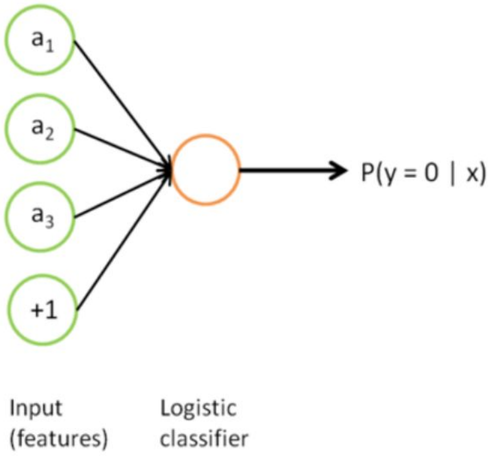
Deep learning in Medical Image Analysis

Lecture 5

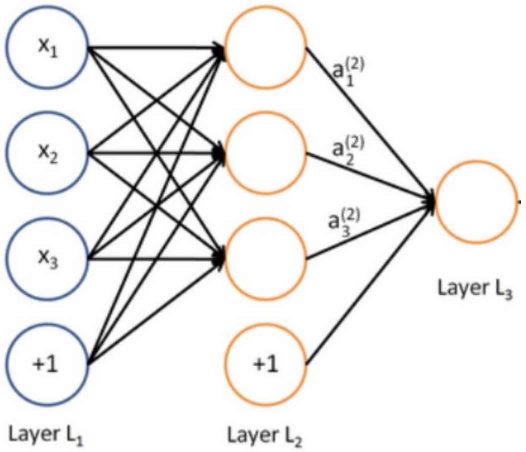
Semen Kiselev

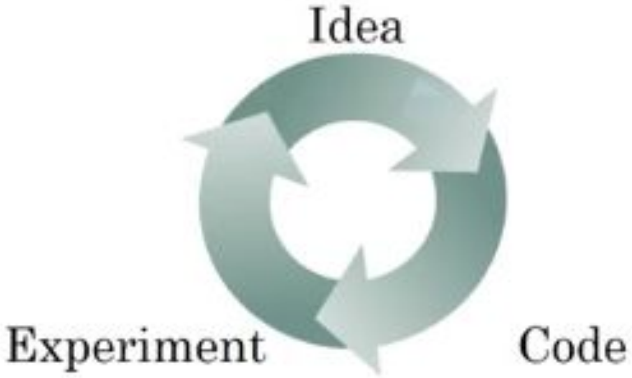
Recap

	Supervised Learning	Unsupervised Learning
Discrete	classification or categorization	clustering
Continuous	regression	dimensionality reduction



Logistic Regression





Classifier	Precision	Recall
A	95%	90%
B	98%	85%

Predicted Class

		Positive	Negative	
Actual Class	Positive	True Positive (TP)	False Negative (FN) **Type II Error**	**Sensitivity** $\frac{TP}{(TP+FN)}$
	Negative	False Positive (FP) **Type I Error**	True Negative (TN)	**Specificity** $\frac{TN}{(TN+FP)}$
		Precision $\frac{TP}{(TP+FP)}$	**Negative Predictive Value** $\frac{TN}{(TN+FN)}$	**Accuracy** $\frac{TP+TN}{(TP+TN+FP+FN)}$

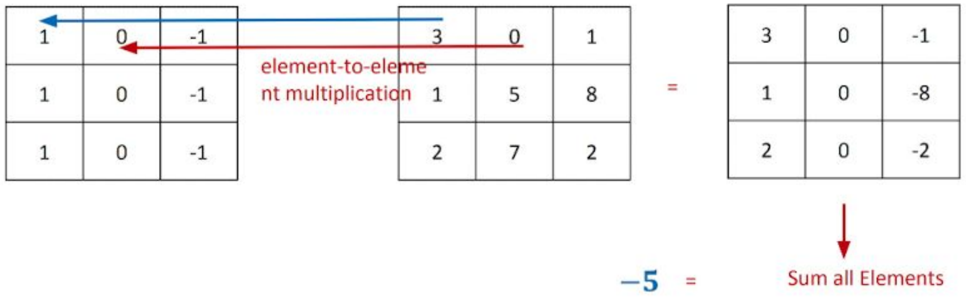
Agenda

1. Convolution
2. Kernels
3. Convolution
4. Max-pooling
5. Flatten
6. CNNs
7. Train loop (optional)

Convolution

“In terms of deep learning, (**image**) **convolution** is an **element-wise multiplication** of two matrices **followed by a sum**”

i. Take **two matrices** (both have the same dimensions).
ii. **Multiply** them, element-by-element (i.e., not the dot product, simple **element-to-element** multiplication).
iii. **Sum** the elements of the resultant Matrix.



The matrix by which the input matrix is multiplied is called **kernel** or **filter**.

Convolution to 2D matrix

Applying convolution **to** a gray-scale image (**2D matrix**)

i. Applying the filter to the matrix of size (***in_d, in_d***) is done using a **sliding window** approach.

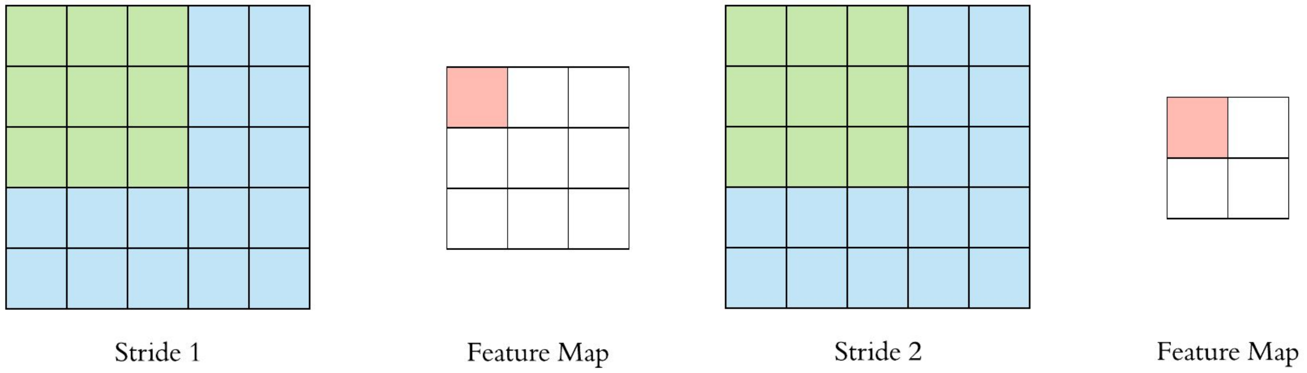
$$\begin{bmatrix}7&2&3&3&8\\4&5&3&8&4\\3&3&2&8&4\\2&8&7&2&7\\5&4&4&5&4\end{bmatrix} * \begin{bmatrix}1&0&-1\\1&0&-1\\1&0&-1\end{bmatrix} = \begin{bmatrix}6& & \\ & & \\ & & \end{bmatrix}$$

7x1+4x1+3x1+
2x0+5x0+3x0+
3x-1+3x-1+2x-1
= 6

We will denote the dimension of the output matrix by ***out_d***. In this example: *in_d = 5*, *out_d = 3*.

