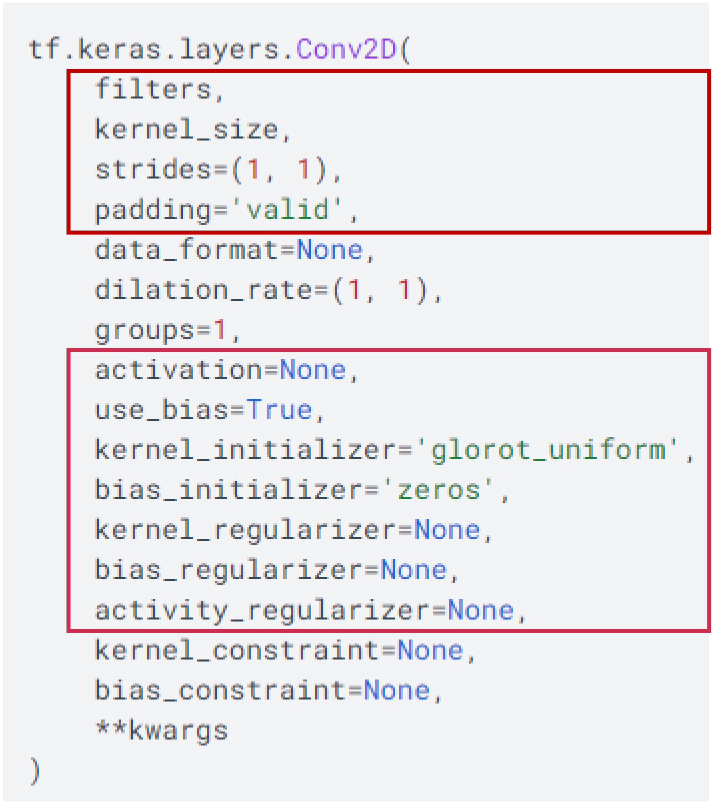
Convolution
padding = 1,
kernel = 3x3,
stride = 1
+
RelU

Max pooling
Kernel = 2x2,
Stride = 2

Flatten

tf.keras.layers.Conv2D

- tf.keras.layers.Conv2D(filters, kernel_size, stride, padding, activation,



```
tf.keras.layers.Conv2D(
    filters,
    kernel_size,
    strides=(1, 1),
    padding='valid',
    data_format=None,
    dilation_rate=(1, 1),
    groups=1,
    activation=None,
    use_bias=True,
    kernel_initializer='glorot_uniform',
    bias_initializer='zeros',
    kernel_regularizer=None,
    bias_regularizer=None,
    activity_regularizer=None,
    kernel_constraint=None,
    bias_constraint=None,
    **kwargs
)
```

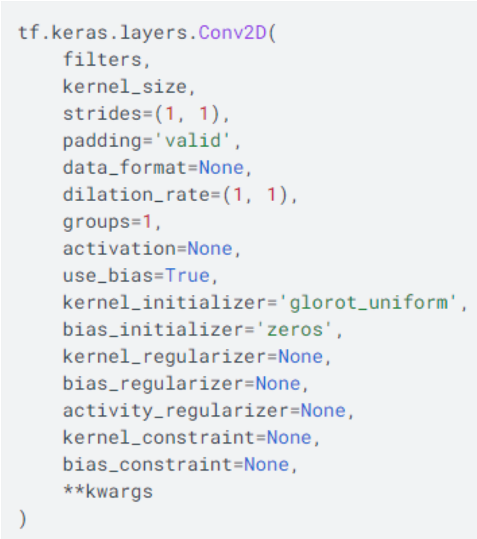
Positional arguments

= Xavier normal initializer

tf.keras.layers.Conv2D

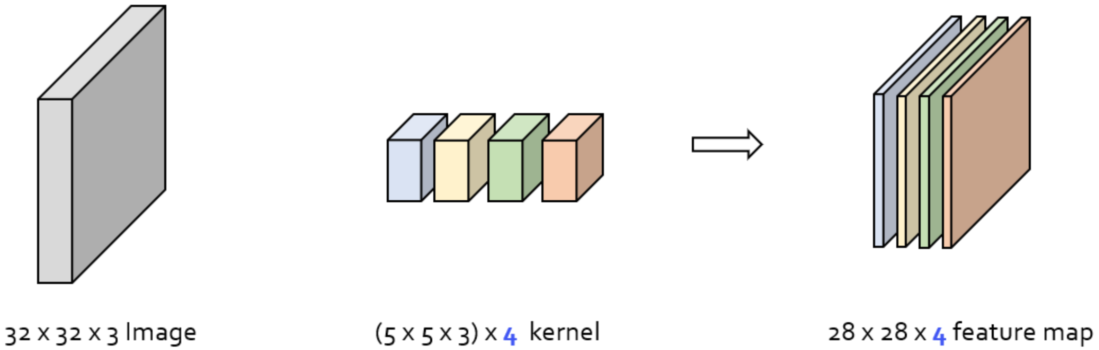
- **filters**
  - The number of filters (i.e. the dimensionality of output space)

- **kernel_size**
  - The size of kernels

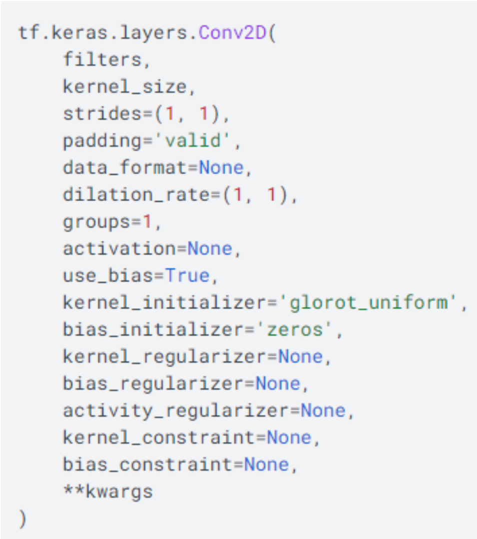


```
tf.keras.layers.Conv2D(
    filters,
    kernel_size,
    strides=(1, 1),
    padding='valid',
    data_format=None,
    dilation_rate=(1, 1),
    groups=1,
    activation=None,
    use_bias=True,
    kernel_initializer='glorot_uniform',
    bias_initializer='zeros',
    kernel_regularizer=None,
    bias_regularizer=None,
    activity_regularizer=None,
    kernel_constraint=None,
    bias_constraint=None,
    **kwargs
)
```

