



## Artificial Intelligence II: Deep learning methods

**Dragoș Burileanu, Ana Neacșu & Vlad Vasilescu, Georgian Nicolae**  
**Lecture 4: Convolutional Neural Networks**

National University of Science and Technology POLITEHNICA Bucharest, Romania  
BIOSINF Master Program

March 2024

Introduction  
oooooooooooo

10 years of CNN revolution  
ooooooooooooooo

Design principles of CNNs  
oooooooooooooooo

We need data, but we don't have!  
oooooooooooo

# Overview

## 1 Introduction

## 2 10 years of CNN revolution

## 3 Design principles of CNNs

## 4 We need data, but we don't have!

Introduction  
●oooooooooooo

10 years of CNN revolution  
ooooooooooooooo

Design principles of CNNs  
ooooooooooooooo

We need data, but we don't have!  
oooooooooooo

## Introduction

# Why Convolutions?

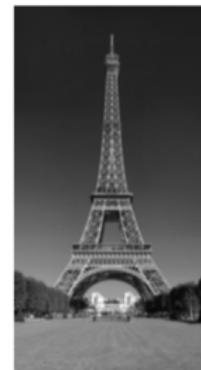
- Convolutions can be used to extract features on data that have a **spatial structure** (locally correlated), e.g. images.
- They can also be used for **temporal series** that have a structure in time.



Original image



Discrete laplacian

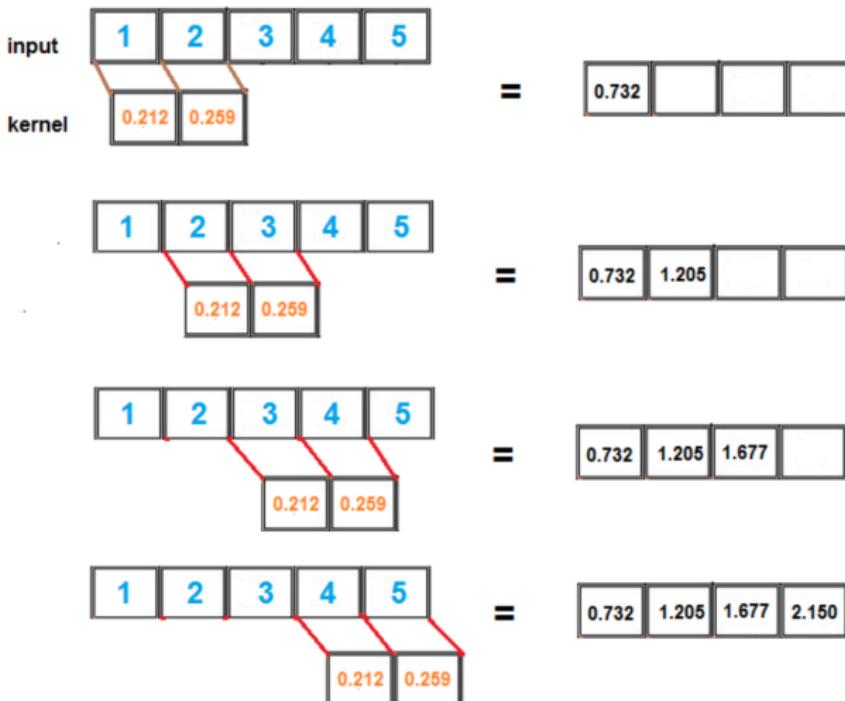


Gaussian blur



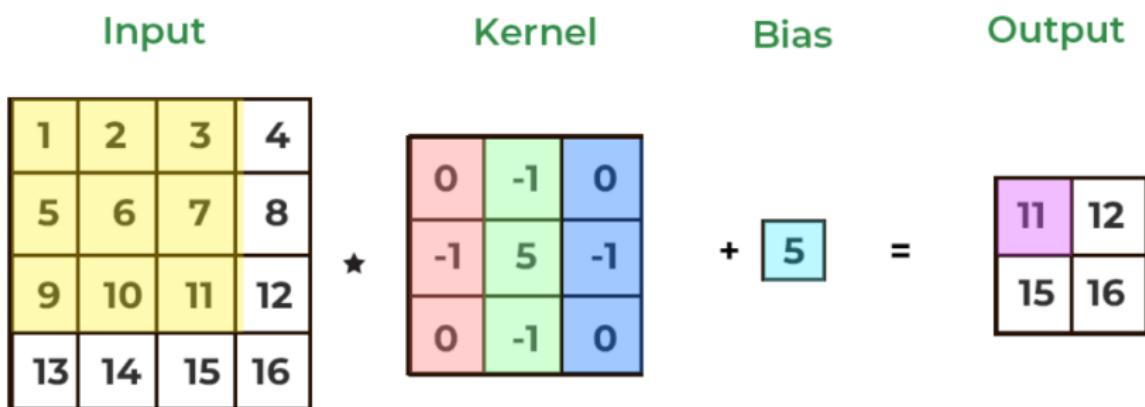
Pattern matching

# Convolution layers – 1D



# Convolution layers – 2D

- Check [this guide](#) for info about how to compute output size.



# Convolution as a sparse matrix multiply

- A convolution can be seen as a matrix multiplication with a Toeplitz-like structure
- Given two 1D vectors: input  $f$  and the kernel  $k = [a, b, c]$ , then:

$$(f * k) = \begin{bmatrix} b & c & 0 & 0 & \cdots & 0 & 0 \\ a & b & c & 0 & \cdots & 0 & 0 \\ 0 & a & b & c & \cdots & 0 & 0 \\ 0 & 0 & a & b & \cdots & 0 & 0 \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ 0 & 0 & 0 & 0 & \cdots & b & c \\ 0 & 0 & 0 & 0 & \cdots & a & b \end{bmatrix} \cdot \begin{bmatrix} f \\ \vdots \\ f \end{bmatrix}$$

- This result can be extended to 2D case

## Learning higher level features

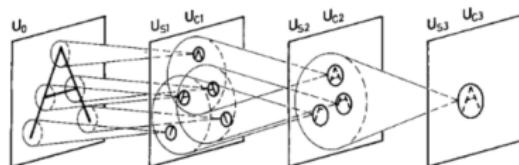
**Idea** : Local features can be combined to learn higher level features.

Example: Build a house detector

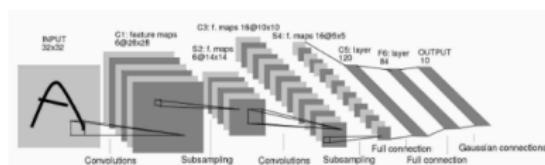
# Architectural concepts

**Main Idea**: Using the structure of the inputs to limit the number of parameters without limiting the expressiveness of the network.

- For inputs with spatial (or temporal) correlations, features can be extracted with convolutions of local kernels.
  - A convolution can be seen as a fully connected layer with :
    - a lot of weights set to 0
    - a lot of weights shared across positions
- strongly regularized!



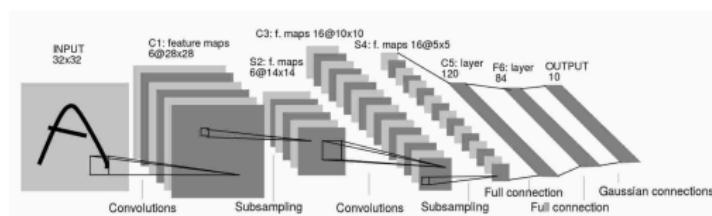
Neocognitron (Fukushima, 1980)



LeNet5 (LeCun et al., 1989)

# LeNet5 architecture

- The architecture of LeNet-5, proposed by LeCun in 1989.



