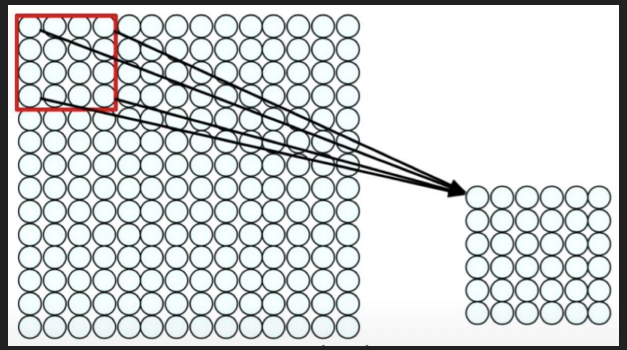
Week 3 Research

Maria Shevchuk

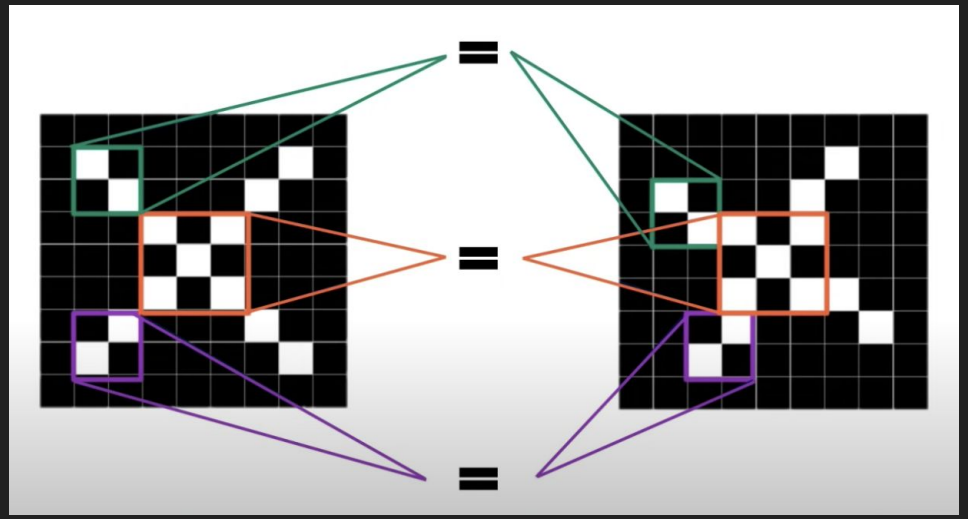
Feature Extraction via Convolution

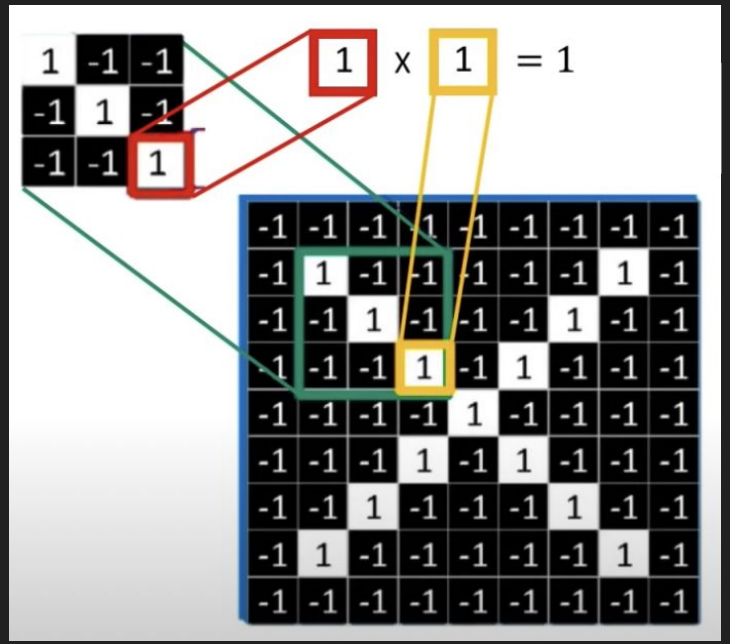


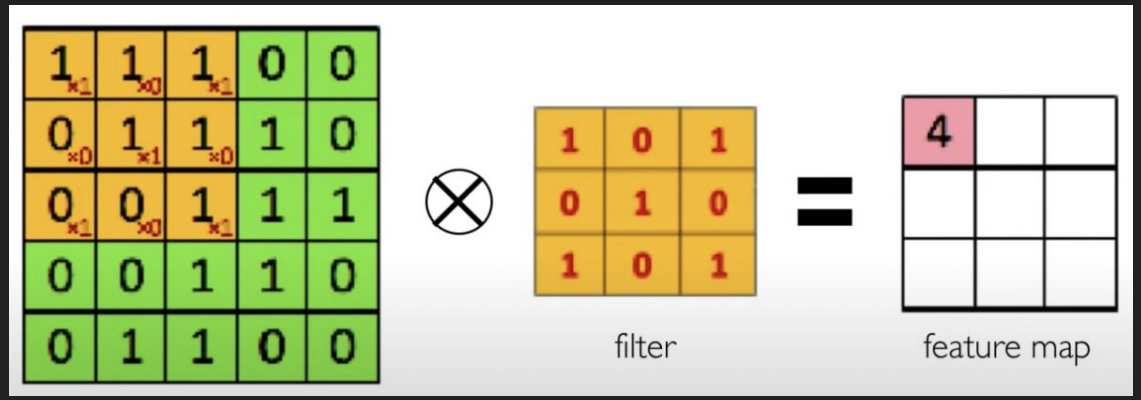
- Apply a set of weights (a filter/kernel)
- Want to extract different features?
  - use multiple filters!
- Are the images below equal?

