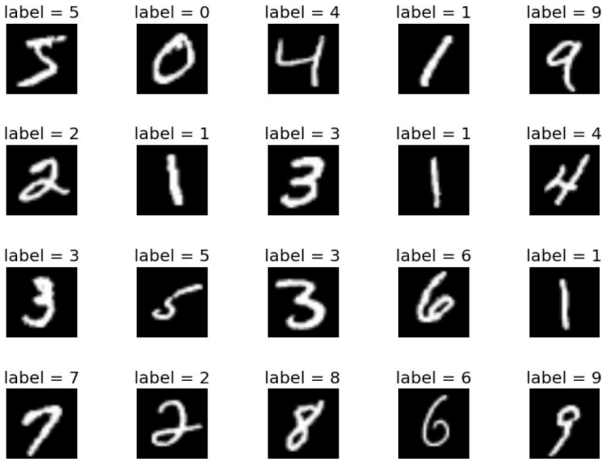
APS360: Applied Fundamentals of Deep Learning

Week 4: Convolutional Neural Network - Part I

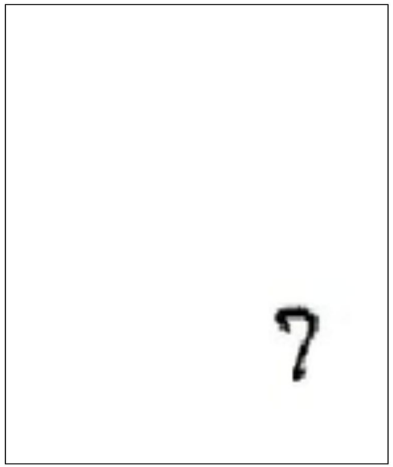
Last Time: MNIST with ANN

MNIST dataset is nicely preprocessed, centred, etc.



MNIST Data (Cleaned)

What happens if the new sample is not preprocessed?



New Data Sample (Raw)

What if we make our deep NN bigger?

Input: 200×200 pixels → 1st hidden layer: 500 → 2nd hidden layer: 200

Q: How many weights are there?

What if we make our deep NN bigger?

Input: 200×200 pixels → 1st hidden layer: 500 → 2nd hidden layer: 200

Q: How many weights are there?

Using a large fully connected network has some down sides

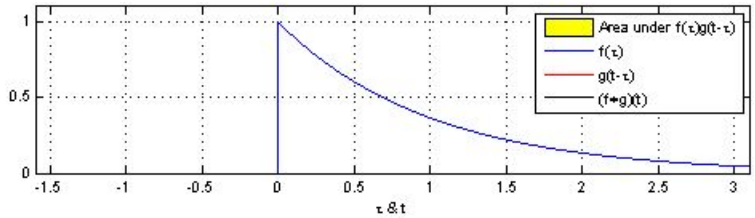
- **Computational complexity grows** → harder to train, **more data** to **generalize**
- **Bad inductive bias** → Ignores geometry of image data
- **Not flexible** → Different image sizes require different models

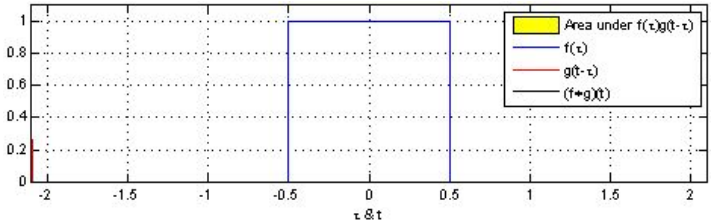
Convolution Operator

Convolution operator in One-Dimension

Convolution is a mathematical operation on two functions f and g that expresses how the shape of one is modified by the other.

$$(f * g)[n] = \sum_{k=-\infty}^{\infty} f[k]\, g[n-k]$$





Convolution in 2D for images

Convolution of Image **I** with filter kernel **K**

1. Multiply each pixel in I in range of kernel by the corresponding element of kernel K
2. Sum all these products and write to a new 2d array
3. Slide kernel across all areas of the image until you reach the ends.

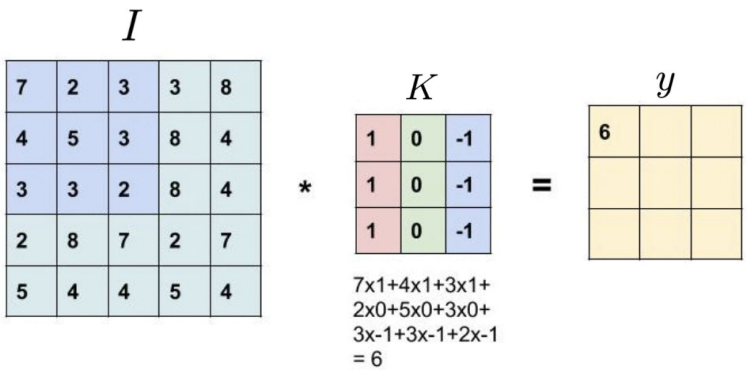
$$y[m,n] = I[m,n] * K[m,n] = \sum_{j=-\infty}^{\infty} \sum_{i=-\infty}^{\infty} I[i,j].K[m-i,n-j]$$

Convolution in 2D for images

Convolution of Image **I** with filter kernel **K**

1. Multiply each pixel in I in range of kernel by the corresponding element of kernel K
2. Sum all these products and write to a new 2d array
3. Slide kernel across all areas of the image until you reach the ends.

$$y[m,n] = I[m,n] * K[m,n] = \sum_{j=-\infty}^{\infty} \sum_{i=-\infty}^{\infty} I[i,j].K[m-i,n-j]$$

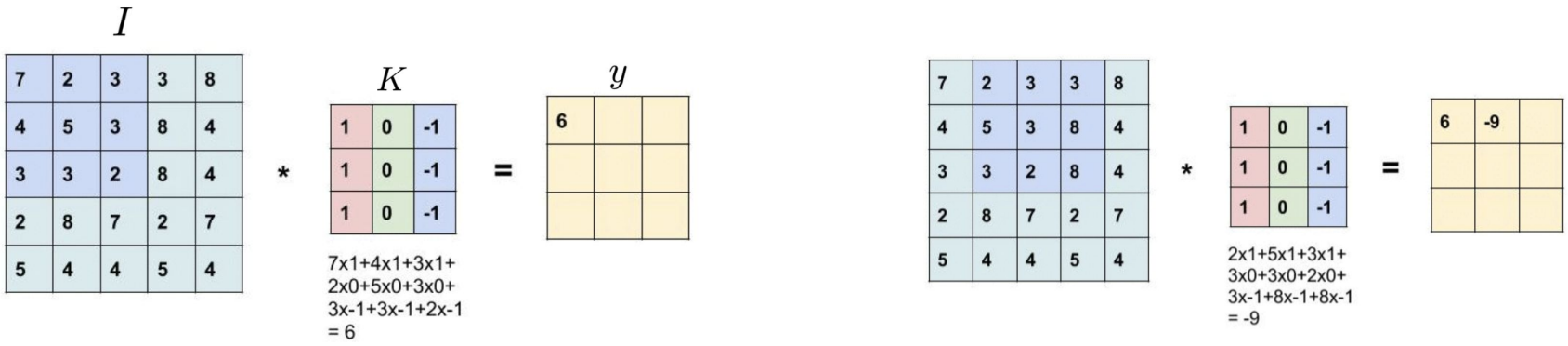


Convolution in 2D for images

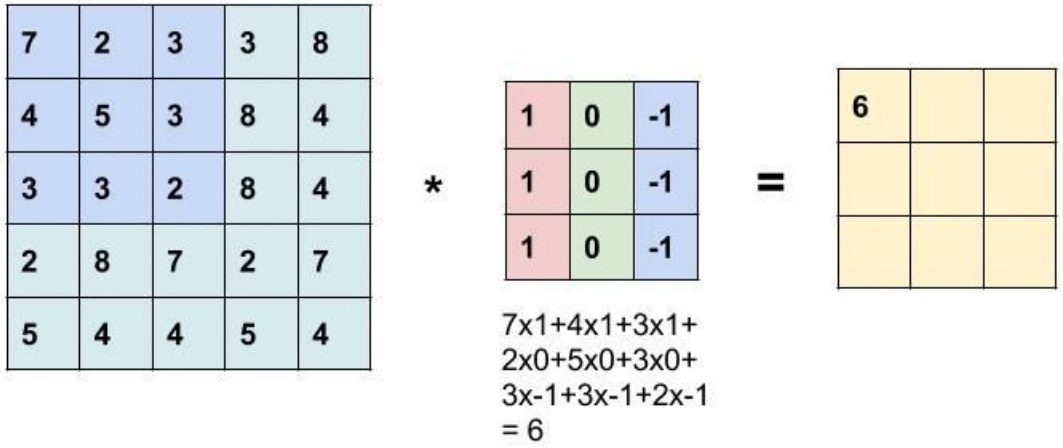
Convolution of Image **I** with filter kernel **K**

1. Multiply each pixel in I in range of kernel by the corresponding element of kernel K
2. Sum all these products and write to a new 2d array
3. Slide kernel across all areas of the image until you reach the ends.