LeNet5 (LeCun et al., 1989)

## Architecture :

Two main parts:

- Convolutional part
- Fully connected part

Specificities:

- Weighted sub-sampling
- Gaussian connections (RBF output layer)
- connectivity pattern ( $S_2 - C_3$ ) to reduce the number of weights

## Number of parameters :

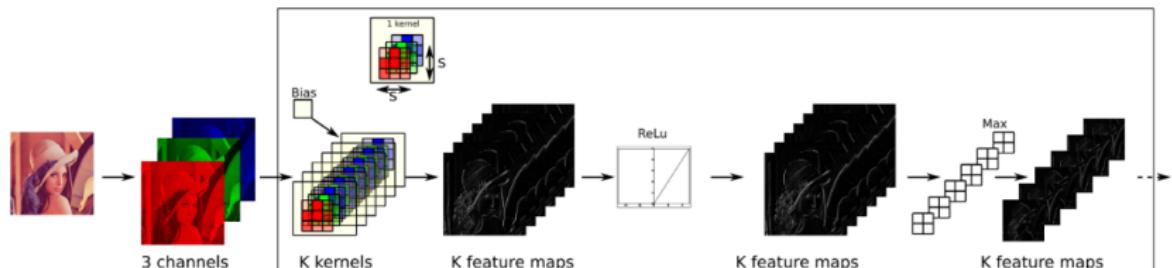
<b>C1</b>	<b>156</b>
<b>S2</b>	<b>12</b>
<b>C3</b>	<b>1.516</b>
<b>S4</b>	<b>32</b>
<b>C5</b>	<b>48.120</b>
<b>F6</b>	<b>10.164</b>

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	X			X	X	X		X	X	X	X	X	X	X	X	X
1	X	X			X	X	X		X	X	X	X	X	X	X	
2	X	X	X			X	X	X		X		X	X	X	X	
3	X	X	X		X	X	X	X		X	X	X	X	X	X	
4	X	X	X		X	X	X	X	X	X	X	X	X	X	X	
5	X	X	X		X	X	X	X	X	X	X	X	X	X	X	

TABLE I

EACH COLUMN INDICATES WHICH FEATURE MAP IN S2 ARE COMBINED BY THE UNITS IN A PARTICULAR FEATURE MAP OF C3.

# CNN Definitions



The building blocks of the convolutional part of a vanilla CNN

## Convolution:

- size (e.g.  $3 \times 3$ ,  $5 \times 5$ )
- padding (e.g. 1, 2)
- stride (e.g. 1)

## Pooling (max/average):

- size (e.g.  $2 \times 2$ )
- padding (e.g. 0)
- stride (e.g. 2)

We work with 4D tensors for 2D images, 3D tensors for nD temporal series (e.g. multiple simultaneous recordings), 2D tensors for 1D temporal series.

In Pytorch, the tensors follow the Batch-Channel-Height-Width (**BCHW**, channel-first) convention. Other frameworks, like TensorFlow or CNTK, use **BHWC** (channel-last)

# Transpose Convolution

Given two 1D vectors, input  $x_1$  and the kernel  $k = [a, b, c]$ , then:

$$y_1 = (x_1 * k) = \begin{bmatrix} b & c & 0 & 0 & \cdots & 0 & 0 \\ a & b & c & 0 & \cdots & 0 & 0 \\ 0 & a & b & c & \cdots & 0 & 0 \\ 0 & 0 & a & b & \cdots & 0 & 0 \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ 0 & 0 & 0 & 0 & \cdots & b & c \\ 0 & 0 & 0 & 0 & \cdots & a & b \end{bmatrix} \cdot \begin{bmatrix} x_1 \end{bmatrix} = W_k x_1$$

If we compute the gradient of the loss, in denominator layout:

$$\frac{\partial L}{\partial x_1} = \frac{\partial y_1}{\partial x_1} \frac{\partial L}{\partial y_1} = W_k^\top \frac{\partial L}{\partial y_1}$$

herefore, the term **transposed convolution** or **backward convolution** is introduced. This terminology will resurface later in discussions regarding deconvolution.

Introduction  
oooooooooooo

10 years of CNN revolution  
●oooooooooooo

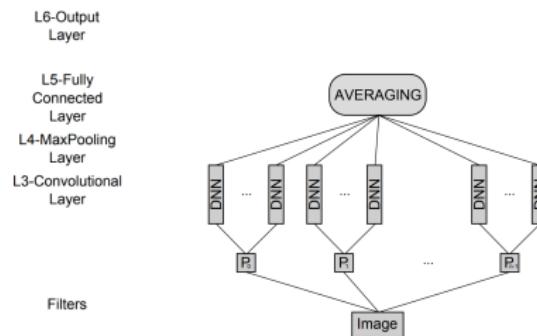
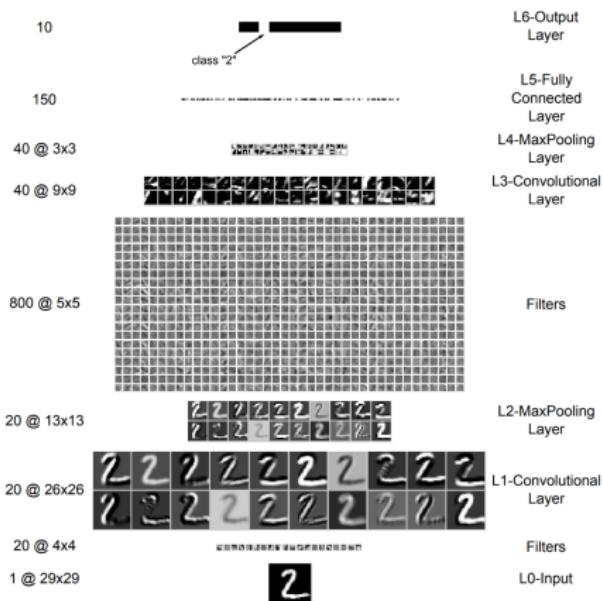
Design principles of CNNs  
oooooooooooo

We need data, but we don't have!  
oooooooooooo

10 years of CNN revolution

# Multi-Column CDNN

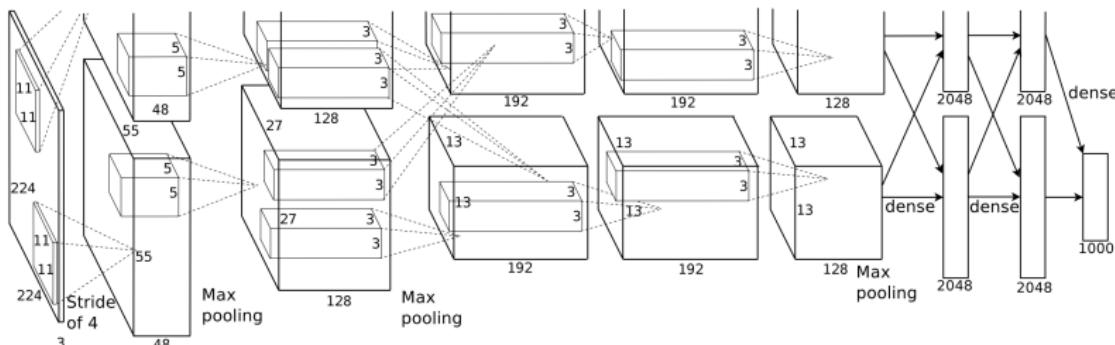
Introduced by [Cireşan et. al.](#) in 2012, first ensemble of CNNs trained with dataset augmentation.