- layers.Conv2d(filters=4, kernel_size=(5,5))
- layers.Conv2d(filters=4, kernel_size=5)



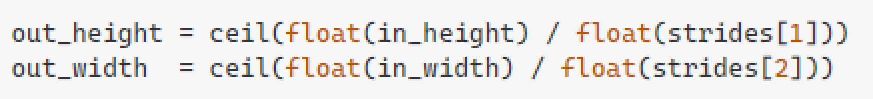
tf.keras.layers.Conv2D



```
tf.keras.layers.Conv2D(
    filters,
    kernel_size,
    strides=(1, 1),
    padding='valid',
    data_format=None,
    dilation_rate=(1, 1),
    groups=1,
    activation=None,
    use_bias=True,
    kernel_initializer='glorot_uniform',
    bias_initializer='zeros',
    kernel_regularizer=None,
    bias_regularizer=None,
    activity_regularizer=None,
    kernel_constraint=None,
    bias_constraint=None,
    **kwargs
)
```

- **strides**
  - The size of stride (height, width)

- **padding**
  - one of '**valid**' or '**same**'.
  - **'valid'** means no padding.
  - When **'same'** and **strides=1**, output size is the same as the input size (using zero padding).



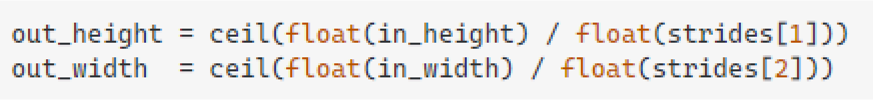
```
out_height = ceil(float(in_height) / float(strides[1]))
out_width  = ceil(float(in_width) / float(strides[2]))
```

- keras.layers.**ZeroPadding2D**(padding=(2, 2))

Output size of convolution

- Output size:
  - $W_2 = (W_1 - F + 2P)/S + 1,$ $\mathrm{H}_2 = (\mathrm{H}_1 - \mathrm{F} + 2P)/S + 1$
  - $D_2 = K$

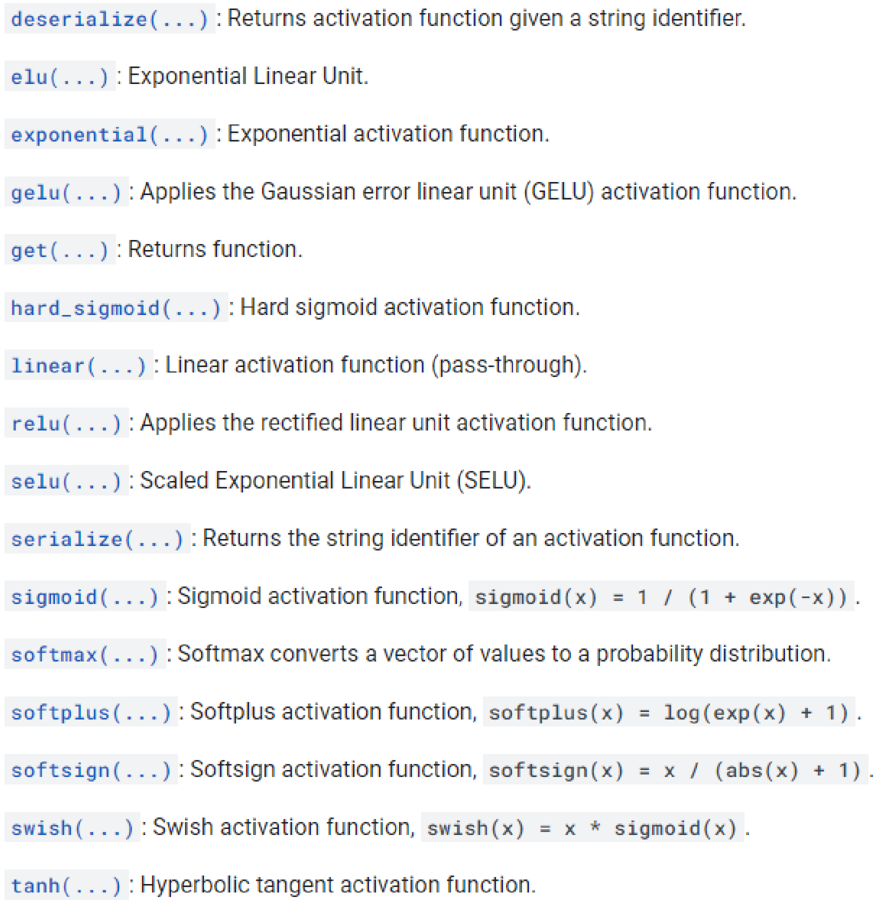
- When $W_1 = 128, F = 11, S = 6,$
- $W_2 = (128 \ - 11 + 2P)/6 + 1 = 19 + 2P + 1 = 20 + 2P$
- If 'valid', $W_2 = 20,$
- If 'same', $W_2 = \mathrm{ceil}(128 / 6) = 22,$ P = 1



```
out_height = ceil(float(in_height) / float(strides[1]))
out_width  = ceil(float(in_width) / float(strides[2]))
```

tf.keras.layers.Conv2D

- activation
  - **tf.keras.activations**
  - tf.keras.layers.Conv2D(…, **activation=tf.keras.activations.relu**)
  - tf.keras.layers.Conv2D(…, **activation=‘relu’**)



`deserialize(...)` : Returns activation function given a string identifier.

`elu(...)` : Exponential Linear Unit.

`exponential(...)` : Exponential activation function.

`gelu(...)` : Applies the Gaussian error linear unit (GELU) activation function.

`get(...)` : Returns function.

`hard_sigmoid(...)` : Hard sigmoid activation function.

`linear(...)` : Linear activation function (pass-through).

`relu(...)` : Applies the rectified linear unit activation function.

`selu(...)` : Scaled Exponential Linear Unit (SELU).

`serialize(...)` : Returns the string identifier of an activation function.

`sigmoid(...)` : Sigmoid activation function, `sigmoid(x) = 1 / (1 + exp(-x))` .

`softmax(...)` : Softmax converts a vector of values to a probability distribution.

`softplus(...)` : Softplus activation function, `softplus(x) = log(exp(x) + 1)` .

`softsign(...)` : Softsign activation function, `softsign(x) = x / (abs(x) + 1)` .

