Deep Learning

Lecture 11: Convolutional Neural Network

Dr. Mehrdad Maleki

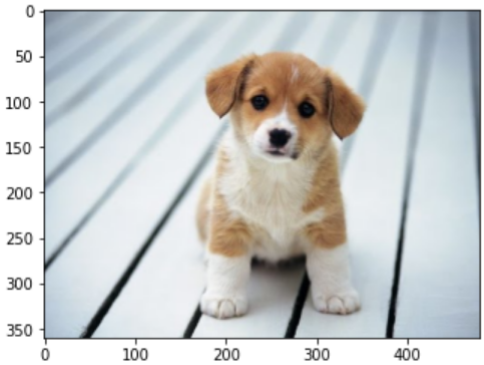
```
1 import matplotlib.pyplot as plt
2 import numpy as np
```

any image in your working directory

```
1 dog=plt.imread("dog.jpg")
```

read image as dog variable

```
1 plt.imshow(dog)
```



```
1 dog.shape
```

colored image is actually a 3 dimentional array

```
(360, 480, 3)
```

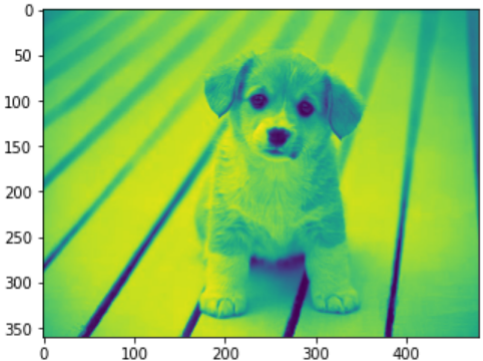
channel

height

width

```
1 red_channel=dog[:,:,0]
```

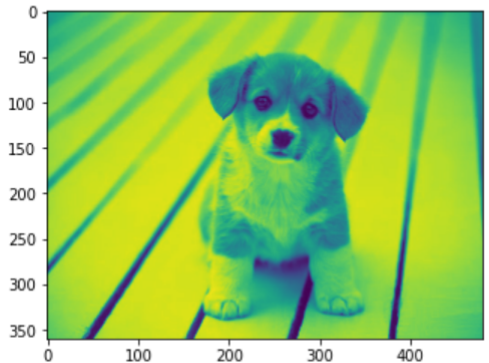
```
1 plt.imshow(red_channel)
```



```
1 green_channel=dog[:,:,1]
```

select just the second channel, green

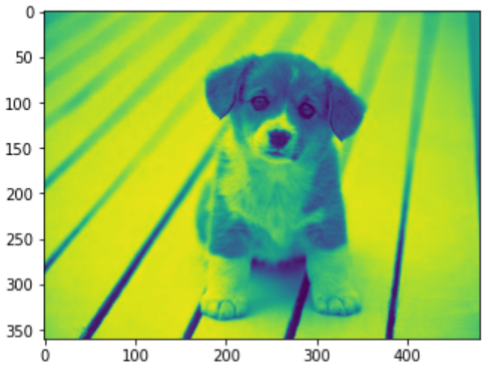
```
1 plt.imshow(green_channel)
```



```
1 blue_channel=dog[:,:,2]
```

select just the third channel, blue

```
1 plt.imshow(blue_channel)
```



```
1 red_channel.shape
```

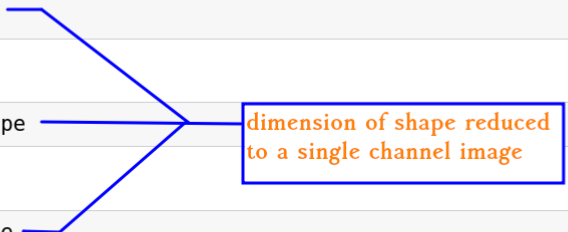
```
(360, 480)
```

```
1 green_channel.shape
```

```
(360, 480)
```

```
1 blue_channel.shape
```