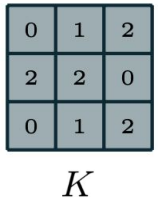
$$y[m,n] = I[m,n] * K[m,n] = \sum_{j=-\infty}^{\infty} \sum_{i=-\infty}^{\infty} I[i,j].K[m-i,n-j]$$

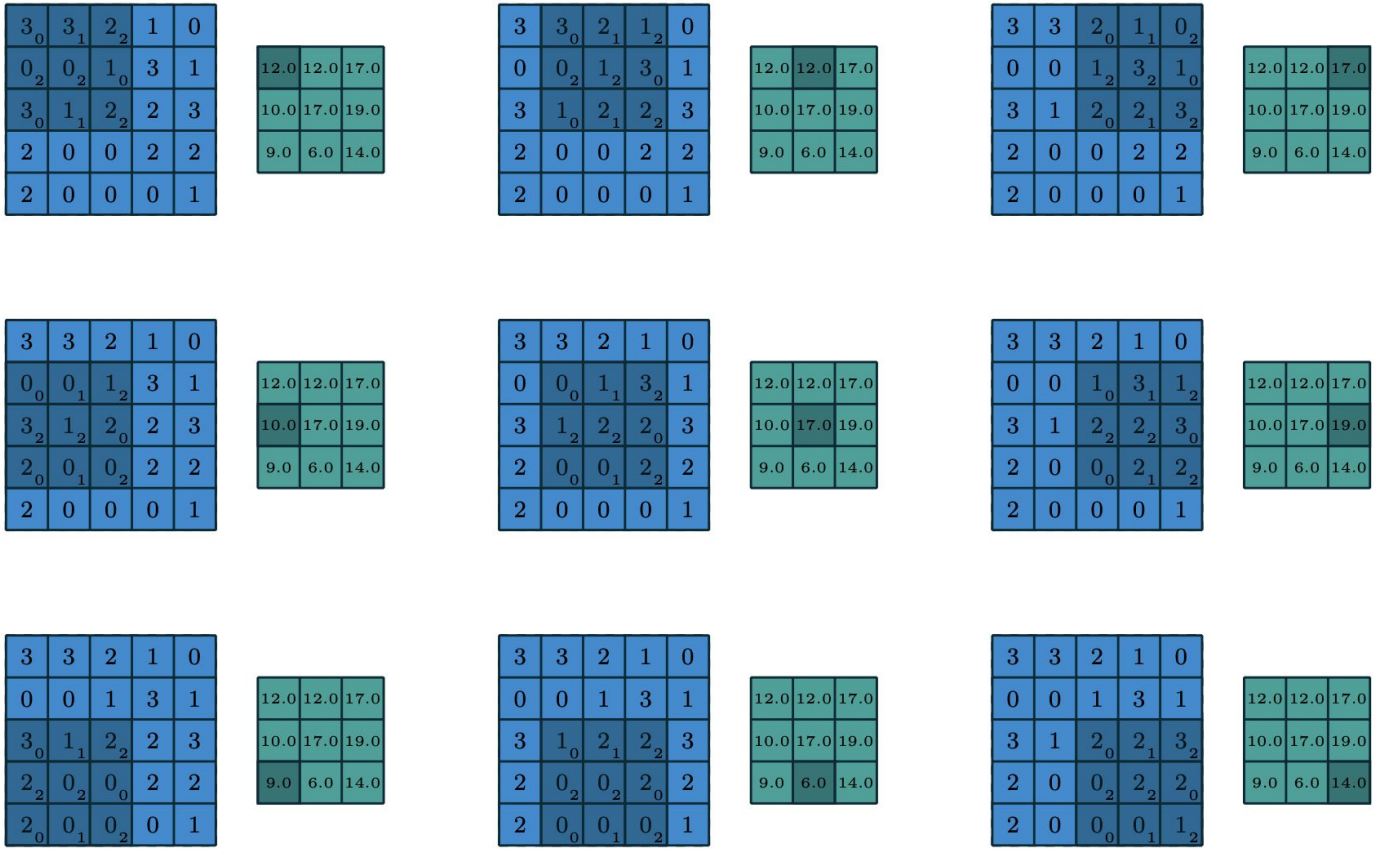


Convolution in 2D for images



Convolution in 2D for images





Convolution in 2D for images

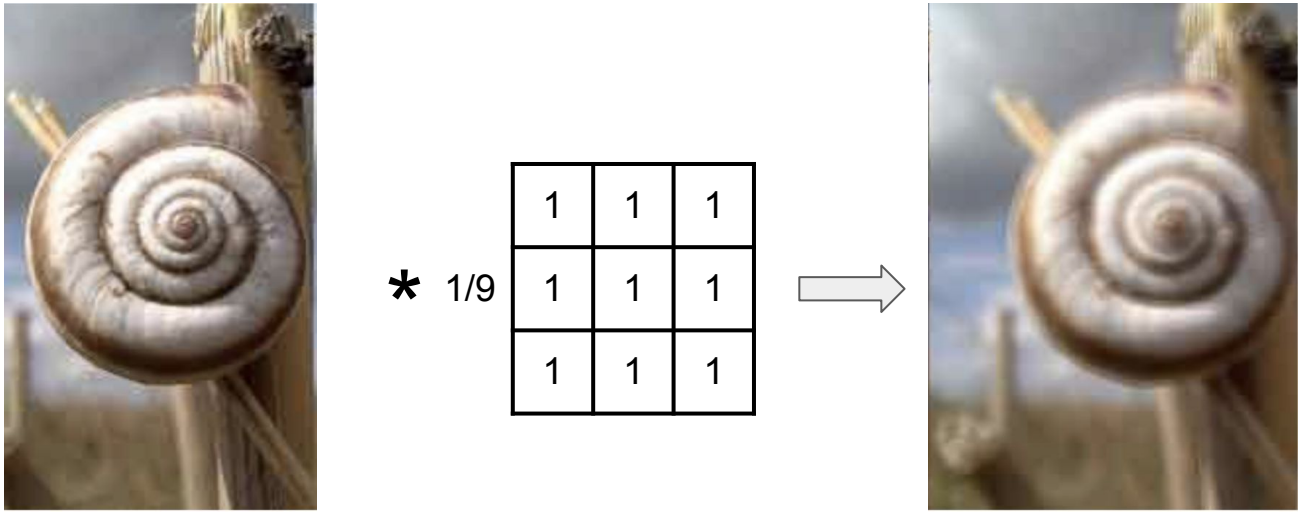
What effect would the following filter have on an image?

$$* \; \frac{1}{9} \begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix}$$

Convolution in 2D for images

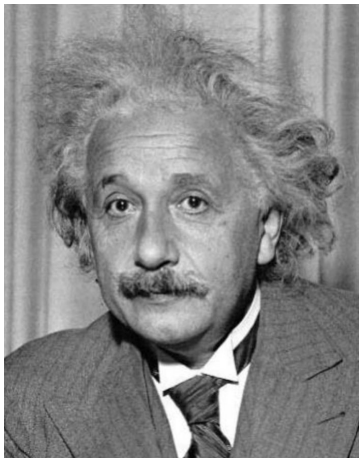
What effect would the following filter have on an image?

Blurring averages out pixel intensities in an image.



Convolution in 2D for images

What effect would the following filter have on an image?



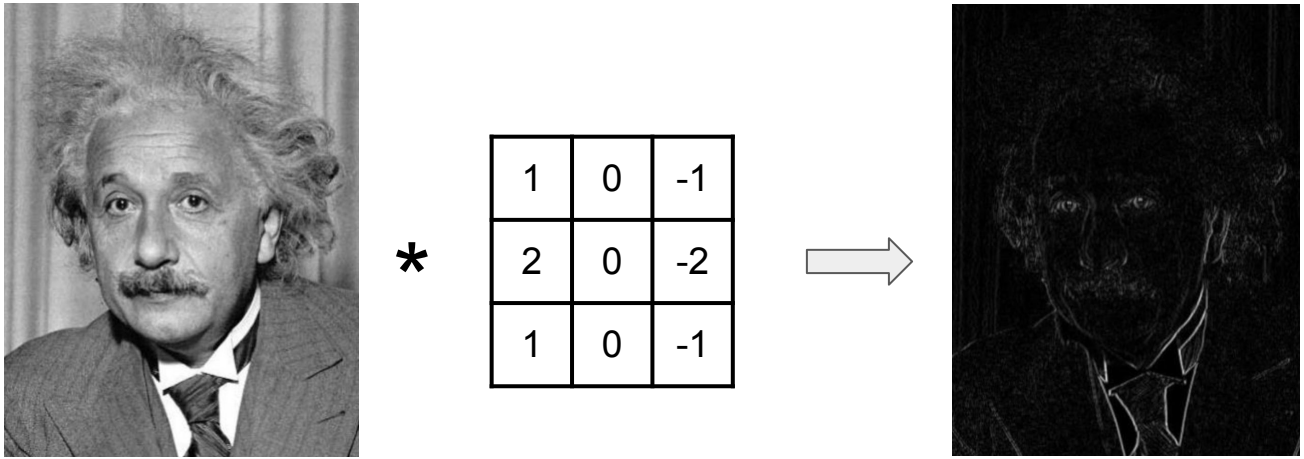
*

1	0	-1
2	0	-2
1	0	-1

Convolution in 2D for images

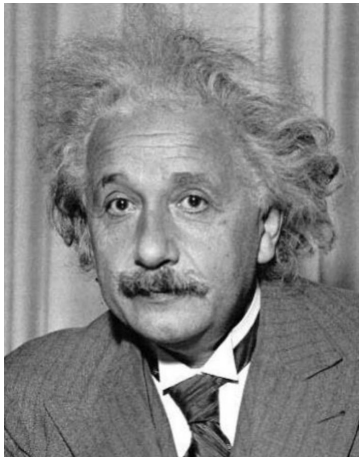
What effect would the following filter have on an image?

Vertical edge detector



Convolution in 2D for images

What effect would the following filter have on an image?



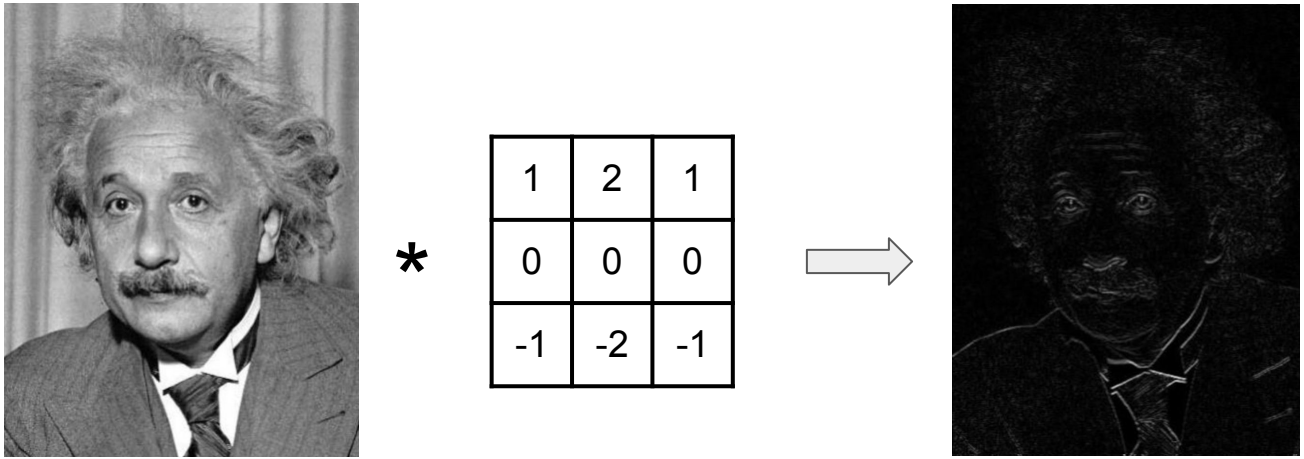
$$*$$

1	2	1
0	0	0
-1	-2	-1

Convolution in 2D for images

What effect would the following filter have on an image?