Convolution hyper-parameters

i. **Filter size** (f): usually is chosen to be odd in order to have a well-defined origin (center) of filter
ii. **Stride** (s): defines the number of items (pixels), the convolution **window** is **shifted** on each step



iii. **Padding** (p): **after** applying **convolution**, the resulting matrix is shrunk **(out_d < in_d)**. The padding is used in order **to preserve the size of the input matrix**. Essentially, padding an image means **adding some number of rows on each side of an image**.

Constant padding

Fill additional rows with some constant value.

i. Zero padding with *p = 1, s = 1, f = 3*

0	0	0	0	0	0
0	105	102	100	97	96
0	103	99	103	101	102
0	101	98	104	102	100
0	99	101	106	104	99
0	104	104	104	100	98

Image Matrix

Kernel Matrix

0	-1	0
-1	5	-1
0	-1	0

320				

Output Matrix

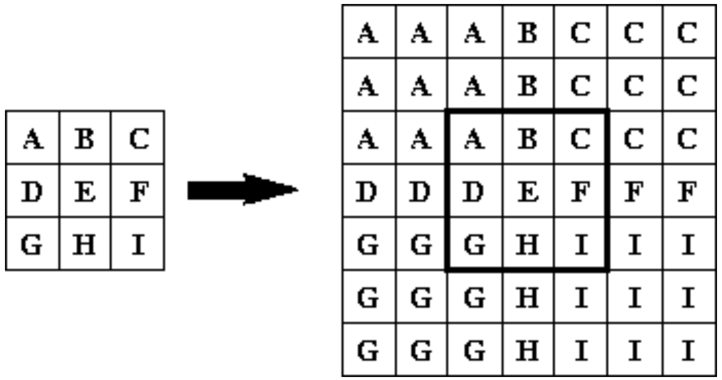
$$\begin{aligned}&0*0+0*-1+0*0\\&+0*-1+105*5+102*-1\\&+0*0+103*-1+99*0 \quad = 320\end{aligned}$$

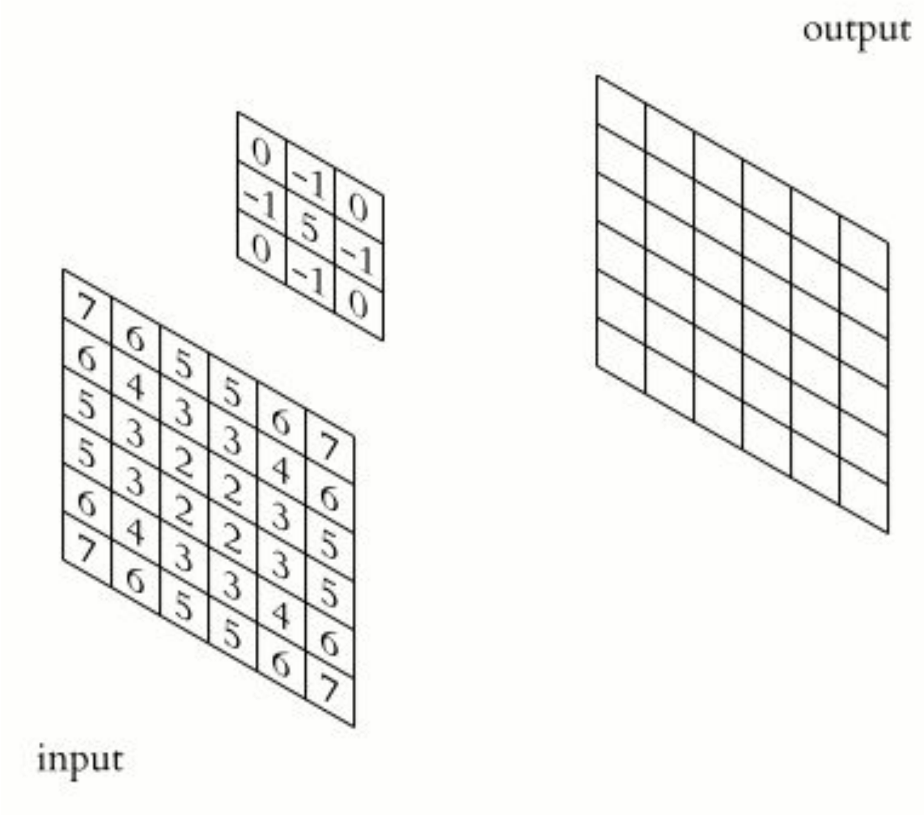
Convolution with horizontal and vertical strides = 1

Replicative padding

Duplicate border pixels *p* times.

i. Replicative padding with *p = 1, s = 1, f = 3*





Valid vs. Same convolution

i. **Valid** convolution: **no padding** of input matrix (p=0)

i. **Same** convolution: apply the padding of the size **enough to keep the dimensions** of the output matrix the same as the dimensions of the input matrix (***p = f // 2; in_d = out_d***)

TASK:

1. given *in_d = 16, s = 1. f = 4* compute the dimensions of output matrix after application of valid convolution
2. given *in_d = 24, s = 2, f = 5* compute the dimensions of output matrix after application of same convolution

Valid vs. Same convolution

i. **Valid** convolution: **no padding** of input matrix (p=0)

i. **Same** convolution: apply the padding of the size **enough to keep the dimensions** of the output matrix the same as the dimensions of the input matrix (***p = f // 2; in_d = out_d***)

In general you can rely on this formula to compute the output matrix dimensions.

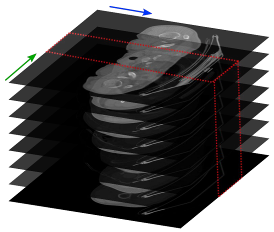
out_d = (in_d + 2*p - f) // s + 1

TASK:

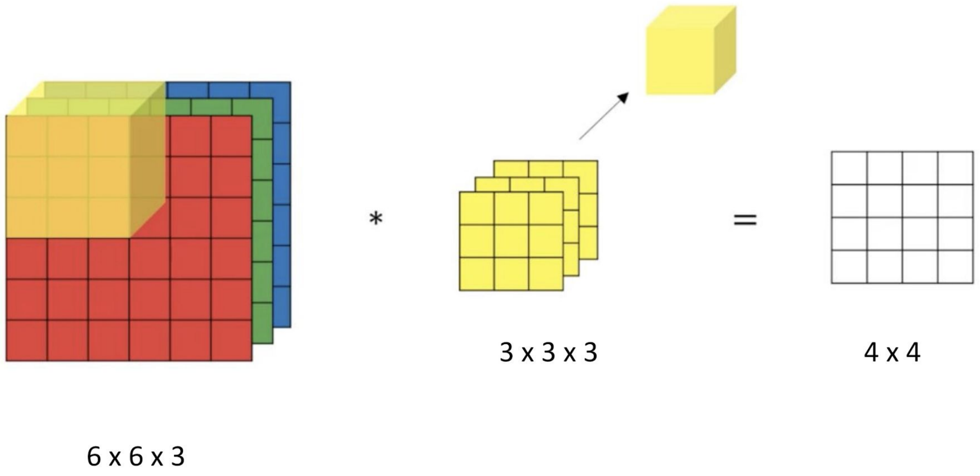
1. given *in_d = 16, s = 1. f = 3* compute the dimensions of output matrix after application of valid convolution = 14
2. given *in_d = 24, s = 2, f = 5* compute the dimensions of output matrix after application of same convolution = 12