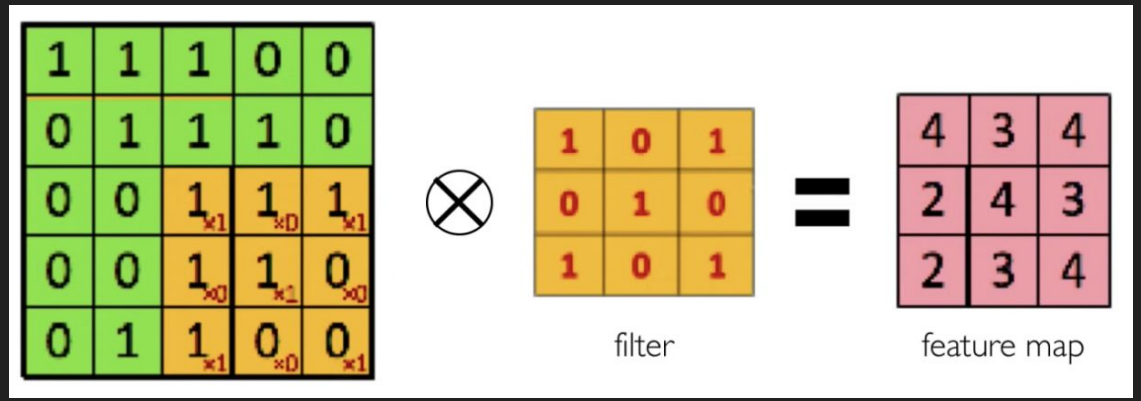
-1	-1	-1	-1	-1	-1	-1	-1	-1
-1	1	-1	-1	-1	-1	-1	1	-1
-1	-1	1	-1	-1	-1	1	-1	-1
-1	-1	-1	1	-1	1	-1	-1	-1
-1	-1	-1	-1	1	-1	-1	-1	-1
-1	-1	-1	1	-1	1	-1	-1	-1
-1	-1	1	-1	-1	-1	1	-1	-1
-1	1	-1	-1	-1	-1	-1	1	-1
-1	-1	-1	-1	-1	-1	-1	-1	-1

-1	-1	-1	-1	-1	-1	-1	-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	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