Horizontal edge detector



Convolution in 2D for images

What effect would the following filter have on an image?

$$*\begin{pmatrix} 0 & 0 & 3 & 2 & 2 & 2 & 3 & 0 & 0 \\ 0 & 2 & 3 & 5 & 5 & 5 & 3 & 2 & 0 \\ 3 & 3 & 5 & 3 & 0 & 3 & 5 & 3 & 3 \\ 2 & 5 & 3 & -12 & -23 & -12 & 3 & 5 & 2 \\ 2 & 5 & 0 & -23 & -40 & -23 & 0 & 5 & 2 \\ 2 & 5 & 3 & -12 & -23 & -12 & 3 & 5 & 2 \\ 3 & 3 & 5 & 3 & 0 & 3 & 5 & 3 & 3 \\ 0 & 2 & 3 & 5 & 5 & 5 & 3 & 2 & 0 \\ 0 & 0 & 3 & 2 & 2 & 2 & 3 & 0 & 0 \end{pmatrix}$$

Convolution in 2D for images

What effect would the following filter have on an image?

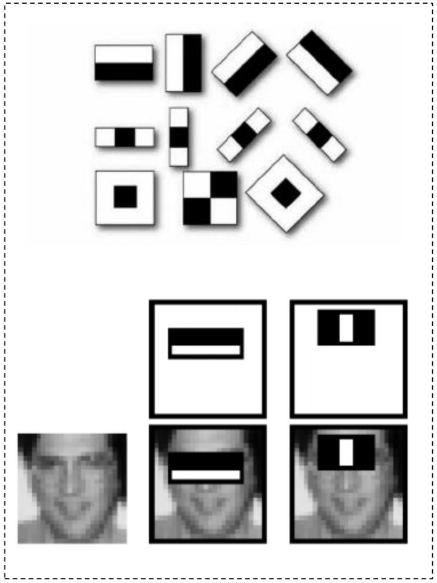
Blob detector → Regions that differ in properties, such as brightness or color, compared to surrounding regions

*

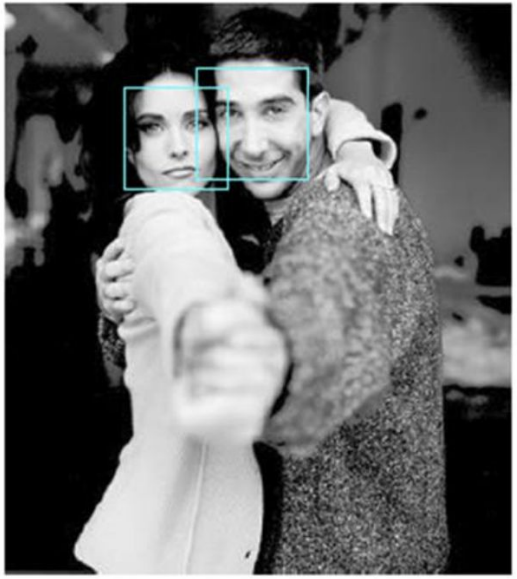
$$\begin{pmatrix} 0 & 0 & 3 & 2 & 2 & 2 & 3 & 0 & 0 \\ 0 & 2 & 3 & 5 & 5 & 5 & 3 & 2 & 0 \\ 3 & 3 & 5 & 3 & 0 & 3 & 5 & 3 & 3 \\ 2 & 5 & 3 & -12 & -23 & -12 & 3 & 5 & 2 \\ 2 & 5 & 0 & -23 & -40 & -23 & 0 & 5 & 2 \\ 2 & 5 & 3 & -12 & -23 & -12 & 3 & 5 & 2 \\ 3 & 3 & 5 & 3 & 0 & 3 & 5 & 3 & 3 \\ 0 & 2 & 3 & 5 & 5 & 5 & 3 & 2 & 0 \\ 0 & 0 & 3 & 2 & 2 & 2 & 3 & 0 & 0 \end{pmatrix}$$

Convolution in 2D for images

Even more complicated kernels for detecting faces



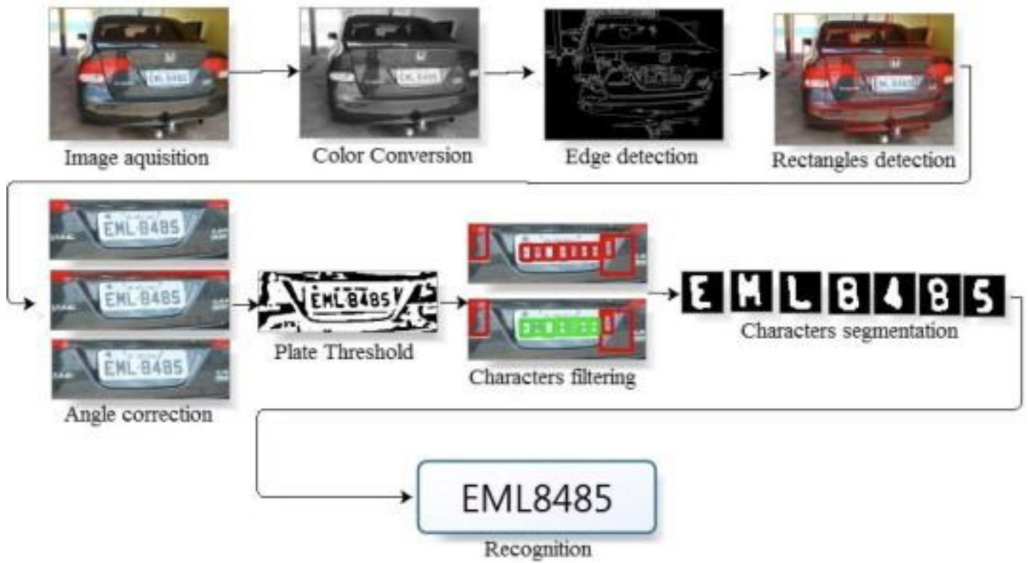
Convolutional kernels



Convolution in 2D for images

Where the kernels come from? They used to be **hand-crafted**

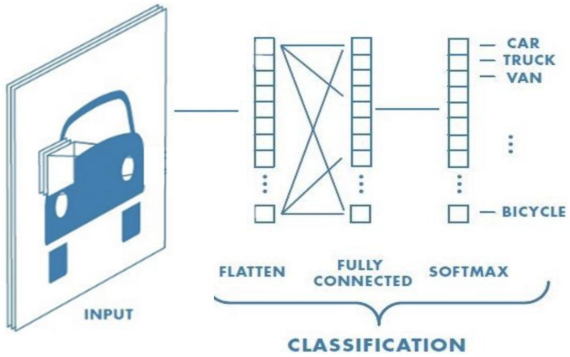
Classic computer vision → **multi-stage feature (kernel) engineering**



Convolutional Neural Networks

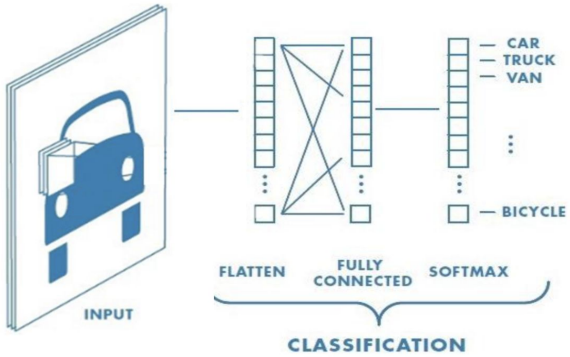
Convolutional Neural Networks

MLP → Requires that we preprocess the input!

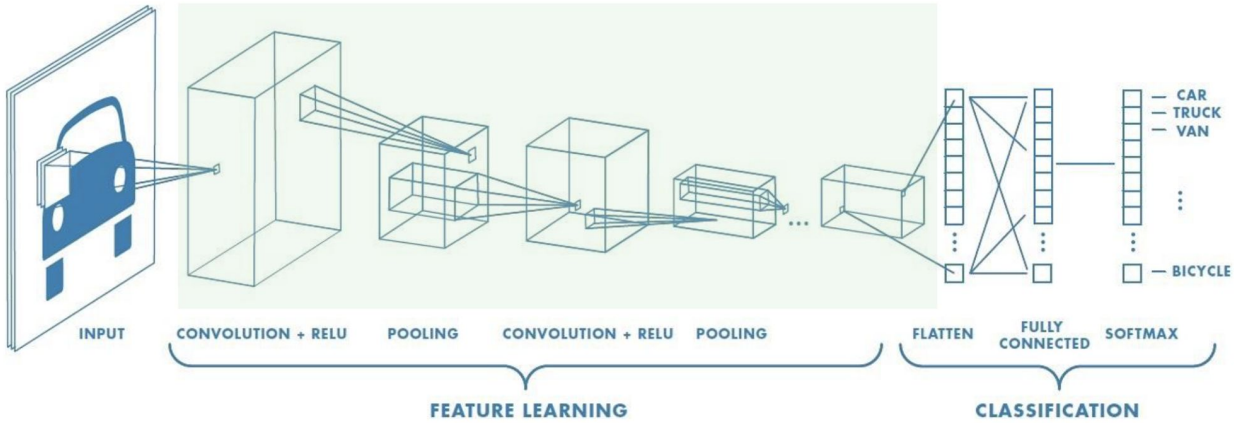


Convolutional Neural Networks

MLP → Requires that we preprocess the input!