- 1.5 million of parameters/
- 0.23% test misclassification on MNIST.

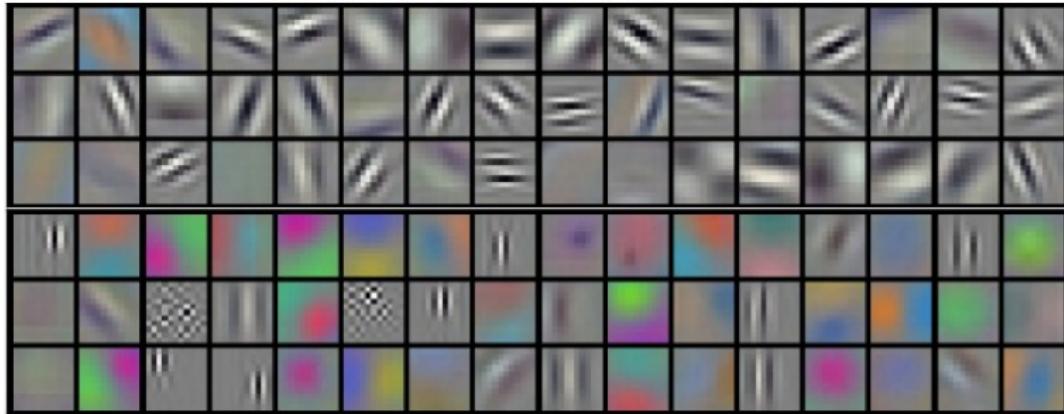
# AlexNet – Supervision

Introduced by [Krizhevsky et. al.](#) in 2012, considered the “spark” giving birth to the revival of neural networks.



- Top 5 error of 16%, runner-up at 26%
- Several convolutions stacked before pooling
- Trained on 2 GPUs, for a week on ImageNet (resized to  $256 \times 256 \times 3$ ), 1M images. (now it's [18 minutes](#)).
- 60 Million parameters, dropout, momentum, L2 penalty, dataset augmentation (rand crop ( $224 \times 224$ ) , translation, reflections, PCA)
- Learning rate 0.001 at divided by 10 when validation error stalls
- At test time, avg probabilities on crops + reflections

# Supervision

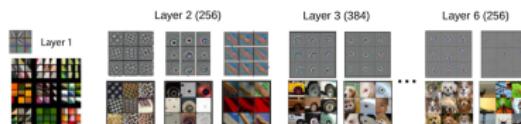


Heatmaps showing how filters from the first layer look like.

## ZFNet

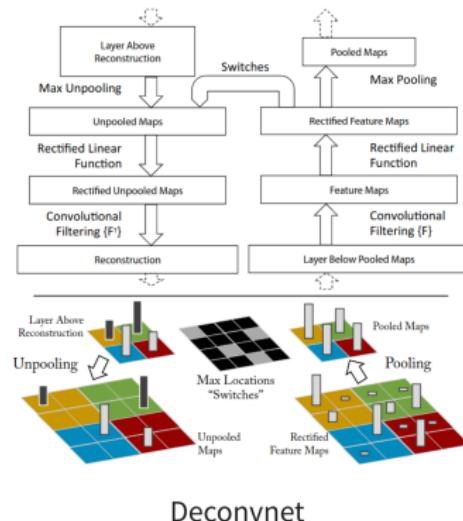
Introduced by [Zeiler and Fergus](#) in 2014, the winner of ILSVRC'13 challenge.

- Introduced visualization techniques to inspect which features are learned.



Inputs computed via deconvolution

- Ablation studies on AlexNet : the FC layers are not that important.
- Introduced the idea of supervised pretraining (pretraining on ImageNet, finetune the softmax for Caltech-101, Caltech-256, Pascal 2012)
- SGD minibatch(128), momentum(0.9), learning rate (0.01) manual schedule.



- Deconvnet computes approximately the gradient of the loss w.r.t. the input. It differs in the way the ReLu is integrated.

# VGG Net

Introduced by [Simonyan and Zisserman](#) in 2015, the winner of ILSVRC'14.

- 16 layers : 13 convolutive, 3 FC
- $3 \times 3$  convolutions,  $2 \times 2$  pooling
- Stacked  $3 \times 3$  convolutions  $\equiv 5 \times 5$  convolution receptive field
  - ✿ If  $c_{in} = K$ ,  $c_{out} = K$ ,  $5 \times 5$  conv.  $\rightarrow 25K^2$  parameters
  - ✿ If  $c_{in} = K$ ,  $c_{out} = K$ , 2 stacked  $3 \times 3$  conv.  $\rightarrow 18K^2$  param.
- 140 million parameters, batch size(256), Momentum(0.9), Weight decay (0.0005), Dropout(0.5) in FC, learning rate (0.01) divided 3 times by 10.
- Initialization of  $B, C, D, E$  from trained  $A$ . Init.  $A$  random from  $\mathcal{N}(0, 10^{-2}, b = 0)$
- can cope with variable input size changing the FC layers to conv  $7 \times 7$ , conv  $1 \times 1$ .

Table 1: ConvNet configurations (shown in columns). The depth of the configurations increases from the left (A) to the right (E), as more layers are added (the added layers are shown in bold). The convolutional layer parameters are denoted as “conv(receptive field size)-(number of channels)”. The ReLU activation function is not shown for brevity.

ConvNet Configuration							
A	A-LRN	B	C	D	E		
11 weight layers							
conv3-64	conv3-64	conv3-64	conv3-64	conv3-64	conv3-64		
	LRN	conv3-64	conv3-64	conv3-64	conv3-64		
		maxpool					
conv3-128	conv3-128	conv3-128	conv3-128	conv3-128	conv3-128		
	conv3-128	conv3-128	conv3-128	conv3-128	conv3-128		
		maxpool					
conv3-256	conv3-256	conv3-256	conv3-256	conv3-256	conv3-256		
conv3-256	conv3-256	conv3-256	conv3-256	conv3-256	conv3-256		
		conv1-256	conv1-256	conv3-256	conv3-256		
				maxpool			
conv3-512	conv3-512	conv3-512	conv3-512	conv3-512	conv3-512		
conv3-512	conv3-512	conv3-512	conv3-512	conv3-512	conv3-512		
		conv1-512	conv1-512	conv3-512	conv3-512		
				maxpool			
conv3-512	conv3-512	conv3-512	conv3-512	conv3-512	conv3-512		
conv3-512	conv3-512	conv3-512	conv3-512	conv3-512	conv3-512		
		conv1-512	conv1-512	conv3-512	conv3-512		
				maxpool			
FC-4096							
FC-4096							
FC-1000							
soft-max							

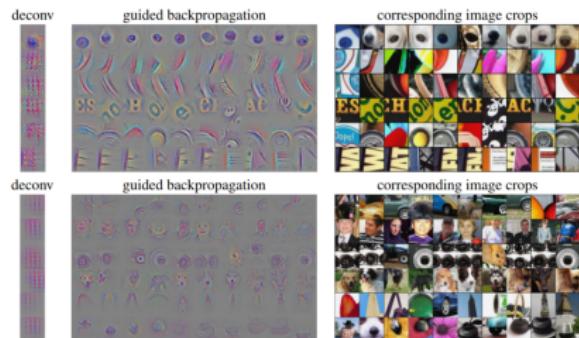
Table 2: Number of parameters (in millions).