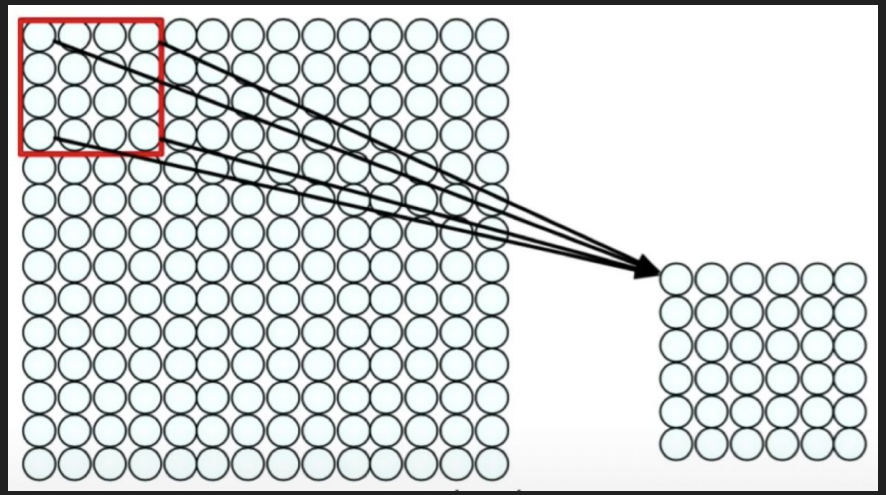

1 -1 -1
-1 1 -1
-1 -1 1
1 x 1 = 1
-1 -1 -1 -1 -1 -1 -1 -1 -1
-1 1 -1 -1 -1 -1 -1 1 -1
-1 -1 1 -1 -1 -1 1 -1 -1
-1 -1 -1 1 -1 1 -1 -1 -1
-1 -1 -1 -1 1 -1 -1 -1 -1
-1 -1 -1 1 -1 1 -1 -1 -1
-1 -1 1 -1 -1 -1 1 -1 -1
-1 1 -1 -1 -1 -1 -1 1 -1
-1 -1 -1 -1 -1 -1 -1 -1 -1


filter
feature map


filter
feature map

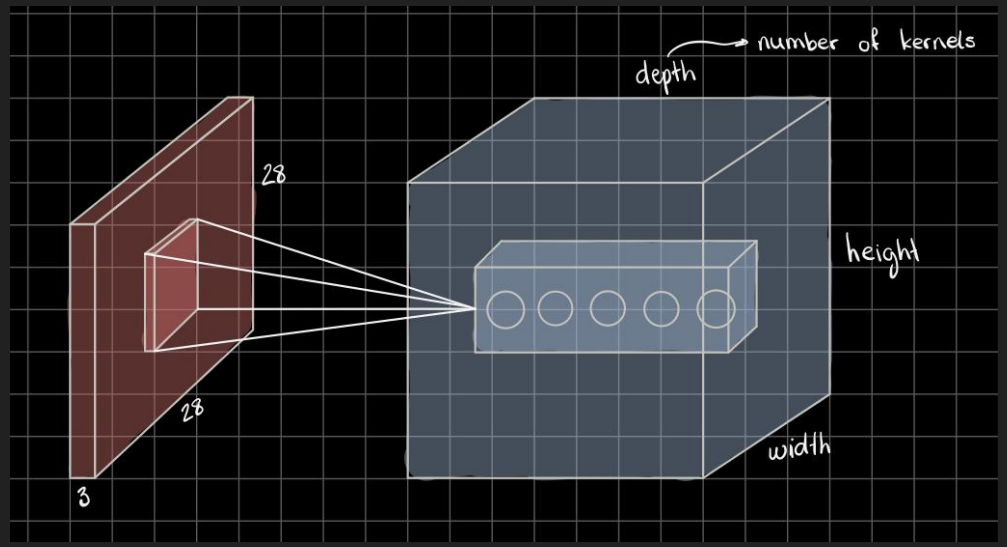
CNNs for Classification

- Feature extraction
  - Convolution
  - Non-linearity (activation function)
  - Pooling