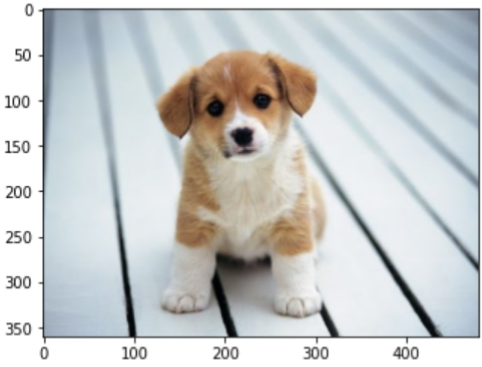
```
(360, 480)
```



```
1 mirror_dog=dog[:,::-1]
```

reverse the columnd of the array

```
1 plt.imshow(mirror_dog)
```

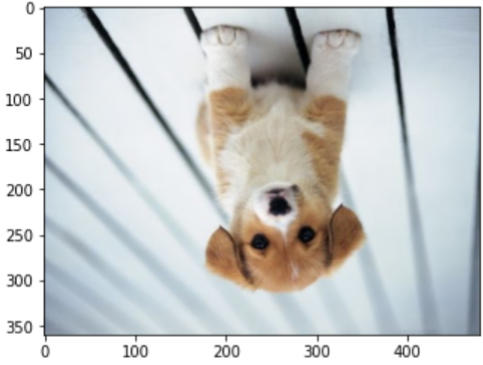


if A is a one-dimensional array then A[2:6] gives us A[2] until A[5]. But A[2:6:2] gives us A[2] and A[4], the third number is the jumping length.
A[6:2:-1] gives us A[6] until A[3] (reverese order)
A[: :-1] produce reverse list of A

```
1 reverse_dog=dog[::-1,:]
```

reverse rows of the array

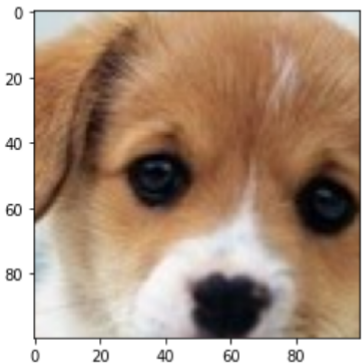
```
1 plt.imshow(reverse_dog)
```

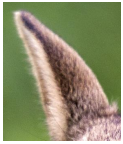


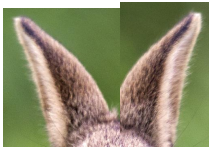
```
1 face_dog=dog[50:150,200:300]
```

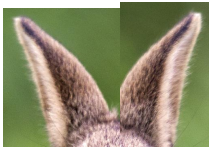
select rows from 50 til 149 and columns from 200 til 299