Network	A,A-LRN	B	C	D	E
Number of parameters	133	133	134	138	144

# Towards simpler models

Introduced by [Springenberg et. al.](#) in 2015.

- uses only convolutions, with various strides, no max pooling
- introduces “guided backpropagation” visualization



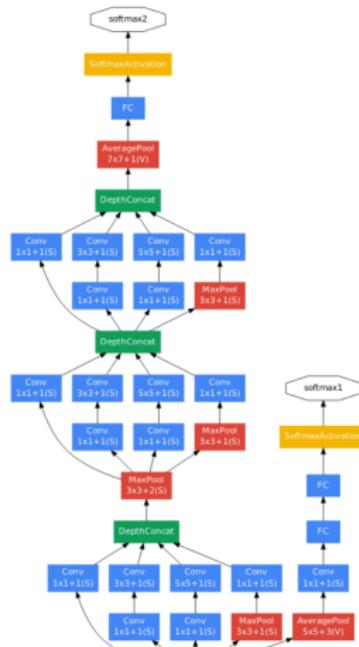
Model		
Strided-CNN-C	ConvPool-CNN-C	All-CNN-C
Input 32 × 32 RGB image		
$3 \times 3$ conv. 96 ReLU	$3 \times 3$ conv. 96 ReLU	$3 \times 3$ conv. 96 ReLU
$3 \times 3$ conv. 96 ReLU with stride $r = 2$	$3 \times 3$ conv. 96 ReLU $3 \times 3$ conv. 96 ReLU $3 \times 3$ conv. 96 ReLU	$3 \times 3$ conv. 96 ReLU $3 \times 3$ conv. 96 ReLU
	$3 \times 3$ max-pooling stride 2	$3 \times 3$ conv. 96 ReLU with stride $r = 2$
$3 \times 3$ conv. 192 ReLU	$3 \times 3$ conv. 192 ReLU	$3 \times 3$ conv. 192 ReLU with stride $r = 2$
$3 \times 3$ conv. 192 ReLU with stride $r = 2$	$3 \times 3$ conv. 192 ReLU $3 \times 3$ conv. 192 ReLU $3 \times 3$ conv. 192 ReLU	$3 \times 3$ conv. 192 ReLU $3 \times 3$ conv. 192 ReLU
	$3 \times 3$ max-pooling stride 2	$3 \times 3$ conv. 192 ReLU with stride $r = 2$

Guided backpropagation examples

# Inception Net

Introduced by [Szegedy et. al.](#) in 2014, first very deep convolutional model that use Multi-scale feature detection and dimensionality reduction

- 22 layers, 6.8M parameters.
- trained in parallel , asynchronous SGD, momentum(0.9), learning rate schedule (4% every 8 epochs)
- at inference : polyack average and ensemble of models
- auxiliary heads to mitigate vanishing gradient.



type	patch size/ stride	output size	depth	#1×1	#3×3 reduce	#3×3	#5×5 reduce	#5×5	pool proj	params	ops
convolution	7×7/2	112×112×64	1							2.7K	34M
max pool	3×3/2	56×56×64	0								
convolution	3×3/1	56×56×192	2		64	192				112K	360M
max pool	3×3/2	28×28×192	0								
inception (3a)		28×28×256	2	64	96	128	16	32	32	159K	128M
inception (3b)		28×28×480	2	128	128	192	32	96	64	380K	304M
max pool	3×3/2	14×14×480	0								
inception (4a)		14×14×512	2	192	96	208	16	48	64	364K	73M
inception (4b)		14×14×512	2	160	112	224	24	64	64	437K	88M
inception (4c)		14×14×512	2	128	128	256	24	64	64	463K	100M
inception (4d)		14×14×528	2	112	144	288	32	64	64	580K	119M
inception (4e)		14×14×832	2	256	160	320	32	128	128	840K	170M
max pool	3×3/2	7×7×832	0								
inception (5a)		7×7×832	2	256	160	320	32	128	128	1072K	54M
inception (5b)		7×7×1024	2	384	192	384	48	128	128	1388K	71M
avg pool	7×7/1	1×1×1024	0								
dropout (40%)		1×1×1024	0								
linear		1×1×1000	1							1000K	1M
softmax		1×1×1000	0								

Table 1: GoogLeNet incarnation of the Inception architecture

# Residual Networks

Introduced by [He et. al.](#) in 2016, the winner of ILSVRC'16.

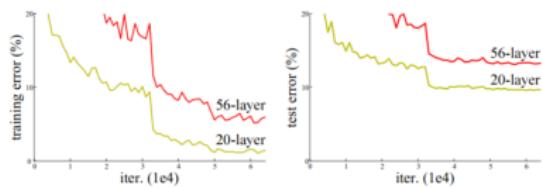
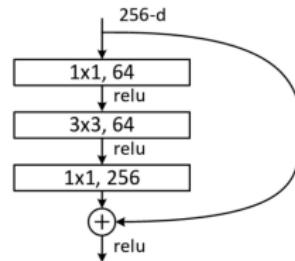


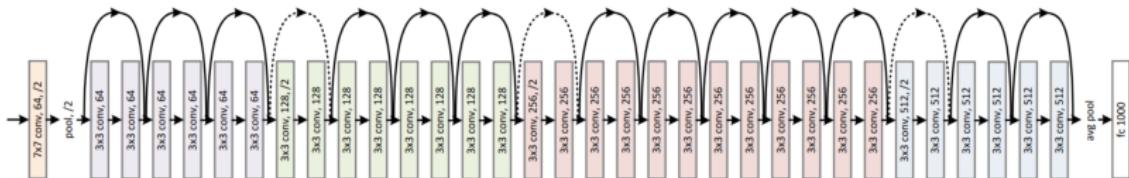
Figure 1. Training error (left) and test error (right) on CIFAR-10 with 20-layer and 56-layer “plain” networks. The deeper network has higher training error, and thus test error. Similar phenomena on ImageNet is presented in Fig. 4.

Deeper is worse ?!



Residual block

# Residual Networks

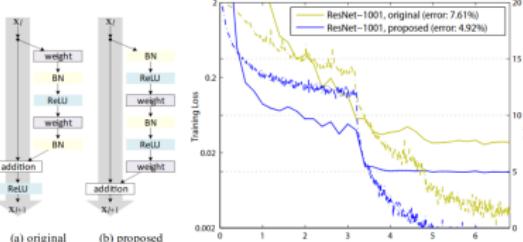


ResNet34. Dotted shortcuts and conv“/2” are stride 2 to match the spatial dimensions. Dotted shortcuts use  $1 \times 1$  conv to match the depth. 0.46M parameters.