Output volume

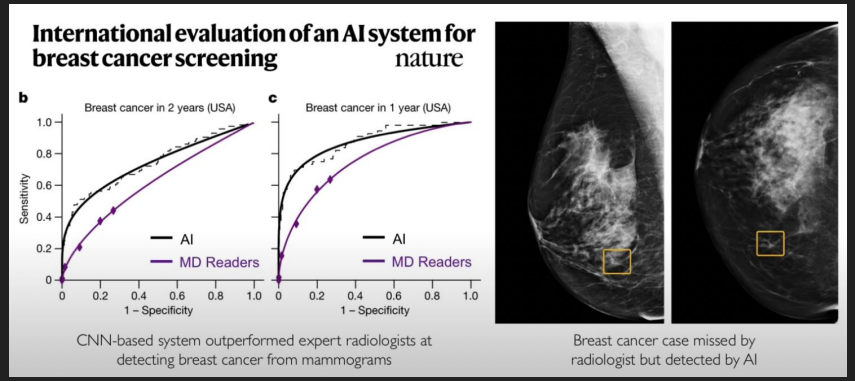
So what if we do want to detect multiple features?

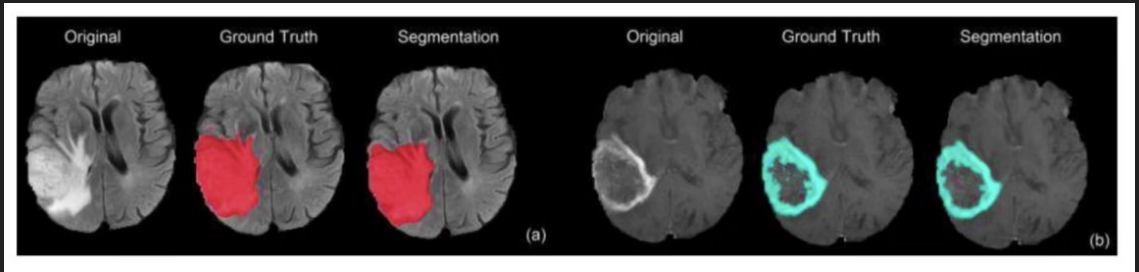


```python
import tensorflow as tf

def generate_model():
    model = tf.keras.Sequential([
        # first convolutional layer
        tf.keras.layers.Conv2D(32, filter_size=3, activation='relu'),
        tf.keras.layers.MaxPool2D(pool_size=2, strides=2),

        # second convolutional layer
        tf.keras.layers.Conv2D(64, filter_size=3, activation='relu'),
        tf.keras.layers.MaxPool2D(pool_size=2, strides=2),

        # fully connected classifier
        tf.keras.layers.Flatten(),
        tf.keras.layers.Dense(1024, activation='relu'),
        tf.keras.layers.Dense(10, activation='softmax')  # 10 outputs
    ])
    return model
```

What's so special about CNNs?

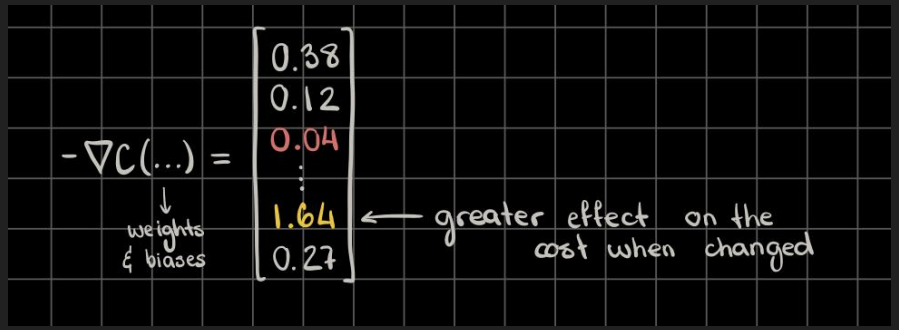
- Feature extraction
- Classification
- Uses:
  - Detection
  - Semantic Segmentation
    - downsampling/upsampling
  - Image captioning
- Applications:
  - Medicine
  - Security
  - Robotics