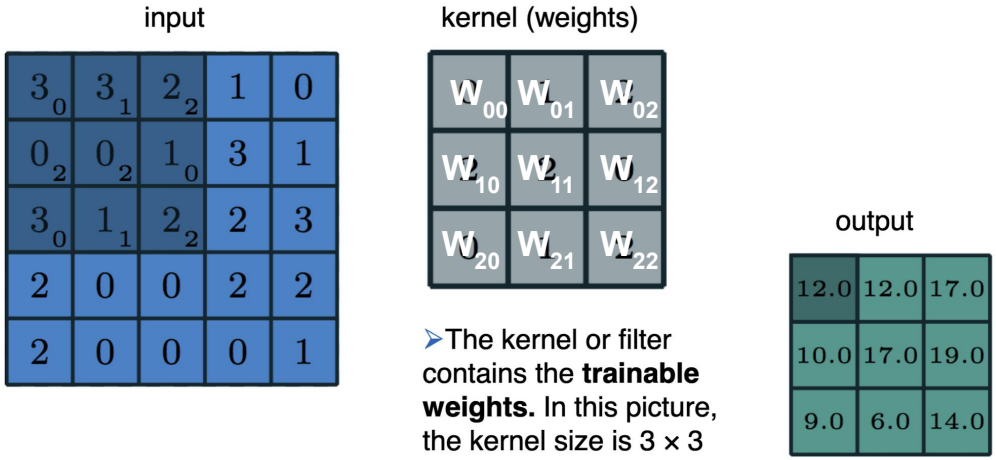


CNN → Apply convolution to image tensors.



Forward & Backward pass

- Initialize the kernels randomly
- In forward pass, convolve the image with the kernel
- In backward pass, update the kernel using gradients

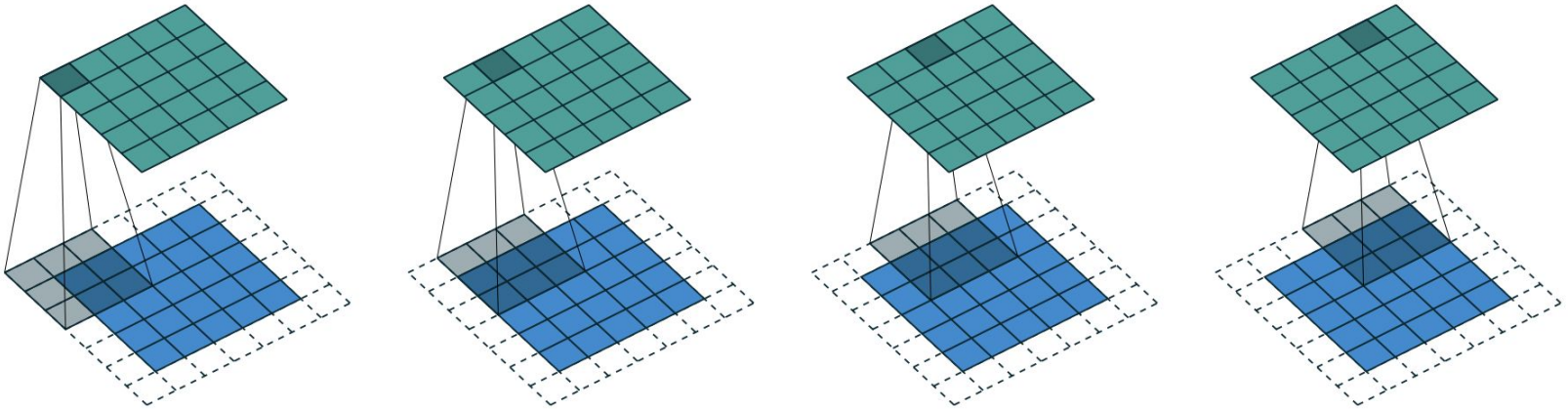


➢The kernel or filter contains the **trainable weights.** In this picture, the kernel size is 3 × 3

Zero Padding

Adding zeros around the border of the image before convolution. Why?

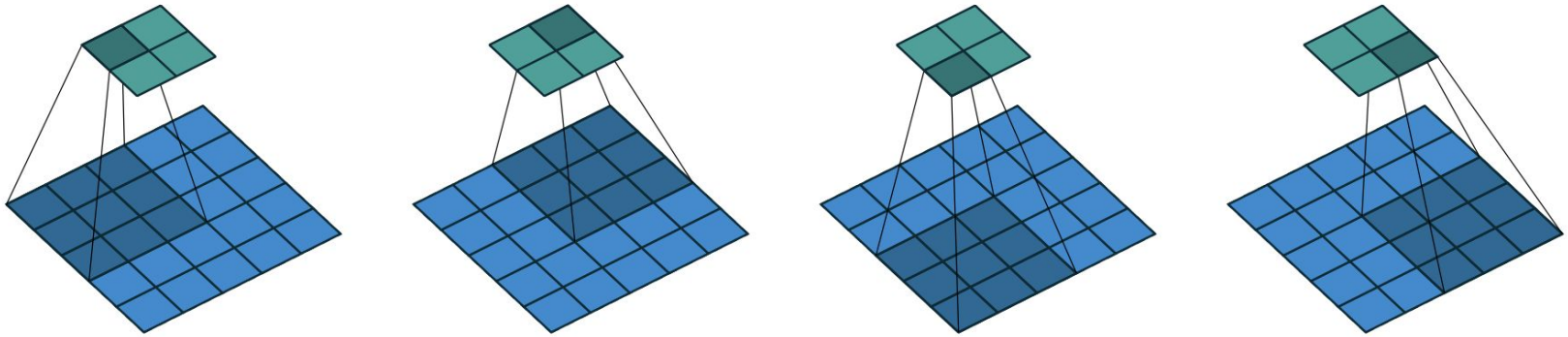
- Keep width and height consistent with the previous layer
- Keep the information around the border of the image



Stride

Distance between two consecutive positions of the kernel

- Allows us to control the output resolution



Computing the output size

For each dimension of an input image with

- Image dimension of size i
- Kernel of size k
- Padding of size p
- Stride of size s

The size of output dimension is computed by:

$$o = \left\lfloor \frac{i + 2p - k}{s} \right\rfloor + 1$$

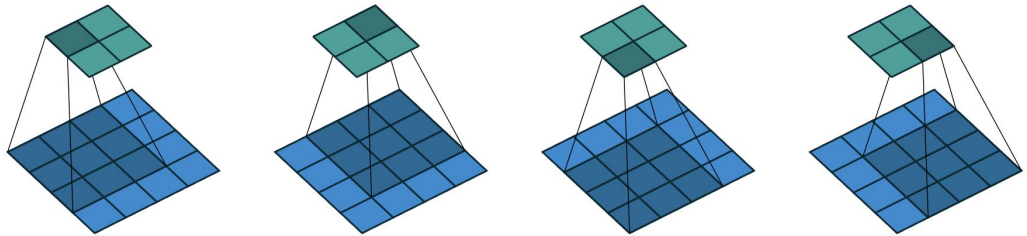
Example

$$o = \left\lfloor \frac{i + 2p - k}{s} \right\rfloor + 1$$

- No padding, unit stride → (the output size < the input size)

$i = 4,\ k = 3,\ s = 1$ and $p = 0$

$$o = \left\lfloor \frac{4 + 2 \times 0 - 3}{1} \right\rfloor + 1 = 2$$



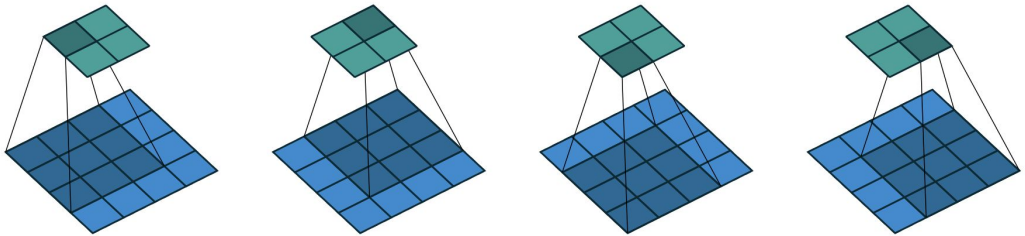
$$o = \left\lfloor \frac{i + 2p - k}{s} \right\rfloor + 1$$

Example

- No padding, unit stride → (the output size < the input size)

$i = 4,\ k = 3,\ s = 1$ and $p = 0$

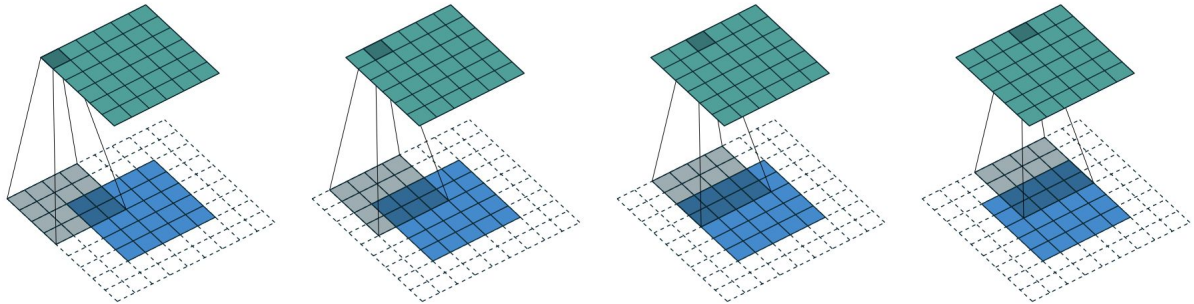
$$o = \left\lfloor \frac{4 + 2 \times 0 - 3}{1} \right\rfloor + 1 = 2$$



- Arbitrary padding, unit strides

$i = 5,\ k = 4,\ s = 1$ and $p = 2$

$$o = \left\lfloor \frac{5 + 2 \times 2 - 4}{1} \right\rfloor + 1 = 6$$



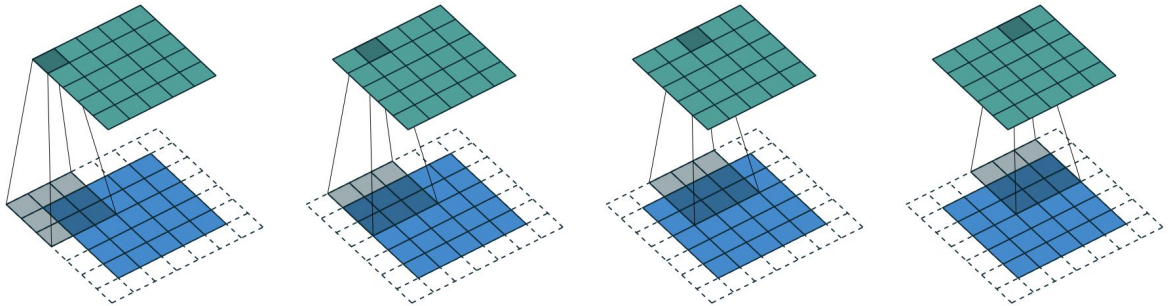
Example

$$o = \left\lfloor \frac{i + 2p - k}{s} \right\rfloor + 1$$

- Half padding, unit strides → (the output size be the same as the input size)