layer name	output size	18-layer	34-layer	50-layer	7×7, 64, stride 2	101-layer	152-layer
					3×3 max pool, stride 2		
conv1	112×112						
conv2.x	56×56	$\left[\begin{matrix} 3\times3, 64 \\ 3\times3, 64 \end{matrix}\right] \times 2$	$\left[\begin{matrix} 3\times3, 64 \\ 3\times3, 64 \end{matrix}\right] \times 3$	$\left[\begin{matrix} 1\times1, 64 \\ 3\times3, 64 \\ 1\times1, 256 \end{matrix}\right] \times 3$	$\left[\begin{matrix} 1\times1, 64 \\ 3\times3, 64 \\ 1\times1, 256 \end{matrix}\right] \times 3$	$\left[\begin{matrix} 1\times1, 64 \\ 3\times3, 64 \\ 1\times1, 256 \end{matrix}\right] \times 3$	$\left[\begin{matrix} 1\times1, 64 \\ 3\times3, 64 \\ 1\times1, 256 \end{matrix}\right] \times 3$
conv3.x	28×28	$\left[\begin{matrix} 3\times3, 128 \\ 3\times3, 128 \end{matrix}\right] \times 2$	$\left[\begin{matrix} 3\times3, 128 \\ 3\times3, 128 \end{matrix}\right] \times 4$	$\left[\begin{matrix} 1\times1, 128 \\ 3\times3, 128 \\ 1\times1, 512 \end{matrix}\right] \times 4$	$\left[\begin{matrix} 1\times1, 128 \\ 3\times3, 128 \\ 1\times1, 512 \end{matrix}\right] \times 4$	$\left[\begin{matrix} 1\times1, 128 \\ 3\times3, 128 \\ 1\times1, 512 \end{matrix}\right] \times 8$	$\left[\begin{matrix} 1\times1, 128 \\ 3\times3, 128 \\ 1\times1, 512 \end{matrix}\right] \times 8$
conv4.x	14×14	$\left[\begin{matrix} 3\times3, 256 \\ 3\times3, 256 \end{matrix}\right] \times 2$	$\left[\begin{matrix} 3\times3, 256 \\ 3\times3, 256 \end{matrix}\right] \times 6$	$\left[\begin{matrix} 1\times1, 256 \\ 3\times3, 256 \\ 1\times1, 1024 \end{matrix}\right] \times 6$	$\left[\begin{matrix} 1\times1, 256 \\ 3\times3, 256 \\ 1\times1, 1024 \end{matrix}\right] \times 23$	$\left[\begin{matrix} 1\times1, 256 \\ 3\times3, 256 \\ 1\times1, 1024 \end{matrix}\right] \times 36$	$\left[\begin{matrix} 1\times1, 256 \\ 3\times3, 256 \\ 1\times1, 1024 \end{matrix}\right] \times 36$
conv5.x	7×7	$\left[\begin{matrix} 3\times3, 512 \\ 3\times3, 512 \end{matrix}\right] \times 2$	$\left[\begin{matrix} 3\times3, 512 \\ 3\times3, 512 \end{matrix}\right] \times 3$	$\left[\begin{matrix} 1\times1, 512 \\ 3\times3, 512 \\ 1\times1, 2048 \end{matrix}\right] \times 3$	$\left[\begin{matrix} 1\times1, 512 \\ 3\times3, 512 \\ 1\times1, 2048 \end{matrix}\right] \times 3$	$\left[\begin{matrix} 1\times1, 512 \\ 3\times3, 512 \\ 1\times1, 2048 \end{matrix}\right] \times 3$	$\left[\begin{matrix} 1\times1, 512 \\ 3\times3, 512 \\ 1\times1, 2048 \end{matrix}\right] \times 3$
	1×1	average pool, 1000-d fc, softmax					
FLOPs		$1.8 \times 10^9$	$3.6 \times 10^9$	$3.8 \times 10^9$	$7.6 \times 10^9$	$11.3 \times 10^9$	

Resnet architectures. Conv are “Conv-BN-Relu”.

ResNet-50 has 23M parameters.



Shortcut variations (He et al., 2016b)

# Other variants of skip connections

DenseNets, introduced by [Huang et. al.](#) in 2019.

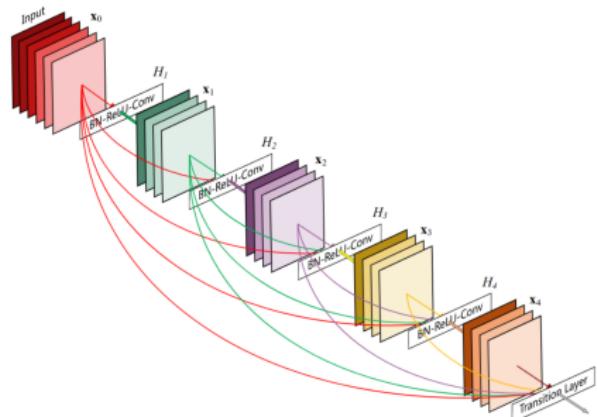
Introduced by [Srivastava et. al.](#) in 2015, the so called **Highway Networks**.  
Uses "Gates" (as in LSTM):

- Transform gate

$$T(x) = \sigma(W_T x + b_T)$$

- Carry gate  $C(x) = \sigma(\rho_c(x))$

$$y = T(x) \cdot H(x) + C(x) \cdot x$$



Introduction  
oooooooooooo

10 years of CNN revolution  
oooooooooooo

Design principles of CNNs  
●oooooooooooo

We need data, but we don't have!  
oooooooooooo

## Design principles of CNNs

# Number of filters

You should increase the number of filters throughout the network :

- the first layer extracts low level features
- the higher layers compose on the lower layer dictionary of features

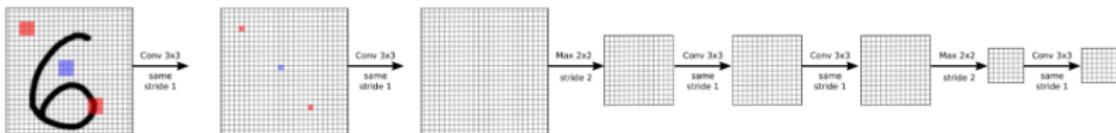
## Some examples:

- LeNet-5 (1998) :  $65 \times 5 \quad 165 \times 5$
- AlexNet (2012) :  $96 \times 11 \quad 256 \times 5 \quad 2 \times (384 \times 3) \quad 256 \times 3$
- VGG (2014) :  $64 - 128 - 256 - 512 \quad \text{all} 3 \times 3$
- ResNet (2015) :  $64 - 128 - 256 - 512 \quad \text{all} 3 \times 3$
- Inception (2015) :  
 $32 - 64 - 80 - 192 - 288 - 768 - 1280 - 2048 \quad 1 \times 1, 3 \times 3, 5 \times 5$

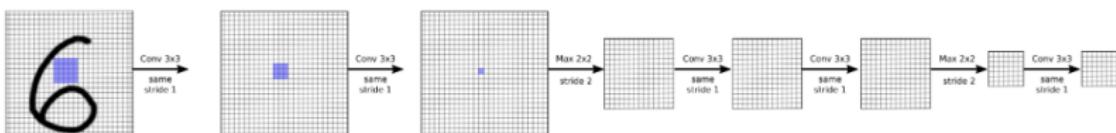
# Receptive field

- The receptive field refers to the area of the input image that a particular neuron in the network "sees".
- It represents the region in the input space that contributes to the activation of a given neuron.
- Receptive fields can be local or global, depending on the architecture and layers of the neural network.
- Understanding the receptive field is crucial for interpreting the behavior and features learned by different layers in a convolutional neural network.
- It plays a key role in tasks like object detection, where the network must recognize objects at different scales and positions within an image.
- For calculating the effective receptive field size, see for example [this calculator](#).