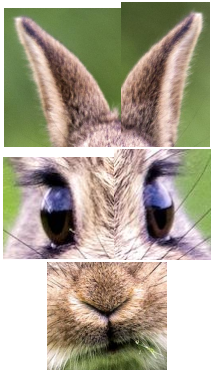
```
1 plt.imshow(face_dog)
```

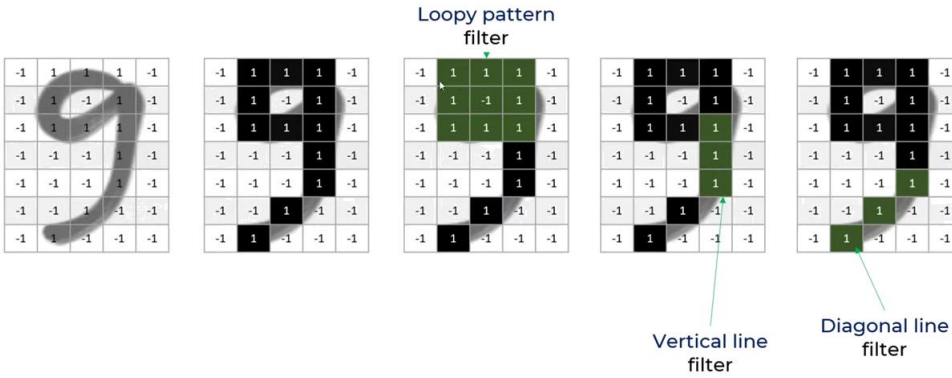











Loopy pattern filter
Vertical line filter
Diagonal line filter

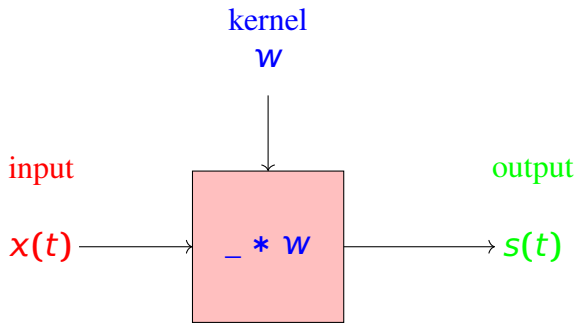
-1	**1**	**1**	**1**	-1
-1	**1**	-1	**1**	-1
-1	**1**	**1**	**1**	-1
-1	-1	-1	**1**	-1
-1	-1	-1	**1**	-1
-1	-1	**1**	-1	-1
-1	**1**	-1	-1	-1

1	1	1
1	-1	1
1	1	1

Convolution

- Let $x(t)$ be a noisy signal of a variable that depend on the time.
- Let $w(t')$ be the weight function at time t'.
- Then $s(t) = (x * w)(t)$ is the smoothed signal with respect to the weight w.

$$s(t) = (x * w)(t) = \int_{-\infty}^{\infty} x(a)w(t-a)da$$


kernel
w
input
output
x(t)
_ * w
s(t)

Discerete Convolution

$$S(i,j) = (X * K)(i,j) = \sum_{m=-\infty}^{\infty} \sum_{n=-\infty}^{\infty} X(i-m, j-n)K(m,n)$$

This operator has commutative property, i.e.,

$$(X * K)(i,j) = (K * X)(i,j)$$

Cross-correlation

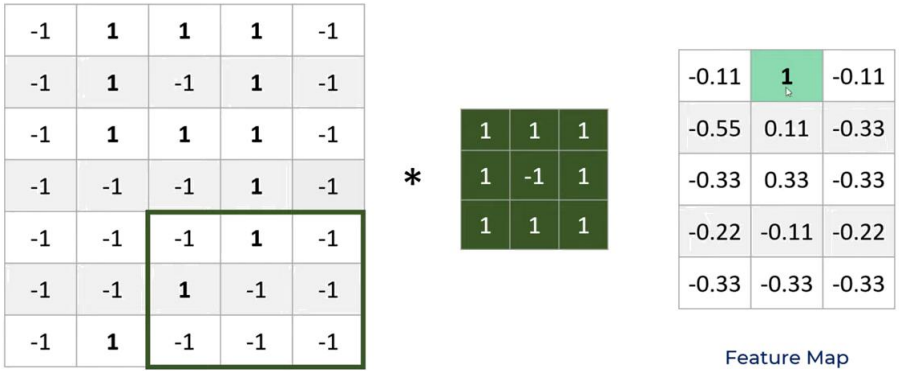
$$S(i,j) = (X * K)(i,j) = \sum_{m=-\infty}^{\infty} \sum_{n=-\infty}^{\infty} X(i+m, j+n) K(m,n)$$

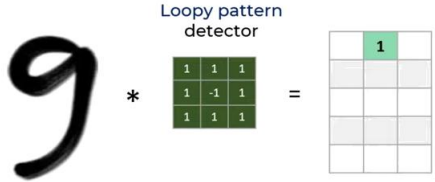
This operator does not have the commutative property, i.e.,

