

Consider using a feedforward network for image recognition:

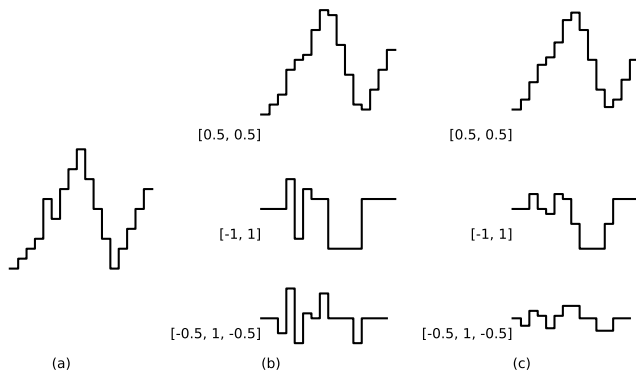
- What happens if the pixels were shuffled (consistently for all examples)?  
How well would humans perform with this new data set?
- If it had learned to recognize cats in the top of images, how does it perform with cats in the bottom of images?
- Convolutional neural networks tackle these problems by using filters that act on small patches of an image, and by sharing the parameters so they learn useful features no matter where in an image they occur.

# Kernels (one-dimensional case)

- A **kernel** (**convolution mask**, or **filter**) is a learned linear operator that is applied to local patches.
- In one dimension, suppose the input is a sequence (list)  $[x_0, \dots, x_{m-1}]$  and there is structure so that  $x_i$  is close to  $x_{i+1}$  in some sense.
- A **one-dimensional kernel** is a vector  $[w_0, \dots, w_{k-1}]$ , where  $k$  is the **kernel size**, which when applied to the sequence  $x = [x_0, \dots, x_{m-1}]$  produces a sequence  $y = [y_0, \dots, y_{m-k}]$  where

$$y[i] = \sum_{j=0}^{k-1} x[i+j] * w[j].$$

# One-dimensional kernels example



- (a) original signal  $[0, 1, 2, 3, 7, 5, 8, 10, 12, 9, 6, 3, 0, 2, 4, 6, 8]$
- (b) kernels  $[0.5, 0.5]$ ,  $[-1, 1]$ , and  $[-0.5, 1, -0.5]$  applied to the signal (a)
- (c) same kernels applied to the top signal in (b)

# Two-dimensional kernels

- A **two-dimensional kernel** is a  $j \times k$  array.
- This kernel  $w[i, j]$  applied to the two-dimensional array  $in$  produces a two-dimensional array  $out$  where

$$out[x, y] := \sum_{i=0}^{j-1} \sum_{j=0}^{k-1} in[x + i, y + j] * w[i, j].$$

The kernel size,  $j \times k$ , is usually much smaller than the size of the dimension of  $in$ .

- Two-dimensional kernels are used for images, where they are applied to patches of adjacent pixels.

## Two-dimensional kernel examples

What do the following kernels applied to a black-and-white image, where  $in[x, y]$  is the brightness of the pixel at position  $(x, y)$  do?

0	1
-1	0

(a)

1	0
0	-1

(b)

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

(c)

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

(d)

- Kernels (a) and (b) were invented by Roberts [1965].
- Kernels (c) and (d) are the Sobel–Feldman operators
- Now kernels are mostly learned.

# Convolutional Neural Networks

Convolutional neural networks learn kernel weights from data.

Two main aspects distinguish convolutional neural networks from ordinary neural networks:

- **Locality**: the values are a function of neighboring positions, rather than being based on all units as they are in a fully-connected layer.
- **Parameter sharing** or **weight tying**: the same parameters in a kernel are used at all locations in an image.

# Conv2D (for $k \times k$ kernel)

```
1: class Conv2D(k)
2:   initialize  $w[i,j]$  randomly, and  $d[i,j]$  to 0
3:   def output(input) ▷ input is  $x_d \times y_d$  array
4:     for each  $x : 0 \leq x \leq x_d - k, y : 0 \leq y \leq y_d - k$  do
5:        $out[x,y] := \sum_{i=0}^{k-1} \sum_{j=0}^{k-1} in[x+i,y+j] * w[i,j]$ 
6:     return out
7:   def Backprop(error)
8:     Initialize  $ierror[x,y]$  to 0
9:     for each  $x : 0 \leq x \leq x_d - k, y : 0 \leq y \leq y_d - k$  do
10:      for each  $i : 0 \leq i < k, j : 0 \leq j < k$  do
11:         $d[i,j] += in[x+i,y+j] * error[x,y]$ 
12:         $ierror[x+i,y+j] += error[x,y] * w[i,j]$ 
13:      return ierror
14:   def update() ▷ Same as for Dense
```

- The dimensions of kernels match the dimensions of input
- A kernel can be any rectangular size.
- The algorithm above removes elements from the edges. It is common to add **zero padding**.
- Multiple kernels can be applied concurrently, in **channels**. Channels at one level are inputs to all the channels at next. The input for camera images typically consists of channels corresponding to the colors red, green, and blue.
- Whether to use a **bias** is a parameter in most libraries.



- **Down-sampling** can reduce the size of an image.  
A **stride** for a dimension is a positive integer  $s$  such that each  $s$ th value is used in the output.  
A stride of  $s$  reduces the size of each dimension by  $s$ .
- A **pooling** layer uses a fixed function of kernel inputs, e.g. **max-pooling** uses the maximum of inputs for each kernel.
- When used for classification of the whole image (e.g., is there is a cat), a CNN typically ends with fully-connected layers.
- A **fully-convolutional network** can makes a prediction of each pixel (e.g., where is there a cat)
- A **shortcut connection** or a **skip connection** is a connection that skips some layers.
- In a **residual network**, the output from one layer are added to the outputs from a lower layer. The layer learns to fix errors of lower layers, as in boosting (Section 7.5.1)