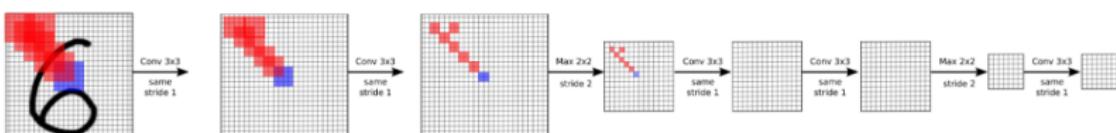
# Effective receptive field



Representation Size	28x28	28x28
Input RF Size	1x1	3x3

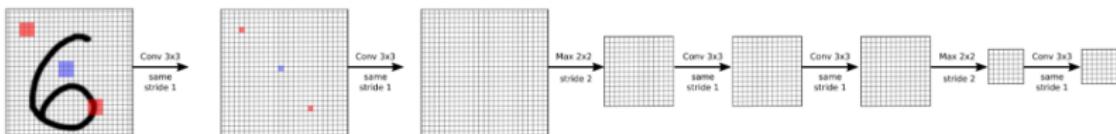


Representation Size	28x28	28x28	28x28
Input RF Size	1x1	3x3	5x5

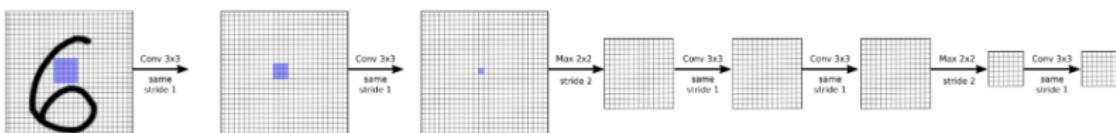


Representation Size	28x28	28x28	28x28	14x14	6x6
Input RF Size	1x1	3x3	5x5	8x8	16x16

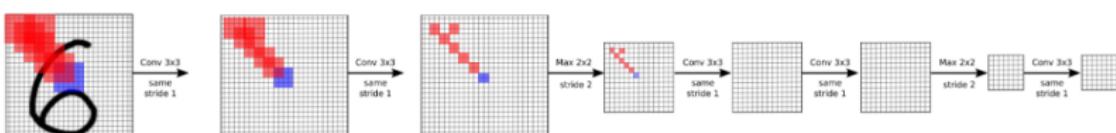
# Effective receptive field



Representation Size	28x28	28x28
Input RF Size	1x1	3x3

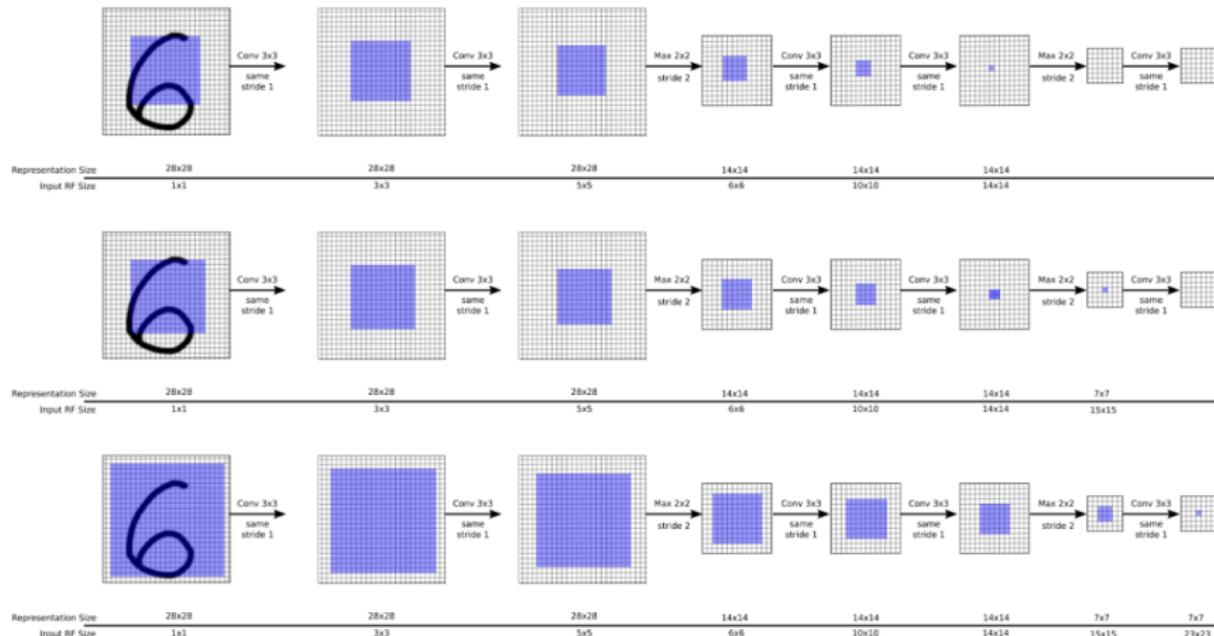


Representation Size	28x28	28x28	28x28
Input RF Size	1x1	3x3	5x5



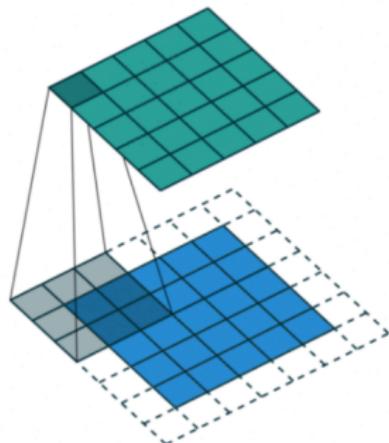
Representation Size	28x28	28x28	28x28	14x14	28x28
Input RF Size	1x1	3x3	5x5	6x6	14x14

# Effective receptive field

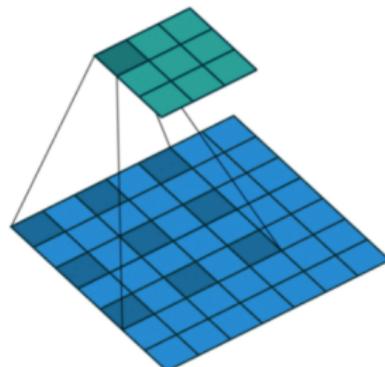


# A-TROU Convolutions

- The effective receptive field can grow faster with a-trou convolutions (or dilated convolutions).
- Introduced by [Yu and Koltun](#) in 2016.



Conv 3, Pad 1, Stride 1



Conv 3, Pad 0, Stride 1, Dilated 1

# Stacking small kernels

Introduced by [Szegedy et. al.](#) in 2015, in Inception v3. A more recent variant was proposed by [Tand and Lee](#) in 2021.

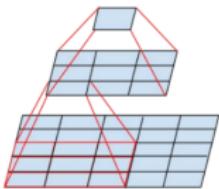


Figure 1. Mini-network replacing the  $5 \times 5$  convolutions.

Stacking  $2(3 \times 3)$  conv

$n$  input filters and  $\alpha n$  output filters :

- $(\alpha n, 5 \times 5)$  conv:  $25\alpha n^2$  param.
- $(\sqrt{\alpha}n, 3 \times 3) - (\alpha n, 3 \times 3)$  :  
 $9\sqrt{\alpha}n^2 + 9\sqrt{\alpha}\alpha n^2$  params;

if  $\alpha = 2 \rightarrow -24\%$



Figure 3. Mini-network replacing the  $3 \times 3$  convolutions. The lower layer of this network consists of a  $3 \times 1$  convolution with 3 output units.