Convolution to 3D matrix

Applying convolution **to** a **3D matrix.** (e.g. RGB image , stacked CT slices)

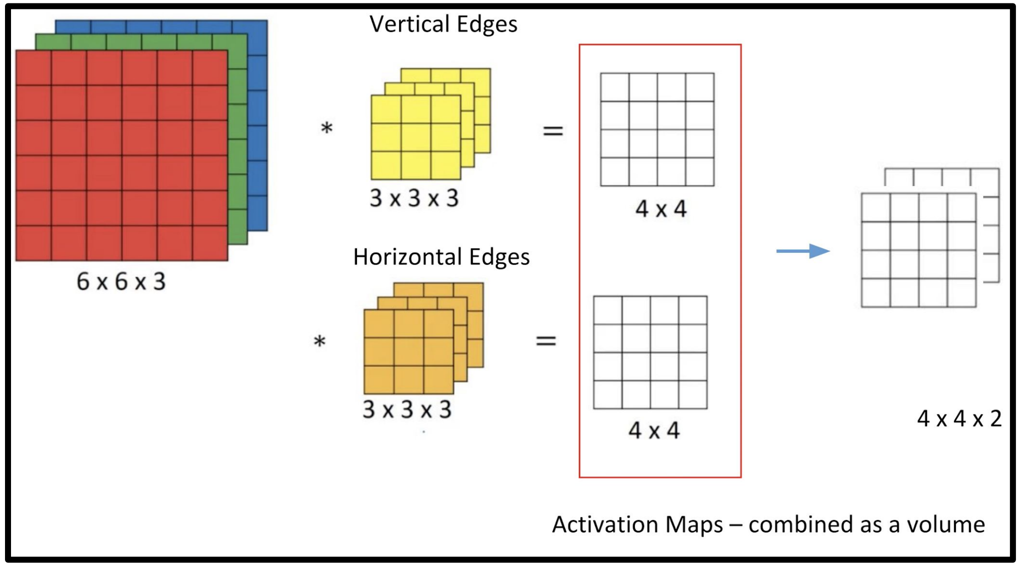


i. Applying the filter to the matrix of size (***in_d, in_d, ch***) is done using a sliding window approach using **3D filters** of shape (**f, f, ch**)

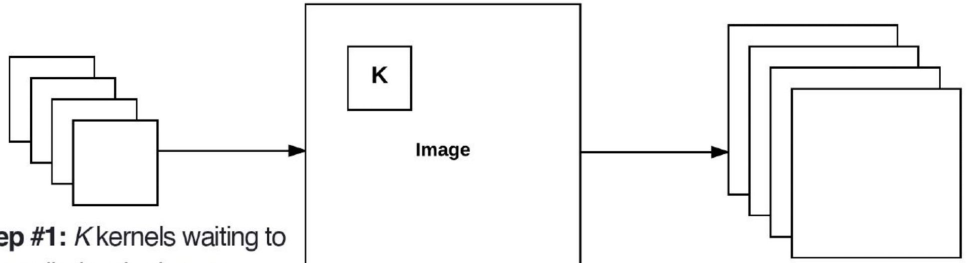


You multiply $f \times f \times ch$ values, and then add the result of all multiplications

Application of 2 different filters to the same input volume.



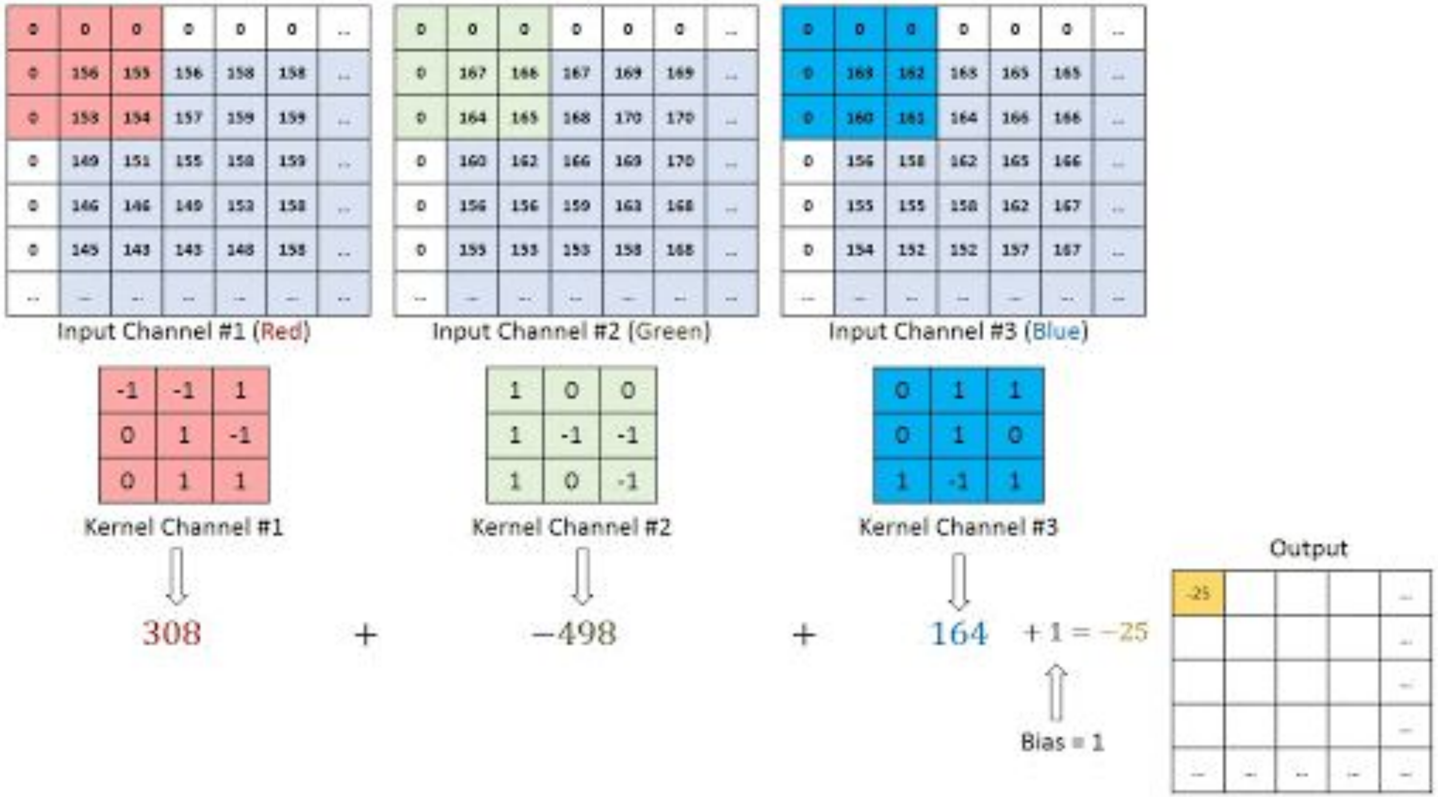
Convolution to 3D matrix (optional)



Step #1: *K* kernels waiting to be applied to the image.