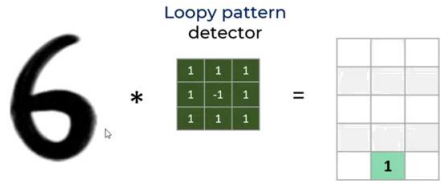
$$(X * K)(i,j) \neq (K * X)(i,j)$$

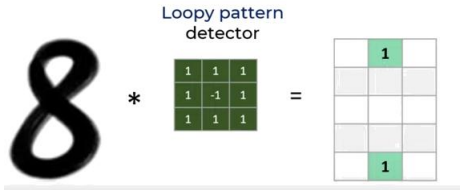
Why Convolutional Networks

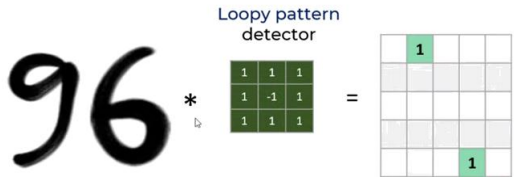
- Sparse interactions.
- Parameter sharing.
- Equivalent representation.


-1 1 1 1 -1
-1 1 -1 1 -1
-1 1 1 1 -1
-1 -1 -1 1 -1
-1 -1 -1 1 -1
-1 -1 1 -1 -1
-1 1 -1 -1 -1
*
1 1 1
1 -1 1
1 1 1
-0.11 1 -0.11
-0.55 0.11 -0.33
-0.33 0.33 -0.33
-0.22 -0.11 -0.22
-0.33 -0.33 -0.33
Feature Map


Loopy pattern detector
1 1 1
1 -1 1
1 1 1
1


Loopy pattern detector
1 1 1
1 -1 1
1 1 1
1

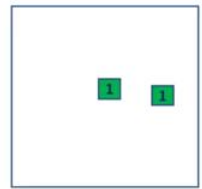

Loopy pattern
detector
1 1 1
1 -1 1
1 1 1
*
=
1
1

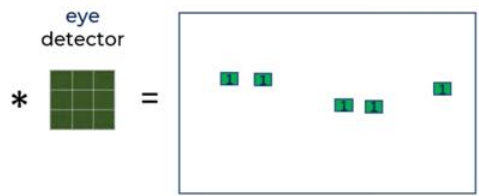

Loopy pattern
detector
1 1 1
1 -1 1
1 1 1
1
1

*

eye
detector

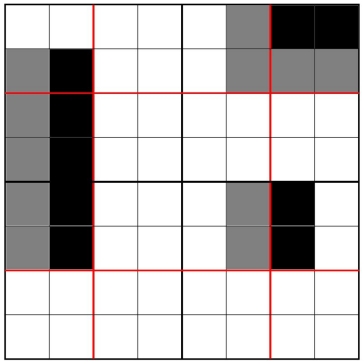
=

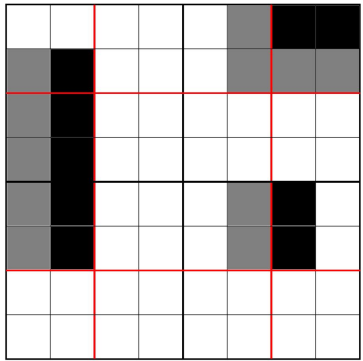


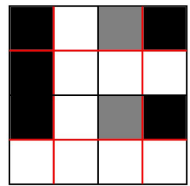

eye
detector
*
=

Pooling

- A pooling function replaces the output of the net at a certain location with a summary statistic of the nearby output.
- For creating a condensed feature map (reduce dimensionality).
- The **max pool** report the maximum output within a rectangular neighborhood.

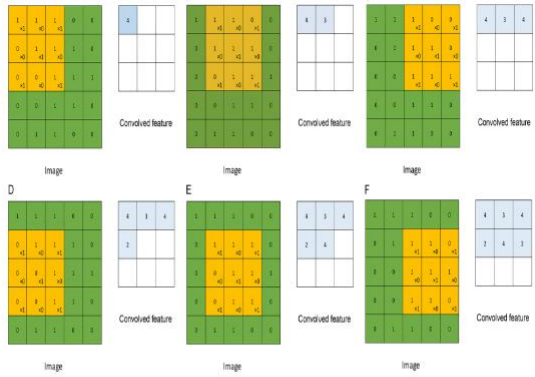






Padding

- The shape of input image X is $m \times n$.
- The shape of kernel K is $f \times f$.
- Then the shape of output $S = X * K$ is $(m - f + 1) \times (n - f + 1)$.
- So the output does not have the same shape as input.