$1 \times 3$  and  $3 \times 1$  conv

$n$  input filters and  $\alpha n$  output filters :

- $(\alpha n, 3 \times 3)$  conv:  $9\alpha n^2$  param.
- $(\sqrt{\alpha}n, 1 \times 3) - (\alpha n, 3 \times 1)$  :  
 $3\sqrt{\alpha}n^2 + 3\sqrt{\alpha}\alpha n^2$  params;

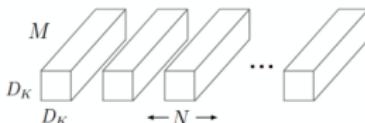
if  $\alpha = 2 \rightarrow -30\%$

# Depthwise separable convolutions

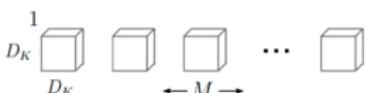
**Idea:** reduce computational complexity while maintaining performance.

Seaparte:

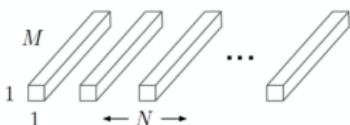
- feature extraction in each channel, in space : depthwise convolution
- feature combination between channels : pointwise convolution  $1 \times 1$



(a) Standard Convolution Filters



(b) Depthwise Convolutional Filters



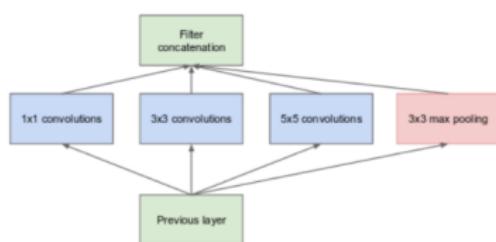
(c)  $1 \times 1$  Convolutional Filters called Pointwise Convolution in the context of Depthwise Separable Convolution

Depthwise and pointwise convolutions (Howard et al., 2017)

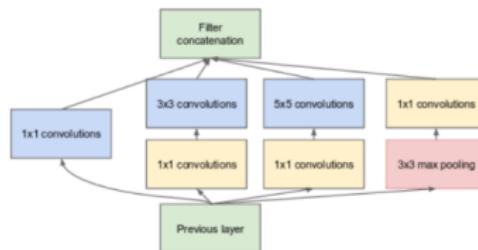
# Multi-Scale Feature extraction

**Idea:** capture information at different scales within an image. It involves analyzing the image at multiple resolutions or levels of detail to extract relevant features.

See Feature Pyramid Network introduced by [Lin et. al.](#) for details.



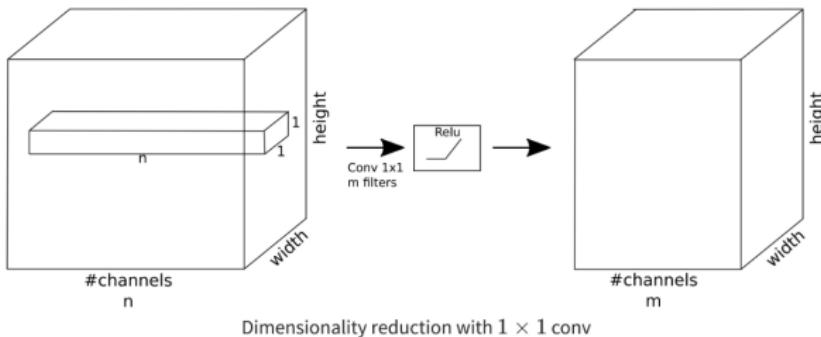
(a) Inception module, naïve version



(b) Inception module with dimension reductions

# Dimensionality reduction

**Idea:** Trainable non-linear transformation of the channels. Network in network.

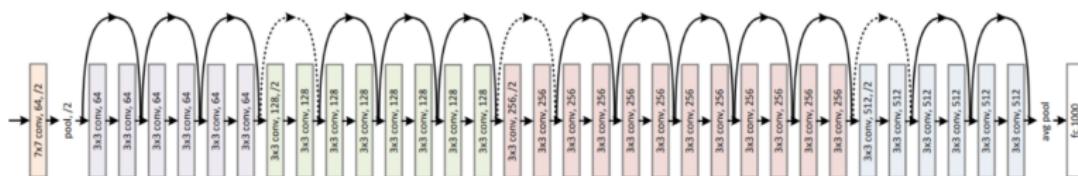


# The Gradient Flow

You can check the norm of the gradient w.r.t. the first layers' parameters to diagnose vanishing gradients.

## Easing the gradient flow :

- shortcut connections (e.g. ResNet, DenseNet, HighwayNet, etc.)
- auxiliary heads (e.g. GoogleNet)



# Are Pooling layers necessary?

**Idea:** Recent architectures remove the max pooling layers and replace them by  $\text{conv}(\text{stride}=2)$  for downsampling.

Input	Operator	$t$	$c$	$n$	$s$
$224^2 \times 3$	conv2d	-	32	1	2
$112^2 \times 32$	bottleneck	1	16	1	1
$112^2 \times 16$	bottleneck	6	24	2	2
$56^2 \times 24$	bottleneck	6	32	3	2
$28^2 \times 32$	bottleneck	6	64	4	2
$14^2 \times 64$	bottleneck	6	96	3	1
$14^2 \times 96$	bottleneck	6	160	3	2
$7^2 \times 160$	bottleneck	6	320	1	1
$7^2 \times 320$	conv2d 1x1	-	1280	1	1
$7^2 \times 1280$	avgpool 7x7	-	-	1	-
$1 \times 1 \times 1280$	conv2d 1x1	-	k	-	-

Model		
Strided-CNN-C	ConvPool-CNN-C	All-CNN-C
Input $32 \times 32$ RGB image		
$3 \times 3$ conv. 96 ReLU	$3 \times 3$ conv. 96 ReLU	$3 \times 3$ conv. 96 ReLU
$3 \times 3$ conv. 96 ReLU with stride $r = 2$	$3 \times 3$ conv. 96 ReLU	$3 \times 3$ conv. 96 ReLU
	$3 \times 3$ max-pooling stride 2	$3 \times 3$ conv. 96 ReLU with stride $r = 2$
$3 \times 3$ conv. 192 ReLU	$3 \times 3$ conv. 192 ReLU	$3 \times 3$ conv. 192 ReLU
$3 \times 3$ conv. 192 ReLU with stride $r = 2$	$3 \times 3$ conv. 192 ReLU	$3 \times 3$ conv. 192 ReLU
	$3 \times 3$ max-pooling stride 2	$3 \times 3$ conv. 192 ReLU with stride $r = 2$

- MobileNet v2 introduced in 2018
- Same Bottleneck architecture is used in other architecture, see EfficientNet

- Striving for simplicity

Introduction  
oooooooooooo

10 years of CNN revolution  
oooooooooooo

Design principles of CNNs  
oooooooooooo

We need data, but we don't have!  
●oooooooo

We need data, but we don't have!

# Using pre-trained models

- All modern frameworks provide a **model zoo** of pre-trained networks. See for example [torchvision documentation](#) where you can find pretrained models for classification, detection, segmentation ...
- What is common practice is to cut the head and finetune the softmax only.
- Do not forget about input normalization!

```
import torchvision.models as models

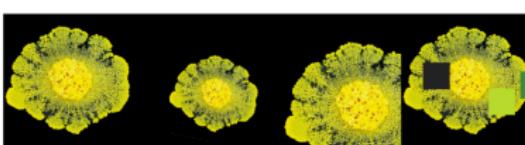
resnet18 = models.resnet18(pretrained=True)
alexnet = models.alexnet(pretrained=True)
vgg16 = models.vgg16(pretrained=True)
inception = models.inception_v3(pretrained=True)
googlenet = models.googlenet(pretrained=True)
mobilenet = models.mobilenet_v2(pretrained=True)
```

# Dataset Augmentation

**Idea:** oversample around the training samples by applying transforms (e.g. color jittering, translations, reflections, rotations, PCA, ...) on the inputs that make predictable changes on the targets.