Step #2: Each kernel is convolved with the input volume.

Step #3: The output of each convolution operation produces a 2D output, called an "activation map".



Kernel applications: Edge detection

$$\boxed{\begin{matrix} -1 & 0 & 1 \\ -1 & 0 & 1 \\ -1 & 0 & 1 \end{matrix}}$$

Vertical

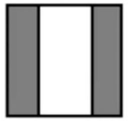
$$\begin{bmatrix} 1 & 0 & -1 \\ 1 & 0 & -1 \\ 1 & 0 & -1 \end{bmatrix}$$

Vertical

$$\begin{bmatrix} 1 & 1 & 1 \\ 0 & 0 & 0 \\ -1 & -1 & -1 \end{bmatrix}$$

Horizontal

$$\begin{bmatrix} 10 & 10 & 10 & 0 & 0 & 0 \\ 10 & 10 & 10 & 0 & 0 & 0 \\ 10 & 10 & 10 & 0 & 0 & 0 \\ 10 & 10 & 10 & 0 & 0 & 0 \\ 10 & 10 & 10 & 0 & 0 & 0 \\ 10 & 10 & 10 & 0 & 0 & 0 \end{bmatrix} * \begin{bmatrix} 1 & 0 & -1 \\ 1 & 0 & -1 \\ 1 & 0 & -1 \end{bmatrix} = \begin{bmatrix} 0 & 30 & 30 & 0 \\ 0 & 30 & 30 & 0 \\ 0 & 30 & 30 & 0 \\ 0 & 30 & 30 & 0 \end{bmatrix}$$

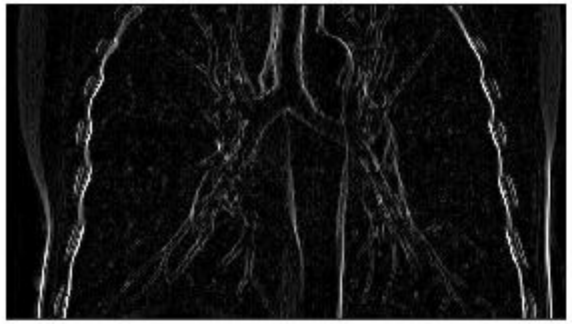


Original

Vertical edges

Original

Vertical edges



What is wrong with this pictures?

Why edges on the one side are better detected than on the other side?

Kernel applications: Smoothing & Sharpening

1	1	1
1	1	1
1	1	1

Unweighted 3x3 smoothing kernel

0	1	0
1	4	1
0	1	0

Weighted 3x3 smoothing kernel with Gaussian blur

0	-1	0
-1	5	-1
0	-1	0

Kernel to make image sharper

-1	-1	-1
-1	9	-1
-1	-1	-1

Intensified sharper image

Gaussian Blur

Sharpened image

Kernel applications: Feature Extractor

Kernel as a feature extractor ↓

i. **Edges represent the boundary** of an object in an image

ii. We can use them to **identify the objects**: face, car, street signs, etc

iii. Thus, an **image containing edges** can be thought of **as feature map** (i.e.input values to our model)

iv. In general: **Kernels** (or filters) and **Convolution** can **help** us **find features** in a given input.

v. **Thus** Kernels (or filters) can be thought of as **feature detectors**.

Idea: can we learn kernels from the data instead of hand-designing them?

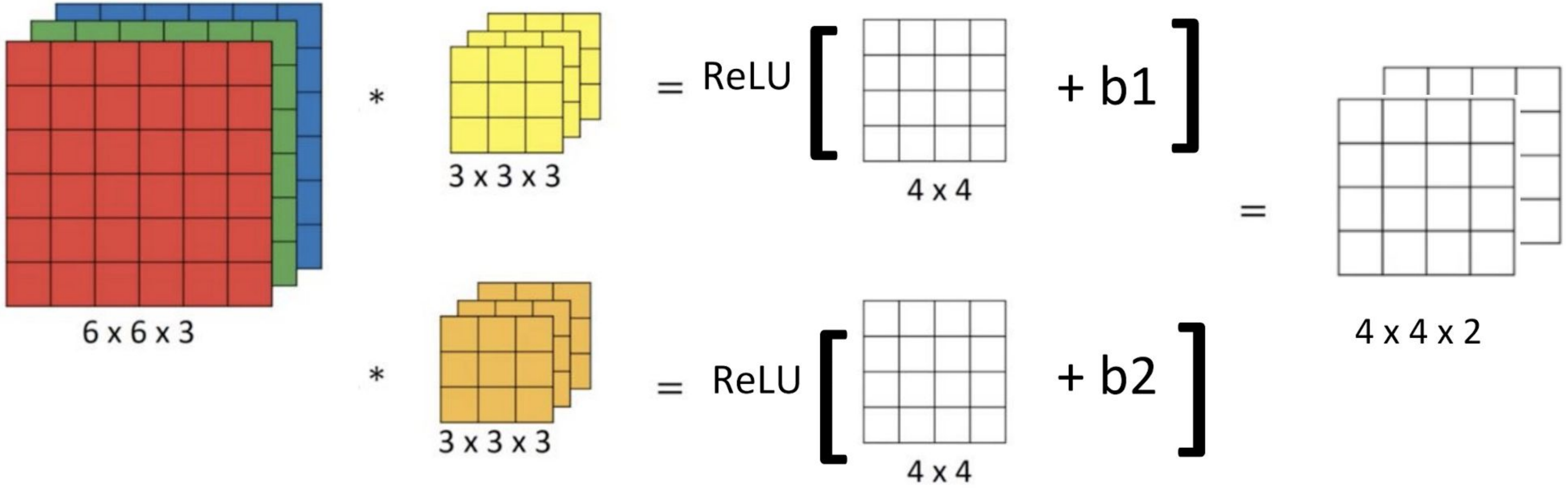
Kernel: learning

i. In deep-learning, **filters are** represented by the **parameters** which we want **to optimize**.
ii. In other words, we **learn** those **filters that** help us discover features that **improve the task** (e.g. classification).