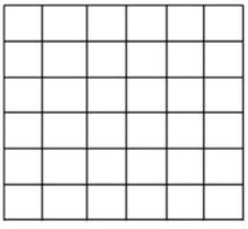
Sign in to download hi-res image

Padding

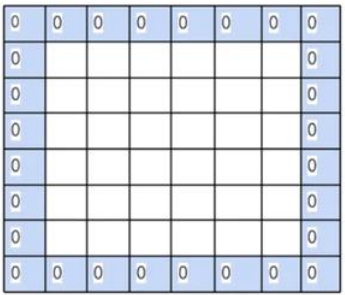
- The process of adding extra layers to the input image to avoid input/output shape difference is called **padding**.

Padding

- The process of adding extra layers to the input image to avoid input/output shape difference is called **padding**.



6x6 image



6x6 image with 1 layer of zero padding

Types of Padding

1. **Valid padding:** no padding, i.e.,

$$\textbf{input: } m \times n \xrightarrow{\textbf{filter: } f \times f} \textbf{output: } (m-f+1) \times (n-f+1)$$

Types of Padding

1. **Valid padding:** no padding, i.e.,

$$\textbf{input: } m \times n \xrightarrow{\textbf{filter: } f \times f} \textbf{output: } (m-f+1) \times (n-f+1)$$

2. **Same padding:**output has the same shape as the input if we add p layes of padding to the border. i.e.,

Types of Padding

1. **Valid padding:** no padding, i.e.,

$$\textbf{input: } m \times n \xrightarrow{\textbf{filter: } f \times f} \textbf{output: } (m-f+1) \times (n-f+1)$$

2. **Same padding:**output has the same shape as the input if we add p layes of padding to the border. i.e.,

$$\textbf{input: } m \times n \xrightarrow{\textbf{padding: } p} \textbf{padded: } (m+2p) \times (n+2p) \xrightarrow{\textbf{filter: } f \times f} \textbf{output: } m \times n$$

Types of Padding

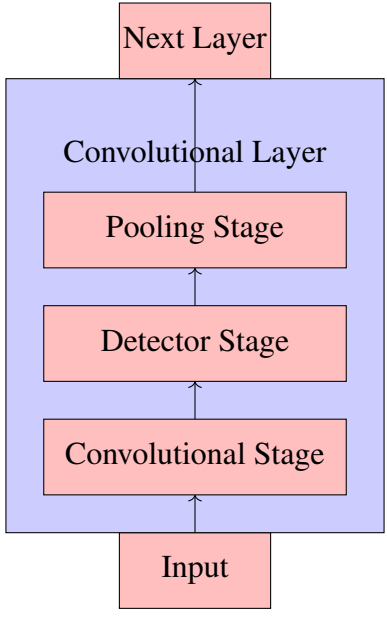
1. **Valid padding:** no padding, i.e.,

$$\textbf{input: } m \times n \xrightarrow{\textbf{filter: } f \times f} \textbf{output: } (m-f+1) \times (n-f+1)$$

2. **Same padding:**output has the same shape as the input if we add p layes of padding to the border. i.e.,

$$\textbf{input: } m \times n \xrightarrow{\textbf{padding: } p} \textbf{padded: } (m+2p) \times (n+2p) \xrightarrow{\textbf{filter: } f \times f} \textbf{output: } m \times n$$

for same padding p should be equal to $\frac{f-1}{2}$.


Next Layer
Convolutional Layer
Pooling Stage
Detector Stage
Convolutional Stage
Input

Thank You