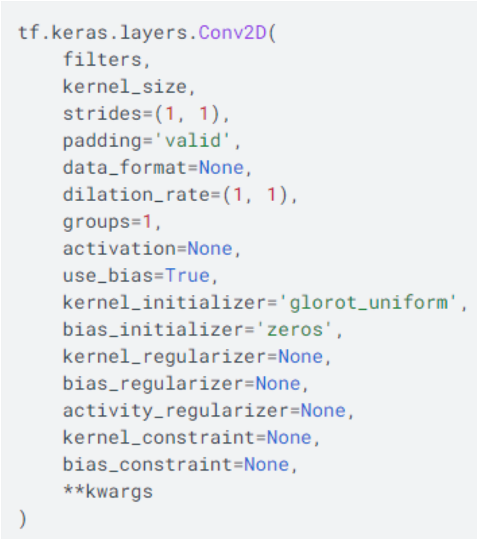
`swish(...)` : Swish activation function, `swish(x) = x * sigmoid(x)` .

`tanh(...)` : Hyperbolic tangent activation function.

https://www.tensorflow.org/api_docs/python/tf/keras/activations

tf.keras.layers.Conv2D

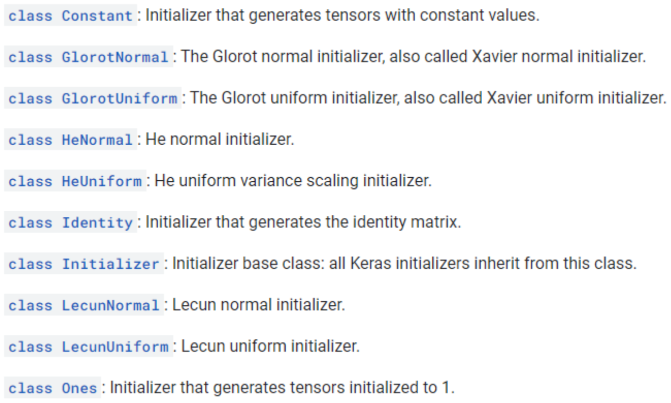
- use_bias
  - Boolean, whether the layer uses a bias vector.



```
tf.keras.layers.Conv2D(
    filters,
    kernel_size,
    strides=(1, 1),
    padding='valid',
    data_format=None,
    dilation_rate=(1, 1),
    groups=1,
    activation=None,
    use_bias=True,
    kernel_initializer='glorot_uniform',
    bias_initializer='zeros',
    kernel_regularizer=None,
    bias_regularizer=None,
    activity_regularizer=None,
    kernel_constraint=None,
    bias_constraint=None,
    **kwargs
)
```

- kernel_initializer
  - Initializer for the kernel weights matrix (default = glorot_uniform = Xavier normal)

- bias_initializer
  - Initializer for the bias vector (default = zeros)



`class Constant` : Initializer that generates tensors with constant values.

`class GlorotNormal` : The Glorot normal initializer, also called Xavier normal initializer.

`class GlorotUniform` : The Glorot uniform initializer, also called Xavier uniform initializer.

`class HeNormal` : He normal initializer.

`class HeUniform` : He uniform variance scaling initializer.

`class Identity` : Initializer that generates the identity matrix.

`class Initializer` : Initializer base class: all Keras initializers inherit from this class.

`class LecunNormal` : Lecun normal initializer.

`class LecunUniform` : Lecun uniform initializer.

`class Ones` : Initializer that generates tensors initialized to 1.

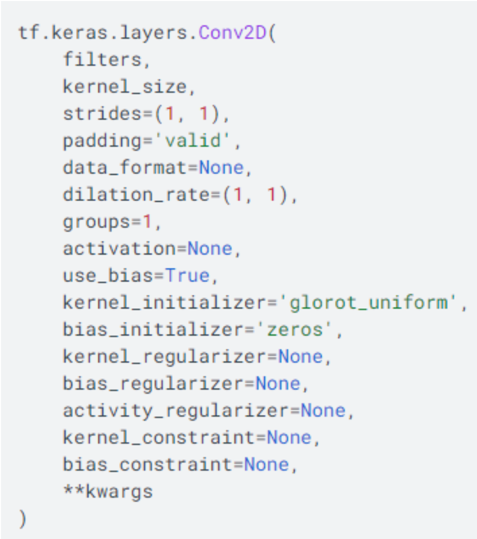
https://www.tensorflow.org/api_docs/python/tf/keras/initializers

tf.keras.layers.Conv2D

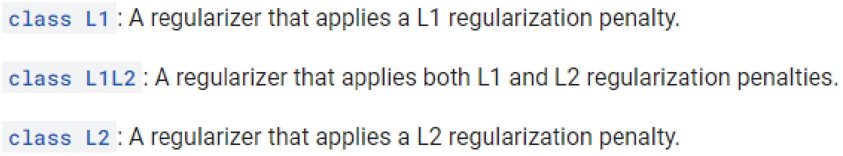
- **kernel_regularizer**
  - Regularizer function applied to the **kernel weights** matrix

- **bias_regularizer**
  - Regularizer function applied to the **bias** vector

- **Activity regularizer**
  - Regularizer function applied to **the output of the layer**



```
tf.keras.layers.Conv2D(
    filters,
    kernel_size,
    strides=(1, 1),
    padding='valid',
    data_format=None,
    dilation_rate=(1, 1),
    groups=1,
    activation=None,
    use_bias=True,
    kernel_initializer='glorot_uniform',
    bias_initializer='zeros',
    kernel_regularizer=None,
    bias_regularizer=None,
    activity_regularizer=None,
    kernel_constraint=None,
    bias_constraint=None,
    **kwargs
)
```



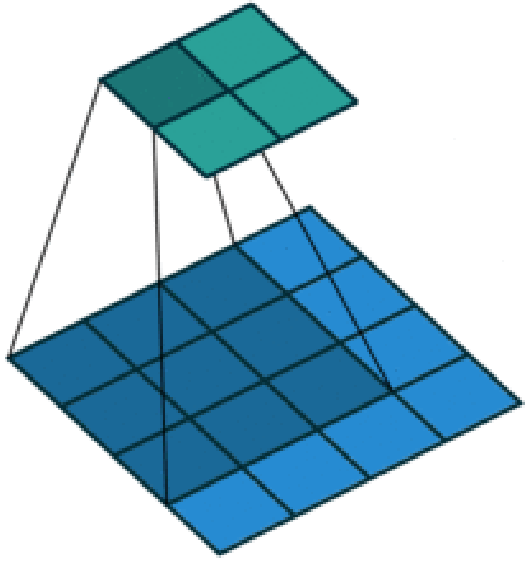
`class L1` : A regularizer that applies a L1 regularization penalty.