## Examples

- Use [imgaug library](#)



- The images represent all *physarum polycephalum*, and was taken from [CNRS](#).
- Another interesting work around this subject is [mixup](#).
- Make sure that the augmentation transforms make sense.



## Example on CIFAR-100

- Has 100 object classes, each having 600 images per class.
- Size:  $32 \times 32 \times 3$
- The training set has  $500 \times 100$  images, and the test set has  $100 \times 100$  images



# Model Architecture

Operator	Res.	RF Size	#Ch
ConvBlock	$32 \times 32$	$5 \times 5$	32
ConvBlock	$32 \times 32$	$9 \times 9$	32
Sub	$16 \times 16$	$15 \times 15$	32
ConvBlock	$16 \times 16$	$15 \times 15$	128
ConvBlock	$16 \times 16$	$23 \times 23$	128
Sub	$8 \times 8$	$31 \times 31$	128
AvgPool	$1 \times 1$		128
Linear	100		

ConvBlock: 2x [Conv(1x3)-(BN)-Relu-  
Conv(3x1)-(BN)-Relu]

Sub : Conv(3x3, stride=2)-(BN)-Relu

No. of. param:  $\approx 2M$

## Settings:

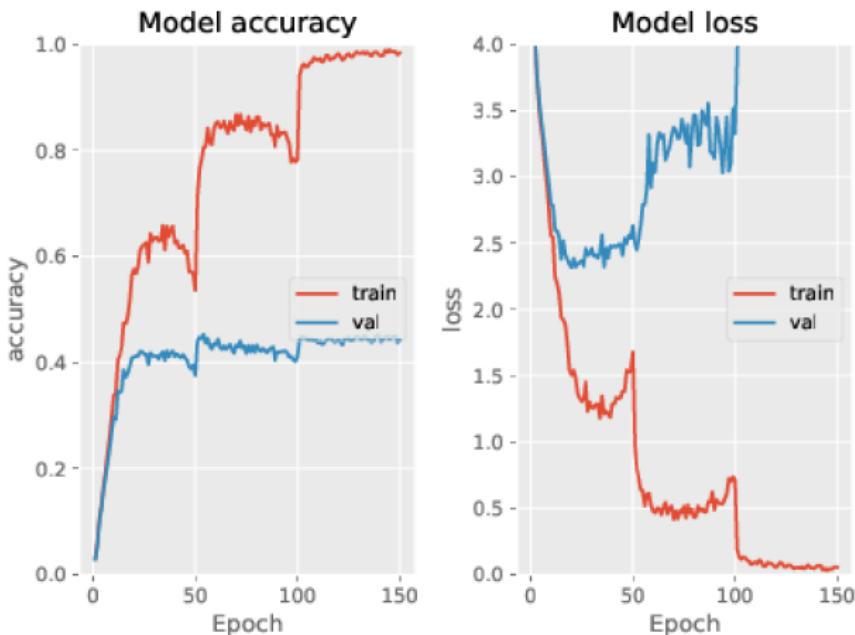
- BatchSize(32)
- SGD(lrate=0.01) with momentum(0.9)
- learning rate halved every 50 epochs
- validation on 20%, early stopping on the val loss

## Other configurations:

- base
- Conv-BN-Relu or Conv-Relu
- dataset augmentation (HFlip, Trans(5pix), Scale(0.8,1.2)), CenterCrop(32)
- Dropout, L2, label smoothing

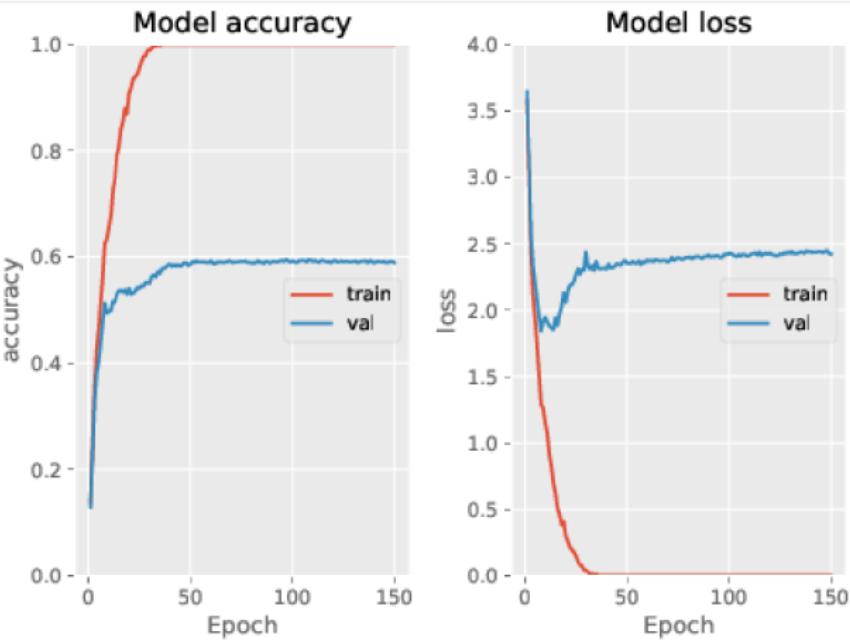
# Baseline

- No regularization, no BatchNorm



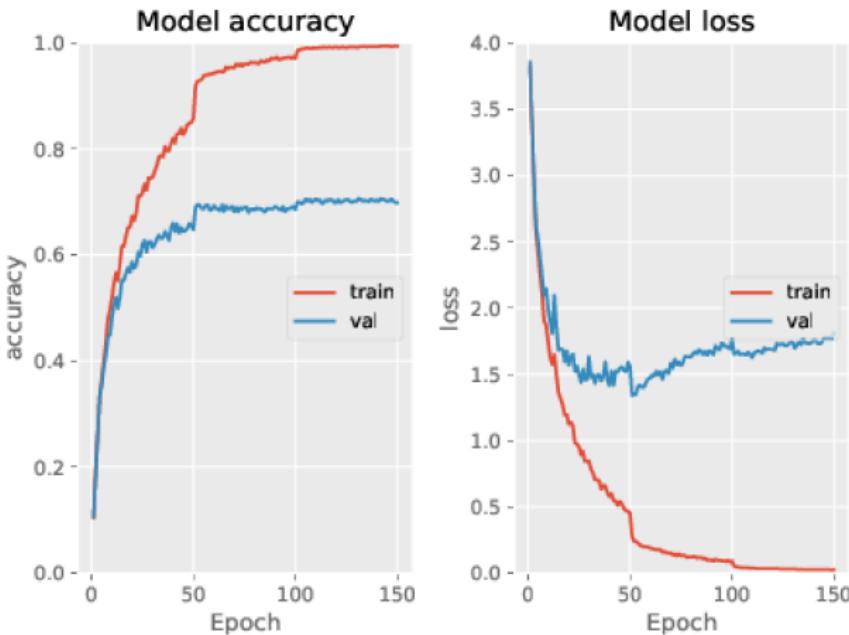
# with BatchNorm

- with BatchNorm after each convolution



# with Augmentation

- with input augmentation: HFlip, Scale, Trans



# with Regularization

- L2 (0.0025), Dropout(0.5), Label smoothing(0.1)