$i = 5,\ k = 3,\ s = 1$ and $p = 1$

$$o = \left\lfloor \frac{5 + 2 \times 1 - 3}{1} \right\rfloor + 1 = 5$$



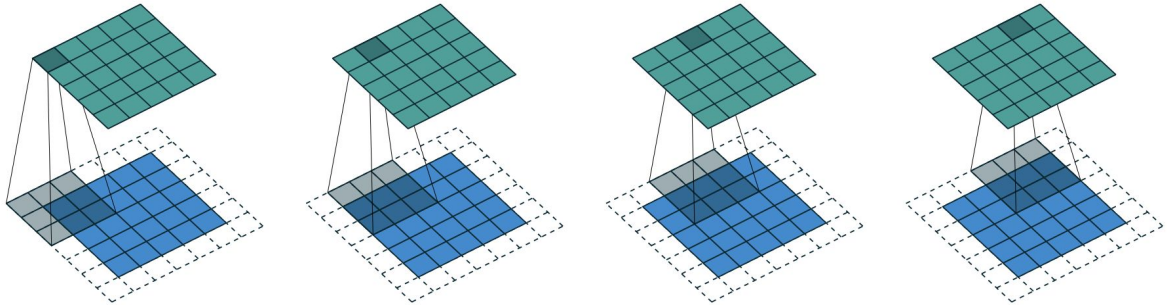
Example

$$o = \left\lfloor \frac{i + 2p - k}{s} \right\rfloor + 1$$

- Half padding, unit strides → (the output size be the same as the input size)

$i = 5,\ k = 3,\ s = 1$ and $p = 1$

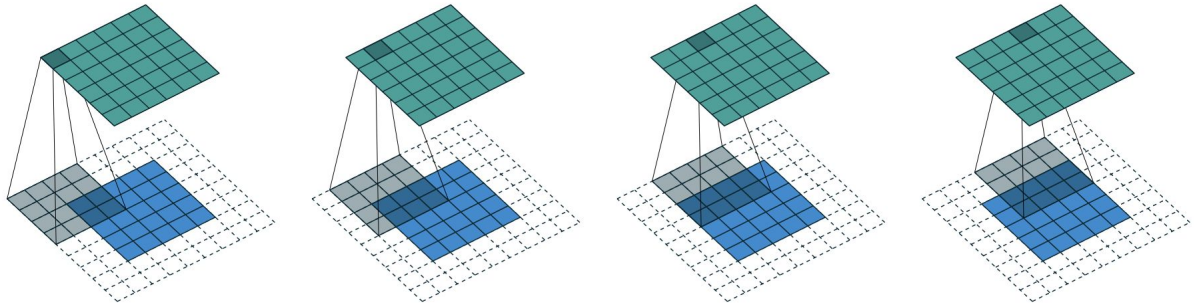
$$o = \left\lfloor \frac{5 + 2 \times 1 - 3}{1} \right\rfloor + 1 = 5$$



- Full padding, unit strides → (the output size >= the input size)

$i = 5,\ k = 3,\ s = 1$ and $p = 2$

$$o = \left\lfloor \frac{5 + 2 \times 2 - 3}{1} \right\rfloor + 1 = 6$$



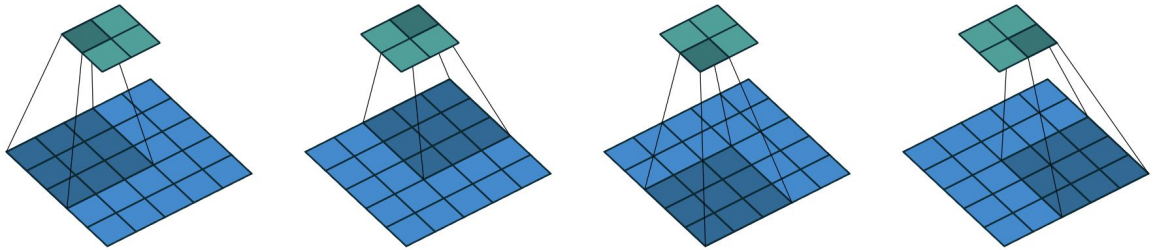
Example

$$o = \left\lfloor \frac{i + 2p - k}{s} \right\rfloor + 1$$

- No padding, arbitrary strides

  $i = 5,\ k = 3,\ s = 2$ and $p = 0$

  $$o = \left\lfloor \frac{5 + 2 \times 0 - 3}{2} \right\rfloor + 1 = 2$$



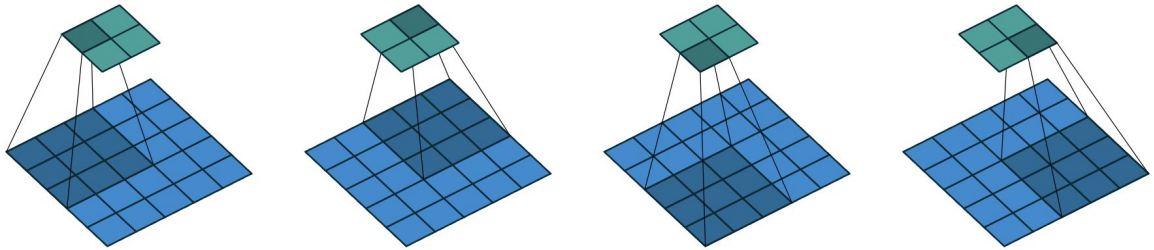
$$o = \left\lfloor \frac{i + 2p - k}{s} \right\rfloor + 1$$

Example

- No padding, arbitrary strides

  $i = 5,\ k = 3,\ s = 2$ and $p = 0$

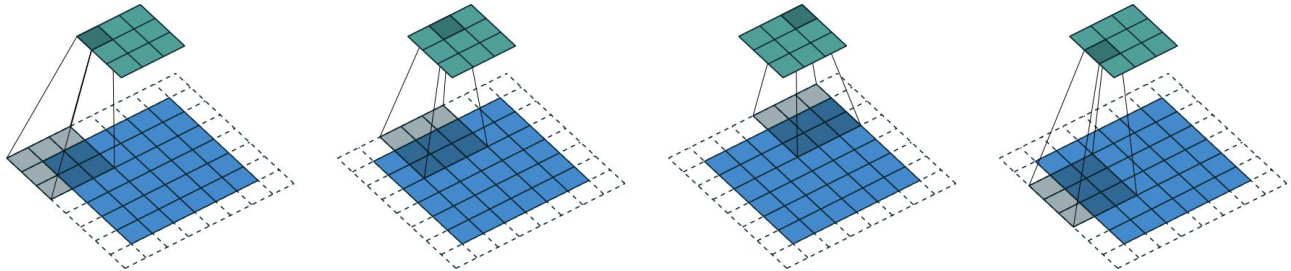
  $$o = \left\lfloor \frac{5 + 2 \times 0 - 3}{2} \right\rfloor + 1 = 2$$



- Arbitrary padding and strides

  $i = 6, k = 3,\ s = 2$ and $p = 1$

  $$o = \left\lfloor \frac{6 + 2 \times 1 - 3}{2} \right\rfloor + 1 = 3$$

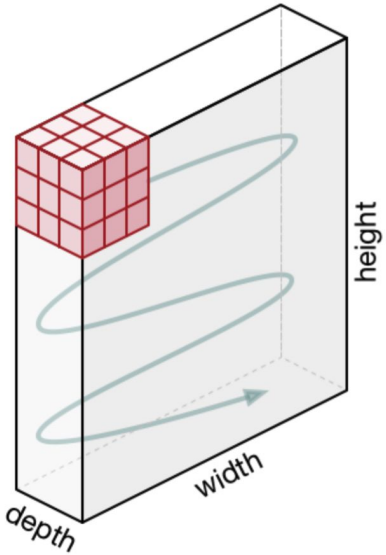


CNN on RGB

What if we have 3 colour channels?

RGB

Convolution on RGB input



Convolution in RGB

kernel dim = $3 \times k \times k$
image dim = $3 \times i \times i$

The kernel becomes a 3-dimensional tensor!