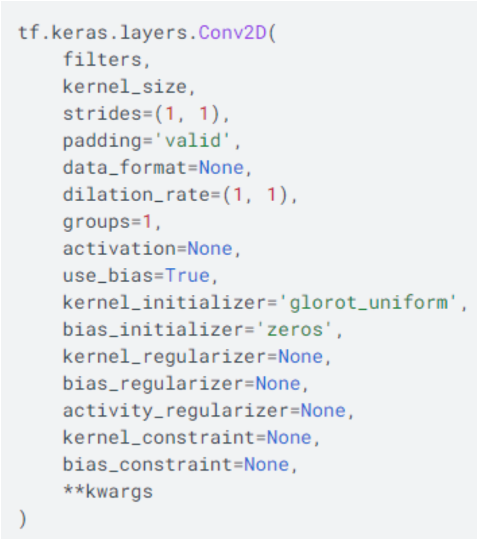
`class L1L2` : A regularizer that applies both L1 and L2 regularization penalties.

`class L2` : A regularizer that applies a L2 regularization penalty.

https://www.tensorflow.org/api_docs/python/tf/keras/regularizers

tf.keras.layers.Conv2D



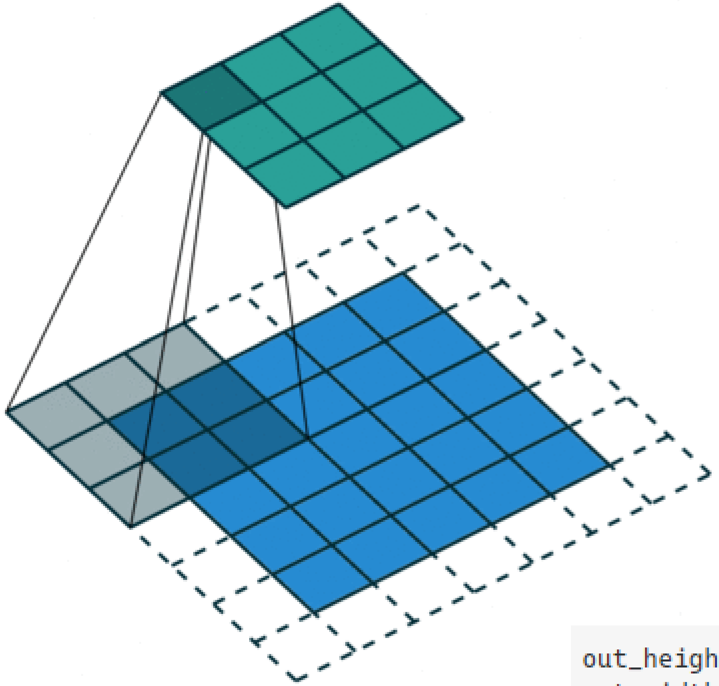


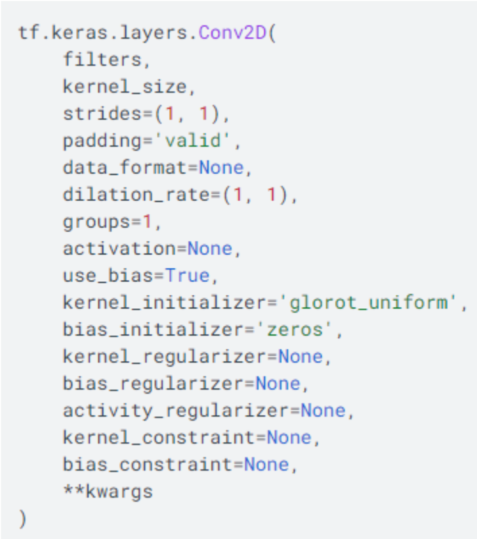
```
tf.keras.layers.Conv2D(
    filters,
    kernel_size,
    strides=(1, 1),
    padding='valid',
    data_format=None,
    dilation_rate=(1, 1),
    groups=1,
    activation=None,
    use_bias=True,
    kernel_initializer='glorot_uniform',
    bias_initializer='zeros',
    kernel_regularizer=None,
    bias_regularizer=None,
    activity_regularizer=None,
    kernel_constraint=None,
    bias_constraint=None,
    **kwargs
)
```

- layers.Conv2D(filters=1, kernel_size=3 *(3,3)*, padding=‘valid’, stride=1 *(1,1)*,

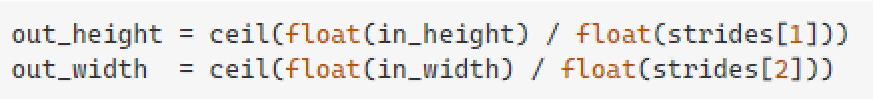
  activation=‘relu’, use_bias=False, kernel_initializer=‘HeNormal’)

tf.keras.layers.Conv2D





```
tf.keras.layers.Conv2D(
    filters,
    kernel_size,
    strides=(1, 1),
    padding='valid',
    data_format=None,
    dilation_rate=(1, 1),
    groups=1,
    activation=None,
    use_bias=True,
    kernel_initializer='glorot_uniform',
    bias_initializer='zeros',
    kernel_regularizer=None,
    bias_regularizer=None,
    activity_regularizer=None,
    kernel_constraint=None,
    bias_constraint=None,
    **kwargs
)
```



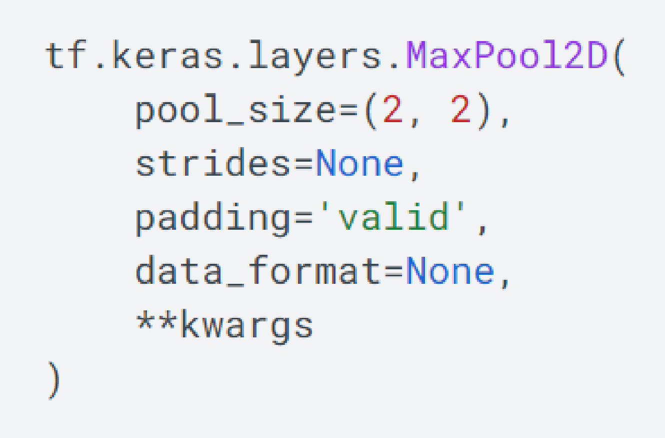
```
out_height = ceil(float(in_height) / float(strides[1]))
out_width  = ceil(float(in_width) / float(strides[2]))
```