3	0	1	2	7	4
1	5	8	9	3	1
2	7	2	5	1	3
0	1	3	1	7	8
4	2	1	6	2	8
2	4	5	2	3	9

w_1	w_2	w_3
w_4	w_5	w_6
w_7	w_8	w_9

Convolution layer



ReLU is just for example, you can apply any other activation function

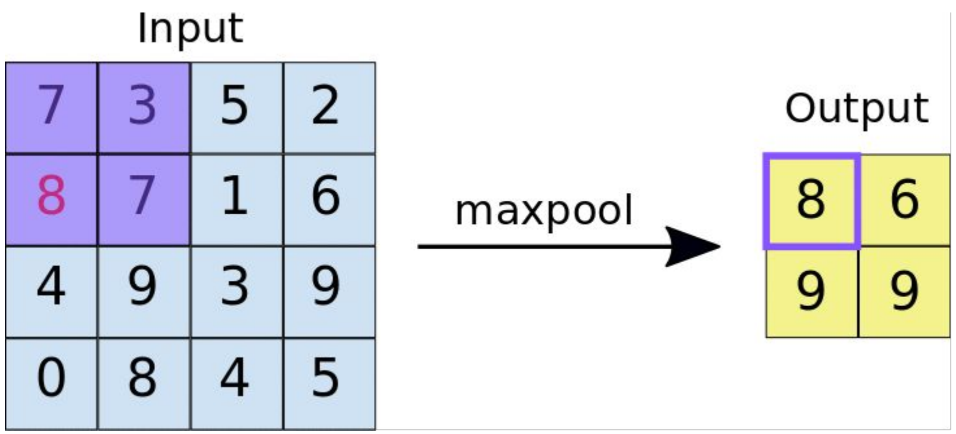
Pooling layer

Processes a **region of size (f, f)** and reduces it **to** a **single value**.

a. **Max-pooling**: reduces to the maximum value in that region

b. **Min-pooling**: reduces to the minimum value in that region

c. **Average-pooling**: reduces to the mean of the values in that region



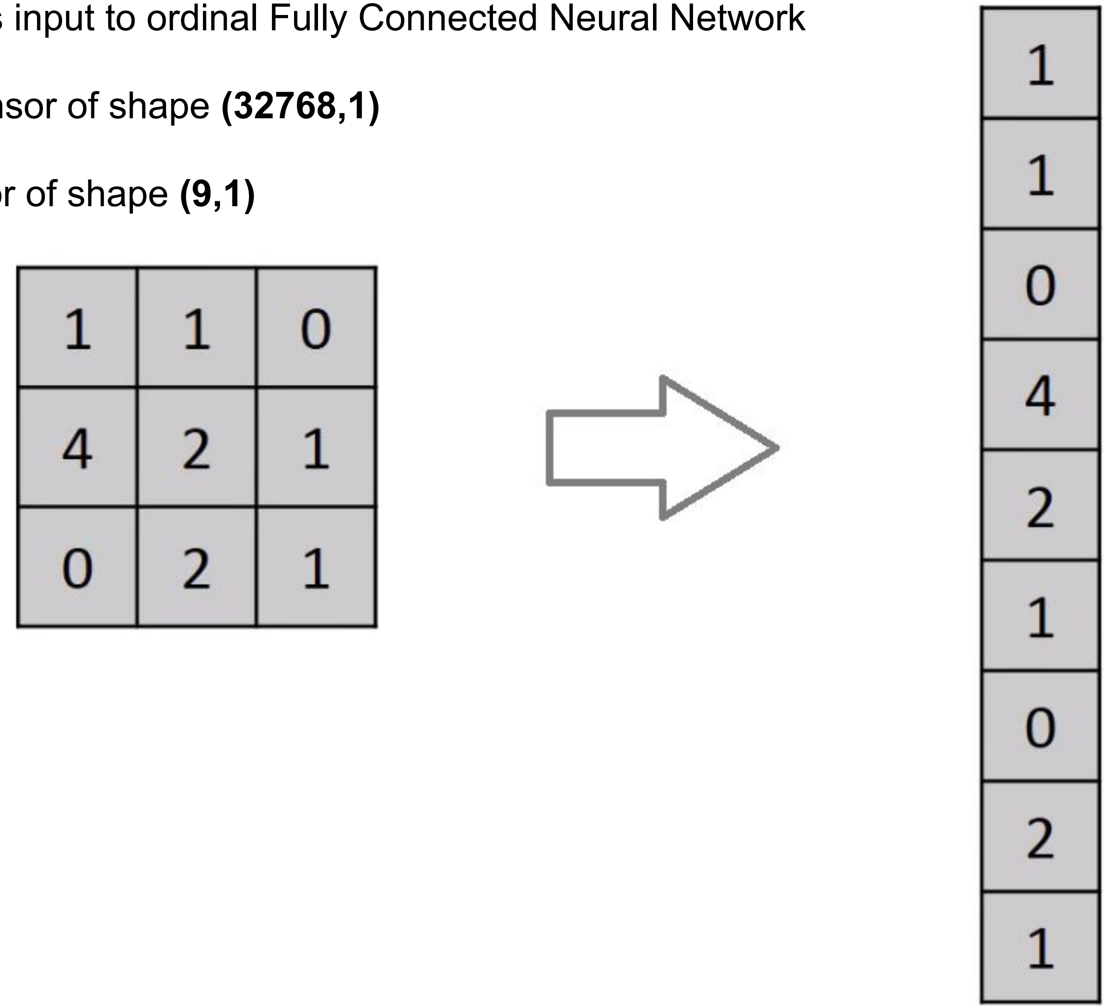
- It is also parametrized by **stride**. Usually, the stride is set to be equal to the size of the polling filter
- Pooling is applied for each 2D activation map separately **!**

Flatten layer

Flatten volume/tensor of feature maps and use it as input to ordinal Fully Connected Neural Network

a. Tensor of shape **(64,64,8) becomes** tensor of shape **(32768,1)**

b. Tensor of shape **(3,3,1) becomes** tensor of shape **(9,1)**



Convolutional Neural Networks (CNNs)

CNNs are constructed using the following layers

i. **Convolutional**

ii. **Pooling**

iii. **Fully connected**

TASK:

Given

1. Input volume of shape (228, 228, **228**)
2. **32 filters** of size **(3,3)**

Compute the number of parameters in this Conv layer

Convolutional Neural Networks (CNNs)

CNNs are constructed using the following layers

i. **Convolutional**

ii. **Pooling**

iii. **Fully connected**

TASK:

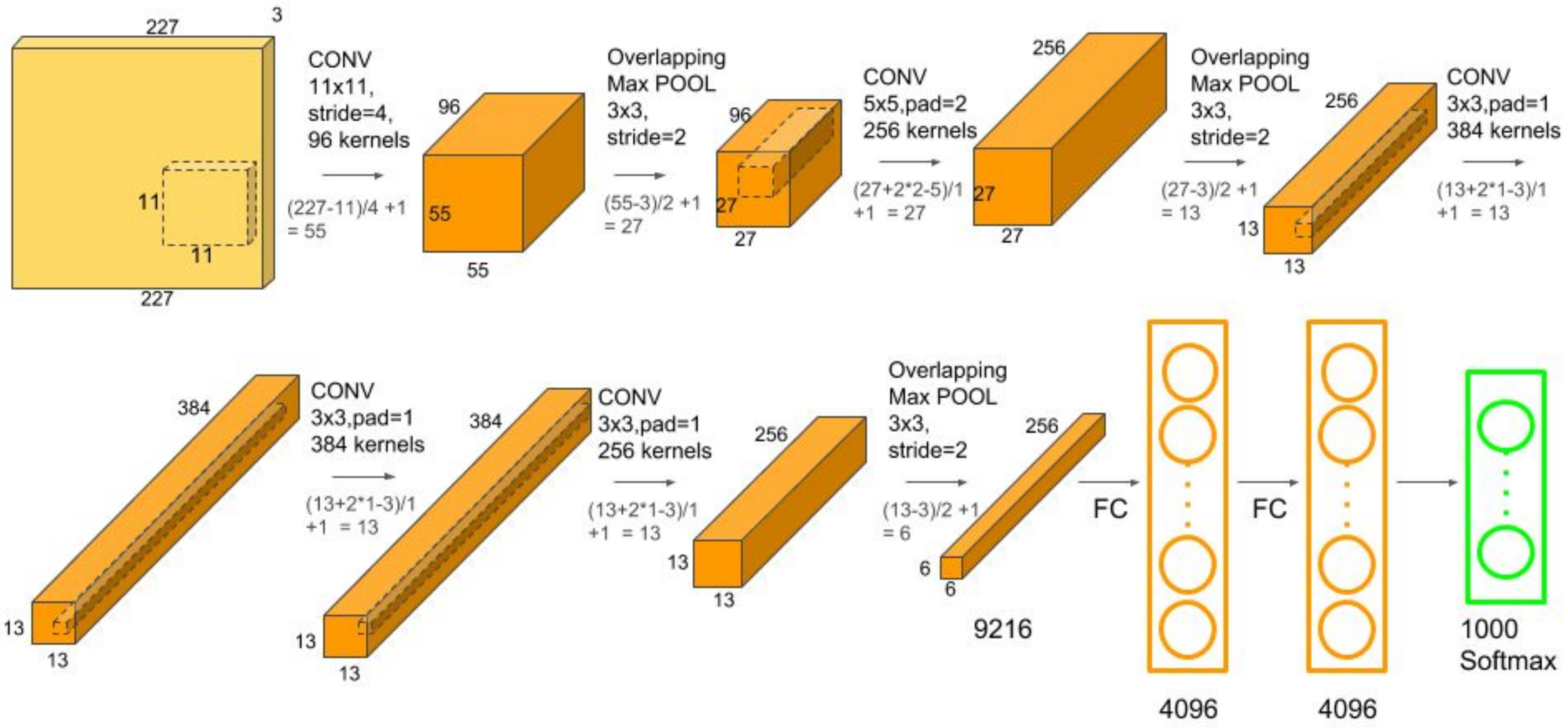
Given

1. Input volume of shape (228, 228, **228**)
2. **32 filters** of size **(3,3)**

Compute the number of parameters in this Conv layer *= [(3 * 3 * 228) + 1] * 32 = 65696*

Architectures: AlexNet

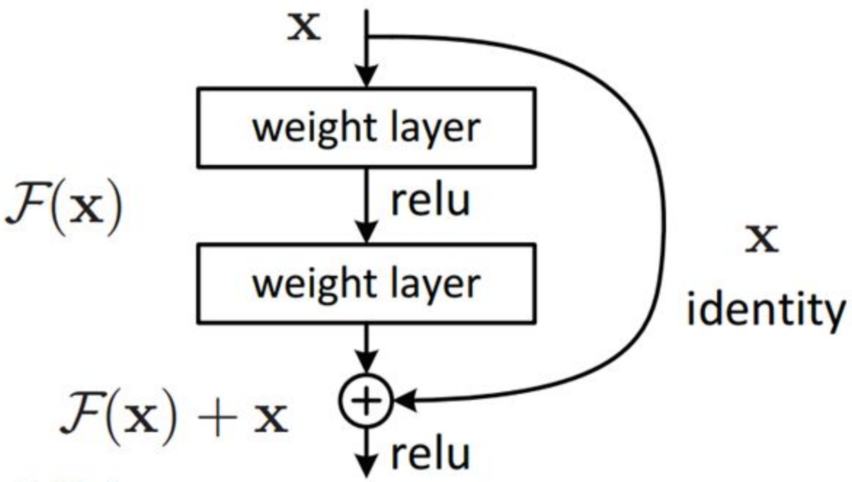
Basic conv net ever



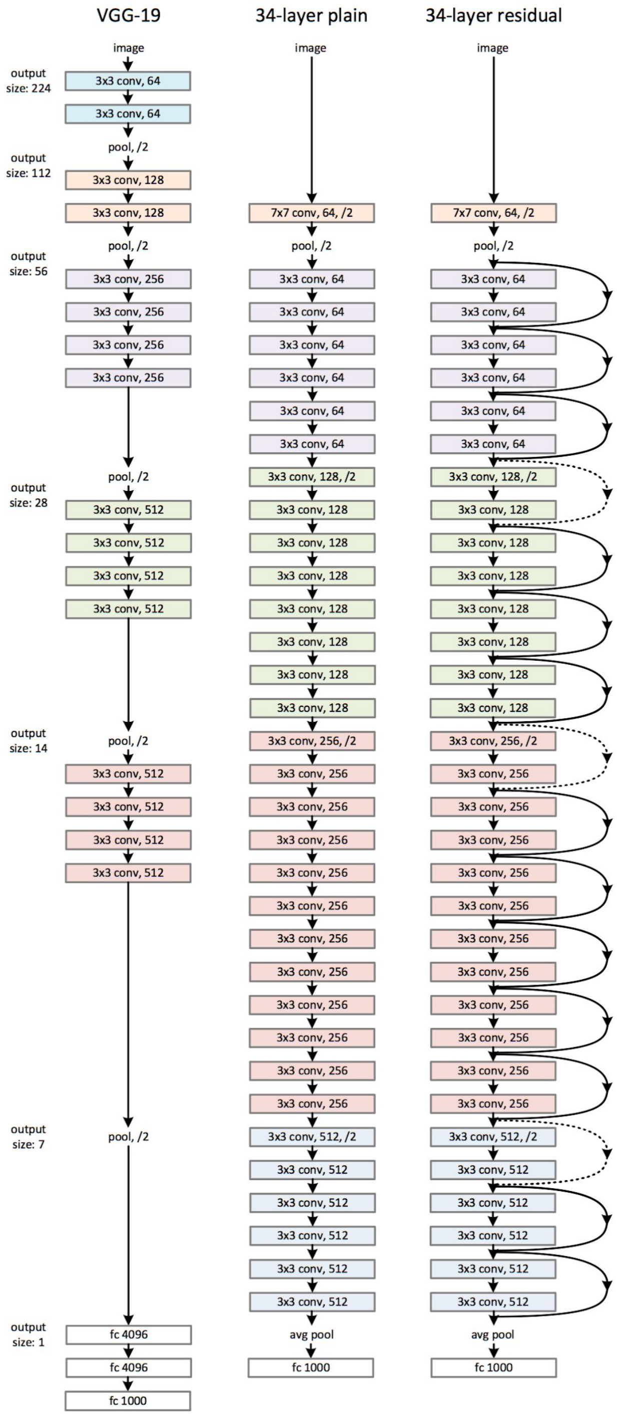
Architectures: ResNet

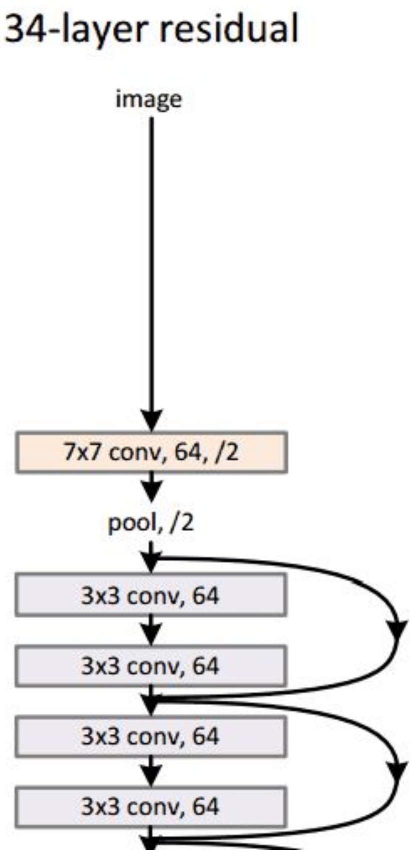
Authors proposed the idea of **residual connections**,
which allowed to train Ultra-deep Conv nets (hundreds of layers)