Convolution on RGB input

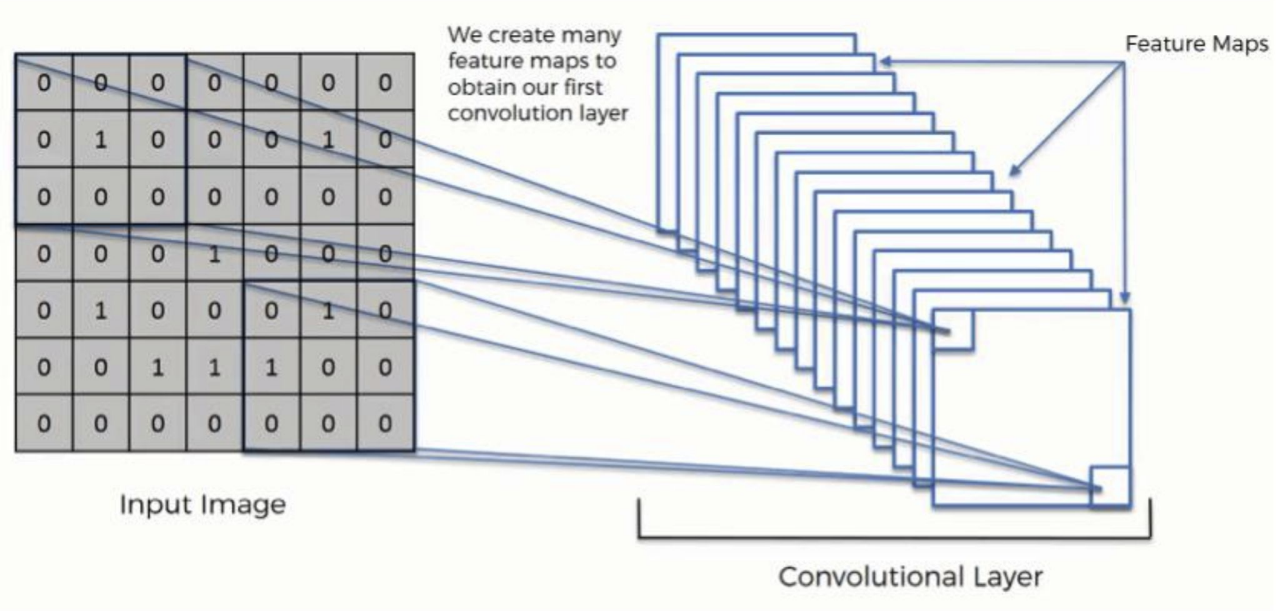
Colour input image: 3×28×28

Convolution kernel: 3×3×3

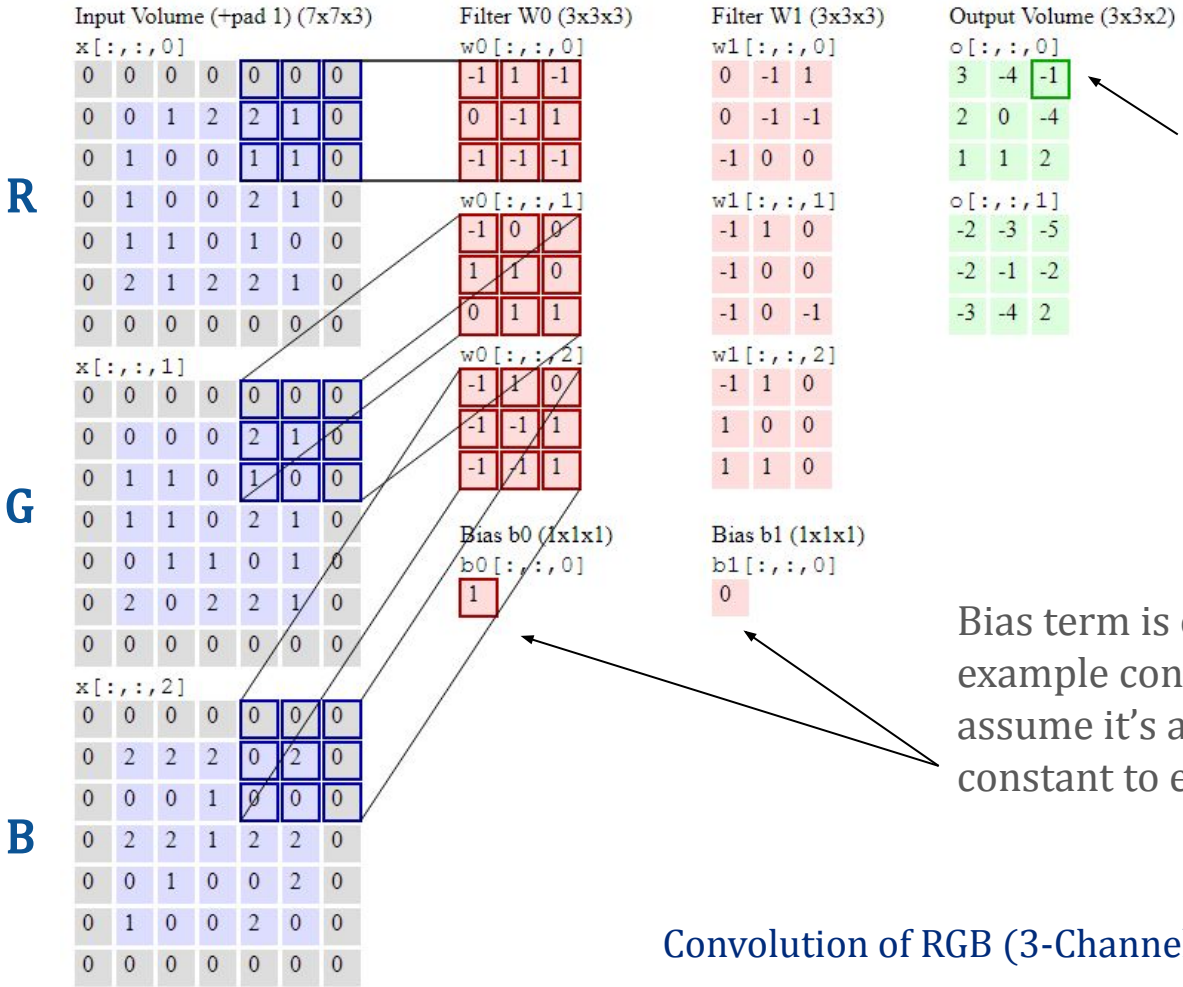
Questions

- How many trainable weights are there?
- What if we want to detect many **features** ?

Expanding Feature Maps



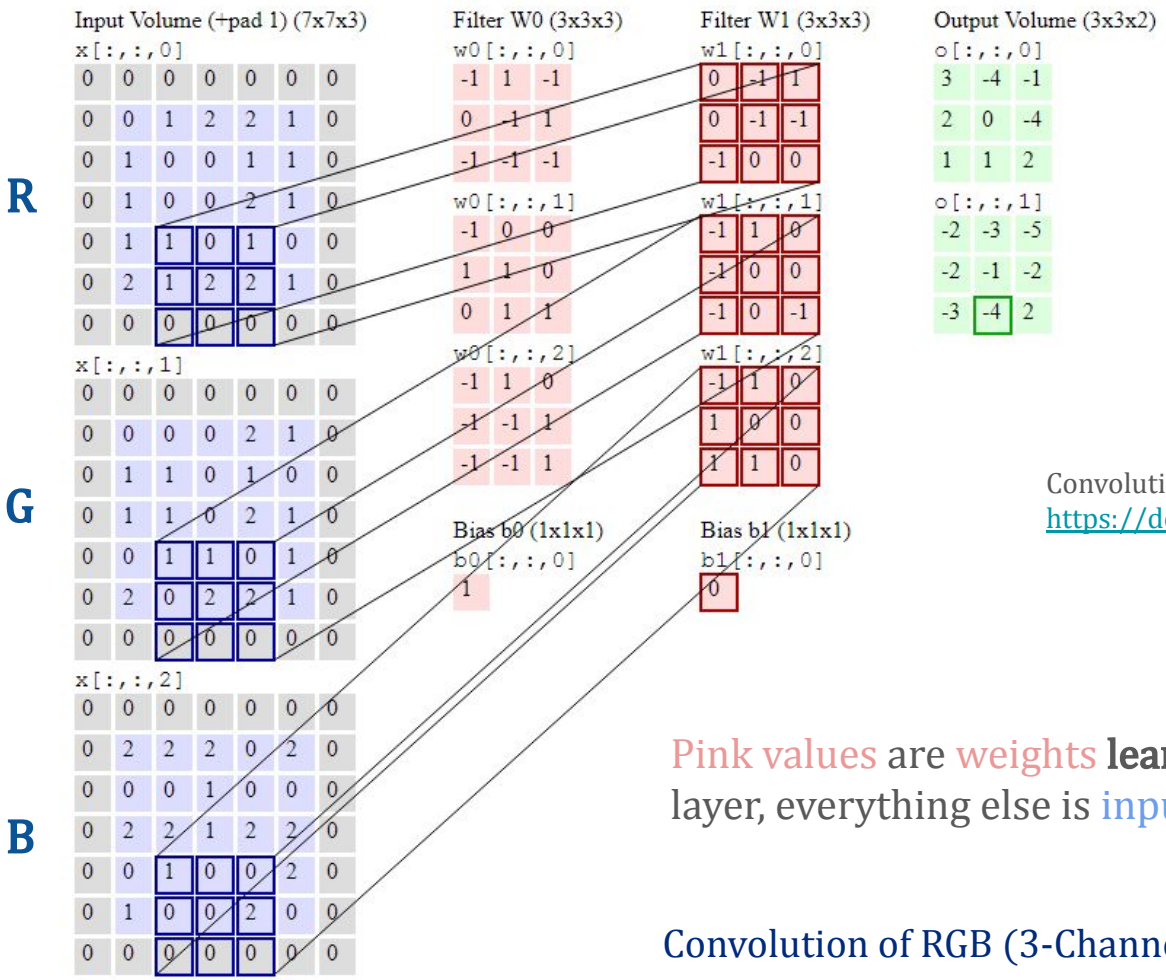
stride = 2
Padding = 1



Result of adding 27 numbers + bias

Bias term is often ignored in the example convolutions, but we can assume it's always there and adds a constant to each result