- layers.Conv2D(filters=1, kernel_size=3 *(3,3)*, padding=‘same’, strides=2 *(2,2)*,

  activation=‘sigmoid’, use_bias=False, bias_regularizer=‘L1’)

tf.keras.layers.MaxPool2D

- Max pooling operation for 2D spatial data.
  - pool_size
  - strides
  - padding
    - **valid**: output_shape = math.floor((input_shape - pool_size) / strides) + 1
    - **same**: output_shape = math.floor((input_shape - 1) / strides) + 1
  - data_format



```
tf.keras.layers.MaxPool2D(
    pool_size=(2, 2),
    strides=None,
    padding='valid',
    data_format=None,
    **kwargs
)
```

$$W_2 = (W_1 - F + 2P)/S + 1$$

When ‘**valid**’,

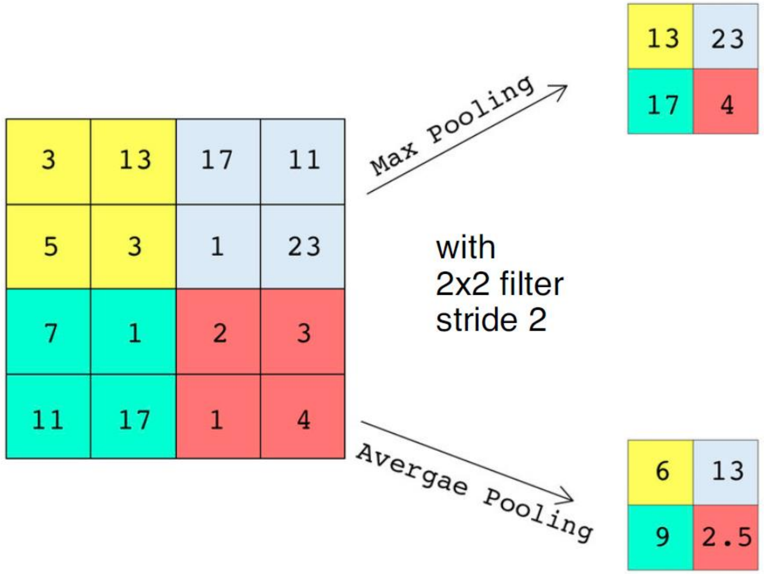
$$2P = 0$$

When ‘**same**’,

$$F - 2P = 1$$

Then, when S = 1, $W_1 = W_2$

tf.keras.layers.MaxPool2D



3	13	17	11
5	3	1	23
7	1	2	3
11	17	1	4

13	23
17	4

6	13
9	2.5

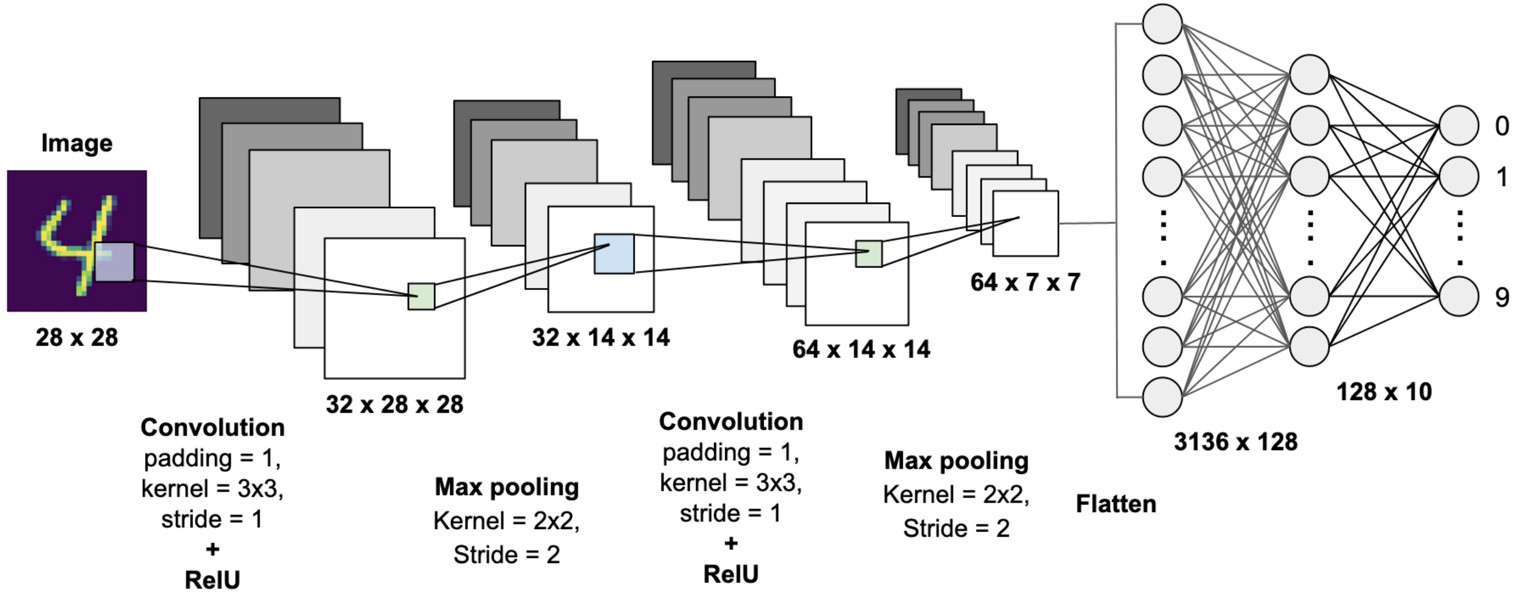
layers.MaxPool2D(pool_size=2, stride=2)

Data format

- data_format
  - **‘channels_last’**
    - (batch_size, rows, cols, **channels**)
  - **‘channels_first’**
    - (batch_size, **channels**, rows, cols)

- Default value is ‘channels_last’.