1. Residual block



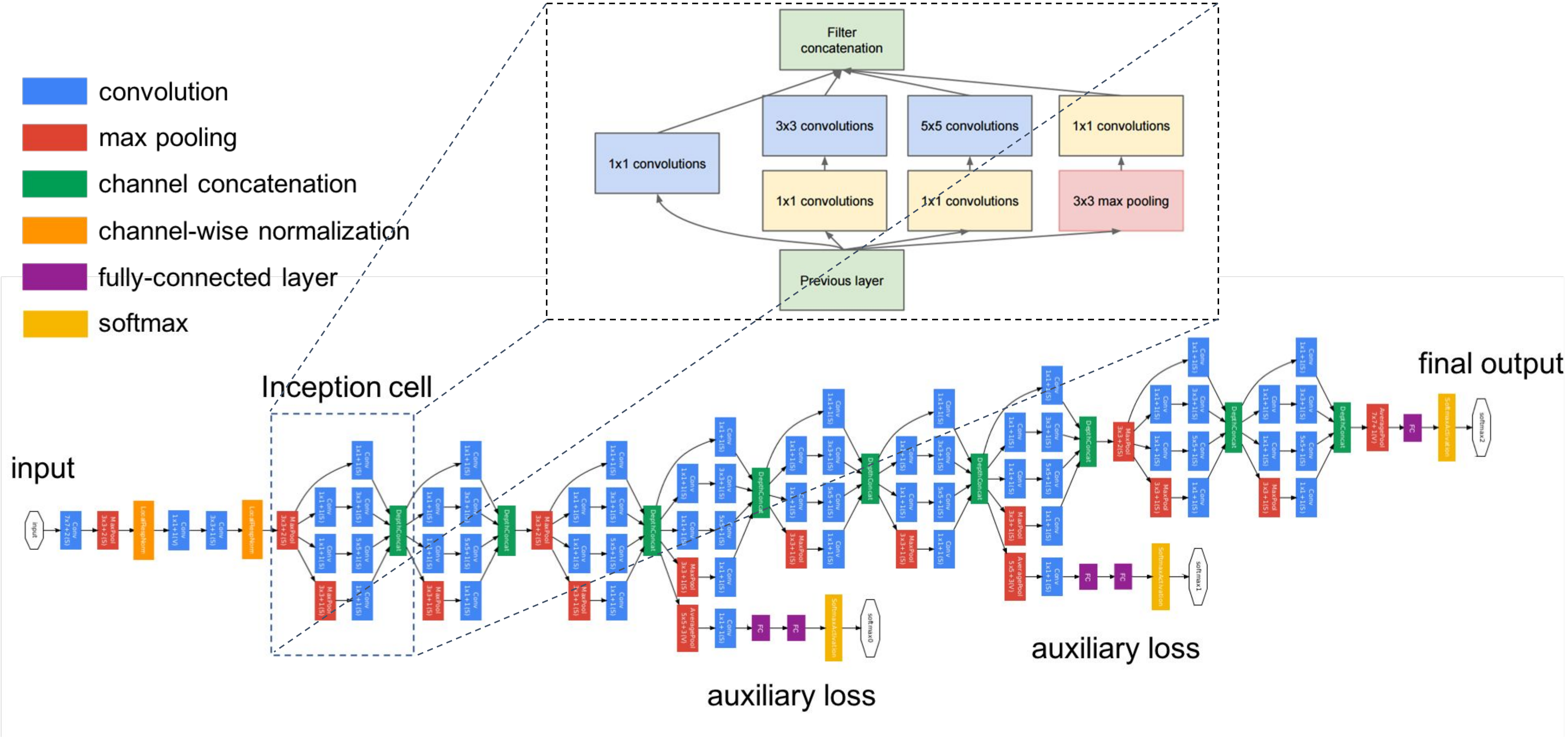
A residual block





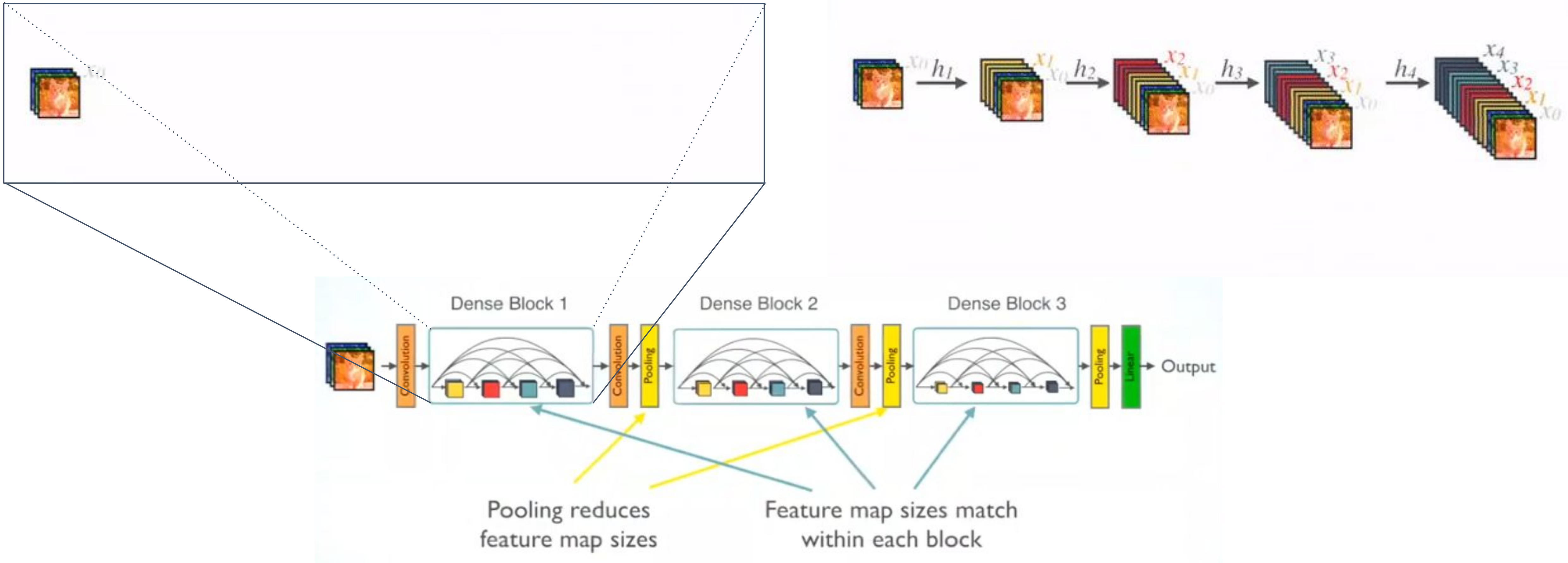
Architectures: Inception

Authors proposed idea of applying **several** convolution **filters of different size** in one layer, which allowed for better feature extraction