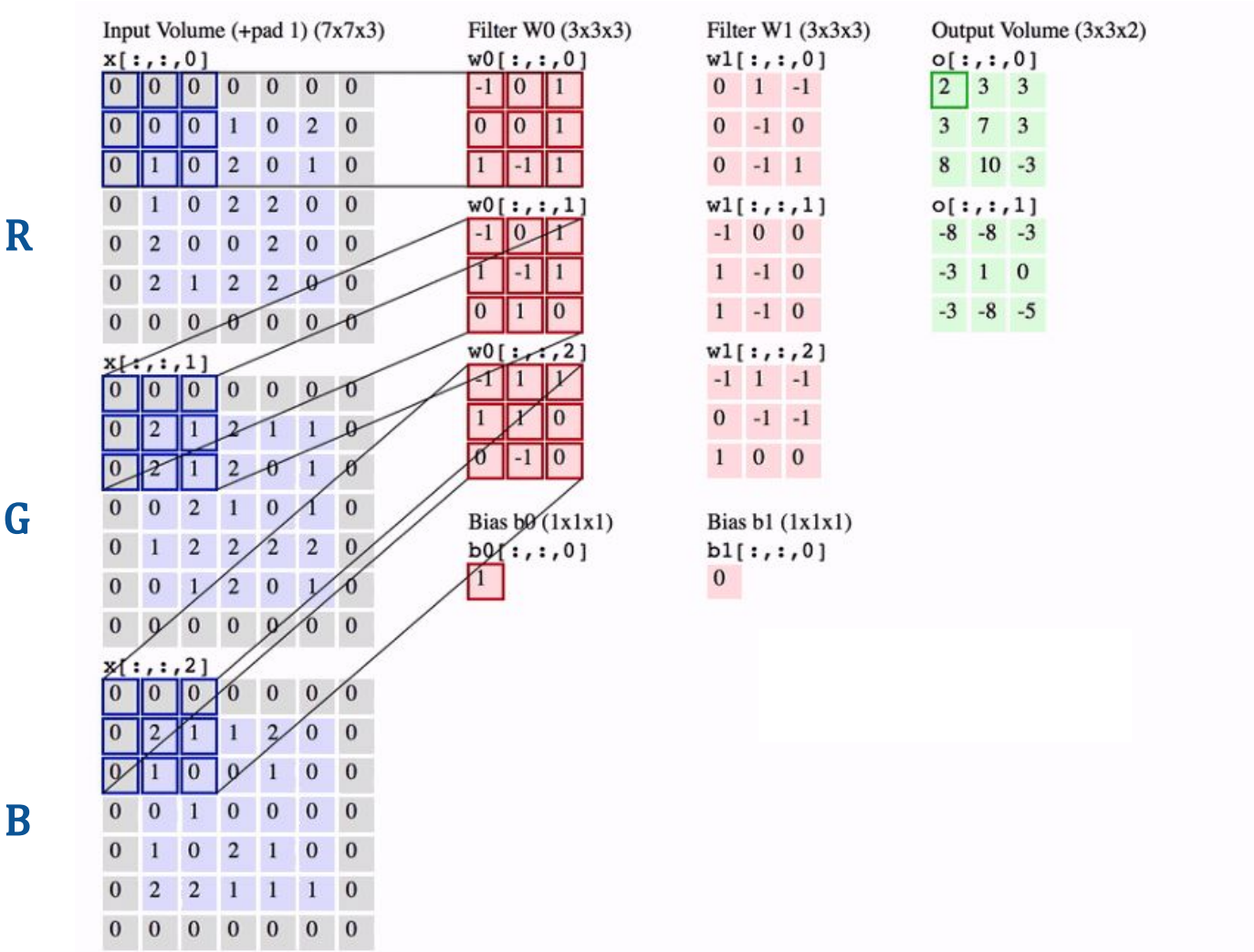
Convolution of RGB (3-Channel) image with Filter "W0"



Convolution Interactive Demo:
https://deeplizard.com/resource/pavq7noze2

Pink values are weights **learned** by convolutional layer, everything else is input/output

Convolution of RGB (3-Channel) image with Filter "W1"


Input Volume (+pad 1) (7x7x3)
x[:,:,0]
0 0 0 0 0 0 0
0 0 0 1 0 2 0
0 1 0 2 0 1 0
0 1 0 2 2 0 0
0 2 0 0 2 0 0
0 2 1 2 2 0 0
0 0 0 0 0 0 0
x[:,:,1]
0 0 0 0 0 0 0
0 2 1 2 1 1 0
0 2 1 2 0 1 0
0 0 2 1 0 1 0
0 1 2 2 2 2 0
0 0 1 2 0 1 0
0 0 0 0 0 0 0
x[:,:,2]
0 0 0 0 0 0 0
0 2 1 1 2 0 0
0 1 0 0 1 0 0
0 0 1 0 0 0 0
0 1 0 2 1 0 0
0 2 2 1 1 1 0
0 0 0 0 0 0 0
R
G
B
Filter W0 (3x3x3)
w0[:,:,0]
-1 0 1
0 0 1
1 -1 1
w0[:,:,1]
-1 0 1
1 -1 1
0 1 0
w0[:,:,2]
-1 1 1
1 1 0
0 -1 0
Bias b0 (1x1x1)
b0[:,:,0]
1
Filter W1 (3x3x3)
w1[:,:,0]
0 1 -1
0 -1 0
0 -1 1
w1[:,:,1]
-1 0 0
1 -1 0
1 -1 0
w1[:,:,2]
-1 1 -1
0 -1 -1
1 0 0
Bias b1 (1x1x1)
b1[:,:,0]
0
Output Volume (3x3x2)
o[:,:,0]
2 3 3
3 7 3
8 10 -3
o[:,:,1]
-8 -8 -3
-3 1 0
-3 -8 -5

Convolution on RGB input

Colour input image: 3×28×28

Convolution kernels: 5×3×8×8

Questions

- How many input channels are there?
- How many output channels are there?
- How many trainable weights are there?

Pooling Operator

Consolidating Information

In a neural network with fully-connected layers, we reduced the number of units before the final output layer.

Q: Why?

Consolidating Information

In a neural network with fully-connected layers, we reduced the number of units before the final output layer.

Q: Why?

To be able to **consolidate information** , and remove information not useful for the current task

Consolidating Information

In a neural network with fully-connected layers, we reduced the number of units before the final output layer.

Q: Why?

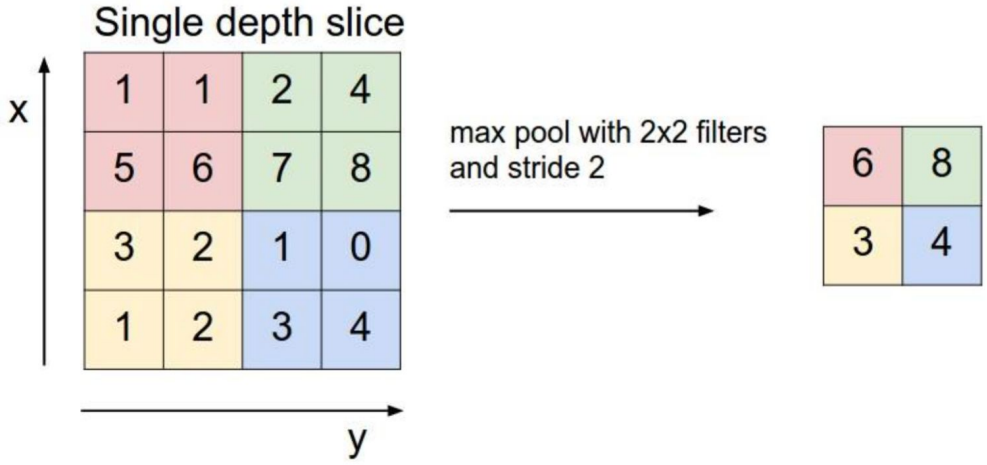
To be able to **consolidate information** , and remove information not useful for the current task

Q: How can we consolidate information in a neural network with convolutional layers?

Strided convolutions , **max pooling** , **average pooling**

Max pooling

pooling layers provide invariance to small translations of the input.

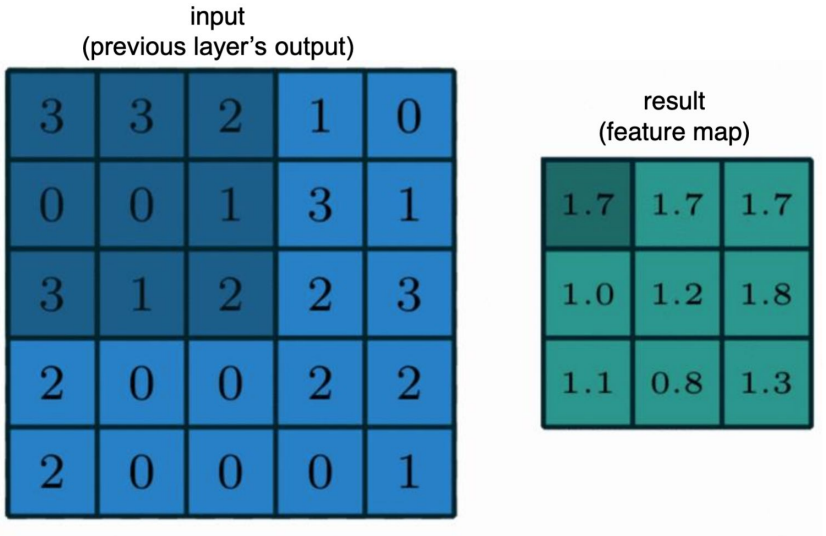


The size of output dimension is computed by:

$$o = \left\lfloor \frac{i-k}{s} \right\rfloor + 1$$

- Image dimension of size i
- Kernel of size k
- Stride of size s

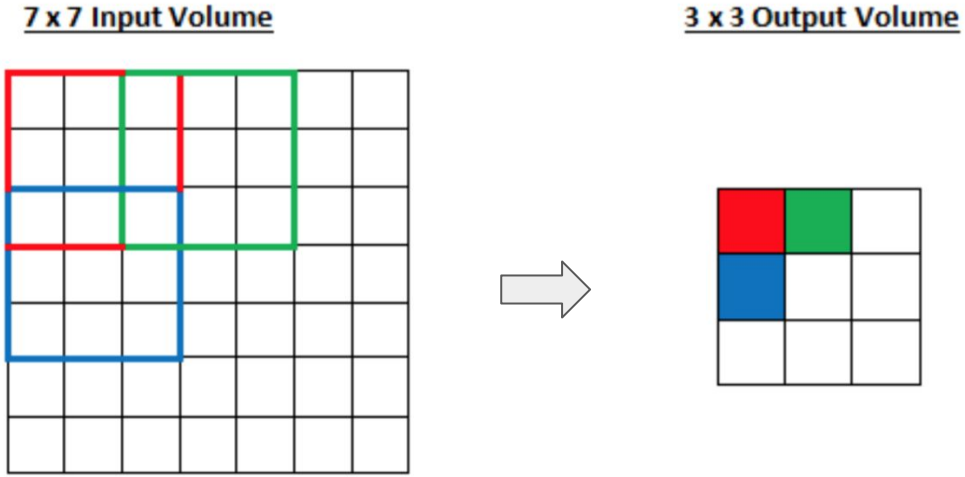
Average pooling



Max pooling generally works better

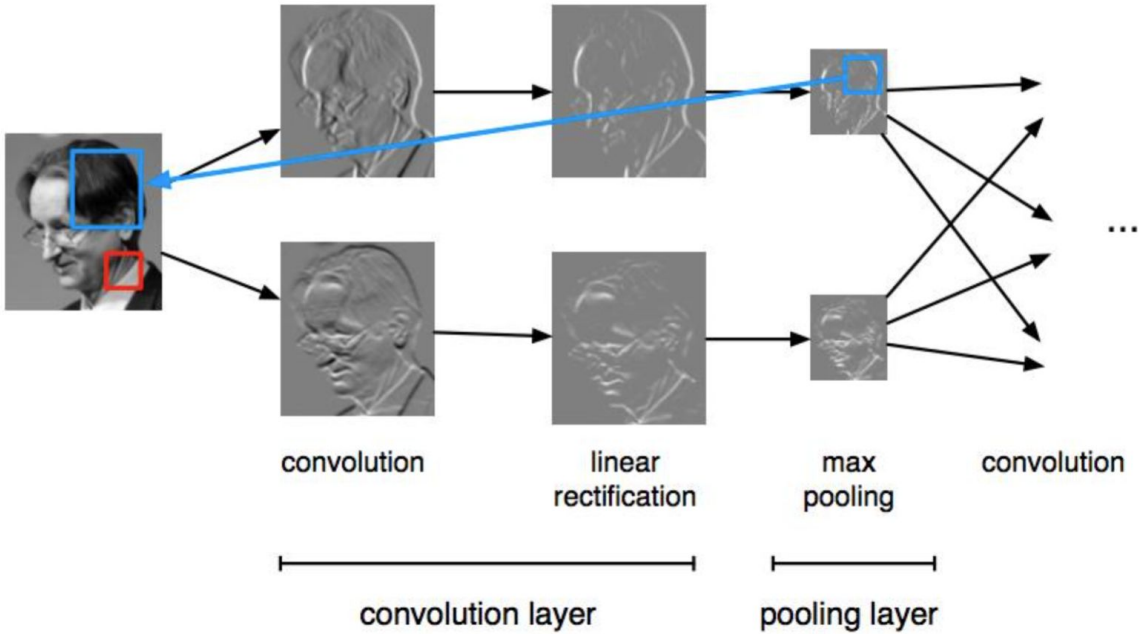
Strided convolution

- More recently people are doing away with pooling operations, **using strided convolutions instead** .
- Shift the kernel by s (e.g. s=2) when computing convolution.



PyTorch Implementation

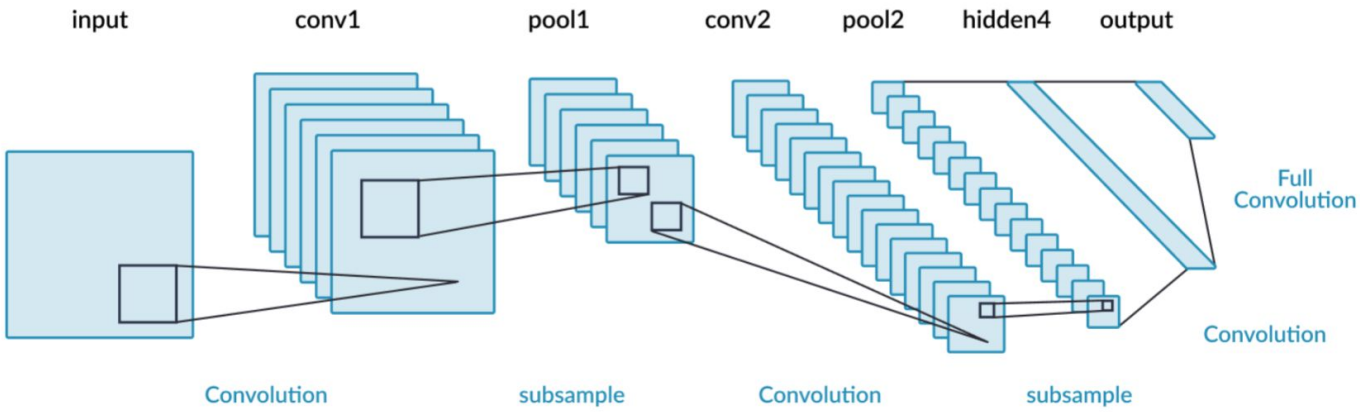
Putting it all together



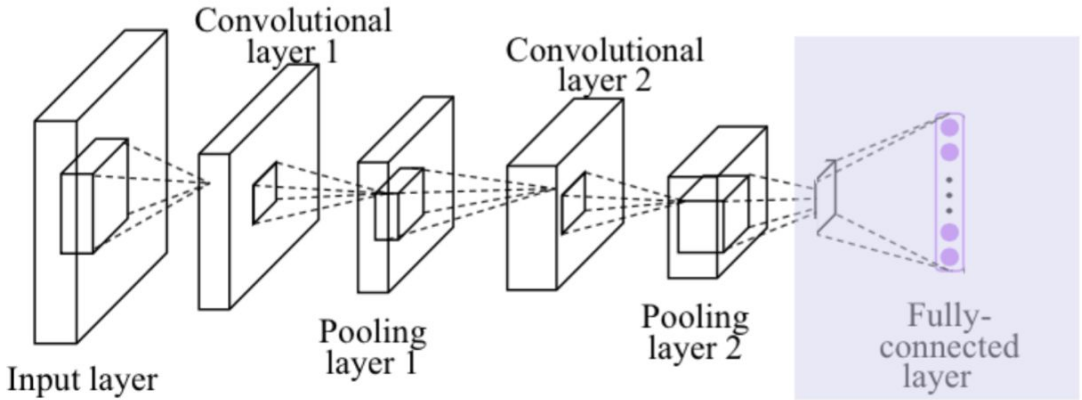
CNN Architecture Blueprint

As we go through the CNN network layer by layer:

- The **filter depth increases**
- The feature map **height and width decreases**

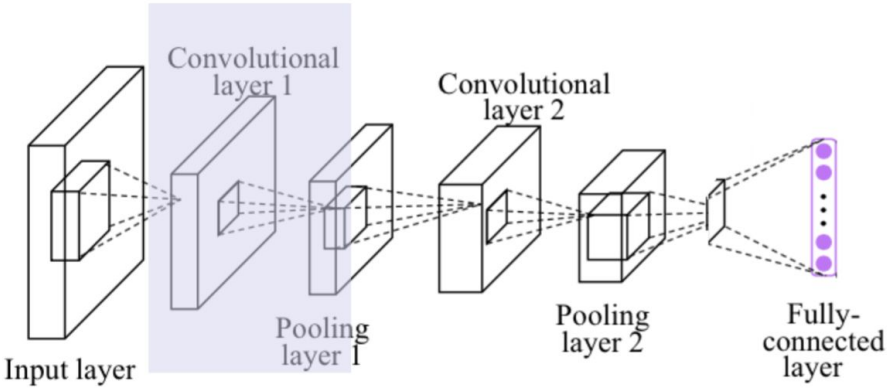


CNN in PyTorch: Recall Linear




```
self.fc = nn.Linear(n, 10)
```

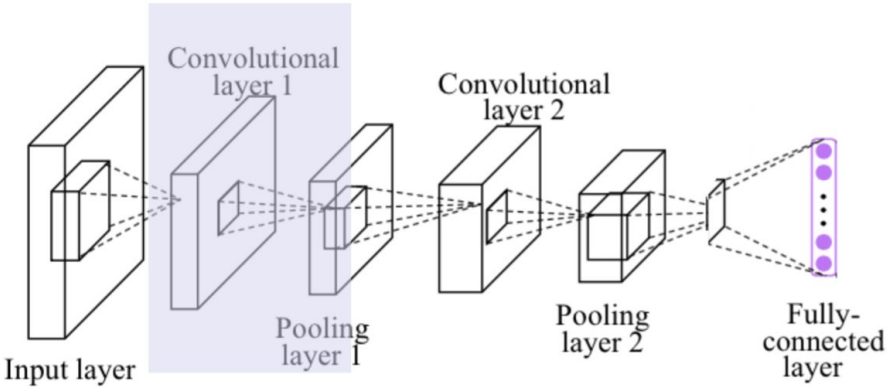
CNN in PyTorch: Conv2D




```
self.conv1 = nn.Conv2d(in_channels = 3,
                       out_channels = 7,
                       kernel_size= 5,
                       stride = 1,
                       padding = 1)
```

CNN in PyTorch: Conv2D

```
# non-square kernels and unequal stride and with padding
m = nn.Conv2d(16, 33, (3, 5), stride=(2, 1), padding=(4, 2))
```




```
self.conv1 = nn.Conv2d(in_channels = 3,
                       out_channels = 7,
                       kernel_size= 5,
                       stride = 1,
                       padding = 1)
```

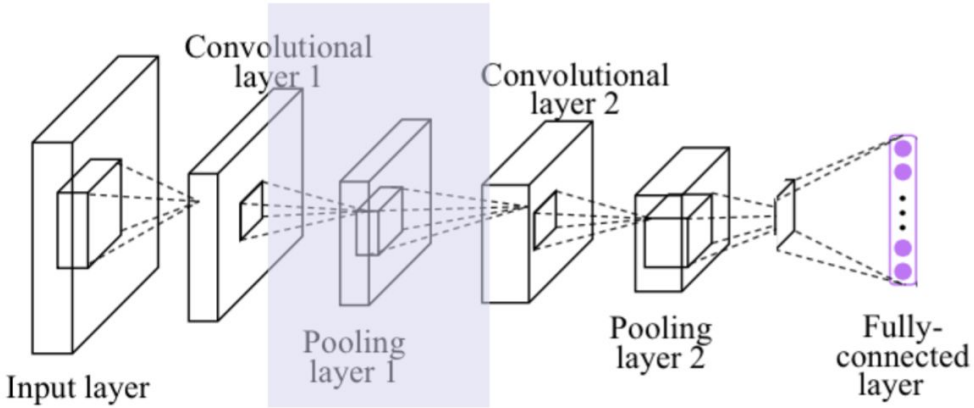
int

int or tuple

Default → 1

Default → 0

CNN in PyTorch: MaxPool2d




```
self.pool = nn.MaxPool2d(kernel_size = 2,
                         stride = 2)
```

CNN in PyTorch

```
class LargeNet(nn.Module):
    def __init__(self):
        super(LargeNet, self).__init__()
        self.name = "large"
        self.conv1 = nn.Conv2d(3, 5, 5)
        self.pool = nn.MaxPool2d(2, 2)
        self.conv2 = nn.Conv2d(5, 10, 5)
        self.fc1 = nn.Linear(10 * 5 * 5, 32)
        self.fc2 = nn.Linear(32,10)

    def forward(self, x):
        x = self.pool(F.relu(self.conv1(x)))
        x = self.pool(F.relu(self.conv2(x)))
        x = x.view(-1, 10 * 5 * 5)
        x = F.relu(self.fc1(x))
        x = self.fc2(x)
        return x
```

Questions?