Practice



Image

28 x 28

Convolution
padding = 1,
kernel = 3x3,
stride = 1
+
RelU

32 x 28 x 28

Max pooling
Kernel = 2x2,
Stride = 2

32 x 14 x 14

Convolution
padding = 1,
kernel = 3x3,
stride = 1
+
RelU

64 x 14 x 14

Max pooling
Kernel = 2x2,
Stride = 2

64 x 7 x 7

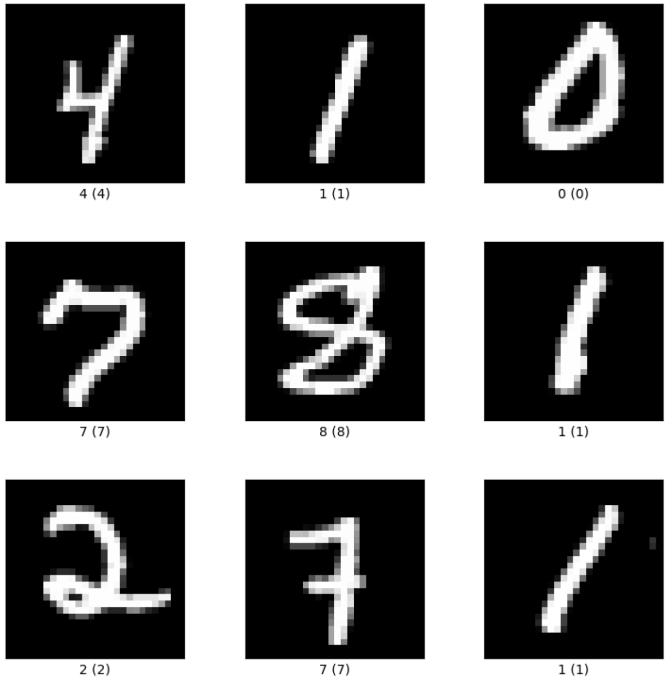
Flatten

3136 x 128

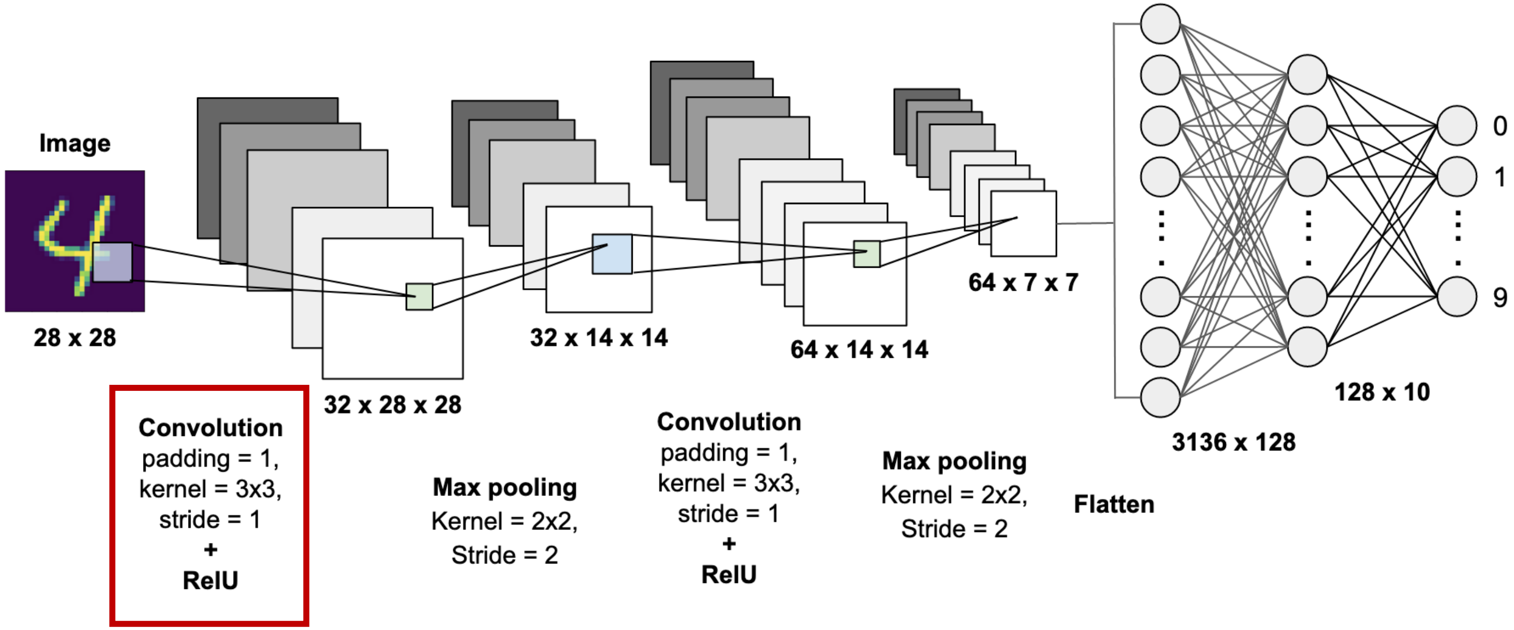
128 x 10

Practice: MNIST dataset

- Handwritten digit dataset