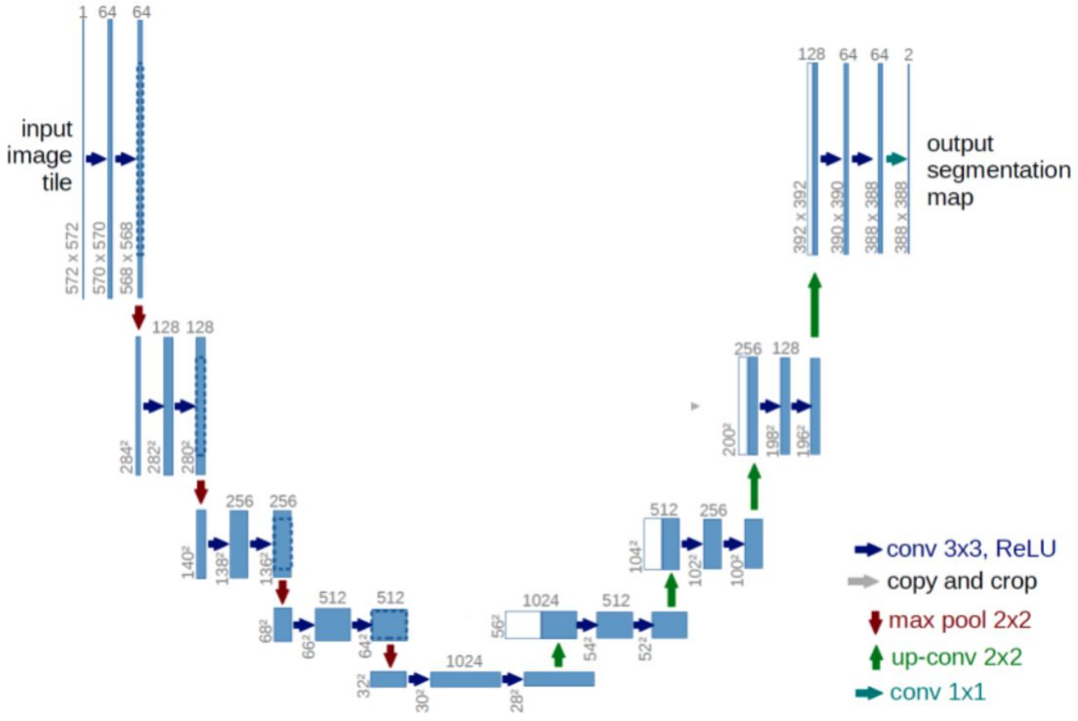
Architectures: DenseNet

Authors proposed idea of **densely connected blocks**,
i.e. **concatenation of results** of all previous layers inside block with current layer result.

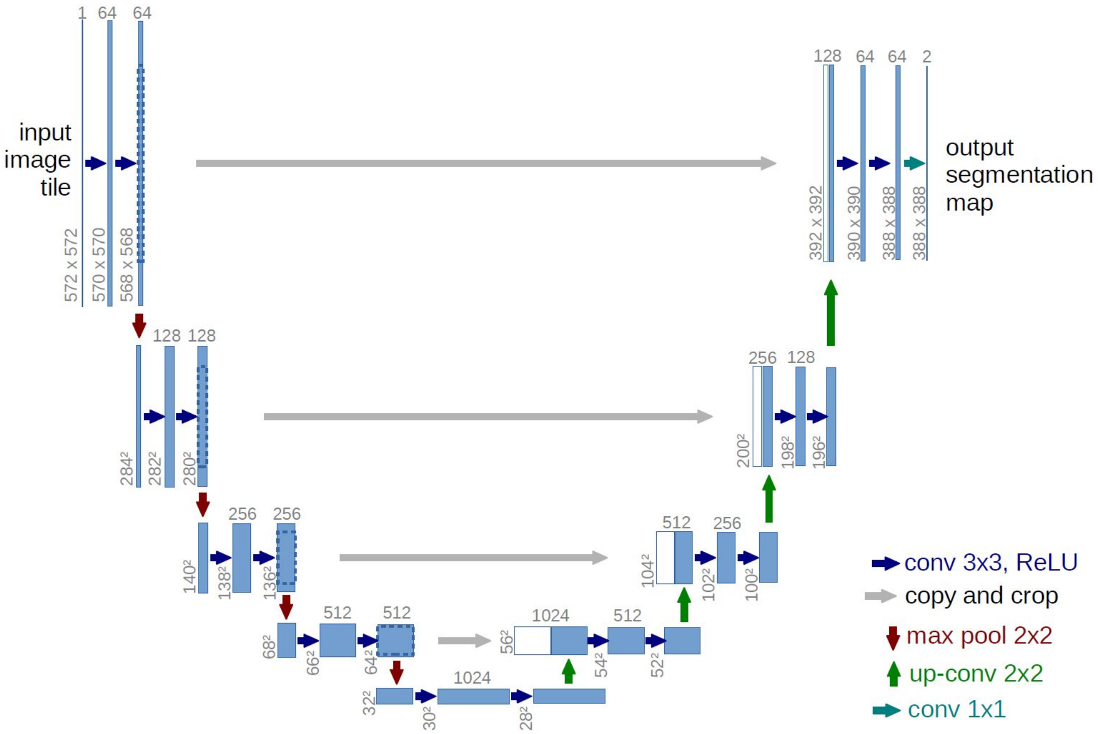


Architectures: Unet

Based on the **autoencoder** concept.
Authors proposed an idea of **skip-connections**, which allowed for better reconstruction of input image features.



Autoencoder



Unet

PyTorch: train loop (Optional)

```
for batch_idx, batch_data in enumerate(train_loader):
        data, target = batch_data["image"], batch_data["target"]
        data, target = data.to(device), target.to(device)
        output = model(data)

        # ------------------------Optimize-------------------------
        optimizer.zero_grad()
        loss = criterion(output, target)     # objective function
        loss.backward()                      # compute partial derivatives wrt. each parameter of the model
        optimizer.step()                     # update each model parameter using gradient descent

        train_loss += loss.item()            # sum up batch losses
    # end of epoch -------------------------------------------------------------
    train_loss /= len(train_loader)

    if isinstance(scheduler, torch.optim.lr_scheduler.StepLR):
        scheduler.step()                     # update learning rate according to some predefined algorithm
```

Спасибо за внимание

INNOPOLIS UNIVERSITY