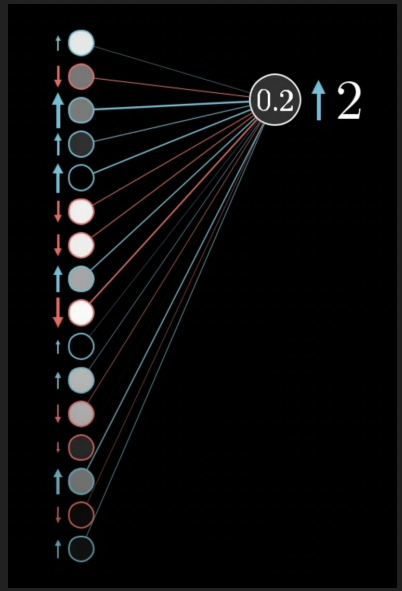
Backpropagation

- Computes negative gradient of the cost function
- Goal: efficiently decrease the cost



We can:

- Increase b
- Increase w_i (in proportion to a_i)
- Change a_i (in proportion to w_i)



							$\cdots$	Average over all training data
w_0	-0.08	$+0.02$	-0.02	$+0.11$	-0.05	-0.14	$\cdots$	-0.08
w_1	-0.11	$+0.11$	$+0.07$	$+0.02$	$+0.09$	$+0.05$	$\cdots$	$+0.12$
w_2	-0.07	-0.04	-0.01	$+0.02$	$+0.13$	-0.15	$\cdots$	-0.06
$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\ddots$	$\vdots$
$w_{13,001}$	$+0.13$	$+0.08$	-0.06	-0.09	-0.02	$+0.04$	$\cdots$	$+0.04$

Fun with(out) numbers

Calculating the gradient component of the individual weights

Notation:

- $L \rightarrow$ l^{th} layer ($l=1 \rightarrow$ first, $l=L \rightarrow$ last)
- $x \rightarrow$ input of dimension $H \times W$, iterators i by j
- $w \rightarrow$ filter/kernel of dimension $k_1 \times k_2$, iterators m by n (feature map)
- $w^l_{m,n} \rightarrow$ weight matrix, connects layer l neurons with previous layer
- $b^l \rightarrow$ bias unit at layer l
- $x^l_{i,j} \rightarrow$ convoluted input vector at layer l + bias

$$\boxed{x^l_{i,j} = \sum_m \sum_n w^l_{m,n}\, o^{l-1}_{i+m,j+n} + b^l}$$

- $o^l_{i,j} \rightarrow$ output vector at layer l

$$\boxed{o^l_{i,j} = f(x^l_{i,j})}$$

- $f(\cdot) \rightarrow$ activation function ($f(x^l_{i,j})$)

Let's do some math!

Gradient component for individual weights

$$\frac{dE}{dw^l_{m',n'}} = \sum_{i=0}^{H-k_1} \sum_{j=0}^{W-k_2} \frac{dE}{dx^l_{i,j}} \frac{dx^l_{i,j}}{dw^l_{m',n'}}$$

$$= \sum_{i=0}^{H-k_1} \sum_{j=0}^{W-k_2} \delta^l_{i,j} \boxed{\frac{dx^l_{i,j}}{dw^l_{m',n'}}}$$

$$\frac{dx^l_{i,j}}{dw^l_{m',n'}} = \frac{d}{dw^l_{m',n'}} \left(\sum_m \sum_n w^l_{m,n}\, o^{l-1}_{i+m,j+n} + b^l \right)$$

$$= \frac{d}{dw^l_{m',n'}} \left(w^l_{m',n'}\, o^{l-1}_{i+m',j+n'} \right)$$

$$= \boxed{o^{l-1}_{i+m',j+n'}}$$

$$= \sum_{i=0}^{H-k_1} \sum_{j=0}^{W-k_2} \delta^l_{i,j}\, o^{l-1}_{i+m',j+n'}$$

$$= \mathrm{rot}_{180^\circ}\{\delta^l_{i,j}\} \times o^{l-1}_{m',n}$$