- 60000 training set, 10000 test set
- 10 class



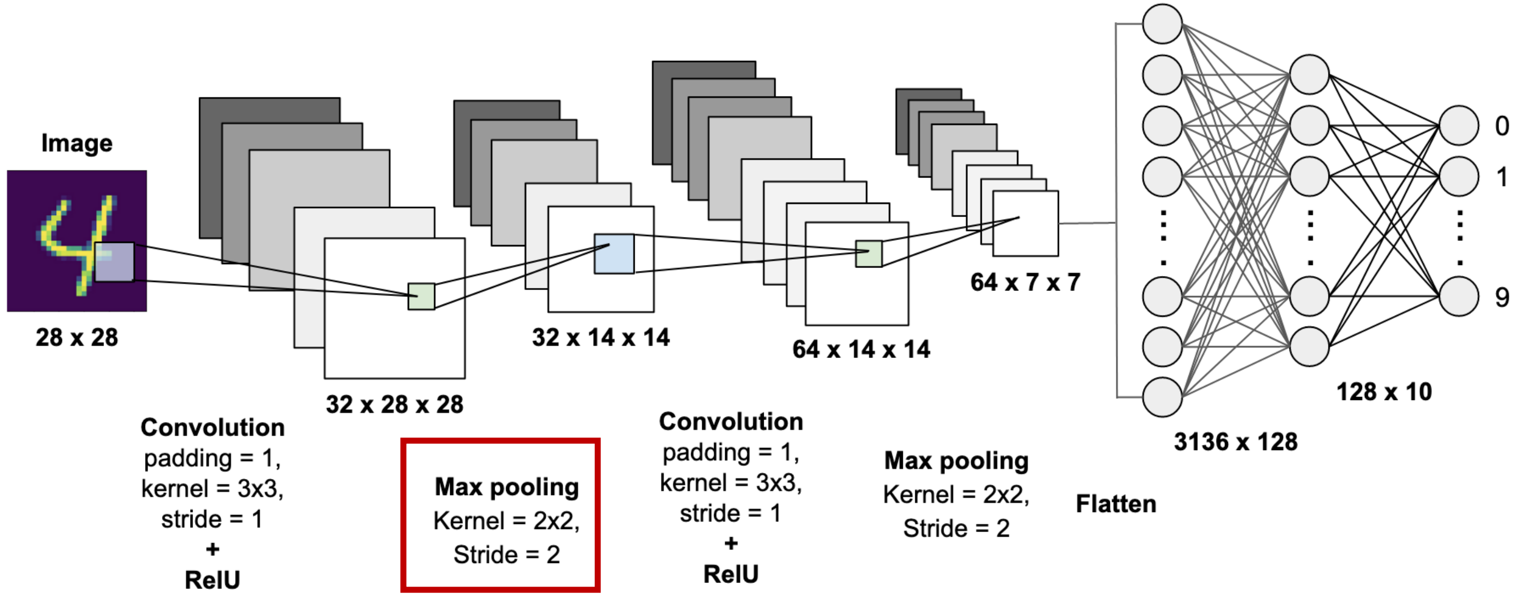
Practice



layers.Conv2D(filters, kernel_size, padding, strides, activation)

layers.Conv2D(filters=**32**, kernel_size=**3**, padding=**'same'**, strides=**1**, activation='**relu**')

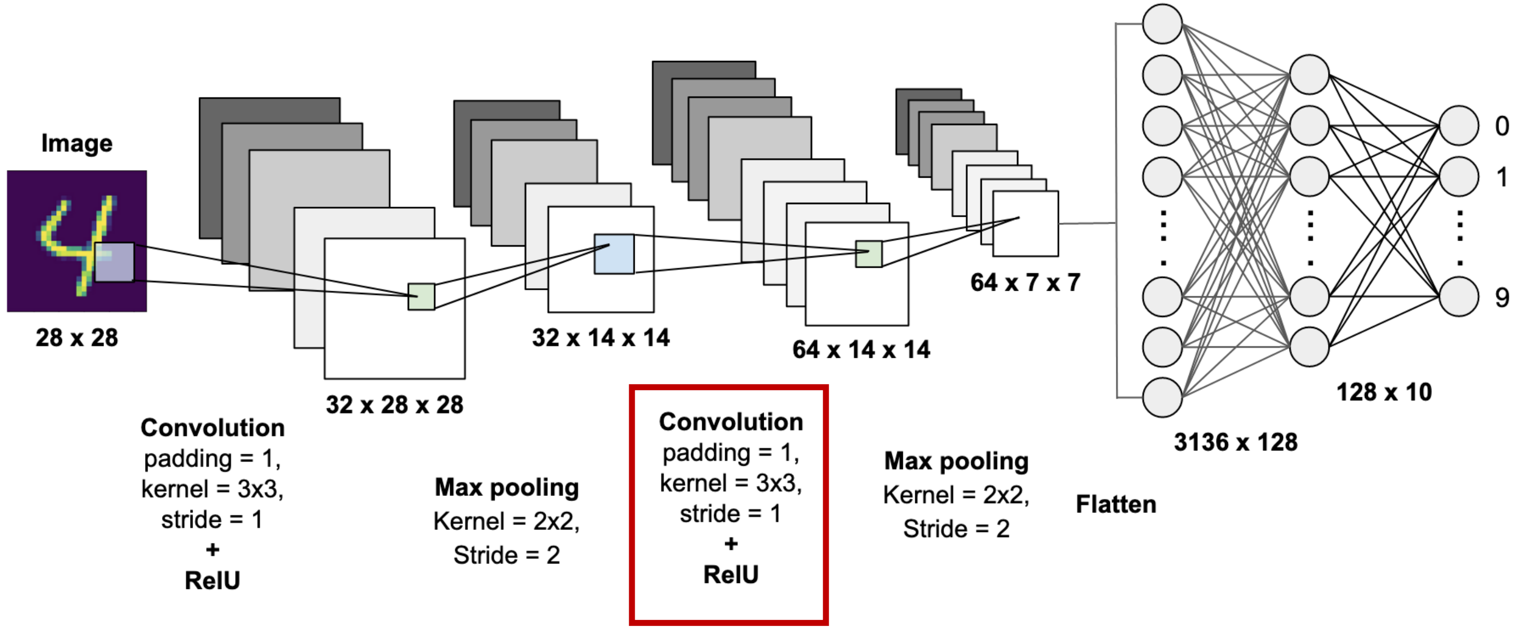
Practice



```
layers.MaxPool2D(pool_size, stride, padding)
```

```
layers.MaxPool2D(pool_size=2, stride=2, padding='same')
```

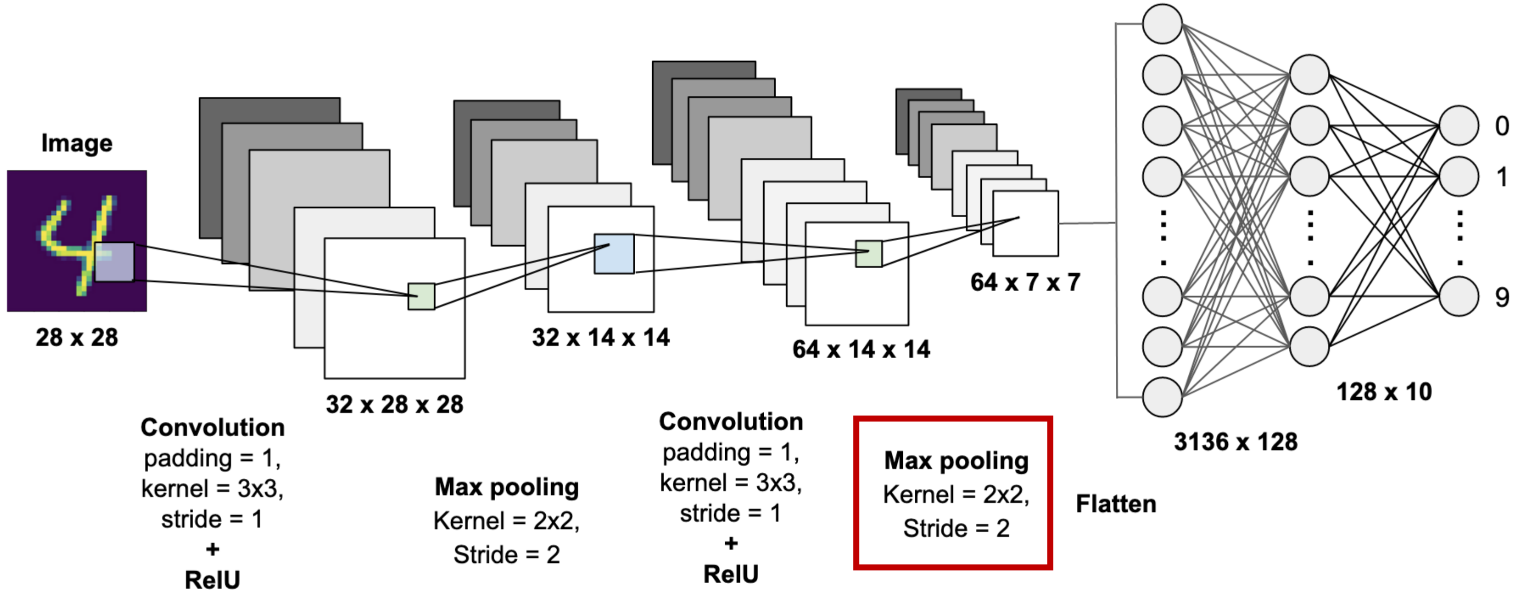
Practice



layers.Conv2D(filters, kernel_size, padding, strides, activation)

layers.Conv2D(filters=64, kernel_size=3, padding='same', strides=1, activation='relu')

Practice



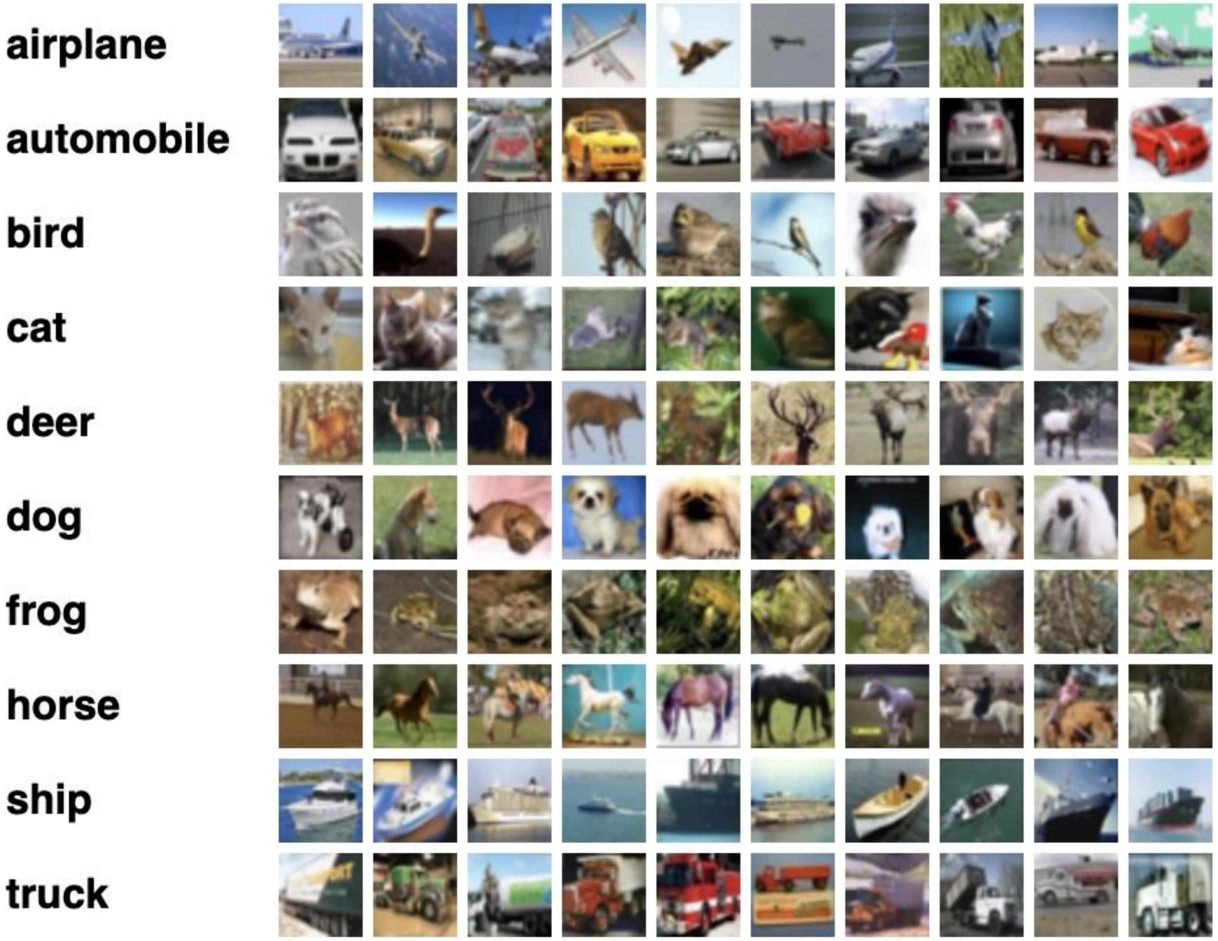
```
layers.MaxPool2D(pool_size, stride, padding)
```

```
layers.MaxPool2D(pool_size=2, stride=2, padding='same')
```

Practice: CIFAR10 dataset

- 32 x 32 x 3 Images
- 60000 colour images in 10 classes with 6000 images per class.
- 50000 training images and 10000 test images.

- It is one of the most widely used to train machine learning.



Practice: CIFAR10 dataset

- Let’s train the network with CIFAR10.
  - See the performance of the network.

- Let’s upgrade the network!

- How can we make models easier?

- Padding 부분 한번 더 확인